

QEverCloud

Generated by Doxygen 1.9.4



<b>1 QEverCloud</b>	<b>1</b>
1.1 What's this	1
1.2 How to contribute	1
1.3 Downloads	1
1.4 How to build	2
1.5 Include files for applications using the library	3
1.6 Seeding random numbers generator for Qt < 5.10	3
1.7 Related projects	3
<b>2 Namespace Index</b>	<b>5</b>
2.1 Namespace List	5
<b>3 Hierarchical Index</b>	<b>7</b>
3.1 Class Hierarchy	7
<b>4 Class Index</b>	<b>13</b>
4.1 Class List	13
<b>5 File Index</b>	<b>17</b>
5.1 File List	17
<b>6 Namespace Documentation</b>	<b>19</b>
6.1 qevercloud Namespace Reference	19
6.1.1 Detailed Description	29
6.1.2 Typedef Documentation	29
6.1.2.1 EverCloudExceptionDataPtr	29
6.1.2.2 Guid	30
6.1.2.3 IdentityID	30
6.1.2.4 IDurableServicePtr	30
6.1.2.5 ILoggerPtr	30
6.1.2.6 INoteStorePtr	30
6.1.2.7 InvalidationSequenceNumber	30
6.1.2.8 IRequestContextPtr	30
6.1.2.9 IRetryPolicyPtr	31
6.1.2.10 IUserStorePtr	31
6.1.2.11 MessageEventID	31
6.1.2.12 MessageThreadID	31
6.1.2.13 Timestamp	31
6.1.2.14 UserID	31
6.1.3 Enumeration Type Documentation	32
6.1.3.1 BusinessInvitationStatus	32
6.1.3.2 BusinessUserRole	32
6.1.3.3 BusinessUserStatus	32
6.1.3.4 CanMoveToContainerStatus	33

6.1.3.5	ContactType	33
6.1.3.6	EDAMErrorCode	34
6.1.3.7	EDAMInvalidContactReason	35
6.1.3.8	EntityType	36
6.1.3.9	LogLevel	36
6.1.3.10	NoteSortOrder	37
6.1.3.11	PremiumOrderStatus	37
6.1.3.12	PrivilegeLevel	38
6.1.3.13	QueryFormat	38
6.1.3.14	RecipientStatus	38
6.1.3.15	RelatedContentAccess	39
6.1.3.16	RelatedContentType	39
6.1.3.17	ReminderEmailConfig	39
6.1.3.18	ServiceLevel	40
6.1.3.19	SharedNotebookInstanceRestrictions	40
6.1.3.20	SharedNotebookPrivilegeLevel	41
6.1.3.21	SharedNotePrivilegeLevel	41
6.1.3.22	ShareRelationshipPrivilegeLevel	43
6.1.3.23	SponsoredGroupRole	43
6.1.3.24	UserIdentityType	44
6.1.4	Function Documentation	44
6.1.4.1	evernoteNetworkProxy()	44
6.1.4.2	initializeQEverCloud()	44
6.1.4.3	libraryVersion()	45
6.1.4.4	logger()	45
6.1.4.5	newDurableService()	45
6.1.4.6	newNoteStore()	45
6.1.4.7	newRequestContext()	45
6.1.4.8	newRetryPolicy()	45
6.1.4.9	newStdErrLogger()	46
6.1.4.10	newUserStore()	46
6.1.4.11	nullLogger()	46
6.1.4.12	nullRetryPolicy()	46
6.1.4.13	operator<<() [1/48]	46
6.1.4.14	operator<<() [2/48]	46
6.1.4.15	operator<<() [3/48]	47
6.1.4.16	operator<<() [4/48]	47
6.1.4.17	operator<<() [5/48]	47
6.1.4.18	operator<<() [6/48]	47
6.1.4.19	operator<<() [7/48]	47
6.1.4.20	operator<<() [8/48]	47
6.1.4.21	operator<<() [9/48]	48

6.1.4.22 operator<<() [10/48]	48
6.1.4.23 operator<<() [11/48]	48
6.1.4.24 operator<<() [12/48]	48
6.1.4.25 operator<<() [13/48]	48
6.1.4.26 operator<<() [14/48]	48
6.1.4.27 operator<<() [15/48]	49
6.1.4.28 operator<<() [16/48]	49
6.1.4.29 operator<<() [17/48]	49
6.1.4.30 operator<<() [18/48]	49
6.1.4.31 operator<<() [19/48]	49
6.1.4.32 operator<<() [20/48]	49
6.1.4.33 operator<<() [21/48]	50
6.1.4.34 operator<<() [22/48]	50
6.1.4.35 operator<<() [23/48]	50
6.1.4.36 operator<<() [24/48]	50
6.1.4.37 operator<<() [25/48]	50
6.1.4.38 operator<<() [26/48]	50
6.1.4.39 operator<<() [27/48]	51
6.1.4.40 operator<<() [28/48]	51
6.1.4.41 operator<<() [29/48]	51
6.1.4.42 operator<<() [30/48]	51
6.1.4.43 operator<<() [31/48]	51
6.1.4.44 operator<<() [32/48]	51
6.1.4.45 operator<<() [33/48]	52
6.1.4.46 operator<<() [34/48]	52
6.1.4.47 operator<<() [35/48]	52
6.1.4.48 operator<<() [36/48]	52
6.1.4.49 operator<<() [37/48]	52
6.1.4.50 operator<<() [38/48]	52
6.1.4.51 operator<<() [39/48]	53
6.1.4.52 operator<<() [40/48]	53
6.1.4.53 operator<<() [41/48]	53
6.1.4.54 operator<<() [42/48]	53
6.1.4.55 operator<<() [43/48]	53
6.1.4.56 operator<<() [44/48]	53
6.1.4.57 operator<<() [45/48]	54
6.1.4.58 operator<<() [46/48]	54
6.1.4.59 operator<<() [47/48]	54
6.1.4.60 operator<<() [48/48]	54
6.1.4.61 qHash() [1/23]	54
6.1.4.62 qHash() [2/23]	54
6.1.4.63 qHash() [3/23]	55

6.1.4.64 qHash() [4/23]	55
6.1.4.65 qHash() [5/23]	55
6.1.4.66 qHash() [6/23]	55
6.1.4.67 qHash() [7/23]	55
6.1.4.68 qHash() [8/23]	55
6.1.4.69 qHash() [9/23]	55
6.1.4.70 qHash() [10/23]	56
6.1.4.71 qHash() [11/23]	56
6.1.4.72 qHash() [12/23]	56
6.1.4.73 qHash() [13/23]	56
6.1.4.74 qHash() [14/23]	56
6.1.4.75 qHash() [15/23]	56
6.1.4.76 qHash() [16/23]	56
6.1.4.77 qHash() [17/23]	57
6.1.4.78 qHash() [18/23]	57
6.1.4.79 qHash() [19/23]	57
6.1.4.80 qHash() [20/23]	57
6.1.4.81 qHash() [21/23]	57
6.1.4.82 qHash() [22/23]	57
6.1.4.83 qHash() [23/23]	57
6.1.4.84 resetEvernoteNetworkProxy()	58
6.1.4.85 setEvernoteNetworkProxy()	58
6.1.4.86 setLogger()	58
6.1.4.87 setNonceGenerator()	58
6.1.4.88 toRange() [1/2]	59
6.1.4.89 toRange() [2/2]	59
6.1.5 Variable Documentation	59
6.1.5.1 CLASSIFICATION_RECIPE_SERVICE_RECIPE	59
6.1.5.2 CLASSIFICATION_RECIPE_USER_NON_RECIPE	59
6.1.5.3 CLASSIFICATION_RECIPE_USER_RECIPE	59
6.1.5.4 EDAM_APP_RATING_MAX	59
6.1.5.5 EDAM_APP_RATING_MIN	60
6.1.5.6 EDAM_APPLICATIONDATA_ENTRY_LEN_MAX	60
6.1.5.7 EDAM_APPLICATIONDATA_NAME_LEN_MAX	60
6.1.5.8 EDAM_APPLICATIONDATA_NAME_LEN_MIN	60
6.1.5.9 EDAM_APPLICATIONDATA_NAME_REGEX	60
6.1.5.10 EDAM_APPLICATIONDATA_VALUE_LEN_MAX	60
6.1.5.11 EDAM_APPLICATIONDATA_VALUE_LEN_MIN	60
6.1.5.12 EDAM_APPLICATIONDATA_VALUE_REGEX	61
6.1.5.13 EDAM_ATTRIBUTE_LEN_MAX	61
6.1.5.14 EDAM_ATTRIBUTE_LEN_MIN	61
6.1.5.15 EDAM_ATTRIBUTE_LIST_MAX	61

6.1.5.16 EDAM_ATTRIBUTE_MAP_MAX	61
6.1.5.17 EDAM_ATTRIBUTE_REGEX	61
6.1.5.18 EDAM_BUSINESS_MARKETING_CODE_REGEX_PATTERN	61
6.1.5.19 EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_LEN_MAX	62
6.1.5.20 EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_LEN_MIN	62
6.1.5.21 EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_REGEX	62
6.1.5.22 EDAM_BUSINESS_NOTEBOOKS_MAX	62
6.1.5.23 EDAM_BUSINESS_NOTES_MAX	62
6.1.5.24 EDAM_BUSINESS_PHONE_NUMBER_LEN_MAX	62
6.1.5.25 EDAM_BUSINESS_TAGS_MAX	62
6.1.5.26 EDAM_BUSINESS_URI_LEN_MAX	63
6.1.5.27 EDAM_BUSINESS_WORKSPACES_MAX	63
6.1.5.28 EDAM_CONNECTED_IDENTITY_REQUEST_MAX	63
6.1.5.29 EDAM_CONTENT_CLASS_FOOD_MEAL	63
6.1.5.30 EDAM_CONTENT_CLASS_HELLO_ENCOUNTER	63
6.1.5.31 EDAM_CONTENT_CLASS_HELLO_PROFILE	63
6.1.5.32 EDAM_CONTENT_CLASS_PENULTIMATE_NOTEBOOK	63
6.1.5.33 EDAM_CONTENT_CLASS_PENULTIMATE_PREFIX	64
6.1.5.34 EDAM_CONTENT_CLASS_SKITCH	64
6.1.5.35 EDAM_CONTENT_CLASS_SKITCH_PDF	64
6.1.5.36 EDAM_CONTENT_CLASS_SKITCH_PREFIX	64
6.1.5.37 EDAM_DEVICE_DESCRIPTION_LEN_MAX	64
6.1.5.38 EDAM_DEVICE_DESCRIPTION_REGEX	64
6.1.5.39 EDAM_DEVICE_ID_LEN_MAX	64
6.1.5.40 EDAM_DEVICE_ID_REGEX	65
6.1.5.41 EDAM_EMAIL_DOMAIN_REGEX	65
6.1.5.42 EDAM_EMAIL_LEN_MAX	65
6.1.5.43 EDAM_EMAIL_LEN_MIN	65
6.1.5.44 EDAM_EMAIL_LOCAL_REGEX	65
6.1.5.45 EDAM_EMAIL_REGEX	65
6.1.5.46 EDAM_FIND_CONTACT_DEFAULT_MAX_RESULTS	65
6.1.5.47 EDAM_FIND_CONTACT_MAX_RESULTS	66
6.1.5.48 EDAM_FOOD_APP_CONTENT_CLASS_PREFIX	66
6.1.5.49 EDAM_GET_ORDERS_MAX_RESULTS	66
6.1.5.50 EDAM_GUID_LEN_MAX	66
6.1.5.51 EDAM_GUID_LEN_MIN	66
6.1.5.52 EDAM_GUID_REGEX	66
6.1.5.53 EDAM_HASH_LEN	66
6.1.5.54 EDAM_HELLO_APP_CONTENT_CLASS_PREFIX	67
6.1.5.55 EDAM_INDEXABLE_PLAINTEXT_MIME_TYPES	67
6.1.5.56 EDAM_INDEXABLE_RESOURCE_MIME_TYPES	67
6.1.5.57 EDAM_MAX_PREFERENCES	67

6.1.5.58 EDAM_MAX_VALUES_PER_PREFERENCE . . . . .	67
6.1.5.59 EDAM_MESSAGE_ATTACHMENT_SNIPPET_LEN_MAX . . . . .	67
6.1.5.60 EDAM_MESSAGE_ATTACHMENT_SNIPPET_REGEX . . . . .	67
6.1.5.61 EDAM_MESSAGE_ATTACHMENT_TITLE_LEN_MAX . . . . .	68
6.1.5.62 EDAM_MESSAGE_ATTACHMENT_TITLE_REGEX . . . . .	68
6.1.5.63 EDAM_MESSAGE_ATTACHMENTS_MAX . . . . .	68
6.1.5.64 EDAM_MESSAGE_BODY_LEN_MAX . . . . .	68
6.1.5.65 EDAM_MESSAGE_BODY_REGEX . . . . .	68
6.1.5.66 EDAM_MESSAGE_RECIPIENTS_MAX . . . . .	68
6.1.5.67 EDAM_MIME_LEN_MAX . . . . .	68
6.1.5.68 EDAM_MIME_LEN_MIN . . . . .	68
6.1.5.69 EDAM_MIME_REGEX . . . . .	69
6.1.5.70 EDAM_MIME_TYPE_AAC . . . . .	69
6.1.5.71 EDAM_MIME_TYPE_AMR . . . . .	69
6.1.5.72 EDAM_MIME_TYPE_BMP . . . . .	69
6.1.5.73 EDAM_MIME_TYPE_DEFAULT . . . . .	69
6.1.5.74 EDAM_MIME_TYPE_GIF . . . . .	69
6.1.5.75 EDAM_MIME_TYPE_INK . . . . .	69
6.1.5.76 EDAM_MIME_TYPE_JPEG . . . . .	69
6.1.5.77 EDAM_MIME_TYPE_M4A . . . . .	70
6.1.5.78 EDAM_MIME_TYPE_MP3 . . . . .	70
6.1.5.79 EDAM_MIME_TYPE_MP4_VIDEO . . . . .	70
6.1.5.80 EDAM_MIME_TYPE_PDF . . . . .	70
6.1.5.81 EDAM_MIME_TYPE_PNG . . . . .	70
6.1.5.82 EDAM_MIME_TYPE_TIFF . . . . .	70
6.1.5.83 EDAM_MIME_TYPE_WAV . . . . .	70
6.1.5.84 EDAM_MIME_TYPES . . . . .	70
6.1.5.85 EDAM_NOTE_BUSINESS_SHARED_NOTE_MAX . . . . .	71
6.1.5.86 EDAM_NOTE_CONTENT_CLASS_LEN_MAX . . . . .	71
6.1.5.87 EDAM_NOTE_CONTENT_CLASS_LEN_MIN . . . . .	71
6.1.5.88 EDAM_NOTE_CONTENT_CLASS_REGEX . . . . .	71
6.1.5.89 EDAM_NOTE_CONTENT_LEN_MAX . . . . .	71
6.1.5.90 EDAM_NOTE_CONTENT_LEN_MIN . . . . .	71
6.1.5.91 EDAM_NOTE_LOCK_VIEWERS_NOTES_MAX . . . . .	71
6.1.5.92 EDAM_NOTE_PERSONAL_SHARED_NOTE_MAX . . . . .	71
6.1.5.93 EDAM_NOTE_RESOURCES_MAX . . . . .	72
6.1.5.94 EDAM_NOTE_SIZE_MAX_FREE . . . . .	72
6.1.5.95 EDAM_NOTE_SIZE_MAX_PREMIUM . . . . .	72
6.1.5.96 EDAM_NOTE_SOURCE_MAIL_CLIP . . . . .	72
6.1.5.97 EDAM_NOTE_SOURCE_MAIL_SMTP_GATEWAY . . . . .	72
6.1.5.98 EDAM_NOTE_SOURCE_WEB_CLIP . . . . .	72
6.1.5.99 EDAM_NOTE_SOURCE_WEB_CLIP_SIMPLIFIED . . . . .	72



6.1.5.100 EDAM_NOTE_TAGS_MAX . . . . .	73
6.1.5.101 EDAM_NOTE_TITLE_LEN_MAX . . . . .	73
6.1.5.102 EDAM_NOTE_TITLE_LEN_MIN . . . . .	73
6.1.5.103 EDAM_NOTE_TITLE_QUALITY_HIGH . . . . .	73
6.1.5.104 EDAM_NOTE_TITLE_QUALITY_LOW . . . . .	73
6.1.5.105 EDAM_NOTE_TITLE_QUALITY_MEDIUM . . . . .	73
6.1.5.106 EDAM_NOTE_TITLE_QUALITY_UNTITLED . . . . .	73
6.1.5.107 EDAM_NOTE_TITLE_REGEX . . . . .	74
6.1.5.108 EDAM_NOTEBOOK_BUSINESS_SHARED_NOTEBOOK_MAX . . . . .	74
6.1.5.109 EDAM_NOTEBOOK_NAME_LEN_MAX . . . . .	74
6.1.5.110 EDAM_NOTEBOOK_NAME_LEN_MIN . . . . .	74
6.1.5.111 EDAM_NOTEBOOK_NAME_REGEX . . . . .	74
6.1.5.112 EDAM_NOTEBOOK_PERSONAL_SHARED_NOTEBOOK_MAX . . . . .	74
6.1.5.113 EDAM_NOTEBOOK_STACK_LEN_MAX . . . . .	74
6.1.5.114 EDAM_NOTEBOOK_STACK_LEN_MIN . . . . .	75
6.1.5.115 EDAM_NOTEBOOK_STACK_REGEX . . . . .	75
6.1.5.116 EDAM_OPEN_ID_ACCESS_TOKEN_MAX . . . . .	75
6.1.5.117 EDAM_PREFERENCE_BUSINESS_DEFAULT_NOTEBOOK . . . . .	75
6.1.5.118 EDAM_PREFERENCE_BUSINESS_QUICKNOTE . . . . .	75
6.1.5.119 EDAM_PREFERENCE_NAME_LEN_MAX . . . . .	75
6.1.5.120 EDAM_PREFERENCE_NAME_LEN_MIN . . . . .	76
6.1.5.121 EDAM_PREFERENCE_NAME_REGEX . . . . .	76
6.1.5.122 EDAM_PREFERENCE_ONLY_ONE_VALUE_LEN_MAX . . . . .	76
6.1.5.123 EDAM_PREFERENCE_ONLY_ONE_VALUE_REGEX . . . . .	76
6.1.5.124 EDAM_PREFERENCE_SHORTCUTS . . . . .	76
6.1.5.125 EDAM_PREFERENCE_SHORTCUTS_MAX_VALUES . . . . .	76
6.1.5.126 EDAM_PREFERENCE_VALUE_LEN_MAX . . . . .	76
6.1.5.127 EDAM_PREFERENCE_VALUE_LEN_MIN . . . . .	77
6.1.5.128 EDAM_PREFERENCE_VALUE_REGEX . . . . .	77
6.1.5.129 EDAM_PROMOTION_ID_LEN_MAX . . . . .	77
6.1.5.130 EDAM_PROMOTION_ID_REGEX . . . . .	77
6.1.5.131 EDAM_PUBLISHING_DESCRIPTION_LEN_MAX . . . . .	77
6.1.5.132 EDAM_PUBLISHING_DESCRIPTION_LEN_MIN . . . . .	77
6.1.5.133 EDAM_PUBLISHING_DESCRIPTION_REGEX . . . . .	77
6.1.5.134 EDAM_PUBLISHING_URI_LEN_MAX . . . . .	78
6.1.5.135 EDAM_PUBLISHING_URI_LEN_MIN . . . . .	78
6.1.5.136 EDAM_PUBLISHING_URI_PROHIBITED . . . . .	78
6.1.5.137 EDAM_PUBLISHING_URI_REGEX . . . . .	78
6.1.5.138 EDAM_RELATED_MAX_EXPERTS . . . . .	78
6.1.5.139 EDAM_RELATED_MAX_NOTEBOOKS . . . . .	78
6.1.5.140 EDAM_RELATED_MAX_NOTES . . . . .	78
6.1.5.141 EDAM_RELATED_MAX_RELATED_CONTENT . . . . .	78

6.1.5.142 EDAM_RELATED_MAX_TAGS . . . . .	79
6.1.5.143 EDAM_RELATED_PLAINTEXT_LEN_MAX . . . . .	79
6.1.5.144 EDAM_RELATED_PLAINTEXT_LEN_MIN . . . . .	79
6.1.5.145 EDAM_RESOURCE_SIZE_MAX_FREE . . . . .	79
6.1.5.146 EDAM_RESOURCE_SIZE_MAX_PREMIUM . . . . .	79
6.1.5.147 EDAM_SAVED_SEARCH_NAME_LEN_MAX . . . . .	79
6.1.5.148 EDAM_SAVED_SEARCH_NAME_LEN_MIN . . . . .	79
6.1.5.149 EDAM_SAVED_SEARCH_NAME_REGEX . . . . .	79
6.1.5.150 EDAM_SEARCH_QUERY_LEN_MAX . . . . .	80
6.1.5.151 EDAM_SEARCH_QUERY_LEN_MIN . . . . .	80
6.1.5.152 EDAM_SEARCH_QUERY_REGEX . . . . .	80
6.1.5.153 EDAM_SEARCH_SUGGESTIONS_MAX . . . . .	80
6.1.5.154 EDAM_SEARCH_SUGGESTIONS_PREFIX_LEN_MAX . . . . .	80
6.1.5.155 EDAM_SEARCH_SUGGESTIONS_PREFIX_LEN_MIN . . . . .	80
6.1.5.156 EDAM_SNIPPETS_NOTES_MAX . . . . .	80
6.1.5.157 EDAM_SOURCE_APPLICATION_ANDROID_SHARE_EXTENSION . . . . .	81
6.1.5.158 EDAM_SOURCE_APPLICATION_EN_SCANSNAP . . . . .	81
6.1.5.159 EDAM_SOURCE_APPLICATION_EWC . . . . .	81
6.1.5.160 EDAM_SOURCE_APPLICATION_IOS_SHARE_EXTENSION . . . . .	81
6.1.5.161 EDAM_SOURCE_APPLICATION_MOLESKINE . . . . .	81
6.1.5.162 EDAM_SOURCE_APPLICATION_POSTIT . . . . .	81
6.1.5.163 EDAM_SOURCE_APPLICATION_WEB_CLIPPER . . . . .	81
6.1.5.164 EDAM_SOURCE_OUTLOOK_CLIPPER . . . . .	82
6.1.5.165 EDAM_TAG_NAME_LEN_MAX . . . . .	82
6.1.5.166 EDAM_TAG_NAME_LEN_MIN . . . . .	82
6.1.5.167 EDAM_TAG_NAME_REGEX . . . . .	82
6.1.5.168 EDAM_TIMEZONE_LEN_MAX . . . . .	82
6.1.5.169 EDAM_TIMEZONE_LEN_MIN . . . . .	82
6.1.5.170 EDAM_TIMEZONE_REGEX . . . . .	82
6.1.5.171 EDAM_USER_LINKED_NOTEBOOK_MAX . . . . .	83
6.1.5.172 EDAM_USER_LINKED_NOTEBOOK_MAX_PREMIUM . . . . .	83
6.1.5.173 EDAM_USER_MAIL_LIMIT_DAILY_FREE . . . . .	83
6.1.5.174 EDAM_USER_MAIL_LIMIT_DAILY_PREMIUM . . . . .	83
6.1.5.175 EDAM_USER_NAME_LEN_MAX . . . . .	83
6.1.5.176 EDAM_USER_NAME_LEN_MIN . . . . .	83
6.1.5.177 EDAM_USER_NAME_REGEX . . . . .	83
6.1.5.178 EDAM_USER_NOTEBOOKS_MAX . . . . .	84
6.1.5.179 EDAM_USER_NOTES_MAX . . . . .	84
6.1.5.180 EDAM_USER_PASSWORD_LEN_MAX . . . . .	84
6.1.5.181 EDAM_USER_PASSWORD_LEN_MIN . . . . .	84
6.1.5.182 EDAM_USER_PASSWORD_REGEX . . . . .	84
6.1.5.183 EDAM_USER_PROFILE_PHOTO_MAX_BYTES . . . . .	84

6.1.5.184 EDAM_USER_RECENT_MAILED_ADDRESSES_MAX	84
6.1.5.185 EDAM_USER_SAVED_SEARCHES_MAX	85
6.1.5.186 EDAM_USER_TAGS_MAX	85
6.1.5.187 EDAM_USER_UPLOAD_LIMIT_BUSINESS	85
6.1.5.188 EDAM_USER_UPLOAD_LIMIT_BUSINESS_FIRST_MONTH	85
6.1.5.189 EDAM_USER_UPLOAD_LIMIT_BUSINESS_NEXT_MONTH	85
6.1.5.190 EDAM_USER_UPLOAD_LIMIT_BUSINESS_PER_USER	85
6.1.5.191 EDAM_USER_UPLOAD_LIMIT_FREE	85
6.1.5.192 EDAM_USER_UPLOAD_LIMIT_PLUS	86
6.1.5.193 EDAM_USER_UPLOAD_LIMIT_PREMIUM	86
6.1.5.194 EDAM_USER_UPLOAD_SURVEY_THRESHOLD	86
6.1.5.195 EDAM_USER_USERNAME_LEN_MAX	86
6.1.5.196 EDAM_USER_USERNAME_LEN_MIN	86
6.1.5.197 EDAM_USER_USERNAME_REGEX	86
6.1.5.198 EDAM_USER_WORKSPACES_MAX	86
6.1.5.199 EDAM_VAT_REGEX	87
6.1.5.200 EDAM_VERSION_MAJOR	87
6.1.5.201 EDAM_VERSION_MINOR	87
6.1.5.202 EDAM_WORKSPACE_DESCRIPTION_LEN_MAX	87
6.1.5.203 EDAM_WORKSPACE_NAME_LEN_MAX	87
6.1.5.204 EDAM_WORKSPACE_NAME_LEN_MIN	87
6.1.5.205 EDAM_WORKSPACE_NAME_REGEX	87
6.1.5.206 EverCloudExceptionData	87
<b>7 Class Documentation</b>	<b>89</b>
7.1 qevercloud::Accounting Struct Reference	89
7.1.1 Detailed Description	90
7.1.2 Member Function Documentation	90
7.1.2.1 operator!=(())	90
7.1.2.2 operator==(())	90
7.1.2.3 print()	90
7.1.3 Member Data Documentation	90
7.1.3.1 availablePoints	91
7.1.3.2 businessId	91
7.1.3.3 businessName	91
7.1.3.4 businessRole	91
7.1.3.5 currency	91
7.1.3.6 lastFailedCharge	91
7.1.3.7 lastFailedChargeReason	91
7.1.3.8 lastRequestedCharge	91
7.1.3.9 lastSuccessfulCharge	92
7.1.3.10 localData	92

7.1.3.11 nextChargeDate . . . . .	92
7.1.3.12 nextPaymentDue . . . . .	92
7.1.3.13 premiumCommerceService . . . . .	92
7.1.3.14 premiumLockUntil . . . . .	92
7.1.3.15 premiumOrderNumber . . . . .	92
7.1.3.16 premiumServiceSKU . . . . .	92
7.1.3.17 premiumServiceStart . . . . .	93
7.1.3.18 premiumServiceStatus . . . . .	93
7.1.3.19 premiumSubscriptionNumber . . . . .	93
7.1.3.20 unitDiscount . . . . .	93
7.1.3.21 unitPrice . . . . .	93
7.1.3.22 updated . . . . .	93
7.1.3.23 uploadLimitEnd . . . . .	93
7.1.3.24 uploadLimitNextMonth . . . . .	94
7.2 qevercloud::AccountLimits Struct Reference . . . . .	94
7.2.1 Detailed Description . . . . .	94
7.2.2 Member Function Documentation . . . . .	94
7.2.2.1 operator"!=( ) . . . . .	95
7.2.2.2 operator==( ) . . . . .	95
7.2.2.3 print( ) . . . . .	95
7.2.3 Member Data Documentation . . . . .	95
7.2.3.1 localData . . . . .	95
7.2.3.2 noteResourceCountMax . . . . .	95
7.2.3.3 noteSizeMax . . . . .	95
7.2.3.4 noteTagCountMax . . . . .	96
7.2.3.5 resourceSizeMax . . . . .	96
7.2.3.6 uploadLimit . . . . .	96
7.2.3.7 userLinkedNotebookMax . . . . .	96
7.2.3.8 userMailLimitDaily . . . . .	96
7.2.3.9 userNotebookCountMax . . . . .	96
7.2.3.10 userNoteCountMax . . . . .	96
7.2.3.11 userSavedSearchesMax . . . . .	97
7.2.3.12 userTagCountMax . . . . .	97
7.3 qevercloud::IDurableService::AsyncRequest Struct Reference . . . . .	97
7.3.1 Constructor & Destructor Documentation . . . . .	97
7.3.1.1 AsyncRequest( ) . . . . .	97
7.3.2 Member Data Documentation . . . . .	97
7.3.2.1 m_call . . . . .	98
7.3.2.2 m_description . . . . .	98
7.3.2.3 m_name . . . . .	98
7.4 qevercloud::AsyncResult Class Reference . . . . .	98
7.4.1 Detailed Description . . . . .	99

7.4.2 Member Typedef Documentation	99
7.4.2.1 ReadFunctionType	99
7.4.3 Constructor & Destructor Documentation	99
7.4.3.1 AsyncResult() [1/3]	99
7.4.3.2 AsyncResult() [2/3]	100
7.4.3.3 AsyncResult() [3/3]	100
7.4.3.4 ~AsyncResult()	100
7.4.4 Member Function Documentation	100
7.4.4.1 asIs()	100
7.4.4.2 finished	100
7.4.4.3 waitForFinished()	101
7.4.5 Friends And Related Function Documentation	101
7.4.5.1 DurableService	101
7.5 qevercloud::AuthenticationResult Struct Reference	101
7.5.1 Detailed Description	102
7.5.2 Member Function Documentation	102
7.5.2.1 operator!=(())	102
7.5.2.2 operator==(())	102
7.5.2.3 print()	102
7.5.3 Member Data Documentation	103
7.5.3.1 authenticationToken	103
7.5.3.2 currentTime	103
7.5.3.3 expiration	103
7.5.3.4 localData	103
7.5.3.5 noteStoreUrl	103
7.5.3.6 publicUserInfo	103
7.5.3.7 secondFactorDeliveryHint	104
7.5.3.8 secondFactorRequired	104
7.5.3.9 urls	104
7.5.3.10 user	104
7.5.3.11 webApiUrlPrefix	104
7.6 qevercloud::BootstrapInfo Struct Reference	104
7.6.1 Detailed Description	105
7.6.2 Member Function Documentation	105
7.6.2.1 operator!=(())	105
7.6.2.2 operator==(())	105
7.6.2.3 print()	105
7.6.3 Member Data Documentation	105
7.6.3.1 localData	106
7.6.3.2 profiles	106
7.7 qevercloud::BootstrapProfile Struct Reference	106
7.7.1 Detailed Description	106

7.7.2 Member Function Documentation	106
7.7.2.1 operator"!=( )	107
7.7.2.2 operator==( )	107
7.7.2.3 print()	107
7.7.3 Member Data Documentation	107
7.7.3.1 localData	107
7.7.3.2 name	107
7.7.3.3 settings	107
7.8 qevercloud::BootstrapSettings Struct Reference	108
7.8.1 Detailed Description	108
7.8.2 Member Function Documentation	108
7.8.2.1 operator"!=( )	108
7.8.2.2 operator==( )	109
7.8.2.3 print()	109
7.8.3 Member Data Documentation	109
7.8.3.1 accountEmailDomain	109
7.8.3.2 enableFacebookSharing	109
7.8.3.3 enableGiftSubscriptions	109
7.8.3.4 enableGoogle	109
7.8.3.5 enableLinkedInSharing	110
7.8.3.6 enablePublicNotebooks	110
7.8.3.7 enableSharedNotebooks	110
7.8.3.8 enableSingleNoteSharing	110
7.8.3.9 enableSponsoredAccounts	110
7.8.3.10 enableSupportTickets	110
7.8.3.11 enableTwitterSharing	110
7.8.3.12 localData	110
7.8.3.13 marketingUrl	111
7.8.3.14 serviceHost	111
7.8.3.15 supportUrl	111
7.9 qevercloud::BusinessInvitation Struct Reference	111
7.9.1 Detailed Description	112
7.9.2 Member Function Documentation	112
7.9.2.1 operator"!=( )	112
7.9.2.2 operator==( )	112
7.9.2.3 print()	112
7.9.3 Member Data Documentation	112
7.9.3.1 businessId	112
7.9.3.2 created	112
7.9.3.3 email	113
7.9.3.4 fromWorkChat	113
7.9.3.5 localData	113

7.9.3.6 mostRecentReminder . . . . .	113
7.9.3.7 requesterId . . . . .	113
7.9.3.8 role . . . . .	113
7.9.3.9 status . . . . .	113
7.10 qevercloud::BusinessNotebook Struct Reference . . . . .	114
7.10.1 Detailed Description . . . . .	114
7.10.2 Member Function Documentation . . . . .	114
7.10.2.1 operator"!=()" . . . . .	114
7.10.2.2 operator=="()" . . . . .	114
7.10.2.3 print() . . . . .	115
7.10.3 Member Data Documentation . . . . .	115
7.10.3.1 localData . . . . .	115
7.10.3.2 notebookDescription . . . . .	115
7.10.3.3 privilege . . . . .	115
7.10.3.4 recommended . . . . .	115
7.11 qevercloud::BusinessUserAttributes Struct Reference . . . . .	115
7.11.1 Detailed Description . . . . .	116
7.11.2 Member Function Documentation . . . . .	116
7.11.2.1 operator"!=()" . . . . .	116
7.11.2.2 operator=="()" . . . . .	116
7.11.2.3 print() . . . . .	116
7.11.3 Member Data Documentation . . . . .	117
7.11.3.1 companyStartDate . . . . .	117
7.11.3.2 department . . . . .	117
7.11.3.3 linkedInProfileUrl . . . . .	117
7.11.3.4 localData . . . . .	117
7.11.3.5 location . . . . .	117
7.11.3.6 mobilePhone . . . . .	117
7.11.3.7 title . . . . .	117
7.11.3.8 workPhone . . . . .	118
7.12 qevercloud::BusinessUserInfo Struct Reference . . . . .	118
7.12.1 Detailed Description . . . . .	118
7.12.2 Member Function Documentation . . . . .	118
7.12.2.1 operator"!=()" . . . . .	118
7.12.2.2 operator=="()" . . . . .	119
7.12.2.3 print() . . . . .	119
7.12.3 Member Data Documentation . . . . .	119
7.12.3.1 businessId . . . . .	119
7.12.3.2 businessName . . . . .	119
7.12.3.3 email . . . . .	119
7.12.3.4 localData . . . . .	119
7.12.3.5 role . . . . .	120

7.12.3.6 updated	120
7.13 qevercloud::CanMoveToContainerRestrictions Struct Reference	120
7.13.1 Detailed Description	120
7.13.2 Member Function Documentation	120
7.13.2.1 operator!=(())	121
7.13.2.2 operator==(())	121
7.13.2.3 print()	121
7.13.3 Member Data Documentation	121
7.13.3.1 canMoveToContainer	121
7.13.3.2 localData	121
7.14 qevercloud::Contact Struct Reference	121
7.14.1 Detailed Description	122
7.14.2 Member Function Documentation	122
7.14.2.1 operator!=(())	122
7.14.2.2 operator==(())	122
7.14.2.3 print()	122
7.14.3 Member Data Documentation	123
7.14.3.1 id	123
7.14.3.2 localData	123
7.14.3.3 messagingPermit	123
7.14.3.4 messagingPermitExpires	123
7.14.3.5 name	123
7.14.3.6 photoLastUpdated	123
7.14.3.7 photoUrl	124
7.14.3.8 type	124
7.15 qevercloud::CreateOrUpdateNotebookSharesResult Struct Reference	124
7.15.1 Detailed Description	124
7.15.2 Member Function Documentation	125
7.15.2.1 operator!=(())	125
7.15.2.2 operator==(())	125
7.15.2.3 print()	125
7.15.3 Member Data Documentation	125
7.15.3.1 localData	125
7.15.3.2 matchingShares	125
7.15.3.3 updateSequenceNum	126
7.15.4 Property Documentation	126
7.15.4.1 matchingShares	126
7.16 qevercloud::Data Struct Reference	126
7.16.1 Detailed Description	126
7.16.2 Member Function Documentation	127
7.16.2.1 operator!=(())	127
7.16.2.2 operator==(())	127



7.16.2.3 print()	127
7.16.3 Member Data Documentation	127
7.16.3.1 body	127
7.16.3.2 bodyHash	127
7.16.3.3 localData	128
7.16.3.4 size	128
7.17 qevercloud::EDAMInvalidContactsException Class Reference	128
7.17.1 Detailed Description	129
7.17.2 Constructor & Destructor Documentation	129
7.17.2.1 EDAMInvalidContactsException() [1/2]	129
7.17.2.2 ~EDAMInvalidContactsException()	129
7.17.2.3 EDAMInvalidContactsException() [2/2]	129
7.17.3 Member Function Documentation	129
7.17.3.1 exceptionData()	130
7.17.3.2 operator"!="()	130
7.17.3.3 operator==()	130
7.17.3.4 print()	130
7.17.3.5 what()	130
7.17.4 Member Data Documentation	130
7.17.4.1 contacts	130
7.17.4.2 parameter	131
7.17.4.3 reasons	131
7.17.5 Property Documentation	131
7.17.5.1 reasons	131
7.18 qevercloud::EDAMInvalidContactsExceptionData Class Reference	131
7.18.1 Detailed Description	132
7.18.2 Constructor & Destructor Documentation	132
7.18.2.1 EDAMInvalidContactsExceptionData()	132
7.18.3 Member Function Documentation	132
7.18.3.1 throwException()	132
7.18.4 Member Data Documentation	132
7.18.4.1 m_contacts	132
7.18.4.2 m_parameter	132
7.18.4.3 m_reasons	133
7.19 qevercloud::EDAMNotFoundException Class Reference	133
7.19.1 Detailed Description	133
7.19.2 Constructor & Destructor Documentation	134
7.19.2.1 EDAMNotFoundException() [1/2]	134
7.19.2.2 ~EDAMNotFoundException()	134
7.19.2.3 EDAMNotFoundException() [2/2]	134
7.19.3 Member Function Documentation	134
7.19.3.1 exceptionData()	134

7.19.3.2 operator"!=()	134
7.19.3.3 operator==( )	135
7.19.3.4 print()	135
7.19.3.5 what()	135
7.19.4 Member Data Documentation	135
7.19.4.1 identifier	135
7.19.4.2 key	135
7.20 qevercloud::EDAMNotFoundExceptionData Class Reference	135
7.20.1 Detailed Description	136
7.20.2 Constructor & Destructor Documentation	136
7.20.2.1 EDAMNotFoundExceptionData()	136
7.20.3 Member Function Documentation	136
7.20.3.1 throwException()	136
7.20.4 Member Data Documentation	136
7.20.4.1 m_identifier	137
7.20.4.2 m_key	137
7.21 qevercloud::EDAMSystemException Class Reference	137
7.21.1 Detailed Description	138
7.21.2 Constructor & Destructor Documentation	138
7.21.2.1 EDAMSystemException() [1/2]	138
7.21.2.2 ~EDAMSystemException()	138
7.21.2.3 EDAMSystemException() [2/2]	138
7.21.3 Member Function Documentation	138
7.21.3.1 exceptionData()	139
7.21.3.2 operator"!=()	139
7.21.3.3 operator==( )	139
7.21.3.4 print()	139
7.21.3.5 what()	139
7.21.4 Member Data Documentation	139
7.21.4.1 errorCode	139
7.21.4.2 message	140
7.21.4.3 rateLimitDuration	140
7.22 qevercloud::EDAMSystemExceptionAuthExpired Class Reference	140
7.22.1 Detailed Description	140
7.22.2 Member Function Documentation	140
7.22.2.1 exceptionData()	141
7.23 qevercloud::EDAMSystemExceptionAuthExpiredData Class Reference	141
7.23.1 Detailed Description	141
7.23.2 Constructor & Destructor Documentation	141
7.23.2.1 EDAMSystemExceptionAuthExpiredData()	142
7.23.3 Member Function Documentation	142
7.23.3.1 throwException()	142

7.24 qevercloud::EDAMSystemExceptionData Class Reference	142
7.24.1 Detailed Description	143
7.24.2 Constructor & Destructor Documentation	143
7.24.2.1 EDAMSystemExceptionData()	143
7.24.3 Member Function Documentation	143
7.24.3.1 throwException()	143
7.24.4 Member Data Documentation	143
7.24.4.1 m_errorCode	143
7.24.4.2 m_message	144
7.24.4.3 m_rateLimitDuration	144
7.25 qevercloud::EDAMSystemExceptionRateLimitReached Class Reference	144
7.25.1 Detailed Description	144
7.25.2 Member Function Documentation	144
7.25.2.1 exceptionData()	145
7.26 qevercloud::EDAMSystemExceptionRateLimitReachedData Class Reference	145
7.26.1 Detailed Description	145
7.26.2 Constructor & Destructor Documentation	145
7.26.2.1 EDAMSystemExceptionRateLimitReachedData()	146
7.26.3 Member Function Documentation	146
7.26.3.1 throwException()	146
7.27 qevercloud::EDAMUserException Class Reference	146
7.27.1 Detailed Description	147
7.27.2 Constructor & Destructor Documentation	147
7.27.2.1 EDAMUserException() [1/2]	147
7.27.2.2 ~EDAMUserException()	147
7.27.2.3 EDAMUserException() [2/2]	147
7.27.3 Member Function Documentation	147
7.27.3.1 exceptionData()	148
7.27.3.2 operator!=(())	148
7.27.3.3 operator==(())	148
7.27.3.4 print()	148
7.27.3.5 what()	148
7.27.4 Member Data Documentation	148
7.27.4.1 errorCode	148
7.27.4.2 parameter	149
7.28 qevercloud::EDAMUserExceptionData Class Reference	149
7.28.1 Detailed Description	149
7.28.2 Constructor & Destructor Documentation	149
7.28.2.1 EDAMUserExceptionData()	149
7.28.3 Member Function Documentation	150
7.28.3.1 throwException()	150
7.28.4 Member Data Documentation	150

7.28.4.1 m_errorCode	150
7.28.4.2 m_parameter	150
7.29 qevercloud::EventLoopFinisher Class Reference	150
7.29.1 Constructor & Destructor Documentation	151
7.29.1.1 EventLoopFinisher()	151
7.29.1.2 ~EventLoopFinisher()	151
7.29.2 Member Function Documentation	151
7.29.2.1 stopEventLoop	151
7.30 qevercloud::EverCloudException Class Reference	151
7.30.1 Detailed Description	152
7.30.2 Constructor & Destructor Documentation	152
7.30.2.1 EverCloudException() [1/4]	152
7.30.2.2 EverCloudException() [2/4]	152
7.30.2.3 EverCloudException() [3/4]	152
7.30.2.4 EverCloudException() [4/4]	152
7.30.2.5 ~EverCloudException()	153
7.30.3 Member Function Documentation	153
7.30.3.1 exceptionData()	153
7.30.3.2 what()	153
7.30.4 Member Data Documentation	153
7.30.4.1 m_error	153
7.31 qevercloud::EverCloudExceptionData Class Reference	153
7.31.1 Detailed Description	154
7.31.2 Constructor & Destructor Documentation	155
7.31.2.1 EverCloudExceptionData()	155
7.31.3 Member Function Documentation	155
7.31.3.1 throwException()	155
7.31.4 Member Data Documentation	155
7.31.4.1 errorMessage	155
7.32 qevercloud::EverCloudLocalData Class Reference	155
7.32.1 Detailed Description	156
7.32.2 Member Typedef Documentation	156
7.32.2.1 Dict	156
7.32.3 Constructor & Destructor Documentation	157
7.32.3.1 EverCloudLocalData()	157
7.32.3.2 ~EverCloudLocalData()	157
7.32.4 Member Function Documentation	157
7.32.4.1 operator!=(())	157
7.32.4.2 operator==(())	157
7.32.4.3 print()	157
7.32.5 Member Data Documentation	157
7.32.5.1 bool	158

7.32.5.2 dict	158
7.32.5.3 dirty	158
7.32.5.4 favorited	158
7.32.5.5 id	158
7.32.6 Property Documentation	158
7.32.6.1 dict	159
7.32.6.2 local	159
7.33 qevercloud::EvernoteException Class Reference	159
7.33.1 Detailed Description	159
7.33.2 Constructor & Destructor Documentation	159
7.33.2.1 EvernoteException() [1/4]	159
7.33.2.2 EvernoteException() [2/4]	160
7.33.2.3 EvernoteException() [3/4]	160
7.33.2.4 EvernoteException() [4/4]	160
7.33.3 Member Function Documentation	160
7.33.3.1 exceptionData()	160
7.34 qevercloud::EvernoteExceptionData Class Reference	160
7.34.1 Detailed Description	161
7.34.2 Constructor & Destructor Documentation	161
7.34.2.1 EvernoteExceptionData()	161
7.34.3 Member Function Documentation	161
7.34.3.1 throwException()	161
7.35 qevercloud::EvernoteOAuthDialog Class Reference	161
7.35.1 Detailed Description	162
7.35.2 Member Typedef Documentation	162
7.35.2.1 OAuthResult	162
7.35.3 Constructor & Destructor Documentation	162
7.35.3.1 EvernoteOAuthDialog()	162
7.35.3.2 ~EvernoteOAuthDialog()	163
7.35.4 Member Function Documentation	163
7.35.4.1 exec()	163
7.35.4.2 isSucceeded()	163
7.35.4.3 oauthError()	163
7.35.4.4 oauthResult()	164
7.35.4.5 open()	164
7.35.4.6 setWebViewSizeHint()	164
7.36 qevercloud::EvernoteOAuthWebView Class Reference	164
7.36.1 Detailed Description	165
7.36.2 Constructor & Destructor Documentation	165
7.36.2.1 EvernoteOAuthWebView()	165
7.36.3 Member Function Documentation	165
7.36.3.1 authenticate()	165

7.36.3.2 authenticationFailed	166
7.36.3.3 authenticationFinished	166
7.36.3.4 authenticationSucceeded	166
7.36.3.5 isSucceeded()	166
7.36.3.6 oauthError()	166
7.36.3.7 oauthResult()	167
7.36.3.8 setSizeHint()	167
7.36.3.9 sizeHint()	167
7.37 qevercloud::Identity Struct Reference	167
7.37.1 Detailed Description	168
7.37.2 Member Function Documentation	168
7.37.2.1 operator"!=()	168
7.37.2.2 operator==(())	168
7.37.2.3 print()	168
7.37.3 Member Data Documentation	168
7.37.3.1 blocked	168
7.37.3.2 contact	168
7.37.3.3 deactivated	169
7.37.3.4 eventId	169
7.37.3.5 id	169
7.37.3.6 localData	169
7.37.3.7 sameBusiness	169
7.37.3.8 userConnected	169
7.37.3.9 userId	170
7.38 qevercloud::IDurableService Class Reference	170
7.38.1 Member Typedef Documentation	170
7.38.1.1 AsyncServiceCall	170
7.38.1.2 SyncResult	170
7.38.1.3 SyncServiceCall	171
7.38.2 Member Function Documentation	171
7.38.2.1 executeAsyncRequest()	171
7.38.2.2 executeSyncRequest()	171
7.39 qevercloud::ILogger Class Reference	171
7.39.1 Member Function Documentation	171
7.39.1.1 level()	171
7.39.1.2 log()	172
7.39.1.3 setLevel()	172
7.39.1.4 shouldLog()	172
7.40 qevercloud::InkNoteImageDownloader Class Reference	172
7.40.1 Detailed Description	173
7.40.2 Constructor & Destructor Documentation	173
7.40.2.1 InkNoteImageDownloader() [1/2]	173

7.40.2.2 InkNoteImageDownloader() [2/2]	173
7.40.2.3 ~InkNoteImageDownloader()	174
7.40.3 Member Function Documentation	174
7.40.3.1 download()	174
7.40.3.2 setAuthenticationToken()	174
7.40.3.3 setHeight()	175
7.40.3.4 setHost()	175
7.40.3.5 setShardId()	175
7.40.3.6 setWidth()	175
7.41 qevercloud::INoteStore Class Reference	176
7.41.1 Detailed Description	180
7.41.2 Constructor & Destructor Documentation	180
7.41.2.1 INoteStore()	180
7.41.3 Member Function Documentation	180
7.41.3.1 authenticateToSharedNote()	180
7.41.3.2 authenticateToSharedNoteAsync()	181
7.41.3.3 authenticateToSharedNotebook()	181
7.41.3.4 authenticateToSharedNotebookAsync()	182
7.41.3.5 copyNote()	182
7.41.3.6 copyNoteAsync()	183
7.41.3.7 createLinkedNotebook()	183
7.41.3.8 createLinkedNotebookAsync()	184
7.41.3.9 createNote()	184
7.41.3.10 createNoteAsync()	186
7.41.3.11 createNotebook()	186
7.41.3.12 createNotebookAsync()	187
7.41.3.13 createOrUpdateNotebookShares()	187
7.41.3.14 createOrUpdateNotebookSharesAsync()	188
7.41.3.15 createSearch()	188
7.41.3.16 createSearchAsync()	189
7.41.3.17 createTag()	189
7.41.3.18 createTagAsync()	190
7.41.3.19 deleteNote()	190
7.41.3.20 deleteNoteAsync()	191
7.41.3.21 emailNote()	191
7.41.3.22 emailNoteAsync()	192
7.41.3.23 expungeLinkedNotebook()	192
7.41.3.24 expungeLinkedNotebookAsync()	193
7.41.3.25 expungeNote()	193
7.41.3.26 expungeNoteAsync()	193
7.41.3.27 expungeNotebook()	194
7.41.3.28 expungeNotebookAsync()	194

7.41.3.29 expungeSearch()	194
7.41.3.30 expungeSearchAsync()	195
7.41.3.31 expungeTag()	195
7.41.3.32 expungeTagAsync()	196
7.41.3.33 findNoteCounts()	196
7.41.3.34 findNoteCountsAsync()	197
7.41.3.35 findNoteOffset()	197
7.41.3.36 findNoteOffsetAsync()	198
7.41.3.37 findNotesMetadata()	198
7.41.3.38 findNotesMetadataAsync()	199
7.41.3.39 findRelated()	199
7.41.3.40 findRelatedAsync()	200
7.41.3.41 getDefaultNotebook()	201
7.41.3.42 getDefaultNotebookAsync()	201
7.41.3.43 getFilteredSyncChunk()	201
7.41.3.44 getFilteredSyncChunkAsync()	202
7.41.3.45 getLinkedNotebookSyncChunk()	202
7.41.3.46 getLinkedNotebookSyncChunkAsync()	203
7.41.3.47 getLinkedNotebookSyncState()	203
7.41.3.48 getLinkedNotebookSyncStateAsync()	204
7.41.3.49 getNote()	204
7.41.3.50 getNoteApplicationData()	204
7.41.3.51 getNoteApplicationDataAsync()	205
7.41.3.52 getNoteApplicationDataEntry()	205
7.41.3.53 getNoteApplicationDataEntryAsync()	205
7.41.3.54 getNoteAsync()	205
7.41.3.55 getNotebook()	206
7.41.3.56 getNotebookAsync()	207
7.41.3.57 getNotebookShares()	207
7.41.3.58 getNotebookSharesAsync()	207
7.41.3.59 getNoteContent()	207
7.41.3.60 getNoteContentAsync()	208
7.41.3.61 getNoteSearchText()	208
7.41.3.62 getNoteSearchTextAsync()	209
7.41.3.63 getNoteTagNames()	209
7.41.3.64 getNoteTagNamesAsync()	209
7.41.3.65 getNoteVersion()	210
7.41.3.66 getNoteVersionAsync()	210
7.41.3.67 getNoteWithResultSpec()	211
7.41.3.68 getNoteWithResultSpecAsync()	211
7.41.3.69 getPublicNotebook()	211
7.41.3.70 getPublicNotebookAsync()	212



7.41.3.71 getResource()	212
7.41.3.72 getResourceAlternateData()	213
7.41.3.73 getResourceAlternateDataAsync()	214
7.41.3.74 getResourceApplicationData()	214
7.41.3.75 getResourceApplicationDataAsync()	214
7.41.3.76 getResourceApplicationDataEntry()	214
7.41.3.77 getResourceApplicationDataEntryAsync()	214
7.41.3.78 getResourceAsync()	215
7.41.3.79 getResourceAttributes()	215
7.41.3.80 getResourceAttributesAsync()	215
7.41.3.81 getResourceByHash()	216
7.41.3.82 getResourceByHashAsync()	216
7.41.3.83 getResourceData()	217
7.41.3.84 getResourceDataAsync()	217
7.41.3.85 getResourceRecognition()	217
7.41.3.86 getResourceRecognitionAsync()	218
7.41.3.87 getResourceSearchText()	218
7.41.3.88 getResourceSearchTextAsync()	219
7.41.3.89 getSearch()	219
7.41.3.90 getSearchAsync()	219
7.41.3.91 getSharedNotebookByAuth()	219
7.41.3.92 getSharedNotebookByAuthAsync()	220
7.41.3.93 getSyncState()	220
7.41.3.94 getSyncStateAsync()	220
7.41.3.95 getTag()	220
7.41.3.96 getTagAsync()	221
7.41.3.97 listAccessibleBusinessNotebooks()	221
7.41.3.98 listAccessibleBusinessNotebooksAsync()	222
7.41.3.99 listLinkedNotebooks()	222
7.41.3.100 listLinkedNotebooksAsync()	222
7.41.3.101 listNotebooks()	222
7.41.3.102 listNotebooksAsync()	222
7.41.3.103 listNoteVersions()	222
7.41.3.104 listNoteVersionsAsync()	223
7.41.3.105 listSearches()	223
7.41.3.106 listSearchesAsync()	223
7.41.3.107 listSharedNotebooks()	223
7.41.3.108 listSharedNotebooksAsync()	224
7.41.3.109 listTags()	224
7.41.3.110 listTagsAsync()	224
7.41.3.111 listTagsByNotebook()	224
7.41.3.112 listTagsByNotebookAsync()	224

7.41.3.113	manageNotebookShares()	225
7.41.3.114	manageNotebookSharesAsync()	225
7.41.3.115	noteStoreUrl()	225
7.41.3.116	setNoteApplicationDataEntry()	225
7.41.3.117	setNoteApplicationDataEntryAsync()	226
7.41.3.118	setNotebookRecipientSettings()	226
7.41.3.119	setNotebookRecipientSettingsAsync()	227
7.41.3.120	setNoteStoreUrl()	227
7.41.3.121	setResourceApplicationDataEntry()	227
7.41.3.122	setResourceApplicationDataEntryAsync()	227
7.41.3.123	shareNote()	228
7.41.3.124	shareNoteAsync()	228
7.41.3.125	shareNotebook()	228
7.41.3.126	shareNotebookAsync()	230
7.41.3.127	stopSharingNote()	230
7.41.3.128	stopSharingNoteAsync()	231
7.41.3.129	unsetNoteApplicationDataEntry()	231
7.41.3.130	unsetNoteApplicationDataEntryAsync()	231
7.41.3.131	unsetResourceApplicationDataEntry()	231
7.41.3.132	unsetResourceApplicationDataEntryAsync()	232
7.41.3.133	untagAll()	232
7.41.3.134	untagAllAsync()	232
7.41.3.135	updateLinkedNotebook()	232
7.41.3.136	updateLinkedNotebookAsync()	233
7.41.3.137	updateNote()	233
7.41.3.138	updateNoteAsync()	234
7.41.3.139	updateNotebook()	235
7.41.3.140	updateNotebookAsync()	235
7.41.3.141	updateNoteIfUsnMatches()	236
7.41.3.142	updateNoteIfUsnMatchesAsync()	236
7.41.3.143	updateResource()	236
7.41.3.144	updateResourceAsync()	237
7.41.3.145	updateSearch()	237
7.41.3.146	updateSearchAsync()	238
7.41.3.147	updateSharedNotebook()	238
7.41.3.148	updateSharedNotebookAsync()	238
7.41.3.149	updateTag()	239
7.41.3.150	updateTagAsync()	240
7.42	qevercloud::InvitationShareRelationship Struct Reference	240
7.42.1	Detailed Description	241
7.42.2	Member Function Documentation	241
7.42.2.1	operator"!="()	241

7.42.2.2 operator==( ) . . . . .	241
7.42.2.3 print() . . . . .	241
7.42.3 Member Data Documentation . . . . .	241
7.42.3.1 displayName . . . . .	242
7.42.3.2 localData . . . . .	242
7.42.3.3 privilege . . . . .	242
7.42.3.4 recipientUserIdentity . . . . .	242
7.42.3.5 sharerUserId . . . . .	242
7.43 qevercloud::IRequestContext Class Reference . . . . .	242
7.43.1 Detailed Description . . . . .	243
7.43.2 Constructor & Destructor Documentation . . . . .	243
7.43.2.1 ~IRequestContext() . . . . .	243
7.43.3 Member Function Documentation . . . . .	243
7.43.3.1 authenticationToken() . . . . .	243
7.43.3.2 clone() . . . . .	243
7.43.3.3 cookies() . . . . .	243
7.43.3.4 increaseRequestTimeoutExponentially() . . . . .	243
7.43.3.5 maxRequestRetryCount() . . . . .	244
7.43.3.6 maxRequestTimeout() . . . . .	244
7.43.3.7 requestId() . . . . .	244
7.43.3.8 requestTimeout() . . . . .	244
7.43.4 Friends And Related Function Documentation . . . . .	244
7.43.4.1 operator<< [1/2] . . . . .	244
7.43.4.2 operator<< [2/2] . . . . .	244
7.44 qevercloud::IRetryPolicy Struct Reference . . . . .	244
7.44.1 Member Function Documentation . . . . .	245
7.44.1.1 shouldRetry() . . . . .	245
7.45 qevercloud::QAssociativeContainerConstReferenceWrapper< Container >::iterator Struct Reference . . . . .	245
7.45.1 Constructor & Destructor Documentation . . . . .	245
7.45.1.1 iterator() . . . . .	245
7.45.2 Member Function Documentation . . . . .	245
7.45.2.1 operator!=( ) . . . . .	246
7.45.2.2 operator*( ) . . . . .	246
7.45.2.3 operator++( ) . . . . .	246
7.45.3 Member Data Documentation . . . . .	246
7.45.3.1 m_iterator . . . . .	246
7.46 qevercloud::QAssociativeContainerReferenceWrapper< Container >::iterator Struct Reference . . . . .	246
7.46.1 Constructor & Destructor Documentation . . . . .	247
7.46.1.1 iterator() . . . . .	247
7.46.2 Member Function Documentation . . . . .	247
7.46.2.1 operator!=( ) . . . . .	247
7.46.2.2 operator*( ) . . . . .	247

7.46.2.3 operator++()	247
7.46.3 Member Data Documentation	247
7.46.3.1 m_iterator	248
7.47 qevercloud::IUserStore Class Reference	248
7.47.1 Detailed Description	249
7.47.2 Constructor & Destructor Documentation	249
7.47.2.1 IUserStore()	249
7.47.3 Member Function Documentation	249
7.47.3.1 authenticateLongSession()	250
7.47.3.2 authenticateLongSessionAsync()	251
7.47.3.3 authenticateToBusiness()	251
7.47.3.4 authenticateToBusinessAsync()	252
7.47.3.5 checkVersion()	252
7.47.3.6 checkVersionAsync()	253
7.47.3.7 completeTwoFactorAuthentication()	253
7.47.3.8 completeTwoFactorAuthenticationAsync()	254
7.47.3.9 getAccountLimits()	254
7.47.3.10 getAccountLimitsAsync()	255
7.47.3.11 getBootstrapInfo()	255
7.47.3.12 getBootstrapInfoAsync()	255
7.47.3.13 getPublicUserInfo()	255
7.47.3.14 getPublicUserInfoAsync()	256
7.47.3.15 getUser()	256
7.47.3.16 getUserAsync()	256
7.47.3.17 getUserUrls()	256
7.47.3.18 getUserUrlsAsync()	256
7.47.3.19 inviteToBusiness()	257
7.47.3.20 inviteToBusinessAsync()	257
7.47.3.21 listBusinessInvitations()	258
7.47.3.22 listBusinessInvitationsAsync()	258
7.47.3.23 listBusinessUsers()	258
7.47.3.24 listBusinessUsersAsync()	259
7.47.3.25 removeFromBusiness()	259
7.47.3.26 removeFromBusinessAsync()	259
7.47.3.27 revokeLongSession()	260
7.47.3.28 revokeLongSessionAsync()	260
7.47.3.29 setUserStoreUrl()	260
7.47.3.30 updateBusinessUserIdentifier()	260
7.47.3.31 updateBusinessUserIdentifierAsync()	261
7.47.3.32 userStoreUrl()	262
7.48 qevercloud::LazyMap Struct Reference	262
7.48.1 Detailed Description	262

7.48.2 Member Typedef Documentation	263
7.48.2.1 FullMap	263
7.48.3 Member Function Documentation	263
7.48.3.1 operator"!=( )	263
7.48.3.2 operator==( )	263
7.48.3.3 print()	263
7.48.4 Member Data Documentation	263
7.48.4.1 fullMap	263
7.48.4.2 keysOnly	264
7.48.4.3 localData	264
7.48.5 Property Documentation	264
7.48.5.1 fullMap	264
7.48.5.2 keysOnly	264
7.49 qevercloud::LinkedNotebook Struct Reference	264
7.49.1 Detailed Description	265
7.49.2 Member Function Documentation	265
7.49.2.1 operator"!=( )	265
7.49.2.2 operator==( )	265
7.49.2.3 print()	265
7.49.3 Member Data Documentation	265
7.49.3.1 businessId	266
7.49.3.2 guid	266
7.49.3.3 localData	266
7.49.3.4 noteStoreUrl	266
7.49.3.5 shardId	266
7.49.3.6 sharedNotebookGlobalId	266
7.49.3.7 shareName	267
7.49.3.8 stack	267
7.49.3.9 updateSequenceNum	267
7.49.3.10 uri	267
7.49.3.11 username	267
7.49.3.12 webApiUrlPrefix	267
7.50 qevercloud::ManageNotebookSharesError Struct Reference	268
7.50.1 Detailed Description	268
7.50.2 Member Function Documentation	268
7.50.2.1 operator"!=( )	268
7.50.2.2 operator==( )	269
7.50.2.3 print()	269
7.50.3 Member Data Documentation	269
7.50.3.1 localData	269
7.50.3.2 notFoundException	269
7.50.3.3 userException	269

7.50.3.4 userIdentity . . . . .	269
7.51 qevercloud::ManageNotebookSharesParameters Struct Reference . . . . .	270
7.51.1 Detailed Description . . . . .	270
7.51.2 Member Function Documentation . . . . .	270
7.51.2.1 operator"!=( ) . . . . .	270
7.51.2.2 operator==( ) . . . . .	271
7.51.2.3 print() . . . . .	271
7.51.3 Member Data Documentation . . . . .	271
7.51.3.1 invitationsToCreateOrUpdate . . . . .	271
7.51.3.2 inviteMessage . . . . .	271
7.51.3.3 localData . . . . .	271
7.51.3.4 membershipsToUpdate . . . . .	271
7.51.3.5 notebookGuid . . . . .	272
7.51.3.6 unshares . . . . .	272
7.51.4 Property Documentation . . . . .	272
7.51.4.1 invitationsToCreateOrUpdate . . . . .	272
7.51.4.2 membershipsToUpdate . . . . .	272
7.51.4.3 unshares . . . . .	272
7.52 qevercloud::ManageNotebookSharesResult Struct Reference . . . . .	272
7.52.1 Detailed Description . . . . .	273
7.52.2 Member Function Documentation . . . . .	273
7.52.2.1 operator"!=( ) . . . . .	273
7.52.2.2 operator==( ) . . . . .	273
7.52.2.3 print() . . . . .	273
7.52.3 Member Data Documentation . . . . .	274
7.52.3.1 errors . . . . .	274
7.52.3.2 localData . . . . .	274
7.52.4 Property Documentation . . . . .	274
7.52.4.1 errors . . . . .	274
7.53 qevercloud::ManageNoteSharesError Struct Reference . . . . .	274
7.53.1 Detailed Description . . . . .	275
7.53.2 Member Function Documentation . . . . .	275
7.53.2.1 operator"!=( ) . . . . .	275
7.53.2.2 operator==( ) . . . . .	275
7.53.2.3 print() . . . . .	275
7.53.3 Member Data Documentation . . . . .	275
7.53.3.1 identityID . . . . .	276
7.53.3.2 localData . . . . .	276
7.53.3.3 notFoundException . . . . .	276
7.53.3.4 userException . . . . .	276
7.53.3.5 userID . . . . .	276
7.54 qevercloud::ManageNoteSharesParameters Struct Reference . . . . .	276

7.54.1 Detailed Description	277
7.54.2 Member Function Documentation	277
7.54.2.1 operator"!=()	277
7.54.2.2 operator==( )	277
7.54.2.3 print()	278
7.54.3 Member Data Documentation	278
7.54.3.1 invitationsToUnshare	278
7.54.3.2 invitationsToUpdate	278
7.54.3.3 localData	278
7.54.3.4 membershipsToUnshare	278
7.54.3.5 membershipsToUpdate	278
7.54.3.6 noteGuid	279
7.54.4 Property Documentation	279
7.54.4.1 invitationsToUnshare	279
7.54.4.2 invitationsToUpdate	279
7.54.4.3 membershipsToUnshare	279
7.54.4.4 membershipsToUpdate	279
7.55 qevercloud::ManageNoteSharesResult Struct Reference	279
7.55.1 Detailed Description	280
7.55.2 Member Function Documentation	280
7.55.2.1 operator"!=()	280
7.55.2.2 operator==( )	280
7.55.2.3 print()	280
7.55.3 Member Data Documentation	281
7.55.3.1 errors	281
7.55.3.2 localData	281
7.55.4 Property Documentation	281
7.55.4.1 errors	281
7.56 qevercloud::MemberShareRelationship Struct Reference	281
7.56.1 Detailed Description	282
7.56.2 Member Function Documentation	282
7.56.2.1 operator"!=()	282
7.56.2.2 operator==( )	282
7.56.2.3 print()	282
7.56.3 Member Data Documentation	282
7.56.3.1 bestPrivilege	283
7.56.3.2 displayName	283
7.56.3.3 individualPrivilege	283
7.56.3.4 localData	283
7.56.3.5 recipientUserId	283
7.56.3.6 restrictions	283
7.56.3.7 sharerUserId	283

7.57 qevercloud::NetworkException Class Reference	284
7.57.1 Detailed Description	284
7.57.2 Constructor & Destructor Documentation	284
7.57.2.1 NetworkException() [1/3]	284
7.57.2.2 NetworkException() [2/3]	285
7.57.2.3 NetworkException() [3/3]	285
7.57.2.4 ~NetworkException()	285
7.57.3 Member Function Documentation	285
7.57.3.1 exceptionData()	285
7.57.3.2 operator!=(())	285
7.57.3.3 operator==(())	285
7.57.3.4 type()	286
7.57.3.5 what()	286
7.57.4 Member Data Documentation	286
7.57.4.1 m_type	286
7.58 qevercloud::NetworkExceptionData Class Reference	286
7.58.1 Detailed Description	287
7.58.2 Constructor & Destructor Documentation	287
7.58.2.1 NetworkExceptionData()	287
7.58.3 Member Function Documentation	287
7.58.3.1 throwException()	287
7.58.4 Member Data Documentation	287
7.58.4.1 m_type	287
7.59 qevercloud::Note Struct Reference	287
7.59.1 Detailed Description	288
7.59.2 Member Function Documentation	288
7.59.2.1 operator!=(())	288
7.59.2.2 operator==(())	289
7.59.2.3 print()	289
7.59.3 Member Data Documentation	289
7.59.3.1 active	289
7.59.3.2 attributes	289
7.59.3.3 content	289
7.59.3.4 contentHash	289
7.59.3.5 contentLength	290
7.59.3.6 created	290
7.59.3.7 deleted	290
7.59.3.8 guid	290
7.59.3.9 limits	290
7.59.3.10 localData	290
7.59.3.11 notebookGuid	291
7.59.3.12 resources	291



7.59.3.13 restrictions	291
7.59.3.14 sharedNotes	291
7.59.3.15 tagGuids	291
7.59.3.16 tagNames	291
7.59.3.17 title	292
7.59.3.18 updated	292
7.59.3.19 updateSequenceNum	292
7.59.4 Property Documentation	292
7.59.4.1 resources	292
7.59.4.2 sharedNotes	292
7.59.4.3 tagGuids	292
7.60 qevercloud::NoteAttributes Struct Reference	293
7.60.1 Detailed Description	294
7.60.2 Member Typedef Documentation	294
7.60.2.1 Classifications	294
7.60.3 Member Function Documentation	294
7.60.3.1 operator"!=( )	294
7.60.3.2 operator==( )	294
7.60.3.3 print()	294
7.60.4 Member Data Documentation	294
7.60.4.1 altitude	295
7.60.4.2 applicationData	295
7.60.4.3 author	295
7.60.4.4 classifications	295
7.60.4.5 conflictSourceNoteGuid	295
7.60.4.6 contentClass	296
7.60.4.7 creatorId	296
7.60.4.8 lastEditedBy	296
7.60.4.9 lastEditorId	296
7.60.4.10 latitude	296
7.60.4.11 localData	297
7.60.4.12 longitude	297
7.60.4.13 noteTitleQuality	297
7.60.4.14 placeName	297
7.60.4.15 reminderDoneTime	297
7.60.4.16 reminderOrder	298
7.60.4.17 reminderTime	298
7.60.4.18 shareDate	298
7.60.4.19 sharedWithBusiness	298
7.60.4.20 source	298
7.60.4.21 sourceApplication	299
7.60.4.22 sourceURL	299

7.60.4.23 subjectDate . . . . .	299
7.60.5 Property Documentation . . . . .	299
7.60.5.1 classifications . . . . .	299
7.61 qevercloud::Notebook Struct Reference . . . . .	299
7.61.1 Detailed Description . . . . .	300
7.61.2 Member Function Documentation . . . . .	300
7.61.2.1 operator"!=( ) . . . . .	300
7.61.2.2 operator==( ) . . . . .	300
7.61.2.3 print() . . . . .	301
7.61.3 Member Data Documentation . . . . .	301
7.61.3.1 businessNotebook . . . . .	301
7.61.3.2 contact . . . . .	301
7.61.3.3 defaultNotebook . . . . .	301
7.61.3.4 guid . . . . .	301
7.61.3.5 localData . . . . .	302
7.61.3.6 name . . . . .	302
7.61.3.7 published . . . . .	302
7.61.3.8 publishing . . . . .	302
7.61.3.9 recipientSettings . . . . .	302
7.61.3.10 restrictions . . . . .	302
7.61.3.11 serviceCreated . . . . .	303
7.61.3.12 serviceUpdated . . . . .	303
7.61.3.13 sharedNotebookIds . . . . .	303
7.61.3.14 sharedNotebooks . . . . .	303
7.61.3.15 stack . . . . .	303
7.61.3.16 updateSequenceNum . . . . .	303
7.61.4 Property Documentation . . . . .	304
7.61.4.1 sharedNotebookIds . . . . .	304
7.61.4.2 sharedNotebooks . . . . .	304
7.62 qevercloud::NotebookDescriptor Struct Reference . . . . .	304
7.62.1 Detailed Description . . . . .	304
7.62.2 Member Function Documentation . . . . .	305
7.62.2.1 operator"!=( ) . . . . .	305
7.62.2.2 operator==( ) . . . . .	305
7.62.2.3 print() . . . . .	305
7.62.3 Member Data Documentation . . . . .	305
7.62.3.1 contactName . . . . .	305
7.62.3.2 guid . . . . .	305
7.62.3.3 hasSharedNotebook . . . . .	305
7.62.3.4 joinedUserCount . . . . .	306
7.62.3.5 localData . . . . .	306
7.62.3.6 notebookDisplayName . . . . .	306

7.63 qevercloud::NotebookRecipientSettings Struct Reference . . . . .	306
7.63.1 Detailed Description . . . . .	307
7.63.2 Member Function Documentation . . . . .	307
7.63.2.1 operator!=(()) . . . . .	307
7.63.2.2 operator==(()) . . . . .	307
7.63.2.3 print() . . . . .	307
7.63.3 Member Data Documentation . . . . .	307
7.63.3.1 inMyList . . . . .	307
7.63.3.2 localData . . . . .	308
7.63.3.3 recipientStatus . . . . .	308
7.63.3.4 reminderNotifyEmail . . . . .	308
7.63.3.5 reminderNotifyInApp . . . . .	308
7.63.3.6 stack . . . . .	308
7.64 qevercloud::NotebookRestrictions Struct Reference . . . . .	308
7.64.1 Detailed Description . . . . .	309
7.64.2 Member Function Documentation . . . . .	310
7.64.2.1 operator!=(()) . . . . .	310
7.64.2.2 operator==(()) . . . . .	310
7.64.2.3 print() . . . . .	310
7.64.3 Member Data Documentation . . . . .	310
7.64.3.1 canMoveToContainerRestrictions . . . . .	310
7.64.3.2 expungeWhichSharedNotebookRestrictions . . . . .	310
7.64.3.3 localData . . . . .	311
7.64.3.4 noCanMoveNote . . . . .	311
7.64.3.5 noChangeContact . . . . .	311
7.64.3.6 noCreateNotes . . . . .	311
7.64.3.7 noCreateSharedNotebooks . . . . .	311
7.64.3.8 noCreateTags . . . . .	311
7.64.3.9 noEmailNotes . . . . .	311
7.64.3.10 noExpungeNotebook . . . . .	312
7.64.3.11 noExpungeNotes . . . . .	312
7.64.3.12 noExpungeTags . . . . .	312
7.64.3.13 noPublishToBusinessLibrary . . . . .	312
7.64.3.14 noPublishToPublic . . . . .	312
7.64.3.15 noReadNotes . . . . .	312
7.64.3.16 noRenameNotebook . . . . .	312
7.64.3.17 noSendMessageToRecipients . . . . .	313
7.64.3.18 noSetDefaultNotebook . . . . .	313
7.64.3.19 noSetInMyList . . . . .	313
7.64.3.20 noSetNotebookStack . . . . .	313
7.64.3.21 noSetParentTag . . . . .	313
7.64.3.22 noSetRecipientSettingsStack . . . . .	313

7.64.3.23 noSetReminderNotifyEmail . . . . .	313
7.64.3.24 noSetReminderNotifyInApp . . . . .	314
7.64.3.25 noShareNotes . . . . .	314
7.64.3.26 noShareNotesWithBusiness . . . . .	314
7.64.3.27 noUpdateNotebook . . . . .	314
7.64.3.28 noUpdateNotes . . . . .	314
7.64.3.29 noUpdateTags . . . . .	314
7.64.3.30 updateWhichSharedNotebookRestrictions . . . . .	314
7.65 qevercloud::NotebookShareTemplate Struct Reference . . . . .	315
7.65.1 Detailed Description . . . . .	315
7.65.2 Member Function Documentation . . . . .	315
7.65.2.1 operator"!=( ) . . . . .	315
7.65.2.2 operator==( ) . . . . .	316
7.65.2.3 print() . . . . .	316
7.65.3 Member Data Documentation . . . . .	316
7.65.3.1 localData . . . . .	316
7.65.3.2 notebookGuid . . . . .	316
7.65.3.3 privilege . . . . .	316
7.65.3.4 recipientContacts . . . . .	316
7.65.3.5 recipientThreadId . . . . .	317
7.65.4 Property Documentation . . . . .	317
7.65.4.1 recipientContacts . . . . .	317
7.66 qevercloud::NoteCollectionCounts Struct Reference . . . . .	317
7.66.1 Detailed Description . . . . .	318
7.66.2 Member Typedef Documentation . . . . .	318
7.66.2.1 TagCounts . . . . .	318
7.66.3 Member Function Documentation . . . . .	318
7.66.3.1 operator"!=( ) . . . . .	318
7.66.3.2 operator==( ) . . . . .	318
7.66.3.3 print() . . . . .	318
7.66.4 Member Data Documentation . . . . .	319
7.66.4.1 localData . . . . .	319
7.66.4.2 notebookCounts . . . . .	319
7.66.4.3 tagCounts . . . . .	319
7.66.4.4 trashCount . . . . .	319
7.66.5 Property Documentation . . . . .	319
7.66.5.1 notebookCounts . . . . .	319
7.66.5.2 tagCounts . . . . .	319
7.67 qevercloud::NoteEmailParameters Struct Reference . . . . .	320
7.67.1 Detailed Description . . . . .	320
7.67.2 Member Function Documentation . . . . .	320
7.67.2.1 operator"!=( ) . . . . .	320

7.67.2.2 operator==( )	321
7.67.2.3 print()	321
7.67.3 Member Data Documentation	321
7.67.3.1 ccAddresses	321
7.67.3.2 guid	321
7.67.3.3 localData	321
7.67.3.4 message	321
7.67.3.5 note	322
7.67.3.6 subject	322
7.67.3.7 toAddresses	322
7.68 qevercloud::NoteFilter Struct Reference	322
7.68.1 Detailed Description	323
7.68.2 Member Function Documentation	323
7.68.2.1 operator!=( )	323
7.68.2.2 operator==( )	323
7.68.2.3 print()	324
7.68.3 Member Data Documentation	324
7.68.3.1 ascending	324
7.68.3.2 context	324
7.68.3.3 emphasized	324
7.68.3.4 inactive	324
7.68.3.5 includeAllReadableNotebooks	324
7.68.3.6 includeAllReadableWorkspaces	325
7.68.3.7 localData	325
7.68.3.8 notebookGuid	325
7.68.3.9 order	325
7.68.3.10 rawWords	325
7.68.3.11 searchContextBytes	325
7.68.3.12 tagGuids	325
7.68.3.13 timeZone	326
7.68.3.14 words	326
7.68.4 Property Documentation	326
7.68.4.1 tagGuids	326
7.69 qevercloud::NoteInvitationShareRelationship Struct Reference	326
7.69.1 Detailed Description	327
7.69.2 Member Function Documentation	327
7.69.2.1 operator!=( )	327
7.69.2.2 operator==( )	327
7.69.2.3 print()	327
7.69.3 Member Data Documentation	327
7.69.3.1 displayName	327
7.69.3.2 localData	327

7.69.3.3 privilege . . . . .	328
7.69.3.4 recipientIdentityId . . . . .	328
7.69.3.5 sharerUserId . . . . .	328
7.70 qevercloud::NoteLimits Struct Reference . . . . .	328
7.70.1 Detailed Description . . . . .	329
7.70.2 Member Function Documentation . . . . .	329
7.70.2.1 operator"!=( ) . . . . .	329
7.70.2.2 operator==( ) . . . . .	329
7.70.2.3 print() . . . . .	329
7.70.3 Member Data Documentation . . . . .	329
7.70.3.1 localData . . . . .	329
7.70.3.2 noteResourceCountMax . . . . .	330
7.70.3.3 noteSizeMax . . . . .	330
7.70.3.4 resourceSizeMax . . . . .	330
7.70.3.5 uploaded . . . . .	330
7.70.3.6 uploadLimit . . . . .	330
7.71 qevercloud::NoteList Struct Reference . . . . .	330
7.71.1 Detailed Description . . . . .	331
7.71.2 Member Function Documentation . . . . .	331
7.71.2.1 operator"!=( ) . . . . .	331
7.71.2.2 operator==( ) . . . . .	331
7.71.2.3 print() . . . . .	331
7.71.3 Member Data Documentation . . . . .	331
7.71.3.1 debugInfo . . . . .	332
7.71.3.2 localData . . . . .	332
7.71.3.3 notes . . . . .	332
7.71.3.4 searchContextBytes . . . . .	332
7.71.3.5 searchedWords . . . . .	332
7.71.3.6 startIndex . . . . .	332
7.71.3.7 stoppedWords . . . . .	332
7.71.3.8 totalNotes . . . . .	333
7.71.3.9 updateCount . . . . .	333
7.72 qevercloud::NoteMemberShareRelationship Struct Reference . . . . .	333
7.72.1 Detailed Description . . . . .	333
7.72.2 Member Function Documentation . . . . .	334
7.72.2.1 operator"!=( ) . . . . .	334
7.72.2.2 operator==( ) . . . . .	334
7.72.2.3 print() . . . . .	334
7.72.3 Member Data Documentation . . . . .	334
7.72.3.1 displayName . . . . .	334
7.72.3.2 localData . . . . .	334
7.72.3.3 privilege . . . . .	334

7.72.3.4 recipientUserId . . . . .	335
7.72.3.5 restrictions . . . . .	335
7.72.3.6 sharerUserId . . . . .	335
7.73 qevercloud::NoteMetadata Struct Reference . . . . .	335
7.73.1 Detailed Description . . . . .	336
7.73.2 Member Function Documentation . . . . .	336
7.73.2.1 operator"!=( ) . . . . .	336
7.73.2.2 operator==( ) . . . . .	336
7.73.2.3 print() . . . . .	336
7.73.3 Member Data Documentation . . . . .	336
7.73.3.1 attributes . . . . .	336
7.73.3.2 contentLength . . . . .	337
7.73.3.3 created . . . . .	337
7.73.3.4 deleted . . . . .	337
7.73.3.5 guid . . . . .	337
7.73.3.6 largestResourceMime . . . . .	337
7.73.3.7 largestResourceSize . . . . .	337
7.73.3.8 localData . . . . .	337
7.73.3.9 notebookGuid . . . . .	338
7.73.3.10 tagGuids . . . . .	338
7.73.3.11 title . . . . .	338
7.73.3.12 updated . . . . .	338
7.73.3.13 updateSequenceNum . . . . .	338
7.73.4 Property Documentation . . . . .	338
7.73.4.1 tagGuids . . . . .	338
7.74 qevercloud::NoteRestrictions Struct Reference . . . . .	338
7.74.1 Detailed Description . . . . .	339
7.74.2 Member Function Documentation . . . . .	339
7.74.2.1 operator"!=( ) . . . . .	340
7.74.2.2 operator==( ) . . . . .	340
7.74.2.3 print() . . . . .	340
7.74.3 Member Data Documentation . . . . .	340
7.74.3.1 localData . . . . .	340
7.74.3.2 noEmail . . . . .	340
7.74.3.3 noShare . . . . .	340
7.74.3.4 noSharePublicly . . . . .	341
7.74.3.5 noUpdateContent . . . . .	341
7.74.3.6 noUpdateTitle . . . . .	341
7.75 qevercloud::NoteResultSpec Struct Reference . . . . .	341
7.75.1 Detailed Description . . . . .	342
7.75.2 Member Function Documentation . . . . .	342
7.75.2.1 operator"!=( ) . . . . .	342

7.75.2.2 operator==( )	342
7.75.2.3 print()	342
7.75.3 Member Data Documentation	342
7.75.3.1 includeAccountLimits	342
7.75.3.2 includeContent	343
7.75.3.3 includeNoteAppDataValues	343
7.75.3.4 includeResourceAppDataValues	343
7.75.3.5 includeResourcesAlternateData	343
7.75.3.6 includeResourcesData	343
7.75.3.7 includeResourcesRecognition	343
7.75.3.8 includeSharedNotes	343
7.75.3.9 localData	344
7.76 qevercloud::NoteShareRelationshipRestrictions Struct Reference	344
7.76.1 Detailed Description	344
7.76.2 Member Function Documentation	344
7.76.2.1 operator!=( )	344
7.76.2.2 operator==( )	345
7.76.2.3 print()	345
7.76.3 Member Data Documentation	345
7.76.3.1 localData	345
7.76.3.2 noSetFullAccess	345
7.76.3.3 noSetModifyNote	345
7.76.3.4 noSetReadNote	345
7.77 qevercloud::NoteShareRelationships Struct Reference	346
7.77.1 Detailed Description	346
7.77.2 Member Function Documentation	346
7.77.2.1 operator!=( )	346
7.77.2.2 operator==( )	347
7.77.2.3 print()	347
7.77.3 Member Data Documentation	347
7.77.3.1 invitationRestrictions	347
7.77.3.2 invitations	347
7.77.3.3 localData	347
7.77.3.4 memberships	347
7.77.4 Property Documentation	348
7.77.4.1 invitations	348
7.77.4.2 memberships	348
7.78 qevercloud::NotesMetadataList Struct Reference	348
7.78.1 Detailed Description	349
7.78.2 Member Function Documentation	349
7.78.2.1 operator!=( )	349
7.78.2.2 operator==( )	349



7.78.2.3 print()	349
7.78.3 Member Data Documentation	349
7.78.3.1 debugInfo	349
7.78.3.2 localData	349
7.78.3.3 notes	350
7.78.3.4 searchContextBytes	350
7.78.3.5 searchedWords	350
7.78.3.6 startIndex	350
7.78.3.7 stoppedWords	350
7.78.3.8 totalNotes	350
7.78.3.9 updateCount	350
7.79 qevercloud::NotesMetadataResultSpec Struct Reference	351
7.79.1 Detailed Description	351
7.79.2 Member Function Documentation	351
7.79.2.1 operator"!=( )	352
7.79.2.2 operator==( )	352
7.79.2.3 print()	352
7.79.3 Member Data Documentation	352
7.79.3.1 includeAttributes	352
7.79.3.2 includeContentLength	352
7.79.3.3 includeCreated	352
7.79.3.4 includeDeleted	353
7.79.3.5 includeLargestResourceMime	353
7.79.3.6 includeLargestResourceSize	353
7.79.3.7 includeNotebookGuid	353
7.79.3.8 includeTagGuids	353
7.79.3.9 includeTitle	353
7.79.3.10 includeUpdated	353
7.79.3.11 includeUpdateSequenceNum	353
7.79.3.12 localData	354
7.80 qevercloud::NoteStoreServer Class Reference	354
7.80.1 Detailed Description	359
7.80.2 Constructor & Destructor Documentation	359
7.80.2.1 NoteStoreServer()	359
7.80.3 Member Function Documentation	359
7.80.3.1 authenticateToSharedNotebookRequest	360
7.80.3.2 authenticateToSharedNotebookRequestReady	360
7.80.3.3 authenticateToSharedNoteRequest	360
7.80.3.4 authenticateToSharedNoteRequestReady	360
7.80.3.5 copyNoteRequest	360
7.80.3.6 copyNoteRequestReady	360
7.80.3.7 createLinkedNotebookRequest	361

7.80.3.8 createLinkedNotebookRequestReady . . . . .	361
7.80.3.9 createNotebookRequest . . . . .	361
7.80.3.10 createNotebookRequestReady . . . . .	361
7.80.3.11 createNoteRequest . . . . .	361
7.80.3.12 createNoteRequestReady . . . . .	361
7.80.3.13 createOrUpdateNotebookSharesRequest . . . . .	362
7.80.3.14 createOrUpdateNotebookSharesRequestReady . . . . .	362
7.80.3.15 createSearchRequest . . . . .	362
7.80.3.16 createSearchRequestReady . . . . .	362
7.80.3.17 createTagRequest . . . . .	362
7.80.3.18 createTagRequestReady . . . . .	362
7.80.3.19 deleteNoteRequest . . . . .	363
7.80.3.20 deleteNoteRequestReady . . . . .	363
7.80.3.21 emailNoteRequest . . . . .	363
7.80.3.22 emailNoteRequestReady . . . . .	363
7.80.3.23 expungeLinkedNotebookRequest . . . . .	363
7.80.3.24 expungeLinkedNotebookRequestReady . . . . .	363
7.80.3.25 expungeNotebookRequest . . . . .	364
7.80.3.26 expungeNotebookRequestReady . . . . .	364
7.80.3.27 expungeNoteRequest . . . . .	364
7.80.3.28 expungeNoteRequestReady . . . . .	364
7.80.3.29 expungeSearchRequest . . . . .	364
7.80.3.30 expungeSearchRequestReady . . . . .	364
7.80.3.31 expungeTagRequest . . . . .	365
7.80.3.32 expungeTagRequestReady . . . . .	365
7.80.3.33 findNoteCountsRequest . . . . .	365
7.80.3.34 findNoteCountsRequestReady . . . . .	365
7.80.3.35 findNoteOffsetRequest . . . . .	365
7.80.3.36 findNoteOffsetRequestReady . . . . .	365
7.80.3.37 findNotesMetadataRequest . . . . .	366
7.80.3.38 findNotesMetadataRequestReady . . . . .	366
7.80.3.39 findRelatedRequest . . . . .	366
7.80.3.40 findRelatedRequestReady . . . . .	366
7.80.3.41 getDefaultNotebookRequest . . . . .	366
7.80.3.42 getDefaultNotebookRequestReady . . . . .	366
7.80.3.43 getFilteredSyncChunkRequest . . . . .	367
7.80.3.44 getFilteredSyncChunkRequestReady . . . . .	367
7.80.3.45 getLinkedNotebookSyncChunkRequest . . . . .	367
7.80.3.46 getLinkedNotebookSyncChunkRequestReady . . . . .	367
7.80.3.47 getLinkedNotebookSyncStateRequest . . . . .	367
7.80.3.48 getLinkedNotebookSyncStateRequestReady . . . . .	367
7.80.3.49 getNoteApplicationDataEntryRequest . . . . .	368

7.80.3.50	getNoteApplicationDataEntryRequestReady	368
7.80.3.51	getNoteApplicationDataRequest	368
7.80.3.52	getNoteApplicationDataRequestReady	368
7.80.3.53	getNotebookRequest	368
7.80.3.54	getNotebookRequestReady	368
7.80.3.55	getNotebookSharesRequest	369
7.80.3.56	getNotebookSharesRequestReady	369
7.80.3.57	getNoteContentRequest	369
7.80.3.58	getNoteContentRequestReady	369
7.80.3.59	getNoteRequest	369
7.80.3.60	getNoteRequestReady	369
7.80.3.61	getNoteSearchTextRequest	370
7.80.3.62	getNoteSearchTextRequestReady	370
7.80.3.63	getNoteTagNamesRequest	370
7.80.3.64	getNoteTagNamesRequestReady	370
7.80.3.65	getNoteVersionRequest	370
7.80.3.66	getNoteVersionRequestReady	370
7.80.3.67	getNoteWithResultSpecRequest	371
7.80.3.68	getNoteWithResultSpecRequestReady	371
7.80.3.69	getPublicNotebookRequest	371
7.80.3.70	getPublicNotebookRequestReady	371
7.80.3.71	getResourceAlternateDataRequest	371
7.80.3.72	getResourceAlternateDataRequestReady	371
7.80.3.73	getResourceApplicationDataEntryRequest	372
7.80.3.74	getResourceApplicationDataEntryRequestReady	372
7.80.3.75	getResourceApplicationDataRequest	372
7.80.3.76	getResourceApplicationDataRequestReady	372
7.80.3.77	getResourceAttributesRequest	372
7.80.3.78	getResourceAttributesRequestReady	372
7.80.3.79	getResourceByHashRequest	373
7.80.3.80	getResourceByHashRequestReady	373
7.80.3.81	getResourceDataRequest	373
7.80.3.82	getResourceDataRequestReady	373
7.80.3.83	getResourceRecognitionRequest	373
7.80.3.84	getResourceRecognitionRequestReady	373
7.80.3.85	getResourceRequest	374
7.80.3.86	getResourceRequestReady	374
7.80.3.87	getResourceSearchTextRequest	374
7.80.3.88	getResourceSearchTextRequestReady	374
7.80.3.89	getSearchRequest	374
7.80.3.90	getSearchRequestReady	374
7.80.3.91	getSharedNotebookByAuthRequest	375

7.80.3.92 getSharedNotebookByAuthRequestReady	375
7.80.3.93 getSyncStateRequest	375
7.80.3.94 getSyncStateRequestReady	375
7.80.3.95 getTagRequest	375
7.80.3.96 getTagRequestReady	375
7.80.3.97 listAccessibleBusinessNotebooksRequest	375
7.80.3.98 listAccessibleBusinessNotebooksRequestReady	376
7.80.3.99 listLinkedNotebooksRequest	376
7.80.3.100 listLinkedNotebooksRequestReady	376
7.80.3.101 listNotebooksRequest	376
7.80.3.102 listNotebooksRequestReady	376
7.80.3.103 listNoteVersionsRequest	376
7.80.3.104 listNoteVersionsRequestReady	376
7.80.3.105 listSearchesRequest	377
7.80.3.106 listSearchesRequestReady	377
7.80.3.107 listSharedNotebooksRequest	377
7.80.3.108 listSharedNotebooksRequestReady	377
7.80.3.109 listTagsByNotebookRequest	377
7.80.3.110 listTagsByNotebookRequestReady	377
7.80.3.111 listTagsRequest	377
7.80.3.112 listTagsRequestReady	378
7.80.3.113 manageNotebookSharesRequest	378
7.80.3.114 manageNotebookSharesRequestReady	378
7.80.3.115 onAuthenticateToSharedNotebookRequestReady	378
7.80.3.116 onAuthenticateToSharedNoteRequestReady	378
7.80.3.117 onCopyNoteRequestReady	378
7.80.3.118 onCreateLinkedNotebookRequestReady	379
7.80.3.119 onCreateNotebookRequestReady	379
7.80.3.120 onCreateNoteRequestReady	379
7.80.3.121 onCreateOrUpdateNotebookSharesRequestReady	379
7.80.3.122 onCreateSearchRequestReady	379
7.80.3.123 onCreateTagRequestReady	379
7.80.3.124 onDeleteNoteRequestReady	380
7.80.3.125 onEmailNoteRequestReady	380
7.80.3.126 onExpungeLinkedNotebookRequestReady	380
7.80.3.127 onExpungeNotebookRequestReady	380
7.80.3.128 onExpungeNoteRequestReady	380
7.80.3.129 onExpungeSearchRequestReady	380
7.80.3.130 onExpungeTagRequestReady	381
7.80.3.131 onFindNoteCountsRequestReady	381
7.80.3.132 onFindNoteOffsetRequestReady	381
7.80.3.133 onFindNotesMetadataRequestReady	381

7.80.3.134 onFindRelatedRequestReady . . . . .	381
7.80.3.135 onGetDefaultNotebookRequestReady . . . . .	381
7.80.3.136 onGetFilteredSyncChunkRequestReady . . . . .	382
7.80.3.137 onGetLinkedNotebookSyncChunkRequestReady . . . . .	382
7.80.3.138 onGetLinkedNotebookSyncStateRequestReady . . . . .	382
7.80.3.139 onGetNoteApplicationDataEntryRequestReady . . . . .	382
7.80.3.140 onGetNoteApplicationDataRequestReady . . . . .	382
7.80.3.141 onGetNotebookRequestReady . . . . .	382
7.80.3.142 onGetNotebookSharesRequestReady . . . . .	383
7.80.3.143 onGetNoteContentRequestReady . . . . .	383
7.80.3.144 onGetNoteRequestReady . . . . .	383
7.80.3.145 onGetNoteSearchTextRequestReady . . . . .	383
7.80.3.146 onGetNoteTagNamesRequestReady . . . . .	383
7.80.3.147 onGetNoteVersionRequestReady . . . . .	383
7.80.3.148 onGetNoteWithResultSpecRequestReady . . . . .	384
7.80.3.149 onGetPublicNotebookRequestReady . . . . .	384
7.80.3.150 onGetResourceAlternateDataRequestReady . . . . .	384
7.80.3.151 onGetResourceApplicationDataEntryRequestReady . . . . .	384
7.80.3.152 onGetResourceApplicationDataRequestReady . . . . .	384
7.80.3.153 onGetResourceAttributesRequestReady . . . . .	384
7.80.3.154 onGetResourceByHashRequestReady . . . . .	385
7.80.3.155 onGetResourceDataRequestReady . . . . .	385
7.80.3.156 onGetResourceRecognitionRequestReady . . . . .	385
7.80.3.157 onGetResourceRequestReady . . . . .	385
7.80.3.158 onGetResourceSearchTextRequestReady . . . . .	385
7.80.3.159 onGetSearchRequestReady . . . . .	385
7.80.3.160 onGetSharedNotebookByAuthRequestReady . . . . .	386
7.80.3.161 onGetSyncStateRequestReady . . . . .	386
7.80.3.162 onGetTagRequestReady . . . . .	386
7.80.3.163 onListAccessibleBusinessNotebooksRequestReady . . . . .	386
7.80.3.164 onListLinkedNotebooksRequestReady . . . . .	386
7.80.3.165 onListNotebooksRequestReady . . . . .	386
7.80.3.166 onListNoteVersionsRequestReady . . . . .	387
7.80.3.167 onListSearchesRequestReady . . . . .	387
7.80.3.168 onListSharedNotebooksRequestReady . . . . .	387
7.80.3.169 onListTagsByNotebookRequestReady . . . . .	387
7.80.3.170 onListTagsRequestReady . . . . .	387
7.80.3.171 onManageNotebookSharesRequestReady . . . . .	387
7.80.3.172 onRequest . . . . .	388
7.80.3.173 onSetNoteApplicationDataEntryRequestReady . . . . .	388
7.80.3.174 onSetNotebookRecipientSettingsRequestReady . . . . .	388
7.80.3.175 onSetResourceApplicationDataEntryRequestReady . . . . .	388

7.80.3.176 onShareNotebookRequestReady . . . . .	388
7.80.3.177 onShareNoteRequestReady . . . . .	388
7.80.3.178 onStopSharingNoteRequestReady . . . . .	389
7.80.3.179 onUnsetNoteApplicationDataEntryRequestReady . . . . .	389
7.80.3.180 onUnsetResourceApplicationDataEntryRequestReady . . . . .	389
7.80.3.181 onUntagAllRequestReady . . . . .	389
7.80.3.182 onUpdateLinkedNotebookRequestReady . . . . .	389
7.80.3.183 onUpdateNotebookRequestReady . . . . .	389
7.80.3.184 onUpdateNoteIfUsnMatchesRequestReady . . . . .	390
7.80.3.185 onUpdateNoteRequestReady . . . . .	390
7.80.3.186 onUpdateResourceRequestReady . . . . .	390
7.80.3.187 onUpdateSearchRequestReady . . . . .	390
7.80.3.188 onUpdateSharedNotebookRequestReady . . . . .	390
7.80.3.189 onUpdateTagRequestReady . . . . .	390
7.80.3.190 setNoteApplicationDataEntryRequest . . . . .	391
7.80.3.191 setNoteApplicationDataEntryRequestReady . . . . .	391
7.80.3.192 setNotebookRecipientSettingsRequest . . . . .	391
7.80.3.193 setNotebookRecipientSettingsRequestReady . . . . .	391
7.80.3.194 setResourceApplicationDataEntryRequest . . . . .	391
7.80.3.195 setResourceApplicationDataEntryRequestReady . . . . .	391
7.80.3.196 shareNotebookRequest . . . . .	392
7.80.3.197 shareNotebookRequestReady . . . . .	392
7.80.3.198 shareNoteRequest . . . . .	392
7.80.3.199 shareNoteRequestReady . . . . .	392
7.80.3.200 stopSharingNoteRequest . . . . .	392
7.80.3.201 stopSharingNoteRequestReady . . . . .	392
7.80.3.202 unsetNoteApplicationDataEntryRequest . . . . .	393
7.80.3.203 unsetNoteApplicationDataEntryRequestReady . . . . .	393
7.80.3.204 unsetResourceApplicationDataEntryRequest . . . . .	393
7.80.3.205 unsetResourceApplicationDataEntryRequestReady . . . . .	393
7.80.3.206 untagAllRequest . . . . .	393
7.80.3.207 untagAllRequestReady . . . . .	393
7.80.3.208 updateLinkedNotebookRequest . . . . .	394
7.80.3.209 updateLinkedNotebookRequestReady . . . . .	394
7.80.3.210 updateNotebookRequest . . . . .	394
7.80.3.211 updateNotebookRequestReady . . . . .	394
7.80.3.212 updateNoteIfUsnMatchesRequest . . . . .	394
7.80.3.213 updateNoteIfUsnMatchesRequestReady . . . . .	394
7.80.3.214 updateNoteRequest . . . . .	395
7.80.3.215 updateNoteRequestReady . . . . .	395
7.80.3.216 updateResourceRequest . . . . .	395
7.80.3.217 updateResourceRequestReady . . . . .	395

7.80.3.218 updateSearchRequest . . . . .	395
7.80.3.219 updateSearchRequestReady . . . . .	395
7.80.3.220 updateSharedNotebookRequest . . . . .	396
7.80.3.221 updateSharedNotebookRequestReady . . . . .	396
7.80.3.222 updateTagRequest . . . . .	396
7.80.3.223 updateTagRequestReady . . . . .	396
7.81 qevercloud::NoteVersionId Struct Reference . . . . .	396
7.81.1 Detailed Description . . . . .	397
7.81.2 Member Function Documentation . . . . .	397
7.81.2.1 operator"!=()	397
7.81.2.2 operator==( ) . . . . .	397
7.81.2.3 print() . . . . .	397
7.81.3 Member Data Documentation . . . . .	397
7.81.3.1 lastEditorId . . . . .	398
7.81.3.2 localData . . . . .	398
7.81.3.3 saved . . . . .	398
7.81.3.4 title . . . . .	398
7.81.3.5 updated . . . . .	398
7.81.3.6 updateSequenceNum . . . . .	398
7.82 qevercloud::EvernoteOAuthWebView::OAuthResult Struct Reference . . . . .	399
7.82.1 Detailed Description . . . . .	399
7.82.2 Member Function Documentation . . . . .	399
7.82.2.1 print() . . . . .	399
7.82.3 Member Data Documentation . . . . .	400
7.82.3.1 authenticationToken . . . . .	400
7.82.3.2 cookies . . . . .	400
7.82.3.3 expires . . . . .	400
7.82.3.4 noteStoreUrl . . . . .	400
7.82.3.5 shardId . . . . .	400
7.82.3.6 userId . . . . .	401
7.82.3.7 webApiUrlPrefix . . . . .	401
7.83 qevercloud::Optional< T > Class Template Reference . . . . .	401
7.83.1 Detailed Description . . . . .	402
7.83.2 Constructor & Destructor Documentation . . . . .	402
7.83.2.1 Optional() [1/7] . . . . .	402
7.83.2.2 Optional() [2/7] . . . . .	402
7.83.2.3 Optional() [3/7] . . . . .	403
7.83.2.4 Optional() [4/7] . . . . .	403
7.83.2.5 Optional() [5/7] . . . . .	403
7.83.2.6 Optional() [6/7] . . . . .	403
7.83.2.7 Optional() [7/7] . . . . .	403
7.83.3 Member Function Documentation . . . . .	403

7.83.3.1 clear()	404
7.83.3.2 init()	404
7.83.3.3 isEqual()	404
7.83.3.4 isSet()	404
7.83.3.5 operator const T &()	405
7.83.3.6 operator T&()	405
7.83.3.7 operator"!=( ) [1/2]	405
7.83.3.8 operator"!=( ) [2/2]	405
7.83.3.9 operator->() [1/2]	405
7.83.3.10 operator->() [2/2]	406
7.83.3.11 operator=( ) [1/6]	406
7.83.3.12 operator=( ) [2/6]	406
7.83.3.13 operator=( ) [3/6]	406
7.83.3.14 operator=( ) [4/6]	406
7.83.3.15 operator=( ) [5/6]	406
7.83.3.16 operator=( ) [6/6]	407
7.83.3.17 operator==( ) [1/2]	407
7.83.3.18 operator==( ) [2/2]	407
7.83.3.19 ref() [1/2]	407
7.83.3.20 ref() [2/2]	407
7.83.3.21 value()	407
7.83.4 Friends And Related Function Documentation	408
7.83.4.1 Optional	408
7.83.4.2 swap	408
7.84 qevercloud::Printable Class Reference	408
7.84.1 Constructor & Destructor Documentation	409
7.84.1.1 Printable()	409
7.84.1.2 ~Printable()	410
7.84.2 Member Function Documentation	410
7.84.2.1 print()	410
7.84.2.2 toString()	410
7.84.3 Friends And Related Function Documentation	410
7.84.3.1 operator<< [1/2]	411
7.84.3.2 operator<< [2/2]	411
7.85 qevercloud::PublicUserInfo Struct Reference	411
7.85.1 Detailed Description	411
7.85.2 Member Function Documentation	412
7.85.2.1 operator"!=( )	412
7.85.2.2 operator==( )	412
7.85.2.3 print()	412
7.85.3 Member Data Documentation	412
7.85.3.1 localData	412



7.85.3.2 noteStoreUrl . . . . .	412
7.85.3.3 serviceLevel . . . . .	413
7.85.3.4 userId . . . . .	413
7.85.3.5 username . . . . .	413
7.85.3.6 webApiUrlPrefix . . . . .	413
7.86 qevercloud::Publishing Struct Reference . . . . .	413
7.86.1 Detailed Description . . . . .	414
7.86.2 Member Function Documentation . . . . .	414
7.86.2.1 operator!=(()) . . . . .	414
7.86.2.2 operator==(()) . . . . .	414
7.86.2.3 print() . . . . .	414
7.86.3 Member Data Documentation . . . . .	414
7.86.3.1 ascending . . . . .	414
7.86.3.2 localData . . . . .	415
7.86.3.3 order . . . . .	415
7.86.3.4 publicDescription . . . . .	415
7.86.3.5 uri . . . . .	415
7.87 qevercloud::QAssociativeContainerConstReferenceWrapper< Container > Class Template Reference . . . . .	415
7.87.1 Constructor & Destructor Documentation . . . . .	416
7.87.1.1 QAssociativeContainerConstReferenceWrapper() . . . . .	416
7.87.2 Member Function Documentation . . . . .	416
7.87.2.1 begin() . . . . .	416
7.87.2.2 end() . . . . .	416
7.88 qevercloud::QAssociativeContainerReferenceWrapper< Container > Class Template Reference . . . . .	416
7.88.1 Constructor & Destructor Documentation . . . . .	417
7.88.1.1 QAssociativeContainerReferenceWrapper() . . . . .	417
7.88.2 Member Function Documentation . . . . .	417
7.88.2.1 begin() . . . . .	417
7.88.2.2 end() . . . . .	417
7.89 qevercloud::RelatedContent Struct Reference . . . . .	417
7.89.1 Detailed Description . . . . .	418
7.89.2 Member Function Documentation . . . . .	418
7.89.2.1 operator!=(()) . . . . .	418
7.89.2.2 operator==(()) . . . . .	418
7.89.2.3 print() . . . . .	419
7.89.3 Member Data Documentation . . . . .	419
7.89.3.1 accessType . . . . .	419
7.89.3.2 authors . . . . .	419
7.89.3.3 clipUrl . . . . .	419
7.89.3.4 contact . . . . .	419
7.89.3.5 contentId . . . . .	419
7.89.3.6 contentType . . . . .	419

7.89.3.7 date	420
7.89.3.8 localData	420
7.89.3.9 sourceFaviconUrl	420
7.89.3.10 sourceId	420
7.89.3.11 sourceName	420
7.89.3.12 sourceUrl	420
7.89.3.13 teaser	420
7.89.3.14 thumbnails	420
7.89.3.15 title	421
7.89.3.16 url	421
7.89.3.17 visibleUrl	421
7.89.4 Property Documentation	421
7.89.4.1 thumbnails	421
7.90 qevercloud::RelatedContentImage Struct Reference	421
7.90.1 Detailed Description	422
7.90.2 Member Function Documentation	422
7.90.2.1 operator!=(())	422
7.90.2.2 operator==(())	422
7.90.2.3 print()	422
7.90.3 Member Data Documentation	422
7.90.3.1 fileSize	423
7.90.3.2 height	423
7.90.3.3 localData	423
7.90.3.4 pixelRatio	423
7.90.3.5 url	423
7.90.3.6 width	423
7.91 qevercloud::RelatedQuery Struct Reference	423
7.91.1 Detailed Description	424
7.91.2 Member Function Documentation	424
7.91.2.1 operator!=(())	424
7.91.2.2 operator==(())	424
7.91.2.3 print()	424
7.91.3 Member Data Documentation	425
7.91.3.1 cacheKey	425
7.91.3.2 context	425
7.91.3.3 filter	425
7.91.3.4 localData	425
7.91.3.5 noteGuid	425
7.91.3.6 plainText	425
7.91.3.7 referenceUri	426
7.92 qevercloud::RelatedResult Struct Reference	426
7.92.1 Detailed Description	427

7.92.2 Member Function Documentation	427
7.92.2.1 operator"!=( )	427
7.92.2.2 operator==( )	427
7.92.2.3 print()	427
7.92.3 Member Data Documentation	427
7.92.3.1 cacheExpires	427
7.92.3.2 cacheKey	428
7.92.3.3 containingNotebooks	428
7.92.3.4 debugInfo	428
7.92.3.5 experts	428
7.92.3.6 localData	429
7.92.3.7 notebooks	429
7.92.3.8 notes	429
7.92.3.9 relatedContent	429
7.92.3.10 tags	429
7.92.4 Property Documentation	429
7.92.4.1 containingNotebooks	429
7.92.4.2 experts	429
7.92.4.3 notebooks	430
7.92.4.4 notes	430
7.92.4.5 relatedContent	430
7.92.4.6 tags	430
7.93 qevercloud::RelatedResultSpec Struct Reference	430
7.93.1 Detailed Description	431
7.93.2 Member Function Documentation	431
7.93.2.1 operator"!=( )	431
7.93.2.2 operator==( )	431
7.93.2.3 print()	431
7.93.3 Member Data Documentation	432
7.93.3.1 includeContainingNotebooks	432
7.93.3.2 includeDebugInfo	432
7.93.3.3 localData	432
7.93.3.4 maxExperts	432
7.93.3.5 maxNotebooks	432
7.93.3.6 maxNotes	432
7.93.3.7 maxRelatedContent	433
7.93.3.8 maxTags	433
7.93.3.9 relatedContentTypes	433
7.93.3.10 writableNotebooksOnly	433
7.93.4 Property Documentation	433
7.93.4.1 relatedContentTypes	433
7.94 qevercloud::Resource Struct Reference	433

7.94.1 Detailed Description	434
7.94.2 Member Function Documentation	434
7.94.2.1 operator"!=()	434
7.94.2.2 operator==(())	434
7.94.2.3 print()	435
7.94.3 Member Data Documentation	435
7.94.3.1 active	435
7.94.3.2 alternateData	435
7.94.3.3 attributes	435
7.94.3.4 data	435
7.94.3.5 duration	435
7.94.3.6 guid	436
7.94.3.7 height	436
7.94.3.8 localData	436
7.94.3.9 mime	436
7.94.3.10 noteGuid	436
7.94.3.11 recognition	436
7.94.3.12 updateSequenceNum	436
7.94.3.13 width	437
7.95 qevercloud::ResourceAttributes Struct Reference	437
7.95.1 Detailed Description	437
7.95.2 Member Function Documentation	437
7.95.2.1 operator"!=()	438
7.95.2.2 operator==(())	438
7.95.2.3 print()	438
7.95.3 Member Data Documentation	438
7.95.3.1 altitude	438
7.95.3.2 applicationData	438
7.95.3.3 attachment	439
7.95.3.4 cameraMake	439
7.95.3.5 cameraModel	439
7.95.3.6 clientWillIndex	439
7.95.3.7 fileName	439
7.95.3.8 latitude	439
7.95.3.9 localData	439
7.95.3.10 longitude	440
7.95.3.11 recoType	440
7.95.3.12 sourceURL	440
7.95.3.13 timestamp	440
7.96 qevercloud::SavedSearch Struct Reference	440
7.96.1 Detailed Description	441
7.96.2 Member Function Documentation	441

7.96.2.1 operator"!=()	441
7.96.2.2 operator==(())	441
7.96.2.3 print()	441
7.96.3 Member Data Documentation	441
7.96.3.1 format	442
7.96.3.2 guid	442
7.96.3.3 localData	442
7.96.3.4 name	442
7.96.3.5 query	442
7.96.3.6 scope	442
7.96.3.7 updateSequenceNum	443
7.97 qevercloud::SavedSearchScope Struct Reference	443
7.97.1 Detailed Description	443
7.97.2 Member Function Documentation	443
7.97.2.1 operator"!=()	443
7.97.2.2 operator==(())	444
7.97.2.3 print()	444
7.97.3 Member Data Documentation	444
7.97.3.1 includeAccount	444
7.97.3.2 includeBusinessLinkedNotebooks	444
7.97.3.3 includePersonalLinkedNotebooks	444
7.97.3.4 localData	444
7.98 qevercloud::SharedNote Struct Reference	445
7.98.1 Detailed Description	445
7.98.2 Member Function Documentation	445
7.98.2.1 operator"!=()	445
7.98.2.2 operator==(())	446
7.98.2.3 print()	446
7.98.3 Member Data Documentation	446
7.98.3.1 localData	446
7.98.3.2 privilege	446
7.98.3.3 recipientIdentity	446
7.98.3.4 serviceAssigned	446
7.98.3.5 serviceCreated	447
7.98.3.6 serviceUpdated	447
7.98.3.7 sharerUserID	447
7.99 qevercloud::SharedNotebook Struct Reference	447
7.99.1 Detailed Description	448
7.99.2 Member Function Documentation	448
7.99.2.1 operator"!=()	448
7.99.2.2 operator==(())	448
7.99.2.3 print()	448

7.99.3 Member Data Documentation	448
7.99.3.1 email	448
7.99.3.2 globalId	448
7.99.3.3 id	449
7.99.3.4 localData	449
7.99.3.5 notebookGuid	449
7.99.3.6 notebookModifiable	449
7.99.3.7 privilege	449
7.99.3.8 recipientIdentityId	449
7.99.3.9 recipientSettings	449
7.99.3.10 recipientUserId	450
7.99.3.11 recipientUsername	450
7.99.3.12 serviceAssigned	450
7.99.3.13 serviceCreated	450
7.99.3.14 serviceUpdated	450
7.99.3.15 sharerUserId	450
7.99.3.16 userId	451
7.99.3.17 username	451
7.100 qevercloud::SharedNotebookRecipientSettings Struct Reference	451
7.100.1 Detailed Description	451
7.100.2 Member Function Documentation	452
7.100.2.1 operator"!=( )	452
7.100.2.2 operator==( )	452
7.100.2.3 print( )	452
7.100.3 Member Data Documentation	452
7.100.3.1 localData	452
7.100.3.2 reminderNotifyEmail	452
7.100.3.3 reminderNotifyInApp	453
7.101 qevercloud::SharedNoteTemplate Struct Reference	453
7.101.1 Detailed Description	453
7.101.2 Member Function Documentation	453
7.101.2.1 operator"!=( )	454
7.101.2.2 operator==( )	454
7.101.2.3 print( )	454
7.101.3 Member Data Documentation	454
7.101.3.1 localData	454
7.101.3.2 noteGuid	454
7.101.3.3 privilege	454
7.101.3.4 recipientContacts	455
7.101.3.5 recipientThreadId	455
7.101.4 Property Documentation	455
7.101.4.1 recipientContacts	455

7.102 qevercloud::ShareRelationshipRestrictions Struct Reference	455
7.102.1 Detailed Description	456
7.102.2 Member Function Documentation	456
7.102.2.1 operator"!=()	456
7.102.2.2 operator==(())	456
7.102.2.3 print()	456
7.102.3 Member Data Documentation	456
7.102.3.1 localData	456
7.102.3.2 noSetFullAccess	456
7.102.3.3 noSetModify	457
7.102.3.4 noSetReadOnly	457
7.102.3.5 noSetReadPlusActivity	457
7.103 qevercloud::ShareRelationships Struct Reference	457
7.103.1 Detailed Description	458
7.103.2 Member Function Documentation	458
7.103.2.1 operator"!=()	458
7.103.2.2 operator==(())	458
7.103.2.3 print()	458
7.103.3 Member Data Documentation	458
7.103.3.1 invitationRestrictions	458
7.103.3.2 invitations	459
7.103.3.3 localData	459
7.103.3.4 memberships	459
7.103.4 Property Documentation	459
7.103.4.1 invitations	459
7.103.4.2 memberships	459
7.104 qevercloud::SyncChunk Struct Reference	459
7.104.1 Detailed Description	460
7.104.2 Member Function Documentation	460
7.104.2.1 operator"!=()	461
7.104.2.2 operator==(())	461
7.104.2.3 print()	461
7.104.3 Member Data Documentation	461
7.104.3.1 chunkHighUSN	461
7.104.3.2 currentTime	461
7.104.3.3 expungedLinkedNotebooks	461
7.104.3.4 expungedNotebooks	462
7.104.3.5 expungedNotes	462
7.104.3.6 expungedSearches	462
7.104.3.7 expungedTags	462
7.104.3.8 linkedNotebooks	462
7.104.3.9 localData	462

7.104.3.10 notebooks	462
7.104.3.11 notes	463
7.104.3.12 resources	463
7.104.3.13 searches	463
7.104.3.14 tags	463
7.104.3.15 updateCount	463
7.104.4 Property Documentation	463
7.104.4.1 expungedLinkedNotebooks	463
7.104.4.2 expungedNotebooks	464
7.104.4.3 expungedNotes	464
7.104.4.4 expungedSearches	464
7.104.4.5 expungedTags	464
7.104.4.6 linkedNotebooks	464
7.104.4.7 notebooks	464
7.104.4.8 notes	464
7.104.4.9 resources	464
7.104.4.10 searches	465
7.104.4.11 tags	465
7.105 qevercloud::SyncChunkFilter Struct Reference	465
7.105.1 Detailed Description	466
7.105.2 Member Function Documentation	466
7.105.2.1 operator"!=( )	466
7.105.2.2 operator==( )	466
7.105.2.3 print()	466
7.105.3 Member Data Documentation	466
7.105.3.1 includeExpunged	466
7.105.3.2 includeLinkedNotebooks	467
7.105.3.3 includeNoteApplicationDataFullMap	467
7.105.3.4 includeNoteAttributes	467
7.105.3.5 includeNotebooks	467
7.105.3.6 includeNoteResourceApplicationDataFullMap	467
7.105.3.7 includeNoteResources	467
7.105.3.8 includeNotes	467
7.105.3.9 includeResourceApplicationDataFullMap	468
7.105.3.10 includeResources	468
7.105.3.11 includeSearches	468
7.105.3.12 includeSharedNotes	468
7.105.3.13 includeTags	468
7.105.3.14 localData	468
7.105.3.15 notebookGuids	468
7.105.3.16 omitSharedNotebooks	469
7.105.3.17 requireNoteContentClass	469



7.105.4 Property Documentation	469
7.105.4.1 notebookGuids	469
7.106 qevercloud::IDurableService::SyncRequest Struct Reference	469
7.106.1 Constructor & Destructor Documentation	469
7.106.1.1 SyncRequest()	469
7.106.2 Member Data Documentation	470
7.106.2.1 m_call	470
7.106.2.2 m_description	470
7.106.2.3 m_name	470
7.107 qevercloud::SyncState Struct Reference	470
7.107.1 Detailed Description	471
7.107.2 Member Function Documentation	471
7.107.2.1 operator"!=()	471
7.107.2.2 operator==(())	471
7.107.2.3 print()	471
7.107.3 Member Data Documentation	471
7.107.3.1 currentTime	471
7.107.3.2 fullSyncBefore	471
7.107.3.3 localData	472
7.107.3.4 updateCount	472
7.107.3.5 uploaded	472
7.107.3.6 userLastUpdated	472
7.107.3.7 userMaxMessageEventId	472
7.108 qevercloud::Tag Struct Reference	473
7.108.1 Detailed Description	473
7.108.2 Member Function Documentation	473
7.108.2.1 operator"!=()	473
7.108.2.2 operator==(())	473
7.108.2.3 print()	474
7.108.3 Member Data Documentation	474
7.108.3.1 guid	474
7.108.3.2 localData	474
7.108.3.3 name	474
7.108.3.4 parentGuid	474
7.108.3.5 updateSequenceNum	475
7.109 qevercloud::ThriftException Class Reference	475
7.109.1 Detailed Description	476
7.109.2 Member Enumeration Documentation	476
7.109.2.1 Type	476
7.109.3 Constructor & Destructor Documentation	476
7.109.3.1 ThriftException() [1/3]	476
7.109.3.2 ThriftException() [2/3]	476

7.109.3.3 ThriftException() [3/3]	476
7.109.3.4 ~ThriftException()	477
7.109.4 Member Function Documentation	477
7.109.4.1 exceptionData()	477
7.109.4.2 operator"!=()"	477
7.109.4.3 operator==(())	477
7.109.4.4 type()	477
7.109.4.5 what()	477
7.109.5 Friends And Related Function Documentation	477
7.109.5.1 operator<<	478
7.109.6 Member Data Documentation	478
7.109.6.1 m_type	478
7.110 qevercloud::ThriftExceptionData Class Reference	478
7.110.1 Detailed Description	478
7.110.2 Constructor & Destructor Documentation	479
7.110.2.1 ThriftExceptionData()	479
7.110.3 Member Function Documentation	479
7.110.3.1 throwException()	479
7.110.4 Member Data Documentation	479
7.110.4.1 m_type	479
7.111 qevercloud::Thumbnail Class Reference	479
7.111.1 Detailed Description	480
7.111.2 Member Enumeration Documentation	480
7.111.2.1 ImageType	480
7.111.3 Constructor & Destructor Documentation	481
7.111.3.1 Thumbnail() [1/2]	481
7.111.3.2 Thumbnail() [2/2]	481
7.111.3.3 ~Thumbnail()	481
7.111.4 Member Function Documentation	482
7.111.4.1 createPostRequest()	482
7.111.4.2 download()	482
7.111.4.3 downloadAsync()	483
7.111.4.4 setAuthenticationToken()	483
7.111.4.5 setHost()	483
7.111.4.6 setImageType()	483
7.111.4.7 setShardId()	483
7.111.4.8 setSize()	484
7.111.5 Friends And Related Function Documentation	484
7.111.5.1 operator<< [1/2]	484
7.111.5.2 operator<< [2/2]	484
7.112 qevercloud::UpdateNotelfUsnMatchesResult Struct Reference	484
7.112.1 Detailed Description	485

7.112.2 Member Function Documentation	485
7.112.2.1 operator"!=()"	485
7.112.2.2 operator=="()"	485
7.112.2.3 print()	485
7.112.3 Member Data Documentation	485
7.112.3.1 localData	486
7.112.3.2 note	486
7.112.3.3 updated	486
7.113 qevercloud::User Struct Reference	486
7.113.1 Detailed Description	487
7.113.2 Member Function Documentation	487
7.113.2.1 operator"!=()"	487
7.113.2.2 operator=="()"	487
7.113.2.3 print()	487
7.113.3 Member Data Documentation	488
7.113.3.1 accounting	488
7.113.3.2 accountLimits	488
7.113.3.3 active	488
7.113.3.4 attributes	488
7.113.3.5 businessUserInfo	488
7.113.3.6 created	488
7.113.3.7 deleted	488
7.113.3.8 email	489
7.113.3.9 id	489
7.113.3.10 localData	489
7.113.3.11 name	489
7.113.3.12 photoLastUpdated	489
7.113.3.13 photoUrl	489
7.113.3.14 privilege	490
7.113.3.15 serviceLevel	490
7.113.3.16 shardId	490
7.113.3.17 timezone	490
7.113.3.18 updated	490
7.113.3.19 username	490
7.114 qevercloud::UserAttributes Struct Reference	491
7.114.1 Detailed Description	492
7.114.2 Member Function Documentation	492
7.114.2.1 operator"!=()"	492
7.114.2.2 operator=="()"	492
7.114.2.3 print()	492
7.114.3 Member Data Documentation	492
7.114.3.1 businessAddress	492

7.114.3.2 clipFullPage . . . . .	492
7.114.3.3 comments . . . . .	493
7.114.3.4 dailyEmailLimit . . . . .	493
7.114.3.5 dateAgreedToTermsOfService . . . . .	493
7.114.3.6 defaultLatitude . . . . .	493
7.114.3.7 defaultLocationName . . . . .	493
7.114.3.8 defaultLongitude . . . . .	493
7.114.3.9 educationalDiscount . . . . .	493
7.114.3.10 emailAddressLastConfirmed . . . . .	494
7.114.3.11 emailOptOutDate . . . . .	494
7.114.3.12 groupName . . . . .	494
7.114.3.13 hideSponsorBilling . . . . .	494
7.114.3.14 incomingEmailAddress . . . . .	494
7.114.3.15 localData . . . . .	494
7.114.3.16 maxReferrals . . . . .	494
7.114.3.17 optOutMachineLearning . . . . .	495
7.114.3.18 partnerEmailOptInDate . . . . .	495
7.114.3.19 passwordUpdated . . . . .	495
7.114.3.20 preactivation . . . . .	495
7.114.3.21 preferredCountry . . . . .	495
7.114.3.22 preferredLanguage . . . . .	495
7.114.3.23 recentMailedAddresses . . . . .	495
7.114.3.24 recognitionLanguage . . . . .	496
7.114.3.25 refererCode . . . . .	496
7.114.3.26 referralCount . . . . .	496
7.114.3.27 referralProof . . . . .	496
7.114.3.28 reminderEmailConfig . . . . .	496
7.114.3.29 salesforcePushEnabled . . . . .	496
7.114.3.30 sentEmailCount . . . . .	496
7.114.3.31 sentEmailDate . . . . .	497
7.114.3.32 shouldLogClientEvent . . . . .	497
7.114.3.33 twitterId . . . . .	497
7.114.3.34 twitterUserName . . . . .	497
7.114.3.35 useEmailAutoFiling . . . . .	497
7.114.3.36 viewedPromotions . . . . .	497
7.115 qevercloud::UserIdentity Struct Reference . . . . .	497
7.115.1 Detailed Description . . . . .	498
7.115.2 Member Function Documentation . . . . .	498
7.115.2.1 operator"!=( ) . . . . .	498
7.115.2.2 operator==( ) . . . . .	498
7.115.2.3 print( ) . . . . .	499
7.115.3 Member Data Documentation . . . . .	499

7.115.3.1 localData . . . . .	499
7.115.3.2 longIdentifier . . . . .	499
7.115.3.3 stringIdentifier . . . . .	499
7.115.3.4 type . . . . .	499
7.116 qevercloud::UserProfile Struct Reference . . . . .	499
7.116.1 Detailed Description . . . . .	500
7.116.2 Member Function Documentation . . . . .	500
7.116.2.1 operator"!=( ) . . . . .	500
7.116.2.2 operator==( ) . . . . .	500
7.116.2.3 print( ) . . . . .	500
7.116.3 Member Data Documentation . . . . .	501
7.116.3.1 attributes . . . . .	501
7.116.3.2 email . . . . .	501
7.116.3.3 id . . . . .	501
7.116.3.4 joined . . . . .	501
7.116.3.5 localData . . . . .	501
7.116.3.6 name . . . . .	501
7.116.3.7 photoLastUpdated . . . . .	501
7.116.3.8 photoUrl . . . . .	502
7.116.3.9 role . . . . .	502
7.116.3.10 status . . . . .	502
7.116.3.11 username . . . . .	502
7.117 qevercloud::UserStoreServer Class Reference . . . . .	502
7.117.1 Detailed Description . . . . .	504
7.117.2 Constructor & Destructor Documentation . . . . .	504
7.117.2.1 UserStoreServer( ) . . . . .	504
7.117.3 Member Function Documentation . . . . .	504
7.117.3.1 authenticateLongSessionRequest . . . . .	504
7.117.3.2 authenticateLongSessionRequestReady . . . . .	505
7.117.3.3 authenticateToBusinessRequest . . . . .	505
7.117.3.4 authenticateToBusinessRequestReady . . . . .	505
7.117.3.5 checkVersionRequest . . . . .	505
7.117.3.6 checkVersionRequestReady . . . . .	505
7.117.3.7 completeTwoFactorAuthenticationRequest . . . . .	505
7.117.3.8 completeTwoFactorAuthenticationRequestReady . . . . .	506
7.117.3.9 getAccountLimitsRequest . . . . .	506
7.117.3.10 getAccountLimitsRequestReady . . . . .	506
7.117.3.11 getBootstrapInfoRequest . . . . .	506
7.117.3.12 getBootstrapInfoRequestReady . . . . .	506
7.117.3.13 getPublicUserInfoRequest . . . . .	506
7.117.3.14 getPublicUserInfoRequestReady . . . . .	507
7.117.3.15 getUserRequest . . . . .	507

7.117.3.16	getUserRequestReady	507
7.117.3.17	getUserUrlsRequest	507
7.117.3.18	getUserUrlsRequestReady	507
7.117.3.19	inviteToBusinessRequest	507
7.117.3.20	inviteToBusinessRequestReady	507
7.117.3.21	listBusinessInvitationsRequest	508
7.117.3.22	listBusinessInvitationsRequestReady	508
7.117.3.23	listBusinessUsersRequest	508
7.117.3.24	listBusinessUsersRequestReady	508
7.117.3.25	onAuthenticateLongSessionRequestReady	508
7.117.3.26	onAuthenticateToBusinessRequestReady	508
7.117.3.27	onCheckVersionRequestReady	509
7.117.3.28	onCompleteTwoFactorAuthenticationRequestReady	509
7.117.3.29	onGetAccountLimitsRequestReady	509
7.117.3.30	onGetBootstrapInfoRequestReady	509
7.117.3.31	onGetPublicUserInfoRequestReady	509
7.117.3.32	onGetUserRequestReady	509
7.117.3.33	onGetUserUrlsRequestReady	510
7.117.3.34	onInviteToBusinessRequestReady	510
7.117.3.35	onListBusinessInvitationsRequestReady	510
7.117.3.36	onListBusinessUsersRequestReady	510
7.117.3.37	onRemoveFromBusinessRequestReady	510
7.117.3.38	onRequest	510
7.117.3.39	onRevokeLongSessionRequestReady	511
7.117.3.40	onUpdateBusinessUserIdentifierRequestReady	511
7.117.3.41	removeFromBusinessRequest	511
7.117.3.42	removeFromBusinessRequestReady	511
7.117.3.43	revokeLongSessionRequest	511
7.117.3.44	revokeLongSessionRequestReady	511
7.117.3.45	updateBusinessUserIdentifierRequest	511
7.117.3.46	updateBusinessUserIdentifierRequestReady	512
7.118	qevercloud::UserUrls Struct Reference	512
7.118.1	Member Function Documentation	512
7.118.1.1	operator"!=( )	512
7.118.1.2	operator==( )	513
7.118.1.3	print( )	513
7.118.2	Member Data Documentation	513
7.118.2.1	localData	513
7.118.2.2	messageStoreUrl	513
7.118.2.3	noteStoreUrl	513
7.118.2.4	userStoreUrl	513
7.118.2.5	userWebSocketUrl	514

7.118.2.6 utilityUrl . . . . .	514
7.118.2.7 webApiUrlPrefix . . . . .	514
<b>8 File Documentation</b>	<b>515</b>
8.1 AsyncResult.h File Reference . . . . .	515
8.2 AsyncResult.h . . . . .	515
8.3 DurableService.h File Reference . . . . .	516
8.4 DurableService.h . . . . .	517
8.5 EventLoopFinisher.h File Reference . . . . .	518
8.6 EventLoopFinisher.h . . . . .	518
8.7 EverCloudException.h File Reference . . . . .	519
8.8 EverCloudException.h . . . . .	520
8.9 Exceptions.h File Reference . . . . .	521
8.10 Exceptions.h . . . . .	521
8.11 Export.h File Reference . . . . .	524
8.11.1 Macro Definition Documentation . . . . .	524
8.11.1.1 QEVERCLOUD_EXPORT . . . . .	524
8.12 Export.h . . . . .	524
8.13 Constants.h File Reference . . . . .	525
8.14 Constants.h . . . . .	529
8.15 EDAMErrorCode.h File Reference . . . . .	536
8.16 EDAMErrorCode.h . . . . .	539
8.17 Servers.h File Reference . . . . .	548
8.18 Servers.h . . . . .	548
8.19 Services.h File Reference . . . . .	560
8.20 Services.h . . . . .	561
8.21 Types.h File Reference . . . . .	571
8.22 Types.h . . . . .	574
8.23 Globals.h File Reference . . . . .	617
8.24 Globals.h . . . . .	617
8.25 Helpers.h File Reference . . . . .	618
8.26 Helpers.h . . . . .	618
8.27 InkNoteImageDownloader.h File Reference . . . . .	620
8.28 InkNoteImageDownloader.h . . . . .	620
8.29 Log.h File Reference . . . . .	621
8.29.1 Macro Definition Documentation . . . . .	622
8.29.1.1 __QEVERCLOUD_LOG_BASE . . . . .	622
8.29.1.2 QEC_DEBUG . . . . .	623
8.29.1.3 QEC_ERROR . . . . .	623
8.29.1.4 QEC_INFO . . . . .	623
8.29.1.5 QEC_TRACE . . . . .	623
8.29.1.6 QEC_WARNING . . . . .	623

8.30 Log.h . . . . .	624
8.31 OAuth.h File Reference . . . . .	625
8.32 OAuth.h . . . . .	626
8.33 Optional.h File Reference . . . . .	627
8.34 Optional.h . . . . .	627
8.35 Printable.h File Reference . . . . .	630
8.36 Printable.h . . . . .	630
8.37 QEverCloud.h File Reference . . . . .	631
8.38 QEverCloud.h . . . . .	631
8.39 QEverCloudOAuth.h File Reference . . . . .	632
8.40 QEverCloudOAuth.h . . . . .	632
8.41 RequestContext.h File Reference . . . . .	632
8.42 RequestContext.h . . . . .	633
8.43 Thumbnail.h File Reference . . . . .	633
8.44 Thumbnail.h . . . . .	634
8.45 README.md File Reference . . . . .	635

<b>Index</b>	<b>637</b>
--------------	------------



# 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 and 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:

<a href="#">qevercloud</a> . . . . .	19
--------------------------------------	----



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

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

qevercloud::IDurableService::AsyncRequest . . . . .	97
std::exception	
qevercloud::EverCloudException . . . . .	151
qevercloud::EvernoteException . . . . .	159
qevercloud::EDAMInvalidContactsException . . . . .	128
qevercloud::EDAMNotFoundException . . . . .	133
qevercloud::EDAMSystemException . . . . .	137
qevercloud::EDAMSystemExceptionAuthExpired . . . . .	140
qevercloud::EDAMSystemExceptionRateLimitReached . . . . .	144
qevercloud::EDAMUserException . . . . .	146
qevercloud::NetworkException . . . . .	284
qevercloud::ThriftException . . . . .	475
qevercloud::IDurableService . . . . .	170
qevercloud::ILogger . . . . .	171
qevercloud::InkNoteImageDownloader . . . . .	172
qevercloud::IRequestContext . . . . .	242
qevercloud::IRetryPolicy . . . . .	244
qevercloud::QAssociativeContainerConstReferenceWrapper< Container >::iterator . . . . .	245
qevercloud::QAssociativeContainerReferenceWrapper< Container >::iterator . . . . .	246
qevercloud::Optional< T > . . . . .	401
qevercloud::Optional< bool > . . . . .	401
qevercloud::Optional< BusinessInvitationStatus > . . . . .	401
qevercloud::Optional< BusinessUserRole > . . . . .	401
qevercloud::Optional< BusinessUserStatus > . . . . .	401
qevercloud::Optional< CanMoveToContainerStatus > . . . . .	401
qevercloud::Optional< ContactType > . . . . .	401
qevercloud::Optional< double > . . . . .	401
qevercloud::Optional< Guid > . . . . .	401
qevercloud::Optional< IdentityID > . . . . .	401
qevercloud::Optional< MessageEventID > . . . . .	401
qevercloud::Optional< MessageThreadID > . . . . .	401
qevercloud::Optional< NoteSortOrder > . . . . .	401
qevercloud::Optional< PremiumOrderStatus > . . . . .	401
qevercloud::Optional< PrivilegeLevel > . . . . .	401
qevercloud::Optional< QByteArray > . . . . .	401

qevercloud::Optional< qevercloud::Accounting > . . . . .	401
qevercloud::Optional< qevercloud::AccountLimits > . . . . .	401
qevercloud::Optional< qevercloud::BusinessNotebook > . . . . .	401
qevercloud::Optional< qevercloud::BusinessUserAttributes > . . . . .	401
qevercloud::Optional< qevercloud::BusinessUserInfo > . . . . .	401
qevercloud::Optional< qevercloud::CanMoveToContainerRestrictions > . . . . .	401
qevercloud::Optional< qevercloud::Contact > . . . . .	401
qevercloud::Optional< qevercloud::Data > . . . . .	401
qevercloud::Optional< qevercloud::EDAMNotFoundException > . . . . .	401
qevercloud::Optional< qevercloud::EDAMUserException > . . . . .	401
qevercloud::Optional< qevercloud::Identity > . . . . .	401
qevercloud::Optional< qevercloud::LazyMap > . . . . .	401
qevercloud::Optional< qevercloud::Note > . . . . .	401
qevercloud::Optional< qevercloud::NoteAttributes > . . . . .	401
qevercloud::Optional< qevercloud::NotebookRecipientSettings > . . . . .	401
qevercloud::Optional< qevercloud::NotebookRestrictions > . . . . .	401
qevercloud::Optional< qevercloud::NoteFilter > . . . . .	401
qevercloud::Optional< qevercloud::NoteLimits > . . . . .	401
qevercloud::Optional< qevercloud::NoteRestrictions > . . . . .	401
qevercloud::Optional< qevercloud::NoteShareRelationshipRestrictions > . . . . .	401
qevercloud::Optional< qevercloud::PublicUserInfo > . . . . .	401
qevercloud::Optional< qevercloud::Publishing > . . . . .	401
qevercloud::Optional< qevercloud::ResourceAttributes > . . . . .	401
qevercloud::Optional< qevercloud::SavedSearchScope > . . . . .	401
qevercloud::Optional< qevercloud::SharedNotebookRecipientSettings > . . . . .	401
qevercloud::Optional< qevercloud::ShareRelationshipRestrictions > . . . . .	401
qevercloud::Optional< qevercloud::User > . . . . .	401
qevercloud::Optional< qevercloud::UserAttributes > . . . . .	401
qevercloud::Optional< qevercloud::UserIdentity > . . . . .	401
qevercloud::Optional< qevercloud::UserUrls > . . . . .	401
qevercloud::Optional< qint16 > . . . . .	401
qevercloud::Optional< qint32 > . . . . .	401
qevercloud::Optional< qint64 > . . . . .	401
qevercloud::Optional< QList< EDAMInvalidContactReason > > . . . . .	401
qevercloud::Optional< QList< Guid > > . . . . .	401
qevercloud::Optional< QList< IdentityID > > . . . . .	401
qevercloud::Optional< QList< qevercloud::Contact > > . . . . .	401
qevercloud::Optional< QList< qevercloud::InvitationShareRelationship > > . . . . .	401
qevercloud::Optional< QList< qevercloud::LinkedNotebook > > . . . . .	401
qevercloud::Optional< QList< qevercloud::ManageNotebookSharesError > > . . . . .	401
qevercloud::Optional< QList< qevercloud::ManageNoteSharesError > > . . . . .	401
qevercloud::Optional< QList< qevercloud::MemberShareRelationship > > . . . . .	401
qevercloud::Optional< QList< qevercloud::Note > > . . . . .	401
qevercloud::Optional< QList< qevercloud::Notebook > > . . . . .	401
qevercloud::Optional< QList< qevercloud::NotebookDescriptor > > . . . . .	401
qevercloud::Optional< QList< qevercloud::NoteInvitationShareRelationship > > . . . . .	401
qevercloud::Optional< QList< qevercloud::NoteMemberShareRelationship > > . . . . .	401
qevercloud::Optional< QList< qevercloud::RelatedContent > > . . . . .	401
qevercloud::Optional< QList< qevercloud::RelatedContentImage > > . . . . .	401
qevercloud::Optional< QList< qevercloud::Resource > > . . . . .	401
qevercloud::Optional< QList< qevercloud::SavedSearch > > . . . . .	401
qevercloud::Optional< QList< qevercloud::SharedNote > > . . . . .	401
qevercloud::Optional< QList< qevercloud::SharedNotebook > > . . . . .	401
qevercloud::Optional< QList< qevercloud::Tag > > . . . . .	401
qevercloud::Optional< QList< qevercloud::UserIdentity > > . . . . .	401
qevercloud::Optional< QList< qevercloud::UserProfile > > . . . . .	401
qevercloud::Optional< QList< qint64 > > . . . . .	401
qevercloud::Optional< QList< UserID > > . . . . .	401



qevercloud::Optional< QMap< Guid, qint32 > > . . . . .	401
qevercloud::Optional< QMap< QString, QString > > . . . . .	401
qevercloud::Optional< QSet< QString > > . . . . .	401
qevercloud::Optional< QSet< RelatedContentType > > . . . . .	401
qevercloud::Optional< QString > . . . . .	401
qevercloud::Optional< QStringList > . . . . .	401
qevercloud::Optional< QueryFormat > . . . . .	401
qevercloud::Optional< RecipientStatus > . . . . .	401
qevercloud::Optional< RelatedContentAccess > . . . . .	401
qevercloud::Optional< RelatedContentType > . . . . .	401
qevercloud::Optional< ReminderEmailConfig > . . . . .	401
qevercloud::Optional< ServiceLevel > . . . . .	401
qevercloud::Optional< SharedNotebookInstanceRestrictions > . . . . .	401
qevercloud::Optional< SharedNotebookPrivilegeLevel > . . . . .	401
qevercloud::Optional< SharedNotePrivilegeLevel > . . . . .	401
qevercloud::Optional< ShareRelationshipPrivilegeLevel > . . . . .	401
qevercloud::Optional< Timestamp > . . . . .	401
qevercloud::Optional< UserID > . . . . .	401
qevercloud::Optional< UserIdentityType > . . . . .	401
qevercloud::Printable . . . . .	408
qevercloud::AccountLimits . . . . .	94
qevercloud::Accounting . . . . .	89
qevercloud::AuthenticationResult . . . . .	101
qevercloud::BootstrapInfo . . . . .	104
qevercloud::BootstrapProfile . . . . .	106
qevercloud::BootstrapSettings . . . . .	108
qevercloud::BusinessInvitation . . . . .	111
qevercloud::BusinessNotebook . . . . .	114
qevercloud::BusinessUserAttributes . . . . .	115
qevercloud::BusinessUserInfo . . . . .	118
qevercloud::CanMoveToContainerRestrictions . . . . .	120
qevercloud::Contact . . . . .	121
qevercloud::CreateOrUpdateNotebookSharesResult . . . . .	124
qevercloud::Data . . . . .	126
qevercloud::EDAMInvalidContactsException . . . . .	128
qevercloud::EDAMNotFoundException . . . . .	133
qevercloud::EDAMSystemException . . . . .	137
qevercloud::EDAMUserException . . . . .	146
qevercloud::EverCloudLocalData . . . . .	155
qevercloud::EvernoteOAuthWebView::OAuthResult . . . . .	399
qevercloud::Identity . . . . .	167
qevercloud::InvitationShareRelationship . . . . .	240
qevercloud::LazyMap . . . . .	262
qevercloud::LinkedNotebook . . . . .	264
qevercloud::ManageNoteSharesError . . . . .	274
qevercloud::ManageNoteSharesParameters . . . . .	276
qevercloud::ManageNoteSharesResult . . . . .	279
qevercloud::ManageNotebookSharesError . . . . .	268
qevercloud::ManageNotebookSharesParameters . . . . .	270
qevercloud::ManageNotebookSharesResult . . . . .	272
qevercloud::MemberShareRelationship . . . . .	281
qevercloud::Note . . . . .	287
qevercloud::NoteAttributes . . . . .	293
qevercloud::NoteCollectionCounts . . . . .	317
qevercloud::NoteEmailParameters . . . . .	320
qevercloud::NoteFilter . . . . .	322
qevercloud::NoteInvitationShareRelationship . . . . .	326
qevercloud::NoteLimits . . . . .	328

qevercloud::NoteList	330
qevercloud::NoteMemberShareRelationship	333
qevercloud::NoteMetadata	335
qevercloud::NoteRestrictions	338
qevercloud::NoteResultSpec	341
qevercloud::NoteShareRelationshipRestrictions	344
qevercloud::NoteShareRelationships	346
qevercloud::NoteVersionId	396
qevercloud::Notebook	299
qevercloud::NotebookDescriptor	304
qevercloud::NotebookRecipientSettings	306
qevercloud::NotebookRestrictions	308
qevercloud::NotebookShareTemplate	315
qevercloud::NotesMetadataList	348
qevercloud::NotesMetadataResultSpec	351
qevercloud::PublicUserInfo	411
qevercloud::Publishing	413
qevercloud::RelatedContent	417
qevercloud::RelatedContentImage	421
qevercloud::RelatedQuery	423
qevercloud::RelatedResult	426
qevercloud::RelatedResultSpec	430
qevercloud::Resource	433
qevercloud::ResourceAttributes	437
qevercloud::SavedSearch	440
qevercloud::SavedSearchScope	443
qevercloud::ShareRelationshipRestrictions	455
qevercloud::ShareRelationships	457
qevercloud::SharedNote	445
qevercloud::SharedNoteTemplate	453
qevercloud::SharedNotebook	447
qevercloud::SharedNotebookRecipientSettings	451
qevercloud::SyncChunk	459
qevercloud::SyncChunkFilter	465
qevercloud::SyncState	470
qevercloud::Tag	473
qevercloud::UpdateNotelfUsnMatchesResult	484
qevercloud::User	486
qevercloud::UserAttributes	491
qevercloud::UserIdentity	497
qevercloud::UserProfile	499
qevercloud::UserUrls	512
qevercloud::QAssociativeContainerConstReferenceWrapper< Container >	415
qevercloud::QAssociativeContainerReferenceWrapper< Container >	416
QDialog	
qevercloud::EvernoteOAuthDialog	161
QObject	
qevercloud::AsyncResult	98
qevercloud::EventLoopFinisher	150
qevercloud::EverCloudExceptionData	153
qevercloud::EvernoteExceptionData	160
qevercloud::EDAMInvalidContactsExceptionData	131
qevercloud::EDAMNotFoundExceptionData	135
qevercloud::EDAMSystemExceptionData	142
qevercloud::EDAMSystemExceptionAuthExpiredData	141
qevercloud::EDAMSystemExceptionRateLimitReachedData	145
qevercloud::EDAMUserExceptionData	149
qevercloud::NetworkExceptionData	286

qevercloud::ThriftExceptionData . . . . .	478
qevercloud::INoteStore . . . . .	176
qevercloud::IUserStore . . . . .	248
qevercloud::NoteStoreServer . . . . .	354
qevercloud::UserStoreServer . . . . .	502
QWidget	
qevercloud::EvernoteOAuthWebView . . . . .	164
qevercloud::IDurableService::SyncRequest . . . . .	469
qevercloud::Thumbnail . . . . .	479



## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">qevercloud::Accounting</a>	89
<a href="#">qevercloud::AccountLimits</a>	94
<a href="#">qevercloud::IDurableService::AsyncRequest</a>	97
<a href="#">qevercloud::AsyncResult</a>	
Returned by asynchronous versions of functions	98
<a href="#">qevercloud::AuthenticationResult</a>	101
<a href="#">qevercloud::BootstrapInfo</a>	104
<a href="#">qevercloud::BootstrapProfile</a>	106
<a href="#">qevercloud::BootstrapSettings</a>	108
<a href="#">qevercloud::BusinessInvitation</a>	111
<a href="#">qevercloud::BusinessNotebook</a>	114
<a href="#">qevercloud::BusinessUserAttributes</a>	115
<a href="#">qevercloud::BusinessUserInfo</a>	118
<a href="#">qevercloud::CanMoveToContainerRestrictions</a>	120
<a href="#">qevercloud::Contact</a>	121
<a href="#">qevercloud::CreateOrUpdateNotebookSharesResult</a>	124
<a href="#">qevercloud::Data</a>	126
<a href="#">qevercloud::EDAMInvalidContactsException</a>	128
<a href="#">qevercloud::EDAMInvalidContactsExceptionData</a>	131
<a href="#">qevercloud::EDAMNotFoundException</a>	133
<a href="#">qevercloud::EDAMNotFoundExceptionData</a>	135
<a href="#">qevercloud::EDAMSystemException</a>	137
<a href="#">qevercloud::EDAMSystemExceptionAuthExpired</a>	140
<a href="#">qevercloud::EDAMSystemExceptionAuthExpiredData</a>	141
<a href="#">qevercloud::EDAMSystemExceptionData</a>	142
<a href="#">qevercloud::EDAMSystemExceptionRateLimitReached</a>	144
<a href="#">qevercloud::EDAMSystemExceptionRateLimitReachedData</a>	145
<a href="#">qevercloud::EDAMUserException</a>	146
<a href="#">qevercloud::EDAMUserExceptionData</a>	149
<a href="#">qevercloud::EventLoopFinisher</a>	150
<a href="#">qevercloud::EverCloudException</a>	151
<a href="#">qevercloud::EverCloudExceptionData</a>	
<a href="#">EverCloudException</a> counterpart for asynchronous API	153

<a href="#">qevercloud::EverCloudLocalData</a>	
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	155
<a href="#">qevercloud::EvernoteException</a>	159
<a href="#">qevercloud::EvernoteExceptionData</a>	160
<a href="#">qevercloud::EvernoteOAuthDialog</a>	
Authorizes your app with the Evernote service by means of OAuth authentication	161
<a href="#">qevercloud::EvernoteOAuthWebView</a>	
The class is tailored specifically for OAuth authorization with Evernote	164
<a href="#">qevercloud::Identity</a>	167
<a href="#">qevercloud::IDurableService</a>	170
<a href="#">qevercloud::ILogger</a>	171
<a href="#">qevercloud::InkNoteImageDownloader</a>	
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)	172
<a href="#">qevercloud::INoteStore</a>	176
<a href="#">qevercloud::InvitationShareRelationship</a>	240
<a href="#">qevercloud::IRequestContext</a>	242
<a href="#">qevercloud::IRetryPolicy</a>	244
<a href="#">qevercloud::QAssociativeContainerConstReferenceWrapper&lt; Container &gt;::iterator</a>	245
<a href="#">qevercloud::QAssociativeContainerReferenceWrapper&lt; Container &gt;::iterator</a>	246
<a href="#">qevercloud::IUserStore</a>	248
<a href="#">qevercloud::LazyMap</a>	262
<a href="#">qevercloud::LinkedNotebook</a>	264
<a href="#">qevercloud::ManageNotebookSharesError</a>	268
<a href="#">qevercloud::ManageNotebookSharesParameters</a>	270
<a href="#">qevercloud::ManageNotebookSharesResult</a>	272
<a href="#">qevercloud::ManageNoteSharesError</a>	274
<a href="#">qevercloud::ManageNoteSharesParameters</a>	276
<a href="#">qevercloud::ManageNoteSharesResult</a>	279
<a href="#">qevercloud::MemberShareRelationship</a>	281
<a href="#">qevercloud::NetworkException</a>	
QNetworkReply level errors	284
<a href="#">qevercloud::NetworkExceptionData</a>	286
<a href="#">qevercloud::Note</a>	287
<a href="#">qevercloud::NoteAttributes</a>	293
<a href="#">qevercloud::Notebook</a>	299
<a href="#">qevercloud::NotebookDescriptor</a>	304
<a href="#">qevercloud::NotebookRecipientSettings</a>	306
<a href="#">qevercloud::NotebookRestrictions</a>	308
<a href="#">qevercloud::NotebookShareTemplate</a>	315
<a href="#">qevercloud::NoteCollectionCounts</a>	317
<a href="#">qevercloud::NoteEmailParameters</a>	320
<a href="#">qevercloud::NoteFilter</a>	322
<a href="#">qevercloud::NoteInvitationShareRelationship</a>	326
<a href="#">qevercloud::NoteLimits</a>	328
<a href="#">qevercloud::NoteList</a>	330
<a href="#">qevercloud::NoteMemberShareRelationship</a>	333
<a href="#">qevercloud::NoteMetadata</a>	335
<a href="#">qevercloud::NoteRestrictions</a>	338
<a href="#">qevercloud::NoteResultSpec</a>	341
<a href="#">qevercloud::NoteShareRelationshipRestrictions</a>	344
<a href="#">qevercloud::NoteShareRelationships</a>	346
<a href="#">qevercloud::NotesMetadataList</a>	348
<a href="#">qevercloud::NotesMetadataResultSpec</a>	351

qevercloud::NoteStoreServer	
Represents customizable server for NoteStore requests. It is primarily used for testing of QEverCloud	354
qevercloud::NoteVersionId	396
qevercloud::EvernoteOAuthWebView::OAuthResult	399
qevercloud::Optional< T >	401
qevercloud::Printable	408
qevercloud::PublicUserInfo	411
qevercloud::Publishing	413
qevercloud::QAssociativeContainerConstReferenceWrapper< Container >	415
qevercloud::QAssociativeContainerReferenceWrapper< Container >	416
qevercloud::RelatedContent	417
qevercloud::RelatedContentImage	421
qevercloud::RelatedQuery	423
qevercloud::RelatedResult	426
qevercloud::RelatedResultSpec	430
qevercloud::Resource	433
qevercloud::ResourceAttributes	437
qevercloud::SavedSearch	440
qevercloud::SavedSearchScope	443
qevercloud::SharedNote	445
qevercloud::SharedNotebook	447
qevercloud::SharedNotebookRecipientSettings	451
qevercloud::SharedNoteTemplate	453
qevercloud::ShareRelationshipRestrictions	455
qevercloud::ShareRelationships	457
qevercloud::SyncChunk	459
qevercloud::SyncChunkFilter	465
qevercloud::IDurableService::SyncRequest	469
qevercloud::SyncState	470
qevercloud::Tag	473
qevercloud::ThriftException	475
qevercloud::ThriftExceptionData	478
qevercloud::Thumbnail	
The class is for downloading thumbnails for notes and resources from Evernote servers	479
qevercloud::UpdateNoteIfUsnMatchesResult	484
qevercloud::User	486
qevercloud::UserAttributes	491
qevercloud::UserIdentity	497
qevercloud::UserProfile	499
qevercloud::UserStoreServer	
Represents customizable server for UserStore requests. It is primarily used for testing of QEverCloud	502
qevercloud::UserUrls	512





## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

<a href="#">AsyncResult.h</a>	515
<a href="#">DurableService.h</a>	516
<a href="#">EventLoopFinisher.h</a>	518
<a href="#">EverCloudException.h</a>	519
<a href="#">Exceptions.h</a>	521
<a href="#">Export.h</a>	524
<a href="#">Constants.h</a>	525
<a href="#">EDAMErrorCode.h</a>	536
<a href="#">Servers.h</a>	548
<a href="#">Services.h</a>	560
<a href="#">Types.h</a>	571
<a href="#">Globals.h</a>	617
<a href="#">Helpers.h</a>	618
<a href="#">InkNoteImageDownloader.h</a>	620
<a href="#">Log.h</a>	621
<a href="#">OAuth.h</a>	625
<a href="#">Optional.h</a>	627
<a href="#">Printable.h</a>	630
<a href="#">QEverCloud.h</a>	631
<a href="#">QEverCloudOAuth.h</a>	632
<a href="#">RequestContext.h</a>	632
<a href="#">Thumbnail.h</a>	633



## Chapter 6

# Namespace Documentation

### 6.1 qevercloud Namespace Reference

#### Classes

- struct [Accounting](#)
- struct [AccountLimits](#)
- class [AsyncResult](#)

*Returned by asynchronous 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](#) = quint64
- using [IdentityID](#) = quint64
- using [UserID](#) = quint32
- using [Guid](#) = QString
- using [Timestamp](#) = quint64
- using [MessageEventID](#) = quint64
- using [MessageThreadID](#) = quint64
- 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.*
- [QEVERTCLOUD\\_EXPORT IRequestContextPtr newRequestContext](#) (QString authenticationToken={}, quint64 requestTimeout=DEFAULT\_REQUEST\_TIMEOUT\_MSEC, bool increaseRequestTimeoutExponentially=DEFAULT\_REQUEST\_TIMEOUT\_EXPONENTIAL\_INCREASE, quint64 maxRequestTimeout=DEFAULT\_MAX\_REQUEST\_TIMEOUT\_MSEC, quint32 maxRequestRetryCount=DEFAULT\_MAX\_REQUEST\_RETRY\_COUNT, QList< QNetworkCookie > cookies={})

## Variables

- class [QEVERTCLOUD\\_EXPORT EverCloudExceptionData](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [EDAM\\_ATTRIBUTE\\_LEN\\_MIN](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [EDAM\\_ATTRIBUTE\\_LEN\\_MAX](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_ATTRIBUTE\\_REGEX](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [EDAM\\_ATTRIBUTE\\_LIST\\_MAX](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [EDAM\\_ATTRIBUTE\\_MAP\\_MAX](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [EDAM\\_GUID\\_LEN\\_MIN](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [EDAM\\_GUID\\_LEN\\_MAX](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_GUID\\_REGEX](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [EDAM\\_EMAIL\\_LEN\\_MIN](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [EDAM\\_EMAIL\\_LEN\\_MAX](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_EMAIL\\_LOCAL\\_REGEX](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_EMAIL\\_DOMAIN\\_REGEX](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_EMAIL\\_REGEX](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_VAT\\_REGEX](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [EDAM\\_TIMEZONE\\_LEN\\_MIN](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [EDAM\\_TIMEZONE\\_LEN\\_MAX](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_TIMEZONE\\_REGEX](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [EDAM\\_MIME\\_LEN\\_MIN](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [EDAM\\_MIME\\_LEN\\_MAX](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_MIME\\_REGEX](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_MIME\\_TYPE\\_GIF](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_MIME\\_TYPE\\_JPEG](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_MIME\\_TYPE\\_PNG](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_MIME\\_TYPE\\_TIFF](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_MIME\\_TYPE\\_BMP](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_MIME\\_TYPE\\_WAV](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_MIME\\_TYPE\\_MP3](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_MIME\\_TYPE\\_AMR](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_MIME\\_TYPE\\_AAC](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_MIME\\_TYPE\\_M4A](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_MIME\\_TYPE\\_MP4\\_VIDEO](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_MIME\\_TYPE\\_INK](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_MIME\\_TYPE\\_PDF](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_MIME\\_TYPE\\_DEFAULT](#)
- [QEVERTCLOUD\\_EXPORT](#) const QSet< QString > [EDAM\\_MIME\\_TYPES](#)
- [QEVERTCLOUD\\_EXPORT](#) const QSet< QString > [EDAM\\_INDEXABLE\\_RESOURCE\\_MIME\\_TYPES](#)
- [QEVERTCLOUD\\_EXPORT](#) const QSet< QString > [EDAM\\_INDEXABLE\\_PLAINTEXT\\_MIME\\_TYPES](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [EDAM\\_SEARCH\\_QUERY\\_LEN\\_MIN](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [EDAM\\_SEARCH\\_QUERY\\_LEN\\_MAX](#)
- [QEVERTCLOUD\\_EXPORT](#) const QString [EDAM\\_SEARCH\\_QUERY\\_REGEX](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [EDAM\\_HASH\\_LEN](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [EDAM\\_USER\\_USERNAME\\_LEN\\_MIN](#)
- [QEVERTCLOUD\\_EXPORT](#) const quint32 [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 quint64
```

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 quint64
```

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 quint64
```

A sequence number for the MessageStore subsystem.

### 6.1.2.12 MessageThreadID

```
using qevercloud::MessageThreadID = typedef quint64
```

A type alias for the primary identifiers for MessageThread objects.

### 6.1.2.13 Timestamp

```
using qevercloud::Timestamp = typedef quint64
```

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 quint32
```

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	
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

<code>DO_NOT_SEND</code>	
<code>SEND_DAILY_EMAIL</code>	

### 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

<code>BASIC</code>	
<code>PLUS</code>	
<code>PREMIUM</code>	
<code>BUSINESS</code>	

### 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

<code>ASSIGNED</code>	
<code>NO_SHARED_NOTEBOOKS</code>	



### 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 the 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 quint32 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 quint32 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 quint32 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 quint32 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 service 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 quint32 qevercloud::EDAM_MAX_PREFERENCES [extern]
```

Maximum number of name/value pairs allowed

#### 6.1.5.58 EDAM\_MAX\_VALUES\_PER\_PREFERENCE

```
QEVERCLOUD_EXPORT const quint32 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 quint32 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 quint32 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 quint32 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 quint32 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 quint32 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 quint32 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 quint32 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 quint32 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 quint32 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 quint32 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 quint32 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 quint32 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 quint32 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 quint32 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 quint32 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 revalidated 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 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 quint32 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 quint32 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 quint32 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 quint32 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 quint32 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 quint32 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 quint32 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 quint32 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 maximum 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 quint32 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 quint32 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 quint32 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 quint32 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](http://en.wikipedia.org/wiki/VAT_identification_number) ref <http://my.safaribooksonline.com/book/programming/r>

#### 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
```





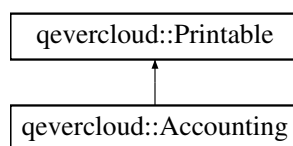
## Chapter 7

# 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 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

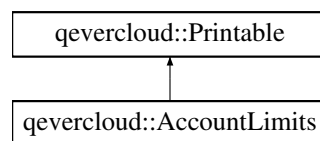
```
Optional< qint64 > qevercloud::Accounting::uploadLimitNextMonth
```

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 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

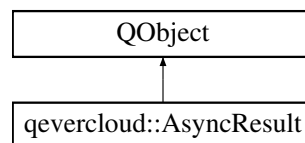
```
const char* qevercloud::IDurableService::AsyncRequest::m_name
```

## 7.4 qevercloud::AsyncResult Class Reference

Returned by asynchronous 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 completion.*

### Public Member Functions

- [AsyncResult](#) (QString url, QByteArray postData, [IRequestContextPtr](#) ctx, [ReadFunctionType](#) read↵  
Function=[AsyncResult::asls](#), bool autoDelete=true, QObject \*parent=nullptr)
- [AsyncResult](#) (QNetworkRequest request, QByteArray postData, [IRequestContextPtr](#) ctx, [ReadFunctionType](#)  
readFunction=[AsyncResult::asls](#), bool autoDelete=true, QObject \*parent=nullptr)
- [AsyncResult](#) (QVariant result, [EverCloudExceptionDataPtr](#) error, [IRequestContextPtr](#) ctx, bool auto↵  
Delete=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 asynchronous 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 completion.

## Parameters

<i>result</i>	Request result
<i>error</i>	Non-nullptr in case of an error. See <a href="#">EverCloudExceptionData</a> for more details
<i>ctx</i>	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 its lifetime explicitly.

7.4.4.3 `waitForFinished()`

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

Wait for asynchronous operation to complete.

## Parameters

<i>timeout</i>	Maximum time to wait in milliseconds. Forever if <code>&lt; 0</code> .
----------------	--

## Returns

true if finished successfully, false in case of the timeout

## 7.4.5 Friends And Related Function Documentation

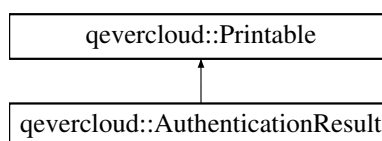
7.4.5.1 `DurableService`

```
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 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

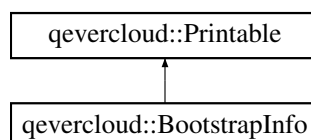
```
Optional< QString > qevercloud::AuthenticationResult::webApiUrlPrefix
```

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 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

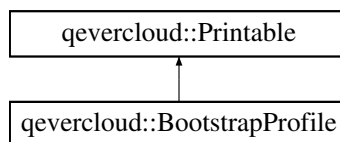
`QList< BootstrapProfile >` `qevercloud::BootstrapInfo::profiles`

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 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

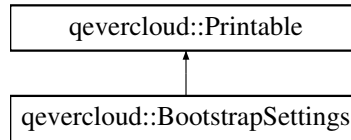
[BootstrapSettings](#) qevercloud::BootstrapProfile::settings

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 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](http://www.evernote.com)) or both the hostname and port (e.g. [www.evernote.com:8080](http://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

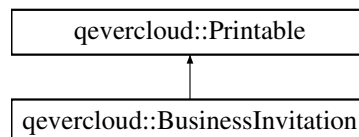
QString qevercloud::BootstrapSettings::supportUrl

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 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

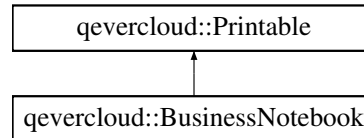
`Optional< BusinessInvitationStatus > qevercloud::BusinessInvitation::status`

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 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

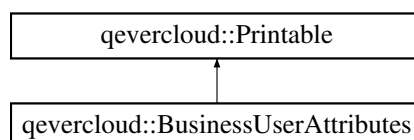
[Optional](#)< bool > qevercloud::BusinessNotebook::recommended

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 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

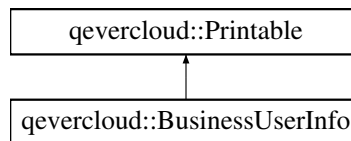
`Optional< QString > qevercloud::BusinessUserAttributes::workPhone`

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 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

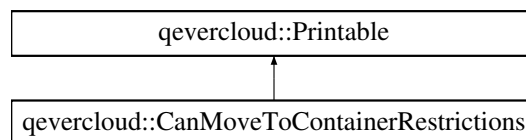
```
Optional< Timestamp > qevercloud::BusinessUserInfo::updated
```

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 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

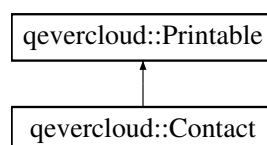
```
EverCloudLocalData qevercloud::CanMoveToContainerRestrictions::localData
```

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 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

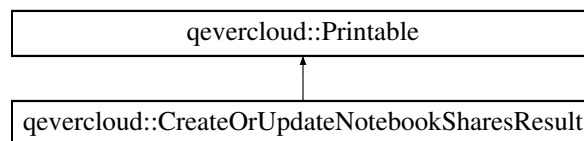
```
Optional< ContactType > qevercloud::Contact::type
```

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 Attributes

- [EverCloudLocalData](#) [localData](#)
- [Optional](#)< qint32 > [updateSequenceNum](#)
- [Optional](#)< QList< [SharedNotebook](#) > > [matchingShares](#)

### Properties

- [Optional](#)QList< [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 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

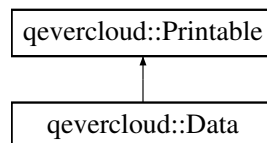
### 7.15.4.1 matchingShares

```
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 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

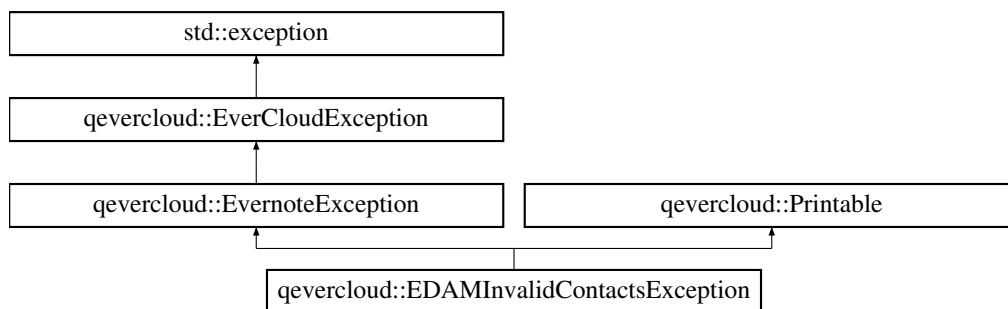
[Optional](#)< `qint32` > `qevercloud::Data::size`

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 Attributes

- QList< [Contact](#) > [contacts](#)
- [Optional](#)< QString > [parameter](#)
- [Optional](#)< QList< [EDAMInvalidContactReason](#) > > [reasons](#)

### Properties

- OptionalQList< [EDAMInvalidContactReason](#) > [reasons](#)



## Additional Inherited Members

### 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

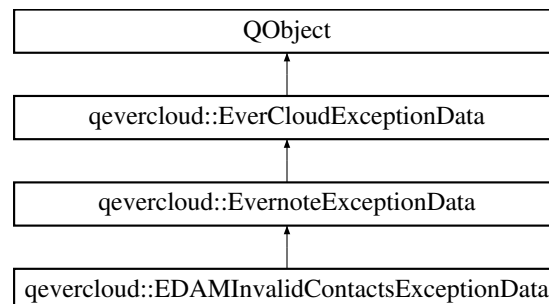
### 7.17.5.1 reasons

`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

## Protected Attributes

- `QList< Contact > m_contacts`
- `Optional< QString > m_parameter`
- `Optional< QList< EDAMInvalidContactReason > > m_reasons`

## Additional Inherited Members

### 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]
```

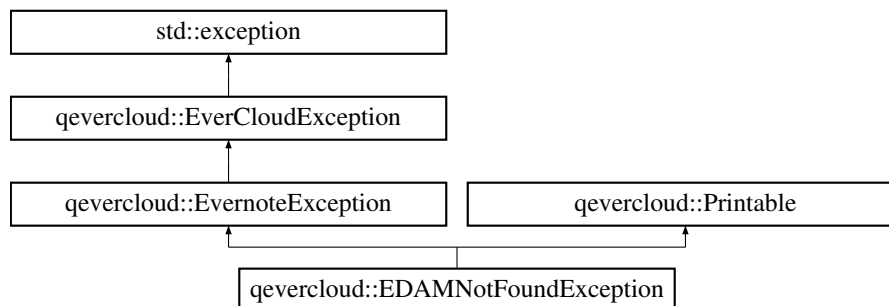
## 7.18.4.3 m\_reasons

```
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 Attributes

- [Optional](#)< QString > [identifier](#)
- [Optional](#)< QString > [key](#)

## Additional Inherited Members

## 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
```

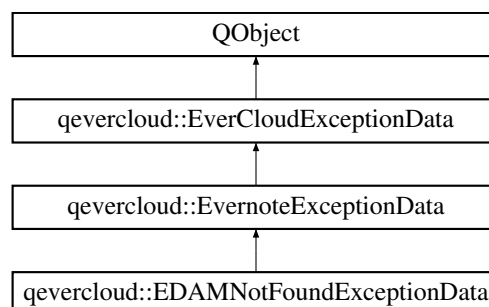
### 7.19.4.2 key

```
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

## Protected Attributes

- [Optional](#)< QString > [m\\_identifier](#)
- [Optional](#)< QString > [m\\_key](#)

## Additional Inherited Members

### 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]`

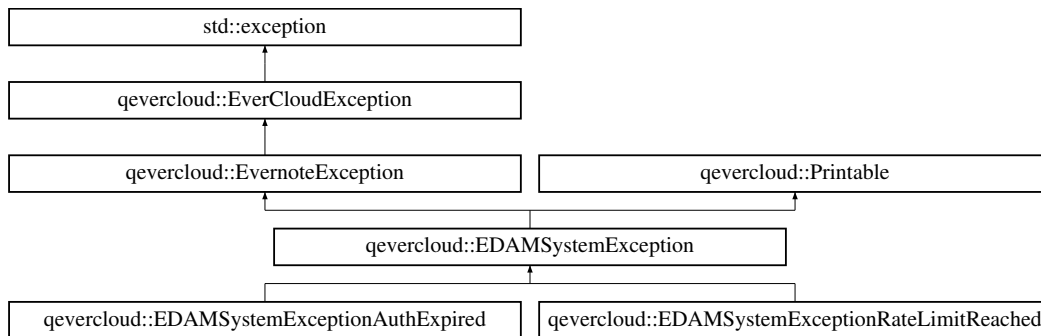
## 7.20.4.2 m\_key

`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 Attributes

- [EDAMErrorCode errorCode](#)
- [Optional< QString > message](#)
- [Optional< qint32 > rateLimitDuration](#)

## Additional Inherited Members

### 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`

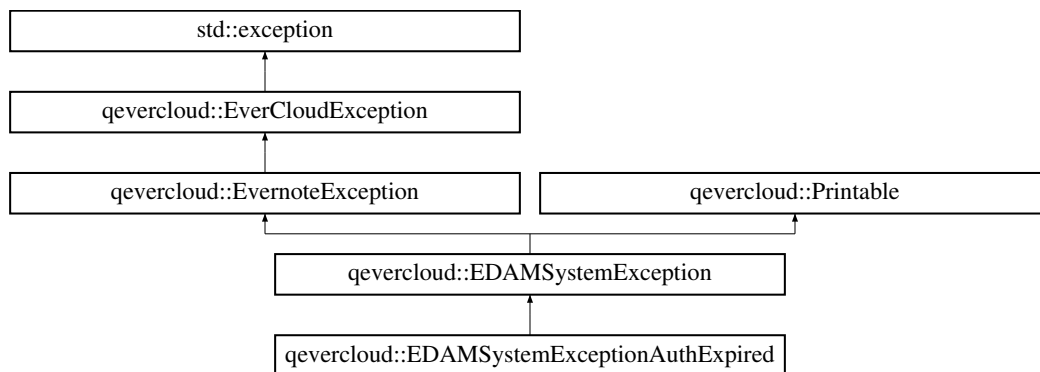
### 7.21.4.3 rateLimitDuration

`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

### Additional Inherited Members

#### 7.22.1 Detailed Description

`EDAMSystemException` for `errorCode = AUTH_EXPIRED`

#### 7.22.2 Member Function Documentation

### 7.22.2.1 exceptionData()

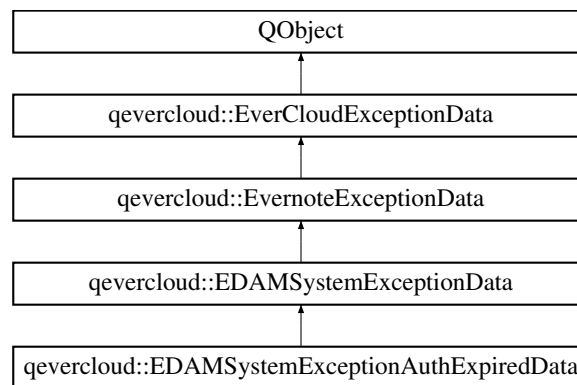
```
virtual EverCloudExceptionDataPtr qevercloud::EDAMSystemExceptionAuthExpired::exceptionData (
) const [override], [virtual]
```

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

### Additional Inherited Members

#### 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]
```

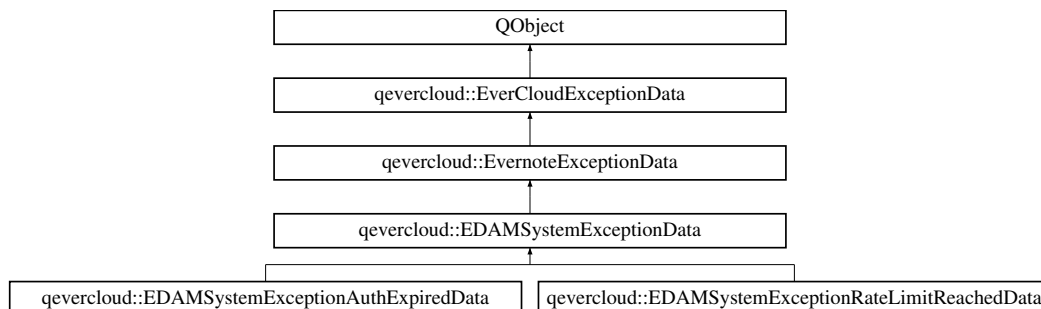
If you want to throw an exception that corresponds to a received [EverCloudExceptionData](#) descendant than call this function. Do not use `throw` statement, it's not polymorphic.

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

### Protected Attributes

- [EDAMErrorCode](#) m\_errorCode
- [Optional](#)< QString > m\_message
- [Optional](#)< qint32 > m\_rateLimitDuration

## Additional Inherited Members

### 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::EDAMSystemExceptionAuthExpiredData](#).

### 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]`

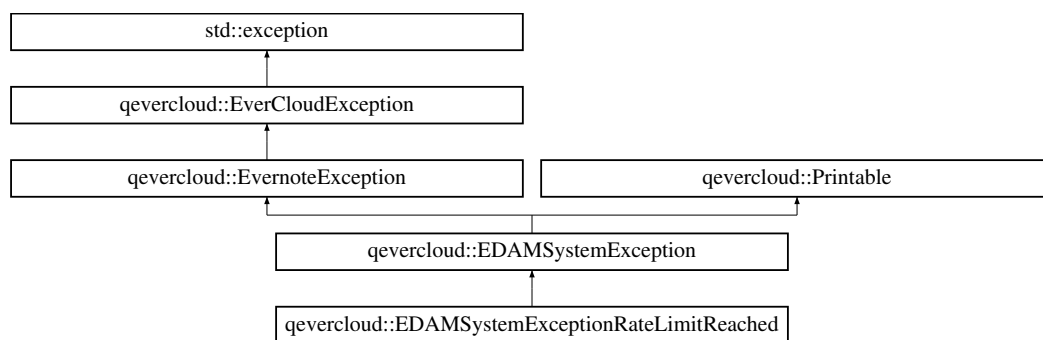
#### 7.24.4.3 m\_rateLimitDuration

`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

### Additional Inherited Members

#### 7.25.1 Detailed Description

`EDAMSystemException` for `errorCode = RATE_LIMIT_REACHED`

#### 7.25.2 Member Function Documentation



### 7.25.2.1 exceptionData()

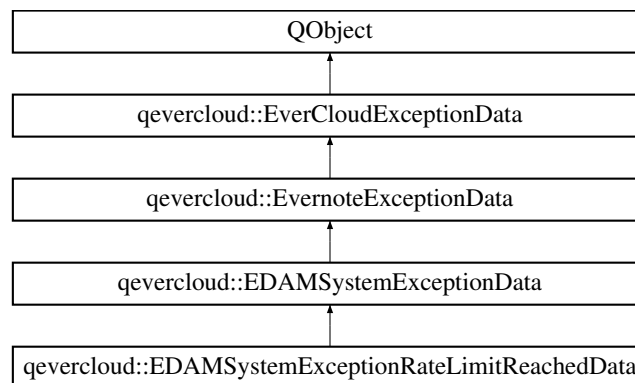
```
virtual EverCloudExceptionDataPtr qevercloud::EDAMSystemExceptionRateLimitReached::exception←
Data ( ) const [override], [virtual]
```

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

### Additional Inherited Members

#### 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]
```

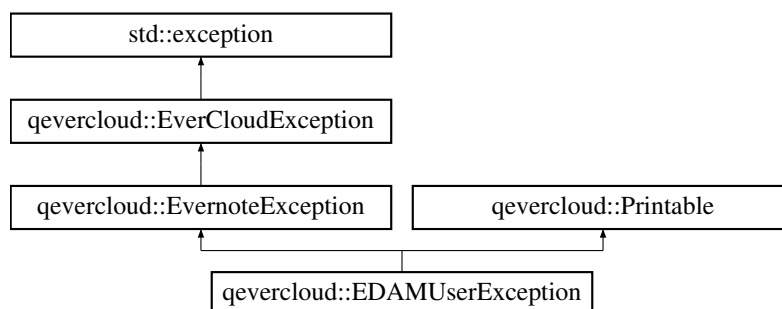
If you want to throw an exception that corresponds to a received [EverCloudExceptionData](#) descendant than call this function. Do not use `throw` statement, it's not polymorphic.

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 Attributes

- [EDAMErrorCode](#) `errorCode`
- [Optional](#)< [QString](#) > `parameter`

## Additional Inherited Members

### 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 [EDAMErrorCode](#).

`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
```

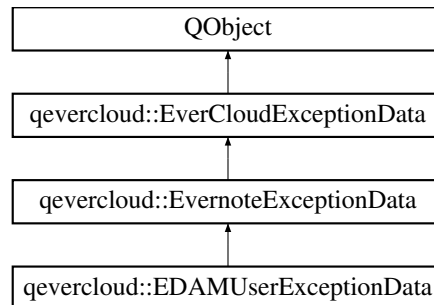
## 7.27.4.2 parameter

```
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

## Protected Attributes

- [EDAMErrorCode](#) m\_errorCode
- [Optional](#)< QString > m\_parameter

## Additional Inherited Members

## 7.28.1 Detailed Description

Asynchronous API counterpart of [EDAMUserException](#). See [EverCloudExceptionData](#) for more details.

## 7.28.2 Constructor &amp; 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]
```

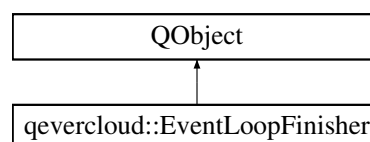
### 7.28.4.2 `m_parameter`

```
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

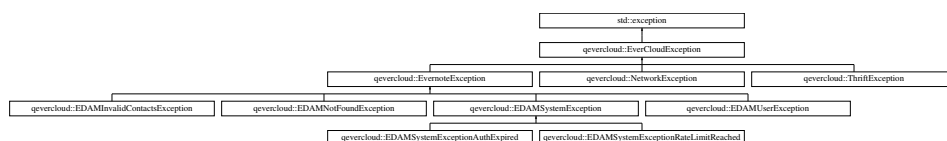
#### 7.29.2.1 stopEventLoop

```
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::exceptionData ( ) const [virtual]
```

Reimplemented in [qevercloud::EvernoteException](#), [qevercloud::NetworkException](#), [qevercloud::ThriftException](#), [qevercloud::EDAMSystemExceptionRateLimitReached](#), [qevercloud::EDAMSystemExceptionAuthExpired](#), [qevercloud::EDAMUserException](#), [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 returned 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<EDAMNotFoundException>(
                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
```

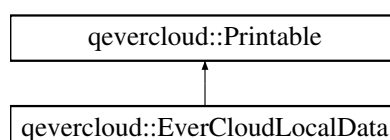
Contains an error message. It's the `std::exception::what()` counterpart.

## 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 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](#) = 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

- [bool local](#)
- [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 bool

```
qevercloud::EverCloudLocalData::bool = 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

### 7.32.5.2 dict

```
QHash<QString, QVariant> qevercloud::EverCloudLocalData::dict
```

dict can be used for storage of any other auxiliary values associated with objects of QEverCloud types

### 7.32.5.3 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

### 7.32.5.4 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

### 7.32.5.5 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

## 7.32.6 Property Documentation

### 7.32.6.1 dict

`Dict` qevercloud::EverCloudLocalData::dict

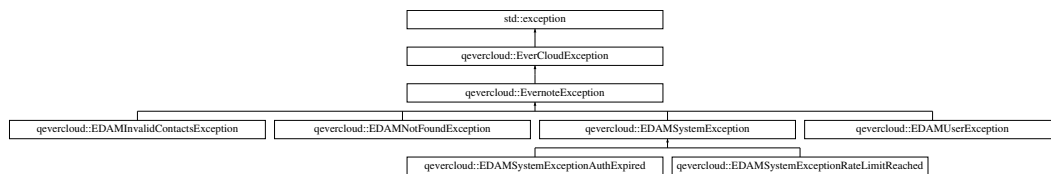
### 7.32.6.2 local

`bool` qevercloud::EverCloudLocalData::local

## 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

### Additional Inherited Members

#### 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]
```

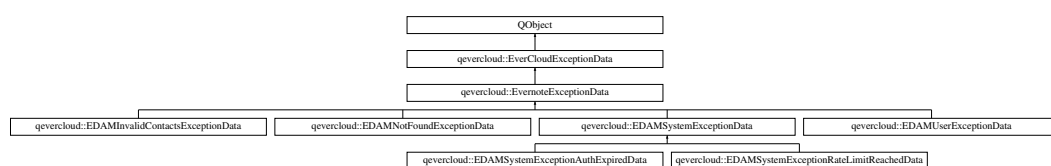
Reimplemented from [qevercloud::EverCloudException](#).

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

## Additional Inherited Members

### 7.34.1 Detailed Description

Asynchronous API counterpart 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](#).

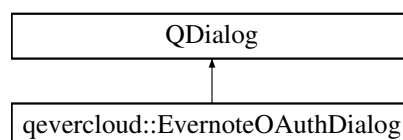
Reimplemented in [qevercloud::EDAMUserExceptionData](#), [qevercloud::EDAMSystemExceptionData](#), [qevercloud::EDAMNotFoundEx](#), [qevercloud::EDAMInvalidContactsExceptionData](#), [qevercloud::EDAMSystemExceptionRateLimitReachedData](#), and [qevercloud::EDAMSystemExceptionAuthExpiredData](#).

## 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 `qrand()` 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

<i>host</i>	Evernote host to authorize with. You need one of this: <ul style="list-style-type: none"><li>• "www.evernote.com" - the production service. It's the default value.</li><li>• "sandbox.evernote.com" - the developers "sandbox" service</li></ul>
<i>consumerKey</i>	get it <a href="#">from the Evernote</a>
<i>consumerSecret</i>	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 authentication 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 preferred size. Use this method to set the size.

##### Parameters

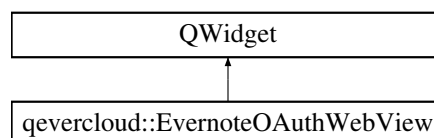
<i>sizeHint</i>	will be used as the preferred size of the contained QWebView.
-----------------	---

## 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 [authenticationSucceeded](#) ()
- void [authenticationFailed](#) ()

## Public Member Functions

- [EvernoteOAuthWebView](#) (QWidget \*parent=Q\_NULLPTR)
- void [authenticate](#) (QString host, QString consumerKey, QString consumerSecret, const quint64 timeout←Msec=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 quint64 timeoutMsec = 30000 )
```

This function starts the OAuth sequence. In the end of the sequence will be emitted one of the signals←: `authenticationSucceeded` or `authenticationFailed`.

Do not call the function while its call is in effect, i.e. one of the signals is not emitted.

## Parameters

<i>host</i>	Evernote host to authorize with. You need one of this: <ul style="list-style-type: none"><li>• "www.evernote.com" - the production service. It's the default value.</li><li>• "sandbox.evernote.com" - the developers "sandbox" service</li></ul>
<i>consumerKey</i>	get it <a href="#">from the Evernote</a>
<i>consumerSecret</i>	along with this
<i>timeoutMsec</i>	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 authenticationSucceeded**

```
void qevercloud::EvernoteOAuthWebView::authenticationSucceeded ( ) [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.

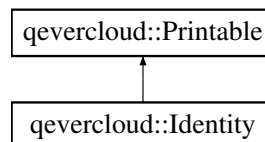
### 7.36.3.9 sizeHint()

```
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 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

<i>host</i>	www.evernote.com or sandbox.evernote.com
<i>shardId</i>	You can get the value from UserStore service or as a result of an authentication.
<i>authenticationToken</i>	For working private ink notes you must supply a valid authentication token. For public resources the value specified is not used.
<i>width</i>	Width of the ink note's resource
<i>height</i>	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

<i>guid</i>	The guid of the ink note's resource
<i>isPublic</i>	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.
<i>timeoutMsec</i>	Timeout for download request in milliseconds

#### Returns

downloaded data.

### 7.40.3.2 setAuthenticationToken()

```
InkNoteImageDownloader & qevercloud::InkNoteImageDownloader::setAuthenticationToken (
    QString authenticationToken )
```

#### Parameters

<i>authenticationToken</i>	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

<i>height</i>	Height of the ink note's resource
---------------	-----------------------------------

### 7.40.3.4 setHost()

```
InkNoteImageDownloader & qevercloud::InkNoteImageDownloader::setHost (
    QString host )
```

#### Parameters

<i>host</i>	www.evernote.com or sandbox.evernote.com
-------------	--

### 7.40.3.5 setShardId()

```
InkNoteImageDownloader & qevercloud::InkNoteImageDownloader::setShardId (
    QString shardId )
```

#### Parameters

<i>shardId</i>	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 )
```

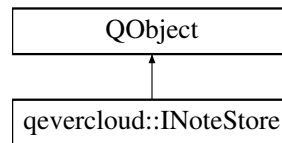
#### Parameters

<i>width</i>	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

<i>guid</i>	The GUID identifying this <a href="#">Note</a> on this shard.
<i>noteKey</i>	The 'noteKey' identifier from the <a href="#">Note</a> that was originally created via a call to <a href="#">shareNote()</a> and then given to a recipient to access.
<i>authenticationToken</i>	An optional authenticationToken that identifies the user accessing the shared note. This parameter may be required to access some shared notes.

## Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• PERMISSION_DENIED "Note" - the <a href="#">Note</a> with that GUID is either not shared, or the noteKey doesn't match the current key for this note</li> <li>• PERMISSION_DENIED "authenticationToken" - an authentication token is required to access this <a href="#">Note</a>, but either no authentication token or a "non-owner" authentication token was provided.</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>• "guid" - the note with that GUID is not found</li> </ul>
<a href="#">EDAMSystemException</a>	<ul style="list-style-type: none"> <li>• TAKEN_DOWN "Note" - The specified shared note is taken down (for all requesters).</li> <li>• TAKEN_DOWN "Country" - The specified shared note is taken down for the requester because of an IP-based country lookup.</li> </ul>

## 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

<i>shareKeyOr↔ GlobalId</i>	<p>May be one of the following:</p> <ul style="list-style-type: none"> <li>• 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 <code>createOrUpdateNotebookShares</code>. 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.</li> <li>• The shared notebook global identifier. May be used to access a notebook that is already joined.</li> <li>• The <a href="#">Notebook</a> GUID. May be used to access a notebook that was already joined, or to access a notebook that was shared with the recipient via <code>createOrUpdateNotebookShares</code>.</li> </ul>
<i>authenticationToken</i>	<p>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.</p>

## Exceptions

<i><a href="#">EDAMSystemException</a></i>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "shareKey" - invalid shareKey string</li> <li>• INVALID_AUTH "shareKey" - bad signature on shareKey string</li> </ul>
<i><a href="#">EDAMNotFoundException</a></i>	<ul style="list-style-type: none"> <li>• "SharedNotebook.id" - the shared notebook no longer exists</li> </ul>
<i><a href="#">EDAMUserException</a></i>	<ul style="list-style-type: none"> <li>• DATA_REQUIRED "authenticationToken" - the share requires login, and no valid authentication token was provided.</li> <li>• PERMISSION_DENIED "SharedNotebook.username" - share requires login, and another username has already been bound to this notebook.</li> </ul>

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

<i>noteGuid</i>	The GUID of the <a href="#">Note</a> to copy.
<i>toNotebookGuid</i>	The GUID of the <a href="#">Notebook</a> that should receive the new <a href="#">Note</a> .

#### 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

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>LIMIT_REACHED "Note" - at max number per account</li> <li>PERMISSION_DENIED "Notebook.guid" - destination not owned by user</li> <li>PERMISSION_DENIED "Note" - user doesn't own</li> <li>QUOTA_REACHED "Accounting.uploadLimit" - note exceeds upload quota</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>"Notebook.guid" - not found, by GUID</li> </ul>

#### 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

<i>linkedNotebook</i>	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 <a href="#">EDAMUserException</a> is thrown.
-----------------------	--

## Returns

The newly created [LinkedNotebook](#). The server-side id will be saved in this object's 'id' field.

## Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• DATA_REQUIRED "LinkedNotebook.shareName" - missing shareName</li> <li>• BAD_DATA_FORMAT "LinkedNotebook.name" - invalid shareName length or pattern</li> <li>• BAD_DATA_FORMAT "LinkedNotebook.username" - bad username format</li> <li>• BAD_DATA_FORMAT "LinkedNotebook.uri" - if public notebook set but bad uri</li> <li>• DATA_REQUIRED "LinkedNotebook.shardId" - if private notebook but shard id not provided</li> <li>• BAD_DATA_FORMAT "LinkedNotebook.stack" - invalid stack name length or pattern</li> </ul>
<a href="#">EDAMSystemException</a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "LinkedNotebook.sharedNotebookGlobalId" - if a bad global identifier was set on a private notebook</li> </ul>

## 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

<i>note</i>	A <a href="#">Note</a> 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

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "Note.title" - invalid length or pattern</li> <li>• BAD_DATA_FORMAT "Note.content" - invalid length for ENML content</li> <li>• BAD_DATA_FORMAT "Resource.mime" - invalid resource MIME type</li> <li>• BAD_DATA_FORMAT "NoteAttributes.*" - bad resource string</li> <li>• BAD_DATA_FORMAT "ResourceAttributes.*" - bad resource string</li> <li>• DATA_CONFLICT "Note.deleted" - deleted time set on active note</li> <li>• DATA_REQUIRED "Resource.data" - resource data body missing</li> <li>• ENML_VALIDATION "*" - note content doesn't validate against DTD</li> <li>• LIMIT_REACHED "Note" - at max number per account</li> <li>• LIMIT_REACHED "Note.size" - total note size too large</li> <li>• LIMIT_REACHED "Note.resources" - too many resources on <a href="#">Note</a></li> <li>• LIMIT_REACHED "Note.tagGuids" - too many Tags on <a href="#">Note</a></li> <li>• LIMIT_REACHED "Resource.data.size" - resource too large</li> <li>• LIMIT_REACHED "NoteAttribute.*" - attribute string too long</li> <li>• LIMIT_REACHED "ResourceAttribute.*" - attribute string too long</li> <li>• PERMISSION_DENIED "Note.notebookGuid" - NB not owned by user</li> <li>• QUOTA_REACHED "Accounting.uploadLimit" - note exceeds upload quota</li> <li>• BAD_DATA_FORMAT "Tag.name" - <a href="#">Note.tagNames</a> was provided, and one of the specified tags had an invalid length or pattern</li> <li>• LIMIT_REACHED "Tag" - <a href="#">Note.tagNames</a> was provided, and the required new tags would exceed the maximum number per account</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>• "Note.notebookGuid" - not found, by GUID</li> </ul>

### 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

<i>notebook</i>	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 <a href="#">EDAMUserException</a> .
-----------------	--

#### Returns

The newly created [Notebook](#). The server-side GUID will be saved in this object's 'guid' field.

#### Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "Notebook.name" - invalid length or pattern</li> <li>• BAD_DATA_FORMAT "Notebook.stack" - invalid length or pattern</li> <li>• BAD_DATA_FORMAT "Publishing.uri" - if publishing set but bad uri</li> <li>• BAD_DATA_FORMAT "Publishing.publicDescription" - if too long</li> <li>• DATA_CONFLICT "Notebook.name" - name already in use</li> <li>• DATA_CONFLICT "Publishing.uri" - if URI already in use</li> <li>• DATA_REQUIRED "Publishing.uri" - if publishing set but uri missing</li> <li>• DATA_REQUIRED "Notebook" - notebook parameter was null</li> <li>• PERMISSION_DENIED "Notebook.defaultNotebook" - if the 'defaultNotebook' field is set to 'true' for a <a href="#">Notebook</a> that is not owned by the user identified by the passed authenticationToken.</li> <li>• LIMIT_REACHED "Notebook" - at max number of notebooks</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>• "Workspace.guid" - if workspaceGuid set and no Workspace exists for the GUID</li> </ul>

**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**

<i>authenticationToken</i>	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 <code>NoteStore.authenticateToNotebook</code> .
<i>shareTemplate</i>	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 Shared↵  
Notebooks.

## Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• DATA_REQUIRED "Notebook.guid" - if no notebook GUID was specified</li> <li>• BAD_DATA_FORMAT "Notebook.guid" - if shareTemplate.notebookGuid is not a valid GUID</li> <li>• DATA_REQUIRED "shareTemplate" - if the shareTemplate parameter was missing</li> <li>• DATA_REQUIRED "NotebookShareTemplate.privilege" - if no privilege was specified</li> <li>• DATA_CONFLICT "NotebookShareTemplate.privilege" - if the specified privilege is not allowed.</li> <li>• DATA_REQUIRED "NotebookShareTemplate.recipients" - if no recipients were specified, either by thread ID or as a list of contacts</li> <li>• LIMIT_REACHED "SharedNotebook" - if the notebook has reached its maximum number of shares</li> </ul>
<a href="#"><i>EDAMInvalidContactsException</i></a>	<ul style="list-style-type: none"> <li>• "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.</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• "Notebook.guid" - if no notebook with the specified GUID was found</li> <li>• "NotebookShareTemplate.recipientThreadId" - if the recipient thread ID was specified, but no thread with that ID exists</li> </ul>

**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

<i>search</i>	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 <a href="#">SavedSearch.format</a> field is ignored by the service.
---------------	---

## Returns

The newly created [SavedSearch](#). The server-side GUID will be saved in this object.

## Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "SavedSearch.name" - invalid length or pattern</li> <li>• BAD_DATA_FORMAT "SavedSearch.query" - invalid length</li> <li>• DATA_CONFLICT "SavedSearch.name" - name already in use</li> <li>• LIMIT_REACHED "SavedSearch" - at max number of searches</li> </ul>
-----------------------------------	---

## 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

<i>tag</i>	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

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "Tag.name" - invalid length or pattern</li> <li>• BAD_DATA_FORMAT "Tag.parentGuid" - malformed GUID</li> <li>• DATA_CONFLICT "Tag.name" - name already in use</li> <li>• LIMIT_REACHED "Tag" - at max number of tags</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• "Tag.parentGuid" - not found, by GUID</li> </ul>

**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

<i>guid</i>	The GUID of the note to delete.
-------------	---------------------------------

## Returns

The Update Sequence Number for this change within the account.

## Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• PERMISSION_DENIED "Note" - user doesn't have permission to update the note.</li> </ul>
<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• DATA_CONFLICT "Note.guid" - the note is already deleted</li> </ul>

## Exceptions

<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"><li>• "Note.guid" - not found, by GUID</li></ul>
---------------------------------------	--

**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

<i>authenticationToken</i>	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.
<i>parameters</i>	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 <a href="#">Note</a> must be passed to this call. This also specifies the additional email fields that will be used in the email.

## Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• LIMIT_REACHED "NoteEmailParameters.toAddresses" - The email can't be sent because this would exceed the user's daily email limit.</li> <li>• BAD_DATA_FORMAT "(email address)" - email address malformed</li> <li>• DATA_REQUIRED "NoteEmailParameters.toAddresses" - if there are no To: or Cc: addresses provided.</li> <li>• DATA_REQUIRED "Note.title" - if the caller provides a <a href="#">Note</a> parameter with no title</li> <li>• DATA_REQUIRED "Note.content" - if the caller provides a <a href="#">Note</a> parameter with no content</li> <li>• ENML_VALIDATION "*" - note content doesn't validate against DTD</li> <li>• DATA_REQUIRED "NoteEmailParameters.note" - if no guid or note provided</li> <li>• PERMISSION_DENIED "Note" - private note, user doesn't own</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• "Note.guid" - not found, by GUID</li> </ul>

## 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

<i>guid</i>	The <a href="#">LinkedNotebook.guid</a> field of the <a href="#">LinkedNotebook</a> 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

<i>guid</i>	The GUID of the note to delete.
-------------	---------------------------------

#### Returns

The Update Sequence Number for this change within the account.

#### Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"><li>PERMISSION_DENIED "Note" - user doesn't own</li></ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"><li>"Note.guid" - not found, by GUID</li></ul>

### 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

<i>guid</i>	The GUID of the notebook to delete.
-------------	-------------------------------------

#### Returns

The Update Sequence Number for this change within the account.

#### Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "Notebook.guid" - if the parameter is missing</li> <li>• LIMIT_REACHED "Notebook" - trying to expunge the last <a href="#">Notebook</a></li> <li>• PERMISSION_DENIED "Notebook" - private notebook, user doesn't own</li> </ul>
-----------------------------------	--

### 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

<i>guid</i>	The GUID of the search to delete.
-------------	-----------------------------------

## Returns

The Update Sequence Number for this change within the account.

## Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"><li>• BAD_DATA_FORMAT "SavedSearch.guid" - if the guid parameter is empty</li><li>• PERMISSION_DENIED "SavedSearch" - user doesn't own</li></ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"><li>• "SavedSearch.guid" - not found, by GUID</li></ul>

### 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

<i>guid</i>	The GUID of the tag to delete.
-------------	--------------------------------

## Returns

The Update Sequence Number for this change within the account.

## Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "Tag.guid" - if the guid parameter is missing</li> <li>• PERMISSION_DENIED "Tag" - user doesn't own tag</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• "Tag.guid" - tag not found, by GUID</li> </ul>

**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

<i>authenticationToken</i>	Must be a valid token for the user's account unless the <a href="#">NoteFilter</a> 'notebookGuid' is the GUID of a public notebook.
<i>filter</i>	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.
<i>withTrash</i>	If true, then the <a href="#">NoteCollectionCounts.trashCount</a> will be calculated and supplied in the reply. Otherwise, the trash value will be omitted.

## Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "NoteFilter.notebookGuid" - if malformed</li> <li>• BAD_DATA_FORMAT "NoteFilter.notebookGuids" - if any are malformed</li> <li>• BAD_DATA_FORMAT "NoteFilter.words" - if search string too long</li> </ul>
--	---

## Exceptions

<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"><li>• "Notebook.guid" - not found, by GUID</li></ul>
---------------------------------------	--

### 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

<i>authenticationToken</i>	Must be a valid token for the user's account unless the <a href="#">NoteFilter</a> 'notebookGuid' is the GUID of a public notebook.
<i>filter</i>	The list of criteria that will constrain the notes to be returned.
<i>guid</i>	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

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "offset" - not between 0 and EDAM_USER_NOTES_MAX</li> <li>• BAD_DATA_FORMAT "maxNotes" - not between 0 and EDAM_USER_NOTES_MAX</li> <li>• BAD_DATA_FORMAT "NoteFilter.notebookGuid" - if malformed</li> <li>• BAD_DATA_FORMAT "NoteFilter.tagGuids" - if any are malformed</li> <li>• BAD_DATA_FORMAT "NoteFilter.words" - if search string too long</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• "Notebook.guid" - not found, by GUID</li> <li>• "Note.guid" - not found, by GUID</li> </ul>

## 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](http://dev.evernote.com/documentation/cloud/chapters/polling_notification.php) for more information.

## Parameters

<i>authenticationToken</i>	Must be a valid token for the user's account unless the <a href="#">NoteFilter</a> 'notebookGuid' is the GUID of a public notebook.
<i>filter</i>	The list of criteria that will constrain the notes to be returned.
<i>offset</i>	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.

## Parameters

<i>maxNotes</i>	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 <a href="#">NoteList.totalNotes</a> 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.
<i>resultSpec</i>	This specifies which information should be returned for each matching <a href="#">Note</a> . 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

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "offset" - not between 0 and EDAM_USER_NOTES_MAX</li> <li>• BAD_DATA_FORMAT "maxNotes" - not between 0 and EDAM_USER_NOTES_MAX</li> <li>• BAD_DATA_FORMAT "NoteFilter.notebookGuid" - if malformed</li> <li>• BAD_DATA_FORMAT "NoteFilter.tagGuids" - if any are malformed</li> <li>• BAD_DATA_FORMAT "NoteFilter.words" - if search string too long</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>• "Notebook.guid" - not found, by GUID</li> </ul>

## 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

<i>query</i>	The information about which we are finding related entities.
<i>resultSpec</i>	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

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "RelatedQuery.plainText" - If you provided a a zero-length plain text value.</li> <li>• BAD_DATA_FORMAT "RelatedQuery.noteGuid" - If you provided an invalid <a href="#">Note</a> GUID, that is, one that does not match the constraints defined by EDAM_GUID_LEN_MIN, EDAM_GUID_LEN_MAX, EDAM_GUID_REGEX.</li> <li>• BAD_DATA_FORMAT "NoteFilter.notebookGuid" - if malformed</li> <li>• BAD_DATA_FORMAT "NoteFilter.tagGuids" - if any are malformed</li> <li>• BAD_DATA_FORMAT "NoteFilter.words" - if search string too long</li> <li>• PERMISSION_DENIED "Note" - If the caller does not have access to the note identified by <a href="#">RelatedQuery.noteGuid</a>.</li> <li>• 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).</li> <li>• DATA_REQUIRED "RelatedResultSpec" - If you did not not set any values in the result spec.</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• "RelatedQuery.noteGuid" - the note with that GUID is not found, if that field has been set in the query.</li> </ul>

## 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**

<i>afterUSN</i>	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.
<i>maxEntries</i>	The maximum number of modified objects that should be returned in the result <a href="#">SyncChunk</a> . This can be used to limit the size of each individual message to be friendly for network transfer.
<i>filter</i>	The caller must set some of the flags in this structure to specify which data types should be returned during the synchronization. See the <a href="#">SyncChunkFilter</a> structure for information on each flag.

**Exceptions**

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>BAD_DATA_FORMAT "afterUSN" - if negative</li> <li>BAD_DATA_FORMAT "maxEntries" - if less than 1</li> </ul>
-----------------------------------	---

### 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/shardId/edam/note must be the same as [LinkedNotebook.shardId](#).)

#### Parameters

<i>authenticationToken</i>	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 <a href="#">NoteStore.authenticateToSharedNotebook</a> )
<i>linkedNotebook</i>	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)
<i>afterUSN</i>	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.
<i>maxEntries</i>	The maximum number of modified objects that should be returned in the result <a href="#">SyncChunk</a> . 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.
<i>fullSyncOnly</i>	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

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "afterUSN" - if negative</li> <li>• BAD_DATA_FORMAT "maxEntries" - if less than 1</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• "LinkedNotebook" - if the provided information doesn't match any valid notebook</li> <li>• "LinkedNotebook.uri" - if the provided public URI doesn't match any valid notebook</li> <li>• "SharedNotebook.id" - if the provided information indicates a shared notebook that no longer exists</li> </ul>

## 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/shardId/edam/note` must be the same as [LinkedNotebook.shardId](#).)

## Parameters

<i>authenticationToken</i>	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 <code>NoteStore.authenticateToSharedNotebook</code> )
<i>linkedNotebook</i>	This structure should contain identifying information and permissions to access the notebook in question.

## Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>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.</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>"LinkedNotebook.username" - If the <a href="#">LinkedNotebook.username</a> field does not correspond to a current user on the service.</li> </ul>
<a href="#"><i>SystemException</i></a>	<ul style="list-style-type: none"> <li>SHARD_UNAVAILABLE - If the provided <a href="#">LinkedNotebook.username</a> corresponds to a user whose account is on a shard other than that on which this method was invoked.</li> </ul>

**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**

<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>• "Note.guid" - note not found, by GUID</li> <li>• "NoteAttributes.applicationData.key" - note not found, by key</li> </ul>
---------------------------------------	--

**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

<i>guid</i>	The GUID of the notebook to be retrieved.
-------------	---

## Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>BAD_DATA_FORMAT "Notebook.guid" - if the parameter is missing</li> <li>PERMISSION_DENIED "Notebook" - private notebook, user doesn't own</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>"Notebook.guid" - tag not found, by GUID</li> </ul>

**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

<i>guid</i>	The GUID of the note to be retrieved.
-------------	---------------------------------------

## Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>BAD_DATA_FORMAT "Note.guid" - if the parameter is missing</li> <li>PERMISSION_DENIED "Note" - private note, user doesn't own</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>"Note.guid" - not found, by GUID</li> </ul>

**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

<i>guid</i>	The GUID of the note to be retrieved.
<i>noteOnly</i>	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)
<i>tokenizeForIndexing</i>	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

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "Note.guid" - if the parameter is missing</li> <li>• PERMISSION_DENIED "Note" - private note, user doesn't own</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• "Note.guid" - not found, by GUID</li> </ul>

**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

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "Note.guid" - if the parameter is missing</li> <li>• PERMISSION_DENIED "Note" - private note, user doesn't own</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• "Note.guid" - not found, by GUID</li> </ul>

**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

<i>noteGuid</i>	The GUID of the note to be retrieved.
<i>updateSequenceNum</i>	The USN of the version of the note that is being retrieved
<i>withResourcesData</i>	If true, any <a href="#">Resource</a> elements in this <a href="#">Note</a> will include the binary contents of their 'data' field's body.
<i>withResourcesRecognition</i>	If true, any <a href="#">Resource</a> elements will include the binary contents of the 'recognition' field's body if recognition data is present.
<i>withResourcesAlternateData</i>	If true, any <a href="#">Resource</a> elements in this <a href="#">Note</a> will include the binary contents of their 'alternateData' fields' body, if an alternate form is present.

#### Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• DATA_REQUIRED "Note.guid" - if GUID is null or empty string.</li> <li>• BAD_DATA_FORMAT "Note.guid" - if GUID is not of correct length.</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>• "Note.guid" - not found, by GUID.</li> <li>• "Note.updateSequenceNumber" - the <a href="#">Note</a> doesn't have a version with the corresponding USN.</li> </ul>

### 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

<i>authenticationToken</i>	An authentication token that grants the caller access to the requested note.
<i>guid</i>	The GUID of the note to be retrieved.
<i>resultSpec</i>	A structure specifying the fields of the note that the caller would like to get.

##### Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>BAD_DATA_FORMAT "Note.guid" - if the parameter is missing</li> <li>PERMISSION_DENIED "Note" - private note, user doesn't own</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>"Note.guid" - not found, by GUID</li> </ul>

#### 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

<i>userId</i>	The numeric identifier for the user who owns the public notebook. To find this value based on a username string, you can invoke <code>UserStore.getPublicUserInfo</code>
<i>publicUri</i>	The uri string for the public notebook, from <code>Notebook.publishing.uri</code> .

#### Exceptions

<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>"Publishing.uri" - not found, by URI</li> </ul>
<a href="#">EDAMSystemException</a>	<ul style="list-style-type: none"> <li>TAKEN_DOWN "PublicNotebook" - The specified public notebook is taken down (for all requesters).</li> <li>TAKEN_DOWN "Country" - The specified public notebook is taken down for the requester because of an IP-based country lookup.</li> </ul>

#### 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

<i>guid</i>	The GUID of the resource to be retrieved.
<i>withData</i>	If true, the <a href="#">Resource</a> will include the binary contents of the 'data' field's body.
<i>withRecognition</i>	If true, the <a href="#">Resource</a> will include the binary contents of the 'recognition' field's body if recognition data is present.
<i>withAttributes</i>	If true, the <a href="#">Resource</a> will include the attributes
<i>withAlternateData</i>	If true, the <a href="#">Resource</a> will include the binary contents of the 'alternateData' field's body, if an alternate form is present.

## Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "Resource.guid" - if the parameter is missing</li> <li>• PERMISSION_DENIED "Resource" - private resource, user doesn't own</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>• "Resource.guid" - not found, by GUID</li> </ul>

## 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

<i>guid</i>	The GUID of the resource whose recognition data should be retrieved.
-------------	--

## Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "Resource.guid" - if the parameter is missing</li> <li>• PERMISSION_DENIED "Resource" - private resource, user doesn't own</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>• "Resource.guid" - not found, by GUID</li> <li>• "Resource.alternateData" - resource has no recognition</li> </ul>

**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**

<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>• "Resource.guid" - <a href="#">Resource</a> not found, by GUID</li> <li>• "ResourceAttributes.applicationData.key" - <a href="#">Resource</a> not found, by key</li> </ul>
---------------------------------------	--

**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

<i>guid</i>	The GUID of the resource whose attributes should be retrieved.
-------------	--

##### Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>BAD_DATA_FORMAT "Resource.guid" - if the parameter is missing</li> <li>PERMISSION_DENIED "Resource" - private resource, user doesn't own</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>"Resource.guid" - not found, by GUID</li> </ul>

#### 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

<i>noteGuid</i>	The GUID of the note that holds the resource to be retrieved.
<i>contentHash</i>	The MD5 checksum of the resource within that note. <a href="#">Note</a> 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.
<i>withData</i>	If true, the <a href="#">Resource</a> will include the binary contents of the 'data' field's body.
<i>withRecognition</i>	If true, the <a href="#">Resource</a> will include the binary contents of the 'recognition' field's body.
<i>withAlternateData</i>	If true, the <a href="#">Resource</a> will include the binary contents of the 'alternateData' field's body, if an alternate form is present.

#### Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• DATA_REQUIRED "Note.guid" - noteGuid param missing</li> <li>• DATA_REQUIRED "Note.contentHash" - contentHash param missing</li> <li>• PERMISSION_DENIED "Resource" - private resource, user doesn't own</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>• "Note" - not found, by guid</li> <li>• "Resource" - not found, by hash</li> </ul>

### 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

<i>guid</i>	The GUID of the resource to be retrieved.
-------------	---

#### Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"><li>• BAD_DATA_FORMAT "Resource.guid" - if the parameter is missing</li><li>• PERMISSION_DENIED "Resource" - private resource, user doesn't own</li></ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"><li>• "Resource.guid" - not found, by GUID</li></ul>

### 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

<i>guid</i>	The GUID of the resource whose recognition data should be retrieved.
-------------	--

## Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "Resource.guid" - if the parameter is missing</li> <li>• PERMISSION_DENIED "Resource" - private resource, user doesn't own</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• "Resource.guid" - not found, by GUID</li> <li>• "Resource.recognition" - resource has no recognition</li> </ul>

**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

<i>guid</i>	The GUID of the resource to be retrieved.
-------------	---

## Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "Resource.guid" - if the parameter is missing</li> <li>• PERMISSION_DENIED "Resource" - private resource, user doesn't own</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• "Resource.guid" - not found, by GUID</li> </ul>

**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**

<i>guid</i>	The GUID of the search to be retrieved.
-------------	---

**Exceptions**

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>BAD_DATA_FORMAT "SavedSearch.guid" - if the parameter is missing</li> <li>PERMISSION_DENIED "SavedSearch" - private <a href="#">Tag</a>, user doesn't own</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>"SavedSearch.guid" - not found, by GUID</li> </ul>

**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 result 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

<i>authenticationToken</i>	Should be the authentication token retrieved from the reply of <a href="#">authenticateToSharedNotebook()</a> , proving access to a particular shared notebook.
----------------------------	---

## Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• PERMISSION_DENIED "authenticationToken" - authentication token doesn't correspond to a valid shared notebook</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>• "SharedNotebook.id" - the shared notebook no longer exists</li> </ul>

**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

<i>guid</i>	The GUID of the tag to be retrieved.
-------------	--------------------------------------

## Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>BAD_DATA_FORMAT "Tag.guid" - if the parameter is missing</li> <li>PERMISSION_DENIED "Tag" - private <a href="#">Tag</a>, user doesn't own</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>"Tag.guid" - tag not found, by GUID</li> </ul>

## 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

<i>authenticationToken</i>	A business authentication token obtained by calling <code>UserStore.authenticateToBusiness</code> .
----------------------------	---

## Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>INVALID_AUTH "authenticationToken" - if the authentication token is not a business auth token.</li> </ul>
--	--

### 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

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• DATA_REQUIRED "Note.guid" - if GUID is null or empty string.</li> <li>• BAD_DATA_FORMAT "Note.guid" - if GUID is not of correct length.</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• "Note.guid" - not found, by GUID.</li> </ul>

**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**

<i>notebookGuid</i>	the GUID of the notebook to use to find tags
---------------------	--

**Exceptions**

<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>"Notebook.guid" - notebook not found by GUID</li> </ul>
---------------------------------------	--

**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

<i>parameters</i>	A structure containing all parameters for the updates. See the structure documentation for details.
-------------------	---

#### Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"><li>• <a href="#">EDAMErrorCode.LIMIT_REACHED</a> "SharedNotebook" - Trying to share a notebook while the notebook already has EDAM_NOTEBOOK_SHARED_NOTEBOOK_MAX shares.</li></ul>
-----------------------------------	--

### 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**

<i>authenticationToken</i>	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**

<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li><a href="#">Notebook.guid</a> - Thrown if the service does not have a notebook record with the notebookGuid on the given shard.</li> <li>Publishing.publishState - Thrown if the business notebook is not shared with the user and is also not published to their business.</li> </ul>
---------------------------------------	---

## Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• PERMISSION_DENIED "authenticationToken" - If the owner of the given token is not allowed to set recipient settings on the specified notebook.</li> <li>• DATA_CONFLICT "recipientSettings.reminderNotifyEmail" - Setting reminderNotifyEmail is allowed only for notebooks which belong to the same business as the user.</li> <li>• DATA_CONFLICT "recipientSettings.inMyList" - If the request is setting inMyList to false and any of reminder* settings to true.</li> </ul>
-----------------------------------	--

**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**

<i>guid</i>	The GUID of the note to be shared.
-------------	------------------------------------

**Exceptions**

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "Note.guid" - if the parameter is missing</li> <li>• PERMISSION_DENIED "Note" - private note, user doesn't own</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>• "Note.guid" - not found, by GUID</li> </ul>

**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 `sharedNotebook` 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

<i>authenticationToken</i>	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.
<i>sharedNotebook</i>	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.
<i>message</i>	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

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "SharedNotebook.email" - if the email was not valid</li> <li>• DATA_REQUIRED "SharedNotebook.privilege" - if the SharedNotebook.privilegeLevel was not set.</li> <li>• BAD_DATA_FORMAT "SharedNotebook.requireLogin" - if requireLogin was set. requireLogin is deprecated.</li> <li>• BAD_DATA_FORMAT "SharedNotebook.privilegeLevel" - if the SharedNotebook.privilegeLevel field was unset or set to GROUP.</li> <li>• PERMISSION_DENIED "user" - if the email address on the authenticationToken's owner's account is not confirmed.</li> <li>• PERMISSION_DENIED "SharedNotebook.recipientSettings" - if recipientSettings is set in the sharedNotebook. Only the recipient can set these values via the setSharedNotebookRecipientSettings method.</li> <li>• <a href="#"><i>EDAMErrorCode.LIMIT_REACHED</i></a> "SharedNotebook" - The notebook already has EDAM_NOTEBOOK_SHARED_NOTEBOOK_MAX shares.</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• <a href="#"><i>Notebook.guid</i></a> - if the notebookGuid is not a valid GUID for the user.</li> </ul>

**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 [\*stopSharingNoteWithRecipients\*](#).

## Parameters

<i>guid</i>	The GUID of the note to be un-shared.
-------------	---------------------------------------

## Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>BAD_DATA_FORMAT "Note.guid" - if the parameter is missing</li> <li>PERMISSION_DENIED "Note" - private note, user doesn't own</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>"Note.guid" - not found, by GUID</li> </ul>

**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 intended 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**

<i>guid</i>	The GUID of the tag to remove from all notes.
-------------	---

**Exceptions**

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>BAD_DATA_FORMAT "Tag.guid" - if the guid parameter is missing</li> <li>PERMISSION_DENIED "Tag" - user doesn't own tag</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>"Tag.guid" - tag not found, by GUID</li> </ul>

**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

<i>linkedNotebook</i>	Updates the name of a linked notebook.
-----------------------	--

## Returns

The Update Sequence Number for this change within the account.

## Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• DATA_REQUIRED "LinkedNotebook.shareName" - missing shareName</li> <li>• BAD_DATA_FORMAT "LinkedNotebook.shareName" - invalid shareName length or pattern</li> <li>• BAD_DATA_FORMAT "LinkedNotebook.stack" - invalid stack name length or pattern</li> </ul>
-----------------------------------	---

## 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

<i>note</i>	A <a href="#">Note</a> 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

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "Note.title" - invalid length or pattern</li> <li>• BAD_DATA_FORMAT "Note.content" - invalid length for ENML body</li> <li>• BAD_DATA_FORMAT "NoteAttributes.*" - bad resource string</li> <li>• BAD_DATA_FORMAT "ResourceAttributes.*" - bad resource string</li> <li>• BAD_DATA_FORMAT "Resource.mime" - invalid resource MIME type</li> <li>• DATA_CONFLICT "Note.deleted" - deleted time set on active note</li> <li>• DATA_REQUIRED "Resource.data" - resource data body missing</li> <li>• ENML_VALIDATION "*" - note content doesn't validate against DTD</li> <li>• LIMIT_REACHED "Note.tagGuids" - too many Tags on <a href="#">Note</a></li> <li>• LIMIT_REACHED "Note.resources" - too many resources on <a href="#">Note</a></li> <li>• LIMIT_REACHED "Note.size" - total note size too large</li> <li>• LIMIT_REACHED "Resource.data.size" - resource too large</li> <li>• LIMIT_REACHED "NoteAttribute.*" - attribute string too long</li> <li>• LIMIT_REACHED "ResourceAttribute.*" - attribute string too long</li> <li>• PERMISSION_DENIED "Note.notebookGuid" - user doesn't own destination</li> <li>• PERMISSION_DENIED "Note.tags" - user doesn't have permission to modify the note's tags. note.tags must be unset.</li> <li>• PERMISSION_DENIED "Note.attributes" - user doesn't have permission to modify the note's attributes. note.attributes must be unset.</li> <li>• QUOTA_REACHED "Accounting.uploadLimit" - note exceeds upload quota</li> <li>• BAD_DATA_FORMAT "Tag.name" - <a href="#">Note.tagNames</a> was provided, and one of the specified tags had an invalid length or pattern</li> <li>• LIMIT_REACHED "Tag" - <a href="#">Note.tagNames</a> was provided, and the required new tags would exceed the maximum number per account</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• "Note.guid" - note not found, by GUID</li> <li>• "Note.notebookGuid" - if notebookGuid provided, but not found</li> </ul>

## 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**

<i>notebook</i>	The notebook object containing the requested changes.
-----------------	---

**Returns**

The Update Sequence Number for this change within the account.

**Exceptions**

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "Notebook.name" - invalid length or pattern</li> <li>• BAD_DATA_FORMAT "Notebook.stack" - invalid length or pattern</li> <li>• BAD_DATA_FORMAT "Publishing.uri" - if publishing set but bad uri</li> <li>• BAD_DATA_FORMAT "Publishing.publicDescription" - if too long</li> <li>• DATA_CONFLICT "Notebook.name" - name already in use</li> <li>• DATA_CONFLICT "Publishing.uri" - if URI already in use</li> <li>• DATA_REQUIRED "Publishing.uri" - if publishing set but uri missing</li> <li>• DATA_REQUIRED "Notebook" - notebook parameter was null</li> <li>• PERMISSION_DENIED "Notebook.defaultNotebook" - if the 'defaultNotebook' field is set to 'true' for a <a href="#">Notebook</a> that is not owned by the user identified by the passed authenticationToken.</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>• "Notebook.guid" - not found, by GUID</li> <li>• "Workspace.guid" - if a non empty workspaceGuid set and no Workspace exists for the GUID</li> </ul>

**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

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• DATA_REQUIRED "Note.updateSequenceNum" - If the update sequence number was not provided. This includes a value that is set as 0.</li> <li>• 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.</li> </ul>
-----------------------------------	---

#### 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

<i>resource</i>	<p>A <a href="#">Resource</a> 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:</p> <ul style="list-style-type: none"> <li>• guid: must be provided to identify the resource</li> <li>• mime</li> <li>• width</li> <li>• height</li> <li>• duration</li> <li>• attributes: optional. if present, the set of attributes will be replaced.</li> </ul>
-----------------	---

## Returns

The Update Sequence Number of the resource after the changes have been applied.

## Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "Resource.guid" - if the parameter is missing</li> <li>• BAD_DATA_FORMAT "Resource.mime" - invalid resource MIME type</li> <li>• BAD_DATA_FORMAT "ResourceAttributes.*" - bad resource string</li> <li>• LIMIT_REACHED "ResourceAttribute.*" - attribute string too long</li> <li>• PERMISSION_DENIED "Resource" - private resource, user doesn't own</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>• "Resource.guid" - not found, by GUID</li> </ul>

## 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

<i>search</i>	The search object containing the requested changes.
---------------	---

## Returns

The Update Sequence Number for this change within the account.

## Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "SavedSearch.name" - invalid length or pattern</li> <li>• BAD_DATA_FORMAT "SavedSearch.query" - invalid length</li> <li>• DATA_CONFLICT "SavedSearch.name" - name already in use</li> <li>• PERMISSION_DENIED "SavedSearch" - user doesn't own tag</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• "SavedSearch.guid" - not found, by GUID</li> </ul>

**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

<i>tag</i>	The tag object containing the requested changes.
------------	--

## Returns

The Update Sequence Number for this change within the account.

## Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• BAD_DATA_FORMAT "Tag.name" - invalid length or pattern</li> <li>• BAD_DATA_FORMAT "Tag.parentGuid" - malformed GUID</li> <li>• DATA_CONFLICT "Tag.name" - name already in use</li> <li>• DATA_CONFLICT "Tag.parentGuid" - can't set parent: circular</li> <li>• PERMISSION_DENIED "Tag" - user doesn't own tag</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• "Tag.guid" - tag not found, by GUID</li> <li>• "Tag.parentGuid" - parent not found, by GUID</li> </ul>

## 7.41.3.150 updateTagAsync()

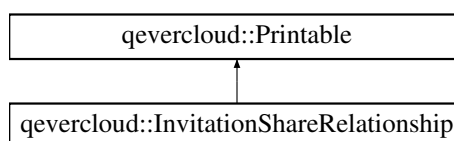
```
virtual AsyncResult * qevercloud::INoteStore::updateTagAsync (
    const Tag & tag,
    IRequestContextPtr ctx = {} ) [pure virtual]
```

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 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 Function 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

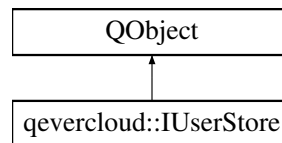
### 7.46.3.1 m\_iterator

```
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 deviceIdIdentifier, QString deviceDescription, bool supports↔TwoFactor, IRequestContextPtr ctx={})=0
- virtual [AsyncResult](#) \* [authenticateLongSessionAsync](#) (QString username, QString password, QString consumerKey, QString consumerSecret, QString deviceIdIdentifier, QString deviceDescription, bool supports↔TwoFactor, IRequestContextPtr ctx={})=0
- virtual [AuthenticationResult](#) [completeTwoFactorAuthentication](#) (QString oneTimeCode, QString device↔Identifier, QString deviceDescription, IRequestContextPtr ctx={})=0
- virtual [AsyncResult](#) \* [completeTwoFactorAuthenticationAsync](#) (QString oneTimeCode, QString device↔Identifier, 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 deviceIdIdentifier,
    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](https://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](https://dev.evernote.com).

#### Parameters

<i>username</i>	The username or registered email address of the account to authenticate against.
<i>password</i>	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).
<i>consumerKey</i>	The "consumer key" portion of the API key issued to the client application by Evernote.
<i>consumerSecret</i>	The "consumer secret" portion of the API key issued to the client application by Evernote.
<i>deviceIdIdentifier</i>	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.
<i>deviceDescription</i>	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.
<i>supportsTwoFactor</i>	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

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• DATA_REQUIRED "username" - username is empty</li> <li>• DATA_REQUIRED "password" - password is empty</li> <li>• DATA_REQUIRED "consumerKey" - consumerKey is empty</li> <li>• DATA_REQUIRED "consumerSecret" - consumerSecret is empty</li> <li>• DATA_REQUIRED "deviceDescription" - deviceDescription is empty</li> <li>• BAD_DATA_FORMAT "deviceDescription" - deviceDescription is not valid.</li> <li>• BAD_DATA_FORMAT "deviceIdIdentifier" - deviceIdIdentifier is not valid.</li> <li>• INVALID_AUTH "username" - username not found</li> <li>• INVALID_AUTH "password" - password did not match</li> <li>• INVALID_AUTH "consumerKey" - consumerKey is not authorized</li> <li>• INVALID_AUTH "consumerSecret" - consumerSecret is incorrect</li> <li>• INVALID_AUTH "businessOnly" - the user is a business-only account</li> <li>• PERMISSION_DENIED "User.active" - user account is closed</li> <li>• PERMISSION_DENIED "User.tooManyFailuresTryAgainLater" - user has failed authentication too often</li> <li>• AUTH_EXPIRED "password" - user password is expired</li> </ul>
--	--

## 7.47.3.2 authenticateLongSessionAsync()

```
virtual AsyncResult * qevercloud::IUserStore::authenticateLongSessionAsync (
    QString username,
    QString password,
    QString consumerKey,
    QString consumerSecret,
    QString deviceIdIdentifier,
    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

<i>authenticationToken</i>	The authentication token for the user. This may not be a shared authentication token (returned by <code>NoteStore.authenticateToSharedNotebook</code> or <code>NoteStore.authenticateToSharedNote</code> ) 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

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• PERMISSION_DENIED "authenticationToken" - the provided authentication token is a shared or business authentication token.</li> <li>• PERMISSION_DENIED "Business" - the user identified by the provided authentication token is not currently a member of a business.</li> <li>• PERMISSION_DENIED "Business.status" - the business that the user is a member of is not currently in an active status.</li> <li>• BUSINESS_SECURITY_LOGIN_REQUIRED "sso" - the user must complete single sign-on before authenticating to the business.</li> </ul>
--	---

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

<i>clientName</i>	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".
<i>edamVersionMajor</i>	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.
<i>edamVersionMinor</i>	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 deviceIdIdentifier,
    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

<i>authenticationToken</i>	An authentication token returned by a previous call to <a href="#">UserStore.authenticate</a> or <a href="#">UserStore.authenticateLongSession</a> that could not be completed in a single call because a second factor was required.
<i>oneTimeCode</i>	The one time code entered by the user. This value is delivered out-of-band, typically via SMS or an authenticator application.
<i>deviceIdIdentifier</i>	See the corresponding parameter in <a href="#">authenticateLongSession</a> .
<i>deviceDescription</i>	See the corresponding parameter in <a href="#">authenticateLongSession</a> .

## 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

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• DATA_REQUIRED "authenticationToken" - authenticationToken is empty</li> <li>• DATA_REQUIRED "oneTimeCode" - oneTimeCode is empty</li> <li>• BAD_DATA_FORMAT "deviceIdIdentifier" - deviceIdIdentifier is not valid</li> <li>• BAD_DATA_FORMAT "authenticationToken" - authenticationToken is not well formed</li> <li>• INVALID_AUTH "oneTimeCode" - oneTimeCode did not match</li> <li>• AUTH_EXPIRED "authenticationToken" - authenticationToken has expired</li> <li>• PERMISSION_DENIED "authenticationToken" - authenticationToken is not valid</li> <li>• PERMISSION_DENIED "User.active" - user account is closed</li> <li>• PERMISSION_DENIED "User.tooManyFailuresTryAgainLater" - user has failed authentication too often</li> <li>• DATA_CONFLICT "User.twoFactorAuthentication" - The user has not enabled two-factor authentication.</li> </ul>
-----------------------------------	--

## 7.47.3.8 completeTwoFactorAuthenticationAsync()

```
virtual AsyncResult * qevercloud::IUserStore::completeTwoFactorAuthenticationAsync (
    QString oneTimeCode,
    QString deviceIdIdentifier,
    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

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• DATA_REQUIRED "serviceLevel" - serviceLevel is null</li> </ul>
-----------------------------------	---

### 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

<i>locale</i>	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

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>• DATA_REQUIRED "username" - username is empty</li> </ul>
-----------------------------------	--

**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

<i>authenticationToken</i>	the authentication token with sufficient privileges to manage Evernote Business membership or a business auth token.
<i>emailAddress</i>	the email address of the user to invite to join the Evernote Business account.

#### Exceptions

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"><li>• DATA_REQUIRED "email" - if no email address was provided</li><li>• BAD_DATA_FORMAT "email" - if the email address is not well formed</li><li>• DATA_CONFLICT "BusinessUser.email" - if there is already a user in the business whose business email address matches the specified email address.</li><li>• LIMIT_REACHED "Business.maxActiveUsers" - if the business has reached its user limit.</li></ul>
-----------------------------------	--

### 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**

<i>authenticationToken</i>	An authentication token with sufficient privileges to manage Evernote Business membership.
<i>includeRequestedInvitations</i>	If true, invitations with a status of <a href="#">BusinessInvitationStatus.REQUESTED</a> will be included in the returned list. If false, only invitations with a status of <a href="#">BusinessInvitationStatus.APPROVED</a> 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**

<i>authenticationToken</i>	A business authentication token returned by <a href="#">authenticateToBusiness</a> 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**

<i>authenticationToken</i>	An authentication token with sufficient privileges to manage Evernote Business membership.
<i>emailAddress</i>	The email address of the user to remove from the Evernote Business account.

**Exceptions**

<a href="#">EDAMUserException</a>	<ul style="list-style-type: none"> <li>DATA_REQUIRED "email" - if no email address was provided</li> <li>BAD_DATA_FORMAT "email" - The email address is not well formed</li> </ul>
<a href="#">EDAMNotFoundException</a>	<ul style="list-style-type: none"> <li>"email" - If there is no user with the specified email address in the business or that user was not invited via external provisioning.</li> </ul>

**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**

<i>authenticationToken</i>	the authentication token to revoke.
----------------------------	-------------------------------------

**Exceptions**

<i>EDAMUserException</i>	<ul style="list-style-type: none"> <li>• DATA_REQUIRED "authenticationToken" - no authentication token provided</li> <li>• BAD_DATA_FORMAT "authenticationToken" - the authentication token is not well formed</li> <li>• INVALID_AUTH "authenticationToken" - the authentication token is invalid</li> <li>• AUTH_EXPIRED "authenticationToken" - the authentication token is expired or is already revoked.</li> </ul>
--------------------------	--

**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") removeFromBusiness("baz@bar.com")
```

#### Parameters

<i>authenticationToken</i>	An authentication token with sufficient privileges to manage Evernote Business membership.
<i>oldEmailAddress</i>	The existing email address used to uniquely identify the user.
<i>newEmailAddress</i>	The new email address used to uniquely identify the user.

#### Exceptions

<a href="#"><i>EDAMUserException</i></a>	<ul style="list-style-type: none"> <li>• DATA_REQUIRED "oldEmailAddress" - No old email address was provided</li> <li>• DATA_REQUIRED "newEmailAddress" - No new email address was provided</li> <li>• BAD_DATA_FORMAT "oldEmailAddress" - The old email address is not well formed</li> <li>• BAD_DATA_FORMAT "newEmailAddress" - The new email address is not well formed</li> <li>• DATA_CONFLICT "oldEmailAddress" - The old and new email addresses were the same</li> <li>• DATA_CONFLICT "newEmailAddress" - There is already an invitation or registered user with the provided new email address.</li> <li>• DATA_CONFLICT "invitation.externallyProvisioned" - The user identified by oldEmailAddress was not added via UserStore.inviteToBusiness and therefore cannot be updated.</li> </ul>
<a href="#"><i>EDAMNotFoundException</i></a>	<ul style="list-style-type: none"> <li>• "oldEmailAddress" - If there is no user or invitation with the specified oldEmailAddress in the business.</li> </ul>

#### 7.47.3.31 updateBusinessUserIdentifierAsync()

```
virtual AsyncResult * qevercloud::IUserStore::updateBusinessUserIdentifierAsync (
    QString oldEmailAddress,
    QString newEmailAddress,
    IRequestContextPtr ctx = {} ) [pure virtual]
```

Asynchronous version of [`updateBusinessUserIdentifier`](#)

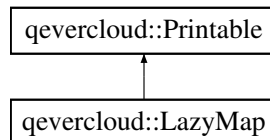
### 7.47.3.32 userStoreUrl()

```
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 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
```

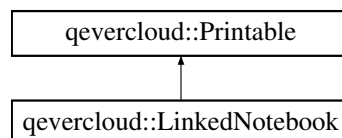
#### 7.48.5.2 keysOnly

```
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 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

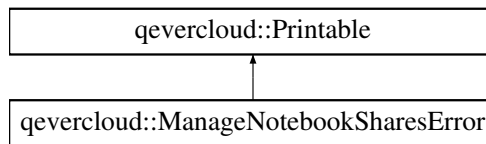
`Optional< QString > qevercloud::LinkedNotebook::webApiUrlPrefix`

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 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

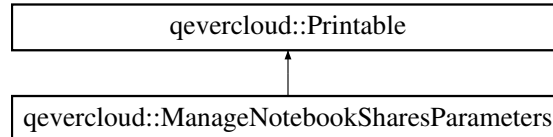
[Optional](#)< [UserIdentity](#) > qevercloud::ManageNotebookSharesError::userIdentity

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 Attributes

- [EverCloudLocalData](#) [localData](#)
- [Optional](#)< QString > [notebookGuid](#)
- [Optional](#)< QString > [inviteMessage](#)
- [Optional](#)< QList< [MemberShareRelationship](#) > > [membershipsToUpdate](#)
- [Optional](#)< QList< [InvitationShareRelationship](#) > > [invitationsToCreateOrUpdate](#)
- [Optional](#)< QList< [UserIdentity](#) > > [unshares](#)

### Properties

- [Optional](#)QList< [MemberShareRelationship](#) > [membershipsToUpdate](#)
- [Optional](#)QList< [InvitationShareRelationship](#) > [invitationsToCreateOrUpdate](#)
- [Optional](#)QList< [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 [manageNotebookShares](#) 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`

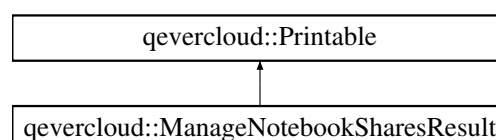
### 7.51.4.3 unshares

`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 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

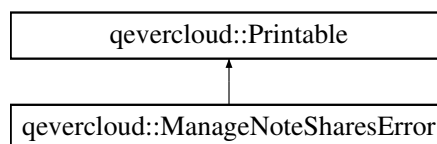
#### 7.52.4.1 errors

```
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 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

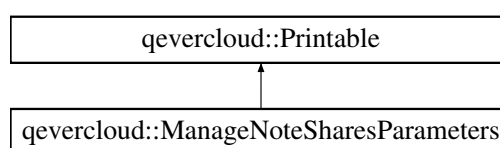
```
Optional< UserID > qevercloud::ManageNoteSharesError::userID
```

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 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 NoteInvitationShareRelationship 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::invitationsToUpdate`

### 7.54.4.3 membershipsToUnshare

`OptionalQList<UserID> qevercloud::ManageNoteSharesParameters::membershipsToUnshare`

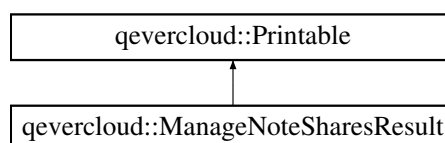
### 7.54.4.4 membershipsToUpdate

`OptionalQList<NoteMemberShareRelationship> qevercloud::ManageNoteSharesParameters::membershipsToUpdate`

## 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 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

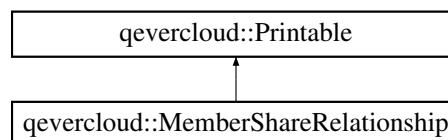
#### 7.55.4.1 errors

`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 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`

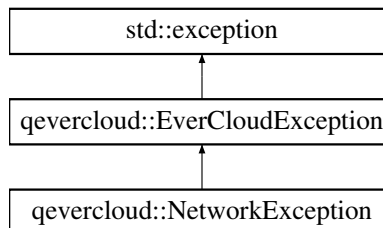
The user id of the user who most recently shared the notebook to this user. This field is currently unset for a [MemberShareRelationship](#) created by joining a notebook that has been published to the business (MemberShare↔Relationships where the individual privilege is unset). This field is used by 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

### Protected Attributes

- QNetworkReply::NetworkError [m\\_type](#)

#### 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

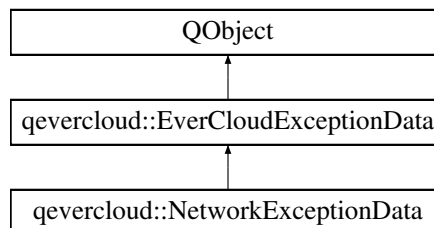
### 7.57.4.1 m\_type

```
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

### Protected Attributes

- QNetworkReply::NetworkError [m\\_type](#)

## Additional Inherited Members

### 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

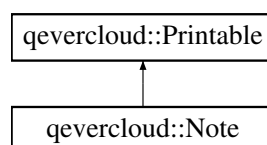
#### 7.58.4.1 m\_type

```
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 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.createNote()`

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 createNote(), 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 updateNote(), 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`

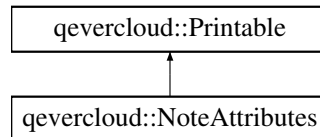
### 7.59.4.3 tagGuids

`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 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 contentClass 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 Timestamp). 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

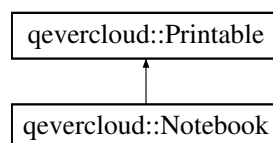
#### 7.60.5.1 classifications

`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 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`

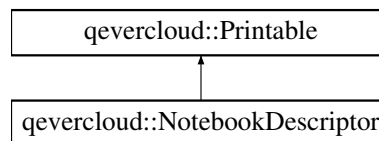
### 7.61.4.2 sharedNotebooks

`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 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

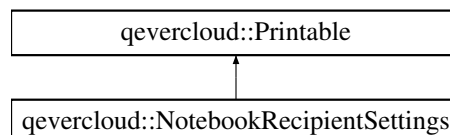
```
Optional< QString > qevercloud::NotebookDescriptor::notebookDisplayName
```

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 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

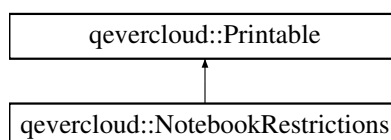
`Optional< QString >` `qevercloud::NotebookRecipientSettings::stack`

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 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 to 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

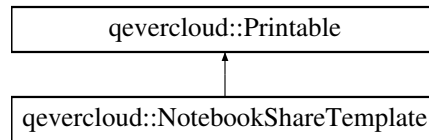
`Optional< SharedNotebookInstanceRestrictions > qevercloud::NotebookRestrictions::updateWhich↔  
SharedNotebookRestrictions`

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 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

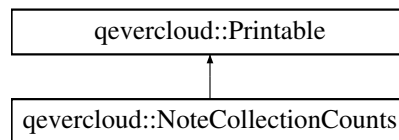
### 7.65.4.1 recipientContacts

```
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 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`

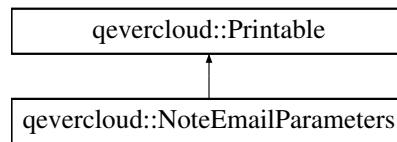
### 7.66.5.2 tagCounts

`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 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 has 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

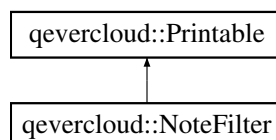
```
Optional< QStringList > qevercloud::NoteEmailParameters::toAddresses
```

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 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

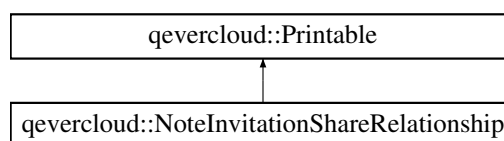
### 7.68.4.1 tagGuids

`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 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

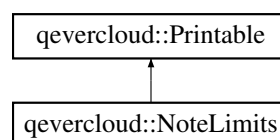
`Optional< UserID > qevercloud::NoteInvitationShareRelationship::sharerUserId`

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 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

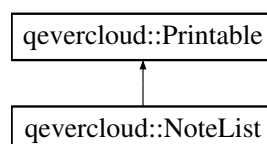
`Optional< qint64 > qevercloud::NoteLimits::uploadLimit`

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 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

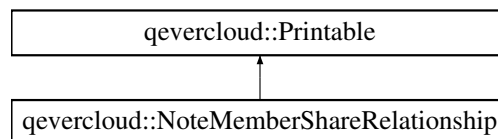
```
Optional< qint32 > qevercloud::NoteList::updateCount
```

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 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

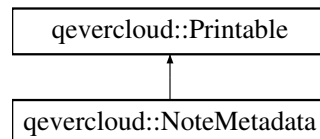
`Optional< UserID > qevercloud::NoteMemberShareRelationship::sharerUserId`

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 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

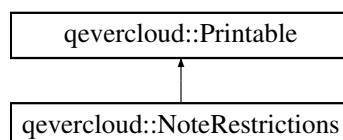
#### 7.73.4.1 tagGuids

`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 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

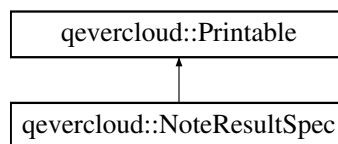
```
Optional< bool > qevercloud::NoteRestrictions::noUpdateTitle
```

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 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

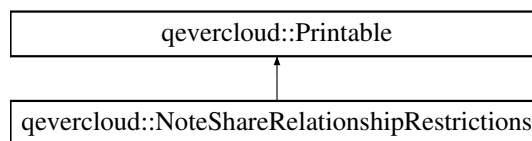
[EverCloudLocalData](#) `qevercloud::NoteResultSpec::localData`

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 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

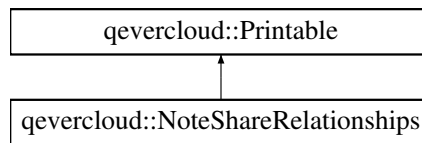
[Optional](#)< bool > qevercloud::NoteShareRelationshipRestrictions::noSetReadNote

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 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::invitationRestrictions

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 note. 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`

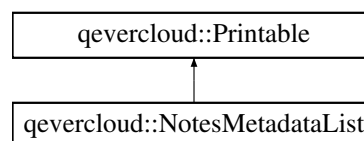
### 7.77.4.2 memberships

`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 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

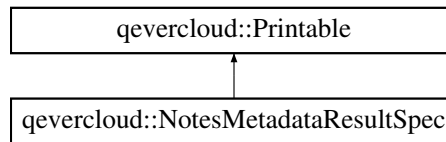
```
Optional< qint32 > qevercloud::NotesMetadataList::updateCount
```

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 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`

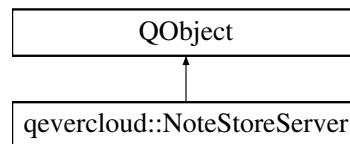
See the declaration of [EverCloudLocalData](#) for details

## 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](#) exceptionData)
- 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](#) exceptionData)
- 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](#) exceptionData)
- 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](#) exceptionData)
- void [onFindNoteCountsRequestReady](#) ([NoteCollectionCounts](#) value, [EverCloudExceptionDataPtr](#) exceptionData)
- void [onGetNoteWithResultSpecRequestReady](#) ([Note](#) value, [EverCloudExceptionDataPtr](#) exceptionData)
- void [onGetNoteRequestReady](#) ([Note](#) value, [EverCloudExceptionDataPtr](#) exceptionData)
- void [onGetNoteApplicationDataRequestReady](#) ([LazyMap](#) value, [EverCloudExceptionDataPtr](#) exceptionData)
- void [onGetNoteApplicationDataEntryRequestReady](#) (QString value, [EverCloudExceptionDataPtr](#) exceptionData)
- void [onSetNoteApplicationDataEntryRequestReady](#) (qint32 value, [EverCloudExceptionDataPtr](#) exceptionData)
- void [onUnsetNoteApplicationDataEntryRequestReady](#) (qint32 value, [EverCloudExceptionDataPtr](#) exceptionData)
- 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](#) exceptionData)
- 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](#) exceptionData)
- 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 [onUpdateNotelfUsnMatchesRequestReady](#) ([UpdateNotelfUsnMatchesResult](#) 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 recipientSettings, [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,
    quint32 offset,
    quint32 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,
    quint32 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]
```

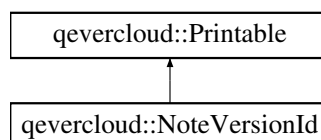
**7.80.3.223 updateTagRequestReady**

```
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 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

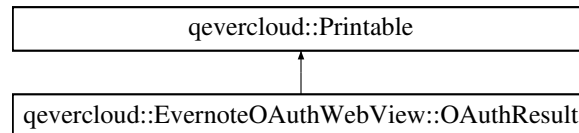
```
qint32 qevercloud::NoteVersionId::updateSequenceNum = 0
```

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 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 QEverCloud 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() != true
// 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]
```

Returns reference to the holded value.

There are contexts in C++ where implicit 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]
```

Returns 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

<code>defaultValue</code>	The value to return if <a href="#">Optional</a> is not set.
---------------------------	---

## Returns

[Optional](#) value if set and `defaultValue` otherwise.

## 7.83.4 Friends And Related Function Documentation

### 7.83.4.1 Optional

```
template<typename T >
template<typename X >
friend class Optional [friend]
```

### 7.83.4.2 swap

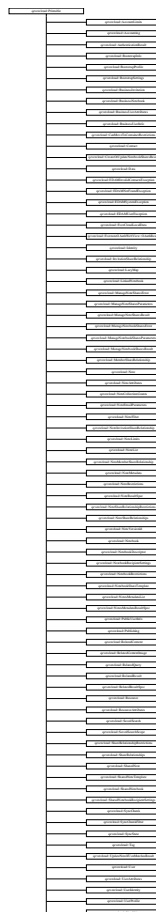
```
template<typename T >
void swap (
    Optional< T > & first,
    Optional< T > & second ) [friend]
```

## 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

- [QEVERTCLOUD\\_EXPORT](#) QTextStream & [operator<<](#) (QTextStream &strm, const [Printable](#) &printable)
- [QEVERTCLOUD\\_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::MemberShareRelationshipRestrictions](#), [qevercloud::NoteMemberShareRelationship](#), [qevercloud::NoteInvitationShareRelationship](#), [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 Function Documentation

## 7.84.3.1 operator&lt;&lt; [1/2]

```
QEVERCLOUD_EXPORT QDebug & operator<< (
    QDebug & dbg,
    const Printable & printable ) [friend]
```

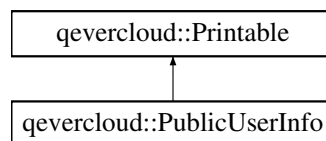
## 7.84.3.2 operator&lt;&lt; [2/2]

```
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 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

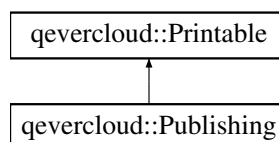
```
Optional< QString > qevercloud::PublicUserInfo::webApiUrlPrefix
```

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 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 >::QAssociativeContainerReferenceWrapper (
    Container & container ) [inline]
```

## 7.88.2 Member Function Documentation

### 7.88.2.1 begin()

```
template<typename Container >
iterator qevercloud::QAssociativeContainerReferenceWrapper< Container >::begin ( ) [inline]
```

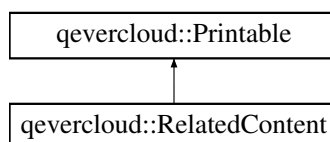
### 7.88.2.2 end()

```
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 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

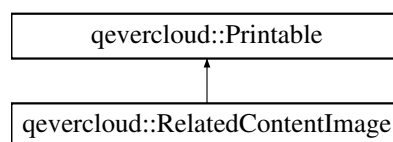
### 7.89.4.1 thumbnails

`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 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

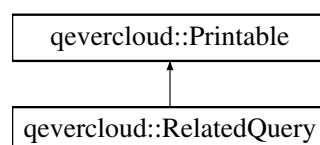
`Optional< qint32 > qevercloud::RelatedContentImage::width`

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 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

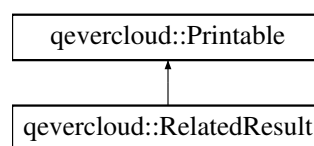
`Optional< QString > qevercloud::RelatedQuery::referenceUri`

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 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 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
```

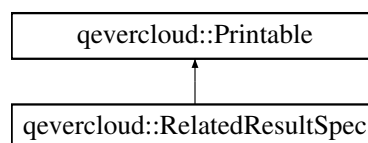
#### 7.92.4.6 tags

```
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 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 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

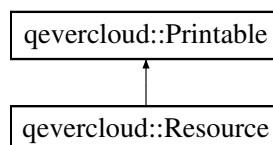
### 7.93.4.1 relatedContentTypes

`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 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

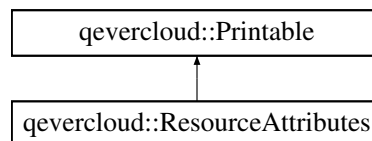
`Optional< quint16 > qevercloud::Resource::width`

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 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

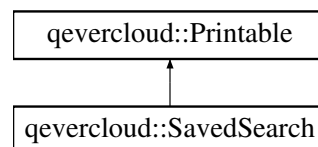
```
Optional< Timestamp > qevercloud::ResourceAttributes::timestamp
```

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 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

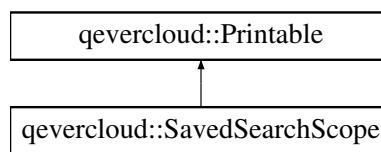
`Optional< qint32 > qevercloud::SavedSearch::updateSequenceNum`

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 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

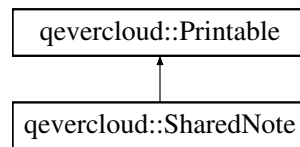
```
EverCloudLocalData qevercloud::SavedSearchScope::localData
```

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 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 may 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

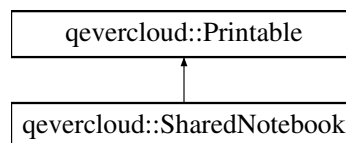
`Optional< UserID > qevercloud::SharedNote::sharerUserID`

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 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 recipientUsername 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 authenticateToSharedNotebook 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`

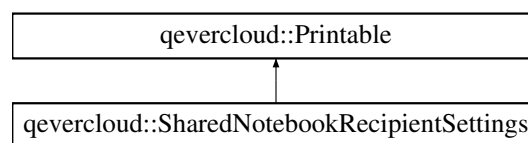
`Optional< QString > qevercloud::SharedNotebook::username`

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 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

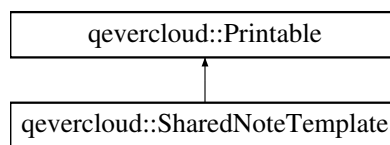
`Optional< bool > qevercloud::SharedNotebookRecipientSettings::reminderNotifyInApp`

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 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

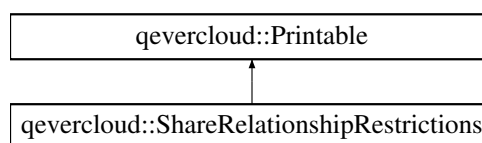
### 7.101.4.1 recipientContacts

`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 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

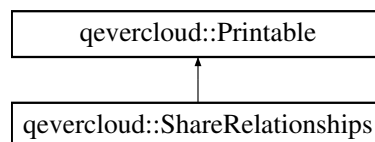
`Optional< bool > qevercloud::ShareRelationshipRestrictions::noSetReadPlusActivity`

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 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
```

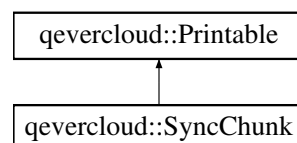
### 7.103.4.2 memberships

```
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 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

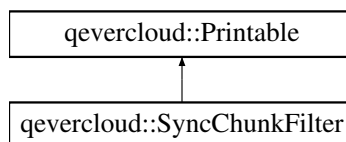
#### 7.104.4.11 tags

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 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

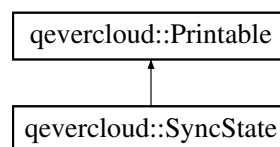
### 7.106.2.3 m\_name

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 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 `UserStore.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`

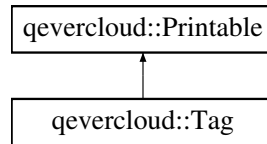
The greatest `MessageEventID` for this user's account. Clients that do a full sync should store this value locally and compare their local copy to the value returned by `getSyncState` to determine if they need to sync with `MessageStore`. This value will be omitted if the user has never sent or received 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 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

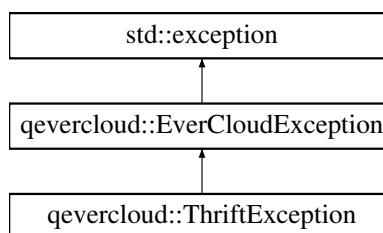
`Optional< qint32 > qevercloud::Tag::updateSequenceNum`

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

### Protected Attributes

- `Type m_type`

### 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	
INVALID_DATA	

### 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 Function Documentation

### 7.109.5.1 operator<<

```
QEVERCLOUD_EXPORT QTextStream & operator<< (
    QTextStream & strm,
    const Type type ) [friend]
```

## 7.109.6 Member Data Documentation

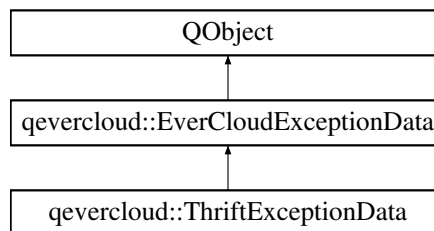
### 7.109.6.1 m\_type

```
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

### Protected Attributes

- [ThriftException::Type](#) m\_type

### Additional Inherited Members

### 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 &amp; 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

<i>host</i>	www.evernote.com or sandbox.evernote.com
<i>shardId</i>	You can get the value from UserStore service or as a result of an authentication.
<i>authenticationToken</i>	For working private notes/resources you must supply a valid authentication token. For public resources the value specified is not used.
<i>size</i>	The size of the thumbnail. Evernote supports values from from 1 to 300. By default 300 is used.
<i>imageType</i>	<a href="#">Thumbnail</a> 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

<i>guid</i>	The note or resource guid
<i>isPublic</i>	Specify true for public notes/resources. In this case authentication token is not sent to with the request as it should be according to the docs.
<i>isResourceGuid</i>	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 quint64 timeoutMsec = 30000 )
```

Downloads the thumbnail for a resource or a note.

#### Parameters

<i>guid</i>	The note or resource guid
<i>isPublic</i>	Specify true for public notes/resources. In this case authentication token is not sent to with the request as it should be according to the docs.
<i>isResourceGuid</i>	true if guid denotes a resource and false if it denotes a note.
<i>timeoutMsec</i>	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

<i>authenticationToken</i>	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

<i>host</i>	www.evernote.com or sandbox.evernote.com
-------------	--

#### 7.111.4.6 setImageType()

```
Thumbnail & qevercloud::Thumbnail::setImageType (
    ImageType imageType )
```

##### Parameters

<i>imageType</i>	<a href="#">Thumbnail</a> image type. See ImageType. By default PNG is used.
------------------	--

#### 7.111.4.7 setShardId()

```
Thumbnail & qevercloud::Thumbnail::setShardId (
    QString shardId )
```

## Parameters

<i>shard</i> ↔ <i>Id</i>	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

<i>size</i>	The size of the thumbnail. Evernote supports values from from 1 to 300. By default 300 is used.
-------------	---

**7.111.5 Friends And Related Function Documentation****7.111.5.1 operator<< [1/2]**

```
QEVERCLOUD_EXPORT QDebug & operator<< (
    QDebug & dbg,
    const ImageType imageType ) [friend]
```

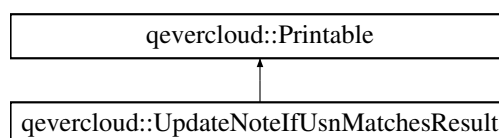
**7.111.5.2 operator<< [2/2]**

```
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 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`

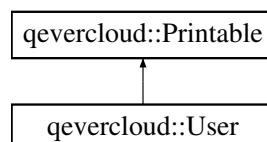
`Optional< bool >` `qevercloud::UpdateNoteIfUsnMatchesResult::updated`

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 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 '-'

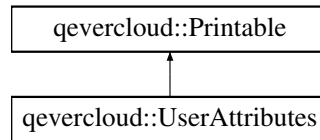
Length: EDAM\_USER\_USERNAME\_LEN\_MIN - EDAM\_USER\_USERNAME\_LEN\_MAX

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 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](#) > [referrerCode](#)
- [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 set, 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::referrerCode`

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`

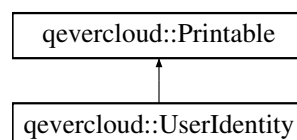
a list of promotions the user has seen. This list may occasionally be modified by the system when promotions are no longer available.

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 Attributes

- [EverCloudLocalData](#) localData
- [Optional](#)< [UserIdentityType](#) > type
- [Optional](#)< QString > [stringIdentifier](#)
- [Optional](#)< quint64 > [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

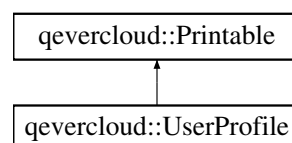
[Optional](#)< [UserIdentityType](#) > qevercloud::UserIdentity::type

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 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
```

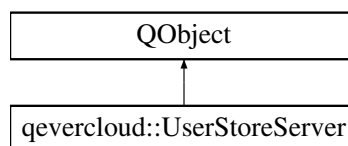
The user's Evernote username.

## 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 deviceIdIdentifier, QString deviceDescription, bool supportsTwoFactor, [IRequestContextPtr](#) ctx)
- void [completeTwoFactorAuthenticationRequest](#) (QString oneTimeCode, QString deviceIdIdentifier, 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,
    quint16 edamVersionMajor,
    quint16 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]
```

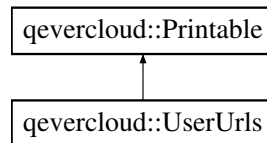
### 7.117.3.46 updateBusinessUserIdentifierRequestReady

```
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 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 asynchronous versions of functions.*

#### Namespaces

- namespace [qevercloud](#)

### 8.2 AsyncResult.h

[Go to the documentation of this file.](#)

```
1
9 #ifndef QEVERCLOUD_ASYNC_RESULT_H
10 #define QEVERCLOUD_ASYNC_RESULT_H
11
12 #include "EverCloudException.h"
13 #include "Export.h"
14 #include "Helpers.h"
15 #include "RequestContext.h"
16
17 #include <QNetworkRequest>
18 #include <QObject>
19 #include <QUuid>
20
21 namespace qevercloud {
22
23 QT_FORWARD_DECLARE_CLASS (AsyncResultPrivate)
```

```

24 QT_FORWARD_DECLARE_CLASS(DurableService)
25
26
53 class QEVERCLOUD_EXPORT AsyncResult: public QObject
54 {
55     Q_OBJECT
56     Q_DISABLE_COPY(AsyncResult)
57 public:
58     static QVariant asIs(QByteArray replyData);
59
60     typedef QVariant (*ReadFunctionType)(QByteArray replyData);
61
62     AsyncResult(QString url, QByteArray postData,
63                 IRequestContextPtr ctx,
64                 ReadFunctionType readFunction = AsyncResult::asIs,
65                 bool autoDelete = true, QObject * parent = nullptr);
66
67     AsyncResult(QNetworkRequest request, QByteArray postData,
68                 IRequestContextPtr ctx,
69                 ReadFunctionType readFunction = AsyncResult::asIs,
70                 bool autoDelete = true, QObject * parent = nullptr);
71
72     AsyncResult(QVariant result, EverCloudExceptionDataPtr error,
73                 IRequestContextPtr ctx, bool autoDelete = true,
74                 QObject * parent = nullptr);
75
76     ~AsyncResult();
77
78     bool waitForFinished(int timeout = -1);
79
80 Q_SIGNALS:
81     void finished(
82         QVariant result,
83         EverCloudExceptionDataPtr error,
84         IRequestContextPtr ctx);
85 private:
86     friend class DurableService;
87 private:
88     AsyncResultPrivate * const d_ptr;
89     Q_DECLARE_PRIVATE(AsyncResult)
90 };
91 // namespace qevercloud
92 #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.](#)

```

1
2 #ifndef QEVERCLOUD_DURABLE_SERVICE_H
3 #define QEVERCLOUD_DURABLE_SERVICE_H
4
5 #include "AsyncResult.h"
6 #include "Export.h"
7 #include "RequestContext.h"
8
9 #include <QDateTime>
10 #include <QVariant>
11
12 #include <functional>
13 #include <memory>
14 #include <utility>
15
16 namespace qevercloud {
17
18 struct QEVERCLOUD_EXPORT IRetryPolicy
19 {
20     virtual bool shouldRetry(
21         const EverCloudExceptionDataPtr & exceptionData) = 0;
22 };
23
24 using IRetryPolicyPtr = std::shared_ptr<IRetryPolicy>;
25
26 QT_FORWARD_DECLARE_CLASS(DurableServicePrivate)
27
28 class QEVERCLOUD_EXPORT IDurableService
29 {
30 public:
31     using SyncResult = std::pair<QVariant, EverCloudExceptionDataPtr>;
32     using SyncServiceCall = std::function<SyncResult (IRequestContextPtr)>;
33     using AsyncServiceCall = std::function<AsyncResult* (IRequestContextPtr)>;
34
35     struct QEVERCLOUD_EXPORT SyncRequest
36     {
37         const char * m_name;
38         QString m_description;
39         SyncServiceCall m_call;
40
41         SyncRequest(const char * name, QString description,
42                     SyncServiceCall && call) :
43             m_name(name),
44             m_description(std::move(description)),
45             m_call(std::move(call))
46         {}
47     };
48
49     struct QEVERCLOUD_EXPORT AsyncRequest
50     {
51         const char * m_name;
52         QString m_description;
53         AsyncServiceCall m_call;
54
55         AsyncRequest(const char * name, QString description,
56                     AsyncServiceCall && call) :
57             m_name(name),

```

```

68         m_description(std::move(description)),
69         m_call(std::move(call))
70     {}
71 };
72
73 public:
74     virtual SyncResult executeSyncRequest(
75         SyncRequest && syncRequest, IRequestContextPtr ctx) = 0;
76
77     virtual AsyncResult * executeAsyncRequest(
78         AsyncRequest && asyncRequest, IRequestContextPtr ctx) = 0;
79 };
80
81 using IDurableServicePtr = std::shared_ptr<IDurableService>;
82
83
84 QEVERCLOUD_EXPORT IRetryPolicyPtr newRetryPolicy();
85 QEVERCLOUD_EXPORT IRetryPolicyPtr nullRetryPolicy();
86
87 QEVERCLOUD_EXPORT IDurableServicePtr newDurableService(
88     IRetryPolicyPtr = {},
89     IRequestContextPtr = {});
90
91 } // namespace qevercloud
92
93 #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.](#)

```

1
2
3
4
5
6
7
8
9 #ifndef QEVERCLOUD_EVENT_LOOP_FINISH_H
10 #define QEVERCLOUD_EVENT_LOOP_FINISH_H
11
12 #include "Export.h"
13 #include "Helpers.h"
14
15 #include <QEventLoop>
16 #include <QObject>
17
18 namespace qevercloud {
19
20 QT_FORWARD_DECLARE_CLASS(EventLoopFinisherPrivate)
21
22 class QEVERCLOUD_EXPORT EventLoopFinisher: public QObject
23 {
24     Q_OBJECT
25 public:

```

```

26     explicit EventLoopFinisher(
27         QEventLoop * loop, int exitCode, QObject * parent = Q_NULLPTR);
28
29     ~EventLoopFinisher();
30
31 public Q_SLOTS:
32     void stopEventLoop();
33
34 private:
35     EventLoopFinisherPrivate * const d_ptr;
36     Q_DECLARE_PRIVATE(EventLoopFinisher)
37 };
38
39 } // namespace qevercloud
40
41 #endif // QEVERCLOUD_EVENT_LOOP_FINISH_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.](#)

```

1
9 #ifndef QEVERCLOUD_EVER_CLOUD_EXCEPTION_H
10 #define QEVERCLOUD_EVER_CLOUD_EXCEPTION_H
11
12 #include "Export.h"
13 #include "Helpers.h"
14
15 #include <QObject>
16 #include <QString>
17
18 #include <exception>
19 #include <memory>
20
21 namespace qevercloud {
22
23
24
25 class QEVERCLOUD_EXPORT EverCloudExceptionData;
26
27
28 class QEVERCLOUD_EXPORT EverCloudException: public std::exception
29 {
30 protected:
31     mutable QByteArray m_error;
32
33 public:
34     explicit EverCloudException();
35     explicit EverCloudException(QString error);
36     explicit EverCloudException(const std::string & error);
37     explicit EverCloudException(const char * error);
38
39     virtual ~EverCloudException() noexcept override;
40
41     virtual const char * what() const noexcept override;
42
43     virtual std::shared_ptr<EverCloudExceptionData> exceptionData() const;
44 };
45
46
47 class QEVERCLOUD_EXPORT EverCloudExceptionData: public QObject
48 {
49     Q_OBJECT
50     Q_DISABLE_COPY(EverCloudExceptionData)
51 public:
52     QString errorMessage;
53
54     explicit EverCloudExceptionData(QString error);
55
56     virtual void throwException() const;
57 };
58
59 using EverCloudExceptionDataPtr = std::shared_ptr<EverCloudExceptionData>;
60
61 class QEVERCLOUD_EXPORT EvernoteException: public EverCloudException
62 {
63 public:
64     explicit EvernoteException();
65     explicit EvernoteException(QString error);
66     explicit EvernoteException(const std::string & error);
67     explicit EvernoteException(const char * error);
68
69     virtual EverCloudExceptionDataPtr exceptionData() const override;
70 };
71
72 class QEVERCLOUD_EXPORT EvernoteExceptionData: public EverCloudExceptionData
73 {
74     Q_OBJECT
75     Q_DISABLE_COPY(EvernoteExceptionData)
76 public:
77     explicit EvernoteExceptionData(QString error);
78     virtual void throwException() const override;
79 };
80
81 } // namespace qevercloud
82
83 #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.](#)

```
1
9 #ifndef QEVERCLOUD_EXCEPTIONS_H
10 #define QEVERCLOUD_EXCEPTIONS_H
11
12 #include "EverCloudException.h"
13 #include "Export.h"
14 #include "Optional.h"
15 #include "generated/EDAMErrorCode.h"
16 #include "generated/Types.h"
17
18 #include <QNetworkReply>
19 #include <QObject>
20 #include <QString>
21
22 namespace qevercloud {
23
24
25
29 class QEVERCLOUD_EXPORT NetworkException: public EverCloudException
30 {
31 public:
32     NetworkException();
33     NetworkException(QNetworkReply::NetworkError error);
34     NetworkException(QNetworkReply::NetworkError error, QString message);
35     virtual ~NetworkException() noexcept override;
```

```

36
37     bool operator==(const NetworkException & other) const;
38     bool operator!=(const NetworkException & other) const;
39
40     QNetworkReply::NetworkError type() const;
41
42     const char * what() const noexcept override;
43
44     virtual EverCloudExceptionDataPtr exceptionData() const override;
45
46 protected:
47     QNetworkReply::NetworkError m_type;
48 };
49
50 class QEVERCLOUD_EXPORT NetworkExceptionData: public EverCloudExceptionData
51 {
52     Q_OBJECT
53     Q_DISABLE_COPY(NetworkExceptionData)
54 public:
55     explicit NetworkExceptionData(QString error, QNetworkReply::NetworkError type);
56     virtual void throwException() const override;
57
58 protected:
59     QNetworkReply::NetworkError m_type;
60 };
61
62 class QEVERCLOUD_EXPORT ThriftException: public EverCloudException
63 {
64 public:
65     enum class Type {
66         UNKNOWN = 0,
67         UNKNOWN_METHOD = 1,
68         INVALID_MESSAGE_TYPE = 2,
69         WRONG_METHOD_NAME = 3,
70         BAD_SEQUENCE_ID = 4,
71         MISSING_RESULT = 5,
72         INTERNAL_ERROR = 6,
73         PROTOCOL_ERROR = 7,
74         INVALID_DATA = 8
75     };
76
77     friend QEVERCLOUD_EXPORT QTextStream & operator<<(
78         QTextStream & strm, const Type type);
79
80     ThriftException();
81     ThriftException(Type type);
82     ThriftException(Type type, QString message);
83     virtual ~ThriftException() noexcept override;
84
85     bool operator==(const ThriftException & other) const;
86     bool operator!=(const ThriftException & other) const;
87
88     Type type() const;
89
90     const char * what() const noexcept override;
91
92     virtual EverCloudExceptionDataPtr exceptionData() const override;
93
94 protected:
95     Type m_type;
96 };
97
98 class QEVERCLOUD_EXPORT ThriftExceptionData: public EverCloudExceptionData
99 {
100     Q_OBJECT
101     Q_DISABLE_COPY(ThriftExceptionData)
102 public:
103     explicit ThriftExceptionData(QString error, ThriftException::Type type);
104     virtual void throwException() const override;
105
106 protected:
107     ThriftException::Type m_type;
108 };
109
110 class QEVERCLOUD_EXPORT EDAMUserExceptionData: public EvernoteExceptionData
111 {
112     Q_OBJECT
113     Q_DISABLE_COPY(EDAMUserExceptionData)
114 public:
115     explicit EDAMUserExceptionData(
116         QString error, EDAMErrorCode errorCode, Optional<QString> parameter);
117     virtual void throwException() const override;
118
119 protected:

```

```

141     EDAMErrorCode      m_errorCode;
142     Optional<QString>   m_parameter;
143 };
144
145
146
147 class QEVERCLOUD_EXPORT EDAMSystemExceptionData: public EvernoteExceptionData
148 {
149     Q_OBJECT
150     Q_DISABLE_COPY(EDAMSystemExceptionData)
151 public:
152     explicit EDAMSystemExceptionData(
153         QString err, EDAMErrorCode errorCode, Optional<QString> message,
154         Optional<qint32> rateLimitDuration);
155
156     virtual void throwException() const override;
157
158 protected:
159     EDAMErrorCode      m_errorCode;
160     Optional<QString>   m_message;
161     Optional<qint32>    m_rateLimitDuration;
162 };
163
164
165
166 class QEVERCLOUD_EXPORT EDAMNotFoundExceptionData: public EvernoteExceptionData
167 {
168     Q_OBJECT
169     Q_DISABLE_COPY(EDAMNotFoundExceptionData)
170 public:
171     explicit EDAMNotFoundException(
172         QString error, Optional<QString> identifier, Optional<QString> key);
173
174     virtual void throwException() const override;
175
176 protected:
177     Optional<QString> m_identifier;
178     Optional<QString> m_key;
179 };
180
181
182
183 class QEVERCLOUD_EXPORT EDAMInvalidContactsExceptionData:
184     public EvernoteExceptionData
185 {
186     Q_OBJECT
187     Q_DISABLE_COPY(EDAMInvalidContactsExceptionData)
188 public:
189     explicit EDAMInvalidContactsExceptionData(
190         QList<Contact> contacts, Optional<QString> parameter,
191         Optional<QList<EDAMInvalidContactReason> > reasons);
192
193     virtual void throwException() const override;
194
195 protected:
196     QList<Contact>      m_contacts;
197     Optional<QString>   m_parameter;
198     Optional<QList<EDAMInvalidContactReason>> m_reasons;
199 };
200
201
202
203 class QEVERCLOUD_EXPORT EDAMSystemExceptionRateLimitReached:
204     public EDAMSystemException
205 {
206 public:
207     virtual EverCloudExceptionDataPtr exceptionData() const override;
208 };
209
210
211
212 class QEVERCLOUD_EXPORT EDAMSystemExceptionRateLimitReachedData:
213     public EDAMSystemExceptionData
214 {
215     Q_OBJECT
216     Q_DISABLE_COPY(EDAMSystemExceptionRateLimitReachedData)
217 public:
218     explicit EDAMSystemExceptionRateLimitReachedData(
219         QString error, EDAMErrorCode errorCode, Optional<QString> message,
220         Optional<qint32> rateLimitDuration);
221
222     virtual void throwException() const override;
223 };
224
225
226
227 class QEVERCLOUD_EXPORT EDAMSystemExceptionAuthExpired: public EDAMSystemException
228 {
229 public:
230     virtual EverCloudExceptionDataPtr exceptionData() const override;
231 };
232
233
234
235

```

```

261 class QEVERCLOUD_EXPORT EDAMSystemExceptionAuthExpiredData:
262     public EDAMSystemExceptionData
263 {
264     Q_OBJECT
265     Q_DISABLE_COPY(EDAMSystemExceptionAuthExpiredData)
266 public:
267     explicit EDAMSystemExceptionAuthExpiredData(
268         QString error, EDAMErrorCode errorCode, Optional<QString> message,
269         Optional<qint32> rateLimitDuration);
270
271     virtual void throwException() const override;
272 };
273
274 } // namespace qevercloud
275
276 #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.](#)

```

1
9 #ifndef QEVERCLOUD_EXPORT_H
10 #define QEVERCLOUD_EXPORT_H
11
12 #include <QtCore/QtGlobal>
13
14 #if defined(QEVERCLOUD_SHARED_LIBRARY)
15 #   define QEVERCLOUD_EXPORT Q_DECL_EXPORT
16 #elif defined(QEVERCLOUD_STATIC_LIBRARY)
17 #   define QEVERCLOUD_EXPORT Q_DECL_EXPORT
18 #else
19 #   define QEVERCLOUD_EXPORT Q_DECL_IMPORT
20 #endif
21
22 #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.](#)

```

1
12 #ifndef QEVERCLOUD_GENERATED_CONSTANTS_H
13 #define QEVERCLOUD_GENERATED_CONSTANTS_H
14
15 #include "../Export.h"
16
17 namespace qevercloud {
18
19
20
21 // Limits.thrift
25 QEVERCLOUD_EXPORT extern const quint32 EDAM_ATTRIBUTE_LEN_MIN;
26
27 // Limits.thrift
31 QEVERCLOUD_EXPORT extern const quint32 EDAM_ATTRIBUTE_LEN_MAX;
32
33 // Limits.thrift
38 QEVERCLOUD_EXPORT extern const QString EDAM_ATTRIBUTE_REGEX;
39
40 // Limits.thrift
45 QEVERCLOUD_EXPORT extern const quint32 EDAM_ATTRIBUTE_LIST_MAX;
46
47 // Limits.thrift
52 QEVERCLOUD_EXPORT extern const quint32 EDAM_ATTRIBUTE_MAP_MAX;
53
54 // Limits.thrift
58 QEVERCLOUD_EXPORT extern const quint32 EDAM_GUID_LEN_MIN;
59
60 // Limits.thrift
64 QEVERCLOUD_EXPORT extern const quint32 EDAM_GUID_LEN_MAX;
65
66 // Limits.thrift
70 QEVERCLOUD_EXPORT extern const QString EDAM_GUID_REGEX;
71
72 // Limits.thrift
76 QEVERCLOUD_EXPORT extern const quint32 EDAM_EMAIL_LEN_MIN;
77
78 // Limits.thrift
82 QEVERCLOUD_EXPORT extern const quint32 EDAM_EMAIL_LEN_MAX;
83
84 // Limits.thrift
89 QEVERCLOUD_EXPORT extern const QString EDAM_EMAIL_LOCAL_REGEX;
90
91 // Limits.thrift
96 QEVERCLOUD_EXPORT extern const QString EDAM_EMAIL_DOMAIN_REGEX;
97
98 // Limits.thrift
103 QEVERCLOUD_EXPORT extern const QString EDAM_EMAIL_REGEX;
104
105 // Limits.thrift
111 QEVERCLOUD_EXPORT extern const QString EDAM_VAT_REGEX;
112
113 // Limits.thrift
117 QEVERCLOUD_EXPORT extern const quint32 EDAM_TIMEZONE_LEN_MIN;
118
119 // Limits.thrift
123 QEVERCLOUD_EXPORT extern const quint32 EDAM_TIMEZONE_LEN_MAX;
124
125 // Limits.thrift
134 QEVERCLOUD_EXPORT extern const QString EDAM_TIMEZONE_REGEX;
135
136 // Limits.thrift
140 QEVERCLOUD_EXPORT extern const quint32 EDAM_MIME_LEN_MIN;
141
142 // Limits.thrift
146 QEVERCLOUD_EXPORT extern const quint32 EDAM_MIME_LEN_MAX;
147
148 // Limits.thrift
153 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_REGEX;
154
155 // Limits.thrift
157 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_GIF;
158
159 // Limits.thrift
161 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_JPEG;
162
163 // Limits.thrift
165 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_PNG;
166
167 // Limits.thrift
169 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_TIFF;
170
171 // Limits.thrift

```

```

173 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_BMP;
174
175 // Limits.thrift
177 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_WAV;
178
179 // Limits.thrift
181 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_MP3;
182
183 // Limits.thrift
185 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_AMR;
186
187 // Limits.thrift
189 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_AAC;
190
191 // Limits.thrift
193 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_M4A;
194
195 // Limits.thrift
197 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_MP4_VIDEO;
198
199 // Limits.thrift
201 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_INK;
202
203 // Limits.thrift
205 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_PDF;
206
207 // Limits.thrift
209 QEVERCLOUD_EXPORT extern const QString EDAM_MIME_TYPE_DEFAULT;
210
211 // Limits.thrift
216 QEVERCLOUD_EXPORT extern const QSet<QString> EDAM_MIME_TYPES;
217
218 // Limits.thrift
224 QEVERCLOUD_EXPORT extern const QSet<QString> EDAM_INDEXABLE_RESOURCE_MIME_TYPES;
225
226 // Limits.thrift
232 QEVERCLOUD_EXPORT extern const QSet<QString> EDAM_INDEXABLE_PLAINTEXT_MIME_TYPES;
233
234 // Limits.thrift
238 QEVERCLOUD_EXPORT extern const quint32 EDAM_SEARCH_QUERY_LEN_MIN;
239
240 // Limits.thrift
244 QEVERCLOUD_EXPORT extern const quint32 EDAM_SEARCH_QUERY_LEN_MAX;
245
246 // Limits.thrift
252 QEVERCLOUD_EXPORT extern const QString EDAM_SEARCH_QUERY_REGEX;
253
254 // Limits.thrift
260 QEVERCLOUD_EXPORT extern const quint32 EDAM_HASH_LEN;
261
262 // Limits.thrift
266 QEVERCLOUD_EXPORT extern const quint32 EDAM_USER_USERNAME_LEN_MIN;
267
268 // Limits.thrift
272 QEVERCLOUD_EXPORT extern const quint32 EDAM_USER_USERNAME_LEN_MAX;
273
274 // Limits.thrift
280 QEVERCLOUD_EXPORT extern const QString EDAM_USER_USERNAME_REGEX;
281
282 // Limits.thrift
286 QEVERCLOUD_EXPORT extern const quint32 EDAM_USER_NAME_LEN_MIN;
287
288 // Limits.thrift
292 QEVERCLOUD_EXPORT extern const quint32 EDAM_USER_NAME_LEN_MAX;
293
294 // Limits.thrift
299 QEVERCLOUD_EXPORT extern const QString EDAM_USER_NAME_REGEX;
300
301 // Limits.thrift
305 QEVERCLOUD_EXPORT extern const quint32 EDAM_TAG_NAME_LEN_MIN;
306
307 // Limits.thrift
311 QEVERCLOUD_EXPORT extern const quint32 EDAM_TAG_NAME_LEN_MAX;
312
313 // Limits.thrift
319 QEVERCLOUD_EXPORT extern const QString EDAM_TAG_NAME_REGEX;
320
321 // Limits.thrift
325 QEVERCLOUD_EXPORT extern const quint32 EDAM_NOTE_TITLE_LEN_MIN;
326
327 // Limits.thrift
331 QEVERCLOUD_EXPORT extern const quint32 EDAM_NOTE_TITLE_LEN_MAX;
332
333 // Limits.thrift
339 QEVERCLOUD_EXPORT extern const QString EDAM_NOTE_TITLE_REGEX;
340
341 // Limits.thrift

```

```

346 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_CONTENT_LEN_MIN;
347
348 // Limits.thrift
353 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_CONTENT_LEN_MAX;
354
355 // Limits.thrift
361 QEVERCLOUD_EXPORT extern const qint32 EDAM_APPLICATIONDATA_NAME_LEN_MIN;
362
363 // Limits.thrift
369 QEVERCLOUD_EXPORT extern const qint32 EDAM_APPLICATIONDATA_NAME_LEN_MAX;
370
371 // Limits.thrift
376 QEVERCLOUD_EXPORT extern const qint32 EDAM_APPLICATIONDATA_VALUE_LEN_MIN;
377
378 // Limits.thrift
386 QEVERCLOUD_EXPORT extern const qint32 EDAM_APPLICATIONDATA_VALUE_LEN_MAX;
387
388 // Limits.thrift
393 QEVERCLOUD_EXPORT extern const qint32 EDAM_APPLICATIONDATA_ENTRY_LEN_MAX;
394
395 // Limits.thrift
404 QEVERCLOUD_EXPORT extern const QString EDAM_APPLICATIONDATA_NAME_REGEX;
405
406 // Limits.thrift
413 QEVERCLOUD_EXPORT extern const QString EDAM_APPLICATIONDATA_VALUE_REGEX;
414
415 // Limits.thrift
419 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTEBOOK_NAME_LEN_MIN;
420
421 // Limits.thrift
425 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTEBOOK_NAME_LEN_MAX;
426
427 // Limits.thrift
433 QEVERCLOUD_EXPORT extern const QString EDAM_NOTEBOOK_NAME_REGEX;
434
435 // Limits.thrift
439 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTEBOOK_STACK_LEN_MIN;
440
441 // Limits.thrift
445 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTEBOOK_STACK_LEN_MAX;
446
447 // Limits.thrift
453 QEVERCLOUD_EXPORT extern const QString EDAM_NOTEBOOK_STACK_REGEX;
454
455 // Limits.thrift
459 QEVERCLOUD_EXPORT extern const qint32 EDAM_WORKSPACE_NAME_LEN_MIN;
460
461 // Limits.thrift
465 QEVERCLOUD_EXPORT extern const qint32 EDAM_WORKSPACE_NAME_LEN_MAX;
466
467 // Limits.thrift
471 QEVERCLOUD_EXPORT extern const qint32 EDAM_WORKSPACE_DESCRIPTION_LEN_MAX;
472
473 // Limits.thrift
479 QEVERCLOUD_EXPORT extern const QString EDAM_WORKSPACE_NAME_REGEX;
480
481 // Limits.thrift
485 QEVERCLOUD_EXPORT extern const qint32 EDAM_PUBLISHING_URI_LEN_MIN;
486
487 // Limits.thrift
491 QEVERCLOUD_EXPORT extern const qint32 EDAM_PUBLISHING_URI_LEN_MAX;
492
493 // Limits.thrift
497 QEVERCLOUD_EXPORT extern const QString EDAM_PUBLISHING_URI_REGEX;
498
499 // Limits.thrift
503 QEVERCLOUD_EXPORT extern const QSet<QString> EDAM_PUBLISHING_URI_PROHIBITED;
504
505 // Limits.thrift
509 QEVERCLOUD_EXPORT extern const qint32 EDAM_PUBLISHING_DESCRIPTION_LEN_MIN;
510
511 // Limits.thrift
515 QEVERCLOUD_EXPORT extern const qint32 EDAM_PUBLISHING_DESCRIPTION_LEN_MAX;
516
517 // Limits.thrift
524 QEVERCLOUD_EXPORT extern const QString EDAM_PUBLISHING_DESCRIPTION_REGEX;
525
526 // Limits.thrift
530 QEVERCLOUD_EXPORT extern const qint32 EDAM_SAVED_SEARCH_NAME_LEN_MIN;
531
532 // Limits.thrift
536 QEVERCLOUD_EXPORT extern const qint32 EDAM_SAVED_SEARCH_NAME_LEN_MAX;
537
538 // Limits.thrift
544 QEVERCLOUD_EXPORT extern const QString EDAM_SAVED_SEARCH_NAME_REGEX;
545
546 // Limits.thrift

```

```
550 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_PASSWORD_LEN_MIN;
551
552 // Limits.thrift
553 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_PASSWORD_LEN_MAX;
554
555 // Limits.thrift
556 QEVERCLOUD_EXPORT extern const QString EDAM_USER_PASSWORD_REGEX;
557
558 // Limits.thrift
559 QEVERCLOUD_EXPORT extern const qint32 EDAM_BUSINESS_URI_LEN_MAX;
560
561 // Limits.thrift
562 QEVERCLOUD_EXPORT extern const QString EDAM_BUSINESS_MARKETING_CODE_REGEX_PATTERN;
563
564 // Limits.thrift
565 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_TAGS_MAX;
566
567 // Limits.thrift
568 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_RESOURCES_MAX;
569
570 // Limits.thrift
571 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_TAGS_MAX;
572
573 // Limits.thrift
574 QEVERCLOUD_EXPORT extern const qint32 EDAM_BUSINESS_TAGS_MAX;
575
576 // Limits.thrift
577 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_SAVED_SEARCHES_MAX;
578
579 // Limits.thrift
580 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_NOTES_MAX;
581
582 // Limits.thrift
583 QEVERCLOUD_EXPORT extern const qint32 EDAM_BUSINESS_NOTES_MAX;
584
585 // Limits.thrift
586 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_NOTEBOOKS_MAX;
587
588 // Limits.thrift
589 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_WORKSPACES_MAX;
590
591 // Limits.thrift
592 QEVERCLOUD_EXPORT extern const qint32 EDAM_BUSINESS_NOTEBOOKS_MAX;
593
594 // Limits.thrift
595 QEVERCLOUD_EXPORT extern const qint32 EDAM_BUSINESS_WORKSPACES_MAX;
596
597 // Limits.thrift
598 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_RECENT_MAILED_ADDRESSES_MAX;
599
600 // Limits.thrift
601 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_MAIL_LIMIT_DAILY_FREE;
602
603 // Limits.thrift
604 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_MAIL_LIMIT_DAILY_PREMIUM;
605
606 // Limits.thrift
607 QEVERCLOUD_EXPORT extern const qint64 EDAM_USER_UPLOAD_LIMIT_FREE;
608
609 // Limits.thrift
610 QEVERCLOUD_EXPORT extern const qint64 EDAM_USER_UPLOAD_LIMIT_PREMIUM;
611
612 // Limits.thrift
613 QEVERCLOUD_EXPORT extern const qint64 EDAM_USER_UPLOAD_LIMIT_BUSINESS_FIRST_MONTH;
614
615 // Limits.thrift
616 QEVERCLOUD_EXPORT extern const qint64 EDAM_USER_UPLOAD_LIMIT_BUSINESS_NEXT_MONTH;
617
618 // Limits.thrift
619 QEVERCLOUD_EXPORT extern const qint64 EDAM_USER_UPLOAD_LIMIT_PLUS;
620
621 // Limits.thrift
622 QEVERCLOUD_EXPORT extern const qint64 EDAM_USER_UPLOAD_SURVEY_THRESHOLD;
623
624 // Limits.thrift
625 QEVERCLOUD_EXPORT extern const qint64 EDAM_USER_UPLOAD_LIMIT_BUSINESS;
626
627 // Limits.thrift
628 QEVERCLOUD_EXPORT extern const qint64 EDAM_USER_UPLOAD_LIMIT_BUSINESS_PER_USER;
629
630 // Limits.thrift
631 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_SIZE_MAX_FREE;
632
633 // Limits.thrift
634 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_SIZE_MAX_PREMIUM;
635
636 // Limits.thrift
```

```
746 QEVERCLOUD_EXPORT extern const qint32 EDAM_RESOURCE_SIZE_MAX_FREE;
747
748 // Limits.thrift
752 QEVERCLOUD_EXPORT extern const qint32 EDAM_RESOURCE_SIZE_MAX_PREMIUM;
753
754 // Limits.thrift
759 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_LINKED_NOTEBOOK_MAX;
760
761 // Limits.thrift
767 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_LINKED_NOTEBOOK_MAX_PREMIUM;
768
769 // Limits.thrift
773 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTEBOOK_BUSINESS_SHARED_NOTEBOOK_MAX;
774
775 // Limits.thrift
779 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTEBOOK_PERSONAL_SHARED_NOTEBOOK_MAX;
780
781 // Limits.thrift
785 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_BUSINESS_SHARED_NOTE_MAX;
786
787 // Limits.thrift
791 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_PERSONAL_SHARED_NOTE_MAX;
792
793 // Limits.thrift
797 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_CONTENT_CLASS_LEN_MIN;
798
799 // Limits.thrift
803 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_CONTENT_CLASS_LEN_MAX;
804
805 // Limits.thrift
810 QEVERCLOUD_EXPORT extern const QString EDAM_NOTE_CONTENT_CLASS_REGEX;
811
812 // Limits.thrift
820 QEVERCLOUD_EXPORT extern const QString EDAM_HELLO_APP_CONTENT_CLASS_PREFIX;
821
822 // Limits.thrift
830 QEVERCLOUD_EXPORT extern const QString EDAM_FOOD_APP_CONTENT_CLASS_PREFIX;
831
832 // Limits.thrift
839 QEVERCLOUD_EXPORT extern const QString EDAM_CONTENT_CLASS_HELLO_ENCOUNTER;
840
841 // Limits.thrift
848 QEVERCLOUD_EXPORT extern const QString EDAM_CONTENT_CLASS_HELLO_PROFILE;
849
850 // Limits.thrift
857 QEVERCLOUD_EXPORT extern const QString EDAM_CONTENT_CLASS_FOOD_MEAL;
858
859 // Limits.thrift
865 QEVERCLOUD_EXPORT extern const QString EDAM_CONTENT_CLASS_SKITCH_PREFIX;
866
867 // Limits.thrift
872 QEVERCLOUD_EXPORT extern const QString EDAM_CONTENT_CLASS_SKITCH;
873
874 // Limits.thrift
879 QEVERCLOUD_EXPORT extern const QString EDAM_CONTENT_CLASS_SKITCH_PDF;
880
881 // Limits.thrift
887 QEVERCLOUD_EXPORT extern const QString EDAM_CONTENT_CLASS_PENULTIMATE_PREFIX;
888
889 // Limits.thrift
894 QEVERCLOUD_EXPORT extern const QString EDAM_CONTENT_CLASS_PENULTIMATE_NOTEBOOK;
895
896 // Limits.thrift
901 QEVERCLOUD_EXPORT extern const QString EDAM_SOURCE_APPLICATION_POSTIT;
902
903 // Limits.thrift
908 QEVERCLOUD_EXPORT extern const QString EDAM_SOURCE_APPLICATION_MOLESKINE;
909
910 // Limits.thrift
915 QEVERCLOUD_EXPORT extern const QString EDAM_SOURCE_APPLICATION_EN_SCANSNAP;
916
917 // Limits.thrift
922 QEVERCLOUD_EXPORT extern const QString EDAM_SOURCE_APPLICATION_EWC;
923
924 // Limits.thrift
929 QEVERCLOUD_EXPORT extern const QString EDAM_SOURCE_APPLICATION_ANDROID_SHARE_EXTENSION;
930
931 // Limits.thrift
936 QEVERCLOUD_EXPORT extern const QString EDAM_SOURCE_APPLICATION_IOS_SHARE_EXTENSION;
937
938 // Limits.thrift
943 QEVERCLOUD_EXPORT extern const QString EDAM_SOURCE_APPLICATION_WEB_CLIPPER;
944
945 // Limits.thrift
949 QEVERCLOUD_EXPORT extern const QString EDAM_SOURCE_OUTLOOK_CLIPPER;
950
951 // Limits.thrift
```

```

956 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_TITLE_QUALITY_UNTITLED;
957
958 // Limits.thrift
964 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_TITLE_QUALITY_LOW;
965
966 // Limits.thrift
972 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_TITLE_QUALITY_MEDIUM;
973
974 // Limits.thrift
980 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_TITLE_QUALITY_HIGH;
981
982 // Limits.thrift
987 QEVERCLOUD_EXPORT extern const qint32 EDAM_RELATED_PLAINTEXT_LEN_MIN;
988
989 // Limits.thrift
994 QEVERCLOUD_EXPORT extern const qint32 EDAM_RELATED_PLAINTEXT_LEN_MAX;
995
996 // Limits.thrift
1001 QEVERCLOUD_EXPORT extern const qint32 EDAM_RELATED_MAX_NOTES;
1002
1003 // Limits.thrift
1008 QEVERCLOUD_EXPORT extern const qint32 EDAM_RELATED_MAX_NOTEBOOKS;
1009
1010 // Limits.thrift
1014 QEVERCLOUD_EXPORT extern const qint32 EDAM_RELATED_MAX_TAGS;
1015
1016 // Limits.thrift
1020 QEVERCLOUD_EXPORT extern const qint32 EDAM_RELATED_MAX_EXPERTS;
1021
1022 // Limits.thrift
1027 QEVERCLOUD_EXPORT extern const qint32 EDAM_RELATED_MAX_RELATED_CONTENT;
1028
1029 // Limits.thrift
1034 QEVERCLOUD_EXPORT extern const qint32 EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_LEN_MIN;
1035
1036 // Limits.thrift
1041 QEVERCLOUD_EXPORT extern const qint32 EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_LEN_MAX;
1042
1043 // Limits.thrift
1049 QEVERCLOUD_EXPORT extern const QString EDAM_BUSINESS_NOTEBOOK_DESCRIPTION_REGEX;
1050
1051 // Limits.thrift
1055 QEVERCLOUD_EXPORT extern const qint32 EDAM_BUSINESS_PHONE_NUMBER_LEN_MAX;
1056
1057 // Limits.thrift
1061 QEVERCLOUD_EXPORT extern const qint32 EDAM_PREFERENCE_NAME_LEN_MIN;
1062
1063 // Limits.thrift
1067 QEVERCLOUD_EXPORT extern const qint32 EDAM_PREFERENCE_NAME_LEN_MAX;
1068
1069 // Limits.thrift
1073 QEVERCLOUD_EXPORT extern const qint32 EDAM_PREFERENCE_VALUE_LEN_MIN;
1074
1075 // Limits.thrift
1079 QEVERCLOUD_EXPORT extern const qint32 EDAM_PREFERENCE_VALUE_LEN_MAX;
1080
1081 // Limits.thrift
1085 QEVERCLOUD_EXPORT extern const qint32 EDAM_MAX_PREFERENCES;
1086
1087 // Limits.thrift
1092 QEVERCLOUD_EXPORT extern const qint32 EDAM_MAX_VALUES_PER_PREFERENCE;
1093
1094 // Limits.thrift
1102 QEVERCLOUD_EXPORT extern const qint32 EDAM_PREFERENCE_ONLY_ONE_VALUE_LEN_MAX;
1103
1104 // Limits.thrift
1108 QEVERCLOUD_EXPORT extern const QString EDAM_PREFERENCE_NAME_REGEX;
1109
1110 // Limits.thrift
1115 QEVERCLOUD_EXPORT extern const QString EDAM_PREFERENCE_VALUE_REGEX;
1116
1117 // Limits.thrift
1122 QEVERCLOUD_EXPORT extern const QString EDAM_PREFERENCE_ONLY_ONE_VALUE_REGEX;
1123
1124 // Limits.thrift
1128 QEVERCLOUD_EXPORT extern const QString EDAM_PREFERENCE_SHORTCUTS;
1129
1130 // Limits.thrift
1141 QEVERCLOUD_EXPORT extern const QString EDAM_PREFERENCE_BUSINESS_DEFAULT_NOTEBOOK;
1142
1143 // Limits.thrift
1158 QEVERCLOUD_EXPORT extern const QString EDAM_PREFERENCE_BUSINESS_QUICKNOTE;
1159
1160 // Limits.thrift
1164 QEVERCLOUD_EXPORT extern const qint32 EDAM_PREFERENCE_SHORTCUTS_MAX_VALUES;
1165
1166 // Limits.thrift

```

```
1170 QEVERCLOUD_EXPORT extern const qint32 EDAM_DEVICE_ID_LEN_MAX;
1171
1172 // Limits.thrift
1176 QEVERCLOUD_EXPORT extern const QString EDAM_DEVICE_ID_REGEX;
1177
1178 // Limits.thrift
1182 QEVERCLOUD_EXPORT extern const qint32 EDAM_DEVICE_DESCRIPTION_LEN_MAX;
1183
1184 // Limits.thrift
1188 QEVERCLOUD_EXPORT extern const QString EDAM_DEVICE_DESCRIPTION_REGEX;
1189
1190 // Limits.thrift
1194 QEVERCLOUD_EXPORT extern const qint32 EDAM_SEARCH_SUGGESTIONS_MAX;
1195
1196 // Limits.thrift
1200 QEVERCLOUD_EXPORT extern const qint32 EDAM_SEARCH_SUGGESTIONS_PREFIX_LEN_MAX;
1201
1202 // Limits.thrift
1206 QEVERCLOUD_EXPORT extern const qint32 EDAM_SEARCH_SUGGESTIONS_PREFIX_LEN_MIN;
1207
1208 // Limits.thrift
1212 QEVERCLOUD_EXPORT extern const qint32 EDAM_FIND_CONTACT_DEFAULT_MAX_RESULTS;
1213
1214 // Limits.thrift
1218 QEVERCLOUD_EXPORT extern const qint32 EDAM_FIND_CONTACT_MAX_RESULTS;
1219
1220 // Limits.thrift
1225 QEVERCLOUD_EXPORT extern const qint32 EDAM_NOTE_LOCK_VIEWERS_NOTES_MAX;
1226
1227 // Limits.thrift
1231 QEVERCLOUD_EXPORT extern const qint32 EDAM_GET_ORDERS_MAX_RESULTS;
1232
1233 // Limits.thrift
1237 QEVERCLOUD_EXPORT extern const qint32 EDAM_MESSAGE_BODY_LEN_MAX;
1238
1239 // Limits.thrift
1243 QEVERCLOUD_EXPORT extern const QString EDAM_MESSAGE_BODY_REGEX;
1244
1245 // Limits.thrift
1249 QEVERCLOUD_EXPORT extern const qint32 EDAM_MESSAGE_RECIPIENTS_MAX;
1250
1251 // Limits.thrift
1255 QEVERCLOUD_EXPORT extern const qint32 EDAM_MESSAGE_ATTACHMENTS_MAX;
1256
1257 // Limits.thrift
1261 QEVERCLOUD_EXPORT extern const qint32 EDAM_MESSAGE_ATTACHMENT_TITLE_LEN_MAX;
1262
1263 // Limits.thrift
1267 QEVERCLOUD_EXPORT extern const QString EDAM_MESSAGE_ATTACHMENT_TITLE_REGEX;
1268
1269 // Limits.thrift
1273 QEVERCLOUD_EXPORT extern const qint32 EDAM_MESSAGE_ATTACHMENT_SNIPPET_LEN_MAX;
1274
1275 // Limits.thrift
1279 QEVERCLOUD_EXPORT extern const QString EDAM_MESSAGE_ATTACHMENT_SNIPPET_REGEX;
1280
1281 // Limits.thrift
1286 QEVERCLOUD_EXPORT extern const qint32 EDAM_USER_PROFILE_PHOTO_MAX_BYTES;
1287
1288 // Limits.thrift
1292 QEVERCLOUD_EXPORT extern const qint32 EDAM_PROMOTION_ID_LEN_MAX;
1293
1294 // Limits.thrift
1298 QEVERCLOUD_EXPORT extern const QString EDAM_PROMOTION_ID_REGEX;
1299
1300 // Limits.thrift
1302 QEVERCLOUD_EXPORT extern const qint16 EDAM_APP_RATING_MIN;
1303
1304 // Limits.thrift
1305 QEVERCLOUD_EXPORT extern const qint16 EDAM_APP_RATING_MAX;
1306
1307 // Limits.thrift
1311 QEVERCLOUD_EXPORT extern const qint32 EDAM_SNIPPETS_NOTES_MAX;
1312
1313 // Limits.thrift
1317 QEVERCLOUD_EXPORT extern const qint32 EDAM_CONNECTED_IDENTITY_REQUEST_MAX;
1318
1319 // Limits.thrift
1324 QEVERCLOUD_EXPORT extern const qint32 EDAM_OPEN_ID_ACCESS_TOKEN_MAX;
1325
1326 // Types.thrift
1331 QEVERCLOUD_EXPORT extern const QString CLASSIFICATION_RECIPE_USER_NON_RECIPE;
1332
1333 // Types.thrift
1338 QEVERCLOUD_EXPORT extern const QString CLASSIFICATION_RECIPE_USER_RECIPE;
1339
1340 // Types.thrift
```

```

1345 QEVERCLOUD_EXPORT extern const QString CLASSIFICATION_RECIPE_SERVICE_RECIPE;
1346
1347 // Types.thrift
1352 QEVERCLOUD_EXPORT extern const QString EDAM_NOTE_SOURCE_WEB_CLIP;
1353
1354 // Types.thrift
1359 QEVERCLOUD_EXPORT extern const QString EDAM_NOTE_SOURCE_WEB_CLIP_SIMPLIFIED;
1360
1361 // Types.thrift
1366 QEVERCLOUD_EXPORT extern const QString EDAM_NOTE_SOURCE_MAIL_CLIP;
1367
1368 // Types.thrift
1373 QEVERCLOUD_EXPORT extern const QString EDAM_NOTE_SOURCE_MAIL_SMTP_GATEWAY;
1374
1375 // UserStore.thrift
1381 QEVERCLOUD_EXPORT extern const quint16 EDAM_VERSION_MAJOR;
1382
1383 // UserStore.thrift
1389 QEVERCLOUD_EXPORT extern const quint16 EDAM_VERSION_MINOR;
1390
1391 } // namespace qevercloud
1392
1393 #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_NUM`  
= 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_NOTEBOOK`  
= 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↵  
Code value)
- `QEVERCLOUD_EXPORT` QDebug & `qevercloud::operator<<` (QDebug &out, const EDAMErrorCode value)
- uint `qevercloud::qHash` (EDAMInvalidContactReason value)
- `QEVERCLOUD_EXPORT` QTextStream & `qevercloud::operator<<` (QTextStream &out, const EDAMInvalid↵  
ContactReason 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 SharedNotebookInstanceRestrictions value`)

- `QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const SharedNotebook↵ InstanceRestrictions value)`
- `uint qevercloud::qHash (ReminderEmailConfig value)`
- `QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const Reminder↵ EmailConfig value)`
- `QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const ReminderEmailConfig value)`
- `uint qevercloud::qHash (BusinessInvitationStatus value)`
- `QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const Business↵ InvitationStatus value)`
- `QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const BusinessInvitation↵ Status 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 Recipient↵ Status value)`
- `QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const RecipientStatus value)`
- `uint qevercloud::qHash (CanMoveToContainerStatus value)`
- `QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const CanMoveTo↵ ContainerStatus value)`
- `QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const CanMoveToContainer↵ Status value)`
- `uint qevercloud::qHash (RelatedContentType value)`
- `QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const Related↵ ContentType value)`
- `QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const RelatedContentType value)`
- `uint qevercloud::qHash (RelatedContentAccess value)`
- `QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const Related↵ ContentAccess value)`
- `QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const RelatedContentAccess value)`
- `uint qevercloud::qHash (UserIdentityType value)`
- `QEVERCLOUD_EXPORT QTextStream & qevercloud::operator<< (QTextStream &out, const UserIdentity↵ Type value)`
- `QEVERCLOUD_EXPORT QDebug & qevercloud::operator<< (QDebug &out, const UserIdentityType value)`

## 8.16 EDAMErrorCode.h

[Go to the documentation of this file.](#)

```

1
12 #ifndef QEVERCLOUD_GENERATED_EDAMERRORCODE_H
13 #define QEVERCLOUD_GENERATED_EDAMERRORCODE_H
14
15 #include "../Export.h"
16
17 #include "../Helpers.h"
18 #include <QDebug>
19 #include <QMetaType>
20 #include <QTextStream>

```

```

21
22 namespace qevercloud {
23
24
25
26 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
27 #if QEVERCLOUD_USES_Q_NAMESPACE
28 Q_NAMESPACE
29 #endif
30 #endif
31
32
33
34 enum class EDAMErrorCode
35 {
36     UNKNOWN = 1,
37     BAD_DATA_FORMAT = 2,
38     PERMISSION_DENIED = 3,
39     INTERNAL_ERROR = 4,
40     DATA_REQUIRED = 5,
41     LIMIT_REACHED = 6,
42     QUOTA_REACHED = 7,
43     INVALID_AUTH = 8,
44     AUTH_EXPIRED = 9,
45     DATA_CONFLICT = 10,
46     ENML_VALIDATION = 11,
47     SHARD_UNAVAILABLE = 12,
48     LEN_TOO_SHORT = 13,
49     LEN_TOO_LONG = 14,
50     TOO_FEW = 15,
51     TOO_MANY = 16,
52     UNSUPPORTED_OPERATION = 17,
53     TAKEN_DOWN = 18,
54     RATE_LIMIT_REACHED = 19,
55     BUSINESS_SECURITY_LOGIN_REQUIRED = 20,
56     DEVICE_LIMIT_REACHED = 21,
57     OPENID_ALREADY_TAKEN = 22,
58     INVALID_OPENID_TOKEN = 23,
59     USER_NOT_ASSOCIATED = 24,
60     USER_NOT_REGISTERED = 25,
61     USER_ALREADY_ASSOCIATED = 26,
62     ACCOUNT_CLEAR = 27,
63     SSO_AUTHENTICATION_REQUIRED = 28
64 };
65
66 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
67 #if QEVERCLOUD_USES_Q_NAMESPACE
68 Q_ENUM_NS(EDAMErrorCode)
69 #endif
70 #endif
71
72 inline uint qHash(EDAMErrorCode value)
73 {
74     return static_cast<uint>(value);
75 }
76
77
78 QEVERCLOUD_EXPORT QTextStream & operator<<(
79     QTextStream & out, const EDAMErrorCode value);
80
81
82 QEVERCLOUD_EXPORT QDebug & operator<<(
83     QDebug & out, const EDAMErrorCode value);
84
85
86 enum class EDAMInvalidContactReason
87 {
88     BAD_ADDRESS,
89     DUPLICATE_CONTACT,
90     NO_CONNECTION
91 };
92
93 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
94 #if QEVERCLOUD_USES_Q_NAMESPACE
95 Q_ENUM_NS(EDAMInvalidContactReason)
96 #endif
97 #endif
98
99 inline uint qHash(EDAMInvalidContactReason value)
100 {
101     return static_cast<uint>(value);
102 }
103
104
105
106 QEVERCLOUD_EXPORT QTextStream & operator<<(
107     QTextStream & out, const EDAMInvalidContactReason value);
108
109
110
111 QEVERCLOUD_EXPORT QDebug & operator<<(

```

```

226     QDebug & out, const EDAMInvalidContactReason value);
227
228
229
251 enum class ShareRelationshipPrivilegeLevel
252 {
253     READ_NOTEBOOK = 0,
254     READ_NOTEBOOK_PLUS_ACTIVITY = 10,
255     MODIFY_NOTEBOOK_PLUS_ACTIVITY = 20,
256     FULL_ACCESS = 30
257 };
258
259 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
260 #if QEVERCLOUD_USES_Q_NAMESPACE
261 Q_ENUM_NS(ShareRelationshipPrivilegeLevel)
262 #endif
263 #endif
264
265 inline uint qHash(ShareRelationshipPrivilegeLevel value)
266 {
267     return static_cast<uint>(value);
268 }
269
270
271
272 QEVERCLOUD_EXPORT QTextStream & operator<<(
273     QTextStream & out, const ShareRelationshipPrivilegeLevel value);
274
275
276
277 QEVERCLOUD_EXPORT QDebug & operator<<(
278     QDebug & out, const ShareRelationshipPrivilegeLevel value);
279
280
281
282 enum class PrivilegeLevel
283 {
284     NORMAL = 1,
285     PREMIUM = 3,
286     VIP = 5,
287     MANAGER = 7,
288     SUPPORT = 8,
289     ADMIN = 9
290 };
291
292 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
293 #if QEVERCLOUD_USES_Q_NAMESPACE
294 Q_ENUM_NS(PrivilegeLevel)
295 #endif
296 #endif
297
298 inline uint qHash(PrivilegeLevel value)
299 {
300     return static_cast<uint>(value);
301 }
302
303
304
305 QEVERCLOUD_EXPORT QTextStream & operator<<(
306     QTextStream & out, const PrivilegeLevel value);
307
308
309
310 QEVERCLOUD_EXPORT QDebug & operator<<(
311     QDebug & out, const PrivilegeLevel value);
312
313
314
315 enum class ServiceLevel
316 {
317     BASIC = 1,
318     PLUS = 2,
319     PREMIUM = 3,
320     BUSINESS = 4
321 };
322
323 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
324 #if QEVERCLOUD_USES_Q_NAMESPACE
325 Q_ENUM_NS(ServiceLevel)
326 #endif
327 #endif
328
329 inline uint qHash(ServiceLevel value)
330 {
331     return static_cast<uint>(value);
332 }
333
334
335
336 QEVERCLOUD_EXPORT QTextStream & operator<<(
337     QTextStream & out, const ServiceLevel value);
338
339
340
341 QEVERCLOUD_EXPORT QDebug & operator<<(
342     QDebug & out, const ServiceLevel value);

```

```
353
354
355
360 enum class QueryFormat
361 {
362     USER = 1,
363     SEXP = 2
364 };
365
366 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
367 #if QEVERCLOUD_USES_Q_NAMESPACE
368 Q_ENUM_NS(QueryFormat)
369 #endif
370 #endif
371
372 inline uint qHash(QueryFormat value)
373 {
374     return static_cast<uint>(value);
375 }
376
377
378
379 QEVERCLOUD_EXPORT QTextStream & operator<<(
380     QTextStream & out, const QueryFormat value);
381
382
383
384 QEVERCLOUD_EXPORT QDebug & operator<<(
385     QDebug & out, const QueryFormat value);
386
387
388
393 enum class NoteSortOrder
394 {
395     CREATED = 1,
396     UPDATED = 2,
397     RELEVANCE = 3,
398     UPDATE_SEQUENCE_NUMBER = 4,
399     TITLE = 5
400 };
401
402 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
403 #if QEVERCLOUD_USES_Q_NAMESPACE
404 Q_ENUM_NS(NoteSortOrder)
405 #endif
406 #endif
407
408 inline uint qHash(NoteSortOrder value)
409 {
410     return static_cast<uint>(value);
411 }
412
413
414
415 QEVERCLOUD_EXPORT QTextStream & operator<<(
416     QTextStream & out, const NoteSortOrder value);
417
418
419
420 QEVERCLOUD_EXPORT QDebug & operator<<(
421     QDebug & out, const NoteSortOrder value);
422
423
424
425 enum class PremiumOrderStatus
426 {
427     NONE = 0,
428     PENDING = 1,
429     ACTIVE = 2,
430     FAILED = 3,
431     CANCELLATION_PENDING = 4,
432     CANCELED = 5
433 };
434
435 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
436 #if QEVERCLOUD_USES_Q_NAMESPACE
437 Q_ENUM_NS(PremiumOrderStatus)
438 #endif
439 #endif
440
441 inline uint qHash(PremiumOrderStatus value)
442 {
443     return static_cast<uint>(value);
444 }
445
446
447
448 QEVERCLOUD_EXPORT QTextStream & operator<<(
449     QTextStream & out, const PremiumOrderStatus value);
450
451
452
453 QEVERCLOUD_EXPORT QDebug & operator<<(
454     QDebug & out, const PremiumOrderStatus value);
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
```

```

513 enum class SharedNotebookPrivilegeLevel
514 {
515     READ_NOTEBOOK = 0,
516     MODIFY_NOTEBOOK_PLUS_ACTIVITY = 1,
517     READ_NOTEBOOK_PLUS_ACTIVITY = 2,
518     GROUP = 3,
519     FULL_ACCESS = 4,
520     BUSINESS_FULL_ACCESS = 5
521 };
522
523 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
524 #if QEVERCLOUD_USES_Q_NAMESPACE
525 Q_ENUM_NS(SharedNotebookPrivilegeLevel)
526 #endif
527 #endif
528
529 inline uint qHash(SharedNotebookPrivilegeLevel value)
530 {
531     return static_cast<uint>(value);
532 }
533
534
535
536 QEVERCLOUD_EXPORT QTextStream & operator<<(
537     QTextStream & out, const SharedNotebookPrivilegeLevel value);
538
539
540
541 QEVERCLOUD_EXPORT QDebug & operator<<(
542     QDebug & out, const SharedNotebookPrivilegeLevel value);
543
544
545
546 enum class SharedNotePrivilegeLevel
547 {
548     READ_NOTE = 0,
549     MODIFY_NOTE = 1,
550     FULL_ACCESS = 2
551 };
552
553 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
554 #if QEVERCLOUD_USES_Q_NAMESPACE
555 Q_ENUM_NS(SharedNotePrivilegeLevel)
556 #endif
557 #endif
558
559 inline uint qHash(SharedNotePrivilegeLevel value)
560 {
561     return static_cast<uint>(value);
562 }
563
564
565
566 QEVERCLOUD_EXPORT QTextStream & operator<<(
567     QTextStream & out, const SharedNotePrivilegeLevel value);
568
569
570
571 QEVERCLOUD_EXPORT QDebug & operator<<(
572     QDebug & out, const SharedNotePrivilegeLevel value);
573
574
575
576 enum class SponsoredGroupRole
577 {
578     GROUP_MEMBER = 1,
579     GROUP_ADMIN = 2,
580     GROUP_OWNER = 3
581 };
582
583 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
584 #if QEVERCLOUD_USES_Q_NAMESPACE
585 Q_ENUM_NS(SponsoredGroupRole)
586 #endif
587 #endif
588
589 inline uint qHash(SponsoredGroupRole value)
590 {
591     return static_cast<uint>(value);
592 }
593
594
595
596 QEVERCLOUD_EXPORT QTextStream & operator<<(
597     QTextStream & out, const SponsoredGroupRole value);
598
599
600
601 QEVERCLOUD_EXPORT QDebug & operator<<(
602     QDebug & out, const SponsoredGroupRole value);
603
604
605
606 enum class BusinessUserRole
607 {
608     ADMIN = 1,

```

```

639     NORMAL = 2
640 };
641
642 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
643 #if QEVERCLOUD_USES_Q_NAMESPACE
644 Q_ENUM_NS(BusinessUserRole)
645 #endif
646 #endif
647
648 inline uint qHash(BusinessUserRole value)
649 {
650     return static_cast<uint>(value);
651 }
652
653
654
655 QEVERCLOUD_EXPORT QTextStream & operator<<(
656     QTextStream & out, const BusinessUserRole value);
657
658
659
660 QEVERCLOUD_EXPORT QDebug & operator<<(
661     QDebug & out, const BusinessUserRole value);
662
663
664
665 enum class BusinessUserStatus
666 {
667     ACTIVE = 1,
668     DEACTIVATED = 2
669 };
670
671 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
672 #if QEVERCLOUD_USES_Q_NAMESPACE
673 Q_ENUM_NS(BusinessUserStatus)
674 #endif
675 #endif
676
677 inline uint qHash(BusinessUserStatus value)
678 {
679     return static_cast<uint>(value);
680 }
681
682
683
684 QEVERCLOUD_EXPORT QTextStream & operator<<(
685     QTextStream & out, const BusinessUserStatus value);
686
687
688
689 QEVERCLOUD_EXPORT QDebug & operator<<(
690     QDebug & out, const BusinessUserStatus value);
691
692
693
694 enum class SharedNotebookInstanceRestrictions
695 {
696     ASSIGNED = 1,
697     NO_SHARED_NOTEBOOKS = 2
698 };
699
700 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
701 #if QEVERCLOUD_USES_Q_NAMESPACE
702 Q_ENUM_NS(SharedNotebookInstanceRestrictions)
703 #endif
704 #endif
705
706 inline uint qHash(SharedNotebookInstanceRestrictions value)
707 {
708     return static_cast<uint>(value);
709 }
710
711
712
713 QEVERCLOUD_EXPORT QTextStream & operator<<(
714     QTextStream & out, const SharedNotebookInstanceRestrictions value);
715
716
717
718 QEVERCLOUD_EXPORT QDebug & operator<<(
719     QDebug & out, const SharedNotebookInstanceRestrictions value);
720
721
722
723 enum class ReminderEmailConfig
724 {
725     DO_NOT_SEND = 1,
726     SEND_DAILY_EMAIL = 2
727 };
728
729 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
730 #if QEVERCLOUD_USES_Q_NAMESPACE
731 Q_ENUM_NS(ReminderEmailConfig)
732 #endif
733 #endif
734
735
736

```



```
768 inline uint qHash(ReminderEmailConfig value)
769 {
770     return static_cast<uint>(value);
771 }
772
773
774
775 QEVERCLOUD_EXPORT QTextStream & operator<<(
776     QTextStream & out, const ReminderEmailConfig value);
777
778
779
780 QEVERCLOUD_EXPORT QDebug & operator<<(
781     QDebug & out, const ReminderEmailConfig value);
782
783
784
785 enum class BusinessInvitationStatus
786 {
787     APPROVED = 0,
788     REQUESTED = 1,
789     REDEEMED = 2
790 };
791
792 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
793 #if QEVERCLOUD_USES_Q_NAMESPACE
794 Q_ENUM_NS(BusinessInvitationStatus)
795 #endif
796 #endif
797
798 inline uint qHash(BusinessInvitationStatus value)
799 {
800     return static_cast<uint>(value);
801 }
802
803
804
805 QEVERCLOUD_EXPORT QTextStream & operator<<(
806     QTextStream & out, const BusinessInvitationStatus value);
807
808
809
810 QEVERCLOUD_EXPORT QDebug & operator<<(
811     QDebug & out, const BusinessInvitationStatus value);
812
813
814
815 enum class ContactType
816 {
817     EVERNOTE = 1,
818     SMS = 2,
819     FACEBOOK = 3,
820     EMAIL = 4,
821     TWITTER = 5,
822     LINKEDIN = 6
823 };
824
825 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
826 #if QEVERCLOUD_USES_Q_NAMESPACE
827 Q_ENUM_NS(ContactType)
828 #endif
829 #endif
830
831 inline uint qHash(ContactType value)
832 {
833     return static_cast<uint>(value);
834 }
835
836
837
838 QEVERCLOUD_EXPORT QTextStream & operator<<(
839     QTextStream & out, const ContactType value);
840
841
842
843 QEVERCLOUD_EXPORT QDebug & operator<<(
844     QDebug & out, const ContactType value);
845
846
847
848 enum class EntityType
849 {
850     NOTE = 1,
851     NOTEBOOK = 2,
852     WORKSPACE = 3
853 };
854
855 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
856 #if QEVERCLOUD_USES_Q_NAMESPACE
857 Q_ENUM_NS(EntityType)
858 #endif
859 #endif
860
861 inline uint qHash(EntityType value)
862 {
863     return static_cast<uint>(value);
864 }
```

```

883 }
884
886
887 QEVERCLOUD_EXPORT QTextStream & operator<<(
888     QTextStream & out, const EntityType value);
889
891
892 QEVERCLOUD_EXPORT QDebug & operator<<(
893     QDebug & out, const EntityType value);
894
896
911 enum class RecipientStatus
912 {
913     NOT_IN_MY_LIST = 1,
914     IN_MY_LIST = 2,
915     IN_MY_LIST_AND_DEFAULT_NOTEBOOK = 3
916 };
917
918 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
919 #if QEVERCLOUD_USES_Q_NAMESPACE
920 Q_ENUM_NS(RecipientStatus)
921 #endif
922 #endif
923
924 inline uint qHash(RecipientStatus value)
925 {
926     return static_cast<uint>(value);
927 }
928
930
931 QEVERCLOUD_EXPORT QTextStream & operator<<(
932     QTextStream & out, const RecipientStatus value);
933
935
936 QEVERCLOUD_EXPORT QDebug & operator<<(
937     QDebug & out, const RecipientStatus value);
938
940
959 enum class CanMoveToContainerStatus
960 {
961     CAN_BE_MOVED = 1,
962     INSUFFICIENT_ENTITY_PRIVILEGE = 2,
963     INSUFFICIENT_CONTAINER_PRIVILEGE = 3
964 };
965
966 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
967 #if QEVERCLOUD_USES_Q_NAMESPACE
968 Q_ENUM_NS(CanMoveToContainerStatus)
969 #endif
970 #endif
971
972 inline uint qHash(CanMoveToContainerStatus value)
973 {
974     return static_cast<uint>(value);
975 }
976
978
979 QEVERCLOUD_EXPORT QTextStream & operator<<(
980     QTextStream & out, const CanMoveToContainerStatus value);
981
983
984 QEVERCLOUD_EXPORT QDebug & operator<<(
985     QDebug & out, const CanMoveToContainerStatus value);
986
988
997 enum class RelatedContentType
998 {
999     NEWS_ARTICLE = 1,
1000     PROFILE_PERSON = 2,
1001     PROFILE_ORGANIZATION = 3,
1002     REFERENCE_MATERIAL = 4
1003 };
1004
1005 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
1006 #if QEVERCLOUD_USES_Q_NAMESPACE
1007 Q_ENUM_NS(RelatedContentType)
1008 #endif
1009 #endif
1010
1011 inline uint qHash(RelatedContentType value)
1012 {
1013     return static_cast<uint>(value);
1014 }
1015
1017
1018 QEVERCLOUD_EXPORT QTextStream & operator<<(
1019     QTextStream & out, const RelatedContentType value);

```

```

1020
1022
1023 QEVERCLOUD_EXPORT QDebug & operator«(
1024     QDebug & out, const RelatedContentType value);
1025
1027
1047 enum class RelatedContentAccess
1048 {
1049     NOT_ACCESSIBLE = 0,
1050     DIRECT_LINK_ACCESS_OK = 1,
1051     DIRECT_LINK_LOGIN_REQUIRED = 2,
1052     DIRECT_LINK_EMBEDDED_VIEW = 3
1053 };
1054
1055 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
1056 #if QEVERCLOUD_USES_Q_NAMESPACE
1057 Q_ENUM_NS(RelatedContentAccess)
1058 #endif
1059 #endif
1060
1061 inline uint qHash(RelatedContentAccess value)
1062 {
1063     return static_cast<uint>(value);
1064 }
1065
1067
1068 QEVERCLOUD_EXPORT QTextStream & operator«(
1069     QTextStream & out, const RelatedContentAccess value);
1070
1072
1073 QEVERCLOUD_EXPORT QDebug & operator«(
1074     QDebug & out, const RelatedContentAccess value);
1075
1077
1081 enum class UserIdentityType
1082 {
1083     EVERNOTE_USERID = 1,
1084     EMAIL = 2,
1085     IDENTITYID = 3
1086 };
1087
1088 #if QT_VERSION >= QT_VERSION_CHECK(5, 8, 0)
1089 #if QEVERCLOUD_USES_Q_NAMESPACE
1090 Q_ENUM_NS(UserIdentityType)
1091 #endif
1092 #endif
1093
1094 inline uint qHash(UserIdentityType value)
1095 {
1096     return static_cast<uint>(value);
1097 }
1098
1100
1101 QEVERCLOUD_EXPORT QTextStream & operator«(
1102     QTextStream & out, const UserIdentityType value);
1103
1105
1106 QEVERCLOUD_EXPORT QDebug & operator«(
1107     QDebug & out, const UserIdentityType value);
1108
1109 } // namespace qevercloud
1110
1111 Q_DECLARE_METATYPE(qevercloud::EDAMErrorCode)
1112 Q_DECLARE_METATYPE(qevercloud::EDAMInvalidContactReason)
1113 Q_DECLARE_METATYPE(qevercloud::ShareRelationshipPrivilegeLevel)
1114 Q_DECLARE_METATYPE(qevercloud::PrivilegeLevel)
1115 Q_DECLARE_METATYPE(qevercloud::ServiceLevel)
1116 Q_DECLARE_METATYPE(qevercloud::QueryFormat)
1117 Q_DECLARE_METATYPE(qevercloud::NoteSortOrder)
1118 Q_DECLARE_METATYPE(qevercloud::PremiumOrderStatus)
1119 Q_DECLARE_METATYPE(qevercloud::SharedNotebookPrivilegeLevel)
1120 Q_DECLARE_METATYPE(qevercloud::SharedNotePrivilegeLevel)
1121 Q_DECLARE_METATYPE(qevercloud::SponsoredGroupRole)
1122 Q_DECLARE_METATYPE(qevercloud::BusinessUserRole)
1123 Q_DECLARE_METATYPE(qevercloud::BusinessUserStatus)
1124 Q_DECLARE_METATYPE(qevercloud::SharedNotebookInstanceRestrictions)
1125 Q_DECLARE_METATYPE(qevercloud::ReminderEmailConfig)
1126 Q_DECLARE_METATYPE(qevercloud::BusinessInvitationStatus)
1127 Q_DECLARE_METATYPE(qevercloud::ContactType)
1128 Q_DECLARE_METATYPE(qevercloud::EntityType)
1129 Q_DECLARE_METATYPE(qevercloud::RecipientStatus)
1130 Q_DECLARE_METATYPE(qevercloud::CanMoveToContainerStatus)
1131 Q_DECLARE_METATYPE(qevercloud::RelatedContentType)
1132 Q_DECLARE_METATYPE(qevercloud::RelatedContentAccess)
1133 Q_DECLARE_METATYPE(qevercloud::UserIdentityType)
1134
1135 #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.](#)

```
1
12 #ifndef QEVERCLOUD_GENERATED_SERVERS_H
13 #define QEVERCLOUD_GENERATED_SERVERS_H
14
15 #include "../Export.h"
16
17 #include "../Optional.h"
18 #include "../RequestContext.h"
19 #include "Constants.h"
20 #include "Types.h"
21 #include <QObject>
22 #include <functional>
23
24 namespace qevercloud {
25
26
27 class QEVERCLOUD_EXPORT NoteStoreServer: public QObject
28 {
29     Q_OBJECT
30     Q_DISABLE_COPY(NoteStoreServer)
31 public:
32     explicit NoteStoreServer(QObject * parent = nullptr);
33
34 Q_SIGNALS:
35     // Signals notifying listeners about incoming requests
36     void getSyncStateRequest(
37         IRequestContextPtr ctx);
38
39     void getFilteredSyncChunkRequest(
40         qint32 afterUSN,
41         qint32 maxEntries,
42         SyncChunkFilter filter,
43         IRequestContextPtr ctx);
44
45     void getLinkedNotebookSyncStateRequest(
46         LinkedNotebook linkedNotebook,
47         IRequestContextPtr ctx);
48 }
```

```
54
55 void getLinkedNotebookSyncChunkRequest (
56     LinkedNotebook linkedNotebook,
57     qint32 afterUSN,
58     qint32 maxEntries,
59     bool fullSyncOnly,
60     IRequestContextPtr ctx);
61
62 void listNotebooksRequest (
63     IRequestContextPtr ctx);
64
65 void listAccessibleBusinessNotebooksRequest (
66     IRequestContextPtr ctx);
67
68 void getNotebookRequest (
69     Guid guid,
70     IRequestContextPtr ctx);
71
72 void getDefaultNotebookRequest (
73     IRequestContextPtr ctx);
74
75 void createNotebookRequest (
76     Notebook notebook,
77     IRequestContextPtr ctx);
78
79 void updateNotebookRequest (
80     Notebook notebook,
81     IRequestContextPtr ctx);
82
83 void expungeNotebookRequest (
84     Guid guid,
85     IRequestContextPtr ctx);
86
87 void listTagsRequest (
88     IRequestContextPtr ctx);
89
90 void listTagsByNotebookRequest (
91     Guid notebookGuid,
92     IRequestContextPtr ctx);
93
94 void getTagRequest (
95     Guid guid,
96     IRequestContextPtr ctx);
97
98 void createTagRequest (
99     Tag tag,
100     IRequestContextPtr ctx);
101
102 void updateTagRequest (
103     Tag tag,
104     IRequestContextPtr ctx);
105
106 void untagAllRequest (
107     Guid guid,
108     IRequestContextPtr ctx);
109
110 void expungeTagRequest (
111     Guid guid,
112     IRequestContextPtr ctx);
113
114 void listSearchesRequest (
115     IRequestContextPtr ctx);
116
117 void getSearchRequest (
118     Guid guid,
119     IRequestContextPtr ctx);
120
121 void createSearchRequest (
122     SavedSearch search,
123     IRequestContextPtr ctx);
124
125 void updateSearchRequest (
126     SavedSearch search,
127     IRequestContextPtr ctx);
128
129 void expungeSearchRequest (
130     Guid guid,
131     IRequestContextPtr ctx);
132
133 void findNoteOffsetRequest (
134     NoteFilter filter,
135     Guid guid,
136     IRequestContextPtr ctx);
137
138 void findNotesMetadataRequest (
139     NoteFilter filter,
140     qint32 offset,
```

```
141         qint32 maxNotes,
142         NotesMetadataResultSpec resultSpec,
143         IRequestContextPtr ctx);
144
145 void findNoteCountsRequest (
146     NoteFilter filter,
147     bool withTrash,
148     IRequestContextPtr ctx);
149
150 void getNoteWithResultSpecRequest (
151     Guid guid,
152     NoteResultSpec resultSpec,
153     IRequestContextPtr ctx);
154
155 void getNoteRequest (
156     Guid guid,
157     bool withContent,
158     bool withResourcesData,
159     bool withResourcesRecognition,
160     bool withResourcesAlternateData,
161     IRequestContextPtr ctx);
162
163 void getNoteApplicationDataRequest (
164     Guid guid,
165     IRequestContextPtr ctx);
166
167 void getNoteApplicationDataEntryRequest (
168     Guid guid,
169     QString key,
170     IRequestContextPtr ctx);
171
172 void setNoteApplicationDataEntryRequest (
173     Guid guid,
174     QString key,
175     QString value,
176     IRequestContextPtr ctx);
177
178 void unsetNoteApplicationDataEntryRequest (
179     Guid guid,
180     QString key,
181     IRequestContextPtr ctx);
182
183 void getNoteContentRequest (
184     Guid guid,
185     IRequestContextPtr ctx);
186
187 void getNoteSearchTextRequest (
188     Guid guid,
189     bool noteOnly,
190     bool tokenizeForIndexing,
191     IRequestContextPtr ctx);
192
193 void getResourceSearchTextRequest (
194     Guid guid,
195     IRequestContextPtr ctx);
196
197 void getNoteTagNamesRequest (
198     Guid guid,
199     IRequestContextPtr ctx);
200
201 void createNoteRequest (
202     Note note,
203     IRequestContextPtr ctx);
204
205 void updateNoteRequest (
206     Note note,
207     IRequestContextPtr ctx);
208
209 void deleteNoteRequest (
210     Guid guid,
211     IRequestContextPtr ctx);
212
213 void expungeNoteRequest (
214     Guid guid,
215     IRequestContextPtr ctx);
216
217 void copyNoteRequest (
218     Guid noteGuid,
219     Guid toNotebookGuid,
220     IRequestContextPtr ctx);
221
222 void listNoteVersionsRequest (
223     Guid noteGuid,
224     IRequestContextPtr ctx);
225
226 void getNoteVersionRequest (
227     Guid noteGuid,
```

```
228         qint32 updateSequenceNum,
229         bool withResourcesData,
230         bool withResourcesRecognition,
231         bool withResourcesAlternateData,
232         IRequestContextPtr ctx);
233
234     void getResourceRequest (
235         Guid guid,
236         bool withData,
237         bool withRecognition,
238         bool withAttributes,
239         bool withAlternateData,
240         IRequestContextPtr ctx);
241
242     void getResourceApplicationDataRequest (
243         Guid guid,
244         IRequestContextPtr ctx);
245
246     void getResourceApplicationDataEntryRequest (
247         Guid guid,
248         QString key,
249         IRequestContextPtr ctx);
250
251     void setResourceApplicationDataEntryRequest (
252         Guid guid,
253         QString key,
254         QString value,
255         IRequestContextPtr ctx);
256
257     void unsetResourceApplicationDataEntryRequest (
258         Guid guid,
259         QString key,
260         IRequestContextPtr ctx);
261
262     void updateResourceRequest (
263         Resource resource,
264         IRequestContextPtr ctx);
265
266     void getResourceDataRequest (
267         Guid guid,
268         IRequestContextPtr ctx);
269
270     void getResourceByHashRequest (
271         Guid noteGuid,
272         QByteArray contentHash,
273         bool withData,
274         bool withRecognition,
275         bool withAlternateData,
276         IRequestContextPtr ctx);
277
278     void getResourceRecognitionRequest (
279         Guid guid,
280         IRequestContextPtr ctx);
281
282     void getResourceAlternateDataRequest (
283         Guid guid,
284         IRequestContextPtr ctx);
285
286     void getResourceAttributesRequest (
287         Guid guid,
288         IRequestContextPtr ctx);
289
290     void getPublicNotebookRequest (
291         UserID userId,
292         QString publicUri,
293         IRequestContextPtr ctx);
294
295     void shareNotebookRequest (
296         SharedNotebook sharedNotebook,
297         QString message,
298         IRequestContextPtr ctx);
299
300     void createOrUpdateNotebookSharesRequest (
301         NotebookShareTemplate shareTemplate,
302         IRequestContextPtr ctx);
303
304     void updateSharedNotebookRequest (
305         SharedNotebook sharedNotebook,
306         IRequestContextPtr ctx);
307
308     void setNotebookRecipientSettingsRequest (
309         QString notebookGuid,
310         NotebookRecipientSettings recipientSettings,
311         IRequestContextPtr ctx);
312
313     void listSharedNotebooksRequest (
314         IRequestContextPtr ctx);
```

```

315
316 void createLinkedNotebookRequest (
317     LinkedNotebook linkedNotebook,
318     IRequestContextPtr ctx);
319
320 void updateLinkedNotebookRequest (
321     LinkedNotebook linkedNotebook,
322     IRequestContextPtr ctx);
323
324 void listLinkedNotebooksRequest (
325     IRequestContextPtr ctx);
326
327 void expungeLinkedNotebookRequest (
328     Guid guid,
329     IRequestContextPtr ctx);
330
331 void authenticateToSharedNotebookRequest (
332     QString shareKeyOrGlobalId,
333     IRequestContextPtr ctx);
334
335 void getSharedNotebookByAuthRequest (
336     IRequestContextPtr ctx);
337
338 void emailNoteRequest (
339     NoteEmailParameters parameters,
340     IRequestContextPtr ctx);
341
342 void shareNoteRequest (
343     Guid guid,
344     IRequestContextPtr ctx);
345
346 void stopSharingNoteRequest (
347     Guid guid,
348     IRequestContextPtr ctx);
349
350 void authenticateToSharedNoteRequest (
351     QString guid,
352     QString noteKey,
353     IRequestContextPtr ctx);
354
355 void findRelatedRequest (
356     RelatedQuery query,
357     RelatedResultSpec resultSpec,
358     IRequestContextPtr ctx);
359
360 void updateNoteIfUsnMatchesRequest (
361     Note note,
362     IRequestContextPtr ctx);
363
364 void manageNotebookSharesRequest (
365     ManageNotebookSharesParameters parameters,
366     IRequestContextPtr ctx);
367
368 void getNotebookSharesRequest (
369     QString notebookGuid,
370     IRequestContextPtr ctx);
371
372 // Signals used to send encoded response data
373 void getSyncStateRequestReady (
374     QByteArray data);
375
376 void getFilteredSyncChunkRequestReady (
377     QByteArray data);
378
379 void getLinkedNotebookSyncStateRequestReady (
380     QByteArray data);
381
382 void getLinkedNotebookSyncChunkRequestReady (
383     QByteArray data);
384
385 void listNotebooksRequestReady (
386     QByteArray data);
387
388 void listAccessibleBusinessNotebooksRequestReady (
389     QByteArray data);
390
391 void getNotebookRequestReady (
392     QByteArray data);
393
394 void getDefaultNotebookRequestReady (
395     QByteArray data);
396
397 void createNotebookRequestReady (
398     QByteArray data);
399
400 void updateNotebookRequestReady (
401     QByteArray data);

```



```
402
403     void expungeNotebookRequestReady (
404         QByteArray data);
405
406     void listTagsRequestReady (
407         QByteArray data);
408
409     void listTagsByNotebookRequestReady (
410         QByteArray data);
411
412     void getTagRequestReady (
413         QByteArray data);
414
415     void createTagRequestReady (
416         QByteArray data);
417
418     void updateTagRequestReady (
419         QByteArray data);
420
421     void untagAllRequestReady (
422         QByteArray data);
423
424     void expungeTagRequestReady (
425         QByteArray data);
426
427     void listSearchesRequestReady (
428         QByteArray data);
429
430     void getSearchRequestReady (
431         QByteArray data);
432
433     void createSearchRequestReady (
434         QByteArray data);
435
436     void updateSearchRequestReady (
437         QByteArray data);
438
439     void expungeSearchRequestReady (
440         QByteArray data);
441
442     void findNoteOffsetRequestReady (
443         QByteArray data);
444
445     void findNotesMetadataRequestReady (
446         QByteArray data);
447
448     void findNoteCountsRequestReady (
449         QByteArray data);
450
451     void getNoteWithResultSpecRequestReady (
452         QByteArray data);
453
454     void getNoteRequestReady (
455         QByteArray data);
456
457     void getNoteApplicationDataRequestReady (
458         QByteArray data);
459
460     void getNoteApplicationDataEntryRequestReady (
461         QByteArray data);
462
463     void setNoteApplicationDataEntryRequestReady (
464         QByteArray data);
465
466     void unsetNoteApplicationDataEntryRequestReady (
467         QByteArray data);
468
469     void getNoteContentRequestReady (
470         QByteArray data);
471
472     void getNoteSearchTextRequestReady (
473         QByteArray data);
474
475     void getResourceSearchTextRequestReady (
476         QByteArray data);
477
478     void getNoteTagNamesRequestReady (
479         QByteArray data);
480
481     void createNoteRequestReady (
482         QByteArray data);
483
484     void updateNoteRequestReady (
485         QByteArray data);
486
487     void deleteNoteRequestReady (
488         QByteArray data);
```

```
489
490 void expungeNoteRequestReady (
491     QByteArray data);
492
493 void copyNoteRequestReady (
494     QByteArray data);
495
496 void listNoteVersionsRequestReady (
497     QByteArray data);
498
499 void getNoteVersionRequestReady (
500     QByteArray data);
501
502 void getResourceRequestReady (
503     QByteArray data);
504
505 void getResourceApplicationDataRequestReady (
506     QByteArray data);
507
508 void getResourceApplicationDataEntryRequestReady (
509     QByteArray data);
510
511 void setResourceApplicationDataEntryRequestReady (
512     QByteArray data);
513
514 void unsetResourceApplicationDataEntryRequestReady (
515     QByteArray data);
516
517 void updateResourceRequestReady (
518     QByteArray data);
519
520 void getResourceDataRequestReady (
521     QByteArray data);
522
523 void getResourceByHashRequestReady (
524     QByteArray data);
525
526 void getResourceRecognitionRequestReady (
527     QByteArray data);
528
529 void getResourceAlternateDataRequestReady (
530     QByteArray data);
531
532 void getResourceAttributesRequestReady (
533     QByteArray data);
534
535 void getPublicNotebookRequestReady (
536     QByteArray data);
537
538 void shareNotebookRequestReady (
539     QByteArray data);
540
541 void createOrUpdateNotebookSharesRequestReady (
542     QByteArray data);
543
544 void updateSharedNotebookRequestReady (
545     QByteArray data);
546
547 void setNotebookRecipientSettingsRequestReady (
548     QByteArray data);
549
550 void listSharedNotebooksRequestReady (
551     QByteArray data);
552
553 void createLinkedNotebookRequestReady (
554     QByteArray data);
555
556 void updateLinkedNotebookRequestReady (
557     QByteArray data);
558
559 void listLinkedNotebooksRequestReady (
560     QByteArray data);
561
562 void expungeLinkedNotebookRequestReady (
563     QByteArray data);
564
565 void authenticateToSharedNotebookRequestReady (
566     QByteArray data);
567
568 void getSharedNotebookByAuthRequestReady (
569     QByteArray data);
570
571 void emailNoteRequestReady (
572     QByteArray data);
573
574 void shareNoteRequestReady (
575     QByteArray data);
```

```

576
577 void stopSharingNoteRequestReady(
578     QByteArray data);
579
580 void authenticateToSharedNoteRequestReady(
581     QByteArray data);
582
583 void findRelatedRequestReady(
584     QByteArray data);
585
586 void updateNoteIfUsnMatchesRequestReady(
587     QByteArray data);
588
589 void manageNotebookSharesRequestReady(
590     QByteArray data);
591
592 void getNotebookSharesRequestReady(
593     QByteArray data);
594
595 public Q_SLOTS:
596     // Slot used to deliver requests to the server
597     void onRequest(QByteArray data);
598
599     // Slots for replies to requests
600     void onGetSyncStateRequestReady(
601         SyncState value,
602         EverCloudExceptionDataPtr exceptionData);
603
604     void onGetFilteredSyncChunkRequestReady(
605         SyncChunk value,
606         EverCloudExceptionDataPtr exceptionData);
607
608     void onGetLinkedNotebookSyncStateRequestReady(
609         SyncState value,
610         EverCloudExceptionDataPtr exceptionData);
611
612     void onGetLinkedNotebookSyncChunkRequestReady(
613         SyncChunk value,
614         EverCloudExceptionDataPtr exceptionData);
615
616     void onListNotebooksRequestReady(
617         QList<Notebook> value,
618         EverCloudExceptionDataPtr exceptionData);
619
620     void onListAccessibleBusinessNotebooksRequestReady(
621         QList<Notebook> value,
622         EverCloudExceptionDataPtr exceptionData);
623
624     void onGetNotebookRequestReady(
625         Notebook value,
626         EverCloudExceptionDataPtr exceptionData);
627
628     void onGetDefaultNotebookRequestReady(
629         Notebook value,
630         EverCloudExceptionDataPtr exceptionData);
631
632     void onCreateNotebookRequestReady(
633         Notebook value,
634         EverCloudExceptionDataPtr exceptionData);
635
636     void onUpdateNotebookRequestReady(
637         qint32 value,
638         EverCloudExceptionDataPtr exceptionData);
639
640     void onExpungeNotebookRequestReady(
641         qint32 value,
642         EverCloudExceptionDataPtr exceptionData);
643
644     void onListTagsRequestReady(
645         QList<Tag> value,
646         EverCloudExceptionDataPtr exceptionData);
647
648     void onListTagsByNotebookRequestReady(
649         QList<Tag> value,
650         EverCloudExceptionDataPtr exceptionData);
651
652     void onGetTagRequestReady(
653         Tag value,
654         EverCloudExceptionDataPtr exceptionData);
655
656     void onCreateTagRequestReady(
657         Tag value,
658         EverCloudExceptionDataPtr exceptionData);
659
660     void onUpdateTagRequestReady(
661         qint32 value,
662         EverCloudExceptionDataPtr exceptionData);

```

```
663
664 void onUntagAllRequestReady (
665     EverCloudExceptionDataPtr exceptionData);
666
667 void onExpungeTagRequestReady (
668     qint32 value,
669     EverCloudExceptionDataPtr exceptionData);
670
671 void onListSearchesRequestReady (
672     QList<SavedSearch> value,
673     EverCloudExceptionDataPtr exceptionData);
674
675 void onGetSearchRequestReady (
676     SavedSearch value,
677     EverCloudExceptionDataPtr exceptionData);
678
679 void onCreateSearchRequestReady (
680     SavedSearch value,
681     EverCloudExceptionDataPtr exceptionData);
682
683 void onUpdateSearchRequestReady (
684     qint32 value,
685     EverCloudExceptionDataPtr exceptionData);
686
687 void onExpungeSearchRequestReady (
688     qint32 value,
689     EverCloudExceptionDataPtr exceptionData);
690
691 void onFindNoteOffsetRequestReady (
692     qint32 value,
693     EverCloudExceptionDataPtr exceptionData);
694
695 void onFindNotesMetadataRequestReady (
696     NotesMetadataList value,
697     EverCloudExceptionDataPtr exceptionData);
698
699 void onFindNoteCountsRequestReady (
700     NoteCollectionCounts value,
701     EverCloudExceptionDataPtr exceptionData);
702
703 void onGetNoteWithResultSpecRequestReady (
704     Note value,
705     EverCloudExceptionDataPtr exceptionData);
706
707 void onGetNoteRequestReady (
708     Note value,
709     EverCloudExceptionDataPtr exceptionData);
710
711 void onGetNoteApplicationDataRequestReady (
712     LazyMap value,
713     EverCloudExceptionDataPtr exceptionData);
714
715 void onGetNoteApplicationDataEntryRequestReady (
716     QString value,
717     EverCloudExceptionDataPtr exceptionData);
718
719 void onSetNoteApplicationDataEntryRequestReady (
720     qint32 value,
721     EverCloudExceptionDataPtr exceptionData);
722
723 void onUnsetNoteApplicationDataEntryRequestReady (
724     qint32 value,
725     EverCloudExceptionDataPtr exceptionData);
726
727 void onGetNoteContentRequestReady (
728     QString value,
729     EverCloudExceptionDataPtr exceptionData);
730
731 void onGetNoteSearchTextRequestReady (
732     QString value,
733     EverCloudExceptionDataPtr exceptionData);
734
735 void onGetResourceSearchTextRequestReady (
736     QString value,
737     EverCloudExceptionDataPtr exceptionData);
738
739 void onGetNoteTagNamesRequestReady (
740     QStringList value,
741     EverCloudExceptionDataPtr exceptionData);
742
743 void onCreateNoteRequestReady (
744     Note value,
745     EverCloudExceptionDataPtr exceptionData);
746
747 void onUpdateNoteRequestReady (
748     Note value,
749     EverCloudExceptionDataPtr exceptionData);
```

```

750
751 void onDeleteNoteRequestReady (
752     qint32 value,
753     EverCloudExceptionDataPtr exceptionData);
754
755 void onExpungeNoteRequestReady (
756     qint32 value,
757     EverCloudExceptionDataPtr exceptionData);
758
759 void onCopyNoteRequestReady (
760     Note value,
761     EverCloudExceptionDataPtr exceptionData);
762
763 void onListNoteVersionsRequestReady (
764     QList<NoteVersionId> value,
765     EverCloudExceptionDataPtr exceptionData);
766
767 void onGetNoteVersionRequestReady (
768     Note value,
769     EverCloudExceptionDataPtr exceptionData);
770
771 void onGetResourceRequestReady (
772     Resource value,
773     EverCloudExceptionDataPtr exceptionData);
774
775 void onGetResourceApplicationDataRequestReady (
776     LazyMap value,
777     EverCloudExceptionDataPtr exceptionData);
778
779 void onGetResourceApplicationDataEntryRequestReady (
780     QString value,
781     EverCloudExceptionDataPtr exceptionData);
782
783 void onSetResourceApplicationDataEntryRequestReady (
784     qint32 value,
785     EverCloudExceptionDataPtr exceptionData);
786
787 void onUnsetResourceApplicationDataEntryRequestReady (
788     qint32 value,
789     EverCloudExceptionDataPtr exceptionData);
790
791 void onUpdateResourceRequestReady (
792     qint32 value,
793     EverCloudExceptionDataPtr exceptionData);
794
795 void onGetResourceDataRequestReady (
796     QByteArray value,
797     EverCloudExceptionDataPtr exceptionData);
798
799 void onGetResourceByHashRequestReady (
800     Resource value,
801     EverCloudExceptionDataPtr exceptionData);
802
803 void onGetResourceRecognitionRequestReady (
804     QByteArray value,
805     EverCloudExceptionDataPtr exceptionData);
806
807 void onGetResourceAlternateDataRequestReady (
808     QByteArray value,
809     EverCloudExceptionDataPtr exceptionData);
810
811 void onGetResourceAttributesRequestReady (
812     ResourceAttributes value,
813     EverCloudExceptionDataPtr exceptionData);
814
815 void onGetPublicNotebookRequestReady (
816     Notebook value,
817     EverCloudExceptionDataPtr exceptionData);
818
819 void onShareNotebookRequestReady (
820     SharedNotebook value,
821     EverCloudExceptionDataPtr exceptionData);
822
823 void onCreateOrUpdateNotebookSharesRequestReady (
824     CreateOrUpdateNotebookSharesResult value,
825     EverCloudExceptionDataPtr exceptionData);
826
827 void onUpdateSharedNotebookRequestReady (
828     qint32 value,
829     EverCloudExceptionDataPtr exceptionData);
830
831 void onSetNotebookRecipientSettingsRequestReady (
832     Notebook value,
833     EverCloudExceptionDataPtr exceptionData);
834
835 void onListSharedNotebooksRequestReady (
836     QList<SharedNotebook> value,

```

```

837     EverCloudExceptionDataPtr exceptionData);
838
839 void onCreateLinkedNotebookRequestReady(
840     LinkedNotebook value,
841     EverCloudExceptionDataPtr exceptionData);
842
843 void onUpdateLinkedNotebookRequestReady(
844     qint32 value,
845     EverCloudExceptionDataPtr exceptionData);
846
847 void onListLinkedNotebooksRequestReady(
848     QList<LinkedNotebook> value,
849     EverCloudExceptionDataPtr exceptionData);
850
851 void onExpungeLinkedNotebookRequestReady(
852     qint32 value,
853     EverCloudExceptionDataPtr exceptionData);
854
855 void onAuthenticateToSharedNotebookRequestReady(
856     AuthenticationResult value,
857     EverCloudExceptionDataPtr exceptionData);
858
859 void onGetSharedNotebookByAuthRequestReady(
860     SharedNotebook value,
861     EverCloudExceptionDataPtr exceptionData);
862
863 void onEmailNoteRequestReady(
864     EverCloudExceptionDataPtr exceptionData);
865
866 void onShareNoteRequestReady(
867     QString value,
868     EverCloudExceptionDataPtr exceptionData);
869
870 void onStopSharingNoteRequestReady(
871     EverCloudExceptionDataPtr exceptionData);
872
873 void onAuthenticateToSharedNoteRequestReady(
874     AuthenticationResult value,
875     EverCloudExceptionDataPtr exceptionData);
876
877 void onFindRelatedRequestReady(
878     RelatedResult value,
879     EverCloudExceptionDataPtr exceptionData);
880
881 void onUpdateNoteIfUsnMatchesRequestReady(
882     UpdateNoteIfUsnMatchesResult value,
883     EverCloudExceptionDataPtr exceptionData);
884
885 void onManageNotebookSharesRequestReady(
886     ManageNotebookSharesResult value,
887     EverCloudExceptionDataPtr exceptionData);
888
889 void onGetNotebookSharesRequestReady(
890     ShareRelationships value,
891     EverCloudExceptionDataPtr exceptionData);
892
893 };
894
895
896 class QEVERCLOUD_EXPORT UserStoreServer: public QObject
897 {
898     Q_OBJECT
899     Q_DISABLE_COPY(UserStoreServer)
900 public:
901     explicit UserStoreServer(QObject * parent = nullptr);
902
903     Q_SIGNALS:
904         // Signals notifying listeners about incoming requests
905     void checkVersionRequest(
906         QString clientName,
907         qint16 edamVersionMajor,
908         qint16 edamVersionMinor,
909         IRequestContextPtr ctx);
910
911     void getBootstrapInfoRequest(
912         QString locale,
913         IRequestContextPtr ctx);
914
915     void authenticateLongSessionRequest(
916         QString username,
917         QString password,
918         QString consumerKey,
919         QString consumerSecret,
920         QString deviceIdIdentifier,
921         QString deviceDescription,
922         bool supportsTwoFactor,
923         IRequestContextPtr ctx);

```

```

930
931 void completeTwoFactorAuthenticationRequest (
932     QString oneTimeCode,
933     QString deviceIdentifier,
934     QString deviceDescription,
935     IRequestContextPtr ctx);
936
937 void revokeLongSessionRequest (
938     IRequestContextPtr ctx);
939
940 void authenticateToBusinessRequest (
941     IRequestContextPtr ctx);
942
943 void getUserRequest (
944     IRequestContextPtr ctx);
945
946 void getPublicUserInfoRequest (
947     QString username,
948     IRequestContextPtr ctx);
949
950 void getUserUrlsRequest (
951     IRequestContextPtr ctx);
952
953 void inviteToBusinessRequest (
954     QString emailAddress,
955     IRequestContextPtr ctx);
956
957 void removeFromBusinessRequest (
958     QString emailAddress,
959     IRequestContextPtr ctx);
960
961 void updateBusinessUserIdentifierRequest (
962     QString oldEmailAddress,
963     QString newEmailAddress,
964     IRequestContextPtr ctx);
965
966 void listBusinessUsersRequest (
967     IRequestContextPtr ctx);
968
969 void listBusinessInvitationsRequest (
970     bool includeRequestedInvitations,
971     IRequestContextPtr ctx);
972
973 void getAccountLimitsRequest (
974     ServiceLevel serviceLevel,
975     IRequestContextPtr ctx);
976
977 // Signals used to send encoded response data
978 void checkVersionRequestReady (
979     QByteArray data);
980
981 void getBootstrapInfoRequestReady (
982     QByteArray data);
983
984 void authenticateLongSessionRequestReady (
985     QByteArray data);
986
987 void completeTwoFactorAuthenticationRequestReady (
988     QByteArray data);
989
990 void revokeLongSessionRequestReady (
991     QByteArray data);
992
993 void authenticateToBusinessRequestReady (
994     QByteArray data);
995
996 void getUserRequestReady (
997     QByteArray data);
998
999 void getPublicUserInfoRequestReady (
1000     QByteArray data);
1001
1002 void getUserUrlsRequestReady (
1003     QByteArray data);
1004
1005 void inviteToBusinessRequestReady (
1006     QByteArray data);
1007
1008 void removeFromBusinessRequestReady (
1009     QByteArray data);
1010
1011 void updateBusinessUserIdentifierRequestReady (
1012     QByteArray data);
1013
1014 void listBusinessUsersRequestReady (
1015     QByteArray data);
1016

```

```

1017     void listBusinessInvitationsRequestReady(
1018         QByteArray data);
1019
1020     void getAccountLimitsRequestReady(
1021         QByteArray data);
1022
1023 public Q_SLOTS:
1024     // Slot used to deliver requests to the server
1025     void onRequest(QByteArray data);
1026
1027     // Slots for replies to requests
1028     void onCheckVersionRequestReady(
1029         bool value,
1030         EverCloudExceptionDataPtr exceptionData);
1031
1032     void onGetBootstrapInfoRequestReady(
1033         BootstrapInfo value,
1034         EverCloudExceptionDataPtr exceptionData);
1035
1036     void onAuthenticateLongSessionRequestReady(
1037         AuthenticationResult value,
1038         EverCloudExceptionDataPtr exceptionData);
1039
1040     void onCompleteTwoFactorAuthenticationRequestReady(
1041         AuthenticationResult value,
1042         EverCloudExceptionDataPtr exceptionData);
1043
1044     void onRevokeLongSessionRequestReady(
1045         EverCloudExceptionDataPtr exceptionData);
1046
1047     void onAuthenticateToBusinessRequestReady(
1048         AuthenticationResult value,
1049         EverCloudExceptionDataPtr exceptionData);
1050
1051     void onGetUserRequestReady(
1052         User value,
1053         EverCloudExceptionDataPtr exceptionData);
1054
1055     void onGetPublicUserInfoRequestReady(
1056         PublicUserInfo value,
1057         EverCloudExceptionDataPtr exceptionData);
1058
1059     void onGetUserUrlsRequestReady(
1060         UserUrls value,
1061         EverCloudExceptionDataPtr exceptionData);
1062
1063     void onInviteToBusinessRequestReady(
1064         EverCloudExceptionDataPtr exceptionData);
1065
1066     void onRemoveFromBusinessRequestReady(
1067         EverCloudExceptionDataPtr exceptionData);
1068
1069     void onUpdateBusinessUserIdentifierRequestReady(
1070         EverCloudExceptionDataPtr exceptionData);
1071
1072     void onListBusinessUsersRequestReady(
1073         QList<UserProfile> value,
1074         EverCloudExceptionDataPtr exceptionData);
1075
1076     void onListBusinessInvitationsRequestReady(
1077         QList<BusinessInvitation> value,
1078         EverCloudExceptionDataPtr exceptionData);
1079
1080     void onGetAccountLimitsRequestReady(
1081         AccountLimits value,
1082         EverCloudExceptionDataPtr exceptionData);
1083
1084 };
1085
1086 } // namespace qevercloud
1087
1088 #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.](#)

```
1
12 #ifndef QEVERCLOUD_GENERATED_SERVICES_H
13 #define QEVERCLOUD_GENERATED_SERVICES_H
14
15 #include "../Export.h"
16
17 #include "../AsyncResult.h"
18 #include "../DurableService.h"
19 #include "../Optional.h"
20 #include "../RequestContext.h"
21 #include "Constants.h"
22 #include "Types.h"
23 #include <QObject>
24
25 namespace qevercloud {
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55 class QEVERCLOUD_EXPORT INoteStore: public QObject
56 {
57     Q_OBJECT
58     Q_DISABLE_COPY(INoteStore)
59 protected:
60     INoteStore(QObject * parent) :
61         QObject(parent)
62     {}
63
64 public:
65     virtual QString noteStoreUrl() const = 0;
66     virtual void setNoteStoreUrl(QString url) = 0;
67 }
```

```

72     virtual SyncState getSyncState(
73         IRequestContextPtr ctx = {}) = 0;
74
76     virtual AsyncResult * getSyncStateAsync(
77         IRequestContextPtr ctx = {}) = 0;
78
112    virtual SyncChunk getFilteredSyncChunk(
113        qint32 afterUSN,
114        qint32 maxEntries,
115        const SyncChunkFilter & filter,
116        IRequestContextPtr ctx = {}) = 0;
117
119    virtual AsyncResult * getFilteredSyncChunkAsync(
120        qint32 afterUSN,
121        qint32 maxEntries,
122        const SyncChunkFilter & filter,
123        IRequestContextPtr ctx = {}) = 0;
124
166    virtual SyncState getLinkedNotebookSyncState(
167        const LinkedNotebook & linkedNotebook,
168        IRequestContextPtr ctx = {}) = 0;
169
171    virtual AsyncResult * getLinkedNotebookSyncStateAsync(
172        const LinkedNotebook & linkedNotebook,
173        IRequestContextPtr ctx = {}) = 0;
174
239    virtual SyncChunk getLinkedNotebookSyncChunk(
240        const LinkedNotebook & linkedNotebook,
241        qint32 afterUSN,
242        qint32 maxEntries,
243        bool fullSyncOnly,
244        IRequestContextPtr ctx = {}) = 0;
245
247    virtual AsyncResult * getLinkedNotebookSyncChunkAsync(
248        const LinkedNotebook & linkedNotebook,
249        qint32 afterUSN,
250        qint32 maxEntries,
251        bool fullSyncOnly,
252        IRequestContextPtr ctx = {}) = 0;
253
257    virtual QList<Notebook> listNotebooks(
258        IRequestContextPtr ctx = {}) = 0;
259
261    virtual AsyncResult * listNotebooksAsync(
262        IRequestContextPtr ctx = {}) = 0;
263
277    virtual QList<Notebook> listAccessibleBusinessNotebooks(
278        IRequestContextPtr ctx = {}) = 0;
279
281    virtual AsyncResult * listAccessibleBusinessNotebooksAsync(
282        IRequestContextPtr ctx = {}) = 0;
283
303    virtual Notebook getNotebook(
304        Guid guid,
305        IRequestContextPtr ctx = {}) = 0;
306
308    virtual AsyncResult * getNotebookAsync(
309        Guid guid,
310        IRequestContextPtr ctx = {}) = 0;
311
316    virtual Notebook getDefaultNotebook(
317        IRequestContextPtr ctx = {}) = 0;
318
320    virtual AsyncResult * getDefaultNotebookAsync(
321        IRequestContextPtr ctx = {}) = 0;
322
357    virtual Notebook createNotebook(
358        const Notebook & notebook,
359        IRequestContextPtr ctx = {}) = 0;
360
362    virtual AsyncResult * createNotebookAsync(
363        const Notebook & notebook,
364        IRequestContextPtr ctx = {}) = 0;
365
405    virtual qint32 updateNotebook(
406        const Notebook & notebook,
407        IRequestContextPtr ctx = {}) = 0;
408
410    virtual AsyncResult * updateNotebookAsync(
411        const Notebook & notebook,
412        IRequestContextPtr ctx = {}) = 0;
413
439    virtual qint32 expungeNotebook(
440        Guid guid,
441        IRequestContextPtr ctx = {}) = 0;
442
444    virtual AsyncResult * expungeNotebookAsync(

```

```

445         Guid guid,
446         IRequestContextPtr ctx = {}) = 0;
447
448 virtual QList<Tag> listTags(
449     IRequestContextPtr ctx = {}) = 0;
450
451 virtual AsyncResult * listTagsAsync(
452     IRequestContextPtr ctx = {}) = 0;
453
454 virtual QList<Tag> listTagsByNotebook(
455     Guid notebookGuid,
456     IRequestContextPtr ctx = {}) = 0;
457
458 virtual AsyncResult * listTagsByNotebookAsync(
459     Guid notebookGuid,
460     IRequestContextPtr ctx = {}) = 0;
461
462 virtual Tag getTag(
463     Guid guid,
464     IRequestContextPtr ctx = {}) = 0;
465
466 virtual AsyncResult * getTagAsync(
467     Guid guid,
468     IRequestContextPtr ctx = {}) = 0;
469
470 virtual Tag createTag(
471     const Tag & tag,
472     IRequestContextPtr ctx = {}) = 0;
473
474 virtual AsyncResult * createTagAsync(
475     const Tag & tag,
476     IRequestContextPtr ctx = {}) = 0;
477
478 virtual qint32 updateTag(
479     const Tag & tag,
480     IRequestContextPtr ctx = {}) = 0;
481
482 virtual AsyncResult * updateTagAsync(
483     const Tag & tag,
484     IRequestContextPtr ctx = {}) = 0;
485
486 virtual void untagAll(
487     Guid guid,
488     IRequestContextPtr ctx = {}) = 0;
489
490 virtual AsyncResult * untagAllAsync(
491     Guid guid,
492     IRequestContextPtr ctx = {}) = 0;
493
494 virtual qint32 expungeTag(
495     Guid guid,
496     IRequestContextPtr ctx = {}) = 0;
497
498 virtual AsyncResult * expungeTagAsync(
499     Guid guid,
500     IRequestContextPtr ctx = {}) = 0;
501
502 virtual QList<SavedSearch> listSearches(
503     IRequestContextPtr ctx = {}) = 0;
504
505 virtual AsyncResult * listSearchesAsync(
506     IRequestContextPtr ctx = {}) = 0;
507
508 virtual SavedSearch getSearch(
509     Guid guid,
510     IRequestContextPtr ctx = {}) = 0;
511
512 virtual AsyncResult * getSearchAsync(
513     Guid guid,
514     IRequestContextPtr ctx = {}) = 0;
515
516 virtual SavedSearch createSearch(
517     const SavedSearch & search,
518     IRequestContextPtr ctx = {}) = 0;
519
520 virtual AsyncResult * createSearchAsync(
521     const SavedSearch & search,
522     IRequestContextPtr ctx = {}) = 0;
523
524 virtual qint32 updateSearch(
525     const SavedSearch & search,
526     IRequestContextPtr ctx = {}) = 0;
527
528 virtual AsyncResult * updateSearchAsync(
529     const SavedSearch & search,
530     IRequestContextPtr ctx = {}) = 0;
531
532

```

```
785     virtual qint32 expungeSearch(  
786         Guid guid,  
787         IRequestContextPtr ctx = {}) = 0;  
788  
790     virtual AsyncResult * expungeSearchAsync(  
791         Guid guid,  
792         IRequestContextPtr ctx = {}) = 0;  
793  
835     virtual qint32 findNoteOffset(  
836         const NoteFilter & filter,  
837         Guid guid,  
838         IRequestContextPtr ctx = {}) = 0;  
839  
841     virtual AsyncResult * findNoteOffsetAsync(  
842         const NoteFilter & filter,  
843         Guid guid,  
844         IRequestContextPtr ctx = {}) = 0;  
845  
903     virtual NotesMetadataList findNotesMetadata(  
904         const NoteFilter & filter,  
905         qint32 offset,  
906         qint32 maxNotes,  
907         const NotesMetadataResultSpec & resultSpec,  
908         IRequestContextPtr ctx = {}) = 0;  
909  
911     virtual AsyncResult * findNotesMetadataAsync(  
912         const NoteFilter & filter,  
913         qint32 offset,  
914         qint32 maxNotes,  
915         const NotesMetadataResultSpec & resultSpec,  
916         IRequestContextPtr ctx = {}) = 0;  
917  
949     virtual NoteCollectionCounts findNoteCounts(  
950         const NoteFilter & filter,  
951         bool withTrash,  
952         IRequestContextPtr ctx = {}) = 0;  
953  
955     virtual AsyncResult * findNoteCountsAsync(  
956         const NoteFilter & filter,  
957         bool withTrash,  
958         IRequestContextPtr ctx = {}) = 0;  
959  
991     virtual Note getNoteWithResultSpec(  
992         Guid guid,  
993         const NoteResultSpec & resultSpec,  
994         IRequestContextPtr ctx = {}) = 0;  
995  
997     virtual AsyncResult * getNoteWithResultSpecAsync(  
998         Guid guid,  
999         const NoteResultSpec & resultSpec,  
1000         IRequestContextPtr ctx = {}) = 0;  
1001  
1009     virtual Note getNote(  
1010         Guid guid,  
1011         bool withContent,  
1012         bool withResourcesData,  
1013         bool withResourcesRecognition,  
1014         bool withResourcesAlternateData,  
1015         IRequestContextPtr ctx = {}) = 0;  
1016  
1018     virtual AsyncResult * getNoteAsync(  
1019         Guid guid,  
1020         bool withContent,  
1021         bool withResourcesData,  
1022         bool withResourcesRecognition,  
1023         bool withResourcesAlternateData,  
1024         IRequestContextPtr ctx = {}) = 0;  
1025  
1034     virtual LazyMap getNoteApplicationData(  
1035         Guid guid,  
1036         IRequestContextPtr ctx = {}) = 0;  
1037  
1039     virtual AsyncResult * getNoteApplicationDataAsync(  
1040         Guid guid,  
1041         IRequestContextPtr ctx = {}) = 0;  
1042  
1052     virtual QString getNoteApplicationDataEntry(  
1053         Guid guid,  
1054         QString key,  
1055         IRequestContextPtr ctx = {}) = 0;  
1056  
1058     virtual AsyncResult * getNoteApplicationDataEntryAsync(  
1059         Guid guid,  
1060         QString key,  
1061         IRequestContextPtr ctx = {}) = 0;  
1062  
1067     virtual qint32 setNoteApplicationDataEntry(  

```

```

1068         Guid guid,
1069         QString key,
1070         QString value,
1071         IRequestContextPtr ctx = {}) = 0;
1072
1073 virtual AsyncResult * setNoteApplicationDataEntryAsync(
1074     Guid guid,
1075     QString key,
1076     QString value,
1077     IRequestContextPtr ctx = {}) = 0;
1078
1079 virtual qint32 unsetNoteApplicationDataEntry(
1080     Guid guid,
1081     QString key,
1082     IRequestContextPtr ctx = {}) = 0;
1083
1084 virtual AsyncResult * unsetNoteApplicationDataEntryAsync(
1085     Guid guid,
1086     QString key,
1087     IRequestContextPtr ctx = {}) = 0;
1088
1089 virtual QString getNoteContent(
1090     Guid guid,
1091     IRequestContextPtr ctx = {}) = 0;
1092
1093 virtual AsyncResult * getNoteContentAsync(
1094     Guid guid,
1095     IRequestContextPtr ctx = {}) = 0;
1096
1097 virtual QString getNoteSearchText(
1098     Guid guid,
1099     bool noteOnly,
1100     bool tokenizeForIndexing,
1101     IRequestContextPtr ctx = {}) = 0;
1102
1103 virtual AsyncResult * getNoteSearchTextAsync(
1104     Guid guid,
1105     bool noteOnly,
1106     bool tokenizeForIndexing,
1107     IRequestContextPtr ctx = {}) = 0;
1108
1109 virtual QString getResourceSearchText(
1110     Guid guid,
1111     IRequestContextPtr ctx = {}) = 0;
1112
1113 virtual AsyncResult * getResourceSearchTextAsync(
1114     Guid guid,
1115     IRequestContextPtr ctx = {}) = 0;
1116
1117 virtual QStringList getNoteTagNames(
1118     Guid guid,
1119     IRequestContextPtr ctx = {}) = 0;
1120
1121 virtual AsyncResult * getNoteTagNamesAsync(
1122     Guid guid,
1123     IRequestContextPtr ctx = {}) = 0;
1124
1125 virtual Note createNote(
1126     const Note & note,
1127     IRequestContextPtr ctx = {}) = 0;
1128
1129 virtual AsyncResult * createNoteAsync(
1130     const Note & note,
1131     IRequestContextPtr ctx = {}) = 0;
1132
1133 virtual Note updateNote(
1134     const Note & note,
1135     IRequestContextPtr ctx = {}) = 0;
1136
1137 virtual AsyncResult * updateNoteAsync(
1138     const Note & note,
1139     IRequestContextPtr ctx = {}) = 0;
1140
1141 virtual qint32 deleteNote(
1142     Guid guid,
1143     IRequestContextPtr ctx = {}) = 0;
1144
1145 virtual AsyncResult * deleteNoteAsync(
1146     Guid guid,
1147     IRequestContextPtr ctx = {}) = 0;
1148
1149 virtual qint32 expungeNote(
1150     Guid guid,
1151     IRequestContextPtr ctx = {}) = 0;
1152
1153 virtual AsyncResult * expungeNoteAsync(
1154     Guid guid,

```

```

1450         IRequestContextPtr ctx = {} ) = 0;
1451
1494 virtual Note copyNote(
1495     Guid noteGuid,
1496     Guid toNotebookGuid,
1497     IRequestContextPtr ctx = {} ) = 0;
1498
1500 virtual AsyncResult * copyNoteAsync(
1501     Guid noteGuid,
1502     Guid toNotebookGuid,
1503     IRequestContextPtr ctx = {} ) = 0;
1504
1527 virtual QList<NoteVersionId> listNoteVersions(
1528     Guid noteGuid,
1529     IRequestContextPtr ctx = {} ) = 0;
1530
1532 virtual AsyncResult * listNoteVersionsAsync(
1533     Guid noteGuid,
1534     IRequestContextPtr ctx = {} ) = 0;
1535
1579 virtual Note getNoteVersion(
1580     Guid noteGuid,
1581     quint32 updateSequenceNum,
1582     bool withResourcesData,
1583     bool withResourcesRecognition,
1584     bool withResourcesAlternateData,
1585     IRequestContextPtr ctx = {} ) = 0;
1586
1588 virtual AsyncResult * getNoteVersionAsync(
1589     Guid noteGuid,
1590     quint32 updateSequenceNum,
1591     bool withResourcesData,
1592     bool withResourcesRecognition,
1593     bool withResourcesAlternateData,
1594     IRequestContextPtr ctx = {} ) = 0;
1595
1633 virtual Resource getResource(
1634     Guid guid,
1635     bool withData,
1636     bool withRecognition,
1637     bool withAttributes,
1638     bool withAlternateData,
1639     IRequestContextPtr ctx = {} ) = 0;
1640
1642 virtual AsyncResult * getResourceAsync(
1643     Guid guid,
1644     bool withData,
1645     bool withRecognition,
1646     bool withAttributes,
1647     bool withAlternateData,
1648     IRequestContextPtr ctx = {} ) = 0;
1649
1658 virtual LazyMap getResourceApplicationData(
1659     Guid guid,
1660     IRequestContextPtr ctx = {} ) = 0;
1661
1663 virtual AsyncResult * getResourceApplicationDataAsync(
1664     Guid guid,
1665     IRequestContextPtr ctx = {} ) = 0;
1666
1676 virtual QString getResourceApplicationDataEntry(
1677     Guid guid,
1678     QString key,
1679     IRequestContextPtr ctx = {} ) = 0;
1680
1682 virtual AsyncResult * getResourceApplicationDataEntryAsync(
1683     Guid guid,
1684     QString key,
1685     IRequestContextPtr ctx = {} ) = 0;
1686
1691 virtual quint32 setResourceApplicationDataEntry(
1692     Guid guid,
1693     QString key,
1694     QString value,
1695     IRequestContextPtr ctx = {} ) = 0;
1696
1698 virtual AsyncResult * setResourceApplicationDataEntryAsync(
1699     Guid guid,
1700     QString key,
1701     QString value,
1702     IRequestContextPtr ctx = {} ) = 0;
1703
1708 virtual quint32 unsetResourceApplicationDataEntry(
1709     Guid guid,
1710     QString key,
1711     IRequestContextPtr ctx = {} ) = 0;
1712

```

```

1714     virtual AsyncResult * unsetResourceApplicationDataEntryAsync(
1715         Guid guid,
1716         QString key,
1717         IRequestContextPtr ctx = {} ) = 0;
1718
1719     virtual qint32 updateResource(
1720         const Resource & resource,
1721         IRequestContextPtr ctx = {} ) = 0;
1722
1723     virtual AsyncResult * updateResourceAsync(
1724         const Resource & resource,
1725         IRequestContextPtr ctx = {} ) = 0;
1726
1727     virtual QByteArray getResourceData(
1728         Guid guid,
1729         IRequestContextPtr ctx = {} ) = 0;
1730
1731     virtual AsyncResult * getResourceDataAsync(
1732         Guid guid,
1733         IRequestContextPtr ctx = {} ) = 0;
1734
1735     virtual Resource getResourceByHash(
1736         Guid noteGuid,
1737         QByteArray contentHash,
1738         bool withData,
1739         bool withRecognition,
1740         bool withAlternateData,
1741         IRequestContextPtr ctx = {} ) = 0;
1742
1743     virtual AsyncResult * getResourceByHashAsync(
1744         Guid noteGuid,
1745         QByteArray contentHash,
1746         bool withData,
1747         bool withRecognition,
1748         bool withAlternateData,
1749         IRequestContextPtr ctx = {} ) = 0;
1750
1751     virtual QByteArray getResourceRecognition(
1752         Guid guid,
1753         IRequestContextPtr ctx = {} ) = 0;
1754
1755     virtual AsyncResult * getResourceRecognitionAsync(
1756         Guid guid,
1757         IRequestContextPtr ctx = {} ) = 0;
1758
1759     virtual QByteArray getResourceAlternateData(
1760         Guid guid,
1761         IRequestContextPtr ctx = {} ) = 0;
1762
1763     virtual AsyncResult * getResourceAlternateDataAsync(
1764         Guid guid,
1765         IRequestContextPtr ctx = {} ) = 0;
1766
1767     virtual ResourceAttributes getResourceAttributes(
1768         Guid guid,
1769         IRequestContextPtr ctx = {} ) = 0;
1770
1771     virtual AsyncResult * getResourceAttributesAsync(
1772         Guid guid,
1773         IRequestContextPtr ctx = {} ) = 0;
1774
1775     virtual Notebook getPublicNotebook(
1776         UserID userId,
1777         QString publicUri,
1778         IRequestContextPtr ctx = {} ) = 0;
1779
1780     virtual AsyncResult * getPublicNotebookAsync(
1781         UserID userId,
1782         QString publicUri,
1783         IRequestContextPtr ctx = {} ) = 0;
1784
1785     virtual SharedNotebook shareNotebook(
1786         const SharedNotebook & sharedNotebook,
1787         QString message,
1788         IRequestContextPtr ctx = {} ) = 0;
1789
1790     virtual AsyncResult * shareNotebookAsync(
1791         const SharedNotebook & sharedNotebook,
1792         QString message,
1793         IRequestContextPtr ctx = {} ) = 0;
1794
1795     virtual CreateOrUpdateNotebookSharesResult createOrUpdateNotebookShares(
1796         const NotebookShareTemplate & shareTemplate,
1797         IRequestContextPtr ctx = {} ) = 0;
1798
1799     virtual AsyncResult * createOrUpdateNotebookSharesAsync(
1800         const NotebookShareTemplate & shareTemplate,

```

```

2155         IRequestContextPtr ctx = {}) = 0;
2156
2160     virtual qint32 updateSharedNotebook(
2161         const SharedNotebook & sharedNotebook,
2162         IRequestContextPtr ctx = {}) = 0;
2163
2165     virtual AsyncResult * updateSharedNotebookAsync(
2166         const SharedNotebook & sharedNotebook,
2167         IRequestContextPtr ctx = {}) = 0;
2168
2205     virtual Notebook setNotebookRecipientSettings(
2206         QString notebookGuid,
2207         const NotebookRecipientSettings & recipientSettings,
2208         IRequestContextPtr ctx = {}) = 0;
2209
2211     virtual AsyncResult * setNotebookRecipientSettingsAsync(
2212         QString notebookGuid,
2213         const NotebookRecipientSettings & recipientSettings,
2214         IRequestContextPtr ctx = {}) = 0;
2215
2223     virtual QList<SharedNotebook> listSharedNotebooks(
2224         IRequestContextPtr ctx = {}) = 0;
2225
2227     virtual AsyncResult * listSharedNotebooksAsync(
2228         IRequestContextPtr ctx = {}) = 0;
2229
2267     virtual LinkedNotebook createLinkedNotebook(
2268         const LinkedNotebook & linkedNotebook,
2269         IRequestContextPtr ctx = {}) = 0;
2270
2272     virtual AsyncResult * createLinkedNotebookAsync(
2273         const LinkedNotebook & linkedNotebook,
2274         IRequestContextPtr ctx = {}) = 0;
2275
2292     virtual qint32 updateLinkedNotebook(
2293         const LinkedNotebook & linkedNotebook,
2294         IRequestContextPtr ctx = {}) = 0;
2295
2297     virtual AsyncResult * updateLinkedNotebookAsync(
2298         const LinkedNotebook & linkedNotebook,
2299         IRequestContextPtr ctx = {}) = 0;
2300
2304     virtual QList<LinkedNotebook> listLinkedNotebooks(
2305         IRequestContextPtr ctx = {}) = 0;
2306
2308     virtual AsyncResult * listLinkedNotebooksAsync(
2309         IRequestContextPtr ctx = {}) = 0;
2310
2322     virtual qint32 expungeLinkedNotebook(
2323         Guid guid,
2324         IRequestContextPtr ctx = {}) = 0;
2325
2327     virtual AsyncResult * expungeLinkedNotebookAsync(
2328         Guid guid,
2329         IRequestContextPtr ctx = {}) = 0;
2330
2381     virtual AuthenticationResult authenticateToSharedNotebook(
2382         QString shareKeyOrGlobalId,
2383         IRequestContextPtr ctx = {}) = 0;
2384
2386     virtual AsyncResult * authenticateToSharedNotebookAsync(
2387         QString shareKeyOrGlobalId,
2388         IRequestContextPtr ctx = {}) = 0;
2389
2415     virtual SharedNotebook getSharedNotebookByAuth(
2416         IRequestContextPtr ctx = {}) = 0;
2417
2419     virtual AsyncResult * getSharedNotebookByAuthAsync(
2420         IRequestContextPtr ctx = {}) = 0;
2421
2471     virtual void emailNote(
2472         const NoteEmailParameters & parameters,
2473         IRequestContextPtr ctx = {}) = 0;
2474
2476     virtual AsyncResult * emailNoteAsync(
2477         const NoteEmailParameters & parameters,
2478         IRequestContextPtr ctx = {}) = 0;
2479
2503     virtual QString shareNote(
2504         Guid guid,
2505         IRequestContextPtr ctx = {}) = 0;
2506
2508     virtual AsyncResult * shareNoteAsync(
2509         Guid guid,
2510         IRequestContextPtr ctx = {}) = 0;
2511
2534     virtual void stopSharingNote(

```



```

2535     Guid guid,
2536     IRequestContextPtr ctx = {}) = 0;
2537
2539     virtual AsyncResult * stopSharingNoteAsync(
2540         Guid guid,
2541         IRequestContextPtr ctx = {}) = 0;
2542
2545     virtual AuthenticationResult authenticateToSharedNote(
2546         QString guid,
2547         QString noteKey,
2548         IRequestContextPtr ctx = {}) = 0;
2549
2551     virtual AsyncResult * authenticateToSharedNoteAsync(
2552         QString guid,
2553         QString noteKey,
2554         IRequestContextPtr ctx = {}) = 0;
2555
2565     virtual RelatedResult findRelated(
2566         const RelatedQuery & query,
2567         const RelatedResultSpec & resultSpec,
2568         IRequestContextPtr ctx = {}) = 0;
2569
2571     virtual AsyncResult * findRelatedAsync(
2572         const RelatedQuery & query,
2573         const RelatedResultSpec & resultSpec,
2574         IRequestContextPtr ctx = {}) = 0;
2575
2583     virtual UpdateNoteIfUsnMatchesResult updateNoteIfUsnMatches(
2584         const Note & note,
2585         IRequestContextPtr ctx = {}) = 0;
2586
2588     virtual AsyncResult * updateNoteIfUsnMatchesAsync(
2589         const Note & note,
2590         IRequestContextPtr ctx = {}) = 0;
2591
2593     virtual ManageNotebookSharesResult manageNotebookShares(
2594         const ManageNotebookSharesParameters & parameters,
2595         IRequestContextPtr ctx = {}) = 0;
2596
2598     virtual AsyncResult * manageNotebookSharesAsync(
2599         const ManageNotebookSharesParameters & parameters,
2600         IRequestContextPtr ctx = {}) = 0;
2601
2603     virtual ShareRelationships getNotebookShares(
2604         QString notebookGuid,
2605         IRequestContextPtr ctx = {}) = 0;
2606
2608     virtual AsyncResult * getNotebookSharesAsync(
2609         QString notebookGuid,
2610         IRequestContextPtr ctx = {}) = 0;
2611
2613 };
2614
2615 using INoteStorePtr = std::shared_ptr<INoteStore>;
2616
2617
2619 class QEVERCLOUD_EXPORT IUserStore: public QObject
2620 {
2621     Q_OBJECT
2622     Q_DISABLE_COPY(IUserStore)
2623 protected:
2624     IUserStore(QObject * parent) :
2625         QObject(parent)
2626     {}
2627
2628 public:
2629     virtual QString userStoreUrl() const = 0;
2630     virtual void setUserStoreUrl(QString url) = 0;
2631
2633     virtual bool checkVersion(
2634         QString clientName,
2635         quint16 edamVersionMajor = EDAM_VERSION_MAJOR,
2636         quint16 edamVersionMinor = EDAM_VERSION_MINOR,
2637         IRequestContextPtr ctx = {}) = 0;
2638
2640     virtual AsyncResult * checkVersionAsync(
2641         QString clientName,
2642         quint16 edamVersionMajor = EDAM_VERSION_MAJOR,
2643         quint16 edamVersionMinor = EDAM_VERSION_MINOR,
2644         IRequestContextPtr ctx = {}) = 0;
2645
2647     virtual BootstrapInfo getBootstrapInfo(
2648         QString locale,
2649         IRequestContextPtr ctx = {}) = 0;
2650
2652     virtual AsyncResult * getBootstrapInfoAsync(
2653         QString locale,

```

```
2830         IRequestContextPtr ctx = {}) = 0;
2831
2918 virtual AuthenticationResult authenticateLongSession(
2919     QString username,
2920     QString password,
2921     QString consumerKey,
2922     QString consumerSecret,
2923     QString deviceIdentifier,
2924     QString deviceDescription,
2925     bool supportsTwoFactor,
2926     IRequestContextPtr ctx = {}) = 0;
2927
2929 virtual AsyncResult * authenticateLongSessionAsync(
2930     QString username,
2931     QString password,
2932     QString consumerKey,
2933     QString consumerSecret,
2934     QString deviceIdentifier,
2935     QString deviceDescription,
2936     bool supportsTwoFactor,
2937     IRequestContextPtr ctx = {}) = 0;
2938
2977 virtual AuthenticationResult completeTwoFactorAuthentication(
2978     QString oneTimeCode,
2979     QString deviceIdentifier,
2980     QString deviceDescription,
2981     IRequestContextPtr ctx = {}) = 0;
2982
2984 virtual AsyncResult * completeTwoFactorAuthenticationAsync(
2985     QString oneTimeCode,
2986     QString deviceIdentifier,
2987     QString deviceDescription,
2988     IRequestContextPtr ctx = {}) = 0;
2989
3008 virtual void revokeLongSession(
3009     IRequestContextPtr ctx = {}) = 0;
3010
3012 virtual AsyncResult * revokeLongSessionAsync(
3013     IRequestContextPtr ctx = {}) = 0;
3014
3048 virtual AuthenticationResult authenticateToBusiness(
3049     IRequestContextPtr ctx = {}) = 0;
3050
3052 virtual AsyncResult * authenticateToBusinessAsync(
3053     IRequestContextPtr ctx = {}) = 0;
3054
3062 virtual User getUser(
3063     IRequestContextPtr ctx = {}) = 0;
3064
3066 virtual AsyncResult * getUserAsync(
3067     IRequestContextPtr ctx = {}) = 0;
3068
3077 virtual PublicUserInfo getPublicUserInfo(
3078     QString username,
3079     IRequestContextPtr ctx = {}) = 0;
3080
3082 virtual AsyncResult * getPublicUserInfoAsync(
3083     QString username,
3084     IRequestContextPtr ctx = {}) = 0;
3085
3095 virtual UserUrls getUserUrls(
3096     IRequestContextPtr ctx = {}) = 0;
3097
3099 virtual AsyncResult * getUserUrlsAsync(
3100     IRequestContextPtr ctx = {}) = 0;
3101
3145 virtual void inviteToBusiness(
3146     QString emailAddress,
3147     IRequestContextPtr ctx = {}) = 0;
3148
3150 virtual AsyncResult * inviteToBusinessAsync(
3151     QString emailAddress,
3152     IRequestContextPtr ctx = {}) = 0;
3153
3178 virtual void removeFromBusiness(
3179     QString emailAddress,
3180     IRequestContextPtr ctx = {}) = 0;
3181
3183 virtual AsyncResult * removeFromBusinessAsync(
3184     QString emailAddress,
3185     IRequestContextPtr ctx = {}) = 0;
3186
3229 virtual void updateBusinessUserIdentifier(
3230     QString oldEmailAddress,
3231     QString newEmailAddress,
3232     IRequestContextPtr ctx = {}) = 0;
3233
```

```

3235     virtual AsyncResult * updateBusinessUserIdentifierAsync(
3236         QString oldEmailAddress,
3237         QString newEmailAddress,
3238         IRequestContextPtr ctx = {}) = 0;
3239
3258     virtual QList<UserProfile> listBusinessUsers(
3259         IRequestContextPtr ctx = {}) = 0;
3260
3262     virtual AsyncResult * listBusinessUsersAsync(
3263         IRequestContextPtr ctx = {}) = 0;
3264
3279     virtual QList<BusinessInvitation> listBusinessInvitations(
3280         bool includeRequestedInvitations,
3281         IRequestContextPtr ctx = {}) = 0;
3282
3284     virtual AsyncResult * listBusinessInvitationsAsync(
3285         bool includeRequestedInvitations,
3286         IRequestContextPtr ctx = {}) = 0;
3287
3298     virtual AccountLimits getAccountLimits(
3299         ServiceLevel serviceLevel,
3300         IRequestContextPtr ctx = {}) = 0;
3301
3303     virtual AsyncResult * getAccountLimitsAsync(
3304         ServiceLevel serviceLevel,
3305         IRequestContextPtr ctx = {}) = 0;
3306
3307 };
3308
3309 using IUserStorePtr = std::shared_ptr<IUserStore>;
3310
3312
3313 QEVERCLOUD_EXPORT INoteStore * newNoteStore(
3314     QString noteStoreUrl = {},
3315     IRequestContextPtr ctx = {},
3316     QObject * parent = nullptr,
3317     IRetryPolicyPtr retryPolicy = {});
3318
3319 QEVERCLOUD_EXPORT IUserStore * newUserStore(
3320     QString userStoreUrl = {},
3321     IRequestContextPtr ctx = {},
3322     QObject * parent = nullptr,
3323     IRetryPolicyPtr retryPolicy = {});
3324
3325 } // namespace qevercloud
3326
3327 Q_DECLARE_METATYPE(QList<qevercloud::Notebook>)
3328 Q_DECLARE_METATYPE(QList<qevercloud::Tag>)
3329 Q_DECLARE_METATYPE(QList<qevercloud::SavedSearch>)
3330 Q_DECLARE_METATYPE(QList<qevercloud::NoteVersionId>)
3331 Q_DECLARE_METATYPE(QList<qevercloud::SharedNotebook>)
3332 Q_DECLARE_METATYPE(QList<qevercloud::LinkedNotebook>)
3333 Q_DECLARE_METATYPE(QList<qevercloud::BusinessInvitation>)
3334 Q_DECLARE_METATYPE(QList<qevercloud::UserProfile>)
3335
3336 #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.](#)

```

1
12 #ifndef QEVERCLOUD_GENERATED_TYPES_H
13 #define QEVERCLOUD_GENERATED_TYPES_H
14
15 #include "../Export.h"
16
17 #include "../Optional.h"
18 #include "../Printable.h"
19 #include "EDAMErrorCode.h"
20 #include <QByteArray>
21 #include <QDateTime>
22 #include <QHash>
23 #include <QList>
24 #include <QMap>
25 #include <QMetaType>
26 #include <QMetaType>
27 #include <QSet>
28 #include <QStringList>
29 #include <QVariant>
30
31 namespace qevercloud {
32
33 using InvalidationTokenSequenceNumber = quint64;
34
35 using IdentityID = quint64;
36
37 using UserID = quint32;
38
39 using Guid = QString;
40
41 using Timestamp = quint64;
42
43 using MessageEventID = quint64;
44
45 using MessageThreadID = quint64;
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102 class QEVERCLOUD_EXPORT EverCloudLocalData: public Printable
103 {
104     Q_GADGET
105 public:
106     EverCloudLocalData();
107     virtual ~EverCloudLocalData() noexcept override;
108
109     virtual void print(QTextStream & strm) const override;
110
111     bool operator==(const EverCloudLocalData & other) const;
112     bool operator!=(const EverCloudLocalData & other) const;
113     QString id;
114
115     bool dirty = false;
116
117     bool local = false;
118
119     bool favorited = false;
120
121     QHash<QString, QVariant> dict;
122
123     // Properties declaration for meta-object system
124     Q_PROPERTY(QString id MEMBER id USER true)
125     Q_PROPERTY(bool dirty MEMBER dirty)
126     Q_PROPERTY(bool local MEMBER local)
127     Q_PROPERTY(bool favorited MEMBER favorited)
128
129     using Dict = QHash<QString, QVariant>;
130     Q_PROPERTY(Dict dict MEMBER dict)
131 };
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165 struct QEVERCLOUD_EXPORT SyncState: public Printable
166 {
167 private:
168     Q_GADGET
169 public:
170     EverCloudLocalData localData;
171
172     Timestamp currentTime = 0;
173     Timestamp fullSyncBefore = 0;
174     quint32 updateCount = 0;
175     Optional<quint64> uploaded;
176     Optional<Timestamp> userLastUpdated;
177     Optional<MessageEventID> userMaxMessageEventId;
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233

```

```

234     virtual void print(QTextStream & strm) const override;
235
236     bool operator==(const SyncState & other) const
237     {
238         return (currentTime == other.currentTime)
239             && (fullSyncBefore == other.fullSyncBefore)
240             && (updateCount == other.updateCount)
241             && uploaded.isEqual(other.uploaded)
242             && userLastUpdated.isEqual(other.userLastUpdated)
243             && userMaxMessageEventId.isEqual(other.userMaxMessageEventId)
244         ;
245     }
246
247     bool operator!=(const SyncState & other) const
248     {
249         return !(*this == other);
250     }
251
252     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
253     Q_PROPERTY(Timestamp currentTime MEMBER currentTime)
254     Q_PROPERTY(Timestamp fullSyncBefore MEMBER fullSyncBefore)
255     Q_PROPERTY(qint32 updateCount MEMBER updateCount)
256     Q_PROPERTY(Optional<qint64> uploaded MEMBER uploaded)
257     Q_PROPERTY(Optional<Timestamp> userLastUpdated MEMBER userLastUpdated)
258     Q_PROPERTY(Optional<MessageEventID> userMaxMessageEventId MEMBER userMaxMessageEventId)
259 };
260
261 struct QEVERCLOUD_EXPORT SyncChunkFilter: public Printable
262 {
263 private:
264     Q_GADGET
265 public:
266     EverCloudLocalData localData;
267
268     Optional<bool> includeNotes;
269     Optional<bool> includeNoteResources;
270     Optional<bool> includeNoteAttributes;
271     Optional<bool> includeNotebooks;
272     Optional<bool> includeTags;
273     Optional<bool> includeSearches;
274     Optional<bool> includeResources;
275     Optional<bool> includeLinkedNotebooks;
276     Optional<bool> includeExpunged;
277     Optional<bool> includeNoteApplicationDataFullMap;
278     Optional<bool> includeResourceApplicationDataFullMap;
279     Optional<bool> includeNoteResourceApplicationDataFullMap;
280     Optional<bool> includeSharedNotes;
281     Optional<bool> omitSharedNotebooks;
282     Optional<QString> requireNoteContentClass;
283     Optional<QSet<QString>> notebookGuids;
284
285     virtual void print(QTextStream & strm) const override;
286
287     bool operator==(const SyncChunkFilter & other) const
288     {
289         return includeNotes.isEqual(other.includeNotes)
290             && includeNoteResources.isEqual(other.includeNoteResources)
291             && includeNoteAttributes.isEqual(other.includeNoteAttributes)
292             && includeNotebooks.isEqual(other.includeNotebooks)
293             && includeTags.isEqual(other.includeTags)
294             && includeSearches.isEqual(other.includeSearches)
295             && includeResources.isEqual(other.includeResources)
296             && includeLinkedNotebooks.isEqual(other.includeLinkedNotebooks)
297             && includeExpunged.isEqual(other.includeExpunged)
298             && includeNoteApplicationDataFullMap.isEqual(other.includeNoteApplicationDataFullMap)
299             && includeResourceApplicationDataFullMap.isEqual(other.includeResourceApplicationDataFullMap)
300             && includeNoteResourceApplicationDataFullMap.isEqual(other.includeNoteResourceApplicationDataFullMap)
301             && includeSharedNotes.isEqual(other.includeSharedNotes)
302             && omitSharedNotebooks.isEqual(other.omitSharedNotebooks)
303             && requireNoteContentClass.isEqual(other.requireNoteContentClass)
304             && notebookGuids.isEqual(other.notebookGuids)
305         ;
306     }
307
308     bool operator!=(const SyncChunkFilter & other) const
309     {
310         return !(*this == other);
311     }
312
313     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
314     Q_PROPERTY(Optional<bool> includeNotes MEMBER includeNotes)
315     Q_PROPERTY(Optional<bool> includeNoteResources MEMBER includeNoteResources)
316     Q_PROPERTY(Optional<bool> includeNoteAttributes MEMBER includeNoteAttributes)
317     Q_PROPERTY(Optional<bool> includeNotebooks MEMBER includeNotebooks)
318     Q_PROPERTY(Optional<bool> includeTags MEMBER includeTags)

```

```

412     Q_PROPERTY(Optional<bool> includeSearches MEMBER includeSearches)
413     Q_PROPERTY(Optional<bool> includeResources MEMBER includeResources)
414     Q_PROPERTY(Optional<bool> includeLinkedNotebooks MEMBER includeLinkedNotebooks)
415     Q_PROPERTY(Optional<bool> includeExpunged MEMBER includeExpunged)
416     Q_PROPERTY(Optional<bool> includeNoteApplicationDataFullMap MEMBER
includeNoteApplicationDataFullMap)
417     Q_PROPERTY(Optional<bool> includeResourceApplicationDataFullMap MEMBER
includeResourceApplicationDataFullMap)
418     Q_PROPERTY(Optional<bool> includeNoteResourceApplicationDataFullMap MEMBER
includeNoteResourceApplicationDataFullMap)
419     Q_PROPERTY(Optional<bool> includeSharedNotes MEMBER includeSharedNotes)
420     Q_PROPERTY(Optional<bool> omitSharedNotebooks MEMBER omitSharedNotebooks)
421     Q_PROPERTY(Optional<QString> requireNoteContentClass MEMBER requireNoteContentClass)
422     Q_PROPERTY(Optional<QSet<QString> notebookGuids MEMBER notebookGuids)
423 };
424
425 struct QEVERCLOUD_EXPORT NoteFilter: public Printable
426 {
427     private:
428         Q_GADGET
429     public:
430         EverCloudLocalData localData;
431
432         Optional<qint32> order;
433         Optional<bool> ascending;
434         Optional<QString> words;
435         Optional<Guid> notebookGuid;
436         Optional<QList<Guid>> tagGuids;
437         Optional<QString> timeZone;
438         Optional<bool> inactive;
439         Optional<QString> emphasized;
440         Optional<bool> includeAllReadableNotebooks;
441         Optional<bool> includeAllReadableWorkspaces;
442         Optional<QString> context;
443         Optional<QString> rawWords;
444         Optional<QByteArray> searchContextBytes;
445
446         virtual void print(QTextStream & strm) const override;
447
448         bool operator==(const NoteFilter & other) const
449         {
450             return order.isEqual(other.order)
451                 && ascending.isEqual(other.ascending)
452                 && words.isEqual(other.words)
453                 && notebookGuid.isEqual(other.notebookGuid)
454                 && tagGuids.isEqual(other.tagGuids)
455                 && timeZone.isEqual(other.timeZone)
456                 && inactive.isEqual(other.inactive)
457                 && emphasized.isEqual(other.emphasized)
458                 && includeAllReadableNotebooks.isEqual(other.includeAllReadableNotebooks)
459                 && includeAllReadableWorkspaces.isEqual(other.includeAllReadableWorkspaces)
460                 && context.isEqual(other.context)
461                 && rawWords.isEqual(other.rawWords)
462                 && searchContextBytes.isEqual(other.searchContextBytes)
463             ;
464         }
465
466         bool operator!=(const NoteFilter & other) const
467         {
468             return !(*this == other);
469         }
470
471         Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
472         Q_PROPERTY(Optional<qint32> order MEMBER order)
473         Q_PROPERTY(Optional<bool> ascending MEMBER ascending)
474         Q_PROPERTY(Optional<QString> words MEMBER words)
475         Q_PROPERTY(Optional<Guid> notebookGuid MEMBER notebookGuid)
476         Q_PROPERTY(Optional<QList<Guid>> tagGuids MEMBER tagGuids)
477         Q_PROPERTY(Optional<QString> timeZone MEMBER timeZone)
478         Q_PROPERTY(Optional<bool> inactive MEMBER inactive)
479         Q_PROPERTY(Optional<QString> emphasized MEMBER emphasized)
480         Q_PROPERTY(Optional<bool> includeAllReadableNotebooks MEMBER includeAllReadableNotebooks)
481         Q_PROPERTY(Optional<bool> includeAllReadableWorkspaces MEMBER includeAllReadableWorkspaces)
482         Q_PROPERTY(Optional<QString> context MEMBER context)
483         Q_PROPERTY(Optional<QString> rawWords MEMBER rawWords)
484         Q_PROPERTY(Optional<QByteArray> searchContextBytes MEMBER searchContextBytes)
485 };
486
487 struct QEVERCLOUD_EXPORT NotesMetadataResultSpec: public Printable
488 {
489     private:
490         Q_GADGET
491     public:
492         EverCloudLocalData localData;
493
494         Optional<bool> includeTitle;
495         Optional<bool> includeContentLength;

```



```

589     Optional<bool> includeCreated;
591     Optional<bool> includeUpdated;
593     Optional<bool> includeDeleted;
595     Optional<bool> includeUpdateSequenceNum;
597     Optional<bool> includeNotebookGuid;
599     Optional<bool> includeTagGuids;
601     Optional<bool> includeAttributes;
603     Optional<bool> includeLargestResourceMime;
605     Optional<bool> includeLargestResourceSize;
606
607     virtual void print(QTextStream & strm) const override;
608
609     bool operator==(const NotesMetadataResultSpec & other) const
610     {
611         return includeTitle.isEqual(other.includeTitle)
612             && includeContentLength.isEqual(other.includeContentLength)
613             && includeCreated.isEqual(other.includeCreated)
614             && includeUpdated.isEqual(other.includeUpdated)
615             && includeDeleted.isEqual(other.includeDeleted)
616             && includeUpdateSequenceNum.isEqual(other.includeUpdateSequenceNum)
617             && includeNotebookGuid.isEqual(other.includeNotebookGuid)
618             && includeTagGuids.isEqual(other.includeTagGuids)
619             && includeAttributes.isEqual(other.includeAttributes)
620             && includeLargestResourceMime.isEqual(other.includeLargestResourceMime)
621             && includeLargestResourceSize.isEqual(other.includeLargestResourceSize)
622     };
623     }
624
625     bool operator!=(const NotesMetadataResultSpec & other) const
626     {
627         return !(*this == other);
628     }
629
630     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
631     Q_PROPERTY(Optional<bool> includeTitle MEMBER includeTitle)
632     Q_PROPERTY(Optional<bool> includeContentLength MEMBER includeContentLength)
633     Q_PROPERTY(Optional<bool> includeCreated MEMBER includeCreated)
634     Q_PROPERTY(Optional<bool> includeUpdated MEMBER includeUpdated)
635     Q_PROPERTY(Optional<bool> includeDeleted MEMBER includeDeleted)
636     Q_PROPERTY(Optional<bool> includeUpdateSequenceNum MEMBER includeUpdateSequenceNum)
637     Q_PROPERTY(Optional<bool> includeNotebookGuid MEMBER includeNotebookGuid)
638     Q_PROPERTY(Optional<bool> includeTagGuids MEMBER includeTagGuids)
639     Q_PROPERTY(Optional<bool> includeAttributes MEMBER includeAttributes)
640     Q_PROPERTY(Optional<bool> includeLargestResourceMime MEMBER includeLargestResourceMime)
641     Q_PROPERTY(Optional<bool> includeLargestResourceSize MEMBER includeLargestResourceSize)
642 };
643
644 struct QEVERCLOUD_EXPORT NoteCollectionCounts: public Printable
645 {
646 private:
647     Q_GADGET
648 public:
649     EverCloudLocalData localData;
650
651     Optional<QMap<Guid, qint32>> notebookCounts;
652     Optional<QMap<Guid, qint32>> tagCounts;
653     Optional<qint32> trashCount;
654
655     virtual void print(QTextStream & strm) const override;
656
657     bool operator==(const NoteCollectionCounts & other) const
658     {
659         return notebookCounts.isEqual(other.notebookCounts)
660             && tagCounts.isEqual(other.tagCounts)
661             && trashCount.isEqual(other.trashCount)
662     };
663     }
664
665     bool operator!=(const NoteCollectionCounts & other) const
666     {
667         return !(*this == other);
668     }
669
670     using TagCounts = QMap<Guid, qint32>;
671
672     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
673     Q_PROPERTY(Optional<TagCounts> notebookCounts MEMBER notebookCounts)
674     Q_PROPERTY(Optional<TagCounts> tagCounts MEMBER tagCounts)
675     Q_PROPERTY(Optional<qint32> trashCount MEMBER trashCount)
676 };
677
678 struct QEVERCLOUD_EXPORT NoteResultSpec: public Printable
679 {
680 private:
681     Q_GADGET
682 public:
683     EverCloudLocalData localData;

```

```

719
723     Optional<bool> includeContent;
728     Optional<bool> includeResourcesData;
733     Optional<bool> includeResourcesRecognition;
738     Optional<bool> includeResourcesAlternateData;
742     Optional<bool> includeSharedNotes;
746     Optional<bool> includeNoteAppDataValues;
750     Optional<bool> includeResourceAppDataValues;
754     Optional<bool> includeAccountLimits;
755
756     virtual void print(QTextStream & strm) const override;
757
758     bool operator==(const NoteResultSpec & other) const
759     {
760         return includeContent.isEqual(other.includeContent)
761             && includeResourcesData.isEqual(other.includeResourcesData)
762             && includeResourcesRecognition.isEqual(other.includeResourcesRecognition)
763             && includeResourcesAlternateData.isEqual(other.includeResourcesAlternateData)
764             && includeSharedNotes.isEqual(other.includeSharedNotes)
765             && includeNoteAppDataValues.isEqual(other.includeNoteAppDataValues)
766             && includeResourceAppDataValues.isEqual(other.includeResourceAppDataValues)
767             && includeAccountLimits.isEqual(other.includeAccountLimits)
768         ;
769     }
770
771     bool operator!=(const NoteResultSpec & other) const
772     {
773         return !(*this == other);
774     }
775
776     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
777     Q_PROPERTY(Optional<bool> includeContent MEMBER includeContent)
778     Q_PROPERTY(Optional<bool> includeResourcesData MEMBER includeResourcesData)
779     Q_PROPERTY(Optional<bool> includeResourcesRecognition MEMBER includeResourcesRecognition)
780     Q_PROPERTY(Optional<bool> includeResourcesAlternateData MEMBER includeResourcesAlternateData)
781     Q_PROPERTY(Optional<bool> includeSharedNotes MEMBER includeSharedNotes)
782     Q_PROPERTY(Optional<bool> includeNoteAppDataValues MEMBER includeNoteAppDataValues)
783     Q_PROPERTY(Optional<bool> includeResourceAppDataValues MEMBER includeResourceAppDataValues)
784     Q_PROPERTY(Optional<bool> includeAccountLimits MEMBER includeAccountLimits)
785 };
786
793 struct QEVERCLOUD_EXPORT NoteVersionId: public Printable
794 {
795 private:
796     Q_GADGET
797 public:
801     EverCloudLocalData localData;
802
808     qint32 updateSequenceNum = 0;
816     Timestamp updated = 0;
821     Timestamp saved = 0;
826     QString title;
831     Optional<UserID> lastEditorId;
832
833     virtual void print(QTextStream & strm) const override;
834
835     bool operator==(const NoteVersionId & other) const
836     {
837         return (updateSequenceNum == other.updateSequenceNum)
838             && (updated == other.updated)
839             && (saved == other.saved)
840             && (title == other.title)
841             && lastEditorId.isEqual(other.lastEditorId)
842         ;
843     }
844
845     bool operator!=(const NoteVersionId & other) const
846     {
847         return !(*this == other);
848     }
849
850     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
851     Q_PROPERTY(qint32 updateSequenceNum MEMBER updateSequenceNum)
852     Q_PROPERTY(Timestamp updated MEMBER updated)
853     Q_PROPERTY(Timestamp saved MEMBER saved)
854     Q_PROPERTY(QString title MEMBER title)
855     Q_PROPERTY(Optional<UserID> lastEditorId MEMBER lastEditorId)
856 };
857
866 struct QEVERCLOUD_EXPORT RelatedQuery: public Printable
867 {
868 private:
869     Q_GADGET
870 public:
874     EverCloudLocalData localData;
875
880     Optional<QString> noteGuid;

```

```

886     Optional<QString> plainText;
893     Optional<NoteFilter> filter;
898     Optional<QString> referenceUri;
904     Optional<QString> context;
916     Optional<QString> cacheKey;
917
918     virtual void print(QTextStream & strm) const override;
919
920     bool operator==(const RelatedQuery & other) const
921     {
922         return noteGuid.isEqual(other.noteGuid)
923             && plainText.isEqual(other.plainText)
924             && filter.isEqual(other.filter)
925             && referenceUri.isEqual(other.referenceUri)
926             && context.isEqual(other.context)
927             && cacheKey.isEqual(other.cacheKey)
928         ;
929     }
930
931     bool operator!=(const RelatedQuery & other) const
932     {
933         return !(*this == other);
934     }
935
936     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
937     Q_PROPERTY(Optional<QString> noteGuid MEMBER noteGuid)
938     Q_PROPERTY(Optional<QString> plainText MEMBER plainText)
939     Q_PROPERTY(Optional<NoteFilter> filter MEMBER filter)
940     Q_PROPERTY(Optional<QString> referenceUri MEMBER referenceUri)
941     Q_PROPERTY(Optional<QString> context MEMBER context)
942     Q_PROPERTY(Optional<QString> cacheKey MEMBER cacheKey)
943 };
944
952 struct QEVERCLOUD_EXPORT RelatedResultSpec: public Printable
953 {
954     private:
955         Q_GADGET
956     public:
957         EverCloudLocalData localData;
958
959         Optional<qint32> maxNotes;
960         Optional<qint32> maxNotebooks;
961         Optional<qint32> maxTags;
962         Optional<bool> writableNotebooksOnly;
963         Optional<bool> includeContainingNotebooks;
1001        Optional<bool> includeDebugInfo;
1008        Optional<qint32> maxExperts;
1014        Optional<qint32> maxRelatedContent;
1018        Optional<QSet<RelatedContentType>> relatedContentTypes;
1019
1020        virtual void print(QTextStream & strm) const override;
1021
1022        bool operator==(const RelatedResultSpec & other) const
1023        {
1024            return maxNotes.isEqual(other.maxNotes)
1025                && maxNotebooks.isEqual(other.maxNotebooks)
1026                && maxTags.isEqual(other.maxTags)
1027                && writableNotebooksOnly.isEqual(other.writableNotebooksOnly)
1028                && includeContainingNotebooks.isEqual(other.includeContainingNotebooks)
1029                && includeDebugInfo.isEqual(other.includeDebugInfo)
1030                && maxExperts.isEqual(other.maxExperts)
1031                && maxRelatedContent.isEqual(other.maxRelatedContent)
1032                && relatedContentTypes.isEqual(other.relatedContentTypes)
1033            ;
1034        }
1035
1036        bool operator!=(const RelatedResultSpec & other) const
1037        {
1038            return !(*this == other);
1039        }
1040
1041        Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
1042        Q_PROPERTY(Optional<qint32> maxNotes MEMBER maxNotes)
1043        Q_PROPERTY(Optional<qint32> maxNotebooks MEMBER maxNotebooks)
1044        Q_PROPERTY(Optional<qint32> maxTags MEMBER maxTags)
1045        Q_PROPERTY(Optional<bool> writableNotebooksOnly MEMBER writableNotebooksOnly)
1046        Q_PROPERTY(Optional<bool> includeContainingNotebooks MEMBER includeContainingNotebooks)
1047        Q_PROPERTY(Optional<bool> includeDebugInfo MEMBER includeDebugInfo)
1048        Q_PROPERTY(Optional<qint32> maxExperts MEMBER maxExperts)
1049        Q_PROPERTY(Optional<qint32> maxRelatedContent MEMBER maxRelatedContent)
1050        Q_PROPERTY(Optional<QSet<RelatedContentType>> relatedContentTypes MEMBER relatedContentTypes)
1051    };
1052
1054 struct QEVERCLOUD_EXPORT ShareRelationshipRestrictions: public Printable
1055 {
1056     private:
1057         Q_GADGET

```

```

1058 public:
1062     EverCloudLocalData localData;
1063
1065     Optional<bool> noSetReadOnly;
1067     Optional<bool> noSetReadPlusActivity;
1069     Optional<bool> noSetModify;
1071     Optional<bool> noSetFullAccess;
1072
1073     virtual void print(QTextStream & strm) const override;
1074
1075     bool operator==(const ShareRelationshipRestrictions & other) const
1076     {
1077         return noSetReadOnly.isEqual(other.noSetReadOnly)
1078             && noSetReadPlusActivity.isEqual(other.noSetReadPlusActivity)
1079             && noSetModify.isEqual(other.noSetModify)
1080             && noSetFullAccess.isEqual(other.noSetFullAccess)
1081         ;
1082     }
1083
1084     bool operator!=(const ShareRelationshipRestrictions & other) const
1085     {
1086         return !(*this == other);
1087     }
1088
1089     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
1090     Q_PROPERTY(Optional<bool> noSetReadOnly MEMBER noSetReadOnly)
1091     Q_PROPERTY(Optional<bool> noSetReadPlusActivity MEMBER noSetReadPlusActivity)
1092     Q_PROPERTY(Optional<bool> noSetModify MEMBER noSetModify)
1093     Q_PROPERTY(Optional<bool> noSetFullAccess MEMBER noSetFullAccess)
1094 };
1095
1101 struct QEVERCLOUD_EXPORT MemberShareRelationship: public Printable
1102 {
1103 private:
1104     Q_GADGET
1105 public:
1106     EverCloudLocalData localData;
1107
1108     Optional<QString> displayName;
1109     Optional<UserID> recipientUserId;
1110     Optional<ShareRelationshipPrivilegeLevel> bestPrivilege;
1111     Optional<ShareRelationshipPrivilegeLevel> individualPrivilege;
1112     Optional<ShareRelationshipRestrictions> restrictions;
1113     Optional<UserID> sharerUserId;
1114
1115     virtual void print(QTextStream & strm) const override;
1116
1117     bool operator==(const MemberShareRelationship & other) const
1118     {
1119         return displayName.isEqual(other.displayName)
1120             && recipientUserId.isEqual(other.recipientUserId)
1121             && bestPrivilege.isEqual(other.bestPrivilege)
1122             && individualPrivilege.isEqual(other.individualPrivilege)
1123             && restrictions.isEqual(other.restrictions)
1124             && sharerUserId.isEqual(other.sharerUserId)
1125         ;
1126     }
1127
1128     bool operator!=(const MemberShareRelationship & other) const
1129     {
1130         return !(*this == other);
1131     }
1132
1133     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
1134     Q_PROPERTY(Optional<QString> displayName MEMBER displayName)
1135     Q_PROPERTY(Optional<UserID> recipientUserId MEMBER recipientUserId)
1136     Q_PROPERTY(Optional<ShareRelationshipPrivilegeLevel> bestPrivilege MEMBER bestPrivilege)
1137     Q_PROPERTY(Optional<ShareRelationshipPrivilegeLevel> individualPrivilege MEMBER
1138         individualPrivilege)
1139     Q_PROPERTY(Optional<ShareRelationshipRestrictions> restrictions MEMBER restrictions)
1140     Q_PROPERTY(Optional<UserID> sharerUserId MEMBER sharerUserId)
1141 };
1142
1143 struct QEVERCLOUD_EXPORT NoteShareRelationshipRestrictions: public Printable
1144 {
1145 private:
1146     Q_GADGET
1147 public:
1148     EverCloudLocalData localData;
1149
1150     Optional<bool> noSetReadNote;
1151     Optional<bool> noSetModifyNote;
1152     Optional<bool> noSetFullAccess;
1153
1154     virtual void print(QTextStream & strm) const override;
1155
1156     bool operator==(const NoteShareRelationshipRestrictions & other) const

```

```

1215 {
1216     return noSetReadNote.isEqual(other.noSetReadNote)
1217         && noSetModifyNote.isEqual(other.noSetModifyNote)
1218         && noSetFullAccess.isEqual(other.noSetFullAccess)
1219     ;
1220 }
1221
1222 bool operator!=(const NoteShareRelationshipRestrictions & other) const
1223 {
1224     return !(*this == other);
1225 }
1226
1227 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
1228 Q_PROPERTY(Optional<bool> noSetReadNote MEMBER noSetReadNote)
1229 Q_PROPERTY(Optional<bool> noSetModifyNote MEMBER noSetModifyNote)
1230 Q_PROPERTY(Optional<bool> noSetFullAccess MEMBER noSetFullAccess)
1231 };
1232
1233 struct QEVERCLOUD_EXPORT NoteMemberShareRelationship: public Printable
1234 {
1235 private:
1236     Q_GADGET
1237 public:
1238     EverCloudLocalData localData;
1239
1240     Optional<QString> displayName;
1241     Optional<UserID> recipientUserId;
1242     Optional<SharedNotePrivilegeLevel> privilege;
1243     Optional<NoteShareRelationshipRestrictions> restrictions;
1244     Optional<UserID> sharerUserId;
1245
1246     virtual void print(QTextStream & strm) const override;
1247
1248     bool operator==(const NoteMemberShareRelationship & other) const
1249     {
1250         return displayName.isEqual(other.displayName)
1251             && recipientUserId.isEqual(other.recipientUserId)
1252             && privilege.isEqual(other.privilege)
1253             && restrictions.isEqual(other.restrictions)
1254             && sharerUserId.isEqual(other.sharerUserId)
1255         ;
1256     }
1257
1258     bool operator!=(const NoteMemberShareRelationship & other) const
1259     {
1260         return !(*this == other);
1261     }
1262
1263     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
1264     Q_PROPERTY(Optional<QString> displayName MEMBER displayName)
1265     Q_PROPERTY(Optional<UserID> recipientUserId MEMBER recipientUserId)
1266     Q_PROPERTY(Optional<SharedNotePrivilegeLevel> privilege MEMBER privilege)
1267     Q_PROPERTY(Optional<NoteShareRelationshipRestrictions> restrictions MEMBER restrictions)
1268     Q_PROPERTY(Optional<UserID> sharerUserId MEMBER sharerUserId)
1269 };
1270
1271 struct QEVERCLOUD_EXPORT NoteInvitationShareRelationship: public Printable
1272 {
1273 private:
1274     Q_GADGET
1275 public:
1276     EverCloudLocalData localData;
1277
1278     Optional<QString> displayName;
1279     Optional<IdentityID> recipientIdentityId;
1280     Optional<SharedNotePrivilegeLevel> privilege;
1281     Optional<UserID> sharerUserId;
1282
1283     virtual void print(QTextStream & strm) const override;
1284
1285     bool operator==(const NoteInvitationShareRelationship & other) const
1286     {
1287         return displayName.isEqual(other.displayName)
1288             && recipientIdentityId.isEqual(other.recipientIdentityId)
1289             && privilege.isEqual(other.privilege)
1290             && sharerUserId.isEqual(other.sharerUserId)
1291         ;
1292     }
1293
1294     bool operator!=(const NoteInvitationShareRelationship & other) const
1295     {
1296         return !(*this == other);
1297     }
1298
1299     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
1300     Q_PROPERTY(Optional<QString> displayName MEMBER displayName)
1301     Q_PROPERTY(Optional<IdentityID> recipientIdentityId MEMBER recipientIdentityId)
1302 };

```

```

1364     Q_PROPERTY(Optional<SharedNotePrivilegeLevel> privilege MEMBER privilege)
1365     Q_PROPERTY(Optional<UserID> sharerUserId MEMBER sharerUserId)
1366 };
1367
1368 struct QEVERCLOUD_EXPORT NoteShareRelationships: public Printable
1369 {
1370 private:
1371     Q_GADGET
1372 public:
1373     EverCloudLocalData localData;
1374
1375     Optional<QList<NoteInvitationShareRelationship>> invitations;
1376     Optional<QList<NoteMemberShareRelationship>> memberships;
1377     Optional<NoteShareRelationshipRestrictions> invitationRestrictions;
1378
1379     virtual void print(QTextStream & strm) const override;
1380
1381     bool operator==(const NoteShareRelationships & other) const
1382     {
1383         return invitations.isEqual(other.invitations)
1384             && memberships.isEqual(other.memberships)
1385             && invitationRestrictions.isEqual(other.invitationRestrictions);
1386     }
1387
1388     bool operator!=(const NoteShareRelationships & other) const
1389     {
1390         return !(*this == other);
1391     }
1392
1393     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
1394     Q_PROPERTY(Optional<QList<NoteInvitationShareRelationship>> invitations MEMBER invitations)
1395     Q_PROPERTY(Optional<QList<NoteMemberShareRelationship>> memberships MEMBER memberships)
1396     Q_PROPERTY(Optional<NoteShareRelationshipRestrictions> invitationRestrictions MEMBER
1397         invitationRestrictions)
1398 };
1399
1400 struct QEVERCLOUD_EXPORT ManageNoteSharesParameters: public Printable
1401 {
1402 private:
1403     Q_GADGET
1404 public:
1405     EverCloudLocalData localData;
1406
1407     Optional<QString> noteGuid;
1408     Optional<QList<NoteMemberShareRelationship>> membershipsToUpdate;
1409     Optional<QList<NoteInvitationShareRelationship>> invitationsToUpdate;
1410     Optional<QList<UserID>> membershipsToUnshare;
1411     Optional<QList<IdentityID>> invitationsToUnshare;
1412
1413     virtual void print(QTextStream & strm) const override;
1414
1415     bool operator==(const ManageNoteSharesParameters & other) const
1416     {
1417         return noteGuid.isEqual(other.noteGuid)
1418             && membershipsToUpdate.isEqual(other.membershipsToUpdate)
1419             && invitationsToUpdate.isEqual(other.invitationsToUpdate)
1420             && membershipsToUnshare.isEqual(other.membershipsToUnshare)
1421             && invitationsToUnshare.isEqual(other.invitationsToUnshare);
1422     }
1423
1424     bool operator!=(const ManageNoteSharesParameters & other) const
1425     {
1426         return !(*this == other);
1427     }
1428
1429     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
1430     Q_PROPERTY(Optional<QString> noteGuid MEMBER noteGuid)
1431     Q_PROPERTY(Optional<QList<NoteMemberShareRelationship>> membershipsToUpdate MEMBER
1432         membershipsToUpdate)
1433     Q_PROPERTY(Optional<QList<NoteInvitationShareRelationship>> invitationsToUpdate MEMBER
1434         invitationsToUpdate)
1435     Q_PROPERTY(Optional<QList<UserID>> membershipsToUnshare MEMBER membershipsToUnshare)
1436     Q_PROPERTY(Optional<QList<IdentityID>> invitationsToUnshare MEMBER invitationsToUnshare)
1437 };
1438
1439 struct QEVERCLOUD_EXPORT Data: public Printable
1440 {
1441 private:
1442     Q_GADGET
1443 public:
1444     EverCloudLocalData localData;
1445
1446     Optional<QByteArray> bodyHash;
1447     Optional<qint32> size;
1448     Optional<QByteArray> body;
1449 };

```

```

1531
1532     virtual void print(QTextStream & strm) const override;
1533
1534     bool operator==(const Data & other) const
1535 {
1536     return bodyHash.isEqual(other.bodyHash)
1537         && size.isEqual(other.size)
1538         && body.isEqual(other.body)
1539     ;
1540 }
1541
1542     bool operator!=(const Data & other) const
1543 {
1544     return !(*this == other);
1545 }
1546
1547     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
1548     Q_PROPERTY(Optional<QByteArray> bodyHash MEMBER bodyHash)
1549     Q_PROPERTY(Optional<qint32> size MEMBER size)
1550     Q_PROPERTY(Optional<QByteArray> body MEMBER body)
1551 };
1552
1553 struct QEVERCLOUD_EXPORT UserAttributes: public Printable
1554 {
1555 private:
1556     Q_GADGET
1557 public:
1558     EverCloudLocalData localData;
1559
1560     Optional<QString> defaultLocationName;
1561     Optional<double> defaultLatitude;
1562     Optional<double> defaultLongitude;
1563     Optional<bool> preactivation;
1564     Optional<QStringList> viewedPromotions;
1565     Optional<QString> incomingEmailAddress;
1566     Optional<QStringList> recentMailedAddresses;
1567     Optional<QString> comments;
1568     Optional<Timestamp> dateAgreedToTermsOfService;
1569     Optional<qint32> maxReferrals;
1570     Optional<qint32> referralCount;
1571     Optional<QString> referrerCode;
1572     Optional<Timestamp> sentEmailDate;
1573     Optional<qint32> sentEmailCount;
1574     Optional<qint32> dailyEmailLimit;
1575     Optional<Timestamp> emailOptOutDate;
1576     Optional<Timestamp> partnerEmailOptInDate;
1577     Optional<QString> preferredLanguage;
1578     Optional<QString> preferredCountry;
1579     Optional<bool> clipFullPage;
1580     Optional<QString> twitterUserName;
1581     Optional<QString> twitterId;
1582     Optional<QString> groupName;
1583     Optional<QString> recognitionLanguage;
1584     Optional<QString> referralProof;
1585     Optional<bool> educationalDiscount;
1586     Optional<QString> businessAddress;
1587     Optional<bool> hideSponsorBilling;
1588     Optional<bool> useEmailAutoFiling;
1589     Optional<ReminderEmailConfig> reminderEmailConfig;
1590     Optional<Timestamp> emailAddressLastConfirmed;
1591     Optional<Timestamp> passwordUpdated;
1592     Optional<bool> salesforcePushEnabled;
1593     Optional<bool> shouldLogClientEvent;
1594     Optional<bool> optOutMachineLearning;
1595
1596     virtual void print(QTextStream & strm) const override;
1597
1598     bool operator==(const UserAttributes & other) const
1599 {
1600     return defaultLocationName.isEqual(other.defaultLocationName)
1601         && defaultLatitude.isEqual(other.defaultLatitude)
1602         && defaultLongitude.isEqual(other.defaultLongitude)
1603         && preactivation.isEqual(other.preactivation)
1604         && viewedPromotions.isEqual(other.viewedPromotions)
1605         && incomingEmailAddress.isEqual(other.incomingEmailAddress)
1606         && recentMailedAddresses.isEqual(other.recentMailedAddresses)
1607         && comments.isEqual(other.comments)
1608         && dateAgreedToTermsOfService.isEqual(other.dateAgreedToTermsOfService)
1609         && maxReferrals.isEqual(other.maxReferrals)
1610         && referralCount.isEqual(other.referralCount)
1611         && referrerCode.isEqual(other.referrerCode)
1612         && sentEmailDate.isEqual(other.sentEmailDate)
1613         && sentEmailCount.isEqual(other.sentEmailCount)
1614         && dailyEmailLimit.isEqual(other.dailyEmailLimit)
1615         && emailOptOutDate.isEqual(other.emailOptOutDate)
1616         && partnerEmailOptInDate.isEqual(other.partnerEmailOptInDate)
1617         && preferredLanguage.isEqual(other.preferredLanguage)

```

```

1788         && preferredCountry.isEqual(other.preferredCountry)
1789         && clipFullPage.isEqual(other.clipFullPage)
1790         && twitterUserName.isEqual(other.twitterUserName)
1791         && twitterId.isEqual(other.twitterId)
1792         && groupName.isEqual(other.groupName)
1793         && recognitionLanguage.isEqual(other.recognitionLanguage)
1794         && referralProof.isEqual(other.referralProof)
1795         && educationalDiscount.isEqual(other.educationalDiscount)
1796         && businessAddress.isEqual(other.businessAddress)
1797         && hideSponsorBilling.isEqual(other.hideSponsorBilling)
1798         && useEmailAutoFiling.isEqual(other.useEmailAutoFiling)
1799         && reminderEmailConfig.isEqual(other.reminderEmailConfig)
1800         && emailAddressLastConfirmed.isEqual(other.emailAddressLastConfirmed)
1801         && passwordUpdated.isEqual(other.passwordUpdated)
1802         && salesforcePushEnabled.isEqual(other.salesforcePushEnabled)
1803         && shouldLogClientEvent.isEqual(other.shouldLogClientEvent)
1804         && optOutMachineLearning.isEqual(other.optOutMachineLearning)
1805     };
1806 }
1807
1808 bool operator!=(const UserAttributes & other) const
1809 {
1810     return !(*this == other);
1811 }
1812
1813 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
1814 Q_PROPERTY(Optional<QString> defaultLocationName MEMBER defaultLocationName)
1815 Q_PROPERTY(Optional<double> defaultLatitude MEMBER defaultLatitude)
1816 Q_PROPERTY(Optional<double> defaultLongitude MEMBER defaultLongitude)
1817 Q_PROPERTY(Optional<bool> preactivation MEMBER preactivation)
1818 Q_PROPERTY(Optional<QStringList> viewedPromotions MEMBER viewedPromotions)
1819 Q_PROPERTY(Optional<QString> incomingEmailAddress MEMBER incomingEmailAddress)
1820 Q_PROPERTY(Optional<QStringList> recentMailedAddresses MEMBER recentMailedAddresses)
1821 Q_PROPERTY(Optional<QString> comments MEMBER comments)
1822 Q_PROPERTY(Optional<Timestamp> dateAgreedToTermsOfService MEMBER dateAgreedToTermsOfService)
1823 Q_PROPERTY(Optional<qint32> maxReferrals MEMBER maxReferrals)
1824 Q_PROPERTY(Optional<qint32> referralCount MEMBER referralCount)
1825 Q_PROPERTY(Optional<QString> refererCode MEMBER refererCode)
1826 Q_PROPERTY(Optional<Timestamp> sentEmailDate MEMBER sentEmailDate)
1827 Q_PROPERTY(Optional<qint32> sentEmailCount MEMBER sentEmailCount)
1828 Q_PROPERTY(Optional<qint32> dailyEmailLimit MEMBER dailyEmailLimit)
1829 Q_PROPERTY(Optional<Timestamp> emailOptOutDate MEMBER emailOptOutDate)
1830 Q_PROPERTY(Optional<Timestamp> partnerEmailOptInDate MEMBER partnerEmailOptInDate)
1831 Q_PROPERTY(Optional<QString> preferredLanguage MEMBER preferredLanguage)
1832 Q_PROPERTY(Optional<QString> preferredCountry MEMBER preferredCountry)
1833 Q_PROPERTY(Optional<bool> clipFullPage MEMBER clipFullPage)
1834 Q_PROPERTY(Optional<QString> twitterUserName MEMBER twitterUserName)
1835 Q_PROPERTY(Optional<QString> twitterId MEMBER twitterId)
1836 Q_PROPERTY(Optional<QString> groupName MEMBER groupName)
1837 Q_PROPERTY(Optional<QString> recognitionLanguage MEMBER recognitionLanguage)
1838 Q_PROPERTY(Optional<Timestamp> referralProof MEMBER referralProof)
1839 Q_PROPERTY(Optional<bool> educationalDiscount MEMBER educationalDiscount)
1840 Q_PROPERTY(Optional<QString> businessAddress MEMBER businessAddress)
1841 Q_PROPERTY(Optional<bool> hideSponsorBilling MEMBER hideSponsorBilling)
1842 Q_PROPERTY(Optional<bool> useEmailAutoFiling MEMBER useEmailAutoFiling)
1843 Q_PROPERTY(Optional<ReminderEmailConfig> reminderEmailConfig MEMBER reminderEmailConfig)
1844 Q_PROPERTY(Optional<Timestamp> emailAddressLastConfirmed MEMBER emailAddressLastConfirmed)
1845 Q_PROPERTY(Optional<Timestamp> passwordUpdated MEMBER passwordUpdated)
1846 Q_PROPERTY(Optional<bool> salesforcePushEnabled MEMBER salesforcePushEnabled)
1847 Q_PROPERTY(Optional<bool> shouldLogClientEvent MEMBER shouldLogClientEvent)
1848 Q_PROPERTY(Optional<bool> optOutMachineLearning MEMBER optOutMachineLearning)
1849 };
1850
1851 struct QEVERCLOUD_EXPORT BusinessUserAttributes: public Printable
1852 {
1853 private:
1854     Q_GADGET
1855 public:
1856     EverCloudLocalData localData;
1857
1858     Optional<QString> title;
1859     Optional<QString> location;
1860     Optional<QString> department;
1861     Optional<QString> mobilePhone;
1862     Optional<QString> linkedInProfileUrl;
1863     Optional<QString> workPhone;
1864     Optional<Timestamp> companyStartDate;
1865
1866     virtual void print(QTextStream & strm) const override;
1867
1868     bool operator==(const BusinessUserAttributes & other) const
1869     {
1870         return title.isEqual(other.title)
1871             && location.isEqual(other.location)
1872             && department.isEqual(other.department)
1873             && mobilePhone.isEqual(other.mobilePhone)
1874             && linkedInProfileUrl.isEqual(other.linkedInProfileUrl)

```



```

1905         && workPhone.isEqual(other.workPhone)
1906         && companyStartDate.isEqual(other.companyStartDate)
1907     ;
1908 }
1909
1910 bool operator!=(const BusinessUserAttributes & other) const
1911 {
1912     return !(*this == other);
1913 }
1914
1915 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
1916 Q_PROPERTY(Optional<QString> title MEMBER title)
1917 Q_PROPERTY(Optional<QString> location MEMBER location)
1918 Q_PROPERTY(Optional<QString> department MEMBER department)
1919 Q_PROPERTY(Optional<QString> mobilePhone MEMBER mobilePhone)
1920 Q_PROPERTY(Optional<QString> linkedInProfileUrl MEMBER linkedInProfileUrl)
1921 Q_PROPERTY(Optional<QString> workPhone MEMBER workPhone)
1922 Q_PROPERTY(Optional<Timestamp> companyStartDate MEMBER companyStartDate)
1923 };
1924
1929 struct QEVERCLOUD_EXPORT Accounting: public Printable
1930 {
1931 private:
1932     Q_GADGET
1933 public:
1934     EverCloudLocalData localData;
1935
1936     Optional<Timestamp> uploadLimitEnd;
1937     Optional<qint64> uploadLimitNextMonth;
1938     Optional<PremiumOrderStatus> premiumServiceStatus;
1939     Optional<QString> premiumOrderNumber;
1940     Optional<QString> premiumCommerceService;
1941     Optional<Timestamp> premiumServiceStart;
1942     Optional<QString> premiumServiceSKU;
1943     Optional<Timestamp> lastSuccessfulCharge;
1944     Optional<Timestamp> lastFailedCharge;
1945     Optional<QString> lastFailedChargeReason;
1946     Optional<Timestamp> nextPaymentDue;
1947     Optional<Timestamp> premiumLockUntil;
1948     Optional<Timestamp> updated;
1949     Optional<QString> premiumSubscriptionNumber;
1950     Optional<Timestamp> lastRequestedCharge;
1951     Optional<QString> currency;
1952     Optional<qint32> unitPrice;
1953     Optional<qint32> businessId;
1954     Optional<QString> businessName;
1955     Optional<BusinessUserRole> businessRole;
1956     Optional<qint32> unitDiscount;
1957     Optional<Timestamp> nextChargeDate;
1958     Optional<qint32> availablePoints;
1959
1960     virtual void print(QTextStream & strm) const override;
1961
1962     bool operator==(const Accounting & other) const
1963     {
1964         return uploadLimitEnd.isEqual(other.uploadLimitEnd)
1965             && uploadLimitNextMonth.isEqual(other.uploadLimitNextMonth)
1966             && premiumServiceStatus.isEqual(other.premiumServiceStatus)
1967             && premiumOrderNumber.isEqual(other.premiumOrderNumber)
1968             && premiumCommerceService.isEqual(other.premiumCommerceService)
1969             && premiumServiceStart.isEqual(other.premiumServiceStart)
1970             && premiumServiceSKU.isEqual(other.premiumServiceSKU)
1971             && lastSuccessfulCharge.isEqual(other.lastSuccessfulCharge)
1972             && lastFailedCharge.isEqual(other.lastFailedCharge)
1973             && lastFailedChargeReason.isEqual(other.lastFailedChargeReason)
1974             && nextPaymentDue.isEqual(other.nextPaymentDue)
1975             && premiumLockUntil.isEqual(other.premiumLockUntil)
1976             && updated.isEqual(other.updated)
1977             && premiumSubscriptionNumber.isEqual(other.premiumSubscriptionNumber)
1978             && lastRequestedCharge.isEqual(other.lastRequestedCharge)
1979             && currency.isEqual(other.currency)
1980             && unitPrice.isEqual(other.unitPrice)
1981             && businessId.isEqual(other.businessId)
1982             && businessName.isEqual(other.businessName)
1983             && businessRole.isEqual(other.businessRole)
1984             && unitDiscount.isEqual(other.unitDiscount)
1985             && nextChargeDate.isEqual(other.nextChargeDate)
1986             && availablePoints.isEqual(other.availablePoints)
1987         ;
1988     }
1989
1990     bool operator!=(const Accounting & other) const
1991     {
1992         return !(*this == other);
1993     }
1994
1995     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)

```

```

2084     Q_PROPERTY(Optional<Timestamp> uploadLimitEnd MEMBER uploadLimitEnd)
2085     Q_PROPERTY(Optional<qint64> uploadLimitNextMonth MEMBER uploadLimitNextMonth)
2086     Q_PROPERTY(Optional<PremiumOrderStatus> premiumServiceStatus MEMBER premiumServiceStatus)
2087     Q_PROPERTY(Optional<QString> premiumOrderNumber MEMBER premiumOrderNumber)
2088     Q_PROPERTY(Optional<QString> premiumCommerceService MEMBER premiumCommerceService)
2089     Q_PROPERTY(Optional<Timestamp> premiumServiceStart MEMBER premiumServiceStart)
2090     Q_PROPERTY(Optional<QString> premiumServiceSKU MEMBER premiumServiceSKU)
2091     Q_PROPERTY(Optional<Timestamp> lastSuccessfulCharge MEMBER lastSuccessfulCharge)
2092     Q_PROPERTY(Optional<Timestamp> lastFailedCharge MEMBER lastFailedCharge)
2093     Q_PROPERTY(Optional<QString> lastFailedChargeReason MEMBER lastFailedChargeReason)
2094     Q_PROPERTY(Optional<Timestamp> nextPaymentDue MEMBER nextPaymentDue)
2095     Q_PROPERTY(Optional<Timestamp> premiumLockUntil MEMBER premiumLockUntil)
2096     Q_PROPERTY(Optional<Timestamp> updated MEMBER updated)
2097     Q_PROPERTY(Optional<QString> premiumSubscriptionNumber MEMBER premiumSubscriptionNumber)
2098     Q_PROPERTY(Optional<Timestamp> lastRequestedCharge MEMBER lastRequestedCharge)
2099     Q_PROPERTY(Optional<QString> currency MEMBER currency)
2100     Q_PROPERTY(Optional<qint32> unitPrice MEMBER unitPrice)
2101     Q_PROPERTY(Optional<qint32> businessId MEMBER businessId)
2102     Q_PROPERTY(Optional<QString> businessName MEMBER businessName)
2103     Q_PROPERTY(Optional<BusinessUserRole> businessRole MEMBER businessRole)
2104     Q_PROPERTY(Optional<qint32> unitDiscount MEMBER unitDiscount)
2105     Q_PROPERTY(Optional<Timestamp> nextChargeDate MEMBER nextChargeDate)
2106     Q_PROPERTY(Optional<qint32> availablePoints MEMBER availablePoints)
2107 };
2108
2114 struct QEVERCLOUD_EXPORT BusinessUserInfo: public Printable
2115 {
2116 private:
2117     Q_GADGET
2118 public:
2119     EverCloudLocalData localData;
2120
2121     Optional<qint32> businessId;
2122     Optional<QString> businessName;
2123     Optional<BusinessUserRole> role;
2124     Optional<QString> email;
2125     Optional<Timestamp> updated;
2126
2127     virtual void print(QTextStream & strm) const override;
2128
2129     bool operator==(const BusinessUserInfo & other) const
2130     {
2131         return businessId.isEqual(other.businessId)
2132             && businessName.isEqual(other.businessName)
2133             && role.isEqual(other.role)
2134             && email.isEqual(other.email)
2135             && updated.isEqual(other.updated)
2136         ;
2137     }
2138
2139     bool operator!=(const BusinessUserInfo & other) const
2140     {
2141         return !(*this == other);
2142     }
2143
2144     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
2145     Q_PROPERTY(Optional<qint32> businessId MEMBER businessId)
2146     Q_PROPERTY(Optional<QString> businessName MEMBER businessName)
2147     Q_PROPERTY(Optional<BusinessUserRole> role MEMBER role)
2148     Q_PROPERTY(Optional<QString> email MEMBER email)
2149     Q_PROPERTY(Optional<Timestamp> updated MEMBER updated)
2150 };
2151
2152 struct QEVERCLOUD_EXPORT AccountLimits: public Printable
2153 {
2154 private:
2155     Q_GADGET
2156 public:
2157     EverCloudLocalData localData;
2158
2159     Optional<qint32> userMailLimitDaily;
2160     Optional<qint64> noteSizeMax;
2161     Optional<qint64> resourceSizeMax;
2162     Optional<qint32> userLinkedNotebookMax;
2163     Optional<qint64> uploadLimit;
2164     Optional<qint32> userNoteCountMax;
2165     Optional<qint32> userNotebookCountMax;
2166     Optional<qint32> userTagCountMax;
2167     Optional<qint32> noteTagCountMax;
2168     Optional<qint32> userSavedSearchesMax;
2169     Optional<qint32> noteResourceCountMax;
2170
2171     virtual void print(QTextStream & strm) const override;
2172
2173     bool operator==(const AccountLimits & other) const
2174     {
2175         return userMailLimitDaily.isEqual(other.userMailLimitDaily)

```

```

2249         && noteSizeMax.isEqual(other.noteSizeMax)
2250         && resourceSizeMax.isEqual(other.resourceSizeMax)
2251         && userLinkedNotebookMax.isEqual(other.userLinkedNotebookMax)
2252         && uploadLimit.isEqual(other.uploadLimit)
2253         && userNoteCountMax.isEqual(other.userNoteCountMax)
2254         && userNotebookCountMax.isEqual(other.userNotebookCountMax)
2255         && userTagCountMax.isEqual(other.userTagCountMax)
2256         && noteTagCountMax.isEqual(other.noteTagCountMax)
2257         && userSavedSearchesMax.isEqual(other.userSavedSearchesMax)
2258         && noteResourceCountMax.isEqual(other.noteResourceCountMax)
2259     };
2260 }
2261
2262 bool operator!=(const AccountLimits & other) const
2263 {
2264     return !(*this == other);
2265 }
2266
2267 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
2268 Q_PROPERTY(Optional<qint32> userMailLimitDaily MEMBER userMailLimitDaily)
2269 Q_PROPERTY(Optional<qint64> noteSizeMax MEMBER noteSizeMax)
2270 Q_PROPERTY(Optional<qint64> resourceSizeMax MEMBER resourceSizeMax)
2271 Q_PROPERTY(Optional<qint32> userLinkedNotebookMax MEMBER userLinkedNotebookMax)
2272 Q_PROPERTY(Optional<qint64> uploadLimit MEMBER uploadLimit)
2273 Q_PROPERTY(Optional<qint32> userNoteCountMax MEMBER userNoteCountMax)
2274 Q_PROPERTY(Optional<qint32> userNotebookCountMax MEMBER userNotebookCountMax)
2275 Q_PROPERTY(Optional<qint32> userTagCountMax MEMBER userTagCountMax)
2276 Q_PROPERTY(Optional<qint32> noteTagCountMax MEMBER noteTagCountMax)
2277 Q_PROPERTY(Optional<qint32> userSavedSearchesMax MEMBER userSavedSearchesMax)
2278 Q_PROPERTY(Optional<qint32> noteResourceCountMax MEMBER noteResourceCountMax)
2279 };
2280
2284 struct QEVERCLOUD_EXPORT User: public Printable
2285 {
2286 private:
2287     Q_GADGET
2288 public:
2289     EverCloudLocalData localData;
2290
2291     Optional<UserID> id;
2292     Optional<QString> username;
2293     Optional<QString> email;
2294     Optional<QString> name;
2295     Optional<QString> timezone;
2296     Optional<PrivilegeLevel> privilege;
2297     Optional<ServiceLevel> serviceLevel;
2298     Optional<Timestamp> created;
2299     Optional<Timestamp> updated;
2300     Optional<Timestamp> deleted;
2301     Optional<bool> active;
2302     Optional<QString> shardId;
2303     Optional<UserAttributes> attributes;
2304     Optional<Accounting> accounting;
2305     Optional<BusinessUserInfo> businessUserInfo;
2306     Optional<QString> photoUrl;
2307     Optional<Timestamp> photoLastUpdated;
2308     Optional<AccountLimits> accountLimits;
2309
2310     virtual void print(QTextStream & strm) const override;
2311
2312 bool operator==(const User & other) const
2313 {
2314     return id.isEqual(other.id)
2315         && username.isEqual(other.username)
2316         && email.isEqual(other.email)
2317         && name.isEqual(other.name)
2318         && timezone.isEqual(other.timezone)
2319         && privilege.isEqual(other.privilege)
2320         && serviceLevel.isEqual(other.serviceLevel)
2321         && created.isEqual(other.created)
2322         && updated.isEqual(other.updated)
2323         && deleted.isEqual(other.deleted)
2324         && active.isEqual(other.active)
2325         && shardId.isEqual(other.shardId)
2326         && attributes.isEqual(other.attributes)
2327         && accounting.isEqual(other.accounting)
2328         && businessUserInfo.isEqual(other.businessUserInfo)
2329         && photoUrl.isEqual(other.photoUrl)
2330         && photoLastUpdated.isEqual(other.photoLastUpdated)
2331         && accountLimits.isEqual(other.accountLimits)
2332     ;
2333 }
2334
2335 bool operator!=(const User & other) const
2336 {
2337     return !(*this == other);
2338 }

```

```

2438
2439     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
2440     Q_PROPERTY(Optional<UserID> id MEMBER id)
2441     Q_PROPERTY(Optional<QString> username MEMBER username)
2442     Q_PROPERTY(Optional<QString> email MEMBER email)
2443     Q_PROPERTY(Optional<QString> name MEMBER name)
2444     Q_PROPERTY(Optional<QString> timezone MEMBER timezone)
2445     Q_PROPERTY(Optional<PrivilegeLevel> privilege MEMBER privilege)
2446     Q_PROPERTY(Optional<ServiceLevel> serviceLevel MEMBER serviceLevel)
2447     Q_PROPERTY(Optional<Timestamp> created MEMBER created)
2448     Q_PROPERTY(Optional<Timestamp> updated MEMBER updated)
2449     Q_PROPERTY(Optional<Timestamp> deleted MEMBER deleted)
2450     Q_PROPERTY(Optional<bool> active MEMBER active)
2451     Q_PROPERTY(Optional<QString> shardId MEMBER shardId)
2452     Q_PROPERTY(Optional<UserAttributes> attributes MEMBER attributes)
2453     Q_PROPERTY(Optional<Accounting> accounting MEMBER accounting)
2454     Q_PROPERTY(Optional<BusinessUserInfo> businessUserInfo MEMBER businessUserInfo)
2455     Q_PROPERTY(Optional<QString> photoUrl MEMBER photoUrl)
2456     Q_PROPERTY(Optional<Timestamp> photoLastUpdated MEMBER photoLastUpdated)
2457     Q_PROPERTY(Optional<AccountLimits> accountLimits MEMBER accountLimits)
2458 };
2459
2460 struct QEVERCLOUD_EXPORT Contact: public Printable
2461 {
2462 private:
2463     Q_GADGET
2464 public:
2465     EverCloudLocalData localData;
2466
2467     Optional<QString> name;
2468     Optional<QString> id;
2469     Optional<ContactType> type;
2470     Optional<QString> photoUrl;
2471     Optional<Timestamp> photoLastUpdated;
2472     Optional<QByteArray> messagingPermit;
2473     Optional<Timestamp> messagingPermitExpires;
2474
2475     virtual void print(QTextStream & strm) const override;
2476
2477     bool operator==(const Contact & other) const
2478     {
2479         return name.isEqual(other.name)
2480             && id.isEqual(other.id)
2481             && type.isEqual(other.type)
2482             && photoUrl.isEqual(other.photoUrl)
2483             && photoLastUpdated.isEqual(other.photoLastUpdated)
2484             && messagingPermit.isEqual(other.messagingPermit)
2485             && messagingPermitExpires.isEqual(other.messagingPermitExpires)
2486     };
2487
2488     bool operator!=(const Contact & other) const
2489     {
2490         return !(*this == other);
2491     }
2492
2493     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
2494     Q_PROPERTY(Optional<QString> name MEMBER name)
2495     Q_PROPERTY(Optional<QString> id MEMBER id)
2496     Q_PROPERTY(Optional<ContactType> type MEMBER type)
2497     Q_PROPERTY(Optional<QString> photoUrl MEMBER photoUrl)
2498     Q_PROPERTY(Optional<Timestamp> photoLastUpdated MEMBER photoLastUpdated)
2499     Q_PROPERTY(Optional<QByteArray> messagingPermit MEMBER messagingPermit)
2500     Q_PROPERTY(Optional<Timestamp> messagingPermitExpires MEMBER messagingPermitExpires)
2501 };
2502
2503 struct QEVERCLOUD_EXPORT Identity: public Printable
2504 {
2505 private:
2506     Q_GADGET
2507 public:
2508     EverCloudLocalData localData;
2509
2510     IdentityID id = 0;
2511     Optional<Contact> contact;
2512     Optional<UserID> userId;
2513     Optional<bool> deactivated;
2514     Optional<bool> sameBusiness;
2515     Optional<bool> blocked;
2516     Optional<bool> userConnected;
2517     Optional<MessageEventID> eventId;
2518
2519     virtual void print(QTextStream & strm) const override;
2520
2521     bool operator==(const Identity & other) const
2522     {
2523         return (id == other.id)

```

```

2616         && contact.isEqual(other.contact)
2617         && userId.isEqual(other.userId)
2618         && deactivated.isEqual(other.deactivated)
2619         && sameBusiness.isEqual(other.sameBusiness)
2620         && blocked.isEqual(other.blocked)
2621         && userConnected.isEqual(other.userConnected)
2622         && eventId.isEqual(other.eventId)
2623     };
2624 }
2625
2626 bool operator!=(const Identity & other) const
2627 {
2628     return !(*this == other);
2629 }
2630
2631 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
2632 Q_PROPERTY(IdentityID id MEMBER id)
2633 Q_PROPERTY(Optional<Contact> contact MEMBER contact)
2634 Q_PROPERTY(Optional<UserID> userId MEMBER userId)
2635 Q_PROPERTY(Optional<bool> deactivated MEMBER deactivated)
2636 Q_PROPERTY(Optional<bool> sameBusiness MEMBER sameBusiness)
2637 Q_PROPERTY(Optional<bool> blocked MEMBER blocked)
2638 Q_PROPERTY(Optional<bool> userConnected MEMBER userConnected)
2639 Q_PROPERTY(Optional<MessageEventID> eventId MEMBER eventId)
2640 };
2641
2642 struct QEVERCLOUD_EXPORT Tag: public Printable
2643 {
2644 private:
2645     Q_GADGET
2646 public:
2647     EverCloudLocalData localData;
2648
2649     Optional<Guid> guid;
2650     Optional<QString> name;
2651     Optional<Guid> parentGuid;
2652     Optional<qint32> updateSequenceNum;
2653
2654     virtual void print(QTextStream & strm) const override;
2655
2656 bool operator==(const Tag & other) const
2657 {
2658     return guid.isEqual(other.guid)
2659         && name.isEqual(other.name)
2660         && parentGuid.isEqual(other.parentGuid)
2661         && updateSequenceNum.isEqual(other.updateSequenceNum)
2662     ;
2663 }
2664
2665 bool operator!=(const Tag & other) const
2666 {
2667     return !(*this == other);
2668 }
2669
2670 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
2671 Q_PROPERTY(Optional<Guid> guid MEMBER guid)
2672 Q_PROPERTY(Optional<QString> name MEMBER name)
2673 Q_PROPERTY(Optional<Guid> parentGuid MEMBER parentGuid)
2674 Q_PROPERTY(Optional<qint32> updateSequenceNum MEMBER updateSequenceNum)
2675 };
2676
2677 struct QEVERCLOUD_EXPORT LazyMap: public Printable
2678 {
2679 private:
2680     Q_GADGET
2681 public:
2682     EverCloudLocalData localData;
2683
2684     Optional<QSet<QString>> keysOnly;
2685     Optional<QMap<QString, QString>> fullMap;
2686
2687     virtual void print(QTextStream & strm) const override;
2688
2689 bool operator==(const LazyMap & other) const
2690 {
2691     return keysOnly.isEqual(other.keysOnly)
2692         && fullMap.isEqual(other.fullMap)
2693     ;
2694 }
2695
2696 bool operator!=(const LazyMap & other) const
2697 {
2698     return !(*this == other);
2699 }
2700
2701 using FullMap = QMap<QString, QString>;
2702

```

```

2776     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
2777     Q_PROPERTY(Optional<QSet<QString> keysOnly MEMBER keysOnly)
2778     Q_PROPERTY(Optional<FullMap> fullMap MEMBER fullMap)
2779 };
2780
2784 struct QEVERCLOUD_EXPORT ResourceAttributes: public Printable
2785 {
2786 private:
2787     Q_GADGET
2788 public:
2792     EverCloudLocalData localData;
2793
2799     Optional<QString> sourceURL;
2804     Optional<Timestamp> timestamp;
2808     Optional<double> latitude;
2812     Optional<double> longitude;
2816     Optional<double> altitude;
2823     Optional<QString> cameraMake;
2830     Optional<QString> cameraModel;
2836     Optional<bool> clientWillIndex;
2841     Optional<QString> recoType;
2847     Optional<QString> fileName;
2852     Optional<bool> attachment;
2869     Optional<LazyMap> applicationData;
2870
2871     virtual void print(QTextStream & strm) const override;
2872
2873     bool operator==(const ResourceAttributes & other) const
2874     {
2875         return sourceURL.isEqual(other.sourceURL)
2876             && timestamp.isEqual(other.timestamp)
2877             && latitude.isEqual(other.latitude)
2878             && longitude.isEqual(other.longitude)
2879             && altitude.isEqual(other.altitude)
2880             && cameraMake.isEqual(other.cameraMake)
2881             && cameraModel.isEqual(other.cameraModel)
2882             && clientWillIndex.isEqual(other.clientWillIndex)
2883             && recoType.isEqual(other.recoType)
2884             && fileName.isEqual(other.fileName)
2885             && attachment.isEqual(other.attachment)
2886             && applicationData.isEqual(other.applicationData)
2887         ;
2888     }
2889
2890     bool operator!=(const ResourceAttributes & other) const
2891     {
2892         return !(*this == other);
2893     }
2894
2895     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
2896     Q_PROPERTY(Optional<QString> sourceURL MEMBER sourceURL)
2897     Q_PROPERTY(Optional<Timestamp> timestamp MEMBER timestamp)
2898     Q_PROPERTY(Optional<double> latitude MEMBER latitude)
2899     Q_PROPERTY(Optional<double> longitude MEMBER longitude)
2900     Q_PROPERTY(Optional<double> altitude MEMBER altitude)
2901     Q_PROPERTY(Optional<QString> cameraMake MEMBER cameraMake)
2902     Q_PROPERTY(Optional<QString> cameraModel MEMBER cameraModel)
2903     Q_PROPERTY(Optional<bool> clientWillIndex MEMBER clientWillIndex)
2904     Q_PROPERTY(Optional<QString> recoType MEMBER recoType)
2905     Q_PROPERTY(Optional<QString> fileName MEMBER fileName)
2906     Q_PROPERTY(Optional<bool> attachment MEMBER attachment)
2907     Q_PROPERTY(Optional<LazyMap> applicationData MEMBER applicationData)
2908 };
2909
2914 struct QEVERCLOUD_EXPORT Resource: public Printable
2915 {
2916 private:
2917     Q_GADGET
2918 public:
2922     EverCloudLocalData localData;
2923
2933     Optional<Guid> guid;
2943     Optional<Guid> noteGuid;
2949     Optional<Data> data;
2957     Optional<QString> mime;
2962     Optional<qint16> width;
2967     Optional<qint16> height;
2971     Optional<qint16> duration;
2975     Optional<bool> active;
2980     Optional<Data> recognition;
2984     Optional<ResourceAttributes> attributes;
2991     Optional<qint32> updateSequenceNum;
2999     Optional<Data> alternateData;
3000
3001     virtual void print(QTextStream & strm) const override;
3002
3003     bool operator==(const Resource & other) const

```

```

3004 {
3005     return guid.isEqual(other.guid)
3006         && noteGuid.isEqual(other.noteGuid)
3007         && data.isEqual(other.data)
3008         && mime.isEqual(other.mime)
3009         && width.isEqual(other.width)
3010         && height.isEqual(other.height)
3011         && duration.isEqual(other.duration)
3012         && active.isEqual(other.active)
3013         && recognition.isEqual(other.recognition)
3014         && attributes.isEqual(other.attributes)
3015         && updateSequenceNum.isEqual(other.updateSequenceNum)
3016         && alternateData.isEqual(other.alternateData)
3017     ;
3018 }
3019
3020 bool operator!=(const Resource & other) const
3021 {
3022     return !(*this == other);
3023 }
3024
3025 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
3026 Q_PROPERTY(Optional<Guid> guid MEMBER guid)
3027 Q_PROPERTY(Optional<Guid> noteGuid MEMBER noteGuid)
3028 Q_PROPERTY(Optional<Data> data MEMBER data)
3029 Q_PROPERTY(Optional<QString> mime MEMBER mime)
3030 Q_PROPERTY(Optional<qint16> width MEMBER width)
3031 Q_PROPERTY(Optional<qint16> height MEMBER height)
3032 Q_PROPERTY(Optional<qint16> duration MEMBER duration)
3033 Q_PROPERTY(Optional<bool> active MEMBER active)
3034 Q_PROPERTY(Optional<Data> recognition MEMBER recognition)
3035 Q_PROPERTY(Optional<ResourceAttributes> attributes MEMBER attributes)
3036 Q_PROPERTY(Optional<qint32> updateSequenceNum MEMBER updateSequenceNum)
3037 Q_PROPERTY(Optional<Data> alternateData MEMBER alternateData)
3038 };
3039
3040 struct QEVERCLOUD_EXPORT NoteAttributes: public Printable
3041 {
3042 private:
3043     Q_GADGET
3044 public:
3045     EverCloudLocalData localData;
3046
3047     Optional<Timestamp> subjectDate;
3048     Optional<double> latitude;
3049     Optional<double> longitude;
3050     Optional<double> altitude;
3051     Optional<QString> author;
3052     Optional<QString> source;
3053     Optional<QString> sourceURL;
3054     Optional<QString> sourceApplication;
3055     Optional<Timestamp> shareDate;
3056     Optional<qint64> reminderOrder;
3057     Optional<Timestamp> reminderDoneTime;
3058     Optional<Timestamp> reminderTime;
3059     Optional<QString> placeName;
3060     Optional<QString> contentClass;
3061     Optional<LazyMap> applicationData;
3062     Optional<QString> lastEditedBy;
3063     Optional<QMap<QString, QString>> classifications;
3064     Optional<UserID> creatorId;
3065     Optional<UserID> lastEditorId;
3066     Optional<bool> sharedWithBusiness;
3067     Optional<Guid> conflictSourceNoteGuid;
3068     Optional<qint32> noteTitleQuality;
3069
3070     virtual void print(QTextStream & strm) const override;
3071
3072     bool operator==(const NoteAttributes & other) const
3073     {
3074         return subjectDate.isEqual(other.subjectDate)
3075             && latitude.isEqual(other.latitude)
3076             && longitude.isEqual(other.longitude)
3077             && altitude.isEqual(other.altitude)
3078             && author.isEqual(other.author)
3079             && source.isEqual(other.source)
3080             && sourceURL.isEqual(other.sourceURL)
3081             && sourceApplication.isEqual(other.sourceApplication)
3082             && shareDate.isEqual(other.shareDate)
3083             && reminderOrder.isEqual(other.reminderOrder)
3084             && reminderDoneTime.isEqual(other.reminderDoneTime)
3085             && reminderTime.isEqual(other.reminderTime)
3086             && placeName.isEqual(other.placeName)
3087             && contentClass.isEqual(other.contentClass)
3088             && applicationData.isEqual(other.applicationData)
3089             && lastEditedBy.isEqual(other.lastEditedBy)
3090             && classifications.isEqual(other.classifications)

```

```

3281         && creatorId.isEqual(other.creatorId)
3282         && lastEditorId.isEqual(other.lastEditorId)
3283         && sharedWithBusiness.isEqual(other.sharedWithBusiness)
3284         && conflictSourceNoteGuid.isEqual(other.conflictSourceNoteGuid)
3285         && noteTitleQuality.isEqual(other.noteTitleQuality)
3286     };
3287 }
3288
3289 bool operator!=(const NoteAttributes & other) const
3290 {
3291     return !(*this == other);
3292 }
3293
3294 using Classifications = QMap<QString, QString>;
3295
3296 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
3297 Q_PROPERTY(Optional<Timestamp> subjectDate MEMBER subjectDate)
3298 Q_PROPERTY(Optional<double> latitude MEMBER latitude)
3299 Q_PROPERTY(Optional<double> longitude MEMBER longitude)
3300 Q_PROPERTY(Optional<double> altitude MEMBER altitude)
3301 Q_PROPERTY(Optional<QString> author MEMBER author)
3302 Q_PROPERTY(Optional<QString> source MEMBER source)
3303 Q_PROPERTY(Optional<QString> sourceURL MEMBER sourceURL)
3304 Q_PROPERTY(Optional<QString> sourceApplication MEMBER sourceApplication)
3305 Q_PROPERTY(Optional<Timestamp> shareDate MEMBER shareDate)
3306 Q_PROPERTY(Optional<qint64> reminderOrder MEMBER reminderOrder)
3307 Q_PROPERTY(Optional<Timestamp> reminderDoneTime MEMBER reminderDoneTime)
3308 Q_PROPERTY(Optional<Timestamp> reminderTime MEMBER reminderTime)
3309 Q_PROPERTY(Optional<QString> placeName MEMBER placeName)
3310 Q_PROPERTY(Optional<QString> contentClass MEMBER contentClass)
3311 Q_PROPERTY(Optional<LazyMap> applicationData MEMBER applicationData)
3312 Q_PROPERTY(Optional<QString> lastEditedBy MEMBER lastEditedBy)
3313 Q_PROPERTY(Optional<Classifications> classifications MEMBER classifications)
3314 Q_PROPERTY(Optional<UserID> creatorId MEMBER creatorId)
3315 Q_PROPERTY(Optional<UserID> lastEditorId MEMBER lastEditorId)
3316 Q_PROPERTY(Optional<bool> sharedWithBusiness MEMBER sharedWithBusiness)
3317 Q_PROPERTY(Optional<Guid> conflictSourceNoteGuid MEMBER conflictSourceNoteGuid)
3318 Q_PROPERTY(Optional<qint32> noteTitleQuality MEMBER noteTitleQuality)
3319 };
3320
3321 struct QEVERCLOUD_EXPORT SharedNote: public Printable
3322 {
3323 private:
3324     Q_GADGET
3325 public:
3326     EverCloudLocalData localData;
3327
3328     Optional<UserID> sharerUserID;
3329     Optional<Identity> recipientIdentity;
3330     Optional<SharedNotePrivilegeLevel> privilege;
3331     Optional<Timestamp> serviceCreated;
3332     Optional<Timestamp> serviceUpdated;
3333     Optional<Timestamp> serviceAssigned;
3334
3335     virtual void print(QTextStream & strm) const override;
3336
3337     bool operator==(const SharedNote & other) const
3338     {
3339         return sharerUserID.isEqual(other.sharerUserID)
3340             && recipientIdentity.isEqual(other.recipientIdentity)
3341             && privilege.isEqual(other.privilege)
3342             && serviceCreated.isEqual(other.serviceCreated)
3343             && serviceUpdated.isEqual(other.serviceUpdated)
3344             && serviceAssigned.isEqual(other.serviceAssigned)
3345     };
3346 }
3347
3348 bool operator!=(const SharedNote & other) const
3349 {
3350     return !(*this == other);
3351 }
3352
3353 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
3354 Q_PROPERTY(Optional<UserID> sharerUserID MEMBER sharerUserID)
3355 Q_PROPERTY(Optional<Identity> recipientIdentity MEMBER recipientIdentity)
3356 Q_PROPERTY(Optional<SharedNotePrivilegeLevel> privilege MEMBER privilege)
3357 Q_PROPERTY(Optional<Timestamp> serviceCreated MEMBER serviceCreated)
3358 Q_PROPERTY(Optional<Timestamp> serviceUpdated MEMBER serviceUpdated)
3359 Q_PROPERTY(Optional<Timestamp> serviceAssigned MEMBER serviceAssigned)
3360 };
3361
3362 struct QEVERCLOUD_EXPORT NoteRestrictions: public Printable
3363 {
3364 private:
3365     Q_GADGET
3366 public:
3367     EverCloudLocalData localData;

```



```

3438
3442     Optional<bool> noUpdateTitle;
3444     Optional<bool> noUpdateContent;
3448     Optional<bool> noEmail;
3453     Optional<bool> noShare;
3457     Optional<bool> noSharePublicly;
3458
3459     virtual void print(QTextStream & strm) const override;
3460
3461     bool operator==(const NoteRestrictions & other) const
3462     {
3463         return noUpdateTitle.isEqual(other.noUpdateTitle)
3464             && noUpdateContent.isEqual(other.noUpdateContent)
3465             && noEmail.isEqual(other.noEmail)
3466             && noShare.isEqual(other.noShare)
3467             && noSharePublicly.isEqual(other.noSharePublicly)
3468         ;
3469     }
3470
3471     bool operator!=(const NoteRestrictions & other) const
3472     {
3473         return !(*this == other);
3474     }
3475
3476     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
3477     Q_PROPERTY(Optional<bool> noUpdateTitle MEMBER noUpdateTitle)
3478     Q_PROPERTY(Optional<bool> noUpdateContent MEMBER noUpdateContent)
3479     Q_PROPERTY(Optional<bool> noEmail MEMBER noEmail)
3480     Q_PROPERTY(Optional<bool> noShare MEMBER noShare)
3481     Q_PROPERTY(Optional<bool> noSharePublicly MEMBER noSharePublicly)
3482 };
3483
3492 struct QEVERCLOUD_EXPORT NoteLimits: public Printable
3493 {
3494 private:
3495     Q_GADGET
3496 public:
3500     EverCloudLocalData localData;
3501
3503     Optional<qint32> noteResourceCountMax;
3505     Optional<qint64> uploadLimit;
3507     Optional<qint64> resourceSizeMax;
3509     Optional<qint64> noteSizeMax;
3511     Optional<qint64> uploaded;
3512
3513     virtual void print(QTextStream & strm) const override;
3514
3515     bool operator==(const NoteLimits & other) const
3516     {
3517         return noteResourceCountMax.isEqual(other.noteResourceCountMax)
3518             && uploadLimit.isEqual(other.uploadLimit)
3519             && resourceSizeMax.isEqual(other.resourceSizeMax)
3520             && noteSizeMax.isEqual(other.noteSizeMax)
3521             && uploaded.isEqual(other.uploaded)
3522         ;
3523     }
3524
3525     bool operator!=(const NoteLimits & other) const
3526     {
3527         return !(*this == other);
3528     }
3529
3530     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
3531     Q_PROPERTY(Optional<qint32> noteResourceCountMax MEMBER noteResourceCountMax)
3532     Q_PROPERTY(Optional<qint64> uploadLimit MEMBER uploadLimit)
3533     Q_PROPERTY(Optional<qint64> resourceSizeMax MEMBER resourceSizeMax)
3534     Q_PROPERTY(Optional<qint64> noteSizeMax MEMBER noteSizeMax)
3535     Q_PROPERTY(Optional<qint64> uploaded MEMBER uploaded)
3536 };
3537
3542 struct QEVERCLOUD_EXPORT Note: public Printable
3543 {
3544 private:
3545     Q_GADGET
3546 public:
3550     EverCloudLocalData localData;
3551
3560     Optional<Guid> guid;
3568     Optional<QString> title;
3579     Optional<QString> content;
3587     Optional<QByteArray> contentHash;
3593     Optional<qint32> contentLength;
3603     Optional<Timestamp> created;
3610     Optional<Timestamp> updated;
3618     Optional<Timestamp> deleted;
3623     Optional<bool> active;
3630     Optional<qint32> updateSequenceNum;

```

```

3640     Optional<QString> notebookGuid;
3651     Optional<QList<Guid>> tagGuids;
3661     Optional<QList<Resource>> resources;
3667     Optional<NoteAttributes> attributes;
3675     Optional<QStringList> tagNames;
3682     Optional<QList<SharedNote>> sharedNotes;
3691     Optional<NoteRestrictions> restrictions;
3693     Optional<NoteLimits> limits;
3694
3695     virtual void print(QTextStream & strm) const override;
3696
3697     bool operator==(const Note & other) const
3698     {
3699         return guid.isEqual(other.guid)
3700             && title.isEqual(other.title)
3701             && content.isEqual(other.content)
3702             && contentHash.isEqual(other.contentHash)
3703             && contentLength.isEqual(other.contentLength)
3704             && created.isEqual(other.created)
3705             && updated.isEqual(other.updated)
3706             && deleted.isEqual(other.deleted)
3707             && active.isEqual(other.active)
3708             && updateSequenceNum.isEqual(other.updateSequenceNum)
3709             && notebookGuid.isEqual(other.notebookGuid)
3710             && tagGuids.isEqual(other.tagGuids)
3711             && resources.isEqual(other.resources)
3712             && attributes.isEqual(other.attributes)
3713             && tagNames.isEqual(other.tagNames)
3714             && sharedNotes.isEqual(other.sharedNotes)
3715             && restrictions.isEqual(other.restrictions)
3716             && limits.isEqual(other.limits)
3717         ;
3718     }
3719
3720     bool operator!=(const Note & other) const
3721     {
3722         return !(*this == other);
3723     }
3724
3725     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
3726     Q_PROPERTY(Optional<Guid> guid MEMBER guid)
3727     Q_PROPERTY(Optional<QString> title MEMBER title)
3728     Q_PROPERTY(Optional<QString> content MEMBER content)
3729     Q_PROPERTY(Optional<QByteArray> contentHash MEMBER contentHash)
3730     Q_PROPERTY(Optional<qint32> contentLength MEMBER contentLength)
3731     Q_PROPERTY(Optional<Timestamp> created MEMBER created)
3732     Q_PROPERTY(Optional<Timestamp> updated MEMBER updated)
3733     Q_PROPERTY(Optional<Timestamp> deleted MEMBER deleted)
3734     Q_PROPERTY(Optional<bool> active MEMBER active)
3735     Q_PROPERTY(Optional<qint32> updateSequenceNum MEMBER updateSequenceNum)
3736     Q_PROPERTY(Optional<QString> notebookGuid MEMBER notebookGuid)
3737     Q_PROPERTY(Optional<QList<Guid>> tagGuids MEMBER tagGuids)
3738     Q_PROPERTY(Optional<QList<Resource>> resources MEMBER resources)
3739     Q_PROPERTY(Optional<NoteAttributes> attributes MEMBER attributes)
3740     Q_PROPERTY(Optional<QStringList> tagNames MEMBER tagNames)
3741     Q_PROPERTY(Optional<QList<SharedNote>> sharedNotes MEMBER sharedNotes)
3742     Q_PROPERTY(Optional<NoteRestrictions> restrictions MEMBER restrictions)
3743     Q_PROPERTY(Optional<NoteLimits> limits MEMBER limits)
3744 };
3745
3751 struct QEVERCLOUD_EXPORT Publishing: public Printable
3752 {
3753 private:
3754     Q_GADGET
3755 public:
3759     EverCloudLocalData localData;
3760
3772     Optional<QString> uri;
3777     Optional<NoteSortOrder> order;
3783     Optional<bool> ascending;
3794     Optional<QString> publicDescription;
3795
3796     virtual void print(QTextStream & strm) const override;
3797
3798     bool operator==(const Publishing & other) const
3799     {
3800         return uri.isEqual(other.uri)
3801             && order.isEqual(other.order)
3802             && ascending.isEqual(other.ascending)
3803             && publicDescription.isEqual(other.publicDescription)
3804         ;
3805     }
3806
3807     bool operator!=(const Publishing & other) const
3808     {
3809         return !(*this == other);
3810     }

```

```

3811
3812     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
3813     Q_PROPERTY(Optional<QString> uri MEMBER uri)
3814     Q_PROPERTY(Optional<NoteSortOrder> order MEMBER order)
3815     Q_PROPERTY(Optional<bool> ascending MEMBER ascending)
3816     Q_PROPERTY(Optional<QString> publicDescription MEMBER publicDescription)
3817 };
3818
3826 struct QEVERCLOUD_EXPORT BusinessNotebook: public Printable
3827 {
3828 private:
3829     Q_GADGET
3830 public:
3831     EverCloudLocalData localData;
3832
3833     Optional<QString> notebookDescription;
3834     Optional<SharedNotebookPrivilegeLevel> privilege;
3835     Optional<bool> recommended;
3836
3837     virtual void print(QTextStream & strm) const override;
3838
3839     bool operator==(const BusinessNotebook & other) const
3840     {
3841         return notebookDescription.isEqual(other.notebookDescription)
3842             && privilege.isEqual(other.privilege)
3843             && recommended.isEqual(other.recommended)
3844         ;
3845     }
3846
3847     bool operator!=(const BusinessNotebook & other) const
3848     {
3849         return !(*this == other);
3850     }
3851
3852     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
3853     Q_PROPERTY(Optional<QString> notebookDescription MEMBER notebookDescription)
3854     Q_PROPERTY(Optional<SharedNotebookPrivilegeLevel> privilege MEMBER privilege)
3855     Q_PROPERTY(Optional<bool> recommended MEMBER recommended)
3856 };
3857
3883 struct QEVERCLOUD_EXPORT SavedSearchScope: public Printable
3884 {
3885 private:
3886     Q_GADGET
3887 public:
3888     EverCloudLocalData localData;
3889
3890     Optional<bool> includeAccount;
3891     Optional<bool> includePersonalLinkedNotebooks;
3892     Optional<bool> includeBusinessLinkedNotebooks;
3893
3894     virtual void print(QTextStream & strm) const override;
3895
3896     bool operator==(const SavedSearchScope & other) const
3897     {
3898         return includeAccount.isEqual(other.includeAccount)
3899             && includePersonalLinkedNotebooks.isEqual(other.includePersonalLinkedNotebooks)
3900             && includeBusinessLinkedNotebooks.isEqual(other.includeBusinessLinkedNotebooks)
3901         ;
3902     }
3903
3904     bool operator!=(const SavedSearchScope & other) const
3905     {
3906         return !(*this == other);
3907     }
3908
3909     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
3910     Q_PROPERTY(Optional<bool> includeAccount MEMBER includeAccount)
3911     Q_PROPERTY(Optional<bool> includePersonalLinkedNotebooks MEMBER includePersonalLinkedNotebooks)
3912     Q_PROPERTY(Optional<bool> includeBusinessLinkedNotebooks MEMBER includeBusinessLinkedNotebooks)
3913 };
3914
3933 struct QEVERCLOUD_EXPORT SavedSearch: public Printable
3934 {
3935 private:
3936     Q_GADGET
3937 public:
3938     EverCloudLocalData localData;
3939
3940     Optional<Guid> guid;
3941     Optional<QString> name;
3942     Optional<QString> query;
3943     Optional<QueryFormat> format;
3944     Optional<qint32> updateSequenceNum;
3945     Optional<SavedSearchScope> scope;
3946
3947     virtual void print(QTextStream & strm) const override;

```

```

3996
3997     bool operator==(const SavedSearch & other) const
3998 {
3999     return guid.isEqual(other.guid)
4000         && name.isEqual(other.name)
4001         && query.isEqual(other.query)
4002         && format.isEqual(other.format)
4003         && updateSequenceNum.isEqual(other.updateSequenceNum)
4004         && scope.isEqual(other.scope)
4005     ;
4006 }
4007
4008     bool operator!=(const SavedSearch & other) const
4009 {
4010     return !(*this == other);
4011 }
4012
4013     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
4014     Q_PROPERTY(Optional<Guid> guid MEMBER guid)
4015     Q_PROPERTY(Optional<QString> name MEMBER name)
4016     Q_PROPERTY(Optional<QString> query MEMBER query)
4017     Q_PROPERTY(Optional<QueryFormat> format MEMBER format)
4018     Q_PROPERTY(Optional<qint32> updateSequenceNum MEMBER updateSequenceNum)
4019     Q_PROPERTY(Optional<SavedSearchScope> scope MEMBER scope)
4020 };
4021
4036 struct QEVERCLOUD_EXPORT SharedNotebookRecipientSettings: public Printable
4037 {
4038 private:
4039     Q_GADGET
4040 public:
4041     EverCloudLocalData localData;
4042
4043     Optional<bool> reminderNotifyEmail;
4044     Optional<bool> reminderNotifyInApp;
4045
4046     virtual void print(QTextStream & strm) const override;
4047
4048     bool operator==(const SharedNotebookRecipientSettings & other) const
4049 {
4050     return reminderNotifyEmail.isEqual(other.reminderNotifyEmail)
4051         && reminderNotifyInApp.isEqual(other.reminderNotifyInApp)
4052     ;
4053 }
4054
4055     bool operator!=(const SharedNotebookRecipientSettings & other) const
4056 {
4057     return !(*this == other);
4058 }
4059
4060     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
4061     Q_PROPERTY(Optional<bool> reminderNotifyEmail MEMBER reminderNotifyEmail)
4062     Q_PROPERTY(Optional<bool> reminderNotifyInApp MEMBER reminderNotifyInApp)
4063 };
4064
4090 struct QEVERCLOUD_EXPORT NotebookRecipientSettings: public Printable
4091 {
4092 private:
4093     Q_GADGET
4094 public:
4095     EverCloudLocalData localData;
4096
4097     Optional<bool> reminderNotifyEmail;
4098     Optional<bool> reminderNotifyInApp;
4099     Optional<bool> inMyList;
4100     Optional<QString> stack;
4101     Optional<RecipientStatus> recipientStatus;
4102
4103     virtual void print(QTextStream & strm) const override;
4104
4105     bool operator==(const NotebookRecipientSettings & other) const
4106 {
4107     return reminderNotifyEmail.isEqual(other.reminderNotifyEmail)
4108         && reminderNotifyInApp.isEqual(other.reminderNotifyInApp)
4109         && inMyList.isEqual(other.inMyList)
4110         && stack.isEqual(other.stack)
4111         && recipientStatus.isEqual(other.recipientStatus)
4112     ;
4113 }
4114
4115     bool operator!=(const NotebookRecipientSettings & other) const
4116 {
4117     return !(*this == other);
4118 }
4119
4120     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
4121     Q_PROPERTY(Optional<bool> reminderNotifyEmail MEMBER reminderNotifyEmail)

```

```

4153     Q_PROPERTY(Optional<bool> reminderNotifyInApp MEMBER reminderNotifyInApp)
4154     Q_PROPERTY(Optional<bool> inMyList MEMBER inMyList)
4155     Q_PROPERTY(Optional<QString> stack MEMBER stack)
4156     Q_PROPERTY(Optional<RecipientStatus> recipientStatus MEMBER recipientStatus)
4157 };
4158
4163 struct QEVERCLOUD_EXPORT SharedNotebook: public Printable
4164 {
4165 private:
4166     Q_GADGET
4167 public:
4171     EverCloudLocalData localData;
4172
4176     Optional<qint64> id;
4180     Optional<UserID> userId;
4184     Optional<Guid> notebookGuid;
4191     Optional<QString> email;
4196     Optional<IdentityID> recipientIdentityId;
4200     Optional<bool> notebookModifiable;
4205     Optional<Timestamp> serviceCreated;
4214     Optional<Timestamp> serviceUpdated;
4222     Optional<QString> globalId;
4228     Optional<QString> username;
4233     Optional<SharedNotebookPrivilegeLevel> privilege;
4240     Optional<SharedNotebookRecipientSettings> recipientSettings;
4249     Optional<UserID> sharerUserId;
4257     Optional<QString> recipientUsername;
4266     Optional<UserID> recipientUserId;
4272     Optional<Timestamp> serviceAssigned;
4273
4274     virtual void print(QTextStream & strm) const override;
4275
4276     bool operator==(const SharedNotebook & other) const
4277 {
4278         return id.isEqual(other.id)
4279             && userId.isEqual(other.userId)
4280             && notebookGuid.isEqual(other.notebookGuid)
4281             && email.isEqual(other.email)
4282             && recipientIdentityId.isEqual(other.recipientIdentityId)
4283             && notebookModifiable.isEqual(other.notebookModifiable)
4284             && serviceCreated.isEqual(other.serviceCreated)
4285             && serviceUpdated.isEqual(other.serviceUpdated)
4286             && globalId.isEqual(other.globalId)
4287             && username.isEqual(other.username)
4288             && privilege.isEqual(other.privilege)
4289             && recipientSettings.isEqual(other.recipientSettings)
4290             && sharerUserId.isEqual(other.sharerUserId)
4291             && recipientUsername.isEqual(other.recipientUsername)
4292             && recipientUserId.isEqual(other.recipientUserId)
4293             && serviceAssigned.isEqual(other.serviceAssigned)
4294         ;
4295     }
4296
4297     bool operator!=(const SharedNotebook & other) const
4298 {
4299         return !(*this == other);
4300     }
4301
4302     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
4303     Q_PROPERTY(Optional<qint64> id MEMBER id)
4304     Q_PROPERTY(Optional<UserID> userId MEMBER userId)
4305     Q_PROPERTY(Optional<Guid> notebookGuid MEMBER notebookGuid)
4306     Q_PROPERTY(Optional<QString> email MEMBER email)
4307     Q_PROPERTY(Optional<IdentityID> recipientIdentityId MEMBER recipientIdentityId)
4308     Q_PROPERTY(Optional<bool> notebookModifiable MEMBER notebookModifiable)
4309     Q_PROPERTY(Optional<Timestamp> serviceCreated MEMBER serviceCreated)
4310     Q_PROPERTY(Optional<Timestamp> serviceUpdated MEMBER serviceUpdated)
4311     Q_PROPERTY(Optional<QString> globalId MEMBER globalId)
4312     Q_PROPERTY(Optional<QString> username MEMBER username)
4313     Q_PROPERTY(Optional<SharedNotebookPrivilegeLevel> privilege MEMBER privilege)
4314     Q_PROPERTY(Optional<SharedNotebookRecipientSettings> recipientSettings MEMBER recipientSettings)
4315     Q_PROPERTY(Optional<UserID> sharerUserId MEMBER sharerUserId)
4316     Q_PROPERTY(Optional<QString> recipientUsername MEMBER recipientUsername)
4317     Q_PROPERTY(Optional<UserID> recipientUserId MEMBER recipientUserId)
4318     Q_PROPERTY(Optional<Timestamp> serviceAssigned MEMBER serviceAssigned)
4319 };
4320
4324 struct QEVERCLOUD_EXPORT CanMoveToContainerRestrictions: public Printable
4325 {
4326 private:
4327     Q_GADGET
4328 public:
4332     EverCloudLocalData localData;
4333
4335     Optional<CanMoveToContainerStatus> canMoveToContainer;
4336
4337     virtual void print(QTextStream & strm) const override;

```

```

4338
4339     bool operator==(const CanMoveToContainerRestrictions & other) const
4340 {
4341     return canMoveToContainer.isEqual(other.canMoveToContainer)
4342     ;
4343 }
4344
4345     bool operator!=(const CanMoveToContainerRestrictions & other) const
4346 {
4347     return !(*this == other);
4348 }
4349
4350     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
4351     Q_PROPERTY(Optional<CanMoveToContainerStatus> canMoveToContainer MEMBER canMoveToContainer)
4352 };
4353
4380 struct QEVERCLOUD_EXPORT NotebookRestrictions: public Printable
4381 {
4382 private:
4383     Q_GADGET
4384 public:
4385     EverCloudLocalData localData;
4386
4394     Optional<bool> noReadNotes;
4398     Optional<bool> noCreateNotes;
4402     Optional<bool> noUpdateNotes;
4406     Optional<bool> noExpungeNotes;
4411     Optional<bool> noShareNotes;
4417     Optional<bool> noEmailNotes;
4422     Optional<bool> noSendMessageToRecipients;
4427     Optional<bool> noUpdateNotebook;
4432     Optional<bool> noExpungeNotebook;
4437     Optional<bool> noSetDefaultNotebook;
4442     Optional<bool> noSetNotebookStack;
4447     Optional<bool> noPublishToPublic;
4451     Optional<bool> noPublishToBusinessLibrary;
4456     Optional<bool> noCreateTags;
4460     Optional<bool> noUpdateTags;
4464     Optional<bool> noExpungeTags;
4469     Optional<bool> noSetParentTag;
4473     Optional<bool> noCreateSharedNotebooks;
4480     Optional<SharedNotebookInstanceRestrictions> updateWhichSharedNotebookRestrictions;
4487     Optional<SharedNotebookInstanceRestrictions> expungeWhichSharedNotebookRestrictions;
4492     Optional<bool> noShareNotesWithBusiness;
4496     Optional<bool> noRenameNotebook;
4501     Optional<bool> noSetInMyList;
4503     Optional<bool> noChangeContact;
4508     Optional<CanMoveToContainerRestrictions> canMoveToContainerRestrictions;
4510     Optional<bool> noSetReminderNotifyEmail;
4512     Optional<bool> noSetReminderNotifyInApp;
4514     Optional<bool> noSetRecipientSettingsStack;
4518     Optional<bool> noCanMoveNote;
4519
4520     virtual void print(QTextStream & strm) const override;
4521
4522     bool operator==(const NotebookRestrictions & other) const
4523 {
4524     return noReadNotes.isEqual(other.noReadNotes)
4525         && noCreateNotes.isEqual(other.noCreateNotes)
4526         && noUpdateNotes.isEqual(other.noUpdateNotes)
4527         && noExpungeNotes.isEqual(other.noExpungeNotes)
4528         && noShareNotes.isEqual(other.noShareNotes)
4529         && noEmailNotes.isEqual(other.noEmailNotes)
4530         && noSendMessageToRecipients.isEqual(other.noSendMessageToRecipients)
4531         && noUpdateNotebook.isEqual(other.noUpdateNotebook)
4532         && noExpungeNotebook.isEqual(other.noExpungeNotebook)
4533         && noSetDefaultNotebook.isEqual(other.noSetDefaultNotebook)
4534         && noSetNotebookStack.isEqual(other.noSetNotebookStack)
4535         && noPublishToPublic.isEqual(other.noPublishToPublic)
4536         && noPublishToBusinessLibrary.isEqual(other.noPublishToBusinessLibrary)
4537         && noCreateTags.isEqual(other.noCreateTags)
4538         && noUpdateTags.isEqual(other.noUpdateTags)
4539         && noExpungeTags.isEqual(other.noExpungeTags)
4540         && noSetParentTag.isEqual(other.noSetParentTag)
4541         && noCreateSharedNotebooks.isEqual(other.noCreateSharedNotebooks)
4542         &&
4543         updateWhichSharedNotebookRestrictions.isEqual(other.updateWhichSharedNotebookRestrictions)
4544         &&
4545         expungeWhichSharedNotebookRestrictions.isEqual(other.expungeWhichSharedNotebookRestrictions)
4546         && noShareNotesWithBusiness.isEqual(other.noShareNotesWithBusiness)
4547         && noRenameNotebook.isEqual(other.noRenameNotebook)
4548         && noSetInMyList.isEqual(other.noSetInMyList)
4549         && noChangeContact.isEqual(other.noChangeContact)
4550         && canMoveToContainerRestrictions.isEqual(other.canMoveToContainerRestrictions)
4551         && noSetReminderNotifyEmail.isEqual(other.noSetReminderNotifyEmail)
4552         && noSetReminderNotifyInApp.isEqual(other.noSetReminderNotifyInApp)
4553         && noSetRecipientSettingsStack.isEqual(other.noSetRecipientSettingsStack)

```

```

4552         && noCanMoveNote.isEqual(other.noCanMoveNote)
4553     ;
4554 }
4555
4556 bool operator!=(const NotebookRestrictions & other) const
4557 {
4558     return !(*this == other);
4559 }
4560
4561 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
4562 Q_PROPERTY(Optional<bool> noReadNotes MEMBER noReadNotes)
4563 Q_PROPERTY(Optional<bool> noCreateNotes MEMBER noCreateNotes)
4564 Q_PROPERTY(Optional<bool> noUpdateNotes MEMBER noUpdateNotes)
4565 Q_PROPERTY(Optional<bool> noExpungeNotes MEMBER noExpungeNotes)
4566 Q_PROPERTY(Optional<bool> noShareNotes MEMBER noShareNotes)
4567 Q_PROPERTY(Optional<bool> noEmailNotes MEMBER noEmailNotes)
4568 Q_PROPERTY(Optional<bool> noSendMessageToRecipients MEMBER noSendMessageToRecipients)
4569 Q_PROPERTY(Optional<bool> noUpdateNotebook MEMBER noUpdateNotebook)
4570 Q_PROPERTY(Optional<bool> noExpungeNotebook MEMBER noExpungeNotebook)
4571 Q_PROPERTY(Optional<bool> noSetDefaultNotebook MEMBER noSetDefaultNotebook)
4572 Q_PROPERTY(Optional<bool> noSetNotebookStack MEMBER noSetNotebookStack)
4573 Q_PROPERTY(Optional<bool> noPublishToPublic MEMBER noPublishToPublic)
4574 Q_PROPERTY(Optional<bool> noPublishToBusinessLibrary MEMBER noPublishToBusinessLibrary)
4575 Q_PROPERTY(Optional<bool> noCreateTags MEMBER noCreateTags)
4576 Q_PROPERTY(Optional<bool> noUpdateTags MEMBER noUpdateTags)
4577 Q_PROPERTY(Optional<bool> noExpungeTags MEMBER noExpungeTags)
4578 Q_PROPERTY(Optional<bool> noSetParentTag MEMBER noSetParentTag)
4579 Q_PROPERTY(Optional<bool> noCreateSharedNotebooks MEMBER noCreateSharedNotebooks)
4580 Q_PROPERTY(Optional<SharedNotebookInstanceRestrictions> updateWhichSharedNotebookRestrictions
MEMBER updateWhichSharedNotebookRestrictions)
4581 Q_PROPERTY(Optional<SharedNotebookInstanceRestrictions> expungeWhichSharedNotebookRestrictions
MEMBER expungeWhichSharedNotebookRestrictions)
4582 Q_PROPERTY(Optional<bool> noShareNotesWithBusiness MEMBER noShareNotesWithBusiness)
4583 Q_PROPERTY(Optional<bool> noRenameNotebook MEMBER noRenameNotebook)
4584 Q_PROPERTY(Optional<bool> noSetInMyList MEMBER noSetInMyList)
4585 Q_PROPERTY(Optional<bool> noChangeContact MEMBER noChangeContact)
4586 Q_PROPERTY(Optional<CanMoveToContainerRestrictions> canMoveToContainerRestrictions MEMBER
canMoveToContainerRestrictions)
4587 Q_PROPERTY(Optional<bool> noSetReminderNotifyEmail MEMBER noSetReminderNotifyEmail)
4588 Q_PROPERTY(Optional<bool> noSetReminderNotifyInApp MEMBER noSetReminderNotifyInApp)
4589 Q_PROPERTY(Optional<bool> noSetRecipientSettingsStack MEMBER noSetRecipientSettingsStack)
4590 Q_PROPERTY(Optional<bool> noCanMoveNote MEMBER noCanMoveNote)
4591 };
4592
4596 struct QEVERCLOUD_EXPORT Notebook: public Printable
4597 {
4598 private:
4599     Q_GADGET
4600 public:
4601     EverCloudLocalData localData;
4602
4603     Optional<Guid> guid;
4604     Optional<QString> name;
4605     Optional<qint32> updateSequenceNum;
4606     Optional<bool> defaultNotebook;
4607     Optional<Timestamp> serviceCreated;
4608     Optional<Timestamp> serviceUpdated;
4609     Optional<Publishing> publishing;
4610     Optional<bool> published;
4611     Optional<QString> stack;
4612     Optional<QList<qint64>> sharedNotebookIds;
4613     Optional<QList<SharedNotebook>> sharedNotebooks;
4614     Optional<BusinessNotebook> businessNotebook;
4615     Optional<User> contact;
4616     Optional<NotebookRestrictions> restrictions;
4617     Optional<NotebookRecipientSettings> recipientSettings;
4618
4619     virtual void print(QTextStream & strm) const override;
4620
4621     bool operator==(const Notebook & other) const
4622     {
4623         return guid.isEqual(other.guid)
4624             && name.isEqual(other.name)
4625             && updateSequenceNum.isEqual(other.updateSequenceNum)
4626             && defaultNotebook.isEqual(other.defaultNotebook)
4627             && serviceCreated.isEqual(other.serviceCreated)
4628             && serviceUpdated.isEqual(other.serviceUpdated)
4629             && publishing.isEqual(other.publishing)
4630             && published.isEqual(other.published)
4631             && stack.isEqual(other.stack)
4632             && sharedNotebookIds.isEqual(other.sharedNotebookIds)
4633             && sharedNotebooks.isEqual(other.sharedNotebooks)
4634             && businessNotebook.isEqual(other.businessNotebook)
4635             && contact.isEqual(other.contact)
4636             && restrictions.isEqual(other.restrictions)
4637             && recipientSettings.isEqual(other.recipientSettings)
4638     };

```

```

4746     }
4747
4748     bool operator!=(const Notebook & other) const
4749 {
4750     return !(*this == other);
4751 }
4752
4753 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
4754 Q_PROPERTY(Optional<Guid> guid MEMBER guid)
4755 Q_PROPERTY(Optional<QString> name MEMBER name)
4756 Q_PROPERTY(Optional<qint32> updateSequenceNum MEMBER updateSequenceNum)
4757 Q_PROPERTY(Optional<bool> defaultNotebook MEMBER defaultNotebook)
4758 Q_PROPERTY(Optional<Timestamp> serviceCreated MEMBER serviceCreated)
4759 Q_PROPERTY(Optional<Timestamp> serviceUpdated MEMBER serviceUpdated)
4760 Q_PROPERTY(Optional<Publishing> publishing MEMBER publishing)
4761 Q_PROPERTY(Optional<bool> published MEMBER published)
4762 Q_PROPERTY(Optional<QString> stack MEMBER stack)
4763 Q_PROPERTY(Optional<QList<qint64> sharedNotebookIds MEMBER sharedNotebookIds)
4764 Q_PROPERTY(Optional<QList<SharedNotebook> sharedNotebooks MEMBER sharedNotebooks)
4765 Q_PROPERTY(Optional<BusinessNotebook> businessNotebook MEMBER businessNotebook)
4766 Q_PROPERTY(Optional<User> contact MEMBER contact)
4767 Q_PROPERTY(Optional<NotebookRestrictions> restrictions MEMBER restrictions)
4768 Q_PROPERTY(Optional<NotebookRecipientSettings> recipientSettings MEMBER recipientSettings)
4769 };
4770
4771 struct QEVERCLOUD_EXPORT LinkedNotebook: public Printable
4772 {
4773 private:
4774     Q_GADGET
4775 public:
4776     EverCloudLocalData localData;
4777
4778     Optional<QString> shareName;
4779     Optional<QString> username;
4780     Optional<QString> shardId;
4781     Optional<QString> sharedNotebookGlobalId;
4782     Optional<QString> uri;
4783     Optional<Guid> guid;
4784     Optional<qint32> updateSequenceNum;
4785     Optional<QString> noteStoreUrl;
4786     Optional<QString> webApiUrlPrefix;
4787     Optional<QString> stack;
4788     Optional<qint32> businessId;
4789
4790     virtual void print(QTextStream & strm) const override;
4791
4792     bool operator==(const LinkedNotebook & other) const
4793 {
4794     return shareName.isEqual(other.shareName)
4795         && username.isEqual(other.username)
4796         && shardId.isEqual(other.shardId)
4797         && sharedNotebookGlobalId.isEqual(other.sharedNotebookGlobalId)
4798         && uri.isEqual(other.uri)
4799         && guid.isEqual(other.guid)
4800         && updateSequenceNum.isEqual(other.updateSequenceNum)
4801         && noteStoreUrl.isEqual(other.noteStoreUrl)
4802         && webApiUrlPrefix.isEqual(other.webApiUrlPrefix)
4803         && stack.isEqual(other.stack)
4804         && businessId.isEqual(other.businessId)
4805     ;
4806 }
4807
4808     bool operator!=(const LinkedNotebook & other) const
4809 {
4810     return !(*this == other);
4811 }
4812
4813 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
4814 Q_PROPERTY(Optional<QString> shareName MEMBER shareName)
4815 Q_PROPERTY(Optional<QString> username MEMBER username)
4816 Q_PROPERTY(Optional<QString> shardId MEMBER shardId)
4817 Q_PROPERTY(Optional<QString> sharedNotebookGlobalId MEMBER sharedNotebookGlobalId)
4818 Q_PROPERTY(Optional<QString> uri MEMBER uri)
4819 Q_PROPERTY(Optional<Guid> guid MEMBER guid)
4820 Q_PROPERTY(Optional<qint32> updateSequenceNum MEMBER updateSequenceNum)
4821 Q_PROPERTY(Optional<QString> noteStoreUrl MEMBER noteStoreUrl)
4822 Q_PROPERTY(Optional<QString> webApiUrlPrefix MEMBER webApiUrlPrefix)
4823 Q_PROPERTY(Optional<QString> stack MEMBER stack)
4824 Q_PROPERTY(Optional<qint32> businessId MEMBER businessId)
4825 };
4826
4827 struct QEVERCLOUD_EXPORT NotebookDescriptor: public Printable
4828 {
4829 private:
4830     Q_GADGET
4831 public:
4832     EverCloudLocalData localData;

```



```

4911
4915     Optional<Guid> guid;
4920     Optional<QString> notebookDisplayName;
4924     Optional<QString> contactName;
4929     Optional<bool> hasSharedNotebook;
4933     Optional<qint32> joinedUserCount;
4934
4935     virtual void print(QTextStream & strm) const override;
4936
4937     bool operator==(const NotebookDescriptor & other) const
4938     {
4939         return guid.isEqual(other.guid)
4940             && notebookDisplayName.isEqual(other.notebookDisplayName)
4941             && contactName.isEqual(other.contactName)
4942             && hasSharedNotebook.isEqual(other.hasSharedNotebook)
4943             && joinedUserCount.isEqual(other.joinedUserCount)
4944         ;
4945     }
4946
4947     bool operator!=(const NotebookDescriptor & other) const
4948     {
4949         return !(*this == other);
4950     }
4951
4952     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
4953     Q_PROPERTY(Optional<Guid> guid MEMBER guid)
4954     Q_PROPERTY(Optional<QString> notebookDisplayName MEMBER notebookDisplayName)
4955     Q_PROPERTY(Optional<QString> contactName MEMBER contactName)
4956     Q_PROPERTY(Optional<bool> hasSharedNotebook MEMBER hasSharedNotebook)
4957     Q_PROPERTY(Optional<qint32> joinedUserCount MEMBER joinedUserCount)
4958 };
4959
4964 struct QEVERCLOUD_EXPORT UserProfile: public Printable
4965 {
4966 private:
4967     Q_GADGET
4968 public:
4972     EverCloudLocalData localData;
4973
4977     Optional<UserID> id;
4981     Optional<QString> name;
4986     Optional<QString> email;
4990     Optional<QString> username;
4994     Optional<BusinessUserAttributes> attributes;
4998     Optional<Timestamp> joined;
5002     Optional<Timestamp> photoLastUpdated;
5006     Optional<QString> photoUrl;
5010     Optional<BusinessUserRole> role;
5014     Optional<BusinessUserStatus> status;
5015
5016     virtual void print(QTextStream & strm) const override;
5017
5018     bool operator==(const UserProfile & other) const
5019     {
5020         return id.isEqual(other.id)
5021             && name.isEqual(other.name)
5022             && email.isEqual(other.email)
5023             && username.isEqual(other.username)
5024             && attributes.isEqual(other.attributes)
5025             && joined.isEqual(other.joined)
5026             && photoLastUpdated.isEqual(other.photoLastUpdated)
5027             && photoUrl.isEqual(other.photoUrl)
5028             && role.isEqual(other.role)
5029             && status.isEqual(other.status)
5030         ;
5031     }
5032
5033     bool operator!=(const UserProfile & other) const
5034     {
5035         return !(*this == other);
5036     }
5037
5038     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
5039     Q_PROPERTY(Optional<UserID> id MEMBER id)
5040     Q_PROPERTY(Optional<QString> name MEMBER name)
5041     Q_PROPERTY(Optional<QString> email MEMBER email)
5042     Q_PROPERTY(Optional<QString> username MEMBER username)
5043     Q_PROPERTY(Optional<BusinessUserAttributes> attributes MEMBER attributes)
5044     Q_PROPERTY(Optional<Timestamp> joined MEMBER joined)
5045     Q_PROPERTY(Optional<Timestamp> photoLastUpdated MEMBER photoLastUpdated)
5046     Q_PROPERTY(Optional<QString> photoUrl MEMBER photoUrl)
5047     Q_PROPERTY(Optional<BusinessUserRole> role MEMBER role)
5048     Q_PROPERTY(Optional<BusinessUserStatus> status MEMBER status)
5049 };
5050
5057 struct QEVERCLOUD_EXPORT RelatedContentImage: public Printable
5058 {

```

```

5059 private:
5060     Q_GADGET
5061 public:
5062     EverCloudLocalData localData;
5063
5064     Optional<QString> url;
5065     Optional<qint32> width;
5066     Optional<qint32> height;
5067     Optional<double> pixelRatio;
5068     Optional<qint32> fileSize;
5069
5070     virtual void print(QTextStream & strm) const override;
5071
5072     bool operator==(const RelatedContentImage & other) const
5073     {
5074         return url.isEqual(other.url)
5075             && width.isEqual(other.width)
5076             && height.isEqual(other.height)
5077             && pixelRatio.isEqual(other.pixelRatio)
5078             && fileSize.isEqual(other.fileSize)
5079         ;
5080     }
5081
5082     bool operator!=(const RelatedContentImage & other) const
5083     {
5084         return !(*this == other);
5085     }
5086
5087     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
5088     Q_PROPERTY(Optional<QString> url MEMBER url)
5089     Q_PROPERTY(Optional<qint32> width MEMBER width)
5090     Q_PROPERTY(Optional<qint32> height MEMBER height)
5091     Q_PROPERTY(Optional<double> pixelRatio MEMBER pixelRatio)
5092     Q_PROPERTY(Optional<qint32> fileSize MEMBER fileSize)
5093 };
5094
5095 struct QEVERCLOUD_EXPORT RelatedContent: public Printable
5096 {
5097 private:
5098     Q_GADGET
5099 public:
5100     EverCloudLocalData localData;
5101
5102     Optional<QString> contentId;
5103     Optional<QString> title;
5104     Optional<QString> url;
5105     Optional<QString> sourceId;
5106     Optional<QString> sourceUrl;
5107     Optional<QString> sourceFaviconUrl;
5108     Optional<QString> sourceName;
5109     Optional<Timestamp> date;
5110     Optional<QString> teaser;
5111     Optional<QList<RelatedContentImage>> thumbnails;
5112     Optional<RelatedContentType> contentType;
5113     Optional<RelatedContentAccess> accessType;
5114     Optional<QString> visibleUrl;
5115     Optional<QString> clipUrl;
5116     Optional<Contact> contact;
5117     Optional<QStringList> authors;
5118
5119     virtual void print(QTextStream & strm) const override;
5120
5121     bool operator==(const RelatedContent & other) const
5122     {
5123         return contentId.isEqual(other.contentId)
5124             && title.isEqual(other.title)
5125             && url.isEqual(other.url)
5126             && sourceId.isEqual(other.sourceId)
5127             && sourceUrl.isEqual(other.sourceUrl)
5128             && sourceFaviconUrl.isEqual(other.sourceFaviconUrl)
5129             && sourceName.isEqual(other.sourceName)
5130             && date.isEqual(other.date)
5131             && teaser.isEqual(other.teaser)
5132             && thumbnails.isEqual(other.thumbnails)
5133             && contentType.isEqual(other.contentType)
5134             && accessType.isEqual(other.accessType)
5135             && visibleUrl.isEqual(other.visibleUrl)
5136             && clipUrl.isEqual(other.clipUrl)
5137             && contact.isEqual(other.contact)
5138             && authors.isEqual(other.authors)
5139         ;
5140     }
5141
5142     bool operator!=(const RelatedContent & other) const
5143     {
5144         return !(*this == other);
5145     }
5146 }

```

```

5227
5228     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
5229     Q_PROPERTY(Optional<QString> contentId MEMBER contentId)
5230     Q_PROPERTY(Optional<QString> title MEMBER title)
5231     Q_PROPERTY(Optional<QString> url MEMBER url)
5232     Q_PROPERTY(Optional<QString> sourceId MEMBER sourceId)
5233     Q_PROPERTY(Optional<QString> sourceUrl MEMBER sourceUrl)
5234     Q_PROPERTY(Optional<QString> sourceFaviconUrl MEMBER sourceFaviconUrl)
5235     Q_PROPERTY(Optional<QString> sourceName MEMBER sourceName)
5236     Q_PROPERTY(Optional<Timestamp> date MEMBER date)
5237     Q_PROPERTY(Optional<QString> teaser MEMBER teaser)
5238     Q_PROPERTY(Optional<QList<RelatedContentImage> thumbnails MEMBER thumbnails)
5239     Q_PROPERTY(Optional<RelatedContentType> contentType MEMBER contentType)
5240     Q_PROPERTY(Optional<RelatedContentAccess> accessType MEMBER accessType)
5241     Q_PROPERTY(Optional<QString> visibleUrl MEMBER visibleUrl)
5242     Q_PROPERTY(Optional<QString> clipUrl MEMBER clipUrl)
5243     Q_PROPERTY(Optional<Contact> contact MEMBER contact)
5244     Q_PROPERTY(Optional<QStringList> authors MEMBER authors)
5245 };
5246
5251 struct QEVERCLOUD_EXPORT BusinessInvitation: public Printable
5252 {
5253 private:
5254     Q_GADGET
5255 public:
5256     EverCloudLocalData localData;
5257
5258     Optional<qint32> businessId;
5259     Optional<QString> email;
5260     Optional<BusinessUserRole> role;
5261     Optional<BusinessInvitationStatus> status;
5262     Optional<UserID> requesterId;
5263     Optional<bool> fromWorkChat;
5264     Optional<Timestamp> created;
5265     Optional<Timestamp> mostRecentReminder;
5266
5267     virtual void print(QTextStream & strm) const override;
5268
5269     bool operator==(const BusinessInvitation & other) const
5270 {
5271     return businessId.isEqual(other.businessId)
5272         && email.isEqual(other.email)
5273         && role.isEqual(other.role)
5274         && status.isEqual(other.status)
5275         && requesterId.isEqual(other.requesterId)
5276         && fromWorkChat.isEqual(other.fromWorkChat)
5277         && created.isEqual(other.created)
5278         && mostRecentReminder.isEqual(other.mostRecentReminder)
5279     ;
5280 }
5281
5282     bool operator!=(const BusinessInvitation & other) const
5283 {
5284     return !(*this == other);
5285 }
5286
5287     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
5288     Q_PROPERTY(Optional<qint32> businessId MEMBER businessId)
5289     Q_PROPERTY(Optional<QString> email MEMBER email)
5290     Q_PROPERTY(Optional<BusinessUserRole> role MEMBER role)
5291     Q_PROPERTY(Optional<BusinessInvitationStatus> status MEMBER status)
5292     Q_PROPERTY(Optional<UserID> requesterId MEMBER requesterId)
5293     Q_PROPERTY(Optional<bool> fromWorkChat MEMBER fromWorkChat)
5294     Q_PROPERTY(Optional<Timestamp> created MEMBER created)
5295     Q_PROPERTY(Optional<Timestamp> mostRecentReminder MEMBER mostRecentReminder)
5296 };
5297
5298 struct QEVERCLOUD_EXPORT UserIdentity: public Printable
5299 {
5300 private:
5301     Q_GADGET
5302 public:
5303     EverCloudLocalData localData;
5304
5305     Optional<UserIdentityType> type;
5306     Optional<QString> stringIdentifier;
5307     Optional<qint64> longIdentifier;
5308
5309     virtual void print(QTextStream & strm) const override;
5310
5311     bool operator==(const UserIdentity & other) const
5312 {
5313     return type.isEqual(other.type)
5314         && stringIdentifier.isEqual(other.stringIdentifier)
5315         && longIdentifier.isEqual(other.longIdentifier)
5316     ;
5317 }
5318

```

```

5383
5384     bool operator!=(const UserIdentity & other) const
5385 {
5386     return !(*this == other);
5387 }
5388
5389 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
5390 Q_PROPERTY(Optional<UserIdentityType> type MEMBER type)
5391 Q_PROPERTY(Optional<QString> stringIdentifier MEMBER stringIdentifier)
5392 Q_PROPERTY(Optional<qint64> longIdentifier MEMBER longIdentifier)
5393 };
5394
5399 struct QEVERCLOUD_EXPORT PublicUserInfo: public Printable
5400 {
5401 private:
5402     Q_GADGET
5403 public:
5404     EverCloudLocalData localData;
5405
5406     UserID userId = 0;
5407     Optional<ServiceLevel> serviceLevel;
5408     Optional<QString> username;
5409     Optional<QString> noteStoreUrl;
5410     Optional<QString> webApiUrlPrefix;
5411
5412     virtual void print(QTextStream & strm) const override;
5413
5414     bool operator==(const PublicUserInfo & other) const
5415 {
5416     return (userId == other.userId)
5417         && serviceLevel.isEqual(other.serviceLevel)
5418         && username.isEqual(other.username)
5419         && noteStoreUrl.isEqual(other.noteStoreUrl)
5420         && webApiUrlPrefix.isEqual(other.webApiUrlPrefix)
5421     ;
5422 }
5423
5424     bool operator!=(const PublicUserInfo & other) const
5425 {
5426     return !(*this == other);
5427 }
5428
5429 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
5430 Q_PROPERTY(UserID userId MEMBER userId)
5431 Q_PROPERTY(Optional<ServiceLevel> serviceLevel MEMBER serviceLevel)
5432 Q_PROPERTY(Optional<QString> username MEMBER username)
5433 Q_PROPERTY(Optional<QString> noteStoreUrl MEMBER noteStoreUrl)
5434 Q_PROPERTY(Optional<QString> webApiUrlPrefix MEMBER webApiUrlPrefix)
5435 };
5436
5441 struct QEVERCLOUD_EXPORT UserUrls: public Printable
5442 {
5443 private:
5444     Q_GADGET
5445 public:
5446     EverCloudLocalData localData;
5447
5448     Optional<QString> noteStoreUrl;
5449     Optional<QString> webApiUrlPrefix;
5450     Optional<QString> userStoreUrl;
5451     Optional<QString> utilityUrl;
5452     Optional<QString> messageStoreUrl;
5453     Optional<QString> userWebSocketUrl;
5454
5455     virtual void print(QTextStream & strm) const override;
5456
5457     bool operator==(const UserUrls & other) const
5458 {
5459     return noteStoreUrl.isEqual(other.noteStoreUrl)
5460         && webApiUrlPrefix.isEqual(other.webApiUrlPrefix)
5461         && userStoreUrl.isEqual(other.userStoreUrl)
5462         && utilityUrl.isEqual(other.utilityUrl)
5463         && messageStoreUrl.isEqual(other.messageStoreUrl)
5464         && userWebSocketUrl.isEqual(other.userWebSocketUrl)
5465     ;
5466 }
5467
5468     bool operator!=(const UserUrls & other) const
5469 {
5470     return !(*this == other);
5471 }
5472
5473 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
5474 Q_PROPERTY(Optional<QString> noteStoreUrl MEMBER noteStoreUrl)
5475 Q_PROPERTY(Optional<QString> webApiUrlPrefix MEMBER webApiUrlPrefix)
5476 Q_PROPERTY(Optional<QString> userStoreUrl MEMBER userStoreUrl)
5477 Q_PROPERTY(Optional<QString> utilityUrl MEMBER utilityUrl)

```

```

5541     Q_PROPERTY(Optional<QString> messageStoreUrl MEMBER messageStoreUrl)
5542     Q_PROPERTY(Optional<QString> userWebSocketUrl MEMBER userWebSocketUrl)
5543 };
5544
5545 struct QEVERCLOUD_EXPORT AuthenticationResult: public Printable
5546 {
5547     private:
5548         Q_GADGET
5549     public:
5550         EverCloudLocalData localData;
5551
5552         Timestamp currentTime = 0;
5553         QString authenticationToken;
5554         Timestamp expiration = 0;
5555         Optional<User> user;
5556         Optional<PublicUserInfo> publicUserInfo;
5557         Optional<QString> noteStoreUrl;
5558         Optional<QString> webApiUrlPrefix;
5559         Optional<bool> secondFactorRequired;
5560         Optional<QString> secondFactorDeliveryHint;
5561         Optional<UserUrls> urls;
5562
5563         virtual void print(QTextStream & strm) const override;
5564
5565         bool operator==(const AuthenticationResult & other) const
5566         {
5567             return (currentTime == other.currentTime)
5568                 && (authenticationToken == other.authenticationToken)
5569                 && (expiration == other.expiration)
5570                 && user.isEqual(other.user)
5571                 && publicUserInfo.isEqual(other.publicUserInfo)
5572                 && noteStoreUrl.isEqual(other.noteStoreUrl)
5573                 && webApiUrlPrefix.isEqual(other.webApiUrlPrefix)
5574                 && secondFactorRequired.isEqual(other.secondFactorRequired)
5575                 && secondFactorDeliveryHint.isEqual(other.secondFactorDeliveryHint)
5576                 && urls.isEqual(other.urls)
5577             ;
5578         }
5579
5580         bool operator!=(const AuthenticationResult & other) const
5581         {
5582             return !(*this == other);
5583         }
5584
5585         Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
5586         Q_PROPERTY(Timestamp currentTime MEMBER currentTime)
5587         Q_PROPERTY(QString authenticationToken MEMBER authenticationToken)
5588         Q_PROPERTY(Timestamp expiration MEMBER expiration)
5589         Q_PROPERTY(Optional<User> user MEMBER user)
5590         Q_PROPERTY(Optional<PublicUserInfo> publicUserInfo MEMBER publicUserInfo)
5591         Q_PROPERTY(Optional<QString> noteStoreUrl MEMBER noteStoreUrl)
5592         Q_PROPERTY(Optional<QString> webApiUrlPrefix MEMBER webApiUrlPrefix)
5593         Q_PROPERTY(Optional<bool> secondFactorRequired MEMBER secondFactorRequired)
5594         Q_PROPERTY(Optional<QString> secondFactorDeliveryHint MEMBER secondFactorDeliveryHint)
5595         Q_PROPERTY(Optional<UserUrls> urls MEMBER urls)
5596 };
5597
5598 struct QEVERCLOUD_EXPORT BootstrapSettings: public Printable
5599 {
5600     private:
5601         Q_GADGET
5602     public:
5603         EverCloudLocalData localData;
5604
5605         QString serviceHost;
5606         QString marketingUrl;
5607         QString supportUrl;
5608         QString accountEmailDomain;
5609         Optional<bool> enableFacebookSharing;
5610         Optional<bool> enableGiftSubscriptions;
5611         Optional<bool> enableSupportTickets;
5612         Optional<bool> enableSharedNotebooks;
5613         Optional<bool> enableSingleNoteSharing;
5614         Optional<bool> enableSponsoredAccounts;
5615         Optional<bool> enableTwitterSharing;
5616         Optional<bool> enableLinkedInSharing;
5617         Optional<bool> enablePublicNotebooks;
5618         Optional<bool> enableGoogle;
5619
5620         virtual void print(QTextStream & strm) const override;
5621
5622         bool operator==(const BootstrapSettings & other) const
5623         {
5624             return (serviceHost == other.serviceHost)
5625                 && (marketingUrl == other.marketingUrl)
5626                 && (supportUrl == other.supportUrl)
5627                 && (accountEmailDomain == other.accountEmailDomain)

```

```

5738         && enableFacebookSharing.isEqual(other.enableFacebookSharing)
5739         && enableGiftSubscriptions.isEqual(other.enableGiftSubscriptions)
5740         && enableSupportTickets.isEqual(other.enableSupportTickets)
5741         && enableSharedNotebooks.isEqual(other.enableSharedNotebooks)
5742         && enableSingleNoteSharing.isEqual(other.enableSingleNoteSharing)
5743         && enableSponsoredAccounts.isEqual(other.enableSponsoredAccounts)
5744         && enableTwitterSharing.isEqual(other.enableTwitterSharing)
5745         && enableLinkedInSharing.isEqual(other.enableLinkedInSharing)
5746         && enablePublicNotebooks.isEqual(other.enablePublicNotebooks)
5747         && enableGoogle.isEqual(other.enableGoogle)
5748     };
5749 }
5750
5751 bool operator!=(const BootstrapSettings & other) const
5752 {
5753     return !(*this == other);
5754 }
5755
5756 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
5757 Q_PROPERTY(QString serviceHost MEMBER serviceHost)
5758 Q_PROPERTY(QString marketingUrl MEMBER marketingUrl)
5759 Q_PROPERTY(QString supportUrl MEMBER supportUrl)
5760 Q_PROPERTY(QString accountEmailDomain MEMBER accountEmailDomain)
5761 Q_PROPERTY(Optional<bool> enableFacebookSharing MEMBER enableFacebookSharing)
5762 Q_PROPERTY(Optional<bool> enableGiftSubscriptions MEMBER enableGiftSubscriptions)
5763 Q_PROPERTY(Optional<bool> enableSupportTickets MEMBER enableSupportTickets)
5764 Q_PROPERTY(Optional<bool> enableSharedNotebooks MEMBER enableSharedNotebooks)
5765 Q_PROPERTY(Optional<bool> enableSingleNoteSharing MEMBER enableSingleNoteSharing)
5766 Q_PROPERTY(Optional<bool> enableSponsoredAccounts MEMBER enableSponsoredAccounts)
5767 Q_PROPERTY(Optional<bool> enableTwitterSharing MEMBER enableTwitterSharing)
5768 Q_PROPERTY(Optional<bool> enableLinkedInSharing MEMBER enableLinkedInSharing)
5769 Q_PROPERTY(Optional<bool> enablePublicNotebooks MEMBER enablePublicNotebooks)
5770 Q_PROPERTY(Optional<bool> enableGoogle MEMBER enableGoogle)
5771 };
5772
5776 struct QEVERCLOUD_EXPORT BootstrapProfile: public Printable
5777 {
5778 private:
5779     Q_GADGET
5780 public:
5781     EverCloudLocalData localData;
5782
5783     QString name;
5784     BootstrapSettings settings;
5785
5786     virtual void print(QTextStream & strm) const override;
5787
5788     bool operator==(const BootstrapProfile & other) const
5789     {
5790         return (name == other.name)
5791             && (settings == other.settings)
5792     };
5793
5794     bool operator!=(const BootstrapProfile & other) const
5795     {
5796         return !(*this == other);
5797     }
5798
5799     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
5800     Q_PROPERTY(QString name MEMBER name)
5801     Q_PROPERTY(BootstrapSettings settings MEMBER settings)
5802 };
5803
5808 struct QEVERCLOUD_EXPORT BootstrapInfo: public Printable
5809 {
5810 private:
5811     Q_GADGET
5812 public:
5813     EverCloudLocalData localData;
5814
5815     QList<BootstrapProfile> profiles;
5816
5817     virtual void print(QTextStream & strm) const override;
5818
5819     bool operator==(const BootstrapInfo & other) const
5820     {
5821         return (profiles == other.profiles)
5822     };
5823
5824     bool operator!=(const BootstrapInfo & other) const
5825     {
5826         return !(*this == other);
5827     }
5828
5829     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)

```

```

5848     Q_PROPERTY(QList<BootstrapProfile> profiles MEMBER profiles)
5849 };
5850
5851 class QEVERCLOUD_EXPORT EDAMUserException: public EvernoteException, public Printable
5852 {
5853     Q_GADGET
5854 public:
5855     EDAMErrorCode errorCode;
5856     Optional<QString> parameter;
5857
5858     EDAMUserException();
5859     virtual ~EDAMUserException() noexcept override;
5860
5861     EDAMUserException(const EDAMUserException & other);
5862     const char * what() const noexcept override;
5863     virtual EverCloudExceptionDataPtr exceptionData() const override;
5864
5865     virtual void print(QTextStream & strm) const override;
5866
5867     bool operator==(const EDAMUserException & other) const
5868     {
5869         return (errorCode == other.errorCode)
5870             && parameter.isEqual(other.parameter)
5871         ;
5872     }
5873
5874     bool operator!=(const EDAMUserException & other) const
5875     {
5876         return !(*this == other);
5877     }
5878
5879     Q_PROPERTY(EDAMErrorCode errorCode MEMBER errorCode)
5880     Q_PROPERTY(Optional<QString> parameter MEMBER parameter)
5881 };
5882
5883 class QEVERCLOUD_EXPORT EDAMSystemException: public EvernoteException, public Printable
5884 {
5885     Q_GADGET
5886 public:
5887     EDAMErrorCode errorCode;
5888     Optional<QString> message;
5889     Optional<qint32> rateLimitDuration;
5890
5891     EDAMSystemException();
5892     virtual ~EDAMSystemException() noexcept override;
5893
5894     EDAMSystemException(const EDAMSystemException & other);
5895     const char * what() const noexcept override;
5896     virtual EverCloudExceptionDataPtr exceptionData() const override;
5897
5898     virtual void print(QTextStream & strm) const override;
5899
5900     bool operator==(const EDAMSystemException & other) const
5901     {
5902         return (errorCode == other.errorCode)
5903             && message.isEqual(other.message)
5904             && rateLimitDuration.isEqual(other.rateLimitDuration)
5905         ;
5906     }
5907
5908     bool operator!=(const EDAMSystemException & other) const
5909     {
5910         return !(*this == other);
5911     }
5912
5913     Q_PROPERTY(EDAMErrorCode errorCode MEMBER errorCode)
5914     Q_PROPERTY(Optional<QString> message MEMBER message)
5915     Q_PROPERTY(Optional<qint32> rateLimitDuration MEMBER rateLimitDuration)
5916 };
5917
5918 class QEVERCLOUD_EXPORT EDAMNotFoundException: public EvernoteException, public Printable
5919 {
5920     Q_GADGET
5921 public:
5922     Optional<QString> identifier;
5923     Optional<QString> key;
5924
5925     EDAMNotFoundException();
5926     virtual ~EDAMNotFoundException() noexcept override;
5927
5928     EDAMNotFoundException(const EDAMNotFoundException & other);
5929     const char * what() const noexcept override;
5930     virtual EverCloudExceptionDataPtr exceptionData() const override;
5931
5932     virtual void print(QTextStream & strm) const override;
5933
5934     bool operator==(const EDAMNotFoundException & other) const

```

```

5980 {
5981     return identifier.isEqual(other.identifier)
5982         && key.isEqual(other.key)
5983     ;
5984 }
5985
5986 bool operator!=(const EDAMNotFoundException & other)const
5987 {
5988     return !(*this == other);
5989 }
5990
5991 Q_PROPERTY(Optional<QString> identifier MEMBER identifier)
5992 Q_PROPERTY(Optional<QString> key MEMBER key)
5993 };
5994
6017 class QEVERCLOUD_EXPORT EDAMInvalidContactsException: public EvernoteException, public Printable
6018 {
6019     Q_GADGET
6020 public:
6021     QList<Contact> contacts;
6022     Optional<QString> parameter;
6023     Optional<QList<EDAMInvalidContactReason>> reasons;
6024
6025     EDAMInvalidContactsException();
6026     virtual ~EDAMInvalidContactsException() noexcept override;
6027
6028     EDAMInvalidContactsException(const EDAMInvalidContactsException & other);
6029     const char * what() const noexcept override;
6030     virtual EverCloudExceptionDataPtr exceptionData() const override;
6031
6032     virtual void print(QTextStream & strm) const override;
6033
6034     bool operator==(const EDAMInvalidContactsException & other)const
6035     {
6036         return (contacts == other.contacts)
6037             && parameter.isEqual(other.parameter)
6038             && reasons.isEqual(other.reasons)
6039         ;
6040     }
6041
6042     bool operator!=(const EDAMInvalidContactsException & other)const
6043     {
6044         return !(*this == other);
6045     }
6046
6047     Q_PROPERTY(QList<Contact> contacts MEMBER contacts)
6048     Q_PROPERTY(Optional<QString> parameter MEMBER parameter)
6049     Q_PROPERTY(Optional<QList<EDAMInvalidContactReason>> reasons MEMBER reasons)
6050 };
6051
6063 struct QEVERCLOUD_EXPORT SyncChunk: public Printable
6064 {
6065 private:
6066     Q_GADGET
6067 public:
6071     EverCloudLocalData localData;
6072
6076     Timestamp currentTime = 0;
6082     Optional<qint32> chunkHighUSN;
6090     qint32 updateCount = 0;
6098     Optional<QList<Note>> notes;
6103     Optional<QList<Notebook>> notebooks;
6108     Optional<QList<Tag>> tags;
6113     Optional<QList<SavedSearch>> searches;
6120     Optional<QList<Resource>> resources;
6125     Optional<QList<Guid>> expungedNotes;
6132     Optional<QList<Guid>> expungedNotebooks;
6137     Optional<QList<Guid>> expungedTags;
6142     Optional<QList<Guid>> expungedSearches;
6147     Optional<QList<LinkedNotebook>> linkedNotebooks;
6152     Optional<QList<Guid>> expungedLinkedNotebooks;
6153
6154     virtual void print(QTextStream & strm) const override;
6155
6156     bool operator==(const SyncChunk & other)const
6157     {
6158         return (currentTime == other.currentTime)
6159             && chunkHighUSN.isEqual(other.chunkHighUSN)
6160             && (updateCount == other.updateCount)
6161             && notes.isEqual(other.notes)
6162             && notebooks.isEqual(other.notebooks)
6163             && tags.isEqual(other.tags)
6164             && searches.isEqual(other.searches)
6165             && resources.isEqual(other.resources)
6166             && expungedNotes.isEqual(other.expungedNotes)
6167             && expungedNotebooks.isEqual(other.expungedNotebooks)
6168             && expungedTags.isEqual(other.expungedTags)

```



```

6169         && expungedSearches.isEqual(other.expungedSearches)
6170         && linkedNotebooks.isEqual(other.linkedNotebooks)
6171         && expungedLinkedNotebooks.isEqual(other.expungedLinkedNotebooks)
6172     ;
6173 }
6174
6175 bool operator!=(const SyncChunk & other) const
6176 {
6177     return !(*this == other);
6178 }
6179
6180 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
6181 Q_PROPERTY(Timestamp currentTime MEMBER currentTime)
6182 Q_PROPERTY(Optional<qint32> chunkHighUSN MEMBER chunkHighUSN)
6183 Q_PROPERTY(qint32 updateCount MEMBER updateCount)
6184 Q_PROPERTY(Optional<QList<Note>> notes MEMBER notes)
6185 Q_PROPERTY(Optional<QList<Notebook>> notebooks MEMBER notebooks)
6186 Q_PROPERTY(Optional<QList<Tag>> tags MEMBER tags)
6187 Q_PROPERTY(Optional<QList<SavedSearch>> searches MEMBER searches)
6188 Q_PROPERTY(Optional<QList<Resource>> resources MEMBER resources)
6189 Q_PROPERTY(Optional<QList<Guid>> expungedNotes MEMBER expungedNotes)
6190 Q_PROPERTY(Optional<QList<Guid>> expungedNotebooks MEMBER expungedNotebooks)
6191 Q_PROPERTY(Optional<QList<Guid>> expungedTags MEMBER expungedTags)
6192 Q_PROPERTY(Optional<QList<Guid>> expungedSearches MEMBER expungedSearches)
6193 Q_PROPERTY(Optional<QList<LinkedNotebook>> linkedNotebooks MEMBER linkedNotebooks)
6194 Q_PROPERTY(Optional<QList<Guid>> expungedLinkedNotebooks MEMBER expungedLinkedNotebooks)
6195 };
6196
6201 struct QEVERCLOUD_EXPORT NoteList: public Printable
6202 {
6203 private:
6204     Q_GADGET
6205 public:
6206     EverCloudLocalData localData;
6207
6208     qint32 startIndex = 0;
6209     qint32 totalNotes = 0;
6210     QList<Note> notes;
6211     Optional<QStringList> stoppedWords;
6212     Optional<QStringList> searchedWords;
6213     Optional<qint32> updateCount;
6214     Optional<QByteArray> searchContextBytes;
6215     Optional<QString> debugInfo;
6216
6217     virtual void print(QTextStream & strm) const override;
6218
6219     bool operator==(const NoteList & other) const
6220     {
6221         return (startIndex == other.startIndex)
6222             && (totalNotes == other.totalNotes)
6223             && (notes == other.notes)
6224             && stoppedWords.isEqual(other.stoppedWords)
6225             && searchedWords.isEqual(other.searchedWords)
6226             && updateCount.isEqual(other.updateCount)
6227             && searchContextBytes.isEqual(other.searchContextBytes)
6228             && debugInfo.isEqual(other.debugInfo)
6229     ;
6230     }
6231
6232     bool operator!=(const NoteList & other) const
6233     {
6234         return !(*this == other);
6235     }
6236
6237     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
6238     Q_PROPERTY(qint32 startIndex MEMBER startIndex)
6239     Q_PROPERTY(qint32 totalNotes MEMBER totalNotes)
6240     Q_PROPERTY(QList<Note> notes MEMBER notes)
6241     Q_PROPERTY(Optional<QStringList> stoppedWords MEMBER stoppedWords)
6242     Q_PROPERTY(Optional<QStringList> searchedWords MEMBER searchedWords)
6243     Q_PROPERTY(Optional<qint32> updateCount MEMBER updateCount)
6244     Q_PROPERTY(Optional<QByteArray> searchContextBytes MEMBER searchContextBytes)
6245     Q_PROPERTY(Optional<QString> debugInfo MEMBER debugInfo)
6246 };
6247
6248 struct QEVERCLOUD_EXPORT NoteMetadata: public Printable
6249 {
6250 private:
6251     Q_GADGET
6252 public:
6253     EverCloudLocalData localData;
6254
6255     Guid guid;
6256     Optional<QString> title;
6257     Optional<qint32> contentLength;
6258     Optional<Timestamp> created;
6259     Optional<Timestamp> updated;

```

```

6322     Optional<Timestamp> deleted;
6323     Optional<qint32> updateSequenceNum;
6324     Optional<QString> notebookGuid;
6325     Optional<QList<Guid>> tagGuids;
6326     Optional<NoteAttributes> attributes;
6327     Optional<QString> largestResourceMime;
6328     Optional<qint32> largestResourceSize;
6329
6330     virtual void print(QTextStream & strm) const override;
6331
6332     bool operator==(const NoteMetadata & other) const
6333     {
6334         return (guid == other.guid)
6335             && title.isEqual(other.title)
6336             && contentLength.isEqual(other.contentLength)
6337             && created.isEqual(other.created)
6338             && updated.isEqual(other.updated)
6339             && deleted.isEqual(other.deleted)
6340             && updateSequenceNum.isEqual(other.updateSequenceNum)
6341             && notebookGuid.isEqual(other.notebookGuid)
6342             && tagGuids.isEqual(other.tagGuids)
6343             && attributes.isEqual(other.attributes)
6344             && largestResourceMime.isEqual(other.largestResourceMime)
6345             && largestResourceSize.isEqual(other.largestResourceSize)
6346         ;
6347     }
6348
6349     bool operator!=(const NoteMetadata & other) const
6350     {
6351         return !(*this == other);
6352     }
6353
6354     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
6355     Q_PROPERTY(Guid guid MEMBER guid)
6356     Q_PROPERTY(Optional<QString> title MEMBER title)
6357     Q_PROPERTY(Optional<qint32> contentLength MEMBER contentLength)
6358     Q_PROPERTY(Optional<Timestamp> created MEMBER created)
6359     Q_PROPERTY(Optional<Timestamp> updated MEMBER updated)
6360     Q_PROPERTY(Optional<Timestamp> deleted MEMBER deleted)
6361     Q_PROPERTY(Optional<qint32> updateSequenceNum MEMBER updateSequenceNum)
6362     Q_PROPERTY(Optional<QString> notebookGuid MEMBER notebookGuid)
6363     Q_PROPERTY(Optional<QList<Guid>> tagGuids MEMBER tagGuids)
6364     Q_PROPERTY(Optional<NoteAttributes> attributes MEMBER attributes)
6365     Q_PROPERTY(Optional<QString> largestResourceMime MEMBER largestResourceMime)
6366     Q_PROPERTY(Optional<qint32> largestResourceSize MEMBER largestResourceSize)
6367 };
6368
6369 struct QEVERCLOUD_EXPORT NotesMetadataList: public Printable
6370 {
6371 private:
6372     Q_GADGET
6373 public:
6374     EverCloudLocalData localData;
6375
6376     qint32 startIndex = 0;
6377     qint32 totalNotes = 0;
6378     QList<NoteMetadata> notes;
6379     Optional<QStringList> stoppedWords;
6380     Optional<QStringList> searchedWords;
6381     Optional<qint32> updateCount;
6382     Optional<QByteArray> searchContextBytes;
6383     Optional<QString> debugInfo;
6384
6385     virtual void print(QTextStream & strm) const override;
6386
6387     bool operator==(const NotesMetadataList & other) const
6388     {
6389         return (startIndex == other.startIndex)
6390             && (totalNotes == other.totalNotes)
6391             && (notes == other.notes)
6392             && stoppedWords.isEqual(other.stoppedWords)
6393             && searchedWords.isEqual(other.searchedWords)
6394             && updateCount.isEqual(other.updateCount)
6395             && searchContextBytes.isEqual(other.searchContextBytes)
6396             && debugInfo.isEqual(other.debugInfo)
6397         ;
6398     }
6399
6400     bool operator!=(const NotesMetadataList & other) const
6401     {
6402         return !(*this == other);
6403     }
6404
6405     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
6406     Q_PROPERTY(qint32 startIndex MEMBER startIndex)
6407     Q_PROPERTY(qint32 totalNotes MEMBER totalNotes)
6408     Q_PROPERTY(QList<NoteMetadata> notes MEMBER notes)

```

```

6474     Q_PROPERTY(Optional<QStringList> stoppedWords MEMBER stoppedWords)
6475     Q_PROPERTY(Optional<QStringList> searchedWords MEMBER searchedWords)
6476     Q_PROPERTY(Optional<qint32> updateCount MEMBER updateCount)
6477     Q_PROPERTY(Optional<QByteArray> searchContextBytes MEMBER searchContextBytes)
6478     Q_PROPERTY(Optional<QString> debugInfo MEMBER debugInfo)
6479 };
6480
6481 struct QEVERCLOUD_EXPORT NoteEmailParameters: public Printable
6482 {
6483 private:
6484     Q_GADGET
6485 public:
6486     EverCloudLocalData localData;
6487
6488     Optional<QString> guid;
6489     Optional<Note> note;
6490     Optional<QStringList> toAddresses;
6491     Optional<QStringList> ccAddresses;
6492     Optional<QString> subject;
6493     Optional<QString> message;
6494
6495     virtual void print(QTextStream & strm) const override;
6496
6497     bool operator==(const NoteEmailParameters & other) const
6498 {
6499     return guid.isEqual(other.guid)
6500         && note.isEqual(other.note)
6501         && toAddresses.isEqual(other.toAddresses)
6502         && ccAddresses.isEqual(other.ccAddresses)
6503         && subject.isEqual(other.subject)
6504         && message.isEqual(other.message)
6505     ;
6506 }
6507
6508     bool operator!=(const NoteEmailParameters & other) const
6509 {
6510     return !(*this == other);
6511 }
6512
6513     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
6514     Q_PROPERTY(Optional<QString> guid MEMBER guid)
6515     Q_PROPERTY(Optional<Note> note MEMBER note)
6516     Q_PROPERTY(Optional<QStringList> toAddresses MEMBER toAddresses)
6517     Q_PROPERTY(Optional<QStringList> ccAddresses MEMBER ccAddresses)
6518     Q_PROPERTY(Optional<QString> subject MEMBER subject)
6519     Q_PROPERTY(Optional<QString> message MEMBER message)
6520 };
6521
6522 struct QEVERCLOUD_EXPORT RelatedResult: public Printable
6523 {
6524 private:
6525     Q_GADGET
6526 public:
6527     EverCloudLocalData localData;
6528
6529     Optional<QList<Note>> notes;
6530     Optional<QList<Notebook>> notebooks;
6531     Optional<QList<Tag>> tags;
6532     Optional<QList<NotebookDescriptor>> containingNotebooks;
6533     Optional<QString> debugInfo;
6534     Optional<QList<UserProfile>> experts;
6535     Optional<QList<RelatedContent>> relatedContent;
6536     Optional<QString> cacheKey;
6537     Optional<qint32> cacheExpires;
6538
6539     virtual void print(QTextStream & strm) const override;
6540
6541     bool operator==(const RelatedResult & other) const
6542 {
6543     return notes.isEqual(other.notes)
6544         && notebooks.isEqual(other.notebooks)
6545         && tags.isEqual(other.tags)
6546         && containingNotebooks.isEqual(other.containingNotebooks)
6547         && debugInfo.isEqual(other.debugInfo)
6548         && experts.isEqual(other.experts)
6549         && relatedContent.isEqual(other.relatedContent)
6550         && cacheKey.isEqual(other.cacheKey)
6551         && cacheExpires.isEqual(other.cacheExpires)
6552     ;
6553 }
6554
6555     bool operator!=(const RelatedResult & other) const
6556 {
6557     return !(*this == other);
6558 }
6559
6560     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)

```

```

6679     Q_PROPERTY(Optional<QList<Note>> notes MEMBER notes)
6680     Q_PROPERTY(Optional<QList<Notebook>> notebooks MEMBER notebooks)
6681     Q_PROPERTY(Optional<QList<Tag>> tags MEMBER tags)
6682     Q_PROPERTY(Optional<QList<NotebookDescriptor>> containingNotebooks MEMBER containingNotebooks)
6683     Q_PROPERTY(Optional<QString> debugInfo MEMBER debugInfo)
6684     Q_PROPERTY(Optional<QList<UserProfile>> experts MEMBER experts)
6685     Q_PROPERTY(Optional<QList<RelatedContent>> relatedContent MEMBER relatedContent)
6686     Q_PROPERTY(Optional<QString> cacheKey MEMBER cacheKey)
6687     Q_PROPERTY(Optional<qint32> cacheExpires MEMBER cacheExpires)
6688 };
6689
6695 struct QEVERCLOUD_EXPORT UpdateNoteIfUsnMatchesResult: public Printable
6696 {
6697 private:
6698     Q_GADGET
6699 public:
6700     EverCloudLocalData localData;
6701
6702     Optional<Note> note;
6703     Optional<bool> updated;
6704
6705     virtual void print(QTextStream & strm) const override;
6706
6707     bool operator==(const UpdateNoteIfUsnMatchesResult & other) const
6708     {
6709         return note.isEqual(other.note)
6710             && updated.isEqual(other.updated);
6711     };
6712
6713     bool operator!=(const UpdateNoteIfUsnMatchesResult & other) const
6714     {
6715         return !(this == other);
6716     }
6717
6718     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
6719     Q_PROPERTY(Optional<Note> note MEMBER note)
6720     Q_PROPERTY(Optional<bool> updated MEMBER updated)
6721 };
6722
6723 struct QEVERCLOUD_EXPORT InvitationShareRelationship: public Printable
6724 {
6725 private:
6726     Q_GADGET
6727 public:
6728     EverCloudLocalData localData;
6729
6730     Optional<QString> displayName;
6731     Optional<UserIdentity> recipientUserIdentity;
6732     Optional<ShareRelationshipPrivilegeLevel> privilege;
6733     Optional<UserID> sharerUserId;
6734
6735     virtual void print(QTextStream & strm) const override;
6736
6737     bool operator==(const InvitationShareRelationship & other) const
6738     {
6739         return displayName.isEqual(other.displayName)
6740             && recipientUserIdentity.isEqual(other.recipientUserIdentity)
6741             && privilege.isEqual(other.privilege)
6742             && sharerUserId.isEqual(other.sharerUserId);
6743     };
6744
6745     bool operator!=(const InvitationShareRelationship & other) const
6746     {
6747         return !(this == other);
6748     }
6749
6750     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
6751     Q_PROPERTY(Optional<QString> displayName MEMBER displayName)
6752     Q_PROPERTY(Optional<UserIdentity> recipientUserIdentity MEMBER recipientUserIdentity)
6753     Q_PROPERTY(Optional<ShareRelationshipPrivilegeLevel> privilege MEMBER privilege)
6754     Q_PROPERTY(Optional<UserID> sharerUserId MEMBER sharerUserId)
6755 };
6756
6757 struct QEVERCLOUD_EXPORT ShareRelationships: public Printable
6758 {
6759 private:
6760     Q_GADGET
6761 public:
6762     EverCloudLocalData localData;
6763
6764     Optional<QList<InvitationShareRelationship>> invitations;
6765     Optional<QList<MemberShareRelationship>> memberships;
6766     Optional<ShareRelationshipRestrictions> invitationRestrictions;
6767
6768     virtual void print(QTextStream & strm) const override;

```

```

6844
6845     bool operator==(const ShareRelationships & other) const
6846 {
6847     return invitations.isEqual(other.invitations)
6848         && memberships.isEqual(other.memberships)
6849         && invitationRestrictions.isEqual(other.invitationRestrictions)
6850     ;
6851 }
6852
6853     bool operator!=(const ShareRelationships & other) const
6854 {
6855     return !(*this == other);
6856 }
6857
6858     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
6859     Q_PROPERTY(Optional<QList<InvitationShareRelationship>> invitations MEMBER invitations)
6860     Q_PROPERTY(Optional<QList<MemberShareRelationship>> memberships MEMBER memberships)
6861     Q_PROPERTY(Optional<ShareRelationshipRestrictions> invitationRestrictions MEMBER
invitationRestrictions)
6862 };
6863
6864 struct QEVERCLOUD_EXPORT ManageNotebookSharesParameters: public Printable
6865 {
6866 private:
6867     Q_GADGET
6868 public:
6869     EverCloudLocalData localData;
6870
6871     Optional<QString> notebookGuid;
6872     Optional<QString> inviteMessage;
6873     Optional<QList<MemberShareRelationship>> membershipsToUpdate;
6874     Optional<QList<InvitationShareRelationship>> invitationsToCreateOrUpdate;
6875     Optional<QList<UserIdentity>> unshares;
6876
6877     virtual void print(QTextStream & strm) const override;
6878
6879     bool operator==(const ManageNotebookSharesParameters & other) const
6880 {
6881     return notebookGuid.isEqual(other.notebookGuid)
6882         && inviteMessage.isEqual(other.inviteMessage)
6883         && membershipsToUpdate.isEqual(other.membershipsToUpdate)
6884         && invitationsToCreateOrUpdate.isEqual(other.invitationsToCreateOrUpdate)
6885         && unshares.isEqual(other.unshares)
6886     ;
6887 }
6888
6889     bool operator!=(const ManageNotebookSharesParameters & other) const
6890 {
6891     return !(*this == other);
6892 }
6893
6894     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
6895     Q_PROPERTY(Optional<QString> notebookGuid MEMBER notebookGuid)
6896     Q_PROPERTY(Optional<QString> inviteMessage MEMBER inviteMessage)
6897     Q_PROPERTY(Optional<QList<MemberShareRelationship>> membershipsToUpdate MEMBER membershipsToUpdate)
6898     Q_PROPERTY(Optional<QList<InvitationShareRelationship>> invitationsToCreateOrUpdate MEMBER
invitationsToCreateOrUpdate)
6899     Q_PROPERTY(Optional<QList<UserIdentity>> unshares MEMBER unshares)
6900 };
6901
6902 struct QEVERCLOUD_EXPORT ManageNotebookSharesError: public Printable
6903 {
6904 private:
6905     Q_GADGET
6906 public:
6907     EverCloudLocalData localData;
6908
6909     Optional<UserIdentity> userIdentity;
6910     Optional<EDAMUserException> userException;
6911     Optional<EDAMNotFoundException> notFoundException;
6912
6913     virtual void print(QTextStream & strm) const override;
6914
6915     bool operator==(const ManageNotebookSharesError & other) const
6916 {
6917     return userIdentity.isEqual(other.userIdentity)
6918         && userException.isEqual(other.userException)
6919         && notFoundException.isEqual(other.notFoundException)
6920     ;
6921 }
6922
6923     bool operator!=(const ManageNotebookSharesError & other) const
6924 {
6925     return !(*this == other);
6926 }
6927
6928     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)

```

```

7005     Q_PROPERTY(Optional<UserIdentity> userIdentity MEMBER userIdentity)
7006     Q_PROPERTY(Optional<EDAMUserException> userException MEMBER userException)
7007     Q_PROPERTY(Optional<EDAMNotFoundException> notFoundException MEMBER notFoundException)
7008 };
7009
7014 struct QEVERCLOUD_EXPORT ManageNotebookSharesResult: public Printable
7015 {
7016 private:
7017     Q_GADGET
7018 public:
7022     EverCloudLocalData localData;
7023
7029     Optional<QList<ManageNotebookSharesError>> errors;
7030
7031     virtual void print(QTextStream & strm) const override;
7032
7033     bool operator==(const ManageNotebookSharesResult & other) const
7034 {
7035     return errors.isEqual(other.errors)
7036     ;
7037 }
7038
7039     bool operator!=(const ManageNotebookSharesResult & other) const
7040 {
7041     return !(*this == other);
7042 }
7043
7044     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
7045     Q_PROPERTY(Optional<QList<ManageNotebookSharesError>> errors MEMBER errors)
7046 };
7047
7052 struct QEVERCLOUD_EXPORT SharedNoteTemplate: public Printable
7053 {
7054 private:
7055     Q_GADGET
7056 public:
7060     EverCloudLocalData localData;
7061
7065     Optional<Guid> noteGuid;
7073     Optional<MessageThreadId> recipientThreadId;
7080     Optional<QList<Contact>> recipientContacts;
7084     Optional<SharedNotePrivilegeLevel> privilege;
7085
7086     virtual void print(QTextStream & strm) const override;
7087
7088     bool operator==(const SharedNoteTemplate & other) const
7089 {
7090     return noteGuid.isEqual(other.noteGuid)
7091     && recipientThreadId.isEqual(other.recipientThreadId)
7092     && recipientContacts.isEqual(other.recipientContacts)
7093     && privilege.isEqual(other.privilege)
7094     ;
7095 }
7096
7097     bool operator!=(const SharedNoteTemplate & other) const
7098 {
7099     return !(*this == other);
7100 }
7101
7102     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
7103     Q_PROPERTY(Optional<Guid> noteGuid MEMBER noteGuid)
7104     Q_PROPERTY(Optional<MessageThreadId> recipientThreadId MEMBER recipientThreadId)
7105     Q_PROPERTY(Optional<QList<Contact>> recipientContacts MEMBER recipientContacts)
7106     Q_PROPERTY(Optional<SharedNotePrivilegeLevel> privilege MEMBER privilege)
7107 };
7108
7113 struct QEVERCLOUD_EXPORT NotebookShareTemplate: public Printable
7114 {
7115 private:
7116     Q_GADGET
7117 public:
7121     EverCloudLocalData localData;
7122
7126     Optional<Guid> notebookGuid;
7134     Optional<MessageThreadId> recipientThreadId;
7141     Optional<QList<Contact>> recipientContacts;
7145     Optional<SharedNotebookPrivilegeLevel> privilege;
7146
7147     virtual void print(QTextStream & strm) const override;
7148
7149     bool operator==(const NotebookShareTemplate & other) const
7150 {
7151     return notebookGuid.isEqual(other.notebookGuid)
7152     && recipientThreadId.isEqual(other.recipientThreadId)
7153     && recipientContacts.isEqual(other.recipientContacts)
7154     && privilege.isEqual(other.privilege)
7155     ;

```

```

7156     }
7157
7158     bool operator!=(const NotebookShareTemplate & other) const
7159 {
7160     return !(*this == other);
7161 }
7162
7163 Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
7164 Q_PROPERTY(Optional<Guid> notebookGuid MEMBER notebookGuid)
7165 Q_PROPERTY(Optional<MessageThreadId> recipientThreadId MEMBER recipientThreadId)
7166 Q_PROPERTY(Optional<QList<Contact> recipientContacts MEMBER recipientContacts)
7167 Q_PROPERTY(Optional<SharedNotebookPrivilegeLevel> privilege MEMBER privilege)
7168 };
7169
7174 struct QEVERCLOUD_EXPORT CreateOrUpdateNotebookSharesResult: public Printable
7175 {
7176 private:
7177     Q_GADGET
7178 public:
7179     EverCloudLocalData localData;
7180
7181     Optional<qint32> updateSequenceNum;
7182     Optional<QList<SharedNotebook>> matchingShares;
7183
7184     virtual void print(QTextStream & strm) const override;
7185
7186     bool operator==(const CreateOrUpdateNotebookSharesResult & other) const
7187 {
7188     return updateSequenceNum.isEqual(other.updateSequenceNum)
7189         && matchingShares.isEqual(other.matchingShares)
7190     ;
7191 }
7192
7193     bool operator!=(const CreateOrUpdateNotebookSharesResult & other) const
7194 {
7195     return !(*this == other);
7196 }
7197
7198     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
7199     Q_PROPERTY(Optional<qint32> updateSequenceNum MEMBER updateSequenceNum)
7200     Q_PROPERTY(Optional<QList<SharedNotebook> matchingShares MEMBER matchingShares)
7201 };
7202
7206 struct QEVERCLOUD_EXPORT ManageNoteSharesError: public Printable
7207 {
7208 private:
7209     Q_GADGET
7210 public:
7211     EverCloudLocalData localData;
7212
7213     Optional<IdentityID> identityID;
7214     Optional<UserID> userID;
7215     Optional<EDAMUserException> userException;
7216     Optional<EDAMNotFoundException> notFoundException;
7217
7218     virtual void print(QTextStream & strm) const override;
7219
7220     bool operator==(const ManageNoteSharesError & other) const
7221 {
7222     return identityID.isEqual(other.identityID)
7223         && userID.isEqual(other.userID)
7224         && userException.isEqual(other.userException)
7225         && notFoundException.isEqual(other.notFoundException)
7226     ;
7227 }
7228
7229     bool operator!=(const ManageNoteSharesError & other) const
7230 {
7231     return !(*this == other);
7232 }
7233
7234     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
7235     Q_PROPERTY(Optional<IdentityID> identityID MEMBER identityID)
7236     Q_PROPERTY(Optional<UserID> userID MEMBER userID)
7237     Q_PROPERTY(Optional<EDAMUserException> userException MEMBER userException)
7238     Q_PROPERTY(Optional<EDAMNotFoundException> notFoundException MEMBER notFoundException)
7239 };
7240
7245 struct QEVERCLOUD_EXPORT ManageNoteSharesResult: public Printable
7246 {
7247 private:
7248     Q_GADGET
7249 public:
7250     EverCloudLocalData localData;
7251
7252     Optional<QList<ManageNoteSharesError>> errors;
7253 }

```

```

7304     virtual void print(QTextStream & strm) const override;
7305
7306     bool operator==(const ManageNoteSharesResult & other) const
7307     {
7308         return errors.isEqual(other.errors)
7309         ;
7310     }
7311
7312     bool operator!=(const ManageNoteSharesResult & other) const
7313     {
7314         return !(*this == other);
7315     }
7316
7317     Q_PROPERTY(EverCloudLocalData localData MEMBER localData)
7318     Q_PROPERTY(Optional<QList<ManageNoteSharesError> errors MEMBER errors)
7319 };
7320
7321 } // namespace qevercloud
7322
7323 Q_DECLARE_METATYPE(qevercloud::EverCloudLocalData)
7324 Q_DECLARE_METATYPE(qevercloud::SyncState)
7325 Q_DECLARE_METATYPE(qevercloud::SyncChunkFilter)
7326 Q_DECLARE_METATYPE(qevercloud::NoteFilter)
7327 Q_DECLARE_METATYPE(qevercloud::NotesMetadataResultSpec)
7328 Q_DECLARE_METATYPE(qevercloud::NoteCollectionCounts)
7329 Q_DECLARE_METATYPE(qevercloud::NoteResultSpec)
7330 Q_DECLARE_METATYPE(qevercloud::NoteVersionId)
7331 Q_DECLARE_METATYPE(qevercloud::RelatedQuery)
7332 Q_DECLARE_METATYPE(qevercloud::RelatedResultSpec)
7333 Q_DECLARE_METATYPE(qevercloud::ShareRelationshipRestrictions)
7334 Q_DECLARE_METATYPE(qevercloud::MemberShareRelationship)
7335 Q_DECLARE_METATYPE(qevercloud::NoteShareRelationshipRestrictions)
7336 Q_DECLARE_METATYPE(qevercloud::NoteMemberShareRelationship)
7337 Q_DECLARE_METATYPE(qevercloud::NoteInvitationShareRelationship)
7338 Q_DECLARE_METATYPE(qevercloud::NoteShareRelationships)
7339 Q_DECLARE_METATYPE(qevercloud::ManageNoteSharesParameters)
7340 Q_DECLARE_METATYPE(qevercloud::Data)
7341 Q_DECLARE_METATYPE(qevercloud::UserAttributes)
7342 Q_DECLARE_METATYPE(qevercloud::BusinessUserAttributes)
7343 Q_DECLARE_METATYPE(qevercloud::Accounting)
7344 Q_DECLARE_METATYPE(qevercloud::BusinessUserInfo)
7345 Q_DECLARE_METATYPE(qevercloud::AccountLimits)
7346 Q_DECLARE_METATYPE(qevercloud::User)
7347 Q_DECLARE_METATYPE(qevercloud::Contact)
7348 Q_DECLARE_METATYPE(qevercloud::Identity)
7349 Q_DECLARE_METATYPE(qevercloud::Tag)
7350 Q_DECLARE_METATYPE(qevercloud::LazyMap)
7351 Q_DECLARE_METATYPE(qevercloud::ResourceAttributes)
7352 Q_DECLARE_METATYPE(qevercloud::Resource)
7353 Q_DECLARE_METATYPE(qevercloud::NoteAttributes)
7354 Q_DECLARE_METATYPE(qevercloud::SharedNote)
7355 Q_DECLARE_METATYPE(qevercloud::NoteRestrictions)
7356 Q_DECLARE_METATYPE(qevercloud::NoteLimits)
7357 Q_DECLARE_METATYPE(qevercloud::Note)
7358 Q_DECLARE_METATYPE(qevercloud::Publishing)
7359 Q_DECLARE_METATYPE(qevercloud::BusinessNotebook)
7360 Q_DECLARE_METATYPE(qevercloud::SavedSearchScope)
7361 Q_DECLARE_METATYPE(qevercloud::SavedSearch)
7362 Q_DECLARE_METATYPE(qevercloud::SharedNotebookRecipientSettings)
7363 Q_DECLARE_METATYPE(qevercloud::NotebookRecipientSettings)
7364 Q_DECLARE_METATYPE(qevercloud::SharedNotebook)
7365 Q_DECLARE_METATYPE(qevercloud::CanMoveToContainerRestrictions)
7366 Q_DECLARE_METATYPE(qevercloud::NotebookRestrictions)
7367 Q_DECLARE_METATYPE(qevercloud::Notebook)
7368 Q_DECLARE_METATYPE(qevercloud::LinkedNotebook)
7369 Q_DECLARE_METATYPE(qevercloud::NotebookDescriptor)
7370 Q_DECLARE_METATYPE(qevercloud::UserProfile)
7371 Q_DECLARE_METATYPE(qevercloud::RelatedContentImage)
7372 Q_DECLARE_METATYPE(qevercloud::RelatedContent)
7373 Q_DECLARE_METATYPE(qevercloud::BusinessInvitation)
7374 Q_DECLARE_METATYPE(qevercloud::UserIdentity)
7375 Q_DECLARE_METATYPE(qevercloud::PublicUserInfo)
7376 Q_DECLARE_METATYPE(qevercloud::UserUrls)
7377 Q_DECLARE_METATYPE(qevercloud::AuthenticationResult)
7378 Q_DECLARE_METATYPE(qevercloud::BootstrapSettings)
7379 Q_DECLARE_METATYPE(qevercloud::BootstrapProfile)
7380 Q_DECLARE_METATYPE(qevercloud::BootstrapInfo)
7381 Q_DECLARE_METATYPE(qevercloud::EDAMUserException)
7382 Q_DECLARE_METATYPE(qevercloud::EDAMSystemException)
7383 Q_DECLARE_METATYPE(qevercloud::EDAMNotFoundException)
7384 Q_DECLARE_METATYPE(qevercloud::EDAMInvalidContactsException)
7385 Q_DECLARE_METATYPE(qevercloud::SyncChunk)
7386 Q_DECLARE_METATYPE(qevercloud::NoteList)
7387 Q_DECLARE_METATYPE(qevercloud::NoteMetadata)
7388 Q_DECLARE_METATYPE(qevercloud::NotesMetadataList)
7389 Q_DECLARE_METATYPE(qevercloud::NoteEmailParameters)
7390 Q_DECLARE_METATYPE(qevercloud::RelatedResult)

```



```

7391 Q_DECLARE_METATYPE(qevercloud::UpdateNoteIfUsnMatchesResult)
7392 Q_DECLARE_METATYPE(qevercloud::InvitationShareRelationship)
7393 Q_DECLARE_METATYPE(qevercloud::ShareRelationships)
7394 Q_DECLARE_METATYPE(qevercloud::ManageNotebookSharesParameters)
7395 Q_DECLARE_METATYPE(qevercloud::ManageNotebookSharesError)
7396 Q_DECLARE_METATYPE(qevercloud::ManageNotebookSharesResult)
7397 Q_DECLARE_METATYPE(qevercloud::SharedNoteTemplate)
7398 Q_DECLARE_METATYPE(qevercloud::NotebookShareTemplate)
7399 Q_DECLARE_METATYPE(qevercloud::CreateOrUpdateNotebookSharesResult)
7400 Q_DECLARE_METATYPE(qevercloud::ManageNoteSharesError)
7401 Q_DECLARE_METATYPE(qevercloud::ManageNoteSharesResult)
7402
7403
7404 #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.](#)

```

1
9 #ifndef QEVERCLOUD_GLOBALS_H
10 #define QEVERCLOUD_GLOBALS_H
11
12 #include "Export.h"
13
14 #include <QNetworkProxy>
15
16 namespace qevercloud {
17
18     QEVERCLOUD_EXPORT QNetworkProxy evernoteNetworkProxy();
19
20     QEVERCLOUD_EXPORT void setEvernoteNetworkProxy(QNetworkProxy proxy);
21
22     QEVERCLOUD_EXPORT void resetEvernoteNetworkProxy();
23
24     Q_DECL_DEPRECATED_X("libraryVersion is deprecated, use qevercloudVersionMajor/Minor/Patch instead")
25     QEVERCLOUD_EXPORT int libraryVersion();
26
27     QEVERCLOUD_EXPORT void initializeQEverCloud();
28
29 } // namespace qevercloud
30
31 #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.](#)

```
1
2
3 8 #ifndef QEVERCLOUD_HELPERS_H
4 9 #define QEVERCLOUD_HELPERS_H
5 10
6 11 #include <QtGlobal>
7 12 #include <QObject>
8 13
9 14 #include "VersionInfo.h"
10 15
11 16 namespace qevercloud {
12 17
13 18
14 19
15 20 #if QT_VERSION < QT_VERSION_CHECK(5, 7, 0)
16 21
17 22 // this adds const to non-const objects (like std::as_const)
18 23 template <typename T>
19 24 Q_DECL_CONSTEXPR
20 25 typename std::add_const<T>::type & qAsConst(T & t) Q_DECL_NOTHROW
21 26 {
22 27     return t;
23 28 }
24 29
25 30 // prevent rvalue arguments:
26 31 template <typename T>
27 32 void qAsConst(const T &&) Q_DECL_EQ_DELETE;
28 33
29 34 #endif // QT_VERSION_CHECK
30 35
31 36
32 37 template <typename Container>
33 38 class QAssociativeContainerReferenceWrapper
34 39 {
35 40 {
```

```

41 public:
42     struct iterator
43     {
44         typename Container::iterator m_iterator;
45         iterator(const typename Container::iterator it) :
46             m_iterator(it)
47         {}
48
49         typename Container::iterator operator*()
50         {
51             return m_iterator;
52         }
53
54         iterator & operator++()
55         {
56             ++m_iterator;
57             return *this;
58         }
59
60         bool operator!=(const iterator & other) const
61     {
62         return m_iterator != other.m_iterator;
63     }
64 };
65
66 public:
67     QAssociativeContainerReferenceWrapper(Container & container)
68         : m_container(container)
69     {}
70
71     iterator begin() {
72         return m_container.begin();
73     }
74
75     iterator end() {
76         return m_container.end();
77     }
78
79 private:
80     Container & m_container;
81 };
82
84
85 template <typename Container>
86 class QAssociativeContainerConstReferenceWrapper
87 {
88 public:
89     struct iterator
90     {
91         typename Container::const_iterator m_iterator;
92         iterator(const typename Container::const_iterator it) :
93             m_iterator(it)
94         {}
95
96         typename Container::const_iterator operator*()
97         {
98             return m_iterator;
99         }
100
101         iterator & operator++()
102         {
103             ++m_iterator;
104             return *this;
105         }
106
107         bool operator!=(const iterator & other) const
108     {
109         return m_iterator != other.m_iterator;
110     }
111 };
112
113 public:
114     QAssociativeContainerConstReferenceWrapper(const Container & container)
115         : m_container(container)
116     {}
117
118     iterator begin() const {
119         return m_container.begin();
120     }
121
122     iterator end() const {
123         return m_container.end();
124     }
125
126 private:
127     const Container & m_container;
128 };

```

```

129
131
132 template <class Container>
133 QAssociativeContainerReferenceWrapper<Container> toRange(Container & container)
134 {
135     return QAssociativeContainerReferenceWrapper<Container>(container);
136 }
137
139
140 template <class Container>
141 QAssociativeContainerConstReferenceWrapper<Container> toRange(
142     const Container & container)
143 {
144     return QAssociativeContainerConstReferenceWrapper<Container>(container);
145 }
146
147 } // namespace qevercloud
148
149 #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.](#)

```

1
2
3 #ifndef QEVERCLOUD_INK_NOTE_IMAGE_DOWNLOADER_H
4 #define QEVERCLOUD_INK_NOTE_IMAGE_DOWNLOADER_H
5
6
7 #include "AsyncResult.h"
8 #include "Export.h"
9
10 #include "generated/Types.h"
11
12 #include <QByteArray>
13 #include <QString>
14 #include <QNetworkAccessManager>
15
16 namespace qevercloud {
17
18     class InkNoteImageDownloaderPrivate;
19     class QEVERCLOUD_EXPORT InkNoteImageDownloader
20     {
21     public:
22         InkNoteImageDownloader();
23

```

```

52
53
54
55
56
57
58     InkNoteImageDownloader(
59         QString host, QString shardId, QString authenticationToken, int width,
60         int height);
61
62     virtual ~InkNoteImageDownloader();
63
64     InkNoteImageDownloader & setHost(QString host);
65
66     InkNoteImageDownloader & setShardId(QString shardId);
67
68     InkNoteImageDownloader & setAuthenticationToken(
69         QString authenticationToken);
70
71     InkNoteImageDownloader & setWidth(int width);
72
73     InkNoteImageDownloader & setHeight(int height);
74
75     QByteArray download(
76         Guid guid, const bool isPublic = false,
77         const quint64 timeoutMsec = 30000);
78
79 private:
80     InkNoteImageDownloaderPrivate * const d_ptr;
81     Q_DECLARE_PRIVATE(InkNoteImageDownloader)
82 };
83
84 } // namespace qevercloud
85
86 #endif // QEVERCLOUD_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) \



### 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.](#)

```

1
2
3
4
5
6
7
8 #ifndef QEVERCLOUD_LOG_H
9 #define QEVERCLOUD_LOG_H
10
11 #include "Export.h"
12 #include "Helpers.h"
13
14 #include <QDateTime>
15 #include <QDebug>
16 #include <QObject>
17 #include <QTextStream>
18
19 #include <memory>
20
21 namespace qevercloud {
22
23
24
25 enum class LogLevel
26 {
27     Trace = 0,
28     Debug,
29     Info,
30     Warn,
31     Error
32 };
33
34 QEVERCLOUD_EXPORT QTextStream & operator«(
35     QTextStream & out, const LogLevel level);
36
37 QEVERCLOUD_EXPORT QDebug & operator«(
38     QDebug & out, const LogLevel level);
39
40
41
42 class QEVERCLOUD_EXPORT ILogger
43 {
44 public:
45     virtual bool shouldLog(
46         const LogLevel level, const char * component) const = 0;
47
48     virtual void log(
49         const LogLevel level, const char * component, const char * fileName,
50         const quint32 lineNumber, const quint64 timestamp,
51         const QString & message) = 0;
52
53     virtual void setLevel(const LogLevel level) = 0;
54
55     virtual LogLevel level() const = 0;
56 };
57
58 using ILoggerPtr = std::shared_ptr<ILogger>;
59
60
61
62 QEVERCLOUD_EXPORT ILoggerPtr logger();
63
64 QEVERCLOUD_EXPORT void setLogger(ILoggerPtr logger);
65
66 QEVERCLOUD_EXPORT ILoggerPtr nullLogger();
67
68 QEVERCLOUD_EXPORT ILoggerPtr newStdErrLogger(LogLevel level = LogLevel::Warn);
69
70
71
72 #define __QEVERCLOUD_LOG_BASE(component, level, message)
73 {
74     auto __qevercloudLogger = ::qevercloud::logger();
75     if (__qevercloudLogger->shouldLog(level, component))
76     {
77         QString msg;
78         QDebug dbg(&msg);
79         dbg.nospace();
80         dbg.noquote();
81         dbg « message;
82         __qevercloudLogger->log(
83             level,
84             component,
85             __FILE__,
86             __LINE__,
87             QDateTime::currentMSecsSinceEpoch(),
88             msg);
89     }
90 }
91 // __QEVERCLOUD_LOG_BASE
92

```



```

93 #define QEC_TRACE(component, message)
94 __QEVERCLOUD_LOG_BASE(component, LogLevel::Trace, message)
95 // QEC_TRACE
96
97 #define QEC_DEBUG(component, message)
98 __QEVERCLOUD_LOG_BASE(component, LogLevel::Debug, message)
99 // QEC_DEBUG
100
101 #define QEC_INFO(component, message)
102 __QEVERCLOUD_LOG_BASE(component, LogLevel::Info, message)
103 // QEC_INFO
104
105 #define QEC_WARNING(component, message)
106 __QEVERCLOUD_LOG_BASE(component, LogLevel::Warn, message)
107 // QEC_WARNING
108
109 #define QEC_ERROR(component, message)
110 __QEVERCLOUD_LOG_BASE(component, LogLevel::Error, message)
111 // QEC_ERROR
112
113 } // namespace qevercloud
114
115 #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.](#)

```

1
9 #ifndef QEVERCLOUD_OAUTH_H
10 #define QEVERCLOUD_OAUTH_H
11
12 #include "Export.h"
13 #include "Helpers.h"
14 #include "Printable.h"
15
16 #include "generated/Types.h"
17
18 #include <QDialog>
19 #include <QList>
20 #include <QNetworkCookie>
21 #include <QString>
22
23 namespace qevercloud {
24
25 void setNonceGenerator(quint64 (*nonceGenerator)());
26
27 class EvernoteOAuthWebViewPrivate;
28 class QEVERCLOUD_EXPORT EvernoteOAuthWebView: public QWidget
29 {
30     Q_OBJECT
31 public:
32     EvernoteOAuthWebView(QWidget * parent = Q_NULLPTR);
33
34     void authenticate(
35         QString host, QString consumerKey, QString consumerSecret,
36         const qint64 timeoutMsec = 30000);
37
38     bool isSucceeded() const;
39
40     QString oauthError() const;
41
42     struct QEVERCLOUD_EXPORT OAuthResult: public Printable
43     {
44         QString noteStoreUrl;
45         Timestamp expires;
46         QString shardId;
47         UserID userId;
48         QString webApiUrlPrefix;
49         QString authenticationToken;
50
51         QList<QNetworkCookie> cookies;
52
53         virtual void print(QTextStream & strm) const override;
54     };
55
56     OAuthResult oauthResult() const;
57
58     void setSizeHint(QSize sizeHint);
59
60     QSize sizeHint() const override;
61
62 Q_SIGNALS:
63     void authenticationFinished(bool success);
64
65     void authenticationSucceeded();
66
67     void authenticationFailed();
68
69 private:
70     EvernoteOAuthWebViewPrivate * const d_ptr;
71     Q_DECLARE_PRIVATE(EvernoteOAuthWebView)
72 };
73
74 class EvernoteOAuthDialogPrivate;
75 class QEVERCLOUD_EXPORT EvernoteOAuthDialog: public QDialog
76 {
77     Q_OBJECT
78 public:
79     using OAuthResult = EvernoteOAuthWebView::OAuthResult;
80
81     EvernoteOAuthDialog(
82         QString consumerKey, QString consumerSecret,
83         QString host = QStringLiteral("www.evernote.com"),
84         QWidget * parent = Q_NULLPTR);
85
86     virtual ~EvernoteOAuthDialog() override;
87
88     void setWebViewSizeHint(QSize sizeHint);
89

```

```

223     bool isSucceeded() const;
224
229     QString oauthError() const;
230
234     OAuthResult oauthResult() const;
235
240     virtual int exec() override;
241
244     virtual void open() override;
245
246 private:
247     EvernoteOAuthDialogPrivate * const d_ptr;
248     Q_DECLARE_PRIVATE(EvernoteOAuthDialog)
249 };
250
251 } // namespace qevercloud
252
253 #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.](#)

```

1
9 #ifndef QEVERCLOUD_OPTIONAL_H
10 #define QEVERCLOUD_OPTIONAL_H
11
12 #include "EverCloudException.h"
13
14 #include <algorithm>
15
16 namespace qevercloud {
17
42 template<typename T>
43 class Optional
44 {
45 public:
49     Optional() :
50         m_isSet(false),
51         m_value(T())
52     {}
53
57     Optional(const Optional & o) :
58         m_isSet(o.m_isSet),
59         m_value(o.m_value)
60     {}
61
66     template<typename X>
67     Optional(const Optional<X> & o) :
68         m_isSet(o.m_isSet),
69         m_value(o.m_value)
70     {}
71

```

```

75     Optional(const T & value) :
76         m_isSet(true),
77         m_value(value)
78     {}
79
80     template<typename X>
81     Optional(const X & value) :
82         m_isSet(true),
83         m_value(value)
84     {}
85
86     Optional & operator=(const Optional & o)
87     {
88         m_value = o.m_value;
89         m_isSet = o.m_isSet;
90         return *this;
91     }
92
93     template<typename X>
94     Optional & operator=(const Optional<X> & o)
95     {
96         m_value = o.m_value;
97         m_isSet = o.m_isSet;
98         return *this;
99     }
100
101     Optional & operator=(const T & value)
102     {
103         m_value = value;
104         m_isSet = true;
105         return *this;
106     }
107
108     template<typename X>
109     Optional & operator=(const X & value)
110     {
111         m_value = value;
112         m_isSet = true;
113         return *this;
114     }
115
116     operator const T&() const
117     {
118         if (!m_isSet) {
119             throw EverCloudException(
120                 "qevercloud::Optional: nonexistent value access");
121         }
122         return m_value;
123     }
124
125     operator T&()
126     {
127         if (!m_isSet) {
128             throw EverCloudException(
129                 "qevercloud::Optional: nonexistent value access");
130         }
131         return m_value;
132     }
133
134     const T & ref()const
135     {
136         if (!m_isSet) {
137             throw EverCloudException(
138                 "qevercloud::Optional: nonexistent value access");
139         }
140         return m_value;
141     }
142
143     T & ref()
144     {
145         if (!m_isSet) {
146             throw EverCloudException(
147                 "qevercloud::Optional: nonexistent value access");
148         }
149         return m_value;
150     }
151
152     bool isSet()const
153     {
154         return m_isSet;
155     }
156
157     void clear()

```

```

237     {
238         m_isSet = false;
239         m_value = T();
240     }
241
242     Optional & init()
243     {
244         m_isSet = true;
245         m_value = T();
246         return *this;
247     }
248
249     T * operator->()
250     {
251         if (!m_isSet) {
252             throw EverCloudException(
253                 "qevercloud::Optional: nonexistent value access");
254         }
255
256         return &m_value;
257     }
258
259     const T * operator->()const
260     {
261         if (!m_isSet) {
262             throw EverCloudException(
263                 "qevercloud::Optional: nonexistent value access");
264         }
265
266         return &m_value;
267     }
268
269     T value(T defaultValue = T())const
270     {
271         return m_isSet ? m_value : defaultValue;
272     }
273
274     bool isEqual(const Optional<T> & other)const
275     {
276         if (m_isSet != other.m_isSet) {
277             return false;
278         }
279
280         return !m_isSet || (m_value == other.m_value);
281     }
282
283     bool operator==(const Optional<T> & other)const
284     {
285         return isEqual(other);
286     }
287
288     bool operator!=(const Optional<T> & other)const
289     {
290         return !operator==(other);
291     }
292
293     bool operator==(const T & other)const
294     {
295         if (!m_isSet) {
296             return false;
297         }
298
299         return m_value == other;
300     }
301
302     bool operator!=(const T & other)const
303     {
304         return !operator==(other);
305     }
306
307     template<typename X> friend class Optional;
308
309     friend void swap(Optional & first, Optional & second)
310     {
311         using std::swap;
312         swap(first.m_isSet, second.m_isSet);
313         swap(first.m_value, second.m_value);
314     }
315
316     Optional(Optional && other)
317     {
318         swap(*this, other);
319     }
320
321     Optional & operator=(Optional && other)
322     {
323         swap(*this, other);
324     }

```

```

395         return *this;
396     }
397
398     Optional(T && other)
399     {
400         using std::swap;
401         m_isSet = true;
402         swap(m_value, other);
403     }
404
405     Optional & operator=(T && other)
406     {
407         using std::swap;
408         m_isSet = true;
409         swap(m_value, other);
410         return *this;
411     }
412
413 private:
414     bool m_isSet;
415     T m_value;
416 };
417
418 } // namespace qevercloud
419
420 #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.](#)

```

1
8 #ifndef QEVERCLOUD_PRINTABLE_H
9 #define QEVERCLOUD_PRINTABLE_H
10
11 #include "Export.h"
12
13 #include <QTextStream>
14 #include <QDebug>
15
16 namespace qevercloud {
17
18 class QEVERCLOUD_EXPORT Printable
19 {
20 public:
21     Printable() = default;
22     virtual ~Printable() = default;
23
24     virtual void print(QTextStream & strm) const = 0;
25
26     virtual QString toString() const;
27

```

```

28     friend QEVERCLOUD_EXPORT QTextStream & operator <<
29         QTextStream & strm, const Printable & printable);
30
31     friend QEVERCLOUD_EXPORT QDebug & operator <<
32         QDebug & dbg, const Printable & printable);
33 };
34
35 } // namespace qevercloud
36
37 #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.](#)

```

1
9 #ifndef QEVERCLOUD_INFTHEDER_H
10 #define QEVERCLOUD_INFTHEDER_H
11
12 #include "AsyncResult.h"
13 #include "DurableService.h"
14 #include "EventLoopFinisher.h"
15 #include "EverCloudException.h"
16 #include "Exceptions.h"
17 #include "Export.h"
18 #include "Globals.h"
19 #include "Helpers.h"
20 #include "InkNoteImageDownloader.h"
21 #include "Log.h"
22 #include "Optional.h"
23 #include "Printable.h"
24 #include "RequestContext.h"
25 #include "Thumbnail.h"
26 #include "VersionInfo.h"
27 #include "generated/EDAMErrorCode.h"
28 #include "generated/Constants.h"
29 #include "generated/Services.h"
30 #include "generated/Types.h"
31
32 #endif // QEVERCLOUD_INFTHEDER_H

```

## 8.39 QEverCloudOAuth.h File Reference

```
#include "OAuth.h"
#include "QEverCloud.h"
```

## 8.40 QEverCloudOAuth.h

[Go to the documentation of this file.](#)

```
1
9 #ifndef QEVERCLOUDOAUTH_INFTHEDER_H
10 #define QEVERCLOUDOAUTH_INFTHEDER_H
11
12 #include "OAuth.h"
13 #include "QEverCloud.h"
14
15 #endif // QEVERCLOUDOAUTH_INFTHEDER_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 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={})



## 8.42 RequestContext.h

[Go to the documentation of this file.](#)

```

1
2
3
4
5
6
7
8 #ifndef QEVERCLOUD_REQUEST_CONTEXT_H
9 #define QEVERCLOUD_REQUEST_CONTEXT_H
10
11 #include "Export.h"
12
13 #include <QDebug>
14 #include <QList>
15 #include <QNetworkCookie>
16 #include <QTextStream>
17 #include <QUuid>
18
19 #include <memory>
20
21 namespace qevercloud {
22
23
24
25 static constexpr quint64 DEFAULT_REQUEST_TIMEOUT_MSEC = 10000ull;
26
27 static constexpr bool DEFAULT_REQUEST_TIMEOUT_EXPONENTIAL_INCREASE = true;
28
29 static constexpr quint64 DEFAULT_MAX_REQUEST_TIMEOUT_MSEC = 600000ull;
30
31 static constexpr quint32 DEFAULT_MAX_REQUEST_RETRY_COUNT = 10;
32
33
34
35
36
37
38
39 class QEVERCLOUD_EXPORT IRequestContext
40 {
41 public:
42     virtual QUuid requestId() const = 0;
43
44     virtual QString authenticationToken() const = 0;
45
46     virtual quint64 requestTimeout() const = 0;
47
48     virtual bool increaseRequestTimeoutExponentially() const = 0;
49
50     virtual quint64 maxRequestTimeout() const = 0;
51
52     virtual quint32 maxRequestRetryCount() const = 0;
53
54     virtual QList<QNetworkCookie> cookies() const = 0;
55
56     virtual IRequestContext * clone() const = 0;
57
58     virtual ~IRequestContext() = default;
59
60     friend QEVERCLOUD_EXPORT QTextStream & operator<<(
61         QTextStream & strm, const IRequestContext & ctx);
62
63     friend QEVERCLOUD_EXPORT QDebug & operator<<(
64         QDebug & dbg, const IRequestContext & ctx);
65 };
66
67 using IRequestContextPtr = std::shared_ptr<IRequestContext>;
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85 QEVERCLOUD_EXPORT IRequestContextPtr newRequestContext(
86     QString authenticationToken = {},
87     quint64 requestTimeout = DEFAULT_REQUEST_TIMEOUT_MSEC,
88     bool increaseRequestTimeoutExponentially = DEFAULT_REQUEST_TIMEOUT_EXPONENTIAL_INCREASE,
89     quint64 maxRequestTimeout = DEFAULT_MAX_REQUEST_TIMEOUT_MSEC,
90     quint32 maxRequestRetryCount = DEFAULT_MAX_REQUEST_RETRY_COUNT,
91     QList<QNetworkCookie> cookies = {});
92
93 } // namespace qevercloud
94
95 #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.](#)

```
1
9 #ifndef QEVERCLOUD_THUMBNAIL_H
10 #define QEVERCLOUD_THUMBNAIL_H
11
12 #include "AsyncResult.h"
13 #include "Export.h"
14
15 #include "generated/Types.h"
16
17 #include <QByteArray>
18 #include <QNetworkAccessManager>
19 #include <QString>
20
21 #include <utility>
22
23 namespace qevercloud {
24
25     class ThumbnailPrivate;
26     class QEVERCLOUD_EXPORT Thumbnail
27     {
28     public:
29         enum class ImageType
30         {
31             PNG,
32             JPEG,
33             GIF,
34             BMP
35         };
36
37         friend QEVERCLOUD_EXPORT QTextStream & operator<<(
38             QTextStream & strm, const ImageType imageType);
39
40         friend QEVERCLOUD_EXPORT QDebug & operator<<(
41             QDebug & dbg, const ImageType imageType);
42
43         Thumbnail();
44
45         Thumbnail(
46             QString host, QString shardId, QString authenticationToken,
47             int size = 300, ImageType imageType = ImageType::PNG);
48
49         virtual ~Thumbnail();
50
51         Thumbnail & setHost(QString host);
52
53         Thumbnail & setShardId(QString shardId);
54
55         Thumbnail & setAuthenticationToken(QString authenticationToken);
56
57         Thumbnail & setSize(int size);
58
59         Thumbnail & setImageType(ImageType imageType);
60
61         QByteArray download(
```

```
149         Guid guid, const bool isPublic = false, const bool isResourceGuid = false,
150         const qint64 timeoutMsec = 30000);
151
152     AsyncResult * downloadAsync(
153         Guid guid, const bool isPublic = false, const bool isResourceGuid = false,
154         const qint64 timeoutMsec = 30000);
155
156     std::pair<QNetworkRequest, QByteArray> createPostRequest(
157         qevercloud::Guid guid, bool isPublic = false, bool isResourceGuid = false);
158
159 private:
160     ThumbnailPrivate * const d_ptr;
161     Q_DECLARE_PRIVATE(Thumbnail)
162 };
163
164 } // namespace qevercloud
165
166 #endif // QEVERCLOUD_THUMBNAIL_H
```

## 8.45 README.md File Reference



# Index

- \_\_QEVERCLOUD\_LOG\_BASE
  - Log.h, [622](#)
- ~AsyncResult
  - qevercloud::AsyncResult, [100](#)
- ~EDAMInvalidContactsException
  - qevercloud::EDAMInvalidContactsException, [129](#)
- ~EDAMNotFoundException
  - qevercloud::EDAMNotFoundException, [134](#)
- ~EDAMSystemException
  - qevercloud::EDAMSystemException, [138](#)
- ~EDAMUserException
  - qevercloud::EDAMUserException, [147](#)
- ~EventLoopFinisher
  - qevercloud::EventLoopFinisher, [151](#)
- ~EverCloudException
  - qevercloud::EverCloudException, [152](#)
- ~EverCloudLocalData
  - qevercloud::EverCloudLocalData, [157](#)
- ~EvernoteOAuthDialog
  - qevercloud::EvernoteOAuthDialog, [163](#)
- ~IRequestContext
  - qevercloud::IRequestContext, [243](#)
- ~InkNoteImageDownloader
  - qevercloud::InkNoteImageDownloader, [173](#)
- ~NetworkException
  - qevercloud::NetworkException, [285](#)
- ~Printable
  - qevercloud::Printable, [409](#)
- ~ThriftException
  - qevercloud::ThriftException, [476](#)
- ~Thumbnail
  - qevercloud::Thumbnail, [481](#)
- accessType
  - qevercloud::RelatedContent, [419](#)
- ACCOUNT\_CLEAR
  - qevercloud, [35](#)
- accountEmailDomain
  - qevercloud::BootstrapSettings, [109](#)
- accounting
  - qevercloud::User, [488](#)
- accountLimits
  - qevercloud::User, [488](#)
- ACTIVE
  - qevercloud, [33](#), [37](#)
- active
  - qevercloud::Note, [289](#)
  - qevercloud::Resource, [435](#)
  - qevercloud::User, [488](#)
- ADMIN
  - qevercloud, [32](#), [38](#)
- alternateData
  - qevercloud::Resource, [435](#)
- altitude
  - qevercloud::NoteAttributes, [294](#)
  - qevercloud::ResourceAttributes, [438](#)
- applicationData
  - qevercloud::NoteAttributes, [295](#)
  - qevercloud::ResourceAttributes, [438](#)
- APPROVED
  - qevercloud, [32](#)
- ascending
  - qevercloud::NoteFilter, [324](#)
  - qevercloud::Publishing, [414](#)
- asIs
  - qevercloud::AsyncResult, [100](#)
- ASSIGNED
  - qevercloud, [40](#)
- AsyncRequest
  - qevercloud::IDurableService::AsyncRequest, [97](#)
- AsyncResult
  - qevercloud::AsyncResult, [99](#), [100](#)
- AsyncResult.h, [515](#)
- AsyncServiceCall
  - qevercloud::IDurableService, [170](#)
- attachment
  - qevercloud::ResourceAttributes, [438](#)
- attributes
  - qevercloud::Note, [289](#)
  - qevercloud::NoteMetadata, [336](#)
  - qevercloud::Resource, [435](#)
  - qevercloud::User, [488](#)
  - qevercloud::UserProfile, [501](#)
- AUTH\_EXPIRED
  - qevercloud, [35](#)
- authenticate
  - qevercloud::EvernoteOAuthWebView, [165](#)
- authenticateLongSession
  - qevercloud::IUserStore, [249](#)
- authenticateLongSessionAsync
  - qevercloud::IUserStore, [251](#)
- authenticateLongSessionRequest
  - qevercloud::UserStoreServer, [504](#)
- authenticateLongSessionRequestReady
  - qevercloud::UserStoreServer, [504](#)
- authenticateToBusiness
  - qevercloud::IUserStore, [251](#)
- authenticateToBusinessAsync
  - qevercloud::IUserStore, [252](#)

- authenticateToBusinessRequest
  - qevercloud::UserStoreServer, 505
- authenticateToBusinessRequestReady
  - qevercloud::UserStoreServer, 505
- authenticateToSharedNote
  - qevercloud::INoteStore, 180
- authenticateToSharedNoteAsync
  - qevercloud::INoteStore, 181
- authenticateToSharedNotebook
  - qevercloud::INoteStore, 181
- authenticateToSharedNotebookAsync
  - qevercloud::INoteStore, 182
- authenticateToSharedNotebookRequest
  - qevercloud::NoteStoreServer, 359
- authenticateToSharedNotebookRequestReady
  - qevercloud::NoteStoreServer, 360
- authenticateToSharedNoteRequest
  - qevercloud::NoteStoreServer, 360
- authenticateToSharedNoteRequestReady
  - qevercloud::NoteStoreServer, 360
- authenticationFailed
  - qevercloud::EvernoteOAuthWebView, 166
- authenticationFinished
  - qevercloud::EvernoteOAuthWebView, 166
- authenticationSucceeded
  - qevercloud::EvernoteOAuthWebView, 166
- authenticationToken
  - qevercloud::AuthenticationResult, 103
  - qevercloud::EvernoteOAuthWebView::OAuthResult, 400
  - qevercloud::IRequestContext, 243
- author
  - qevercloud::NoteAttributes, 295
- authors
  - qevercloud::RelatedContent, 419
- availablePoints
  - qevercloud::Accounting, 90
- BAD\_ADDRESS
  - qevercloud, 36
- BAD\_DATA\_FORMAT
  - qevercloud, 35
- BAD\_SEQUENCE\_ID
  - qevercloud::ThriftException, 476
- BASIC
  - qevercloud, 40
- begin
  - qevercloud::QAssociativeContainerConstReferenceWrapper<Container >, 416
  - qevercloud::QAssociativeContainerReferenceWrapper<Container >, 417
- bestPrivilege
  - qevercloud::MemberShareRelationship, 282
- blocked
  - qevercloud::Identity, 168
- BMP
  - qevercloud::Thumbnail, 481
- body
  - qevercloud::Data, 127
- bodyHash
  - qevercloud::Data, 127
- bool
  - qevercloud::EverCloudLocalData, 157
- BUSINESS
  - qevercloud, 40
- BUSINESS\_FULL\_ACCESS
  - qevercloud, 41
- BUSINESS\_SECURITY\_LOGIN\_REQUIRED
  - qevercloud, 35
- businessAddress
  - qevercloud::UserAttributes, 492
- businessId
  - qevercloud::Accounting, 91
  - qevercloud::BusinessInvitation, 112
  - qevercloud::BusinessUserInfo, 119
  - qevercloud::LinkedNotebook, 265
- BusinessInvitationStatus
  - qevercloud, 32
- businessName
  - qevercloud::Accounting, 91
  - qevercloud::BusinessUserInfo, 119
- businessNotebook
  - qevercloud::Notebook, 301
- businessRole
  - qevercloud::Accounting, 91
- businessUserInfo
  - qevercloud::User, 488
- BusinessUserRole
  - qevercloud, 32
- BusinessUserStatus
  - qevercloud, 32
- cacheExpires
  - qevercloud::RelatedResult, 427
- cacheKey
  - qevercloud::RelatedQuery, 425
  - qevercloud::RelatedResult, 427
- cameraMake
  - qevercloud::ResourceAttributes, 439
- cameraModel
  - qevercloud::ResourceAttributes, 439
- CAN\_BE\_MOVED
  - qevercloud, 33
- CANCELED
  - qevercloud, 37
- CANCELLATION\_PENDING
  - qevercloud, 37
- canMoveToContainer
  - qevercloud::CanMoveToContainerRestrictions, 121
- canMoveToContainerRestrictions
  - qevercloud::NotebookRestrictions, 310
- CanMoveToContainerStatus
  - qevercloud, 33
- ccAddresses
  - qevercloud::NoteEmailParameters, 321
- checkVersion
  - qevercloud::IUserStore, 252
- checkVersionAsync

- qevercloud::IUserStore, 253
- checkVersionRequest
  - qevercloud::UserStoreServer, 505
- checkVersionRequestReady
  - qevercloud::UserStoreServer, 505
- chunkHighUSN
  - qevercloud::SyncChunk, 461
- CLASSIFICATION\_RECIPE\_SERVICE\_RECIPE
  - qevercloud, 59
- CLASSIFICATION\_RECIPE\_USER\_NON\_RECIPE
  - qevercloud, 59
- CLASSIFICATION\_RECIPE\_USER\_RECIPE
  - qevercloud, 59
- Classifications
  - qevercloud::NoteAttributes, 294
- classifications
  - qevercloud::NoteAttributes, 295, 299
- clear
  - qevercloud::Optional< T >, 403
- clientWillIndex
  - qevercloud::ResourceAttributes, 439
- clipFullPage
  - qevercloud::UserAttributes, 492
- clipUrl
  - qevercloud::RelatedContent, 419
- clone
  - qevercloud::IRequestContext, 243
- comments
  - qevercloud::UserAttributes, 492
- companyStartDate
  - qevercloud::BusinessUserAttributes, 117
- completeTwoFactorAuthentication
  - qevercloud::IUserStore, 253
- completeTwoFactorAuthenticationAsync
  - qevercloud::IUserStore, 254
- completeTwoFactorAuthenticationRequest
  - qevercloud::UserStoreServer, 505
- completeTwoFactorAuthenticationRequestReady
  - qevercloud::UserStoreServer, 505
- conflictSourceNoteGuid
  - qevercloud::NoteAttributes, 295
- Constants.h, 525, 529
- contact
  - qevercloud::Identity, 168
  - qevercloud::Notebook, 301
  - qevercloud::RelatedContent, 419
- contactName
  - qevercloud::NotebookDescriptor, 305
- contacts
  - qevercloud::EDAMInvalidContactsException, 130
- ContactType
  - qevercloud, 33
- containingNotebooks
  - qevercloud::RelatedResult, 428, 429
- content
  - qevercloud::Note, 289
- contentClass
  - qevercloud::NoteAttributes, 295
- contentHash
  - qevercloud::Note, 289
- contentId
  - qevercloud::RelatedContent, 419
- contentLength
  - qevercloud::Note, 289
  - qevercloud::NoteMetadata, 336
- contentType
  - qevercloud::RelatedContent, 419
- context
  - qevercloud::NoteFilter, 324
  - qevercloud::RelatedQuery, 425
- cookies
  - qevercloud::EvernoteOAuthWebView::OAuthResult, 400
  - qevercloud::IRequestContext, 243
- copyNote
  - qevercloud::INoteStore, 182
- copyNoteAsync
  - qevercloud::INoteStore, 183
- copyNoteRequest
  - qevercloud::NoteStoreServer, 360
- copyNoteRequestReady
  - qevercloud::NoteStoreServer, 360
- CREATED
  - qevercloud, 37
- created
  - qevercloud::BusinessInvitation, 112
  - qevercloud::Note, 290
  - qevercloud::NoteMetadata, 337
  - qevercloud::User, 488
- createLinkedNotebook
  - qevercloud::INoteStore, 183
- createLinkedNotebookAsync
  - qevercloud::INoteStore, 184
- createLinkedNotebookRequest
  - qevercloud::NoteStoreServer, 360
- createLinkedNotebookRequestReady
  - qevercloud::NoteStoreServer, 361
- createNote
  - qevercloud::INoteStore, 184
- createNoteAsync
  - qevercloud::INoteStore, 185
- createNotebook
  - qevercloud::INoteStore, 186
- createNotebookAsync
  - qevercloud::INoteStore, 187
- createNotebookRequest
  - qevercloud::NoteStoreServer, 361
- createNotebookRequestReady
  - qevercloud::NoteStoreServer, 361
- createNoteRequest
  - qevercloud::NoteStoreServer, 361
- createNoteRequestReady
  - qevercloud::NoteStoreServer, 361
- createOrUpdateNotebookShares
  - qevercloud::INoteStore, 187
- createOrUpdateNotebookSharesAsync

- qevercloud::INoteStore, 188
- createOrUpdateNotebookSharesRequest
  - qevercloud::NoteStoreServer, 361
- createOrUpdateNotebookSharesRequestReady
  - qevercloud::NoteStoreServer, 362
- createPostRequest
  - qevercloud::Thumbnail, 482
- createSearch
  - qevercloud::INoteStore, 188
- createSearchAsync
  - qevercloud::INoteStore, 189
- createSearchRequest
  - qevercloud::NoteStoreServer, 362
- createSearchRequestReady
  - qevercloud::NoteStoreServer, 362
- createTag
  - qevercloud::INoteStore, 189
- createTagAsync
  - qevercloud::INoteStore, 190
- createTagRequest
  - qevercloud::NoteStoreServer, 362
- createTagRequestReady
  - qevercloud::NoteStoreServer, 362
- creatorId
  - qevercloud::NoteAttributes, 296
- currency
  - qevercloud::Accounting, 91
- currentTime
  - qevercloud::AuthenticationResult, 103
  - qevercloud::SyncChunk, 461
  - qevercloud::SyncState, 471
- dailyEmailLimit
  - qevercloud::UserAttributes, 493
- data
  - qevercloud::Resource, 435
- DATA\_CONFLICT
  - qevercloud, 35
- DATA\_REQUIRED
  - qevercloud, 35
- date
  - qevercloud::RelatedContent, 419
- dateAgreedToTermsOfService
  - qevercloud::UserAttributes, 493
- DEACTIVATED
  - qevercloud, 33
- deactivated
  - qevercloud::Identity, 168
- Debug
  - qevercloud, 36
- debugInfo
  - qevercloud::NoteList, 331
  - qevercloud::NotesMetadataList, 349
  - qevercloud::RelatedResult, 428
- defaultLatitude
  - qevercloud::UserAttributes, 493
- defaultLocationName
  - qevercloud::UserAttributes, 493
- defaultLongitude
  - qevercloud::UserAttributes, 493
- defaultNotebook
  - qevercloud::Notebook, 301
- deleted
  - qevercloud::Note, 290
  - qevercloud::NoteMetadata, 337
  - qevercloud::User, 488
- deleteNote
  - qevercloud::INoteStore, 190
- deleteNoteAsync
  - qevercloud::INoteStore, 191
- deleteNoteRequest
  - qevercloud::NoteStoreServer, 362
- deleteNoteRequestReady
  - qevercloud::NoteStoreServer, 363
- department
  - qevercloud::BusinessUserAttributes, 117
- DEVICE\_LIMIT\_REACHED
  - qevercloud, 35
- Dict
  - qevercloud::EverCloudLocalData, 156
- dict
  - qevercloud::EverCloudLocalData, 158
- DIRECT\_LINK\_ACCESS\_OK
  - qevercloud, 39
- DIRECT\_LINK\_EMBEDDED\_VIEW
  - qevercloud, 39
- DIRECT\_LINK\_LOGIN\_REQUIRED
  - qevercloud, 39
- dirty
  - qevercloud::EverCloudLocalData, 158
- displayName
  - qevercloud::InvitationShareRelationship, 241
  - qevercloud::MemberShareRelationship, 283
  - qevercloud::NoteInvitationShareRelationship, 327
  - qevercloud::NoteMemberShareRelationship, 334
- DO\_NOT\_SEND
  - qevercloud, 40
- download
  - qevercloud::InkNoteImageDownloader, 174
  - qevercloud::Thumbnail, 482
- downloadAsync
  - qevercloud::Thumbnail, 482
- DUPLICATE\_CONTACT
  - qevercloud, 36
- DurableService
  - qevercloud::AsyncResult, 101
- DurableService.h, 516, 517
- duration
  - qevercloud::Resource, 435
- EDAM\_APP\_RATING\_MAX
  - qevercloud, 59
- EDAM\_APP\_RATING\_MIN
  - qevercloud, 59
- EDAM\_APPLICATIONDATA\_ENTRY\_LEN\_MAX
  - qevercloud, 60
- EDAM\_APPLICATIONDATA\_NAME\_LEN\_MAX
  - qevercloud, 60



EDAM\_APPLICATIONDATA\_NAME\_LEN\_MIN  
qevercloud, 60

EDAM\_APPLICATIONDATA\_NAME\_REGEX  
qevercloud, 60

EDAM\_APPLICATIONDATA\_VALUE\_LEN\_MAX  
qevercloud, 60

EDAM\_APPLICATIONDATA\_VALUE\_LEN\_MIN  
qevercloud, 60

EDAM\_APPLICATIONDATA\_VALUE\_REGEX  
qevercloud, 60

EDAM\_ATTRIBUTE\_LEN\_MAX  
qevercloud, 61

EDAM\_ATTRIBUTE\_LEN\_MIN  
qevercloud, 61

EDAM\_ATTRIBUTE\_LIST\_MAX  
qevercloud, 61

EDAM\_ATTRIBUTE\_MAP\_MAX  
qevercloud, 61

EDAM\_ATTRIBUTE\_REGEX  
qevercloud, 61

EDAM\_BUSINESS\_MARKETING\_CODE\_REGEX\_PATTERN  
qevercloud, 61

EDAM\_BUSINESS\_NOTEBOOK\_DESCRIPTION\_LEN\_MAX  
qevercloud, 61

EDAM\_BUSINESS\_NOTEBOOK\_DESCRIPTION\_LEN\_MIN  
qevercloud, 62

EDAM\_BUSINESS\_NOTEBOOK\_DESCRIPTION\_REGEX  
qevercloud, 62

EDAM\_BUSINESS\_NOTEBOOKS\_MAX  
qevercloud, 62

EDAM\_BUSINESS\_NOTES\_MAX  
qevercloud, 62

EDAM\_BUSINESS\_PHONE\_NUMBER\_LEN\_MAX  
qevercloud, 62

EDAM\_BUSINESS\_TAGS\_MAX  
qevercloud, 62

EDAM\_BUSINESS\_URI\_LEN\_MAX  
qevercloud, 62

EDAM\_BUSINESS\_WORKSPACES\_MAX  
qevercloud, 63

EDAM\_CONNECTED\_IDENTITY\_REQUEST\_MAX  
qevercloud, 63

EDAM\_CONTENT\_CLASS\_FOOD\_MEAL  
qevercloud, 63

EDAM\_CONTENT\_CLASS\_HELLO\_ENCOUNTER  
qevercloud, 63

EDAM\_CONTENT\_CLASS\_HELLO\_PROFILE  
qevercloud, 63

EDAM\_CONTENT\_CLASS\_PENULTIMATE\_NOTEBOOK  
qevercloud, 63

EDAM\_CONTENT\_CLASS\_PENULTIMATE\_PREFIX  
qevercloud, 63

EDAM\_CONTENT\_CLASS\_SKETCH  
qevercloud, 64

EDAM\_CONTENT\_CLASS\_SKETCH\_PDF  
qevercloud, 64

EDAM\_CONTENT\_CLASS\_SKETCH\_PREFIX  
qevercloud, 64

EDAM\_DEVICE\_DESCRIPTION\_LEN\_MAX  
qevercloud, 64

EDAM\_DEVICE\_DESCRIPTION\_REGEX  
qevercloud, 64

EDAM\_DEVICE\_ID\_LEN\_MAX  
qevercloud, 64

EDAM\_DEVICE\_ID\_REGEX  
qevercloud, 64

EDAM\_EMAIL\_DOMAIN\_REGEX  
qevercloud, 65

EDAM\_EMAIL\_LEN\_MAX  
qevercloud, 65

EDAM\_EMAIL\_LEN\_MIN  
qevercloud, 65

EDAM\_EMAIL\_LOCAL\_REGEX  
qevercloud, 65

EDAM\_EMAIL\_REGEX  
qevercloud, 65

EDAM\_FIND\_CONTACT\_DEFAULT\_MAX\_RESULTS  
qevercloud, 65

EDAM\_FIND\_CONTACT\_MAX\_RESULTS  
qevercloud, 65

EDAM\_FOOD\_APP\_CONTENT\_CLASS\_PREFIX  
qevercloud, 66

EDAM\_GET\_ORDERS\_MAX\_RESULTS  
qevercloud, 66

EDAM\_GUID\_LEN\_MAX  
qevercloud, 66

EDAM\_GUID\_LEN\_MIN  
qevercloud, 66

EDAM\_GUID\_REGEX  
qevercloud, 66

EDAM\_HASH\_LEN  
qevercloud, 66

EDAM\_HELLO\_APP\_CONTENT\_CLASS\_PREFIX  
qevercloud, 66

EDAM\_INDEXABLE\_PLAINTEXT\_MIME\_TYPES  
qevercloud, 67

EDAM\_INDEXABLE\_RESOURCE\_MIME\_TYPES  
qevercloud, 67

EDAM\_MAX\_PREFERENCES  
qevercloud, 67

EDAM\_MAX\_VALUES\_PER\_PREFERENCE  
qevercloud, 67

EDAM\_MESSAGE\_ATTACHMENT\_SNIPPET\_LEN\_MAX  
qevercloud, 67

EDAM\_MESSAGE\_ATTACHMENT\_SNIPPET\_REGEX  
qevercloud, 67

EDAM\_MESSAGE\_ATTACHMENT\_TITLE\_LEN\_MAX  
qevercloud, 67

EDAM\_MESSAGE\_ATTACHMENT\_TITLE\_REGEX  
qevercloud, 68

EDAM\_MESSAGE\_ATTACHMENTS\_MAX  
qevercloud, 68

EDAM\_MESSAGE\_BODY\_LEN\_MAX  
qevercloud, 68

EDAM\_MESSAGE\_BODY\_REGEX  
qevercloud, 68

EDAM\_MESSAGE\_RECIPIENTS\_MAX  
qevercloud, 68

EDAM\_MIME\_LEN\_MAX  
qevercloud, 68

EDAM\_MIME\_LEN\_MIN  
qevercloud, 68

EDAM\_MIME\_REGEX  
qevercloud, 68

EDAM\_MIME\_TYPE\_AAC  
qevercloud, 69

EDAM\_MIME\_TYPE\_AMR  
qevercloud, 69

EDAM\_MIME\_TYPE\_BMP  
qevercloud, 69

EDAM\_MIME\_TYPE\_DEFAULT  
qevercloud, 69

EDAM\_MIME\_TYPE\_GIF  
qevercloud, 69

EDAM\_MIME\_TYPE\_INK  
qevercloud, 69

EDAM\_MIME\_TYPE\_JPEG  
qevercloud, 69

EDAM\_MIME\_TYPE\_M4A  
qevercloud, 69

EDAM\_MIME\_TYPE\_MP3  
qevercloud, 70

EDAM\_MIME\_TYPE\_MP4\_VIDEO  
qevercloud, 70

EDAM\_MIME\_TYPE\_PDF  
qevercloud, 70

EDAM\_MIME\_TYPE\_PNG  
qevercloud, 70

EDAM\_MIME\_TYPE\_TIFF  
qevercloud, 70

EDAM\_MIME\_TYPE\_WAV  
qevercloud, 70

EDAM\_MIME\_TYPES  
qevercloud, 70

EDAM\_NOTE\_BUSINESS\_SHARED\_NOTE\_MAX  
qevercloud, 70

EDAM\_NOTE\_CONTENT\_CLASS\_LEN\_MAX  
qevercloud, 71

EDAM\_NOTE\_CONTENT\_CLASS\_LEN\_MIN  
qevercloud, 71

EDAM\_NOTE\_CONTENT\_CLASS\_REGEX  
qevercloud, 71

EDAM\_NOTE\_CONTENT\_LEN\_MAX  
qevercloud, 71

EDAM\_NOTE\_CONTENT\_LEN\_MIN  
qevercloud, 71

EDAM\_NOTE\_LOCK\_VIEWERS\_NOTES\_MAX  
qevercloud, 71

EDAM\_NOTE\_PERSONAL\_SHARED\_NOTE\_MAX  
qevercloud, 71

EDAM\_NOTE\_RESOURCES\_MAX  
qevercloud, 71

EDAM\_NOTE\_SIZE\_MAX\_FREE  
qevercloud, 72

EDAM\_NOTE\_SIZE\_MAX\_PREMIUM  
qevercloud, 72

EDAM\_NOTE\_SOURCE\_MAIL\_CLIP  
qevercloud, 72

EDAM\_NOTE\_SOURCE\_MAIL\_SMTP\_GATEWAY  
qevercloud, 72

EDAM\_NOTE\_SOURCE\_WEB\_CLIP  
qevercloud, 72

EDAM\_NOTE\_SOURCE\_WEB\_CLIP\_SIMPLIFIED  
qevercloud, 72

EDAM\_NOTE\_TAGS\_MAX  
qevercloud, 72

EDAM\_NOTE\_TITLE\_LEN\_MAX  
qevercloud, 73

EDAM\_NOTE\_TITLE\_LEN\_MIN  
qevercloud, 73

EDAM\_NOTE\_TITLE\_QUALITY\_HIGH  
qevercloud, 73

EDAM\_NOTE\_TITLE\_QUALITY\_LOW  
qevercloud, 73

EDAM\_NOTE\_TITLE\_QUALITY\_MEDIUM  
qevercloud, 73

EDAM\_NOTE\_TITLE\_QUALITY\_UNTITLED  
qevercloud, 73

EDAM\_NOTE\_TITLE\_REGEX  
qevercloud, 73

EDAM\_NOTEBOOK\_BUSINESS\_SHARED\_NOTEBOOK\_MAX  
qevercloud, 74

EDAM\_NOTEBOOK\_NAME\_LEN\_MAX  
qevercloud, 74

EDAM\_NOTEBOOK\_NAME\_LEN\_MIN  
qevercloud, 74

EDAM\_NOTEBOOK\_NAME\_REGEX  
qevercloud, 74

EDAM\_NOTEBOOK\_PERSONAL\_SHARED\_NOTEBOOK\_MAX  
qevercloud, 74

EDAM\_NOTEBOOK\_STACK\_LEN\_MAX  
qevercloud, 74

EDAM\_NOTEBOOK\_STACK\_LEN\_MIN  
qevercloud, 74

EDAM\_NOTEBOOK\_STACK\_REGEX  
qevercloud, 75

EDAM\_OPEN\_ID\_ACCESS\_TOKEN\_MAX  
qevercloud, 75

EDAM\_PREFERENCE\_BUSINESS\_DEFAULT\_NOTEBOOK  
qevercloud, 75

EDAM\_PREFERENCE\_BUSINESS\_QUICKNOTE  
qevercloud, 75

EDAM\_PREFERENCE\_NAME\_LEN\_MAX  
qevercloud, 75

EDAM\_PREFERENCE\_NAME\_LEN\_MIN  
qevercloud, 75

EDAM\_PREFERENCE\_NAME\_REGEX  
qevercloud, 76

EDAM\_PREFERENCE\_ONLY\_ONE\_VALUE\_LEN\_MAX  
qevercloud, 76

EDAM\_PREFERENCE\_ONLY\_ONE\_VALUE\_REGEX  
qevercloud, 76

- EDAM\_PREFERENCE\_SHORTCUTS
  - qevercloud, [76](#)
- EDAM\_PREFERENCE\_SHORTCUTS\_MAX\_VALUES
  - qevercloud, [76](#)
- EDAM\_PREFERENCE\_VALUE\_LEN\_MAX
  - qevercloud, [76](#)
- EDAM\_PREFERENCE\_VALUE\_LEN\_MIN
  - qevercloud, [76](#)
- EDAM\_PREFERENCE\_VALUE\_REGEX
  - qevercloud, [77](#)
- EDAM\_PROMOTION\_ID\_LEN\_MAX
  - qevercloud, [77](#)
- EDAM\_PROMOTION\_ID\_REGEX
  - qevercloud, [77](#)
- EDAM\_PUBLISHING\_DESCRIPTION\_LEN\_MAX
  - qevercloud, [77](#)
- EDAM\_PUBLISHING\_DESCRIPTION\_LEN\_MIN
  - qevercloud, [77](#)
- EDAM\_PUBLISHING\_DESCRIPTION\_REGEX
  - qevercloud, [77](#)
- EDAM\_PUBLISHING\_URI\_LEN\_MAX
  - qevercloud, [77](#)
- EDAM\_PUBLISHING\_URI\_LEN\_MIN
  - qevercloud, [78](#)
- EDAM\_PUBLISHING\_URI\_PROHIBITED
  - qevercloud, [78](#)
- EDAM\_PUBLISHING\_URI\_REGEX
  - qevercloud, [78](#)
- EDAM\_RELATED\_MAX\_EXPERTS
  - qevercloud, [78](#)
- EDAM\_RELATED\_MAX\_NOTEBOOKS
  - qevercloud, [78](#)
- EDAM\_RELATED\_MAX\_NOTES
  - qevercloud, [78](#)
- EDAM\_RELATED\_MAX\_RELATED\_CONTENT
  - qevercloud, [78](#)
- EDAM\_RELATED\_MAX\_TAGS
  - qevercloud, [78](#)
- EDAM\_RELATED\_PLAINTEXT\_LEN\_MAX
  - qevercloud, [79](#)
- EDAM\_RELATED\_PLAINTEXT\_LEN\_MIN
  - qevercloud, [79](#)
- EDAM\_RESOURCE\_SIZE\_MAX\_FREE
  - qevercloud, [79](#)
- EDAM\_RESOURCE\_SIZE\_MAX\_PREMIUM
  - qevercloud, [79](#)
- EDAM\_SAVED\_SEARCH\_NAME\_LEN\_MAX
  - qevercloud, [79](#)
- EDAM\_SAVED\_SEARCH\_NAME\_LEN\_MIN
  - qevercloud, [79](#)
- EDAM\_SAVED\_SEARCH\_NAME\_REGEX
  - qevercloud, [79](#)
- EDAM\_SEARCH\_QUERY\_LEN\_MAX
  - qevercloud, [79](#)
- EDAM\_SEARCH\_QUERY\_LEN\_MIN
  - qevercloud, [80](#)
- EDAM\_SEARCH\_QUERY\_REGEX
  - qevercloud, [80](#)
- EDAM\_SEARCH\_SUGGESTIONS\_MAX
  - qevercloud, [80](#)
- EDAM\_SEARCH\_SUGGESTIONS\_PREFIX\_LEN\_MAX
  - qevercloud, [80](#)
- EDAM\_SEARCH\_SUGGESTIONS\_PREFIX\_LEN\_MIN
  - qevercloud, [80](#)
- EDAM\_SNIPPETS\_NOTES\_MAX
  - qevercloud, [80](#)
- EDAM\_SOURCE\_APPLICATION\_ANDROID\_SHARE\_EXTENSION
  - qevercloud, [80](#)
- EDAM\_SOURCE\_APPLICATION\_EN\_SCANSNAP
  - qevercloud, [81](#)
- EDAM\_SOURCE\_APPLICATION\_EWC
  - qevercloud, [81](#)
- EDAM\_SOURCE\_APPLICATION\_IOS\_SHARE\_EXTENSION
  - qevercloud, [81](#)
- EDAM\_SOURCE\_APPLICATION\_MOLESKINE
  - qevercloud, [81](#)
- EDAM\_SOURCE\_APPLICATION\_POSTIT
  - qevercloud, [81](#)
- EDAM\_SOURCE\_APPLICATION\_WEB\_CLIPPER
  - qevercloud, [81](#)
- EDAM\_SOURCE\_OUTLOOK\_CLIPPER
  - qevercloud, [81](#)
- EDAM\_TAG\_NAME\_LEN\_MAX
  - qevercloud, [82](#)
- EDAM\_TAG\_NAME\_LEN\_MIN
  - qevercloud, [82](#)
- EDAM\_TAG\_NAME\_REGEX
  - qevercloud, [82](#)
- EDAM\_TIMEZONE\_LEN\_MAX
  - qevercloud, [82](#)
- EDAM\_TIMEZONE\_LEN\_MIN
  - qevercloud, [82](#)
- EDAM\_TIMEZONE\_REGEX
  - qevercloud, [82](#)
- EDAM\_USER\_LINKED\_NOTEBOOK\_MAX
  - qevercloud, [82](#)
- EDAM\_USER\_LINKED\_NOTEBOOK\_MAX\_PREMIUM
  - qevercloud, [83](#)
- EDAM\_USER\_MAIL\_LIMIT\_DAILY\_FREE
  - qevercloud, [83](#)
- EDAM\_USER\_MAIL\_LIMIT\_DAILY\_PREMIUM
  - qevercloud, [83](#)
- EDAM\_USER\_NAME\_LEN\_MAX
  - qevercloud, [83](#)
- EDAM\_USER\_NAME\_LEN\_MIN
  - qevercloud, [83](#)
- EDAM\_USER\_NAME\_REGEX
  - qevercloud, [83](#)
- EDAM\_USER\_NOTEBOOKS\_MAX
  - qevercloud, [83](#)
- EDAM\_USER\_NOTES\_MAX
  - qevercloud, [84](#)
- EDAM\_USER\_PASSWORD\_LEN\_MAX
  - qevercloud, [84](#)
- EDAM\_USER\_PASSWORD\_LEN\_MIN
  - qevercloud, [84](#)

- EDAM\_USER\_PASSWORD\_REGEX
  - qevercloud, [84](#)
- EDAM\_USER\_PROFILE\_PHOTO\_MAX\_BYTES
  - qevercloud, [84](#)
- EDAM\_USER\_RECENT\_MAILED\_ADDRESSES\_MAX
  - qevercloud, [84](#)
- EDAM\_USER\_SAVED\_SEARCHES\_MAX
  - qevercloud, [84](#)
- EDAM\_USER\_TAGS\_MAX
  - qevercloud, [85](#)
- EDAM\_USER\_UPLOAD\_LIMIT\_BUSINESS
  - qevercloud, [85](#)
- EDAM\_USER\_UPLOAD\_LIMIT\_BUSINESS\_FIRST\_MONTH
  - qevercloud, [85](#)
- EDAM\_USER\_UPLOAD\_LIMIT\_BUSINESS\_NEXT\_MONTH
  - qevercloud, [85](#)
- EDAM\_USER\_UPLOAD\_LIMIT\_BUSINESS\_PER\_USER
  - qevercloud, [85](#)
- EDAM\_USER\_UPLOAD\_LIMIT\_FREE
  - qevercloud, [85](#)
- EDAM\_USER\_UPLOAD\_LIMIT\_PLUS
  - qevercloud, [85](#)
- EDAM\_USER\_UPLOAD\_LIMIT\_PREMIUM
  - qevercloud, [86](#)
- EDAM\_USER\_UPLOAD\_SURVEY\_THRESHOLD
  - qevercloud, [86](#)
- EDAM\_USER\_USERNAME\_LEN\_MAX
  - qevercloud, [86](#)
- EDAM\_USER\_USERNAME\_LEN\_MIN
  - qevercloud, [86](#)
- EDAM\_USER\_USERNAME\_REGEX
  - qevercloud, [86](#)
- EDAM\_USER\_WORKSPACES\_MAX
  - qevercloud, [86](#)
- EDAM\_VAT\_REGEX
  - qevercloud, [86](#)
- EDAM\_VERSION\_MAJOR
  - qevercloud, [87](#)
- EDAM\_VERSION\_MINOR
  - qevercloud, [87](#)
- EDAM\_WORKSPACE\_DESCRIPTION\_LEN\_MAX
  - qevercloud, [87](#)
- EDAM\_WORKSPACE\_NAME\_LEN\_MAX
  - qevercloud, [87](#)
- EDAM\_WORKSPACE\_NAME\_LEN\_MIN
  - qevercloud, [87](#)
- EDAM\_WORKSPACE\_NAME\_REGEX
  - qevercloud, [87](#)
- EDAMErrorCode
  - qevercloud, [34](#)
- EDAMErrorCode.h, [536](#), [539](#)
- EDAMInvalidContactReason
  - qevercloud, [35](#)
- EDAMInvalidContactsException
  - qevercloud::EDAMInvalidContactsException, [129](#)
- EDAMInvalidContactsExceptionData
  - qevercloud::EDAMInvalidContactsExceptionData, [132](#)
- EDAMNotFoundException
  - qevercloud::EDAMNotFoundException, [134](#)
- EDAMNotFoundExceptionData
  - qevercloud::EDAMNotFoundExceptionData, [136](#)
- EDAMSystemException
  - qevercloud::EDAMSystemException, [138](#)
- EDAMSystemExceptionAuthExpiredData
  - qevercloud::EDAMSystemExceptionAuthExpiredData, [141](#)
- EDAMSystemExceptionData
  - qevercloud::EDAMSystemExceptionData, [143](#)
- EDAMSystemExceptionRateLimitReachedData
  - qevercloud::EDAMSystemExceptionRateLimitReachedData, [145](#)
- EDAMUserException
  - qevercloud::EDAMUserException, [147](#)
- EDAMUserExceptionData
  - qevercloud::EDAMUserExceptionData, [149](#)
- educationalDiscount
  - qevercloud::UserAttributes, [493](#)
- EMAIL
  - qevercloud, [33](#), [44](#)
- email
  - qevercloud::BusinessInvitation, [112](#)
  - qevercloud::BusinessUserInfo, [119](#)
  - qevercloud::SharedNotebook, [448](#)
  - qevercloud::User, [488](#)
  - qevercloud::UserProfile, [501](#)
- emailAddressLastConfirmed
  - qevercloud::UserAttributes, [493](#)
- emailNote
  - qevercloud::INoteStore, [191](#)
- emailNoteAsync
  - qevercloud::INoteStore, [192](#)
- emailNoteRequest
  - qevercloud::NoteStoreServer, [363](#)
- emailNoteRequestReady
  - qevercloud::NoteStoreServer, [363](#)
- emailOptOutDate
  - qevercloud::UserAttributes, [494](#)
- emphasized
  - qevercloud::NoteFilter, [324](#)
- enableFacebookSharing
  - qevercloud::BootstrapSettings, [109](#)
- enableGiftSubscriptions
  - qevercloud::BootstrapSettings, [109](#)
- enableGoogle
  - qevercloud::BootstrapSettings, [109](#)
- enableLinkedInSharing
  - qevercloud::BootstrapSettings, [109](#)
- enablePublicNotebooks
  - qevercloud::BootstrapSettings, [110](#)
- enableSharedNotebooks
  - qevercloud::BootstrapSettings, [110](#)
- enableSingleNoteSharing
  - qevercloud::BootstrapSettings, [110](#)
- enableSponsoredAccounts
  - qevercloud::BootstrapSettings, [110](#)

- enableSupportTickets
  - qevercloud::BootstrapSettings, 110
- enableTwitterSharing
  - qevercloud::BootstrapSettings, 110
- end
  - qevercloud::QAssociativeContainerConstReferenceWrapper
    - Container >, 416
  - qevercloud::QAssociativeContainerReferenceWrapper
    - Container >, 417
- ENML\_VALIDATION
  - qevercloud, 35
- EntityType
  - qevercloud, 36
- Error
  - qevercloud, 36
- errorCode
  - qevercloud::EDAMSystemException, 139
  - qevercloud::EDAMUserException, 148
- errorMessage
  - qevercloud::EverCloudExceptionData, 155
- errors
  - qevercloud::ManageNotebookSharesResult, 274
  - qevercloud::ManageNoteSharesResult, 281
- eventId
  - qevercloud::Identity, 169
- EventLoopFinisher
  - qevercloud::EventLoopFinisher, 151
- EventLoopFinisher.h, 518
- EverCloudException
  - qevercloud::EverCloudException, 152
- EverCloudException.h, 519, 520
- EverCloudExceptionData
  - qevercloud, 87
  - qevercloud::EverCloudExceptionData, 155
- EverCloudExceptionDataPtr
  - qevercloud, 29
- EverCloudLocalData
  - qevercloud::EverCloudLocalData, 157
- EVERNOTE
  - qevercloud, 33
- EVERNOTE\_USERID
  - qevercloud, 44
- EvernoteException
  - qevercloud::EvernoteException, 159, 160
- EvernoteExceptionData
  - qevercloud::EvernoteExceptionData, 161
- evernoteNetworkProxy
  - qevercloud, 44
- EvernoteOAuthDialog
  - qevercloud::EvernoteOAuthDialog, 162
- EvernoteOAuthWebView
  - qevercloud::EvernoteOAuthWebView, 165
- exceptionData
  - qevercloud::EDAMInvalidContactsException, 129
  - qevercloud::EDAMNotFoundException, 134
  - qevercloud::EDAMSystemException, 138
  - qevercloud::EDAMSystemExceptionAuthExpired, 140
  - qevercloud::EDAMSystemExceptionRateLimitReached, 144
  - qevercloud::EDAMUserException, 147
  - qevercloud::EverCloudException, 153
  - qevercloud::EvernoteException, 160
  - qevercloud::NetworkException, 285
  - qevercloud::ThriftException, 477
- Exceptions.h, 521
- exec
  - qevercloud::EvernoteOAuthDialog, 163
- executeAsyncRequest
  - qevercloud::IDurableService, 171
- executeSyncRequest
  - qevercloud::IDurableService, 171
- experts
  - qevercloud::RelatedResult, 428, 429
- expiration
  - qevercloud::AuthenticationResult, 103
- expires
  - qevercloud::EvernoteOAuthWebView::OAuthResult, 400
- Export.h, 524
  - QEVERCLOUD\_EXPORT, 524
- expungedLinkedNotebooks
  - qevercloud::SyncChunk, 461, 463
- expungedNotebooks
  - qevercloud::SyncChunk, 461, 463
- expungedNotes
  - qevercloud::SyncChunk, 462, 464
- expungedSearches
  - qevercloud::SyncChunk, 462, 464
- expungedTags
  - qevercloud::SyncChunk, 462, 464
- expungeLinkedNotebook
  - qevercloud::INoteStore, 192
- expungeLinkedNotebookAsync
  - qevercloud::INoteStore, 192
- expungeLinkedNotebookRequest
  - qevercloud::NoteStoreServer, 363
- expungeLinkedNotebookRequestReady
  - qevercloud::NoteStoreServer, 363
- expungeNote
  - qevercloud::INoteStore, 193
- expungeNoteAsync
  - qevercloud::INoteStore, 193
- expungeNotebook
  - qevercloud::INoteStore, 193
- expungeNotebookAsync
  - qevercloud::INoteStore, 194
- expungeNotebookRequest
  - qevercloud::NoteStoreServer, 363
- expungeNotebookRequestReady
  - qevercloud::NoteStoreServer, 364
- expungeNoteRequest
  - qevercloud::NoteStoreServer, 364
- expungeNoteRequestReady
  - qevercloud::NoteStoreServer, 364
- expungeSearch

- qevercloud::INoteStore, 194
- expungeSearchAsync
  - qevercloud::INoteStore, 195
- expungeSearchRequest
  - qevercloud::NoteStoreServer, 364
- expungeSearchRequestReady
  - qevercloud::NoteStoreServer, 364
- expungeTag
  - qevercloud::INoteStore, 195
- expungeTagAsync
  - qevercloud::INoteStore, 196
- expungeTagRequest
  - qevercloud::NoteStoreServer, 364
- expungeTagRequestReady
  - qevercloud::NoteStoreServer, 365
- expungeWhichSharedNotebookRestrictions
  - qevercloud::NotebookRestrictions, 310
- FACEBOOK
  - qevercloud, 33
- FAILED
  - qevercloud, 37
- favorited
  - qevercloud::EverCloudLocalData, 158
- fileName
  - qevercloud::ResourceAttributes, 439
- fileSize
  - qevercloud::RelatedContentImage, 422
- filter
  - qevercloud::RelatedQuery, 425
- findNoteCounts
  - qevercloud::INoteStore, 196
- findNoteCountsAsync
  - qevercloud::INoteStore, 197
- findNoteCountsRequest
  - qevercloud::NoteStoreServer, 365
- findNoteCountsRequestReady
  - qevercloud::NoteStoreServer, 365
- findNoteOffset
  - qevercloud::INoteStore, 197
- findNoteOffsetAsync
  - qevercloud::INoteStore, 198
- findNoteOffsetRequest
  - qevercloud::NoteStoreServer, 365
- findNoteOffsetRequestReady
  - qevercloud::NoteStoreServer, 365
- findNotesMetadata
  - qevercloud::INoteStore, 198
- findNotesMetadataAsync
  - qevercloud::INoteStore, 199
- findNotesMetadataRequest
  - qevercloud::NoteStoreServer, 365
- findNotesMetadataRequestReady
  - qevercloud::NoteStoreServer, 366
- findRelated
  - qevercloud::INoteStore, 199
- findRelatedAsync
  - qevercloud::INoteStore, 200
- findRelatedRequest
  - qevercloud::NoteStoreServer, 366
- findRelatedRequestReady
  - qevercloud::NoteStoreServer, 366
- finished
  - qevercloud::AsyncResult, 100
- format
  - qevercloud::SavedSearch, 441
- fromWorkChat
  - qevercloud::BusinessInvitation, 113
- FULL\_ACCESS
  - qevercloud, 41, 43
- FullMap
  - qevercloud::LazyMap, 263
- fullMap
  - qevercloud::LazyMap, 263, 264
- fullSyncBefore
  - qevercloud::SyncState, 471
- getAccountLimits
  - qevercloud::IUserStore, 254
- getAccountLimitsAsync
  - qevercloud::IUserStore, 255
- getAccountLimitsRequest
  - qevercloud::UserStoreServer, 506
- getAccountLimitsRequestReady
  - qevercloud::UserStoreServer, 506
- getBootstrapInfo
  - qevercloud::IUserStore, 255
- getBootstrapInfoAsync
  - qevercloud::IUserStore, 255
- getBootstrapInfoRequest
  - qevercloud::UserStoreServer, 506
- getBootstrapInfoRequestReady
  - qevercloud::UserStoreServer, 506
- getDefaultNotebook
  - qevercloud::INoteStore, 200
- getDefaultNotebookAsync
  - qevercloud::INoteStore, 201
- getDefaultNotebookRequest
  - qevercloud::NoteStoreServer, 366
- getDefaultNotebookRequestReady
  - qevercloud::NoteStoreServer, 366
- getFilteredSyncChunk
  - qevercloud::INoteStore, 201
- getFilteredSyncChunkAsync
  - qevercloud::INoteStore, 201
- getFilteredSyncChunkRequest
  - qevercloud::NoteStoreServer, 366
- getFilteredSyncChunkRequestReady
  - qevercloud::NoteStoreServer, 367
- getLinkedNotebookSyncChunk
  - qevercloud::INoteStore, 202
- getLinkedNotebookSyncChunkAsync
  - qevercloud::INoteStore, 203
- getLinkedNotebookSyncChunkRequest
  - qevercloud::NoteStoreServer, 367
- getLinkedNotebookSyncChunkRequestReady
  - qevercloud::NoteStoreServer, 367
- getLinkedNotebookSyncState



qevercloud::INoteStore, 203  
getLinkedNotebookSyncStateAsync  
    qevercloud::INoteStore, 204  
getLinkedNotebookSyncStateRequest  
    qevercloud::NoteStoreServer, 367  
getLinkedNotebookSyncStateRequestReady  
    qevercloud::NoteStoreServer, 367  
getNote  
    qevercloud::INoteStore, 204  
getNoteApplicationData  
    qevercloud::INoteStore, 204  
getNoteApplicationDataAsync  
    qevercloud::INoteStore, 204  
getNoteApplicationDataEntry  
    qevercloud::INoteStore, 205  
getNoteApplicationDataEntryAsync  
    qevercloud::INoteStore, 205  
getNoteApplicationDataEntryRequest  
    qevercloud::NoteStoreServer, 367  
getNoteApplicationDataEntryRequestReady  
    qevercloud::NoteStoreServer, 368  
getNoteApplicationDataRequest  
    qevercloud::NoteStoreServer, 368  
getNoteApplicationDataRequestReady  
    qevercloud::NoteStoreServer, 368  
getNoteAsync  
    qevercloud::INoteStore, 205  
getNotebook  
    qevercloud::INoteStore, 205  
getNotebookAsync  
    qevercloud::INoteStore, 207  
getNotebookRequest  
    qevercloud::NoteStoreServer, 368  
getNotebookRequestReady  
    qevercloud::NoteStoreServer, 368  
getNotebookShares  
    qevercloud::INoteStore, 207  
getNotebookSharesAsync  
    qevercloud::INoteStore, 207  
getNotebookSharesRequest  
    qevercloud::NoteStoreServer, 368  
getNotebookSharesRequestReady  
    qevercloud::NoteStoreServer, 369  
getNoteContent  
    qevercloud::INoteStore, 207  
getNoteContentAsync  
    qevercloud::INoteStore, 208  
getNoteContentRequest  
    qevercloud::NoteStoreServer, 369  
getNoteContentRequestReady  
    qevercloud::NoteStoreServer, 369  
getNoteRequest  
    qevercloud::NoteStoreServer, 369  
getNoteRequestReady  
    qevercloud::NoteStoreServer, 369  
getNoteSearchText  
    qevercloud::INoteStore, 208  
getNoteSearchTextAsync  
    qevercloud::INoteStore, 209  
getNoteSearchTextRequest  
    qevercloud::NoteStoreServer, 369  
getNoteSearchTextRequestReady  
    qevercloud::NoteStoreServer, 370  
getNoteTagNames  
    qevercloud::INoteStore, 209  
getNoteTagNamesAsync  
    qevercloud::INoteStore, 209  
getNoteTagNamesRequest  
    qevercloud::NoteStoreServer, 370  
getNoteTagNamesRequestReady  
    qevercloud::NoteStoreServer, 370  
getNoteVersion  
    qevercloud::INoteStore, 210  
getNoteVersionAsync  
    qevercloud::INoteStore, 210  
getNoteVersionRequest  
    qevercloud::NoteStoreServer, 370  
getNoteVersionRequestReady  
    qevercloud::NoteStoreServer, 370  
getNoteWithResultSpec  
    qevercloud::INoteStore, 211  
getNoteWithResultSpecAsync  
    qevercloud::INoteStore, 211  
getNoteWithResultSpecRequest  
    qevercloud::NoteStoreServer, 370  
getNoteWithResultSpecRequestReady  
    qevercloud::NoteStoreServer, 371  
getPublicNotebook  
    qevercloud::INoteStore, 211  
getPublicNotebookAsync  
    qevercloud::INoteStore, 212  
getPublicNotebookRequest  
    qevercloud::NoteStoreServer, 371  
getPublicNotebookRequestReady  
    qevercloud::NoteStoreServer, 371  
getPublicUserInfo  
    qevercloud::IUserStore, 255  
getPublicUserInfoAsync  
    qevercloud::IUserStore, 256  
getPublicUserInfoRequest  
    qevercloud::UserStoreServer, 506  
getPublicUserInfoRequestReady  
    qevercloud::UserStoreServer, 506  
getResource  
    qevercloud::INoteStore, 212  
getResourceAlternateData  
    qevercloud::INoteStore, 213  
getResourceAlternateDataAsync  
    qevercloud::INoteStore, 213  
getResourceAlternateDataRequest  
    qevercloud::NoteStoreServer, 371  
getResourceAlternateDataRequestReady  
    qevercloud::NoteStoreServer, 371  
getResourceApplicationData  
    qevercloud::INoteStore, 214  
getResourceApplicationDataAsync

- qevercloud::INoteStore, 214
- getResourceApplicationDataEntry
  - qevercloud::INoteStore, 214
- getResourceApplicationDataEntryAsync
  - qevercloud::INoteStore, 214
- getResourceApplicationDataEntryRequest
  - qevercloud::NoteStoreServer, 371
- getResourceApplicationDataEntryRequestReady
  - qevercloud::NoteStoreServer, 372
- getResourceApplicationDataRequest
  - qevercloud::NoteStoreServer, 372
- getResourceApplicationDataRequestReady
  - qevercloud::NoteStoreServer, 372
- getResourceAsync
  - qevercloud::INoteStore, 215
- getResourceAttributes
  - qevercloud::INoteStore, 215
- getResourceAttributesAsync
  - qevercloud::INoteStore, 215
- getResourceAttributesRequest
  - qevercloud::NoteStoreServer, 372
- getResourceAttributesRequestReady
  - qevercloud::NoteStoreServer, 372
- getResourceByHash
  - qevercloud::INoteStore, 215
- getResourceByHashAsync
  - qevercloud::INoteStore, 216
- getResourceByHashRequest
  - qevercloud::NoteStoreServer, 372
- getResourceByHashRequestReady
  - qevercloud::NoteStoreServer, 373
- getResourceData
  - qevercloud::INoteStore, 216
- getResourceDataAsync
  - qevercloud::INoteStore, 217
- getResourceDataRequest
  - qevercloud::NoteStoreServer, 373
- getResourceDataRequestReady
  - qevercloud::NoteStoreServer, 373
- getResourceRecognition
  - qevercloud::INoteStore, 217
- getResourceRecognitionAsync
  - qevercloud::INoteStore, 218
- getResourceRecognitionRequest
  - qevercloud::NoteStoreServer, 373
- getResourceRecognitionRequestReady
  - qevercloud::NoteStoreServer, 373
- getResourceRequest
  - qevercloud::NoteStoreServer, 373
- getResourceRequestReady
  - qevercloud::NoteStoreServer, 374
- getResourceSearchText
  - qevercloud::INoteStore, 218
- getResourceSearchTextAsync
  - qevercloud::INoteStore, 219
- getResourceSearchTextRequest
  - qevercloud::NoteStoreServer, 374
- getResourceSearchTextRequestReady
  - qevercloud::NoteStoreServer, 374
- getSearch
  - qevercloud::INoteStore, 219
- getSearchAsync
  - qevercloud::INoteStore, 219
- getSearchRequest
  - qevercloud::NoteStoreServer, 374
- getSearchRequestReady
  - qevercloud::NoteStoreServer, 374
- getSharedNotebookByAuth
  - qevercloud::INoteStore, 219
- getSharedNotebookByAuthAsync
  - qevercloud::INoteStore, 220
- getSharedNotebookByAuthRequest
  - qevercloud::NoteStoreServer, 374
- getSharedNotebookByAuthRequestReady
  - qevercloud::NoteStoreServer, 375
- getSyncState
  - qevercloud::INoteStore, 220
- getSyncStateAsync
  - qevercloud::INoteStore, 220
- getSyncStateRequest
  - qevercloud::NoteStoreServer, 375
- getSyncStateRequestReady
  - qevercloud::NoteStoreServer, 375
- getTag
  - qevercloud::INoteStore, 220
- getTagAsync
  - qevercloud::INoteStore, 221
- getTagRequest
  - qevercloud::NoteStoreServer, 375
- getTagRequestReady
  - qevercloud::NoteStoreServer, 375
- getUser
  - qevercloud::IUserStore, 256
- getUserAsync
  - qevercloud::IUserStore, 256
- getUserRequest
  - qevercloud::UserStoreServer, 507
- getUserRequestReady
  - qevercloud::UserStoreServer, 507
- getUserUrls
  - qevercloud::IUserStore, 256
- getUserUrlsAsync
  - qevercloud::IUserStore, 256
- getUserUrlsRequest
  - qevercloud::UserStoreServer, 507
- getUserUrlsRequestReady
  - qevercloud::UserStoreServer, 507
- GIF
  - qevercloud::Thumbnail, 481
- globalId
  - qevercloud::SharedNotebook, 448
- Globals.h, 617
- GROUP
  - qevercloud, 41
- GROUP\_ADMIN
  - qevercloud, 44



- GROUP\_MEMBER
  - qevercloud, [44](#)
- GROUP\_OWNER
  - qevercloud, [44](#)
- groupName
  - qevercloud::UserAttributes, [494](#)
- Guid
  - qevercloud, [29](#)
- guid
  - qevercloud::LinkedNotebook, [266](#)
  - qevercloud::Note, [290](#)
  - qevercloud::Notebook, [301](#)
  - qevercloud::NotebookDescriptor, [305](#)
  - qevercloud::NoteEmailParameters, [321](#)
  - qevercloud::NoteMetadata, [337](#)
  - qevercloud::Resource, [435](#)
  - qevercloud::SavedSearch, [442](#)
  - qevercloud::Tag, [474](#)
- hasSharedNotebook
  - qevercloud::NotebookDescriptor, [305](#)
- height
  - qevercloud::RelatedContentImage, [423](#)
  - qevercloud::Resource, [436](#)
- Helpers.h, [618](#)
- hideSponsorBilling
  - qevercloud::UserAttributes, [494](#)
- id
  - qevercloud::Contact, [123](#)
  - qevercloud::EverCloudLocalData, [158](#)
  - qevercloud::Identity, [169](#)
  - qevercloud::SharedNotebook, [448](#)
  - qevercloud::User, [489](#)
  - qevercloud::UserProfile, [501](#)
- identifier
  - qevercloud::EDAMNotFoundException, [135](#)
- IDENTITYID
  - qevercloud, [44](#)
- IdentityID
  - qevercloud, [30](#)
- identityID
  - qevercloud::ManageNoteSharesError, [275](#)
- IDurableServicePtr
  - qevercloud, [30](#)
- ILoggerPtr
  - qevercloud, [30](#)
- ImageType
  - qevercloud::Thumbnail, [480](#)
- IN\_MY\_LIST
  - qevercloud, [38](#)
- IN\_MY\_LIST\_AND\_DEFAULT\_NOTEBOOK
  - qevercloud, [38](#)
- inactive
  - qevercloud::NoteFilter, [324](#)
- includeAccount
  - qevercloud::SavedSearchScope, [444](#)
- includeAccountLimits
  - qevercloud::NoteResultSpec, [342](#)
- includeAllReadableNotebooks
  - qevercloud::NoteFilter, [324](#)
- includeAllReadableWorkspaces
  - qevercloud::NoteFilter, [324](#)
- includeAttributes
  - qevercloud::NotesMetadataResultSpec, [352](#)
- includeBusinessLinkedNotebooks
  - qevercloud::SavedSearchScope, [444](#)
- includeContainingNotebooks
  - qevercloud::RelatedResultSpec, [432](#)
- includeContent
  - qevercloud::NoteResultSpec, [342](#)
- includeContentLength
  - qevercloud::NotesMetadataResultSpec, [352](#)
- includeCreated
  - qevercloud::NotesMetadataResultSpec, [352](#)
- includeDebugInfo
  - qevercloud::RelatedResultSpec, [432](#)
- includeDeleted
  - qevercloud::NotesMetadataResultSpec, [352](#)
- includeExpunged
  - qevercloud::SyncChunkFilter, [466](#)
- includeLargestResourceMime
  - qevercloud::NotesMetadataResultSpec, [353](#)
- includeLargestResourceSize
  - qevercloud::NotesMetadataResultSpec, [353](#)
- includeLinkedNotebooks
  - qevercloud::SyncChunkFilter, [466](#)
- includeNoteAppDataValues
  - qevercloud::NoteResultSpec, [343](#)
- includeNoteApplicationDataFullMap
  - qevercloud::SyncChunkFilter, [467](#)
- includeNoteAttributes
  - qevercloud::SyncChunkFilter, [467](#)
- includeNotebookGuid
  - qevercloud::NotesMetadataResultSpec, [353](#)
- includeNotebooks
  - qevercloud::SyncChunkFilter, [467](#)
- includeNoteResourceApplicationDataFullMap
  - qevercloud::SyncChunkFilter, [467](#)
- includeNoteResources
  - qevercloud::SyncChunkFilter, [467](#)
- includeNotes
  - qevercloud::SyncChunkFilter, [467](#)
- includePersonalLinkedNotebooks
  - qevercloud::SavedSearchScope, [444](#)
- includeResourceAppDataValues
  - qevercloud::NoteResultSpec, [343](#)
- includeResourceApplicationDataFullMap
  - qevercloud::SyncChunkFilter, [467](#)
- includeResources
  - qevercloud::SyncChunkFilter, [468](#)
- includeResourcesAlternateData
  - qevercloud::NoteResultSpec, [343](#)
- includeResourcesData
  - qevercloud::NoteResultSpec, [343](#)
- includeResourcesRecognition
  - qevercloud::NoteResultSpec, [343](#)

- includeSearches
  - qevercloud::SyncChunkFilter, 468
- includeSharedNotes
  - qevercloud::NoteResultSpec, 343
  - qevercloud::SyncChunkFilter, 468
- includeTagGuids
  - qevercloud::NotesMetadataResultSpec, 353
- includeTags
  - qevercloud::SyncChunkFilter, 468
- includeTitle
  - qevercloud::NotesMetadataResultSpec, 353
- includeUpdated
  - qevercloud::NotesMetadataResultSpec, 353
- includeUpdateSequenceNum
  - qevercloud::NotesMetadataResultSpec, 353
- incomingEmailAddress
  - qevercloud::UserAttributes, 494
- increaseRequestTimeoutExponentially
  - qevercloud::IRequestContext, 243
- individualPrivilege
  - qevercloud::MemberShareRelationship, 283
- Info
  - qevercloud, 36
- init
  - qevercloud::Optional< T >, 404
- initializeQEverCloud
  - qevercloud, 44
- InkNoteImageDownloader
  - qevercloud::InkNoteImageDownloader, 173
- InkNoteImageDownloader.h, 620
- inMyList
  - qevercloud::NotebookRecipientSettings, 307
- INoteStore
  - qevercloud::INoteStore, 180
- INoteStorePtr
  - qevercloud, 30
- INSUFFICIENT\_CONTAINER\_PRIVILEGE
  - qevercloud, 33
- INSUFFICIENT\_ENTITY\_PRIVILEGE
  - qevercloud, 33
- INTERNAL\_ERROR
  - qevercloud, 35
  - qevercloud::ThriftException, 476
- INVALID\_AUTH
  - qevercloud, 35
- INVALID\_DATA
  - qevercloud::ThriftException, 476
- INVALID\_MESSAGE\_TYPE
  - qevercloud::ThriftException, 476
- INVALID\_OPENID\_TOKEN
  - qevercloud, 35
- InvalidationSequenceNumber
  - qevercloud, 30
- invitationRestrictions
  - qevercloud::NoteShareRelationships, 347
  - qevercloud::ShareRelationships, 458
- invitations
  - qevercloud::NoteShareRelationships, 347, 348
  - qevercloud::ShareRelationships, 458, 459
- invitationsToCreateOrUpdate
  - qevercloud::ManageNotebookSharesParameters, 271, 272
- invitationsToUnshare
  - qevercloud::ManageNoteSharesParameters, 278, 279
- invitationsToUpdate
  - qevercloud::ManageNoteSharesParameters, 278, 279
- inviteMessage
  - qevercloud::ManageNotebookSharesParameters, 271
- inviteToBusiness
  - qevercloud::IUserStore, 256
- inviteToBusinessAsync
  - qevercloud::IUserStore, 257
- inviteToBusinessRequest
  - qevercloud::UserStoreServer, 507
- inviteToBusinessRequestReady
  - qevercloud::UserStoreServer, 507
- IRequestContextPtr
  - qevercloud, 30
- IRetryPolicyPtr
  - qevercloud, 30
- isEqual
  - qevercloud::Optional< T >, 404
- isSet
  - qevercloud::Optional< T >, 404
- isSucceeded
  - qevercloud::EvernoteOAuthDialog, 163
  - qevercloud::EvernoteOAuthWebView, 166
- iterator
  - qevercloud::QAssociativeContainerConstReferenceWrapper< Container >::iterator, 245
  - qevercloud::QAssociativeContainerReferenceWrapper< Container >::iterator, 247
- IUserStore
  - qevercloud::IUserStore, 249
- IUserStorePtr
  - qevercloud, 31
- joined
  - qevercloud::UserProfile, 501
- joinedUserCount
  - qevercloud::NotebookDescriptor, 305
- JPEG
  - qevercloud::Thumbnail, 481
- key
  - qevercloud::EDAMNotFoundException, 135
- keysOnly
  - qevercloud::LazyMap, 263, 264
- largestResourceMime
  - qevercloud::NoteMetadata, 337
- largestResourceSize
  - qevercloud::NoteMetadata, 337
- lastEditedBy

- qevercloud::NoteAttributes, 296
- lastEditorId
  - qevercloud::NoteAttributes, 296
  - qevercloud::NoteVersionId, 397
- lastFailedCharge
  - qevercloud::Accounting, 91
- lastFailedChargeReason
  - qevercloud::Accounting, 91
- lastRequestedCharge
  - qevercloud::Accounting, 91
- lastSuccessfulCharge
  - qevercloud::Accounting, 91
- latitude
  - qevercloud::NoteAttributes, 296
  - qevercloud::ResourceAttributes, 439
- LEN\_TOO\_LONG
  - qevercloud, 35
- LEN\_TOO\_SHORT
  - qevercloud, 35
- level
  - qevercloud::ILogger, 171
- libraryVersion
  - qevercloud, 44
- LIMIT\_REACHED
  - qevercloud, 35
- limits
  - qevercloud::Note, 290
- LINKEDIN
  - qevercloud, 33
- linkedInProfileUrl
  - qevercloud::BusinessUserAttributes, 117
- linkedNotebooks
  - qevercloud::SyncChunk, 462, 464
- listAccessibleBusinessNotebooks
  - qevercloud::INoteStore, 221
- listAccessibleBusinessNotebooksAsync
  - qevercloud::INoteStore, 221
- listAccessibleBusinessNotebooksRequest
  - qevercloud::NoteStoreServer, 375
- listAccessibleBusinessNotebooksRequestReady
  - qevercloud::NoteStoreServer, 375
- listBusinessInvitations
  - qevercloud::IUserStore, 257
- listBusinessInvitationsAsync
  - qevercloud::IUserStore, 258
- listBusinessInvitationsRequest
  - qevercloud::UserStoreServer, 507
- listBusinessInvitationsRequestReady
  - qevercloud::UserStoreServer, 508
- listBusinessUsers
  - qevercloud::IUserStore, 258
- listBusinessUsersAsync
  - qevercloud::IUserStore, 259
- listBusinessUsersRequest
  - qevercloud::UserStoreServer, 508
- listBusinessUsersRequestReady
  - qevercloud::UserStoreServer, 508
- listLinkedNotebooks
  - qevercloud::INoteStore, 222
- listLinkedNotebooksAsync
  - qevercloud::INoteStore, 222
- listLinkedNotebooksRequest
  - qevercloud::NoteStoreServer, 376
- listLinkedNotebooksRequestReady
  - qevercloud::NoteStoreServer, 376
- listNotebooks
  - qevercloud::INoteStore, 222
- listNotebooksAsync
  - qevercloud::INoteStore, 222
- listNotebooksRequest
  - qevercloud::NoteStoreServer, 376
- listNotebooksRequestReady
  - qevercloud::NoteStoreServer, 376
- listNoteVersions
  - qevercloud::INoteStore, 222
- listNoteVersionsAsync
  - qevercloud::INoteStore, 223
- listNoteVersionsRequest
  - qevercloud::NoteStoreServer, 376
- listNoteVersionsRequestReady
  - qevercloud::NoteStoreServer, 376
- listSearches
  - qevercloud::INoteStore, 223
- listSearchesAsync
  - qevercloud::INoteStore, 223
- listSearchesRequest
  - qevercloud::NoteStoreServer, 376
- listSearchesRequestReady
  - qevercloud::NoteStoreServer, 377
- listSharedNotebooks
  - qevercloud::INoteStore, 223
- listSharedNotebooksAsync
  - qevercloud::INoteStore, 223
- listSharedNotebooksRequest
  - qevercloud::NoteStoreServer, 377
- listSharedNotebooksRequestReady
  - qevercloud::NoteStoreServer, 377
- listTags
  - qevercloud::INoteStore, 224
- listTagsAsync
  - qevercloud::INoteStore, 224
- listTagsByNotebook
  - qevercloud::INoteStore, 224
- listTagsByNotebookAsync
  - qevercloud::INoteStore, 224
- listTagsByNotebookRequest
  - qevercloud::NoteStoreServer, 377
- listTagsByNotebookRequestReady
  - qevercloud::NoteStoreServer, 377
- listTagsRequest
  - qevercloud::NoteStoreServer, 377
- listTagsRequestReady
  - qevercloud::NoteStoreServer, 377
- local
  - qevercloud::EverCloudLocalData, 159
- localData

- qevercloud::Accounting, 92
- qevercloud::AccountLimits, 95
- qevercloud::AuthenticationResult, 103
- qevercloud::BootstrapInfo, 105
- qevercloud::BootstrapProfile, 107
- qevercloud::BootstrapSettings, 110
- qevercloud::BusinessInvitation, 113
- qevercloud::BusinessNotebook, 115
- qevercloud::BusinessUserAttributes, 117
- qevercloud::BusinessUserInfo, 119
- qevercloud::CanMoveToContainerRestrictions, 121
- qevercloud::Contact, 123
- qevercloud::CreateOrUpdateNotebookSharesResult, 125
- qevercloud::Data, 127
- qevercloud::Identity, 169
- qevercloud::InvitationShareRelationship, 242
- qevercloud::LazyMap, 264
- qevercloud::LinkedNotebook, 266
- qevercloud::ManageNotebookSharesError, 269
- qevercloud::ManageNotebookSharesParameters, 271
- qevercloud::ManageNotebookSharesResult, 274
- qevercloud::ManageNoteSharesError, 276
- qevercloud::ManageNoteSharesParameters, 278
- qevercloud::ManageNoteSharesResult, 281
- qevercloud::MemberShareRelationship, 283
- qevercloud::Note, 290
- qevercloud::NoteAttributes, 296
- qevercloud::Notebook, 301
- qevercloud::NotebookDescriptor, 306
- qevercloud::NotebookRecipientSettings, 307
- qevercloud::NotebookRestrictions, 310
- qevercloud::NotebookShareTemplate, 316
- qevercloud::NoteCollectionCounts, 319
- qevercloud::NoteEmailParameters, 321
- qevercloud::NoteFilter, 325
- qevercloud::NoteInvitationShareRelationship, 327
- qevercloud::NoteLimits, 329
- qevercloud::NoteList, 332
- qevercloud::NoteMemberShareRelationship, 334
- qevercloud::NoteMetadata, 337
- qevercloud::NoteRestrictions, 340
- qevercloud::NoteResultSpec, 343
- qevercloud::NoteShareRelationshipRestrictions, 345
- qevercloud::NoteShareRelationships, 347
- qevercloud::NotesMetadataList, 349
- qevercloud::NotesMetadataResultSpec, 353
- qevercloud::NoteVersionId, 398
- qevercloud::PublicUserInfo, 412
- qevercloud::Publishing, 414
- qevercloud::RelatedContent, 420
- qevercloud::RelatedContentImage, 423
- qevercloud::RelatedQuery, 425
- qevercloud::RelatedResult, 428
- qevercloud::RelatedResultSpec, 432
- qevercloud::Resource, 436
- qevercloud::ResourceAttributes, 439
- qevercloud::SavedSearch, 442
- qevercloud::SavedSearchScope, 444
- qevercloud::SharedNote, 446
- qevercloud::SharedNotebook, 449
- qevercloud::SharedNotebookRecipientSettings, 452
- qevercloud::SharedNoteTemplate, 454
- qevercloud::ShareRelationshipRestrictions, 456
- qevercloud::ShareRelationships, 459
- qevercloud::SyncChunk, 462
- qevercloud::SyncChunkFilter, 468
- qevercloud::SyncState, 471
- qevercloud::Tag, 474
- qevercloud::UpdateNotelfUsnMatchesResult, 485
- qevercloud::User, 489
- qevercloud::UserAttributes, 494
- qevercloud::UserIdentity, 499
- qevercloud::UserProfile, 501
- qevercloud::UserUrls, 513
- location
  - qevercloud::BusinessUserAttributes, 117
- log
  - qevercloud::ILogger, 171
- Log.h, 621, 624
  - \_\_QEVERCLOUD\_LOG\_BASE, 622
  - QEC\_DEBUG, 622
  - QEC\_ERROR, 623
  - QEC\_INFO, 623
  - QEC\_TRACE, 623
  - QEC\_WARNING, 623
- logger
  - qevercloud, 45
- LogLevel
  - qevercloud, 36
- longIdentifier
  - qevercloud::UserIdentity, 499
- longitude
  - qevercloud::NoteAttributes, 297
  - qevercloud::ResourceAttributes, 439
- m\_call
  - qevercloud::IDurableService::AsyncRequest, 97
  - qevercloud::IDurableService::SyncRequest, 470
- m\_contacts
  - qevercloud::EDAMInvalidContactsExceptionData, 132
- m\_description
  - qevercloud::IDurableService::AsyncRequest, 98
  - qevercloud::IDurableService::SyncRequest, 470
- m\_error
  - qevercloud::EverCloudException, 153
- m\_errorCode
  - qevercloud::EDAMSystemExceptionData, 143
  - qevercloud::EDAMUserExceptionData, 150
- m\_identifier
  - qevercloud::EDAMNotFoundExceptionData, 136
- m\_iterator

- qevercloud::QAssociativeContainerConstReferenceWrapper
  - Container >::iterator, [246](#)
- qevercloud::QAssociativeContainerReferenceWrapper<
  - Container >::iterator, [247](#)
- m\_key
  - qevercloud::EDAMNotFoundExceptionData, [137](#)
- m\_message
  - qevercloud::EDAMSystemExceptionData, [143](#)
- m\_name
  - qevercloud::IDurableService::AsyncRequest, [98](#)
  - qevercloud::IDurableService::SyncRequest, [470](#)
- m\_parameter
  - qevercloud::EDAMInvalidContactsExceptionData, [132](#)
  - qevercloud::EDAMUserExceptionData, [150](#)
- m\_rateLimitDuration
  - qevercloud::EDAMSystemExceptionData, [144](#)
- m\_reasons
  - qevercloud::EDAMInvalidContactsExceptionData, [132](#)
- m\_type
  - qevercloud::NetworkException, [286](#)
  - qevercloud::NetworkExceptionData, [287](#)
  - qevercloud::ThriftException, [478](#)
  - qevercloud::ThriftExceptionData, [479](#)
- manageNotebookShares
  - qevercloud::INoteStore, [224](#)
- manageNotebookSharesAsync
  - qevercloud::INoteStore, [225](#)
- manageNotebookSharesRequest
  - qevercloud::NoteStoreServer, [378](#)
- manageNotebookSharesRequestReady
  - qevercloud::NoteStoreServer, [378](#)
- MANAGER
  - qevercloud, [38](#)
- marketingUrl
  - qevercloud::BootstrapSettings, [110](#)
- matchingShares
  - qevercloud::CreateOrUpdateNotebookSharesResult, [125](#), [126](#)
- maxExperts
  - qevercloud::RelatedResultSpec, [432](#)
- maxNotebooks
  - qevercloud::RelatedResultSpec, [432](#)
- maxNotes
  - qevercloud::RelatedResultSpec, [432](#)
- maxReferrals
  - qevercloud::UserAttributes, [494](#)
- maxRelatedContent
  - qevercloud::RelatedResultSpec, [432](#)
- maxRequestRetryCount
  - qevercloud::IRequestContext, [243](#)
- maxRequestTimeout
  - qevercloud::IRequestContext, [244](#)
- maxTags
  - qevercloud::RelatedResultSpec, [433](#)
- memberships
  - qevercloud::NoteShareRelationships, [347](#), [348](#)
- qevercloud::ShareRelationships, [459](#)
  - membershipsToUnshare
    - qevercloud::ManageNoteSharesParameters, [278](#), [279](#)
  - membershipsToUpdate
    - qevercloud::ManageNotebookSharesParameters, [271](#), [272](#)
    - qevercloud::ManageNoteSharesParameters, [278](#), [279](#)
- message
  - qevercloud::EDAMSystemException, [139](#)
  - qevercloud::NoteEmailParameters, [321](#)
- MessageEventID
  - qevercloud, [31](#)
- messageStoreUrl
  - qevercloud::UserUrls, [513](#)
- MessageThreadID
  - qevercloud, [31](#)
- messagingPermit
  - qevercloud::Contact, [123](#)
- messagingPermitExpires
  - qevercloud::Contact, [123](#)
- mime
  - qevercloud::Resource, [436](#)
- MISSING\_RESULT
  - qevercloud::ThriftException, [476](#)
- mobilePhone
  - qevercloud::BusinessUserAttributes, [117](#)
- MODIFY\_NOTE
  - qevercloud, [43](#)
- MODIFY\_NOTEBOOK\_PLUS\_ACTIVITY
  - qevercloud, [41](#), [43](#)
- mostRecentReminder
  - qevercloud::BusinessInvitation, [113](#)
- name
  - qevercloud::BootstrapProfile, [107](#)
  - qevercloud::Contact, [123](#)
  - qevercloud::Notebook, [302](#)
  - qevercloud::SavedSearch, [442](#)
  - qevercloud::Tag, [474](#)
  - qevercloud::User, [489](#)
  - qevercloud::UserProfile, [501](#)
- NetworkException
  - qevercloud::NetworkException, [284](#), [285](#)
- NetworkExceptionData
  - qevercloud::NetworkExceptionData, [287](#)
- newDurableService
  - qevercloud, [45](#)
- newNoteStore
  - qevercloud, [45](#)
- newRequestContext
  - qevercloud, [45](#)
- newRetryPolicy
  - qevercloud, [45](#)
- NEWS\_ARTICLE
  - qevercloud, [39](#)
- newStdErrLogger
  - qevercloud, [45](#)

- newUserStore
  - qevercloud, [46](#)
- nextChargeDate
  - qevercloud::Accounting, [92](#)
- nextPaymentDue
  - qevercloud::Accounting, [92](#)
- NO\_CONNECTION
  - qevercloud, [36](#)
- NO\_SHARED\_NOTEBOOKS
  - qevercloud, [40](#)
- noCanMoveNote
  - qevercloud::NotebookRestrictions, [311](#)
- noChangeContact
  - qevercloud::NotebookRestrictions, [311](#)
- noCreateNotes
  - qevercloud::NotebookRestrictions, [311](#)
- noCreateSharedNotebooks
  - qevercloud::NotebookRestrictions, [311](#)
- noCreateTags
  - qevercloud::NotebookRestrictions, [311](#)
- noEmail
  - qevercloud::NoteRestrictions, [340](#)
- noEmailNotes
  - qevercloud::NotebookRestrictions, [311](#)
- noExpungeNotebook
  - qevercloud::NotebookRestrictions, [311](#)
- noExpungeNotes
  - qevercloud::NotebookRestrictions, [312](#)
- noExpungeTags
  - qevercloud::NotebookRestrictions, [312](#)
- NONE
  - qevercloud, [37](#)
- noPublishToBusinessLibrary
  - qevercloud::NotebookRestrictions, [312](#)
- noPublishToPublic
  - qevercloud::NotebookRestrictions, [312](#)
- noReadNotes
  - qevercloud::NotebookRestrictions, [312](#)
- noRenameNotebook
  - qevercloud::NotebookRestrictions, [312](#)
- NORMAL
  - qevercloud, [32](#), [38](#)
- noSendMessageToRecipients
  - qevercloud::NotebookRestrictions, [312](#)
- noSetDefaultNotebook
  - qevercloud::NotebookRestrictions, [313](#)
- noSetFullAccess
  - qevercloud::NoteShareRelationshipRestrictions, [345](#)
  - qevercloud::ShareRelationshipRestrictions, [456](#)
- noSetInMyList
  - qevercloud::NotebookRestrictions, [313](#)
- noSetModify
  - qevercloud::ShareRelationshipRestrictions, [456](#)
- noSetModifyNote
  - qevercloud::NoteShareRelationshipRestrictions, [345](#)
- noSetNotebookStack
  - qevercloud::NotebookRestrictions, [313](#)
- noSetParentTag
  - qevercloud::NotebookRestrictions, [313](#)
- noSetReadNote
  - qevercloud::NoteShareRelationshipRestrictions, [345](#)
- noSetReadOnly
  - qevercloud::ShareRelationshipRestrictions, [457](#)
- noSetReadPlusActivity
  - qevercloud::ShareRelationshipRestrictions, [457](#)
- noSetRecipientSettingsStack
  - qevercloud::NotebookRestrictions, [313](#)
- noSetReminderNotifyEmail
  - qevercloud::NotebookRestrictions, [313](#)
- noSetReminderNotifyInApp
  - qevercloud::NotebookRestrictions, [313](#)
- noShare
  - qevercloud::NoteRestrictions, [340](#)
- noShareNotes
  - qevercloud::NotebookRestrictions, [314](#)
- noShareNotesWithBusiness
  - qevercloud::NotebookRestrictions, [314](#)
- noSharePublicly
  - qevercloud::NoteRestrictions, [340](#)
- NOT\_ACCESSIBLE
  - qevercloud, [39](#)
- NOT\_IN\_MY\_LIST
  - qevercloud, [38](#)
- NOTE
  - qevercloud, [36](#)
- note
  - qevercloud::NoteEmailParameters, [321](#)
  - qevercloud::UpdateNoteIfUsnMatchesResult, [486](#)
- NOTEBOOK
  - qevercloud, [36](#)
- notebookCounts
  - qevercloud::NoteCollectionCounts, [319](#)
- notebookDescription
  - qevercloud::BusinessNotebook, [115](#)
- notebookDisplayName
  - qevercloud::NotebookDescriptor, [306](#)
- notebookGuid
  - qevercloud::ManageNotebookSharesParameters, [271](#)
  - qevercloud::Note, [290](#)
  - qevercloud::NotebookShareTemplate, [316](#)
  - qevercloud::NoteFilter, [325](#)
  - qevercloud::NoteMetadata, [337](#)
  - qevercloud::SharedNotebook, [449](#)
- notebookGuids
  - qevercloud::SyncChunkFilter, [468](#), [469](#)
- notebookModifiable
  - qevercloud::SharedNotebook, [449](#)
- notebooks
  - qevercloud::RelatedResult, [429](#)
  - qevercloud::SyncChunk, [462](#), [464](#)
- noteGuid
  - qevercloud::ManageNoteSharesParameters, [278](#)



- qevercloud::RelatedQuery, 425
- qevercloud::Resource, 436
- qevercloud::SharedNoteTemplate, 454
- noteResourceCountMax
  - qevercloud::AccountLimits, 95
  - qevercloud::NoteLimits, 329
- notes
  - qevercloud::NoteList, 332
  - qevercloud::NotesMetadataList, 349
  - qevercloud::RelatedResult, 429, 430
  - qevercloud::SyncChunk, 462, 464
- noteSizeMax
  - qevercloud::AccountLimits, 95
  - qevercloud::NoteLimits, 330
- NoteSortOrder
  - qevercloud, 37
- NoteStoreServer
  - qevercloud::NoteStoreServer, 359
- noteStoreUrl
  - qevercloud::AuthenticationResult, 103
  - qevercloud::EvernoteOAuthWebView::OAuthResult, 400
  - qevercloud::INoteStore, 225
  - qevercloud::LinkedNotebook, 266
  - qevercloud::PublicUserInfo, 412
  - qevercloud::UserUrls, 513
- noteTagCountMax
  - qevercloud::AccountLimits, 95
- noteTitleQuality
  - qevercloud::NoteAttributes, 297
- notFoundException
  - qevercloud::ManageNotebookSharesError, 269
  - qevercloud::ManageNoteSharesError, 276
- noUpdateContent
  - qevercloud::NoteRestrictions, 341
- noUpdateNotebook
  - qevercloud::NotebookRestrictions, 314
- noUpdateNotes
  - qevercloud::NotebookRestrictions, 314
- noUpdateTags
  - qevercloud::NotebookRestrictions, 314
- noUpdateTitle
  - qevercloud::NoteRestrictions, 341
- nullLogger
  - qevercloud, 46
- nullRetryPolicy
  - qevercloud, 46
- OAuth.h, 625, 626
- oauthError
  - qevercloud::EvernoteOAuthDialog, 163
  - qevercloud::EvernoteOAuthWebView, 166
- OAuthResult
  - qevercloud::EvernoteOAuthDialog, 162
- oauthResult
  - qevercloud::EvernoteOAuthDialog, 163
  - qevercloud::EvernoteOAuthWebView, 166
- omitSharedNotebooks
  - qevercloud::SyncChunkFilter, 468
- onAuthenticateLongSessionRequestReady
  - qevercloud::UserStoreServer, 508
- onAuthenticateToBusinessRequestReady
  - qevercloud::UserStoreServer, 508
- onAuthenticateToSharedNotebookRequestReady
  - qevercloud::NoteStoreServer, 378
- onAuthenticateToSharedNoteRequestReady
  - qevercloud::NoteStoreServer, 378
- onCheckVersionRequestReady
  - qevercloud::UserStoreServer, 508
- onCompleteTwoFactorAuthenticationRequestReady
  - qevercloud::UserStoreServer, 509
- onCopyNoteRequestReady
  - qevercloud::NoteStoreServer, 378
- onCreateLinkedNotebookRequestReady
  - qevercloud::NoteStoreServer, 378
- onCreateNotebookRequestReady
  - qevercloud::NoteStoreServer, 379
- onCreateNoteRequestReady
  - qevercloud::NoteStoreServer, 379
- onCreateOrUpdateNotebookSharesRequestReady
  - qevercloud::NoteStoreServer, 379
- onCreateSearchRequestReady
  - qevercloud::NoteStoreServer, 379
- onCreateTagRequestReady
  - qevercloud::NoteStoreServer, 379
- onDeleteNoteRequestReady
  - qevercloud::NoteStoreServer, 379
- onEmailNoteRequestReady
  - qevercloud::NoteStoreServer, 380
- onExpungeLinkedNotebookRequestReady
  - qevercloud::NoteStoreServer, 380
- onExpungeNotebookRequestReady
  - qevercloud::NoteStoreServer, 380
- onExpungeNoteRequestReady
  - qevercloud::NoteStoreServer, 380
- onExpungeSearchRequestReady
  - qevercloud::NoteStoreServer, 380
- onExpungeTagRequestReady
  - qevercloud::NoteStoreServer, 380
- onFindNoteCountsRequestReady
  - qevercloud::NoteStoreServer, 381
- onFindNoteOffsetRequestReady
  - qevercloud::NoteStoreServer, 381
- onFindNotesMetadataRequestReady
  - qevercloud::NoteStoreServer, 381
- onFindRelatedRequestReady
  - qevercloud::NoteStoreServer, 381
- onGetAccountLimitsRequestReady
  - qevercloud::UserStoreServer, 509
- onGetBootstrapInfoRequestReady
  - qevercloud::UserStoreServer, 509
- onGetDefaultNotebookRequestReady
  - qevercloud::NoteStoreServer, 381
- onGetFilteredSyncChunkRequestReady
  - qevercloud::NoteStoreServer, 381
- onGetLinkedNotebookSyncChunkRequestReady
  - qevercloud::NoteStoreServer, 382

- onGetLinkedNotebookSyncStateRequestReady  
    qevercloud::NoteStoreServer, 382
- onGetNoteApplicationDataEntryRequestReady  
    qevercloud::NoteStoreServer, 382
- onGetNoteApplicationDataRequestReady  
    qevercloud::NoteStoreServer, 382
- onGetNotebookRequestReady  
    qevercloud::NoteStoreServer, 382
- onGetNotebookSharesRequestReady  
    qevercloud::NoteStoreServer, 382
- onGetNoteContentRequestReady  
    qevercloud::NoteStoreServer, 383
- onGetNoteRequestReady  
    qevercloud::NoteStoreServer, 383
- onGetNoteSearchTextRequestReady  
    qevercloud::NoteStoreServer, 383
- onGetNoteTagNamesRequestReady  
    qevercloud::NoteStoreServer, 383
- onGetNoteVersionRequestReady  
    qevercloud::NoteStoreServer, 383
- onGetNoteWithResultSpecRequestReady  
    qevercloud::NoteStoreServer, 383
- onGetPublicNotebookRequestReady  
    qevercloud::NoteStoreServer, 384
- onGetPublicUserInfoRequestReady  
    qevercloud::UserStoreServer, 509
- onGetResourceAlternateDataRequestReady  
    qevercloud::NoteStoreServer, 384
- onGetResourceApplicationDataEntryRequestReady  
    qevercloud::NoteStoreServer, 384
- onGetResourceApplicationDataRequestReady  
    qevercloud::NoteStoreServer, 384
- onGetResourceAttributesRequestReady  
    qevercloud::NoteStoreServer, 384
- onGetResourceByHashRequestReady  
    qevercloud::NoteStoreServer, 384
- onGetResourceDataRequestReady  
    qevercloud::NoteStoreServer, 385
- onGetResourceRecognitionRequestReady  
    qevercloud::NoteStoreServer, 385
- onGetResourceRequestReady  
    qevercloud::NoteStoreServer, 385
- onGetResourceSearchTextRequestReady  
    qevercloud::NoteStoreServer, 385
- onGetSearchRequestReady  
    qevercloud::NoteStoreServer, 385
- onGetSharedNotebookByAuthRequestReady  
    qevercloud::NoteStoreServer, 385
- onGetSyncStateRequestReady  
    qevercloud::NoteStoreServer, 386
- onGetTagRequestReady  
    qevercloud::NoteStoreServer, 386
- onGetUserRequestReady  
    qevercloud::UserStoreServer, 509
- onGetUserUrlsRequestReady  
    qevercloud::UserStoreServer, 509
- onInviteToBusinessRequestReady  
    qevercloud::UserStoreServer, 510
- onListAccessibleBusinessNotebooksRequestReady  
    qevercloud::NoteStoreServer, 386
- onListBusinessInvitationsRequestReady  
    qevercloud::UserStoreServer, 510
- onListBusinessUsersRequestReady  
    qevercloud::UserStoreServer, 510
- onListLinkedNotebooksRequestReady  
    qevercloud::NoteStoreServer, 386
- onListNotebooksRequestReady  
    qevercloud::NoteStoreServer, 386
- onListNoteVersionsRequestReady  
    qevercloud::NoteStoreServer, 386
- onListSearchesRequestReady  
    qevercloud::NoteStoreServer, 387
- onListSharedNotebooksRequestReady  
    qevercloud::NoteStoreServer, 387
- onListTagsByNotebookRequestReady  
    qevercloud::NoteStoreServer, 387
- onListTagsRequestReady  
    qevercloud::NoteStoreServer, 387
- onManageNotebookSharesRequestReady  
    qevercloud::NoteStoreServer, 387
- onRemoveFromBusinessRequestReady  
    qevercloud::UserStoreServer, 510
- onRequest  
    qevercloud::NoteStoreServer, 387  
    qevercloud::UserStoreServer, 510
- onRevokeLongSessionRequestReady  
    qevercloud::UserStoreServer, 510
- onSetNoteApplicationDataEntryRequestReady  
    qevercloud::NoteStoreServer, 388
- onSetNotebookRecipientSettingsRequestReady  
    qevercloud::NoteStoreServer, 388
- onSetResourceApplicationDataEntryRequestReady  
    qevercloud::NoteStoreServer, 388
- onShareNotebookRequestReady  
    qevercloud::NoteStoreServer, 388
- onShareNoteRequestReady  
    qevercloud::NoteStoreServer, 388
- onStopSharingNoteRequestReady  
    qevercloud::NoteStoreServer, 388
- onUnsetNoteApplicationDataEntryRequestReady  
    qevercloud::NoteStoreServer, 389
- onUnsetResourceApplicationDataEntryRequestReady  
    qevercloud::NoteStoreServer, 389
- onUntagAllRequestReady  
    qevercloud::NoteStoreServer, 389
- onUpdateBusinessUserIdentifierRequestReady  
    qevercloud::UserStoreServer, 511
- onUpdateLinkedNotebookRequestReady  
    qevercloud::NoteStoreServer, 389
- onUpdateNotebookRequestReady  
    qevercloud::NoteStoreServer, 389
- onUpdateNoteIfUsnMatchesRequestReady  
    qevercloud::NoteStoreServer, 389
- onUpdateNoteRequestReady  
    qevercloud::NoteStoreServer, 390
- onUpdateResourceRequestReady



- qevercloud::NoteStoreServer, 390
- onUpdateSearchRequestReady
  - qevercloud::NoteStoreServer, 390
- onUpdateSharedNotebookRequestReady
  - qevercloud::NoteStoreServer, 390
- onUpdateTagRequestReady
  - qevercloud::NoteStoreServer, 390
- open
  - qevercloud::EvernoteOAuthDialog, 164
- OPENID\_ALREADY\_TAKEN
  - qevercloud, 35
- operator const T &
  - qevercloud::Optional< T >, 404
- operator T&
  - qevercloud::Optional< T >, 405
- operator!=
  - qevercloud::Accounting, 90
  - qevercloud::AccountLimits, 94
  - qevercloud::AuthenticationResult, 102
  - qevercloud::BootstrapInfo, 105
  - qevercloud::BootstrapProfile, 106
  - qevercloud::BootstrapSettings, 108
  - qevercloud::BusinessInvitation, 112
  - qevercloud::BusinessNotebook, 114
  - qevercloud::BusinessUserAttributes, 116
  - qevercloud::BusinessUserInfo, 118
  - qevercloud::CanMoveToContainerRestrictions, 120
  - qevercloud::Contact, 122
  - qevercloud::CreateOrUpdateNotebookSharesResult, 125
  - qevercloud::Data, 127
  - qevercloud::EDAMInvalidContactsException, 130
  - qevercloud::EDAMNotFoundException, 134
  - qevercloud::EDAMSystemException, 139
  - qevercloud::EDAMUserException, 148
  - qevercloud::EverCloudLocalData, 157
  - qevercloud::Identity, 168
  - qevercloud::InvitationShareRelationship, 241
  - qevercloud::LazyMap, 263
  - qevercloud::LinkedNotebook, 265
  - qevercloud::ManageNotebookSharesError, 268
  - qevercloud::ManageNotebookSharesParameters, 270
  - qevercloud::ManageNotebookSharesResult, 273
  - qevercloud::ManageNoteSharesError, 275
  - qevercloud::ManageNoteSharesParameters, 277
  - qevercloud::ManageNoteSharesResult, 280
  - qevercloud::MemberShareRelationship, 282
  - qevercloud::NetworkException, 285
  - qevercloud::Note, 288
  - qevercloud::NoteAttributes, 294
  - qevercloud::Notebook, 300
  - qevercloud::NotebookDescriptor, 305
  - qevercloud::NotebookRecipientSettings, 307
  - qevercloud::NotebookRestrictions, 310
  - qevercloud::NotebookShareTemplate, 315
  - qevercloud::NoteCollectionCounts, 318
  - qevercloud::NoteEmailParameters, 320
  - qevercloud::NoteFilter, 323
  - qevercloud::NoteInvitationShareRelationship, 327
  - qevercloud::NoteLimits, 329
  - qevercloud::NoteList, 331
  - qevercloud::NoteMemberShareRelationship, 334
  - qevercloud::NoteMetadata, 336
  - qevercloud::NoteRestrictions, 339
  - qevercloud::NoteResultSpec, 342
  - qevercloud::NoteShareRelationshipRestrictions, 344
  - qevercloud::NoteShareRelationships, 346
  - qevercloud::NotesMetadataList, 349
  - qevercloud::NotesMetadataResultSpec, 351
  - qevercloud::NoteVersionId, 397
  - qevercloud::Optional< T >, 405
  - qevercloud::PublicUserInfo, 412
  - qevercloud::Publishing, 414
  - qevercloud::QAssociativeContainerConstReferenceWrapper< Container >::iterator, 245
  - qevercloud::QAssociativeContainerReferenceWrapper< Container >::iterator, 247
  - qevercloud::RelatedContent, 418
  - qevercloud::RelatedContentImage, 422
  - qevercloud::RelatedQuery, 424
  - qevercloud::RelatedResult, 427
  - qevercloud::RelatedResultSpec, 431
  - qevercloud::Resource, 434
  - qevercloud::ResourceAttributes, 437
  - qevercloud::SavedSearch, 441
  - qevercloud::SavedSearchScope, 443
  - qevercloud::SharedNote, 445
  - qevercloud::SharedNotebook, 448
  - qevercloud::SharedNotebookRecipientSettings, 452
  - qevercloud::SharedNoteTemplate, 453
  - qevercloud::ShareRelationshipRestrictions, 456
  - qevercloud::ShareRelationships, 458
  - qevercloud::SyncChunk, 460
  - qevercloud::SyncChunkFilter, 466
  - qevercloud::SyncState, 471
  - qevercloud::Tag, 473
  - qevercloud::ThriftException, 477
  - qevercloud::UpdateNoteIfUsnMatchesResult, 485
  - qevercloud::User, 487
  - qevercloud::UserAttributes, 492
  - qevercloud::UserIdentity, 498
  - qevercloud::UserProfile, 500
  - qevercloud::UserUrls, 512
  - operator<<
    - qevercloud, 46–54
    - qevercloud::IRequestContext, 244
    - qevercloud::Printable, 410, 411
    - qevercloud::ThriftException, 477
    - qevercloud::Thumbnail, 484
  - operator\*
    - qevercloud::QAssociativeContainerConstReferenceWrapper< Container >::iterator, 246

- qevercloud::QAssociativeContainerReferenceWrapper< Container >::iterator, [247](#)
- operator++
  - qevercloud::QAssociativeContainerConstReferenceWrapper< Container >::iterator, [246](#)
  - qevercloud::QAssociativeContainerReferenceWrapper< Container >::iterator, [247](#)
- operator->
  - qevercloud::Optional< T >, [405](#)
- operator=
  - qevercloud::Optional< T >, [406](#)
- operator==
  - qevercloud::Accounting, [90](#)
  - qevercloud::AccountLimits, [95](#)
  - qevercloud::AuthenticationResult, [102](#)
  - qevercloud::BootstrapInfo, [105](#)
  - qevercloud::BootstrapProfile, [107](#)
  - qevercloud::BootstrapSettings, [108](#)
  - qevercloud::BusinessInvitation, [112](#)
  - qevercloud::BusinessNotebook, [114](#)
  - qevercloud::BusinessUserAttributes, [116](#)
  - qevercloud::BusinessUserInfo, [118](#)
  - qevercloud::CanMoveToContainerRestrictions, [121](#)
  - qevercloud::Contact, [122](#)
  - qevercloud::CreateOrUpdateNotebookSharesResult, [125](#)
  - qevercloud::Data, [127](#)
  - qevercloud::EDAMInvalidContactsException, [130](#)
  - qevercloud::EDAMNotFoundException, [134](#)
  - qevercloud::EDAMSystemException, [139](#)
  - qevercloud::EDAMUserException, [148](#)
  - qevercloud::EverCloudLocalData, [157](#)
  - qevercloud::Identity, [168](#)
  - qevercloud::InvitationShareRelationship, [241](#)
  - qevercloud::LazyMap, [263](#)
  - qevercloud::LinkedNotebook, [265](#)
  - qevercloud::ManageNotebookSharesError, [268](#)
  - qevercloud::ManageNotebookSharesParameters, [270](#)
  - qevercloud::ManageNotebookSharesResult, [273](#)
  - qevercloud::ManageNoteSharesError, [275](#)
  - qevercloud::ManageNoteSharesParameters, [277](#)
  - qevercloud::ManageNoteSharesResult, [280](#)
  - qevercloud::MemberShareRelationship, [282](#)
  - qevercloud::NetworkException, [285](#)
  - qevercloud::Note, [288](#)
  - qevercloud::NoteAttributes, [294](#)
  - qevercloud::Notebook, [300](#)
  - qevercloud::NotebookDescriptor, [305](#)
  - qevercloud::NotebookRecipientSettings, [307](#)
  - qevercloud::NotebookRestrictions, [310](#)
  - qevercloud::NotebookShareTemplate, [315](#)
  - qevercloud::NoteCollectionCounts, [318](#)
  - qevercloud::NoteEmailParameters, [320](#)
  - qevercloud::NoteFilter, [323](#)
  - qevercloud::NoteInvitationShareRelationship, [327](#)
  - qevercloud::NoteLimits, [329](#)
  - qevercloud::NoteList, [331](#)
  - qevercloud::NoteMemberShareRelationship, [334](#)
  - qevercloud::NoteMetadata, [336](#)
  - qevercloud::NoteRestrictions, [340](#)
  - qevercloud::NoteResultSpec, [342](#)
  - qevercloud::NoteShareRelationshipRestrictions, [344](#)
  - qevercloud::NoteShareRelationships, [346](#)
  - qevercloud::NotesMetadataList, [349](#)
  - qevercloud::NotesMetadataResultSpec, [352](#)
  - qevercloud::NoteVersionId, [397](#)
  - qevercloud::Optional< T >, [407](#)
  - qevercloud::PublicUserInfo, [412](#)
  - qevercloud::Publishing, [414](#)
  - qevercloud::RelatedContent, [418](#)
  - qevercloud::RelatedContentImage, [422](#)
  - qevercloud::RelatedQuery, [424](#)
  - qevercloud::RelatedResult, [427](#)
  - qevercloud::RelatedResultSpec, [431](#)
  - qevercloud::Resource, [434](#)
  - qevercloud::ResourceAttributes, [438](#)
  - qevercloud::SavedSearch, [441](#)
  - qevercloud::SavedSearchScope, [443](#)
  - qevercloud::SharedNote, [445](#)
  - qevercloud::SharedNotebook, [448](#)
  - qevercloud::SharedNotebookRecipientSettings, [452](#)
  - qevercloud::SharedNoteTemplate, [454](#)
  - qevercloud::ShareRelationshipRestrictions, [456](#)
  - qevercloud::ShareRelationships, [458](#)
  - qevercloud::SyncChunk, [461](#)
  - qevercloud::SyncChunkFilter, [466](#)
  - qevercloud::SyncState, [471](#)
  - qevercloud::Tag, [473](#)
  - qevercloud::ThriftException, [477](#)
  - qevercloud::UpdateNoteIfUsnMatchesResult, [485](#)
  - qevercloud::User, [487](#)
  - qevercloud::UserAttributes, [492](#)
  - qevercloud::UserIdentity, [498](#)
  - qevercloud::UserProfile, [500](#)
  - qevercloud::UserUrls, [512](#)
  - Optional
    - qevercloud::Optional< T >, [402](#), [403](#), [408](#)
  - Optional.h, [627](#)
  - optOutMachineLearning
    - qevercloud::UserAttributes, [494](#)
  - order
    - qevercloud::NoteFilter, [325](#)
    - qevercloud::Publishing, [415](#)
  - parameter
    - qevercloud::EDAMInvalidContactsException, [130](#)
    - qevercloud::EDAMUserException, [148](#)
  - parentGuid
    - qevercloud::Tag, [474](#)
  - partnerEmailOptInDate
    - qevercloud::UserAttributes, [495](#)
  - passwordUpdated
    - qevercloud::UserAttributes, [495](#)
  - PENDING

- qevercloud, [37](#)
- PERMISSION\_DENIED
  - qevercloud, [35](#)
- photoLastUpdated
  - qevercloud::Contact, [123](#)
  - qevercloud::User, [489](#)
  - qevercloud::UserProfile, [501](#)
- photoUrl
  - qevercloud::Contact, [123](#)
  - qevercloud::User, [489](#)
  - qevercloud::UserProfile, [501](#)
- pixelRatio
  - qevercloud::RelatedContentImage, [423](#)
- placeName
  - qevercloud::NoteAttributes, [297](#)
- plainText
  - qevercloud::RelatedQuery, [425](#)
- PLUS
  - qevercloud, [40](#)
- PNG
  - qevercloud::Thumbnail, [481](#)
- preactivation
  - qevercloud::UserAttributes, [495](#)
- preferredCountry
  - qevercloud::UserAttributes, [495](#)
- preferredLanguage
  - qevercloud::UserAttributes, [495](#)
- PREMIUM
  - qevercloud, [38, 40](#)
- premiumCommerceService
  - qevercloud::Accounting, [92](#)
- premiumLockUntil
  - qevercloud::Accounting, [92](#)
- premiumOrderNumber
  - qevercloud::Accounting, [92](#)
- PremiumOrderStatus
  - qevercloud, [37](#)
- premiumServiceSKU
  - qevercloud::Accounting, [92](#)
- premiumServiceStart
  - qevercloud::Accounting, [92](#)
- premiumServiceStatus
  - qevercloud::Accounting, [93](#)
- premiumSubscriptionNumber
  - qevercloud::Accounting, [93](#)
- print
  - qevercloud::Accounting, [90](#)
  - qevercloud::AccountLimits, [95](#)
  - qevercloud::AuthenticationResult, [102](#)
  - qevercloud::BootstrapInfo, [105](#)
  - qevercloud::BootstrapProfile, [107](#)
  - qevercloud::BootstrapSettings, [109](#)
  - qevercloud::BusinessInvitation, [112](#)
  - qevercloud::BusinessNotebook, [114](#)
  - qevercloud::BusinessUserAttributes, [116](#)
  - qevercloud::BusinessUserInfo, [119](#)
  - qevercloud::CanMoveToContainerRestrictions, [121](#)
  - qevercloud::Contact, [122](#)
  - qevercloud::CreateOrUpdateNotebookSharesResult, [125](#)
  - qevercloud::Data, [127](#)
  - qevercloud::EDAMInvalidContactsException, [130](#)
  - qevercloud::EDAMNotFoundException, [135](#)
  - qevercloud::EDAMSystemException, [139](#)
  - qevercloud::EDAMUserException, [148](#)
  - qevercloud::EverCloudLocalData, [157](#)
  - qevercloud::EvernoteOAuthWebView::OAuthResult, [399](#)
  - qevercloud::Identity, [168](#)
  - qevercloud::InvitationShareRelationship, [241](#)
  - qevercloud::LazyMap, [263](#)
  - qevercloud::LinkedNotebook, [265](#)
  - qevercloud::ManageNotebookSharesError, [269](#)
  - qevercloud::ManageNotebookSharesParameters, [271](#)
  - qevercloud::ManageNotebookSharesResult, [273](#)
  - qevercloud::ManageNoteSharesError, [275](#)
  - qevercloud::ManageNoteSharesParameters, [277](#)
  - qevercloud::ManageNoteSharesResult, [280](#)
  - qevercloud::MemberShareRelationship, [282](#)
  - qevercloud::Note, [289](#)
  - qevercloud::NoteAttributes, [294](#)
  - qevercloud::Notebook, [300](#)
  - qevercloud::NotebookDescriptor, [305](#)
  - qevercloud::NotebookRecipientSettings, [307](#)
  - qevercloud::NotebookRestrictions, [310](#)
  - qevercloud::NotebookShareTemplate, [316](#)
  - qevercloud::NoteCollectionCounts, [318](#)
  - qevercloud::NoteEmailParameters, [321](#)
  - qevercloud::NoteFilter, [323](#)
  - qevercloud::NoteInvitationShareRelationship, [327](#)
  - qevercloud::NoteLimits, [329](#)
  - qevercloud::NoteList, [331](#)
  - qevercloud::NoteMemberShareRelationship, [334](#)
  - qevercloud::NoteMetadata, [336](#)
  - qevercloud::NoteRestrictions, [340](#)
  - qevercloud::NoteResultSpec, [342](#)
  - qevercloud::NoteShareRelationshipRestrictions, [345](#)
  - qevercloud::NoteShareRelationships, [347](#)
  - qevercloud::NotesMetadataList, [349](#)
  - qevercloud::NotesMetadataResultSpec, [352](#)
  - qevercloud::NoteVersionId, [397](#)
  - qevercloud::Printable, [410](#)
  - qevercloud::PublicUserInfo, [412](#)
  - qevercloud::Publishing, [414](#)
  - qevercloud::RelatedContent, [418](#)
  - qevercloud::RelatedContentImage, [422](#)
  - qevercloud::RelatedQuery, [424](#)
  - qevercloud::RelatedResult, [427](#)
  - qevercloud::RelatedResultSpec, [431](#)
  - qevercloud::Resource, [434](#)
  - qevercloud::ResourceAttributes, [438](#)
  - qevercloud::SavedSearch, [441](#)
  - qevercloud::SavedSearchScope, [444](#)
  - qevercloud::SharedNote, [446](#)

- qevercloud::SharedNotebook, 448
- qevercloud::SharedNotebookRecipientSettings, 452
- qevercloud::SharedNoteTemplate, 454
- qevercloud::ShareRelationshipRestrictions, 456
- qevercloud::ShareRelationships, 458
- qevercloud::SyncChunk, 461
- qevercloud::SyncChunkFilter, 466
- qevercloud::SyncState, 471
- qevercloud::Tag, 473
- qevercloud::UpdateNoteIfUsnMatchesResult, 485
- qevercloud::User, 487
- qevercloud::UserAttributes, 492
- qevercloud::UserIdentity, 498
- qevercloud::UserProfile, 500
- qevercloud::UserUrls, 513
- Printable
  - qevercloud::Printable, 409
- Printable.h, 630
- privilege
  - qevercloud::BusinessNotebook, 115
  - qevercloud::InvitationShareRelationship, 242
  - qevercloud::NotebookShareTemplate, 316
  - qevercloud::NoteInvitationShareRelationship, 327
  - qevercloud::NoteMemberShareRelationship, 334
  - qevercloud::SharedNote, 446
  - qevercloud::SharedNotebook, 449
  - qevercloud::SharedNoteTemplate, 454
  - qevercloud::User, 489
- PrivilegeLevel
  - qevercloud, 37
- PROFILE\_ORGANIZATION
  - qevercloud, 39
- PROFILE\_PERSON
  - qevercloud, 39
- profiles
  - qevercloud::BootstrapInfo, 106
- PROTOCOL\_ERROR
  - qevercloud::ThriftException, 476
- publicDescription
  - qevercloud::Publishing, 415
- publicUserInfo
  - qevercloud::AuthenticationResult, 103
- published
  - qevercloud::Notebook, 302
- publishing
  - qevercloud::Notebook, 302
- QAssociativeContainerConstReferenceWrapper
  - qevercloud::QAssociativeContainerConstReferenceWrapper<Container >, 416
- QAssociativeContainerReferenceWrapper
  - qevercloud::QAssociativeContainerReferenceWrapper<Container >, 417
- QEC\_DEBUG
  - Log.h, 622
- QEC\_ERROR
  - Log.h, 623
- QEC\_INFO
  - Log.h, 623
- QEC\_TRACE
  - Log.h, 623
- QEC\_WARNING
  - Log.h, 623
- qevercloud, 19
  - ACCOUNT\_CLEAR, 35
  - ACTIVE, 33, 37
  - ADMIN, 32, 38
  - APPROVED, 32
  - ASSIGNED, 40
  - AUTH\_EXPIRED, 35
  - BAD\_ADDRESS, 36
  - BAD\_DATA\_FORMAT, 35
  - BASIC, 40
  - BUSINESS, 40
  - BUSINESS\_FULL\_ACCESS, 41
  - BUSINESS\_SECURITY\_LOGIN\_REQUIRED, 35
  - BusinessInvitationStatus, 32
  - BusinessUserRole, 32
  - BusinessUserStatus, 32
  - CAN\_BE\_MOVED, 33
  - CANCELED, 37
  - CANCELLATION\_PENDING, 37
  - CanMoveToContainerStatus, 33
  - CLASSIFICATION\_RECIPE\_SERVICE\_RECIPE, 59
  - CLASSIFICATION\_RECIPE\_USER\_NON\_RECIPE, 59
  - CLASSIFICATION\_RECIPE\_USER\_RECIPE, 59
  - ContactType, 33
  - CREATED, 37
  - DATA\_CONFLICT, 35
  - DATA\_REQUIRED, 35
  - DEACTIVATED, 33
  - Debug, 36
  - DEVICE\_LIMIT\_REACHED, 35
  - DIRECT\_LINK\_ACCESS\_OK, 39
  - DIRECT\_LINK\_EMBEDDED\_VIEW, 39
  - DIRECT\_LINK\_LOGIN\_REQUIRED, 39
  - DO\_NOT\_SEND, 40
  - DUPLICATE\_CONTACT, 36
  - EDAM\_APP\_RATING\_MAX, 59
  - EDAM\_APP\_RATING\_MIN, 59
  - EDAM\_APPLICATIONDATA\_ENTRY\_LEN\_MAX, 60
  - EDAM\_APPLICATIONDATA\_NAME\_LEN\_MAX, 60
  - EDAM\_APPLICATIONDATA\_NAME\_LEN\_MIN, 60
  - EDAM\_APPLICATIONDATA\_NAME\_REGEX, 60
  - EDAM\_APPLICATIONDATA\_VALUE\_LEN\_MAX, 60
  - EDAM\_APPLICATIONDATA\_VALUE\_LEN\_MIN, 60
  - EDAM\_APPLICATIONDATA\_VALUE\_REGEX, 60
  - EDAM\_ATTRIBUTE\_LEN\_MAX, 61
  - EDAM\_ATTRIBUTE\_LEN\_MIN, 61
  - EDAM\_ATTRIBUTE\_LIST\_MAX, 61

- EDAM\_ATTRIBUTE\_MAP\_MAX, [61](#)
- EDAM\_ATTRIBUTE\_REGEX, [61](#)
- EDAM\_BUSINESS\_MARKETING\_CODE\_REGEX\_PATTERN, [61](#)
- EDAM\_BUSINESS\_NOTEBOOK\_DESCRIPTION\_LEN\_MAX, [61](#)
- EDAM\_BUSINESS\_NOTEBOOK\_DESCRIPTION\_LEN\_MIN, [62](#)
- EDAM\_BUSINESS\_NOTEBOOK\_DESCRIPTION\_REGEX, [62](#)
- EDAM\_BUSINESS\_NOTEBOOKS\_MAX, [62](#)
- EDAM\_BUSINESS\_NOTES\_MAX, [62](#)
- EDAM\_BUSINESS\_PHONE\_NUMBER\_LEN\_MAX, [62](#)
- EDAM\_BUSINESS\_TAGS\_MAX, [62](#)
- EDAM\_BUSINESS\_URI\_LEN\_MAX, [62](#)
- EDAM\_BUSINESS\_WORKSPACES\_MAX, [63](#)
- EDAM\_CONNECTED\_IDENTITY\_REQUEST\_MAX, [63](#)
- EDAM\_CONTENT\_CLASS\_FOOD\_MEAL, [63](#)
- EDAM\_CONTENT\_CLASS\_HELLO\_ENCOUNTER, [63](#)
- EDAM\_CONTENT\_CLASS\_HELLO\_PROFILE, [63](#)
- EDAM\_CONTENT\_CLASS\_PENULTIMATE\_NOTEBOOK, [63](#)
- EDAM\_CONTENT\_CLASS\_PENULTIMATE\_PREFIX, [63](#)
- EDAM\_CONTENT\_CLASS\_SKITCH, [64](#)
- EDAM\_CONTENT\_CLASS\_SKITCH\_PDF, [64](#)
- EDAM\_CONTENT\_CLASS\_SKITCH\_PREFIX, [64](#)
- EDAM\_DEVICE\_DESCRIPTION\_LEN\_MAX, [64](#)
- EDAM\_DEVICE\_DESCRIPTION\_REGEX, [64](#)
- EDAM\_DEVICE\_ID\_LEN\_MAX, [64](#)
- EDAM\_DEVICE\_ID\_REGEX, [64](#)
- EDAM\_EMAIL\_DOMAIN\_REGEX, [65](#)
- EDAM\_EMAIL\_LEN\_MAX, [65](#)
- EDAM\_EMAIL\_LEN\_MIN, [65](#)
- EDAM\_EMAIL\_LOCAL\_REGEX, [65](#)
- EDAM\_EMAIL\_REGEX, [65](#)
- EDAM\_FIND\_CONTACT\_DEFAULT\_MAX\_RESULTS, [65](#)
- EDAM\_FIND\_CONTACT\_MAX\_RESULTS, [65](#)
- EDAM\_FOOD\_APP\_CONTENT\_CLASS\_PREFIX, [66](#)
- EDAM\_GET\_ORDERS\_MAX\_RESULTS, [66](#)
- EDAM\_GUID\_LEN\_MAX, [66](#)
- EDAM\_GUID\_LEN\_MIN, [66](#)
- EDAM\_GUID\_REGEX, [66](#)
- EDAM\_HASH\_LEN, [66](#)
- EDAM\_HELLO\_APP\_CONTENT\_CLASS\_PREFIX, [66](#)
- EDAM\_INDEXABLE\_PLAINTEXT\_MIME\_TYPES, [67](#)
- EDAM\_INDEXABLE\_RESOURCE\_MIME\_TYPES, [67](#)
- EDAM\_MAX\_PREFERENCES, [67](#)
- EDAM\_MAX\_VALUES\_PER\_PREFERENCE, [67](#)
- EDAM\_MESSAGE\_ATTACHMENT\_SNIPPET\_LEN\_MAX, [67](#)
- EDAM\_MESSAGE\_ATTACHMENT\_SNIPPET\_REGEX, [67](#)
- EDAM\_MESSAGE\_ATTACHMENT\_TITLE\_LEN\_MAX, [67](#)
- EDAM\_MESSAGE\_ATTACHMENT\_TITLE\_REGEX, [68](#)
- EDAM\_MESSAGE\_ATTACHMENTS\_MAX, [68](#)
- EDAM\_MESSAGE\_BODY\_LEN\_MAX, [68](#)
- EDAM\_MESSAGE\_BODY\_REGEX, [68](#)
- EDAM\_MESSAGE\_RECIPIENTS\_MAX, [68](#)
- EDAM\_MIME\_LEN\_MAX, [68](#)
- EDAM\_MIME\_LEN\_MIN, [68](#)
- EDAM\_MIME\_REGEX, [68](#)
- EDAM\_MIME\_TYPE\_AAC, [69](#)
- EDAM\_MIME\_TYPE\_AMR, [69](#)
- EDAM\_MIME\_TYPE\_BMP, [69](#)
- EDAM\_MIME\_TYPE\_DEFAULT, [69](#)
- EDAM\_MIME\_TYPE\_GIF, [69](#)
- EDAM\_MIME\_TYPE\_INK, [69](#)
- EDAM\_MIME\_TYPE\_JPEG, [69](#)
- EDAM\_MIME\_TYPE\_M4A, [69](#)
- EDAM\_MIME\_TYPE\_MP3, [70](#)
- EDAM\_MIME\_TYPE\_MP4\_VIDEO, [70](#)
- EDAM\_MIME\_TYPE\_PDF, [70](#)
- EDAM\_MIME\_TYPE\_PNG, [70](#)
- EDAM\_MIME\_TYPE\_TIFF, [70](#)
- EDAM\_MIME\_TYPE\_WAV, [70](#)
- EDAM\_MIME\_TYPES, [70](#)
- EDAM\_NOTE\_BUSINESS\_SHARED\_NOTE\_MAX, [70](#)
- EDAM\_NOTE\_CONTENT\_CLASS\_LEN\_MAX, [71](#)
- EDAM\_NOTE\_CONTENT\_CLASS\_LEN\_MIN, [71](#)
- EDAM\_NOTE\_CONTENT\_CLASS\_REGEX, [71](#)
- EDAM\_NOTE\_CONTENT\_LEN\_MAX, [71](#)
- EDAM\_NOTE\_CONTENT\_LEN\_MIN, [71](#)
- EDAM\_NOTE\_LOCK\_VIEWERS\_NOTES\_MAX, [71](#)
- EDAM\_NOTE\_PERSONAL\_SHARED\_NOTE\_MAX, [71](#)
- EDAM\_NOTE\_RESOURCES\_MAX, [71](#)
- EDAM\_NOTE\_SIZE\_MAX\_FREE, [72](#)
- EDAM\_NOTE\_SIZE\_MAX\_PREMIUM, [72](#)
- EDAM\_NOTE\_SOURCE\_MAIL\_CLIP, [72](#)
- EDAM\_NOTE\_SOURCE\_MAIL\_SMTP\_GATEWAY, [72](#)
- EDAM\_NOTE\_SOURCE\_WEB\_CLIP, [72](#)
- EDAM\_NOTE\_SOURCE\_WEB\_CLIP\_SIMPLIFIED, [72](#)
- EDAM\_NOTE\_TAGS\_MAX, [72](#)
- EDAM\_NOTE\_TITLE\_LEN\_MAX, [73](#)
- EDAM\_NOTE\_TITLE\_LEN\_MIN, [73](#)
- EDAM\_NOTE\_TITLE\_QUALITY\_HIGH, [73](#)
- EDAM\_NOTE\_TITLE\_QUALITY\_LOW, [73](#)
- EDAM\_NOTE\_TITLE\_QUALITY\_MEDIUM, [73](#)
- EDAM\_NOTE\_TITLE\_QUALITY\_UNTITLED, [73](#)
- EDAM\_NOTE\_TITLE\_REGEX, [73](#)

- EDAM\_NOTEBOOK\_BUSINESS\_SHARED\_NOTEBOOK\_MAX, 74
- EDAM\_NOTEBOOK\_NAME\_LEN\_MAX, 74
- EDAM\_NOTEBOOK\_NAME\_LEN\_MIN, 74
- EDAM\_NOTEBOOK\_NAME\_REGEX, 74
- EDAM\_NOTEBOOK\_PERSONAL\_SHARED\_NOTEBOOK\_MAX, 74
- EDAM\_NOTEBOOK\_STACK\_LEN\_MAX, 74
- EDAM\_NOTEBOOK\_STACK\_LEN\_MIN, 74
- EDAM\_NOTEBOOK\_STACK\_REGEX, 75
- EDAM\_OPEN\_ID\_ACCESS\_TOKEN\_MAX, 75
- EDAM\_PREFERENCE\_BUSINESS\_DEFAULT\_NOTEBOOK, 75
- EDAM\_PREFERENCE\_BUSINESS\_QUICKNOTE, 75
- EDAM\_PREFERENCE\_NAME\_LEN\_MAX, 75
- EDAM\_PREFERENCE\_NAME\_LEN\_MIN, 75
- EDAM\_PREFERENCE\_NAME\_REGEX, 76
- EDAM\_PREFERENCE\_ONLY\_ONE\_VALUE\_LEN\_MAX, 76
- EDAM\_PREFERENCE\_ONLY\_ONE\_VALUE\_REGEX, 76
- EDAM\_PREFERENCE\_SHORTCUTS, 76
- EDAM\_PREFERENCE\_SHORTCUTS\_MAX\_VALUES, 76
- EDAM\_PREFERENCE\_VALUE\_LEN\_MAX, 76
- EDAM\_PREFERENCE\_VALUE\_LEN\_MIN, 76
- EDAM\_PREFERENCE\_VALUE\_REGEX, 77
- EDAM\_PROMOTION\_ID\_LEN\_MAX, 77
- EDAM\_PROMOTION\_ID\_REGEX, 77
- EDAM\_PUBLISHING\_DESCRIPTION\_LEN\_MAX, 77
- EDAM\_PUBLISHING\_DESCRIPTION\_LEN\_MIN, 77
- EDAM\_PUBLISHING\_DESCRIPTION\_REGEX, 77
- EDAM\_PUBLISHING\_URI\_LEN\_MAX, 77
- EDAM\_PUBLISHING\_URI\_LEN\_MIN, 78
- EDAM\_PUBLISHING\_URI\_PROHIBITED, 78
- EDAM\_PUBLISHING\_URI\_REGEX, 78
- EDAM\_RELATED\_MAX\_EXPERTS, 78
- EDAM\_RELATED\_MAX\_NOTEBOOKS, 78
- EDAM\_RELATED\_MAX\_NOTES, 78
- EDAM\_RELATED\_MAX\_RELATED\_CONTENT, 78
- EDAM\_RELATED\_MAX\_TAGS, 78
- EDAM\_RELATED\_PLAINTEXT\_LEN\_MAX, 79
- EDAM\_RELATED\_PLAINTEXT\_LEN\_MIN, 79
- EDAM\_RESOURCE\_SIZE\_MAX\_FREE, 79
- EDAM\_RESOURCE\_SIZE\_MAX\_PREMIUM, 79
- EDAM\_SAVED\_SEARCH\_NAME\_LEN\_MAX, 79
- EDAM\_SAVED\_SEARCH\_NAME\_LEN\_MIN, 79
- EDAM\_SAVED\_SEARCH\_NAME\_REGEX, 79
- EDAM\_SEARCH\_QUERY\_LEN\_MAX, 79
- EDAM\_SEARCH\_QUERY\_LEN\_MIN, 80
- EDAM\_SEARCH\_QUERY\_REGEX, 80
- EDAM\_SEARCH\_SUGGESTIONS\_MAX, 80
- EDAM\_SEARCH\_SUGGESTIONS\_PREFIX\_LEN\_MAX, 80
- EDAM\_SEARCH\_SUGGESTIONS\_PREFIX\_LEN\_MIN, 80
- EDAM\_SNIPPETS\_NOTES\_MAX, 80
- EDAM\_SOURCE\_APPLICATION\_ANDROID\_SHARE\_EXTENSION, 80
- EDAM\_SOURCE\_APPLICATION\_EN\_SCANSNAP, 81
- EDAM\_SOURCE\_APPLICATION\_EWC, 81
- EDAM\_SOURCE\_APPLICATION\_IOS\_SHARE\_EXTENSION, 81
- EDAM\_SOURCE\_APPLICATION\_MOLESKINE, 81
- EDAM\_SOURCE\_APPLICATION\_POSTIT, 81
- EDAM\_SOURCE\_APPLICATION\_WEB\_CLIPPER, 81
- EDAM\_SOURCE\_OUTLOOK\_CLIPPER, 81
- EDAM\_TAG\_NAME\_LEN\_MAX, 82
- EDAM\_TAG\_NAME\_LEN\_MIN, 82
- EDAM\_TAG\_NAME\_REGEX, 82
- EDAM\_TIMEZONE\_LEN\_MAX, 82
- EDAM\_TIMEZONE\_LEN\_MIN, 82
- EDAM\_TIMEZONE\_REGEX, 82
- EDAM\_USER\_LINKED\_NOTEBOOK\_MAX, 82
- EDAM\_USER\_LINKED\_NOTEBOOK\_MAX\_PREMIUM, 83
- EDAM\_USER\_MAIL\_LIMIT\_DAILY\_FREE, 83
- EDAM\_USER\_MAIL\_LIMIT\_DAILY\_PREMIUM, 83
- EDAM\_USER\_NAME\_LEN\_MAX, 83
- EDAM\_USER\_NAME\_LEN\_MIN, 83
- EDAM\_USER\_NAME\_REGEX, 83
- EDAM\_USER\_NOTEBOOKS\_MAX, 83
- EDAM\_USER\_NOTES\_MAX, 84
- EDAM\_USER\_PASSWORD\_LEN\_MAX, 84
- EDAM\_USER\_PASSWORD\_LEN\_MIN, 84
- EDAM\_USER\_PASSWORD\_REGEX, 84
- EDAM\_USER\_PROFILE\_PHOTO\_MAX\_BYTES, 84
- EDAM\_USER\_RECENT\_MAILED\_ADDRESSES\_MAX, 84
- EDAM\_USER\_SAVED\_SEARCHES\_MAX, 84
- EDAM\_USER\_TAGS\_MAX, 85
- EDAM\_USER\_UPLOAD\_LIMIT\_BUSINESS, 85
- EDAM\_USER\_UPLOAD\_LIMIT\_BUSINESS\_FIRST\_MONTH, 85
- EDAM\_USER\_UPLOAD\_LIMIT\_BUSINESS\_NEXT\_MONTH, 85
- EDAM\_USER\_UPLOAD\_LIMIT\_BUSINESS\_PER\_USER, 85
- EDAM\_USER\_UPLOAD\_LIMIT\_FREE, 85
- EDAM\_USER\_UPLOAD\_LIMIT\_PLUS, 85
- EDAM\_USER\_UPLOAD\_LIMIT\_PREMIUM, 86
- EDAM\_USER\_UPLOAD\_SURVEY\_THRESHOLD, 86
- EDAM\_USER\_USERNAME\_LEN\_MAX, 86
- EDAM\_USER\_USERNAME\_LEN\_MIN, 86
- EDAM\_USER\_USERNAME\_REGEX, 86
- EDAM\_USER\_WORKSPACES\_MAX, 86
- EDAM\_VAT\_REGEX, 86



- EDAM\_VERSION\_MAJOR, 87
- EDAM\_VERSION\_MINOR, 87
- EDAM\_WORKSPACE\_DESCRIPTION\_LEN\_MAX, 87
- EDAM\_WORKSPACE\_NAME\_LEN\_MAX, 87
- EDAM\_WORKSPACE\_NAME\_LEN\_MIN, 87
- EDAM\_WORKSPACE\_NAME\_REGEX, 87
- EDAMErrorCode, 34
- EDAMInvalidContactReason, 35
- EMAIL, 33, 44
- ENML\_VALIDATION, 35
- EntityType, 36
- Error, 36
- EverCloudExceptionData, 87
- EverCloudExceptionDataPtr, 29
- EVERNOTE, 33
- EVERNOTE\_USERID, 44
- evernoteNetworkProxy, 44
- FACEBOOK, 33
- FAILED, 37
- FULL\_ACCESS, 41, 43
- GROUP, 41
- GROUP\_ADMIN, 44
- GROUP\_MEMBER, 44
- GROUP\_OWNER, 44
- Guid, 29
- IDENTITYID, 44
- IdentityID, 30
- IDurableServicePtr, 30
- ILoggerPtr, 30
- IN\_MY\_LIST, 38
- IN\_MY\_LIST\_AND\_DEFAULT\_NOTEBOOK, 38
- Info, 36
- initializeQEverCloud, 44
- INoteStorePtr, 30
- INSUFFICIENT\_CONTAINER\_PRIVILEGE, 33
- INSUFFICIENT\_ENTITY\_PRIVILEGE, 33
- INTERNAL\_ERROR, 35
- INVALID\_AUTH, 35
- INVALID\_OPENID\_TOKEN, 35
- InvalidationSequenceNumber, 30
- IRequestContextPtr, 30
- IRetryPolicyPtr, 30
- IUserStorePtr, 31
- LEN\_TOO\_LONG, 35
- LEN\_TOO\_SHORT, 35
- libraryVersion, 44
- LIMIT\_REACHED, 35
- LINKEDIN, 33
- logger, 45
- LogLevel, 36
- MANAGER, 38
- MessageEventID, 31
- MessageThreadID, 31
- MODIFY\_NOTE, 43
- MODIFY\_NOTEBOOK\_PLUS\_ACTIVITY, 41, 43
- newDurableService, 45
- newNoteStore, 45
- newRequestContext, 45
- newRetryPolicy, 45
- NEWS\_ARTICLE, 39
- newStdErrLogger, 45
- newUserStore, 46
- NO\_CONNECTION, 36
- NO\_SHARED\_NOTEBOOKS, 40
- NONE, 37
- NORMAL, 32, 38
- NOT\_ACCESSIBLE, 39
- NOT\_IN\_MY\_LIST, 38
- NOTE, 36
- NOTEBOOK, 36
- NoteSortOrder, 37
- nullLogger, 46
- nullRetryPolicy, 46
- OPENID\_ALREADY\_TAKEN, 35
- operator<<, 46–54
- PENDING, 37
- PERMISSION\_DENIED, 35
- PLUS, 40
- PREMIUM, 38, 40
- PremiumOrderStatus, 37
- PrivilegeLevel, 37
- PROFILE\_ORGANIZATION, 39
- PROFILE\_PERSON, 39
- qHash, 54–57
- QueryFormat, 38
- QUOTA\_REACHED, 35
- RATE\_LIMIT\_REACHED, 35
- READ\_NOTE, 43
- READ\_NOTEBOOK, 41, 43
- READ\_NOTEBOOK\_PLUS\_ACTIVITY, 41, 43
- RecipientStatus, 38
- REDEEMED, 32
- REFERENCE\_MATERIAL, 39
- RelatedContentAccess, 39
- RelatedContentType, 39
- RELEVANCE, 37
- ReminderEmailConfig, 39
- REQUESTED, 32
- resetEvernoteNetworkProxy, 57
- SEND\_DAILY\_EMAIL, 40
- ServiceLevel, 40
- setEvernoteNetworkProxy, 58
- setLogger, 58
- setNonceGenerator, 58
- SEXP, 38
- SHARD\_UNAVAILABLE, 35
- SharedNotebookInstanceRestrictions, 40
- SharedNotebookPrivilegeLevel, 40
- SharedNotePrivilegeLevel, 41
- ShareRelationshipPrivilegeLevel, 43
- SMS, 33
- SponsoredGroupRole, 43
- SSO\_AUTHENTICATION\_REQUIRED, 35
- SUPPORT, 38
- TAKEN\_DOWN, 35

- Timestamp, 31
- TITLE, 37
- TOO\_FEW, 35
- TOO\_MANY, 35
- toRange, 58, 59
- Trace, 36
- TWITTER, 33
- UNKNOWN, 35
- UNSUPPORTED\_OPERATION, 35
- UPDATE\_SEQUENCE\_NUMBER, 37
- UPDATED, 37
- USER, 38
- USER\_ALREADY\_ASSOCIATED, 35
- USER\_NOT\_ASSOCIATED, 35
- USER\_NOT\_REGISTERED, 35
- UserID, 31
- UserIdentityType, 44
- VIP, 38
- Warn, 36
- WORKSPACE, 36
- QEverCloud.h, 631
- qevercloud::Accounting, 89
  - availablePoints, 90
  - businessId, 91
  - businessName, 91
  - businessRole, 91
  - currency, 91
  - lastFailedCharge, 91
  - lastFailedChargeReason, 91
  - lastRequestedCharge, 91
  - lastSuccessfulCharge, 91
  - localData, 92
  - nextChargeDate, 92
  - nextPaymentDue, 92
  - operator!=, 90
  - operator==, 90
  - premiumCommerceService, 92
  - premiumLockUntil, 92
  - premiumOrderNumber, 92
  - premiumServiceSKU, 92
  - premiumServiceStart, 92
  - premiumServiceStatus, 93
  - premiumSubscriptionNumber, 93
  - print, 90
  - unitDiscount, 93
  - unitPrice, 93
  - updated, 93
  - uploadLimitEnd, 93
  - uploadLimitNextMonth, 93
- qevercloud::AccountLimits, 94
  - localData, 95
  - noteResourceCountMax, 95
  - noteSizeMax, 95
  - noteTagCountMax, 95
  - operator!=, 94
  - operator==, 95
  - print, 95
  - resourceSizeMax, 96
  - uploadLimit, 96
  - userLinkedNotebookMax, 96
  - userMailLimitDaily, 96
  - userNotebookCountMax, 96
  - userNoteCountMax, 96
  - userSavedSearchesMax, 96
  - userTagCountMax, 97
- qevercloud::AsyncResult, 98
  - ~AsyncResult, 100
  - asIs, 100
  - AsyncResult, 99, 100
  - DurableService, 101
  - finished, 100
  - ReadFunctionType, 99
  - waitForFinished, 101
- qevercloud::AuthenticationResult, 101
  - authenticationToken, 103
  - currentTime, 103
  - expiration, 103
  - localData, 103
  - noteStoreUrl, 103
  - operator!=, 102
  - operator==, 102
  - print, 102
  - publicUserInfo, 103
  - secondFactorDeliveryHint, 103
  - secondFactorRequired, 104
  - urls, 104
  - user, 104
  - webApiUrlPrefix, 104
- qevercloud::BootstrapInfo, 104
  - localData, 105
  - operator!=, 105
  - operator==, 105
  - print, 105
  - profiles, 106
- qevercloud::BootstrapProfile, 106
  - localData, 107
  - name, 107
  - operator!=, 106
  - operator==, 107
  - print, 107
  - settings, 107
- qevercloud::BootstrapSettings, 108
  - accountEmailDomain, 109
  - enableFacebookSharing, 109
  - enableGiftSubscriptions, 109
  - enableGoogle, 109
  - enableLinkedInSharing, 109
  - enablePublicNotebooks, 110
  - enableSharedNotebooks, 110
  - enableSingleNoteSharing, 110
  - enableSponsoredAccounts, 110
  - enableSupportTickets, 110
  - enableTwitterSharing, 110
  - localData, 110
  - marketingUrl, 110
  - operator!=, 108



- operator==, 108
- print, 109
- serviceHost, 111
- supportUrl, 111
- qevercloud::BusinessInvitation, 111
  - businessId, 112
  - created, 112
  - email, 112
  - fromWorkChat, 113
  - localData, 113
  - mostRecentReminder, 113
  - operator!=, 112
  - operator==, 112
  - print, 112
  - requesterId, 113
  - role, 113
  - status, 113
- qevercloud::BusinessNotebook, 114
  - localData, 115
  - notebookDescription, 115
  - operator!=, 114
  - operator==, 114
  - print, 114
  - privilege, 115
  - recommended, 115
- qevercloud::BusinessUserAttributes, 115
  - companyStartDate, 117
  - department, 117
  - linkedinProfileUrl, 117
  - localData, 117
  - location, 117
  - mobilePhone, 117
  - operator!=, 116
  - operator==, 116
  - print, 116
  - title, 117
  - workPhone, 117
- qevercloud::BusinessUserInfo, 118
  - businessId, 119
  - businessName, 119
  - email, 119
  - localData, 119
  - operator!=, 118
  - operator==, 118
  - print, 119
  - role, 119
  - updated, 120
- qevercloud::CanMoveToContainerRestrictions, 120
  - canMoveToContainer, 121
  - localData, 121
  - operator!=, 120
  - operator==, 121
  - print, 121
- qevercloud::Contact, 121
  - id, 123
  - localData, 123
  - messagingPermit, 123
  - messagingPermitExpires, 123
  - name, 123
  - operator!=, 122
  - operator==, 122
  - photoLastUpdated, 123
  - photoUrl, 123
  - print, 122
  - type, 124
- qevercloud::CreateOrUpdateNotebookSharesResult, 124
  - localData, 125
  - matchingShares, 125, 126
  - operator!=, 125
  - operator==, 125
  - print, 125
  - updateSequenceNum, 125
- qevercloud::Data, 126
  - body, 127
  - bodyHash, 127
  - localData, 127
  - operator!=, 127
  - operator==, 127
  - print, 127
  - size, 128
- qevercloud::EDAMInvalidContactsException, 128
  - ~EDAMInvalidContactsException, 129
  - contacts, 130
  - EDAMInvalidContactsException, 129
  - exceptionData, 129
  - operator!=, 130
  - operator==, 130
  - parameter, 130
  - print, 130
  - reasons, 131
  - what, 130
- qevercloud::EDAMInvalidContactsExceptionData, 131
  - EDAMInvalidContactsExceptionData, 132
  - m\_contacts, 132
  - m\_parameter, 132
  - m\_reasons, 132
  - throwException, 132
- qevercloud::EDAMNotFoundException, 133
  - ~EDAMNotFoundException, 134
  - EDAMNotFoundException, 134
  - exceptionData, 134
  - identifier, 135
  - key, 135
  - operator!=, 134
  - operator==, 134
  - print, 135
  - what, 135
- qevercloud::EDAMNotFoundExceptionData, 135
  - EDAMNotFoundExceptionData, 136
  - m\_identifier, 136
  - m\_key, 137
  - throwException, 136
- qevercloud::EDAMSystemException, 137
  - ~EDAMSystemException, 138
  - EDAMSystemException, 138

- errorCode, 139
  - exceptionData, 138
  - message, 139
  - operator!=, 139
  - operator==, 139
  - print, 139
  - rateLimitDuration, 140
  - what, 139
- qevercloud::EDAMSystemExceptionAuthExpired, 140
  - exceptionData, 140
- qevercloud::EDAMSystemExceptionAuthExpiredData, 141
  - EDAMSystemExceptionAuthExpiredData, 141
  - throwException, 142
- qevercloud::EDAMSystemExceptionData, 142
  - EDAMSystemExceptionData, 143
  - m\_errorCode, 143
  - m\_message, 143
  - m\_rateLimitDuration, 144
  - throwException, 143
- qevercloud::EDAMSystemExceptionRateLimitReached, 144
  - exceptionData, 144
- qevercloud::EDAMSystemExceptionRateLimitReachedData, 145
  - EDAMSystemExceptionRateLimitReachedData, 145
  - throwException, 146
- qevercloud::EDAMUserException, 146
  - ~EDAMUserException, 147
  - EDAMUserException, 147
  - errorCode, 148
  - exceptionData, 147
  - operator!=, 148
  - operator==, 148
  - parameter, 148
  - print, 148
  - what, 148
- qevercloud::EDAMUserExceptionData, 149
  - EDAMUserExceptionData, 149
  - m\_errorCode, 150
  - m\_parameter, 150
  - throwException, 150
- qevercloud::EventLoopFinisher, 150
  - ~EventLoopFinisher, 151
  - EventLoopFinisher, 151
  - stopEventLoop, 151
- qevercloud::EverCloudException, 151
  - ~EverCloudException, 152
  - EverCloudException, 152
  - exceptionData, 153
  - m\_error, 153
  - what, 153
- qevercloud::EverCloudExceptionData, 153
  - errorMessage, 155
  - EverCloudExceptionData, 155
  - throwException, 155
- qevercloud::EverCloudLocalData, 155
  - ~EverCloudLocalData, 157
  - bool, 157
  - Dict, 156
  - dict, 158
  - dirty, 158
  - EverCloudLocalData, 157
  - favorited, 158
  - id, 158
  - local, 159
  - operator!=, 157
  - operator==, 157
  - print, 157
- qevercloud::EvernoteException, 159
  - EvernoteException, 159, 160
  - exceptionData, 160
- qevercloud::EvernoteExceptionData, 160
  - EvernoteExceptionData, 161
  - throwException, 161
- qevercloud::EvernoteOAuthDialog, 161
  - ~EvernoteOAuthDialog, 163
  - EvernoteOAuthDialog, 162
  - exec, 163
  - isSucceeded, 163
  - oauthError, 163
  - OAuthResult, 162
  - oauthResult, 163
  - open, 164
  - setWebViewSizeHint, 164
- qevercloud::EvernoteOAuthWebView, 164
  - authenticate, 165
  - authenticationFailed, 166
  - authenticationFinished, 166
  - authenticationSucceeded, 166
  - EvernoteOAuthWebView, 165
  - isSucceeded, 166
  - oauthError, 166
  - oauthResult, 166
  - setSizeHint, 167
  - sizeHint, 167
- qevercloud::EvernoteOAuthWebView::OAuthResult, 399
  - authenticationToken, 400
  - cookies, 400
  - expires, 400
  - noteStoreUrl, 400
  - print, 399
  - shardId, 400
  - userId, 400
  - webApiUrlPrefix, 401
- qevercloud::Identity, 167
  - blocked, 168
  - contact, 168
  - deactivated, 168
  - eventId, 169
  - id, 169
  - localData, 169
  - operator!=, 168
  - operator==, 168
  - print, 168

- sameBusiness, 169
- userConnected, 169
- userId, 169
- qevercloud::IDurableService, 170
  - AsyncServiceCall, 170
  - executeAsyncRequest, 171
  - executeSyncRequest, 171
  - SyncResult, 170
  - SyncServiceCall, 170
- qevercloud::IDurableService::AsyncRequest, 97
  - AsyncRequest, 97
  - m\_call, 97
  - m\_description, 98
  - m\_name, 98
- qevercloud::IDurableService::SyncRequest, 469
  - m\_call, 470
  - m\_description, 470
  - m\_name, 470
  - SyncRequest, 469
- qevercloud::ILogger, 171
  - level, 171
  - log, 171
  - setLevel, 172
  - shouldLog, 172
- qevercloud::InkNoteImageDownloader, 172
  - ~InkNoteImageDownloader, 173
  - download, 174
  - InkNoteImageDownloader, 173
  - setAuthenticationToken, 174
  - setHeight, 175
  - setHost, 175
  - setShardId, 175
  - setWidth, 175
- qevercloud::INoteStore, 176
  - authenticateToSharedNote, 180
  - authenticateToSharedNoteAsync, 181
  - authenticateToSharedNotebook, 181
  - authenticateToSharedNotebookAsync, 182
  - copyNote, 182
  - copyNoteAsync, 183
  - createLinkedNotebook, 183
  - createLinkedNotebookAsync, 184
  - createNote, 184
  - createNoteAsync, 185
  - createNotebook, 186
  - createNotebookAsync, 187
  - createOrUpdateNotebookShares, 187
  - createOrUpdateNotebookSharesAsync, 188
  - createSearch, 188
  - createSearchAsync, 189
  - createTag, 189
  - createTagAsync, 190
  - deleteNote, 190
  - deleteNoteAsync, 191
  - emailNote, 191
  - emailNoteAsync, 192
  - expungeLinkedNotebook, 192
  - expungeLinkedNotebookAsync, 192
  - expungeNote, 193
  - expungeNoteAsync, 193
  - expungeNotebook, 193
  - expungeNotebookAsync, 194
  - expungeSearch, 194
  - expungeSearchAsync, 195
  - expungeTag, 195
  - expungeTagAsync, 196
  - findNoteCounts, 196
  - findNoteCountsAsync, 197
  - findNoteOffset, 197
  - findNoteOffsetAsync, 198
  - findNotesMetadata, 198
  - findNotesMetadataAsync, 199
  - findRelated, 199
  - findRelatedAsync, 200
  - getDefaultNotebook, 200
  - getDefaultNotebookAsync, 201
  - getFilteredSyncChunk, 201
  - getFilteredSyncChunkAsync, 201
  - getLinkedNotebookSyncChunk, 202
  - getLinkedNotebookSyncChunkAsync, 203
  - getLinkedNotebookSyncState, 203
  - getLinkedNotebookSyncStateAsync, 204
  - getNote, 204
  - getNoteApplicationData, 204
  - getNoteApplicationDataAsync, 204
  - getNoteApplicationDataEntry, 205
  - getNoteApplicationDataEntryAsync, 205
  - getNoteAsync, 205
  - getNotebook, 205
  - getNotebookAsync, 207
  - getNotebookShares, 207
  - getNotebookSharesAsync, 207
  - getNoteContent, 207
  - getNoteContentAsync, 208
  - getNoteSearchText, 208
  - getNoteSearchTextAsync, 209
  - getNoteTagNames, 209
  - getNoteTagNamesAsync, 209
  - getNoteVersion, 210
  - getNoteVersionAsync, 210
  - getNoteWithResultSpec, 211
  - getNoteWithResultSpecAsync, 211
  - getPublicNotebook, 211
  - getPublicNotebookAsync, 212
  - getResource, 212
  - getResourceAlternateData, 213
  - getResourceAlternateDataAsync, 213
  - getResourceApplicationData, 214
  - getResourceApplicationDataAsync, 214
  - getResourceApplicationDataEntry, 214
  - getResourceApplicationDataEntryAsync, 214
  - getResourceAsync, 215
  - getResourceAttributes, 215
  - getResourceAttributesAsync, 215
  - getResourceByHash, 215
  - getResourceByHashAsync, 216

- getResourceData, 216
- getResourceDataAsync, 217
- getResourceRecognition, 217
- getResourceRecognitionAsync, 218
- getResourceSearchText, 218
- getResourceSearchTextAsync, 219
- getSearch, 219
- getSearchAsync, 219
- getSharedNotebookByAuth, 219
- getSharedNotebookByAuthAsync, 220
- getSyncState, 220
- getSyncStateAsync, 220
- getTag, 220
- getTagAsync, 221
- INoteStore, 180
- listAccessibleBusinessNotebooks, 221
- listAccessibleBusinessNotebooksAsync, 221
- listLinkedNotebooks, 222
- listLinkedNotebooksAsync, 222
- listNotebooks, 222
- listNotebooksAsync, 222
- listNoteVersions, 222
- listNoteVersionsAsync, 223
- listSearches, 223
- listSearchesAsync, 223
- listSharedNotebooks, 223
- listSharedNotebooksAsync, 223
- listTags, 224
- listTagsAsync, 224
- listTagsByNotebook, 224
- listTagsByNotebookAsync, 224
- manageNotebookShares, 224
- manageNotebookSharesAsync, 225
- noteStoreUrl, 225
- setNoteApplicationDataEntry, 225
- setNoteApplicationDataEntryAsync, 225
- setNotebookRecipientSettings, 226
- setNotebookRecipientSettingsAsync, 227
- setNoteStoreUrl, 227
- setResourceApplicationDataEntry, 227
- setResourceApplicationDataEntryAsync, 227
- shareNote, 227
- shareNoteAsync, 228
- shareNotebook, 228
- shareNotebookAsync, 230
- stopSharingNote, 230
- stopSharingNoteAsync, 231
- unsetNoteApplicationDataEntry, 231
- unsetNoteApplicationDataEntryAsync, 231
- unsetResourceApplicationDataEntry, 231
- unsetResourceApplicationDataEntryAsync, 231
- untagAll, 232
- untagAllAsync, 232
- updateLinkedNotebook, 232
- updateLinkedNotebookAsync, 233
- updateNote, 233
- updateNoteAsync, 234
- updateNotebook, 234
- updateNotebookAsync, 235
- updateNoteIfUsnMatches, 236
- updateNoteIfUsnMatchesAsync, 236
- updateResource, 236
- updateResourceAsync, 237
- updateSearch, 237
- updateSearchAsync, 238
- updateSharedNotebook, 238
- updateSharedNotebookAsync, 238
- updateTag, 238
- updateTagAsync, 240
- qevercloud::InvitationShareRelationship, 240
  - displayName, 241
  - localData, 242
  - operator!=, 241
  - operator==, 241
  - print, 241
  - privilege, 242
  - recipientUserIdentity, 242
  - sharerUserId, 242
- qevercloud::IRequestContext, 242
  - ~IRequestContext, 243
  - authenticationToken, 243
  - clone, 243
  - cookies, 243
  - increaseRequestTimeoutExponentially, 243
  - maxRequestRetryCount, 243
  - maxRequestTimeout, 244
  - operator<<, 244
  - requestId, 244
  - requestTimeout, 244
- qevercloud::IRetryPolicy, 244
  - shouldRetry, 245
- qevercloud::IUserStore, 248
  - authenticateLongSession, 249
  - authenticateLongSessionAsync, 251
  - authenticateToBusiness, 251
  - authenticateToBusinessAsync, 252
  - checkVersion, 252
  - checkVersionAsync, 253
  - completeTwoFactorAuthentication, 253
  - completeTwoFactorAuthenticationAsync, 254
  - getAccountLimits, 254
  - getAccountLimitsAsync, 255
  - getBootstrapInfo, 255
  - getBootstrapInfoAsync, 255
  - getPublicUserInfo, 255
  - getPublicUserInfoAsync, 256
  - getUser, 256
  - getUserAsync, 256
  - getUserUrls, 256
  - getUserUrlsAsync, 256
  - inviteToBusiness, 256
  - inviteToBusinessAsync, 257
  - IUserStore, 249
  - listBusinessInvitations, 257
  - listBusinessInvitationsAsync, 258
  - listBusinessUsers, 258

- listBusinessUsersAsync, 259
- removeFromBusiness, 259
- removeFromBusinessAsync, 259
- revokeLongSession, 259
- revokeLongSessionAsync, 260
- setUserStoreUrl, 260
- updateBusinessUserIdentifier, 260
- updateBusinessUserIdentifierAsync, 261
- userStoreUrl, 261
- qevercloud::LazyMap, 262
  - FullMap, 263
  - fullMap, 263, 264
  - keysOnly, 263, 264
  - localData, 264
  - operator!=, 263
  - operator==, 263
  - print, 263
- qevercloud::LinkedNotebook, 264
  - businessId, 265
  - guid, 266
  - localData, 266
  - noteStoreUrl, 266
  - operator!=, 265
  - operator==, 265
  - print, 265
  - shardId, 266
  - sharedNotebookGlobalId, 266
  - shareName, 266
  - stack, 267
  - updateSequenceNum, 267
  - uri, 267
  - username, 267
  - webApiUrlPrefix, 267
- qevercloud::ManageNotebookSharesError, 268
  - localData, 269
  - notFoundException, 269
  - operator!=, 268
  - operator==, 268
  - print, 269
  - userException, 269
  - userIdentity, 269
- qevercloud::ManageNotebookSharesParameters, 270
  - invitationsToCreateOrUpdate, 271, 272
  - inviteMessage, 271
  - localData, 271
  - membershipsToUpdate, 271, 272
  - notebookGuid, 271
  - operator!=, 270
  - operator==, 270
  - print, 271
  - unshares, 272
- qevercloud::ManageNotebookSharesResult, 272
  - errors, 274
  - localData, 274
  - operator!=, 273
  - operator==, 273
  - print, 273
- qevercloud::ManageNoteSharesError, 274
  - identityID, 275
  - localData, 276
  - notFoundException, 276
  - operator!=, 275
  - operator==, 275
  - print, 275
  - userException, 276
  - userId, 276
- qevercloud::ManageNoteSharesParameters, 276
  - invitationsToUnshare, 278, 279
  - invitationsToUpdate, 278, 279
  - localData, 278
  - membershipsToUnshare, 278, 279
  - membershipsToUpdate, 278, 279
  - noteGuid, 278
  - operator!=, 277
  - operator==, 277
  - print, 277
- qevercloud::ManageNoteSharesResult, 279
  - errors, 281
  - localData, 281
  - operator!=, 280
  - operator==, 280
  - print, 280
- qevercloud::MemberShareRelationship, 281
  - bestPrivilege, 282
  - displayName, 283
  - individualPrivilege, 283
  - localData, 283
  - operator!=, 282
  - operator==, 282
  - print, 282
  - recipientUserId, 283
  - restrictions, 283
  - sharerUserId, 283
- qevercloud::NetworkException, 284
  - ~NetworkException, 285
  - exceptionData, 285
  - m\_type, 286
  - NetworkException, 284, 285
  - operator!=, 285
  - operator==, 285
  - type, 285
  - what, 286
- qevercloud::NetworkExceptionData, 286
  - m\_type, 287
  - NetworkExceptionData, 287
  - throwException, 287
- qevercloud::Note, 287
  - active, 289
  - attributes, 289
  - content, 289
  - contentHash, 289
  - contentLength, 289
  - created, 290
  - deleted, 290
  - guid, 290
  - limits, 290

- localData, 290
- notebookGuid, 290
- operator!=, 288
- operator==, 288
- print, 289
- resources, 291, 292
- restrictions, 291
- sharedNotes, 291, 292
- tagGuids, 291, 292
- tagNames, 291
- title, 291
- updated, 292
- updateSequenceNum, 292
- qevercloud::NoteAttributes, 293
  - altitude, 294
  - applicationData, 295
  - author, 295
  - Classifications, 294
  - classifications, 295, 299
  - conflictSourceNoteGuid, 295
  - contentClass, 295
  - creatorId, 296
  - lastEditedBy, 296
  - lastEditorId, 296
  - latitude, 296
  - localData, 296
  - longitude, 297
  - noteTitleQuality, 297
  - operator!=, 294
  - operator==, 294
  - placeName, 297
  - print, 294
  - reminderDoneTime, 297
  - reminderOrder, 297
  - reminderTime, 298
  - shareDate, 298
  - sharedWithBusiness, 298
  - source, 298
  - sourceApplication, 298
  - sourceURL, 299
  - subjectDate, 299
- qevercloud::Notebook, 299
  - businessNotebook, 301
  - contact, 301
  - defaultNotebook, 301
  - guid, 301
  - localData, 301
  - name, 302
  - operator!=, 300
  - operator==, 300
  - print, 300
  - published, 302
  - publishing, 302
  - recipientSettings, 302
  - restrictions, 302
  - serviceCreated, 302
  - serviceUpdated, 303
  - sharedNotebookIds, 303, 304
  - sharedNotebooks, 303, 304
  - stack, 303
  - updateSequenceNum, 303
- qevercloud::NotebookDescriptor, 304
  - contactName, 305
  - guid, 305
  - hasSharedNotebook, 305
  - joinedUserCount, 305
  - localData, 306
  - notebookDisplayName, 306
  - operator!=, 305
  - operator==, 305
  - print, 305
- qevercloud::NotebookRecipientSettings, 306
  - inMyList, 307
  - localData, 307
  - operator!=, 307
  - operator==, 307
  - print, 307
  - recipientStatus, 308
  - reminderNotifyEmail, 308
  - reminderNotifyInApp, 308
  - stack, 308
- qevercloud::NotebookRestrictions, 308
  - canMoveToContainerRestrictions, 310
  - expungeWhichSharedNotebookRestrictions, 310
  - localData, 310
  - noCanMoveNote, 311
  - noChangeContact, 311
  - noCreateNotes, 311
  - noCreateSharedNotebooks, 311
  - noCreateTags, 311
  - noEmailNotes, 311
  - noExpungeNotebook, 311
  - noExpungeNotes, 312
  - noExpungeTags, 312
  - noPublishToBusinessLibrary, 312
  - noPublishToPublic, 312
  - noReadNotes, 312
  - noRenameNotebook, 312
  - noSendMessageToRecipients, 312
  - noSetDefaultNotebook, 313
  - noSetInMyList, 313
  - noSetNotebookStack, 313
  - noSetParentTag, 313
  - noSetRecipientSettingsStack, 313
  - noSetReminderNotifyEmail, 313
  - noSetReminderNotifyInApp, 313
  - noShareNotes, 314
  - noShareNotesWithBusiness, 314
  - noUpdateNotebook, 314
  - noUpdateNotes, 314
  - noUpdateTags, 314
  - operator!=, 310
  - operator==, 310
  - print, 310
  - updateWhichSharedNotebookRestrictions, 314
- qevercloud::NotebookShareTemplate, 315

- localData, 316
- notebookGuid, 316
- operator!=, 315
- operator==, 315
- print, 316
- privilege, 316
- recipientContacts, 316, 317
- recipientThreadId, 316
- qevercloud::NoteCollectionCounts, 317
  - localData, 319
  - notebookCounts, 319
  - operator!=, 318
  - operator==, 318
  - print, 318
  - TagCounts, 318
  - tagCounts, 319
  - trashCount, 319
- qevercloud::NoteEmailParameters, 320
  - ccAddresses, 321
  - guid, 321
  - localData, 321
  - message, 321
  - note, 321
  - operator!=, 320
  - operator==, 320
  - print, 321
  - subject, 322
  - toAddresses, 322
- qevercloud::NoteFilter, 322
  - ascending, 324
  - context, 324
  - emphasized, 324
  - inactive, 324
  - includeAllReadableNotebooks, 324
  - includeAllReadableWorkspaces, 324
  - localData, 325
  - notebookGuid, 325
  - operator!=, 323
  - operator==, 323
  - order, 325
  - print, 323
  - rawWords, 325
  - searchContextBytes, 325
  - tagGuids, 325, 326
  - timeZone, 325
  - words, 326
- qevercloud::NoteInvitationShareRelationship, 326
  - displayName, 327
  - localData, 327
  - operator!=, 327
  - operator==, 327
  - print, 327
  - privilege, 327
  - recipientIdentityId, 328
  - sharerUserId, 328
- qevercloud::NoteLimits, 328
  - localData, 329
  - noteResourceCountMax, 329
  - noteSizeMax, 330
  - operator!=, 329
  - operator==, 329
  - print, 329
  - resourceSizeMax, 330
  - uploaded, 330
  - uploadLimit, 330
- qevercloud::NoteList, 330
  - debugInfo, 331
  - localData, 332
  - notes, 332
  - operator!=, 331
  - operator==, 331
  - print, 331
  - searchContextBytes, 332
  - searchedWords, 332
  - startIndex, 332
  - stoppedWords, 332
  - totalNotes, 332
  - updateCount, 333
- qevercloud::NoteMemberShareRelationship, 333
  - displayName, 334
  - localData, 334
  - operator!=, 334
  - operator==, 334
  - print, 334
  - privilege, 334
  - recipientUserId, 334
  - restrictions, 335
  - sharerUserId, 335
- qevercloud::NoteMetadata, 335
  - attributes, 336
  - contentLength, 336
  - created, 337
  - deleted, 337
  - guid, 337
  - largestResourceMime, 337
  - largestResourceSize, 337
  - localData, 337
  - notebookGuid, 337
  - operator!=, 336
  - operator==, 336
  - print, 336
  - tagGuids, 338
  - title, 338
  - updated, 338
  - updateSequenceNum, 338
- qevercloud::NoteRestrictions, 338
  - localData, 340
  - noEmail, 340
  - noShare, 340
  - noSharePublicly, 340
  - noUpdateContent, 341
  - noUpdateTitle, 341
  - operator!=, 339
  - operator==, 340
  - print, 340
- qevercloud::NoteResultSpec, 341



- includeAccountLimits, 342
- includeContent, 342
- includeNoteAppDataValues, 343
- includeResourceAppDataValues, 343
- includeResourcesAlternateData, 343
- includeResourcesData, 343
- includeResourcesRecognition, 343
- includeSharedNotes, 343
- localData, 343
- operator!=, 342
- operator==, 342
- print, 342
- qevercloud::NoteShareRelationshipRestrictions, 344
  - localData, 345
  - noSetFullAccess, 345
  - noSetModifyNote, 345
  - noSetReadNote, 345
  - operator!=, 344
  - operator==, 344
  - print, 345
- qevercloud::NoteShareRelationships, 346
  - invitationRestrictions, 347
  - invitations, 347, 348
  - localData, 347
  - memberships, 347, 348
  - operator!=, 346
  - operator==, 346
  - print, 347
- qevercloud::NotesMetadataList, 348
  - debugInfo, 349
  - localData, 349
  - notes, 349
  - operator!=, 349
  - operator==, 349
  - print, 349
  - searchContextBytes, 350
  - searchedWords, 350
  - startIndex, 350
  - stoppedWords, 350
  - totalNotes, 350
  - updateCount, 350
- qevercloud::NotesMetadataResultSpec, 351
  - includeAttributes, 352
  - includeContentLength, 352
  - includeCreated, 352
  - includeDeleted, 352
  - includeLargestResourceMime, 353
  - includeLargestResourceSize, 353
  - includeNotebookGuid, 353
  - includeTagGuids, 353
  - includeTitle, 353
  - includeUpdated, 353
  - includeUpdateSequenceNum, 353
  - localData, 353
  - operator!=, 351
  - operator==, 352
  - print, 352
- qevercloud::NoteStoreServer, 354
  - authenticateToSharedNotebookRequest, 359
  - authenticateToSharedNotebookRequestReady, 360
  - authenticateToSharedNoteRequest, 360
  - authenticateToSharedNoteRequestReady, 360
  - copyNoteRequest, 360
  - copyNoteRequestReady, 360
  - createLinkedNotebookRequest, 360
  - createLinkedNotebookRequestReady, 361
  - createNotebookRequest, 361
  - createNotebookRequestReady, 361
  - createNoteRequest, 361
  - createNoteRequestReady, 361
  - createOrUpdateNotebookSharesRequest, 361
  - createOrUpdateNotebookSharesRequestReady, 362
  - createSearchRequest, 362
  - createSearchRequestReady, 362
  - createTagRequest, 362
  - createTagRequestReady, 362
  - deleteNoteRequest, 362
  - deleteNoteRequestReady, 363
  - emailNoteRequest, 363
  - emailNoteRequestReady, 363
  - expungeLinkedNotebookRequest, 363
  - expungeLinkedNotebookRequestReady, 363
  - expungeNotebookRequest, 363
  - expungeNotebookRequestReady, 364
  - expungeNoteRequest, 364
  - expungeNoteRequestReady, 364
  - expungeSearchRequest, 364
  - expungeSearchRequestReady, 364
  - expungeTagRequest, 364
  - expungeTagRequestReady, 365
  - findNoteCountsRequest, 365
  - findNoteCountsRequestReady, 365
  - findNoteOffsetRequest, 365
  - findNoteOffsetRequestReady, 365
  - findNotesMetadataRequest, 365
  - findNotesMetadataRequestReady, 366
  - findRelatedRequest, 366
  - findRelatedRequestReady, 366
  - getDefaultNotebookRequest, 366
  - getDefaultNotebookRequestReady, 366
  - getFilteredSyncChunkRequest, 366
  - getFilteredSyncChunkRequestReady, 367
  - getLinkedNotebookSyncChunkRequest, 367
  - getLinkedNotebookSyncChunkRequestReady, 367
  - getLinkedNotebookSyncStateRequest, 367
  - getLinkedNotebookSyncStateRequestReady, 367
  - getNoteApplicationDataEntryRequest, 367
  - getNoteApplicationDataEntryRequestReady, 368
  - getNoteApplicationDataRequest, 368
  - getNoteApplicationDataRequestReady, 368
  - getNotebookRequest, 368
  - getNotebookRequestReady, 368
  - getNotebookSharesRequest, 368
  - getNotebookSharesRequestReady, 369



getNoteContentRequest, 369  
getNoteContentRequestReady, 369  
getNoteRequest, 369  
getNoteRequestReady, 369  
getNoteSearchTextRequest, 369  
getNoteSearchTextRequestReady, 370  
getNoteTagNamesRequest, 370  
getNoteTagNamesRequestReady, 370  
getNoteVersionRequest, 370  
getNoteVersionRequestReady, 370  
getNoteWithResultSpecRequest, 370  
getNoteWithResultSpecRequestReady, 371  
getPublicNotebookRequest, 371  
getPublicNotebookRequestReady, 371  
getResourceAlternateDataRequest, 371  
getResourceAlternateDataRequestReady, 371  
getResourceApplicationDataEntryRequest, 371  
getResourceApplicationDataEntryRequestReady, 372  
getResourceApplicationDataRequest, 372  
getResourceApplicationDataRequestReady, 372  
getResourceAttributesRequest, 372  
getResourceAttributesRequestReady, 372  
getResourceByHashRequest, 372  
getResourceByHashRequestReady, 373  
getResourceDataRequest, 373  
getResourceDataRequestReady, 373  
getResourceRecognitionRequest, 373  
getResourceRecognitionRequestReady, 373  
getResourceRequest, 373  
getResourceRequestReady, 374  
getResourceSearchTextRequest, 374  
getResourceSearchTextRequestReady, 374  
getSearchRequest, 374  
getSearchRequestReady, 374  
getSharedNotebookByAuthRequest, 374  
getSharedNotebookByAuthRequestReady, 375  
getSyncStateRequest, 375  
getSyncStateRequestReady, 375  
getTagRequest, 375  
getTagRequestReady, 375  
listAccessibleBusinessNotebooksRequest, 375  
listAccessibleBusinessNotebooksRequestReady, 375  
listLinkedNotebooksRequest, 376  
listLinkedNotebooksRequestReady, 376  
listNotebooksRequest, 376  
listNotebooksRequestReady, 376  
listNoteVersionsRequest, 376  
listNoteVersionsRequestReady, 376  
listSearchesRequest, 376  
listSearchesRequestReady, 377  
listSharedNotebooksRequest, 377  
listSharedNotebooksRequestReady, 377  
listTagsByNotebookRequest, 377  
listTagsByNotebookRequestReady, 377  
listTagsRequest, 377  
listTagsRequestReady, 377  
manageNotebookSharesRequest, 378  
manageNotebookSharesRequestReady, 378  
NoteStoreServer, 359  
onAuthenticateToSharedNotebookRequestReady, 378  
onAuthenticateToSharedNoteRequestReady, 378  
onCopyNoteRequestReady, 378  
onCreateLinkedNotebookRequestReady, 378  
onCreateNotebookRequestReady, 379  
onCreateNoteRequestReady, 379  
onCreateOrUpdateNotebookSharesRequestReady, 379  
onCreateSearchRequestReady, 379  
onCreateTagRequestReady, 379  
onDeleteNoteRequestReady, 379  
onEmailNoteRequestReady, 380  
onExpungeLinkedNotebookRequestReady, 380  
onExpungeNotebookRequestReady, 380  
onExpungeNoteRequestReady, 380  
onExpungeSearchRequestReady, 380  
onExpungeTagRequestReady, 380  
onFindNoteCountsRequestReady, 381  
onFindNoteOffsetRequestReady, 381  
onFindNotesMetadataRequestReady, 381  
onFindRelatedRequestReady, 381  
onGetDefaultNotebookRequestReady, 381  
onGetFilteredSyncChunkRequestReady, 381  
onGetLinkedNotebookSyncChunkRequestReady, 382  
onGetLinkedNotebookSyncStateRequestReady, 382  
onGetNoteApplicationDataEntryRequestReady, 382  
onGetNoteApplicationDataRequestReady, 382  
onGetNotebookRequestReady, 382  
onGetNotebookSharesRequestReady, 382  
onGetNoteContentRequestReady, 383  
onGetNoteRequestReady, 383  
onGetNoteSearchTextRequestReady, 383  
onGetNoteTagNamesRequestReady, 383  
onGetNoteVersionRequestReady, 383  
onGetNoteWithResultSpecRequestReady, 383  
onGetPublicNotebookRequestReady, 384  
onGetResourceAlternateDataRequestReady, 384  
onGetResourceApplicationDataEntryRequestReady, 384  
onGetResourceApplicationDataRequestReady, 384  
onGetResourceAttributesRequestReady, 384  
onGetResourceByHashRequestReady, 384  
onGetResourceDataRequestReady, 385  
onGetResourceRecognitionRequestReady, 385  
onGetResourceRequestReady, 385  
onGetResourceSearchTextRequestReady, 385  
onGetSearchRequestReady, 385  
onGetSharedNotebookByAuthRequestReady, 385  
onGetSyncStateRequestReady, 386  
onGetTagRequestReady, 386

- onListAccessibleBusinessNotebooksRequestReady, 386
- onListLinkedNotebooksRequestReady, 386
- onListNotebooksRequestReady, 386
- onListNoteVersionsRequestReady, 386
- onListSearchesRequestReady, 387
- onListSharedNotebooksRequestReady, 387
- onListTagsByNotebookRequestReady, 387
- onListTagsRequestReady, 387
- onManageNotebookSharesRequestReady, 387
- onRequest, 387
- onSetNoteApplicationDataEntryRequestReady, 388
- onSetNotebookRecipientSettingsRequestReady, 388
- onSetResourceApplicationDataEntryRequestReady, 388
- onShareNotebookRequestReady, 388
- onShareNoteRequestReady, 388
- onStopSharingNoteRequestReady, 388
- onUnsetNoteApplicationDataEntryRequestReady, 389
- onUnsetResourceApplicationDataEntryRequestReady, 389
- onUntagAllRequestReady, 389
- onUpdateLinkedNotebookRequestReady, 389
- onUpdateNotebookRequestReady, 389
- onUpdateNoteIfUsnMatchesRequestReady, 389
- onUpdateNoteRequestReady, 390
- onUpdateResourceRequestReady, 390
- onUpdateSearchRequestReady, 390
- onUpdateSharedNotebookRequestReady, 390
- onUpdateTagRequestReady, 390
- setNoteApplicationDataEntryRequest, 390
- setNoteApplicationDataEntryRequestReady, 391
- setNotebookRecipientSettingsRequest, 391
- setNotebookRecipientSettingsRequestReady, 391
- setResourceApplicationDataEntryRequest, 391
- setResourceApplicationDataEntryRequestReady, 391
- shareNotebookRequest, 391
- shareNotebookRequestReady, 392
- shareNoteRequest, 392
- shareNoteRequestReady, 392
- stopSharingNoteRequest, 392
- stopSharingNoteRequestReady, 392
- unsetNoteApplicationDataEntryRequest, 392
- unsetNoteApplicationDataEntryRequestReady, 393
- unsetResourceApplicationDataEntryRequest, 393
- unsetResourceApplicationDataEntryRequestReady, 393
- untagAllRequest, 393
- untagAllRequestReady, 393
- updateLinkedNotebookRequest, 393
- updateLinkedNotebookRequestReady, 394
- updateNotebookRequest, 394
- updateNotebookRequestReady, 394
- updateNoteIfUsnMatchesRequest, 394
- updateNoteIfUsnMatchesRequestReady, 394
- updateNoteRequest, 394
- updateNoteRequestReady, 395
- updateResourceRequest, 395
- updateResourceRequestReady, 395
- updateSearchRequest, 395
- updateSearchRequestReady, 395
- updateSharedNotebookRequest, 395
- updateSharedNotebookRequestReady, 396
- updateTagRequest, 396
- updateTagRequestReady, 396
- qevercloud::NoteVersionId, 396
- lastEditorId, 397
- localData, 398
- operator!=, 397
- operator==, 397
- print, 397
- saved, 398
- title, 398
- updated, 398
- updateSequenceNum, 398
- qevercloud::Optional< T >, 401
- clear, 403
- init, 404
- isEqual, 404
- isSet, 404
- operator const T &, 404
- operator T&, 405
- operator!=, 405
- operator->, 405
- operator=, 406
- operator==, 407
- Optional, 402, 403, 408
- ref, 407
- swap, 408
- value, 407
- qevercloud::Printable, 408
- ~Printable, 409
- operator<<, 410, 411
- print, 410
- Printable, 409
- toString, 410
- qevercloud::PublicUserInfo, 411
- localData, 412
- noteStoreUrl, 412
- operator!=, 412
- operator==, 412
- print, 412
- serviceLevel, 412
- userId, 413
- username, 413
- webApiUrlPrefix, 413
- qevercloud::Publishing, 413
- ascending, 414
- localData, 414
- operator!=, 414
- operator==, 414

- order, [415](#)
- print, [414](#)
- publicDescription, [415](#)
- uri, [415](#)
- qevercloud::QAssociativeContainerConstReferenceWrapper< Container >, [415](#)
- begin, [416](#)
- end, [416](#)
- QAssociativeContainerConstReferenceWrapper, [416](#)
- qevercloud::QAssociativeContainerConstReferenceWrapper< Container >::iterator, [245](#)
- iterator, [245](#)
- m\_iterator, [246](#)
- operator!=, [245](#)
- operator\*, [246](#)
- operator++, [246](#)
- qevercloud::QAssociativeContainerReferenceWrapper< Container >, [416](#)
- begin, [417](#)
- end, [417](#)
- QAssociativeContainerReferenceWrapper, [417](#)
- qevercloud::QAssociativeContainerReferenceWrapper< Container >::iterator, [246](#)
- iterator, [247](#)
- m\_iterator, [247](#)
- operator!=, [247](#)
- operator\*, [247](#)
- operator++, [247](#)
- qevercloud::RelatedContent, [417](#)
- accessType, [419](#)
- authors, [419](#)
- clipUrl, [419](#)
- contact, [419](#)
- contentId, [419](#)
- contentType, [419](#)
- date, [419](#)
- localData, [420](#)
- operator!=, [418](#)
- operator==, [418](#)
- print, [418](#)
- sourceFaviconUrl, [420](#)
- sourceId, [420](#)
- sourceName, [420](#)
- sourceUrl, [420](#)
- teaser, [420](#)
- thumbnails, [420](#), [421](#)
- title, [420](#)
- url, [421](#)
- visibleUrl, [421](#)
- qevercloud::RelatedContentImage, [421](#)
- fileSize, [422](#)
- height, [423](#)
- localData, [423](#)
- operator!=, [422](#)
- operator==, [422](#)
- pixelRatio, [423](#)
- print, [422](#)
- url, [423](#)
- width, [423](#)
- qevercloud::RelatedQuery, [423](#)
- cacheKey, [425](#)
- context, [425](#)
- filter, [425](#)
- localData, [425](#)
- noteGuid, [425](#)
- operator!=, [424](#)
- operator==, [424](#)
- plainText, [425](#)
- print, [424](#)
- referenceUri, [425](#)
- qevercloud::RelatedResult, [426](#)
- cacheExpires, [427](#)
- cacheKey, [427](#)
- containingNotebooks, [428](#), [429](#)
- debugInfo, [428](#)
- experts, [428](#), [429](#)
- localData, [428](#)
- notebooks, [429](#)
- notes, [429](#), [430](#)
- operator!=, [427](#)
- operator==, [427](#)
- print, [427](#)
- relatedContent, [429](#), [430](#)
- tags, [429](#), [430](#)
- qevercloud::RelatedResultSpec, [430](#)
- includeContainingNotebooks, [432](#)
- includeDebugInfo, [432](#)
- localData, [432](#)
- maxExperts, [432](#)
- maxNotebooks, [432](#)
- maxNotes, [432](#)
- maxRelatedContent, [432](#)
- maxTags, [433](#)
- operator!=, [431](#)
- operator==, [431](#)
- print, [431](#)
- relatedContentTypes, [433](#)
- writableNotebooksOnly, [433](#)
- qevercloud::Resource, [433](#)
- active, [435](#)
- alternateData, [435](#)
- attributes, [435](#)
- data, [435](#)
- duration, [435](#)
- guid, [435](#)
- height, [436](#)
- localData, [436](#)
- mime, [436](#)
- noteGuid, [436](#)
- operator!=, [434](#)
- operator==, [434](#)
- print, [434](#)
- recognition, [436](#)
- updateSequenceNum, [436](#)
- width, [436](#)

- qevercloud::ResourceAttributes, 437
  - altitude, 438
  - applicationData, 438
  - attachment, 438
  - cameraMake, 439
  - cameraModel, 439
  - clientWillIndex, 439
  - fileName, 439
  - latitude, 439
  - localData, 439
  - longitude, 439
  - operator!=, 437
  - operator==, 438
  - print, 438
  - recoType, 440
  - sourceURL, 440
  - timestamp, 440
- qevercloud::SavedSearch, 440
  - format, 441
  - guid, 442
  - localData, 442
  - name, 442
  - operator!=, 441
  - operator==, 441
  - print, 441
  - query, 442
  - scope, 442
  - updateSequenceNum, 442
- qevercloud::SavedSearchScope, 443
  - includeAccount, 444
  - includeBusinessLinkedNotebooks, 444
  - includePersonalLinkedNotebooks, 444
  - localData, 444
  - operator!=, 443
  - operator==, 443
  - print, 444
- qevercloud::SharedNote, 445
  - localData, 446
  - operator!=, 445
  - operator==, 445
  - print, 446
  - privilege, 446
  - recipientIdentity, 446
  - serviceAssigned, 446
  - serviceCreated, 446
  - serviceUpdated, 447
  - sharerUserID, 447
- qevercloud::SharedNotebook, 447
  - email, 448
  - globalId, 448
  - id, 448
  - localData, 449
  - notebookGuid, 449
  - notebookModifiable, 449
  - operator!=, 448
  - operator==, 448
  - print, 448
  - privilege, 449
  - recipientIdentityId, 449
  - recipientSettings, 449
  - recipientUserId, 449
  - recipientUsername, 450
  - serviceAssigned, 450
  - serviceCreated, 450
  - serviceUpdated, 450
  - sharerUserId, 450
  - userId, 450
  - username, 451
- qevercloud::SharedNotebookRecipientSettings, 451
  - localData, 452
  - operator!=, 452
  - operator==, 452
  - print, 452
  - reminderNotifyEmail, 452
  - reminderNotifyInApp, 452
- qevercloud::SharedNoteTemplate, 453
  - localData, 454
  - noteGuid, 454
  - operator!=, 453
  - operator==, 454
  - print, 454
  - privilege, 454
  - recipientContacts, 454, 455
  - recipientThreadId, 455
- qevercloud::ShareRelationshipRestrictions, 455
  - localData, 456
  - noSetFullAccess, 456
  - noSetModify, 456
  - noSetReadOnly, 457
  - noSetReadPlusActivity, 457
  - operator!=, 456
  - operator==, 456
  - print, 456
- qevercloud::ShareRelationships, 457
  - invitationRestrictions, 458
  - invitations, 458, 459
  - localData, 459
  - memberships, 459
  - operator!=, 458
  - operator==, 458
  - print, 458
- qevercloud::SyncChunk, 459
  - chunkHighUSN, 461
  - currentTime, 461
  - expungedLinkedNotebooks, 461, 463
  - expungedNotebooks, 461, 463
  - expungedNotes, 462, 464
  - expungedSearches, 462, 464
  - expungedTags, 462, 464
  - linkedNotebooks, 462, 464
  - localData, 462
  - notebooks, 462, 464
  - notes, 462, 464
  - operator!=, 460
  - operator==, 461
  - print, 461

- resources, [463](#), [464](#)
- searches, [463](#), [464](#)
- tags, [463](#), [465](#)
- updateCount, [463](#)
- qevercloud::SyncChunkFilter, [465](#)
  - includeExpunged, [466](#)
  - includeLinkedNotebooks, [466](#)
  - includeNoteApplicationDataFullMap, [467](#)
  - includeNoteAttributes, [467](#)
  - includeNotebooks, [467](#)
  - includeNoteResourceApplicationDataFullMap, [467](#)
  - includeNoteResources, [467](#)
  - includeNotes, [467](#)
  - includeResourceApplicationDataFullMap, [467](#)
  - includeResources, [468](#)
  - includeSearches, [468](#)
  - includeSharedNotes, [468](#)
  - includeTags, [468](#)
  - localData, [468](#)
  - notebookGuids, [468](#), [469](#)
  - omitSharedNotebooks, [468](#)
  - operator!=, [466](#)
  - operator==, [466](#)
  - print, [466](#)
  - requireNoteContentClass, [469](#)
- qevercloud::SyncState, [470](#)
  - currentTime, [471](#)
  - fullSyncBefore, [471](#)
  - localData, [471](#)
  - operator!=, [471](#)
  - operator==, [471](#)
  - print, [471](#)
  - updateCount, [472](#)
  - uploaded, [472](#)
  - userLastUpdated, [472](#)
  - userMaxMessageEventId, [472](#)
- qevercloud::Tag, [473](#)
  - guid, [474](#)
  - localData, [474](#)
  - name, [474](#)
  - operator!=, [473](#)
  - operator==, [473](#)
  - parentGuid, [474](#)
  - print, [473](#)
  - updateSequenceNum, [474](#)
- qevercloud::ThriftException, [475](#)
  - ~ThriftException, [476](#)
  - BAD\_SEQUENCE\_ID, [476](#)
  - exceptionData, [477](#)
  - INTERNAL\_ERROR, [476](#)
  - INVALID\_DATA, [476](#)
  - INVALID\_MESSAGE\_TYPE, [476](#)
  - m\_type, [478](#)
  - MISSING\_RESULT, [476](#)
  - operator!=, [477](#)
  - operator<<, [477](#)
  - operator==, [477](#)
  - PROTOCOL\_ERROR, [476](#)
  - ThriftException, [476](#)
  - Type, [476](#)
  - type, [477](#)
  - UNKNOWN, [476](#)
  - UNKNOWN\_METHOD, [476](#)
  - what, [477](#)
  - WRONG\_METHOD\_NAME, [476](#)
- qevercloud::ThriftExceptionData, [478](#)
  - m\_type, [479](#)
  - ThriftExceptionData, [479](#)
  - throwException, [479](#)
- qevercloud::Thumbnail, [479](#)
  - ~Thumbnail, [481](#)
  - BMP, [481](#)
  - createPostRequest, [482](#)
  - download, [482](#)
  - downloadAsync, [482](#)
  - GIF, [481](#)
  - ImageType, [480](#)
  - JPEG, [481](#)
  - operator<<, [484](#)
  - PNG, [481](#)
  - setAuthenticationToken, [483](#)
  - setHost, [483](#)
  - setImageType, [483](#)
  - setShardId, [483](#)
  - setSize, [484](#)
  - Thumbnail, [481](#)
- qevercloud::UpdateNoteIfUsnMatchesResult, [484](#)
  - localData, [485](#)
  - note, [486](#)
  - operator!=, [485](#)
  - operator==, [485](#)
  - print, [485](#)
  - updated, [486](#)
- qevercloud::User, [486](#)
  - accounting, [488](#)
  - accountLimits, [488](#)
  - active, [488](#)
  - attributes, [488](#)
  - businessUserInfo, [488](#)
  - created, [488](#)
  - deleted, [488](#)
  - email, [488](#)
  - id, [489](#)
  - localData, [489](#)
  - name, [489](#)
  - operator!=, [487](#)
  - operator==, [487](#)
  - photoLastUpdated, [489](#)
  - photoUrl, [489](#)
  - print, [487](#)
  - privilege, [489](#)
  - serviceLevel, [490](#)
  - shardId, [490](#)
  - timezone, [490](#)
  - updated, [490](#)
  - username, [490](#)

- qevercloud::UserAttributes, 491
  - businessAddress, 492
  - clipFullPage, 492
  - comments, 492
  - dailyEmailLimit, 493
  - dateAgreedToTermsOfService, 493
  - defaultLatitude, 493
  - defaultLocationName, 493
  - defaultLongitude, 493
  - educationalDiscount, 493
  - emailAddressLastConfirmed, 493
  - emailOptOutDate, 494
  - groupName, 494
  - hideSponsorBilling, 494
  - incomingEmailAddress, 494
  - localData, 494
  - maxReferrals, 494
  - operator!=, 492
  - operator==, 492
  - optOutMachineLearning, 494
  - partnerEmailOptInDate, 495
  - passwordUpdated, 495
  - preactivation, 495
  - preferredCountry, 495
  - preferredLanguage, 495
  - print, 492
  - recentMailedAddresses, 495
  - recognitionLanguage, 495
  - referrerCode, 496
  - referralCount, 496
  - referralProof, 496
  - reminderEmailConfig, 496
  - salesforcePushEnabled, 496
  - sentEmailCount, 496
  - sentEmailDate, 496
  - shouldLogClientEvent, 497
  - twitterId, 497
  - twitterUserName, 497
  - useEmailAutoFiling, 497
  - viewedPromotions, 497
- qevercloud::UserIdentity, 497
  - localData, 499
  - longIdentifier, 499
  - operator!=, 498
  - operator==, 498
  - print, 498
  - stringIdentifier, 499
  - type, 499
- qevercloud::UserProfile, 499
  - attributes, 501
  - email, 501
  - id, 501
  - joined, 501
  - localData, 501
  - name, 501
  - operator!=, 500
  - operator==, 500
  - photoLastUpdated, 501
  - photoUrl, 501
  - print, 500
  - role, 502
  - status, 502
  - username, 502
- qevercloud::UserStoreServer, 502
  - authenticateLongSessionRequest, 504
  - authenticateLongSessionRequestReady, 504
  - authenticateToBusinessRequest, 505
  - authenticateToBusinessRequestReady, 505
  - checkVersionRequest, 505
  - checkVersionRequestReady, 505
  - completeTwoFactorAuthenticationRequest, 505
  - completeTwoFactorAuthenticationRequestReady, 505
  - getAccountLimitsRequest, 506
  - getAccountLimitsRequestReady, 506
  - getBootstrapInfoRequest, 506
  - getBootstrapInfoRequestReady, 506
  - getPublicUserInfoRequest, 506
  - getPublicUserInfoRequestReady, 506
  - getUserRequest, 507
  - getUserRequestReady, 507
  - getUserUrlsRequest, 507
  - getUserUrlsRequestReady, 507
  - inviteToBusinessRequest, 507
  - inviteToBusinessRequestReady, 507
  - listBusinessInvitationsRequest, 507
  - listBusinessInvitationsRequestReady, 508
  - listBusinessUsersRequest, 508
  - listBusinessUsersRequestReady, 508
  - onAuthenticateLongSessionRequestReady, 508
  - onAuthenticateToBusinessRequestReady, 508
  - onCheckVersionRequestReady, 508
  - onCompleteTwoFactorAuthenticationRequestReady, 509
  - onGetAccountLimitsRequestReady, 509
  - onGetBootstrapInfoRequestReady, 509
  - onGetPublicUserInfoRequestReady, 509
  - onGetUserRequestReady, 509
  - onGetUserUrlsRequestReady, 509
  - onInviteToBusinessRequestReady, 510
  - onListBusinessInvitationsRequestReady, 510
  - onListBusinessUsersRequestReady, 510
  - onRemoveFromBusinessRequestReady, 510
  - onRequest, 510
  - onRevokeLongSessionRequestReady, 510
  - onUpdateBusinessUserIdentifierRequestReady, 511
  - removeFromBusinessRequest, 511
  - removeFromBusinessRequestReady, 511
  - revokeLongSessionRequest, 511
  - revokeLongSessionRequestReady, 511
  - updateBusinessUserIdentifierRequest, 511
  - updateBusinessUserIdentifierRequestReady, 511
  - UserStoreServer, 504
- qevercloud::UserUrls, 512
  - localData, 513



- messageStoreUrl, 513
- noteStoreUrl, 513
- operator!=, 512
- operator==, 512
- print, 513
- userStoreUrl, 513
- userWebSocketUrl, 513
- utilityUrl, 514
- webApiUrlPrefix, 514
- QEVERCLOUD\_EXPORT
  - Export.h, 524
- QEverCloudOAuth.h, 632
- qHash
  - qevercloud, 54–57
- query
  - qevercloud::SavedSearch, 442
- QueryFormat
  - qevercloud, 38
- QUOTA\_REACHED
  - qevercloud, 35
- RATE\_LIMIT\_REACHED
  - qevercloud, 35
- rateLimitDuration
  - qevercloud::EDAMSystemException, 140
- rawWords
  - qevercloud::NoteFilter, 325
- READ\_NOTE
  - qevercloud, 43
- READ\_NOTEBOOK
  - qevercloud, 41, 43
- READ\_NOTEBOOK\_PLUS\_ACTIVITY
  - qevercloud, 41, 43
- ReadFunctionType
  - qevercloud::AsyncResult, 99
- README.md, 635
- reasons
  - qevercloud::EDAMInvalidContactsException, 131
- recentMailedAddresses
  - qevercloud::UserAttributes, 495
- recipientContacts
  - qevercloud::NotebookShareTemplate, 316, 317
  - qevercloud::SharedNoteTemplate, 454, 455
- recipientIdentity
  - qevercloud::SharedNote, 446
- recipientIdentityId
  - qevercloud::NoteInvitationShareRelationship, 328
  - qevercloud::SharedNotebook, 449
- recipientSettings
  - qevercloud::Notebook, 302
  - qevercloud::SharedNotebook, 449
- RecipientStatus
  - qevercloud, 38
- recipientStatus
  - qevercloud::NotebookRecipientSettings, 308
- recipientThreadId
  - qevercloud::NotebookShareTemplate, 316
  - qevercloud::SharedNoteTemplate, 455
- recipientUserId
  - qevercloud::MemberShareRelationship, 283
  - qevercloud::NoteMemberShareRelationship, 334
  - qevercloud::SharedNotebook, 449
- recipientUserIdentity
  - qevercloud::InvitationShareRelationship, 242
- recipientUsername
  - qevercloud::SharedNotebook, 450
- recognition
  - qevercloud::Resource, 436
- recognitionLanguage
  - qevercloud::UserAttributes, 495
- recommended
  - qevercloud::BusinessNotebook, 115
- recoType
  - qevercloud::ResourceAttributes, 440
- REDEEMED
  - qevercloud, 32
- ref
  - qevercloud::Optional< T >, 407
- REFERENCE\_MATERIAL
  - qevercloud, 39
- referenceUri
  - qevercloud::RelatedQuery, 425
- referrerCode
  - qevercloud::UserAttributes, 496
- referralCount
  - qevercloud::UserAttributes, 496
- referralProof
  - qevercloud::UserAttributes, 496
- relatedContent
  - qevercloud::RelatedResult, 429, 430
- RelatedContentAccess
  - qevercloud, 39
- RelatedContentType
  - qevercloud, 39
- relatedContentTypes
  - qevercloud::RelatedResultSpec, 433
- RELEVANCE
  - qevercloud, 37
- reminderDoneTime
  - qevercloud::NoteAttributes, 297
- ReminderEmailConfig
  - qevercloud, 39
- reminderEmailConfig
  - qevercloud::UserAttributes, 496
- reminderNotifyEmail
  - qevercloud::NotebookRecipientSettings, 308
  - qevercloud::SharedNotebookRecipientSettings, 452
- reminderNotifyInApp
  - qevercloud::NotebookRecipientSettings, 308
  - qevercloud::SharedNotebookRecipientSettings, 452
- reminderOrder
  - qevercloud::NoteAttributes, 297
- reminderTime
  - qevercloud::NoteAttributes, 298
- removeFromBusiness

- qevercloud::IUserStore, 259
- removeFromBusinessAsync
  - qevercloud::IUserStore, 259
- removeFromBusinessRequest
  - qevercloud::UserStoreServer, 511
- removeFromBusinessRequestReady
  - qevercloud::UserStoreServer, 511
- RequestContext.h, 632, 633
- REQUESTED
  - qevercloud, 32
- requesterId
  - qevercloud::BusinessInvitation, 113
- requestId
  - qevercloud::IRequestContext, 244
- requestTimeout
  - qevercloud::IRequestContext, 244
- requireNoteContentClass
  - qevercloud::SyncChunkFilter, 469
- resetEvernoteNetworkProxy
  - qevercloud, 57
- resources
  - qevercloud::Note, 291, 292
  - qevercloud::SyncChunk, 463, 464
- resourceSizeMax
  - qevercloud::AccountLimits, 96
  - qevercloud::NoteLimits, 330
- restrictions
  - qevercloud::MemberShareRelationship, 283
  - qevercloud::Note, 291
  - qevercloud::Notebook, 302
  - qevercloud::NoteMemberShareRelationship, 335
- revokeLongSession
  - qevercloud::IUserStore, 259
- revokeLongSessionAsync
  - qevercloud::IUserStore, 260
- revokeLongSessionRequest
  - qevercloud::UserStoreServer, 511
- revokeLongSessionRequestReady
  - qevercloud::UserStoreServer, 511
- role
  - qevercloud::BusinessInvitation, 113
  - qevercloud::BusinessUserInfo, 119
  - qevercloud::UserProfile, 502
- salesforcePushEnabled
  - qevercloud::UserAttributes, 496
- sameBusiness
  - qevercloud::Identity, 169
- saved
  - qevercloud::NoteVersionId, 398
- scope
  - qevercloud::SavedSearch, 442
- searchContextBytes
  - qevercloud::NoteFilter, 325
  - qevercloud::NoteList, 332
  - qevercloud::NotesMetadataList, 350
- searchedWords
  - qevercloud::NoteList, 332
  - qevercloud::NotesMetadataList, 350
- searches
  - qevercloud::SyncChunk, 463, 464
- secondFactorDeliveryHint
  - qevercloud::AuthenticationResult, 103
- secondFactorRequired
  - qevercloud::AuthenticationResult, 104
- SEND\_DAILY\_EMAIL
  - qevercloud, 40
- sentEmailCount
  - qevercloud::UserAttributes, 496
- sentEmailDate
  - qevercloud::UserAttributes, 496
- Servers.h, 548
- serviceAssigned
  - qevercloud::SharedNote, 446
  - qevercloud::SharedNotebook, 450
- serviceCreated
  - qevercloud::Notebook, 302
  - qevercloud::SharedNote, 446
  - qevercloud::SharedNotebook, 450
- serviceHost
  - qevercloud::BootstrapSettings, 111
- ServiceLevel
  - qevercloud, 40
- serviceLevel
  - qevercloud::PublicUserInfo, 412
  - qevercloud::User, 490
- Services.h, 560, 561
- serviceUpdated
  - qevercloud::Notebook, 303
  - qevercloud::SharedNote, 447
  - qevercloud::SharedNotebook, 450
- setAuthenticationToken
  - qevercloud::InkNoteImageDownloader, 174
  - qevercloud::Thumbnail, 483
- setEvernoteNetworkProxy
  - qevercloud, 58
- setHeight
  - qevercloud::InkNoteImageDownloader, 175
- setHost
  - qevercloud::InkNoteImageDownloader, 175
  - qevercloud::Thumbnail, 483
- setImageType
  - qevercloud::Thumbnail, 483
- setLevel
  - qevercloud::ILogger, 172
- setLogger
  - qevercloud, 58
- setNonceGenerator
  - qevercloud, 58
- setNoteApplicationDataEntry
  - qevercloud::INoteStore, 225
- setNoteApplicationDataEntryAsync
  - qevercloud::INoteStore, 225
- setNoteApplicationDataEntryRequest
  - qevercloud::NoteStoreServer, 390
- setNoteApplicationDataEntryRequestReady
  - qevercloud::NoteStoreServer, 391



- setNotebookRecipientSettings
  - qevercloud::INoteStore, 226
- setNotebookRecipientSettingsAsync
  - qevercloud::INoteStore, 227
- setNotebookRecipientSettingsRequest
  - qevercloud::NoteStoreServer, 391
- setNotebookRecipientSettingsRequestReady
  - qevercloud::NoteStoreServer, 391
- setNoteStoreUrl
  - qevercloud::INoteStore, 227
- setResourceApplicationDataEntry
  - qevercloud::INoteStore, 227
- setResourceApplicationDataEntryAsync
  - qevercloud::INoteStore, 227
- setResourceApplicationDataEntryRequest
  - qevercloud::NoteStoreServer, 391
- setResourceApplicationDataEntryRequestReady
  - qevercloud::NoteStoreServer, 391
- setShardId
  - qevercloud::InkNoteImageDownloader, 175
  - qevercloud::Thumbnail, 483
- setSize
  - qevercloud::Thumbnail, 484
- setSizeHint
  - qevercloud::EvernoteOAuthWebView, 167
- settings
  - qevercloud::BootstrapProfile, 107
- setUserStoreUrl
  - qevercloud::IUserStore, 260
- setWebViewSizeHint
  - qevercloud::EvernoteOAuthDialog, 164
- setWidth
  - qevercloud::InkNoteImageDownloader, 175
- SEXP
  - qevercloud, 38
- SHARD\_UNAVAILABLE
  - qevercloud, 35
- shardId
  - qevercloud::EvernoteOAuthWebView::OAuthResult, 400
  - qevercloud::LinkedNotebook, 266
  - qevercloud::User, 490
- shareDate
  - qevercloud::NoteAttributes, 298
- sharedNotebookGlobalId
  - qevercloud::LinkedNotebook, 266
- sharedNotebookIds
  - qevercloud::Notebook, 303, 304
- SharedNotebookInstanceRestrictions
  - qevercloud, 40
- SharedNotebookPrivilegeLevel
  - qevercloud, 40
- sharedNotebooks
  - qevercloud::Notebook, 303, 304
- SharedNotePrivilegeLevel
  - qevercloud, 41
- sharedNotes
  - qevercloud::Note, 291, 292
- sharedWithBusiness
  - qevercloud::NoteAttributes, 298
- shareName
  - qevercloud::LinkedNotebook, 266
- shareNote
  - qevercloud::INoteStore, 227
- shareNoteAsync
  - qevercloud::INoteStore, 228
- shareNotebook
  - qevercloud::INoteStore, 228
- shareNotebookAsync
  - qevercloud::INoteStore, 230
- shareNotebookRequest
  - qevercloud::NoteStoreServer, 391
- shareNotebookRequestReady
  - qevercloud::NoteStoreServer, 392
- shareNoteRequest
  - qevercloud::NoteStoreServer, 392
- shareNoteRequestReady
  - qevercloud::NoteStoreServer, 392
- ShareRelationshipPrivilegeLevel
  - qevercloud, 43
- sharerUserId
  - qevercloud::SharedNote, 447
- sharerUserId
  - qevercloud::InvitationShareRelationship, 242
  - qevercloud::MemberShareRelationship, 283
  - qevercloud::NoteInvitationShareRelationship, 328
  - qevercloud::NoteMemberShareRelationship, 335
  - qevercloud::SharedNotebook, 450
- shouldLog
  - qevercloud::ILogger, 172
- shouldLogClientEvent
  - qevercloud::UserAttributes, 497
- shouldRetry
  - qevercloud::IRetryPolicy, 245
- size
  - qevercloud::Data, 128
- sizeHint
  - qevercloud::EvernoteOAuthWebView, 167
- SMS
  - qevercloud, 33
- source
  - qevercloud::NoteAttributes, 298
- sourceApplication
  - qevercloud::NoteAttributes, 298
- sourceFaviconUrl
  - qevercloud::RelatedContent, 420
- sourceId
  - qevercloud::RelatedContent, 420
- sourceName
  - qevercloud::RelatedContent, 420
- sourceURL
  - qevercloud::NoteAttributes, 299
  - qevercloud::ResourceAttributes, 440
- sourceUrl
  - qevercloud::RelatedContent, 420
- SponsoredGroupRole

- qevercloud, [43](#)
- SSO\_AUTHENTICATION\_REQUIRED
  - qevercloud, [35](#)
- stack
  - qevercloud::LinkedNotebook, [267](#)
  - qevercloud::Notebook, [303](#)
  - qevercloud::NotebookRecipientSettings, [308](#)
- startIndex
  - qevercloud::NoteList, [332](#)
  - qevercloud::NotesMetadataList, [350](#)
- status
  - qevercloud::BusinessInvitation, [113](#)
  - qevercloud::UserProfile, [502](#)
- stopEventLoop
  - qevercloud::EventLoopFinisher, [151](#)
- stoppedWords
  - qevercloud::NoteList, [332](#)
  - qevercloud::NotesMetadataList, [350](#)
- stopSharingNote
  - qevercloud::INoteStore, [230](#)
- stopSharingNoteAsync
  - qevercloud::INoteStore, [231](#)
- stopSharingNoteRequest
  - qevercloud::NoteStoreServer, [392](#)
- stopSharingNoteRequestReady
  - qevercloud::NoteStoreServer, [392](#)
- stringIdentifier
  - qevercloud::UserIdentity, [499](#)
- subject
  - qevercloud::NoteEmailParameters, [322](#)
- subjectDate
  - qevercloud::NoteAttributes, [299](#)
- SUPPORT
  - qevercloud, [38](#)
- supportUrl
  - qevercloud::BootstrapSettings, [111](#)
- swap
  - qevercloud::Optional< T >, [408](#)
- SyncRequest
  - qevercloud::IDurableService::SyncRequest, [469](#)
- SyncResult
  - qevercloud::IDurableService, [170](#)
- SyncServiceCall
  - qevercloud::IDurableService, [170](#)
- TagCounts
  - qevercloud::NoteCollectionCounts, [318](#)
- tagCounts
  - qevercloud::NoteCollectionCounts, [319](#)
- tagGuids
  - qevercloud::Note, [291](#), [292](#)
  - qevercloud::NoteFilter, [325](#), [326](#)
  - qevercloud::NoteMetadata, [338](#)
- tagNames
  - qevercloud::Note, [291](#)
- tags
  - qevercloud::RelatedResult, [429](#), [430](#)
  - qevercloud::SyncChunk, [463](#), [465](#)
- TAKEN\_DOWN
  - qevercloud, [35](#)
- teaser
  - qevercloud::RelatedContent, [420](#)
- ThriftException
  - qevercloud::ThriftException, [476](#)
- ThriftExceptionData
  - qevercloud::ThriftExceptionData, [479](#)
- throwException
  - qevercloud::EDAMInvalidContactsExceptionData, [132](#)
  - qevercloud::EDAMNotFoundExceptionData, [136](#)
  - qevercloud::EDAMSystemExceptionAuthExpiredData, [142](#)
  - qevercloud::EDAMSystemExceptionData, [143](#)
  - qevercloud::EDAMSystemExceptionRateLimitReachedData, [146](#)
  - qevercloud::EDAMUserExceptionData, [150](#)
  - qevercloud::EverCloudExceptionData, [155](#)
  - qevercloud::EvernoteExceptionData, [161](#)
  - qevercloud::NetworkExceptionData, [287](#)
  - qevercloud::ThriftExceptionData, [479](#)
- Thumbnail
  - qevercloud::Thumbnail, [481](#)
- Thumbnail.h, [633](#), [634](#)
- thumbnails
  - qevercloud::RelatedContent, [420](#), [421](#)
- Timestamp
  - qevercloud, [31](#)
- timestamp
  - qevercloud::ResourceAttributes, [440](#)
- timeZone
  - qevercloud::NoteFilter, [325](#)
- timezone
  - qevercloud::User, [490](#)
- TITLE
  - qevercloud, [37](#)
- title
  - qevercloud::BusinessUserAttributes, [117](#)
  - qevercloud::Note, [291](#)
  - qevercloud::NoteMetadata, [338](#)
  - qevercloud::NoteVersionId, [398](#)
  - qevercloud::RelatedContent, [420](#)
- toAddresses
  - qevercloud::NoteEmailParameters, [322](#)
- TOO\_FEW
  - qevercloud, [35](#)
- TOO\_MANY
  - qevercloud, [35](#)
- toRange
  - qevercloud, [58](#), [59](#)
- toString
  - qevercloud::Printable, [410](#)
- totalNotes
  - qevercloud::NoteList, [332](#)
  - qevercloud::NotesMetadataList, [350](#)
- Trace
  - qevercloud, [36](#)
- trashCount

- qevercloud::NoteCollectionCounts, 319
- TWITTER
  - qevercloud, 33
- twitterId
  - qevercloud::UserAttributes, 497
- twitterUserName
  - qevercloud::UserAttributes, 497
- Type
  - qevercloud::ThriftException, 476
- type
  - qevercloud::Contact, 124
  - qevercloud::NetworkException, 285
  - qevercloud::ThriftException, 477
  - qevercloud::UserIdentity, 499
- Types.h, 571, 574
- unitDiscount
  - qevercloud::Accounting, 93
- unitPrice
  - qevercloud::Accounting, 93
- UNKNOWN
  - qevercloud, 35
  - qevercloud::ThriftException, 476
- UNKNOWN\_METHOD
  - qevercloud::ThriftException, 476
- unsetNoteApplicationDataEntry
  - qevercloud::INoteStore, 231
- unsetNoteApplicationDataEntryAsync
  - qevercloud::INoteStore, 231
- unsetNoteApplicationDataEntryRequest
  - qevercloud::NoteStoreServer, 392
- unsetNoteApplicationDataEntryRequestReady
  - qevercloud::NoteStoreServer, 393
- unsetResourceApplicationDataEntry
  - qevercloud::INoteStore, 231
- unsetResourceApplicationDataEntryAsync
  - qevercloud::INoteStore, 231
- unsetResourceApplicationDataEntryRequest
  - qevercloud::NoteStoreServer, 393
- unsetResourceApplicationDataEntryRequestReady
  - qevercloud::NoteStoreServer, 393
- unshares
  - qevercloud::ManageNotebookSharesParameters, 272
- UNSUPPORTED\_OPERATION
  - qevercloud, 35
- untagAll
  - qevercloud::INoteStore, 232
- untagAllAsync
  - qevercloud::INoteStore, 232
- untagAllRequest
  - qevercloud::NoteStoreServer, 393
- untagAllRequestReady
  - qevercloud::NoteStoreServer, 393
- UPDATE\_SEQUENCE\_NUMBER
  - qevercloud, 37
- updateBusinessUserIdentifier
  - qevercloud::IUserStore, 260
- updateBusinessUserIdentifierAsync
  - qevercloud::IUserStore, 261
- updateBusinessUserIdentifierRequest
  - qevercloud::UserStoreServer, 511
- updateBusinessUserIdentifierRequestReady
  - qevercloud::UserStoreServer, 511
- updateCount
  - qevercloud::NoteList, 333
  - qevercloud::NotesMetadataList, 350
  - qevercloud::SyncChunk, 463
  - qevercloud::SyncState, 472
- UPDATED
  - qevercloud, 37
- updated
  - qevercloud::Accounting, 93
  - qevercloud::BusinessUserInfo, 120
  - qevercloud::Note, 292
  - qevercloud::NoteMetadata, 338
  - qevercloud::NoteVersionId, 398
  - qevercloud::UpdateNoteIfUsnMatchesResult, 486
  - qevercloud::User, 490
- updateLinkedNotebook
  - qevercloud::INoteStore, 232
- updateLinkedNotebookAsync
  - qevercloud::INoteStore, 233
- updateLinkedNotebookRequest
  - qevercloud::NoteStoreServer, 393
- updateLinkedNotebookRequestReady
  - qevercloud::NoteStoreServer, 394
- updateNote
  - qevercloud::INoteStore, 233
- updateNoteAsync
  - qevercloud::INoteStore, 234
- updateNotebook
  - qevercloud::INoteStore, 234
- updateNotebookAsync
  - qevercloud::INoteStore, 235
- updateNotebookRequest
  - qevercloud::NoteStoreServer, 394
- updateNotebookRequestReady
  - qevercloud::NoteStoreServer, 394
- updateNoteIfUsnMatches
  - qevercloud::INoteStore, 236
- updateNoteIfUsnMatchesAsync
  - qevercloud::INoteStore, 236
- updateNoteIfUsnMatchesRequest
  - qevercloud::NoteStoreServer, 394
- updateNoteIfUsnMatchesRequestReady
  - qevercloud::NoteStoreServer, 394
- updateNoteRequest
  - qevercloud::NoteStoreServer, 394
- updateNoteRequestReady
  - qevercloud::NoteStoreServer, 395
- updateResource
  - qevercloud::INoteStore, 236
- updateResourceAsync
  - qevercloud::INoteStore, 237
- updateResourceRequest
  - qevercloud::NoteStoreServer, 395

- updateResourceRequestReady
  - qevercloud::NoteStoreServer, 395
- updateSearch
  - qevercloud::INoteStore, 237
- updateSearchAsync
  - qevercloud::INoteStore, 238
- updateSearchRequest
  - qevercloud::NoteStoreServer, 395
- updateSearchRequestReady
  - qevercloud::NoteStoreServer, 395
- updateSequenceNum
  - qevercloud::CreateOrUpdateNotebookSharesResult, 125
  - qevercloud::LinkedNotebook, 267
  - qevercloud::Note, 292
  - qevercloud::Notebook, 303
  - qevercloud::NoteMetadata, 338
  - qevercloud::NoteVersionId, 398
  - qevercloud::Resource, 436
  - qevercloud::SavedSearch, 442
  - qevercloud::Tag, 474
- updateSharedNotebook
  - qevercloud::INoteStore, 238
- updateSharedNotebookAsync
  - qevercloud::INoteStore, 238
- updateSharedNotebookRequest
  - qevercloud::NoteStoreServer, 395
- updateSharedNotebookRequestReady
  - qevercloud::NoteStoreServer, 396
- updateTag
  - qevercloud::INoteStore, 238
- updateTagAsync
  - qevercloud::INoteStore, 240
- updateTagRequest
  - qevercloud::NoteStoreServer, 396
- updateTagRequestReady
  - qevercloud::NoteStoreServer, 396
- updateWhichSharedNotebookRestrictions
  - qevercloud::NotebookRestrictions, 314
- uploaded
  - qevercloud::NoteLimits, 330
  - qevercloud::SyncState, 472
- uploadLimit
  - qevercloud::AccountLimits, 96
  - qevercloud::NoteLimits, 330
- uploadLimitEnd
  - qevercloud::Accounting, 93
- uploadLimitNextMonth
  - qevercloud::Accounting, 93
- uri
  - qevercloud::LinkedNotebook, 267
  - qevercloud::Publishing, 415
- url
  - qevercloud::RelatedContent, 421
  - qevercloud::RelatedContentImage, 423
- urls
  - qevercloud::AuthenticationResult, 104
- useEmailAutoFiling
  - qevercloud::UserAttributes, 497
- USER
  - qevercloud, 38
- user
  - qevercloud::AuthenticationResult, 104
- USER\_ALREADY\_ASSOCIATED
  - qevercloud, 35
- USER\_NOT\_ASSOCIATED
  - qevercloud, 35
- USER\_NOT\_REGISTERED
  - qevercloud, 35
- userConnected
  - qevercloud::Identity, 169
- userException
  - qevercloud::ManageNotebookSharesError, 269
  - qevercloud::ManageNoteSharesError, 276
- UserID
  - qevercloud, 31
- userID
  - qevercloud::ManageNoteSharesError, 276
- userId
  - qevercloud::EvernoteOAuthWebView::OAuthResult, 400
  - qevercloud::Identity, 169
  - qevercloud::PublicUserInfo, 413
  - qevercloud::SharedNotebook, 450
- userIdentity
  - qevercloud::ManageNotebookSharesError, 269
- UserIdentityType
  - qevercloud, 44
- userLastUpdated
  - qevercloud::SyncState, 472
- userLinkedNotebookMax
  - qevercloud::AccountLimits, 96
- userMailLimitDaily
  - qevercloud::AccountLimits, 96
- userMaxMessageEventId
  - qevercloud::SyncState, 472
- username
  - qevercloud::LinkedNotebook, 267
  - qevercloud::PublicUserInfo, 413
  - qevercloud::SharedNotebook, 451
  - qevercloud::User, 490
  - qevercloud::UserProfile, 502
- userNotebookCountMax
  - qevercloud::AccountLimits, 96
- userNoteCountMax
  - qevercloud::AccountLimits, 96
- userSavedSearchesMax
  - qevercloud::AccountLimits, 96
- UserStoreServer
  - qevercloud::UserStoreServer, 504
- userStoreUrl
  - qevercloud::IUserStore, 261
  - qevercloud::UserUrls, 513
- userTagCountMax
  - qevercloud::AccountLimits, 97
- userWebSocketUrl

- qevercloud::UserUrls, [513](#)
- utilityUrl
  - qevercloud::UserUrls, [514](#)
- value
  - qevercloud::Optional< T >, [407](#)
- viewedPromotions
  - qevercloud::UserAttributes, [497](#)
- VIP
  - qevercloud, [38](#)
- visibleUrl
  - qevercloud::RelatedContent, [421](#)
- waitForFinished
  - qevercloud::AsyncResult, [101](#)
- Warn
  - qevercloud, [36](#)
- webApiUrlPrefix
  - qevercloud::AuthenticationResult, [104](#)
  - qevercloud::EvernoteOAuthWebView::OAuthResult, [401](#)
  - qevercloud::LinkedNotebook, [267](#)
  - qevercloud::PublicUserInfo, [413](#)
  - qevercloud::UserUrls, [514](#)
- what
  - qevercloud::EDAMInvalidContactsException, [130](#)
  - qevercloud::EDAMNotFoundException, [135](#)
  - qevercloud::EDAMSystemException, [139](#)
  - qevercloud::EDAMUserException, [148](#)
  - qevercloud::EverCloudException, [153](#)
  - qevercloud::NetworkException, [286](#)
  - qevercloud::ThriftException, [477](#)
- width
  - qevercloud::RelatedContentImage, [423](#)
  - qevercloud::Resource, [436](#)
- words
  - qevercloud::NoteFilter, [326](#)
- workPhone
  - qevercloud::BusinessUserAttributes, [117](#)
- WORKSPACE
  - qevercloud, [36](#)
- writableNotebooksOnly
  - qevercloud::RelatedResultSpec, [433](#)
- WRONG\_METHOD\_NAME
  - qevercloud::ThriftException, [476](#)