FlexNet Publisher 2020 R2 (11.17.0)

C/C++ Function Reference

Legal Information

Book Name: FlexNet Publisher 2020 R2 (11.17.0) C/C++ Function Reference

Part Number: FNP-11170-CCFR00

Product Release Date: April 2020

Copyright Notice

Copyright © 2020 Flexera. All Rights Reserved.

This publication contains proprietary and confidential information and creative works owned by Flexera and its licensors, if any. Any use, copying, publication, distribution, display, modification, or transmission of such publication in whole or in part in any form or by any means without the prior express written permission of Flexera is strictly prohibited. Except where expressly provided by Flexera in writing, possession of this publication shall not be construed to confer any license or rights under any Flexera intellectual property rights, whether by estoppel, implication, or otherwise.

All copies of the technology and related information, if allowed by Flexera, must display this notice of copyright and ownership in full.

FlexNet Publisher incorporates software developed by others and redistributed according to license agreements. Copyright notices and licenses for these external libraries are provided in a supplementary document that accompanies this one.

Intellectual Property

For a list of trademarks and patents that are owned by Flexera, see https://www.flexerasoftware.com/producer/company/about/intellectual-property/. All other brand and product names mentioned in Flexera products, product documentation, and marketing materials are the trademarks and registered trademarks of their respective owners.

Restricted Rights Legend

The Software is commercial computer software. If the user or licensee of the Software is an agency, department, or other entity of the United States Government, the use, duplication, reproduction, release, modification, disclosure, or transfer of the Software, or any related documentation of any kind, including technical data and manuals, is restricted by a license agreement or by the terms of this Agreement in accordance with Federal Acquisition Regulation 12.212 for civilian purposes and Defense Federal Acquisition Regulation Supplement 227.7202 for military purposes. The Software was developed fully at private expense. All other use is prohibited.

Contents

1 Introduction . . . 15

Toolkit Functionality . . . 16

FlexNet Publisher Product Documentation . . . 16

2 Flexible API . . . 17

Required Flexible API Functions . . . 17

Flexible API Functions by Category . . . 18

Job Handling . . . 18

FlexNet Publisher and Multithreaded Applications . . . 19

License Checkin and Checkout . . . 19

Error and Warning Reporting . . . 20

Initialization Routines for Trusted Storage and SDTs . . . 20

Heartbeat Management and Communication . . . 21

Informational . . . 22

License File Installation . . . 23

License File Generation . . . 23

Flexible API Function Descriptions . . . 24

lc_auth_data . . . 25

lc_checkin . . . 28

lc_checkout . . . 29

lc_cryptstr . . . 38

lc_err_info . . . 42

lc_errstring . . . 44

lc_expire_days . . . 45

lc_feat_list . . . 46

lc_first_job . . . 47

lc_flexinit . . . 48

lc_flexinit_cleanup . . . 50

lc_flexinit_property_handle_create . . . 51

lc_flexinit_property_handle_free . . . 52
lc_flexinit_property_handle_get . . . 52
lc_flexinit_property_handle_set . . . 53
lc_fnpservice_present . . . 54
lc_free_job . . . 55
lc_get_attr . . . 56
lc_get_errno . . . 57
lc_get_version . . . 58
lc_get_server_version . . . 59
lc_heartbeat . . . 60
lc_hostid . . . 63
lc_idle . . . 67
lc_init . . . 68
lc_log . . . 70
lc_new_job . . . 71
lc_next_job . . . 74
lc_perror . . . 75
lc_set_attr . . . 76
lc_status . . . 78
lc_tpmstatusget (Windows only) . . . 79
lc_userlist . . . 81
lc_virtualstatusget . . . 82
lc_vsend . . . 83

3 Advanced Flexible API Functions . . . 87

Summary . . . 87
l_new_hostid . . . 88
lc_borrow_return . . . 88
lc_check_key . . . 90
lc_cleanup (Windows only) . . . 92
lc_free_hostid . . . 92
lc_get_config . . . 93
lc_init_simple_composite . . . 95
lc_next_conf . . . 97
lc_remove . . . 98
lc_set_attr_vendor . . . 100
lc_shutdown . . . 101
lc_test_conf . . . 102

4 Flexible API Attributes . . . 105

Summary of Flexible API Attributes . . . 106
LM_A_APP_DISABLE_CACHE_READ . . . 108
LM_A_BORROW_EXPIRE . . . 108
LM_A_BORROW_RETURN_BY_VERSION . . . 109

LM_A_BORROW_STAT 110
LM_A_CHECK_BADDATE 110
LM_A_CHECK_INTERVAL 111
LM_A_CKOUT_INSTALL_LIC 111
LM_A_CHECK_LOCAL_TS_SIGNATURE 112
LM_A_CONN_TIMEOUT 112
LM_A_FLEXID9_HASP4_SUPPORT 113
LM_A_LF_LIST 113
LM_A_LICENSE_DEFAULT 114
LM_A_LICENSE_FMT_VER 114
LM_A_LINGER 115
LM_A_LONG_ERRMSG 115
LM_A_PERROR_MSGBOX (Windows Only) 116
LM_A_PHYSICAL_ETHERNETID (Windows Only) 116
LM_A_PORT_HOST_PLUS 117
LM_A_PROMPT_FOR_FILE (Windows Only) 117
LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL 118
LM_A_SORT_TS_FIRST 118
LM_A_TS_ONLY 119
LM_A_TCP_TIMEOUT 120
LM_A_TS_CHECK_BADDATE 120
LM_A_USE_FQDN 121
LM_A_USER_EXITCALL, LM_A_USER_EXITCALL_EX 122
LM_A_USER_RECONNECT, LM_A_USER_RECONNECT_EX 123
LM_A_USER_RECONNECT_DONE, LM_A_USER_RECONNECT_DONE_EX 124
LM_A_USER_RECONNECT_V1* 125
LM_A_USER_RECONNECT_V1_DATA 125
LM_A_USER_RECONNECT_V1 126
LM_A_USER_RECONNECT_V1_DONE 127
LM_A_VD_GENERIC_INFO, LM_A_VD_FEATURE_INFO 128
LM_A_VENDOR_ID_DECLARE 130
LM_A_VM_FAMILY 131
LM_A_VM_NAME 131
LM_A_WINDOWS_MODULE_HANDLE (Windows only) 132
Determining License Source and Count of a Feature 133

5 Advanced Flexible API Attributes 135
Summary 135
LM_A_CHECKOUT_DATA 137

Incremental License Count in a Single Job . . . 137
Custom Duplicate Grouping . . . 138
User Visibility . . . 138
LM_A_MULTIPLE_CHECKOUT_DATA . . . 139
Bulk Custom Dub Checkout . . . 139
LM_A_CHECKOUTFILTER, LM_A_CHECKOUTFILTER_EX . . . 140
LM_A_CHECKOUTFILTERLAST_EX . . . 142
LM_A_CROSS_TZ_OK . . . 143
LM_A_DIAGS_ENABLED . . . 144
LM_A_DISABLE_ENV . . . 144
LM_A_DISPLAY_OVERRIDE . . . 144
LM_A_HOST_OVERRIDE . . . 145
LM_A_IGNORE_CACHED_SERVER_DATA . . . 146
LM_A_IGNORE_FILE_CHECK_IN_HEARTBEAT (Linux Only) . . . 146
LM_A_INTERNET_OVERRIDE . . . 146
LM_A_LICENSE_CASE_SENSITIVE . . . 147
LM_A_LKEY_CASE_SENSITIVE . . . 148
LM_A_MT_HEARTBEAT (UNIX Only) . . . 148
LM_A_PERIODIC_CALL . . . 149
LM_A_PERIODIC_COUNT . . . 149
LM_A_PLATFORM_OVERRIDE . . . 149
LM_A_RETRY_CHECKOUT . . . 150
LM_A_TZ_OVERRIDE . . . 150
LM_A_HEARTBEAT_LOCK (Windows only) . . . 151
LM_A_USER_OVERRIDE . . . 151
LM_A_VENDOR_CALLBACK_DATA . . . 152
LM_A_VSEND_NOWAIT . . . 152

6 Activation Borrow API . . . 153
Activation Borrow Functions . . . 153
flxActBorrowActivate . . . 154
flxActBorrowReturn . . . 156
flxActBorrowTSViewCreate . . . 157
flxActBorrowTSViewFRAttributeGet . . . 158
flxActBorrowTSViewDelete . . . 160

7 FlexNet Licensing Service API . . . 163
Installing the FlexNet Licensing Service on Windows . . . 163
Installing the FlexNet Licensing Service on Linux . . . 164
Installing the FlexNet Licensing Service on OS X . . . 166
FlexNet Licensing Service Functions . . . 166

fnpActSvcInstallWin . . . 167
fnpActSvcUninstallWin . . . 168
fnpActSvcGetLastErrorWin . . . 170
fnpActSvcInstallForMSI . . . 172
fnpActSvcUninstallForMSI . . . 173

8 Activation API . . . 177

API Categories in This Chapter . . . 177

API Structure—Objects . . . 178

Common Library Functions . . . 179

flxActCommonDedSpcCountGet . . . 183
flxActCommonDedSpcDestinationFulfillmentIdGet . . . 184
flxActCommonDedSpcDestinationSystemNameGet . . . 185
flxActCommonDedSpcExpDateGet . . . 186
flxActCommonDedSpcTypeGet . . . 187
flxActCommonGetFRCountFromServerTS . . . 188
flxActCommonGetProtectionMode . . . 190
flxActCommonHandleClose . . . 191
flxActCommonHandleDeleteProduct . . . 191
flxActCommonHandleGetBuildInfo . . . 193
flxActCommonHandleGetCommType . . . 194
flxActCommonHandleGetError . . . 194
flxActCommonHandleGetLastOpsError . . . 195
flxActCommonHandleGetLastOpsErrorString . . . 195
flxActCommonHandleGetLastResponseError . . . 196
flxActCommonHandleGetLastResponseErrorReason . . . 197
flxActCommonHandleGetLastResponseErrorString . . . 198
flxActCommonHandleGetRemoteServer . . . 199
flxActCommonHandleGetServiceBuildInfo . . . 199
flxActCommonHandleGetServiceVersion . . . 200
flxActCommonHandleGetUniqueMachineNumber . . . 201
flxActCommonHandleGetVersion . . . 202
flxActCommonHandleOpen . . . 203
flxActCommonHandleSetCommType . . . 204
flxActCommonHandleSetPollInterval . . . 206
flxActCommonHandleSetProxyDetails . . . 207
flxActCommonHandleSetRemoteServer . . . 208
flxActCommonHandleSetSSLDetails . . . 210
flxActCommonHandleSetStatusCallback . . . 211
flxActCommonHandleSetTimeout . . . 214
flxActCommonLibraryCleanup . . . 215
flxActCommonLibraryInit . . . 216
flxActCommonLicSpcAddASRFromBuffer . . . 220
flxActCommonLicSpcAddASRs . . . 221
flxActCommonLicSpcCheckASR . . . 223
flxActCommonLicSpcCheckASRFromBuffer . . . 224

flxActCommonLicSpcCreate . . . 225
flxActCommonLicSpcDelete . . . 227
flxActCommonLicSpcGet . . . 228
flxActCommonLicSpcGetNumberProducts . . . 228
flxActCommonLicSpcGetNumProdEntries . . . 229
flxActCommonLicSpcGetProd . . . 230
flxActCommonLicSpcPopulateAllFromServerTS . . . 231
flxActCommonLicSpcPopulateAllFromTS . . . 233
flxActCommonLicSpcPopulateFromServerTS . . . 234
flxActCommonLicSpcPopulateFromTS . . . 237
flxActCommonLicSpcProductDelete . . . 238
flxActCommonProdLicSpcActServerChainGet . . . 239
flxActCommonProdLicSpcCountGet . . . 240
flxActCommonProdLicSpcDedSpcGet . . . 241
flxActCommonProdLicSpcEntitlementIdGet . . . 241
flxActCommonProdLicSpcExpDateGet . . . 242
flxActCommonProdLicSpcFeatureLineGet . . . 243
flxActCommonProdLicSpcFulfillmentIdGet . . . 243
flxActCommonProdLicSpcFulfillmentTypeGet . . . 244
flxActCommonProdLicSpcIsDisabled . . . 246
flxActCommonProdLicSpcNumberDedSpcGet . . . 247
flxActCommonProdLicSpcNumberValidDedSpcGet . . . 248
flxActCommonProdLicSpcPopulateFRFromServerTS . . . 249
flxActCommonProdLicSpcProductIdGet . . . 251
flxActCommonProdLicSpcSuiteIdGet . . . 252
flxActCommonProdLicSpcTrustFlagsGet . . . 252
flxActCommonProdLicSpcUniqueIdGet . . . 253
flxActCommonProdLicSpcVendorDataGetByIndex . . . 254
flxActCommonProdLicSpcVendorDataGetByKey . . . 255
flxActCommonProdLicSpcVendorDataGetCount . . . 256
flxActCommonRepairLocalTrustedStorage . . . 258
flxActCommonResetTrialASRFromBuffer . . . 259
flxActCommonResetTrialASRs . . . 260
flxActCommonRespProdLicSpcGet . . . 262
flxActDiagTrustedStorageHealthCheck . . . 263
flxActCommonValidateDateString . . . 264
flxActCommonVirtualFamilyGet . . . 265
flxActCommonVirtualGenidGet . . . 266
flxActCommonVirtualNameGet . . . 267
flxActCommonVirtualStatusGet . . . 268
flxActCommonVirtualUuidGet . . . 269
Error and Warning Reporting . . . 270
Composite Transaction API Functions . . . 285
Additional Information . . . 285
Function Summary . . . 285
flxActTranDictionaryGetByIndex . . . 287

flxActTranDictionaryGetByKey . . . 288
flxActTranDictionaryGetCount . . . 289
flxActTranDictionarySetByKey . . . 290
flxActTranReqActionGetAttribute . . . 291
flxActTranReqActionGetType . . . 291
flxActTranReqActionGetVendorDictionary . . . 292
flxActTranReqActionSetAttribute . . . 293
flxActTranRequestAddAction . . . 294
flxActTranRequestAddExistingFulfillment . . . 294
flxActTranRequestClearExistingFulfillments . . . 295
flxActTranRequestGetAction . . . 296
flxActTranRequestGetActionCount . . . 297
flxActTranRequestGetAttribute . . . 297
flxActTranRequestGetStatus . . . 298
flxActTranRequestGetVendorDictionary . . . 299
flxActTranRequestGetXML . . . 300
flxActTranRequestMatchStored . . . 301
flxActTranRequestSave . . . 301
flxActTranRequestSetAttribute . . . 302
flxActTranRequestSetExistingFulfillmentDetails . . . 303
flxActTranResponseGetAction . . . 304
flxActTranResponseGetActionCount . . . 305
flxActTranResponseGetVendorDictionary . . . 305
flxActTranRspActionGetAttribute . . . 306
flxActTranRspActionGetVendorDictionary . . . 307
flxActTranRspActionGetResult . . . 308
flxActTranRspActionGetType . . . 309
flxActTransactionCreate . . . 310
flxActTransactionDestroy . . . 311
flxActTransactionCreateRequest . . . 311
flxActTransactionDeleteRequestBySeqNo . . . 312
flxActTransactionDeleteStoredRequest . . . 313
flxActTransactionGetCommsAttribute . . . 313
flxActTransactionGetRequest . . . 314
flxActTransactionGetResponse . . . 315
flxActTransactionGetResponseXML . . . 316
flxActTransactionLoadStoredRequest . . . 317
flxActTransactionProcessResponse . . . 317
flxActTransactionSend . . . 319
flxActTransactionSetCommsAttribute . . . 320
flxActTransactionGetStoredRequestCount . . . 322
Error and Warning Reporting . . . 323
Short Code Transaction APIs . . . 328
Function Summary . . . 328
flxActShortCodeCreate . . . 328
flxActShortCodeCreateFromBuffer . . . 329

flxActShortCodeCancelRequest . . . 330
flxActShortCodeDestroy . . . 331
flxActShortCodeProcessResponse . . . 331
flxActShortCodeGetActivationRequest . . . 332
flxActShortCodeGetDenyReason . . . 334
flxActShortCodeGetErrorDetail . . . 334
flxActShortCodeGetPendingRequest . . . 335
flxActShortCodeGetPendingRequestProdSpc . . . 336
flxActShortCodeGetRepairRequest . . . 337
flxActShortCodeGetReturnRequest . . . 338
flxActShortCodeGetWarning . . . 339
flxActShortCodeSetVendorData . . . 340
Deprecated Single-Action Application Activation Functions . . . 341
Additional Information . . . 342
Function Summary . . . 342
flxActAppActivationCreate . . . 345
flxActAppActivationDelete . . . 346
flxActAppActivationSend . . . 347
flxActAppActivationReqSet . . . 348
flxActAppActivationRespProcess . . . 350
flxActAppActivationRespProdLicSpcGet . . . 351
flxActAppActivationEntitlementIdGet . . . 352
flxActAppActivationEntitlementIdSet . . . 353
flxActAppActivationDurationGet . . . 353
flxActAppActivationDurationSet . . . 354
flxActAppActivationExpDateGet . . . 355
flxActAppActivationExpDateSet . . . 355
flxActAppActivationReasonGet . . . 356
flxActAppActivationReasonSet . . . 357
flxActAppActivationProductIdGet . . . 358
flxActAppActivationProductIdSet . . . 358
flxActAppActivationVendorDataGet . . . 359
flxActAppActivationVendorDataSet . . . 359
flxActAppReturnCreate . . . 361
flxActAppReturnDelete . . . 362
flxActAppReturnSend . . . 363
flxActAppReturnReqSet . . . 364
flxActAppReturnRespProcess . . . 366
flxActAppReturnCancel . . . 367
flxActAppReturnProdLicSpcSet . . . 369
flxActAppReturnProductIdGet . . . 369
flxActAppReturnSuiteIdGet . . . 370
flxActAppReturnEntitlementIdGet . . . 371
flxActAppReturnFRIdSet . . . 371
flxActAppReturnFRIdGet . . . 372
flxActAppReturnReasonGet . . . 373

flxActAppReturnReasonSet . . . 373
flxActAppReturnVendorDataGet . . . 374
flxActAppReturnVendorDataSet . . . 375
flxActAppRepairCreate . . . 376
flxActAppRepairDelete . . . 378
flxActAppRepairSend . . . 378
flxActAppRepairReqSet . . . 380
flxActAppRepairRespProcess . . . 381
flxActAppRepairProdLicSpcSet . . . 383
flxActAppRepairProductIdGet . . . 383
flxActAppRepairSuiteIdGet . . . 384
flxActAppRepairEntitlementIdGet . . . 385
flxActAppRepairFRIdSet . . . 385
flxActAppRepairFRIdGet . . . 386
flxActAppRepairVendorDataGet . . . 387
flxActAppRepairVendorDataSet . . . 387
Deprecated License Server Activation Functions . . . 389
Additional Information . . . 389
Function Summary . . . 390
flxActSvrActivationCreate . . . 393
flxActSvrActivationDelete . . . 394
flxActSvrActivationSend . . . 394
flxActSvrActivationReqSet . . . 396
flxActSvrActivationRespProcess . . . 398
flxActSvrActivationCountGet . . . 399
flxActSvrActivationCountSet . . . 400
flxActSvrActivationEntitlementIdGet . . . 401
flxActSvrActivationEntitlementIdSet . . . 402
flxActSvrActivationExpDateGet . . . 403
flxActSvrActivationExpDateSet . . . 403
flxActSvrActivationReasonGet . . . 404
flxActSvrActivationReasonSet . . . 405
flxActSvrActivationUsernameGet . . . 406
flxActSvrActivationUsernameSet . . . 406
flxActSvrActivationHostnameGet . . . 407
flxActSvrActivationHostnameSet . . . 407
flxActSvrActivationIncludeEnterpriseData . . . 408
flxActSvrActivationProductIdGet . . . 409
flxActSvrActivationProductIdSet . . . 410
flxActSvrActivationVendorDataGet . . . 410
flxActSvrActivationVendorDataSet . . . 411
flxActSvrReturnCreate . . . 412
flxActSvrReturnDelete . . . 414
flxActSvrReturnSend . . . 414
flxActSvrReturnReqSet . . . 417
flxActSvrReturnRespProcess . . . 418

flxActSvrReturnCancel . . . 420
flxActSvrReturnProdLicSpcSet . . . 421
flxActSvrReturnReasonGet . . . 421
flxActSvrReturnReasonSet . . . 422
flxActSvrReturnNumberValidDedSpcGet . . . 423
flxActSvrReturnForceIncompleteGet . . . 424
flxActSvrReturnForceIncompleteSet . . . 425
flxActSvrReturnProductIdGet . . . 426
flxActSvrReturnSuiteIdGet . . . 426
flxActSvrReturnEntitlementIdGet . . . 427
flxActSvrReturnFRIdGet . . . 428
flxActSvrReturnVendorDataGet . . . 428
flxActSvrReturnVendorDataSet . . . 429
flxActSvrRepairCreate . . . 430
flxActSvrRepairDelete . . . 431
flxActSvrRepairSend . . . 432
flxActSvrRepairReqSet . . . 433
flxActSvrRepairRespProcess . . . 434
flxActSvrRepairProdLicSpcSet . . . 436
flxActSvrRepairProductIdGet . . . 436
flxActSvrRepairSuiteIdGet . . . 437
flxActSvrRepairEntitlementIdGet . . . 438
flxActSvrRepairFRIdGet . . . 438
flxActSvrRepairVendorDataGet . . . 439
flxActSvrRepairVendorDataSet . . . 439

Deprecated Activation Functions . . . 441
flxActAppActivationReqCreate . . . 441
flxActAppReturnReqCreate . . . 442
flxActAppRepairReqCreate . . . 443
flxActSvrActivationReqCreate . . . 445
flxActSvrReturnReqCreate . . . 447
flxActSvrRepairReqCreate . . . 448

Deprecated Short Code Functions . . . 449

Common Data . . . 452
Unique Machine Number (UMN) . . . 452
Platform Type . . . 453

9 Obsolete Flexible API Features . . . 455

Obsolete Flexible API Functions . . . 455

Obsolete Flexible API Attributes . . . 458

10 Flexible API Error Return Values . . . 461

Glossary . . . 485

Index 497

1

Introduction

This manual *C/C++ Function Reference* contains details of all the API functions used for Licensing. It includes functions used with license files and with trusted storage. It contains the following chapters and appendices:

Table 1-1 • *FlexNet Publisher C/C++ Function Reference* Chapter Overview

Section	Content
Product Introduction	Provides an overview of the *FlexNet Publisher* C/C++ Function Reference, including Help Library conventions and usage.
Flexible API	Descriptions of the most commonly used Flexible API functions.
Advanced Flexible API Functions	Descriptions of less commonly used Flexible API functions.
Flexible API Attributes	Descriptions of commonly used attributes used with the Flexible API function **lc_set_attr** to control licensing behavior.
Advanced Flexible API Attributes	Descriptions of less commonly used attributes.
Activation Borrow API	Descriptions of the Activation Borrow API functions.
FlexNet Licensing Service API	API used on Windows platforms to install and uninstall the FlexNet Licensing Service.
Activation API	Descriptions of the Activation API functions.
Obsolete Flexible API Features	Descriptions of Flexible API functions and attributes that are obsolete.
Flexible API Error Return Values	Error values returned by **lc_xxx** functions.
Glossary	Glossary of FlexNet Publisher terms.

Toolkit Functionality

The licensing and security functionality available in the toolkit depends on the platform on which the toolkit and FlexEnabled applications are run. There are three levels of functionality:

- **Level One: Licensing using license files only**—See information in the following sections except where it references trusted storage: Chapter 2, Flexible API, Chapter 3, Advanced Flexible API Functions, Chapter 10, Flexible API Error Return Values, Chapter 4, Flexible API Attributes and Chapter 5, Advanced Flexible API Attributes.
- **Level Two: Licensing using license files and trusted storage but no additional security**—Most of the functionality described in this document is available. This level of functionality is referred to as *notr* within this document.
- **Level Three: Licensing using license files and trusted storage and additional security**—All of the functionality described in this document is available. This level of functionality is referred to as *standard* within this document to distinguish it from notr.

For details of the functionality available in your toolkit, see the *Detailed Platform Information* section in the Release Notes.

FlexNet Publisher Product Documentation

Flexera provides a comprehensive documentation set to assist you in working with the FlexNet Publisher Licensing Toolkit. See the *Reference Bulletin for Documentation* for the current release for information about the most up-to-date documentation.

2

Flexible API

Flexible API functions are used in the following contexts:

- **License rights usage**—By a FlexEnabled application to use license rights held either in license files or trusted storage. To report the status and usage of license rights.
- **Trusted storage access**—To load the activation library that gives access to trusted storage.
- **License server**—Heartbeat management by the FlexEnabled application.
- **License file**—To create utilities to generate and install license files.

The following manuals provide Flexible API usage examples:

- *Programming Reference for License File–Based Licensing*—Adding the client API to an application
- *Programming Reference for Trusted Storage–Based Licensing*—Adding function calls to a FlexEnabled application

Required Flexible API Functions

The following table lists a minimal set of required Flexible API functions for a FlexEnabled application. Note that the functions required for access to license rights in local trusted storage also allow a FlexEnabled application to use license rights held in a local license file or from a license server.

Table 2-1 • Minimal Set of Flexible API functions

License Rights in Local License File or from a License Server	License Rights in Local Trusted Storage
	1. lc_flexinit_property_handle_create
	2. lc_flexinit_property_handle_set
	3. lc_flexinit
1. lc_new_job	4. lc_new_job
2. lc_set_attr	5. lc_set_attr

Table 2-1 • Minimal Set of Flexible API functions

License Rights in Local License File or from a License Server	License Rights in Local Trusted Storage
3. lc_checkout	6. lc_checkout
4. lc_perror or lc_errstring	7. lc_perror or lc_errstring
5. lc_checkin	8. lc_checkin
6. lc_free_job	9. lc_free_job
	10. lc_flexinit_cleanup
	11. lc_flexinit_property_handle_free
7. lc_virtualstatusget	12. lc_virtualstatusget

Flexible API Functions by Category

Incorporating the Flexible API into your application involves adding calls to functions from several different function categories, which are described in the following sections. Each section includes the category's API functions and related attributes that are used with the **lc_set_attr** and **lc_get_attr** functions. Full descriptions for the Flexible API functions are presented later in the chapter, and the attributes are described in Flexible API Attributes.

Job Handling

The Flexible API uses a job handle to hold information required for related API functions. Thus a job handle is required to initialize FlexNet Licensing. Functions that are related to the job handle apply to all items controlled by that job handle. For example:

- Attributes set for the job handle (using the function **lc_set_attr**) apply to all subsequent checkouts using that job handle.
- When a job handle is freed, all licenses checked out using that job handle are no longer available to the application. If the license was obtained from a license server, then those licenses become available for checkout.

When a license server is used to obtain license rights, then the job handle represents a connection to the server.

Job-handling API functions manage job handles.

Table 2-2 • Job Handling Functions

Function Name	Description
lc_first_job	Locates the first job on the list of jobs.
lc_free_job	Frees the memory associated with a job.
lc_new_job	Initializes FlexNet Licensing and creates a new job.
lc_next_job	Walks the list of jobs.

FlexNet Publisher and Multithreaded Applications

None of the FlexNet Publisher APIs (flexible or activation) were designed to be thread safe. FlexNet Publisher APIs are not tested in multithreaded scenarios. Publishers wishing to access FlexNet Publisher APIs from multiple threads should synchronize all calls through a mutex.

License Checkin and Checkout

The FlexEnabled application interfaces to FlexNet Licensing via a set of routines that request (check out) and release (check in) licenses for selected features. The following tables list the API functions and attributes related to checking in and checking out a feature.

Table 2-3 • Checkin and Checkout Functions

Function Name	Description
lc_auth_data	Gets the `FEATURE` or `INCREMENT` line for a checked-out feature.
lc_checkin	Checks in (releases) the licenses of the specified feature.
lc_checkout	Requests a license of a feature.
lc_status	Returns the status of the requested feature.

Table 2-4 • Checkin and Checkout Attributes

Attribute Name	Description
LM_A_BORROW_EXPIRE	Sets the date and time when a license borrowed using the `BORROW` keyword expires.
LM_A_CHECK_BADDATE	Checks system date on the machine.
LM_A_TS_CHECK_BADDATE	Checks system date against timestamp in trusted storage.
LM_A_LICENSE_DEFAULT	The expected license file location.
LM_A_LINGER	Controls the license linger time.
LM_A_PROMPT_FOR_FILE (Windows Only)	Prompts the user for a license-file path or server name.

Error and Warning Reporting

These functions provide a way to report errors and warnings from the FlexEnabled application to the user.

Table 2-5 • Error and Warning Reporting Functions

Function Name	Description
lc_err_info	Used for internationalization.
lc_errstring	Returns the Flexible API error string for the most recent Flexible API error.
lc_get_errno	Returns the most recent Flexible API error.
lc_perror	Prints a Flexible API error message to `stderr` on UNIX and to a message box on Windows.

Table 2-6 • Error and Warning Reporting Attributes

Attribute Name	Description
LM_A_LONG_ERRMSG	Presents error messages in the long format.
LM_A_PERROR_MSGBOX (Windows Only)	Presents the error message in an error dialog box.
LM_A_WINDOWS_MODULE_HANDLE (Windows only)	Enables dialogs and error messages to be displayed properly (used with DLL only)

Initialization Routines for Trusted Storage and SDTs

These initialization functions are required for activation support, the tamper-resistant run time, Secure Data Type (SDT) implementation.

Table 2-7 • Initialization Routines for Trusted Storage

Function Name	Description
lc_flexinit	Loads the activation library and initializes FlexNet Licensing.
lc_flexinit_cleanup	Unloads the activation library and performs any cleanup functions required.
lc_flexinit_property_handle_create	Creates a property handle.
lc_flexinit_property_handle_free	Frees an initialization property handle.
lc_flexinit_property_handle_set	Sets a type and value into the initialization property handle.
lc_flexinit_property_handle_get	Returns the value for a specified property.

The use of these routines must be coordinated with:

- Use of Secure Data Types (SDTs)—You must call **lc_flexinit** before any SDTs are used in the code. See the *Programming Reference for Trusted Storage–Based Licensing* for more information about SDTs.
- FlexNet Licensing job management—You must call **lc_flexinit** before the call to **lc_new_job**, and call **lc_flexinit_cleanup** after the call to **lc_free_job**.

In multi-threaded environments, you must ensure synchronization between threads. The following are areas where particular care is needed:

- Executable exit logic—Your code must explicitly check in all licenses, using **lc_checkin**, and free all jobs, using **lm_free_job**, before calling **lc_flexinit_cleanup** and exiting. Do not rely on the FlexNet Licensing automatic checkin mechanism to perform proper cleanup upon exit.
- Lost connection to the license server—When a concurrent license is in use and the connection to the license server breaks, the code must explicitly check in all licenses before calling **lc_flexinit_cleanup**.

See Also
Programming Reference for Trusted Storage–Based Licensing

Heartbeat Management and Communication

Heartbeat exchange is the mechanism by which a FlexEnabled application keeps track of the license server status. Your FlexEnabled application can use either automatic or manual heartbeats. If you are using manual heartbeats, the API functions in Table 2-8 and the attributes in Table 2-9 are required

Table 2-8 • Heartbeat Management and Communication Functions

Function Name	Description
lc_heartbeat	Exchanges heartbeat messages with the license server. Used for manual heartbeat implementations.
lc_idle	Informs the license server when the FlexEnabled application is idle.

Table 2-9 • Heartbeat Management and Communication Attributes

Attribute Name	Description
LM_A_CHECK_INTERVAL	Controls the FlexEnabled application's detection of license server failures.
LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL	Specifies the number of and the interval between automatic reconnection attempts.
LM_A_TCP_TIMEOUT	Specifies the number of seconds to wait, after TCP/IP disconnection, before license is automatically checked in.
LM_A_USER_EXITCALL, LM_A_USER_EXITCALL_EX	Pointer to a function that receives control if reconnection fails.

Table 2-9 • Heartbeat Management and Communication Attributes

Attribute Name	Description
LM_A_USER_RECONNECT, LM_A_USER_RECONNECT_EX	Pointer to a function that is called just before a reconnection attempt.
LM_A_USER_RECONNECT_DONE, LM_A_USER_RECONNECT_DONE_EX	Pointer to a function that is called after a successful reconnection.
LM_A_USER_RECONNECT_V1_DATA	Use to specify a callback routine to pass reconnection data.
LM_A_USER_RECONNECT_V1	Use to specify a callback routine, called before every reconnection.
LM_A_USER_RECONNECT_V1_DONE	Use to specify a callback routine used after successfully reconnecting to a license server.

Informational

The following tables list the informational functions and attributes. They provide status and usage information not critical to the license management process.

Table 2-10 • Informational Functions

Function Name	Description
lc_expire_days	Returns the number of days until a license expires.
lc_feat_list	Gets the list of all features in the license file
lc_fnpservice_present	Provides the status of whether the FlexNet Licensing Service is present.
lc_get_version	Gets the version of the Licensing toolkit client library.
lc_get_server_version	Gets the version of Licensing Server.
lc_log	Logs a message in the debug log file.
lc_tpmstatusget (Windows only)	Gets the Trusted Platform Module (TPM) status of the system.
lc_userlist	Provides a list of who is using a feature. Helpful to explain why a license is denied.
lc_vsend	Sends a message to the vendor daemon and returns a result string.

Table 2-11 • Informational Attributes

Attribute Name	Description
LM_A_BORROW_STAT	Provides an programming interface to the lmborrow functionality.

Table 2-11 • Informational Attributes

Attribute Name	Description
LM_A_LF_LIST	Lists all license files searched for features.
LM_A_VD_GENERIC_INFO, LM_A_VD_FEATURE_INFO	Retrieves information from the vendor daemon: ● `LM_A_VD_GENERIC_INFO` gets information about the server. ● `LM_A_FEATURE_INFO` gets information about a specific feature.

License File Installation

Software publishers may want to supply a utility along with their FlexEnabled application that performs license file installation.

Table 2-12 • License File Installation Functions

Function Name	Description
lc_hostid	Provides the hostid in string format.

License File Generation

These functions and attributes are provided to the vendor who wishes to develop a proprietary license file generation utility.

Table 2-13 • License File Generation Functions

Function Name	Description
lc_cryptstr	Generates a valid signatures for a license file.
lc_init	Creates a job handle for use by **lc_cryptstr**.

Table 2-14 • License File Generation Attributes

Attribute Name	Description
LM_A_LICENSE_FMT_VER	Specifies the version format of the license that **lc_cryptstr** generates.

Flexible API Function Descriptions

The following table lists the Flexible API functions described in this chapter with a link to the full description. For a grouping by function category, see Flexible API Functions by Category.

Table 2-15 • Flexible API Function Summary

Function Name	Description
lc_auth_data	Gets the `FEATURE` or `INCREMENT` line for a checked-out feature.
lc_checkin	Checks in (releases) the licenses of the specified feature.
lc_checkout	Requests a license of a feature.
lc_cryptstr	Generates a valid signature in a feature line.
lc_err_info	Returns error information to an `LM_ERR_INFO` structure pointer.
lc_errstring	Returns the Flexible API error string for the most recent error.
lc_expire_days	Returns the number of days until a license expires.
lc_feat_list	Gets the list of all features in the license file
lc_first_job	Locates the first job on the list of jobs.
lc_flexinit	Loads and initializes the activation library component.
lc_flexinit_cleanup	Unloads the activation library.
lc_flexinit_property_handle_create	Creates a property handle for use by **lc_flexinit**.
lc_flexinit_property_handle_free	Frees the property handle.
lc_flexinit_property_handle_get	Gets the current value of the specified property handle.
lc_flexinit_property_handle_set	Sets the value of the specified property handle.
lc_free_job	Frees the memory associated with a job.
lc_get_attr	Retrieves a Flexible API attribute.
lc_get_errno	Returns the most recent Flexible API error.
lc_get_version	Gets the version of the Licensing toolkit client library.
lc_get_server_version	Gets the version of the Licensing Server.
lc_heartbeat	Exchanges heartbeat messages with the license server.

Table 2-15 • Flexible API Function Summary

Function Name	Description
lc_hostid	Provides the hostid in string format.
lc_idle	Informs the license server when the FlexEnabled application is idle.
lc_init	Creates a job handle for use by **lc_cryptstr**.
lc_log	Logs a message in the debug log file.
lc_new_job	Initializes FlexNet Licensing and creates a new job.
lc_next_job	Walks the list of jobs.
lc_perror	Prints a Flexible API error message.
lc_set_attr	Sets a Flexible API attribute.
lc_status	Returns the status of the requested feature.
lc_tpmstatusget (Windows only)	Gets the Trusted Platform Module (TPM) status of the system.
lc_userlist	Provides a list of who is using a feature.
lc_vsend	Sends a message to the vendor daemon and returns a result string.

See Also
Advanced Flexible API Functions

lc_auth_data

Gets the `FEATURE` or `INCREMENT` line for a checked-out feature.

Syntax

```
CONFIG *lc_auth_data(
    LM_HANDLE *job,
    char *feature);
```

Parameters

The **lc_auth_data** function uses the following parameters:

Table 2-16 • lc_auth_data Parameters

Parameter	Description
job	From **lc_new_job**.

Table 2-16 • lc_auth_data Parameters

Parameter	Description
feature	The desired feature.

Return Values

The following values are returned by **lc_auth_data:**

Table 2-17 • lc_auth_data Return Values

Return Value	Description
CONFIG	The `CONFIG` struct, or NULL if error. The `CONFIG` struct is defined in the header file `lmclient.h`.

Error Returns

When NULL is returned instead of a CONFIG, the following errors are set in the job handle used when calling **lc_auth_data:**

Table 2-18 • lc_auth_data Error Returns

Error Return	Description
LM_FUNCNOTAVAIL	Vendor keys do not support this function.
LM_NOFEATURE	Feature not found.

Note • *If you want a checkout to depend on information in the license line, call* **lc_set_attr** *with `LM_A_CHECKOUTFILTER`, `LM_A_CHECKOUTFILTER_EX`, or `LM_A_CHECKOUTFILTERLAST_EX` before the call to* **lc_checkout**.

Discussion

Gets authenticated information corresponding to the specific license line for a feature that has been checked out. Use a separate call to **lc_auth_data** for each checked out feature.

The FlexNet license server may combine `FEATURE` and `INCREMENT` lines for the same feature into a single license pool. When this *pooling* takes place, the license server reports only on the first `FEATURE` or `INCREMENT` that contributed to the pool; the details of all others are unavailable. Thus in this case, lc_auth_data returns only the first `FEATURE/INCREMENT` line for the feature in the license pool. When license rights are held in license files and trusted storage, license rights are pooled from both sources. For further information about license pools, see *Programming Reference for License File–Based Licensing*. For information about license rights held in trusted storage, see "Description of License Rights in Server-side Trusted Storage" in *Programming Reference for Trusted Storage–Based Licensing*.

Note • *Compare this function with the attribute `LM_A_PORT_HOST_PLUS`, which retrieves authenticated information for all features in a license file without their having to be checked out first.*

Example

The following code excerpt demonstrates using this function. In this example, the application obtains the feature line for the feature just checked out in order to process the `NOTICE` field. For this example, consider the following feature line:

```
FEATURE f1 demo 1.0 permanent 4 NOTICE="ab" SIGN=xxxxxxxxxxxx
```

The code appears as follows in the application:

```
Fnp_config_type config_type;
...
/* after the call to lc_new_job() */
if (lc_checkout(job, feature, ...))
{
    /* error processing...*/
}
else
{
    conf = lc_auth_data(job, feature);

    if (conf)
    {
        config_type = conf->nConfigType;
        /* Use conf->lc_notice for intellectual property info. */
        if(conf->lc_notice[0] == 'a')
        {
            /* "a" processing */
        }
        else if(conf->lc_notice[1] == 'b')
        {
            /* "b" processing */
        }

        ...
        if (config_type == FNP_CONFIG_TYPE_CLIENT_TS)
        {
            /* CONFIG from client trusted storage */
            ...
        }
        else if (config_type == FNP_CONFIG_TYPE_UNSERVED_CERT)
        {
            /* CONFIG from unserved license file */
            ...
        }
        else if (config_type == FNP_CONFIG_TYPE_SERVED_TS)
        {
            /* CONFIG from server trusted storage */
            ...
        }
        else if (config_type == FNP_CONFIG_TYPE_SERVED_CERT)
        {
            /* CONFIG from served license file */
            ...
        }
        /* Display information for the given feature  */
        printf("feature=%s version=%s daemon=%s, feature count = %d, total feature count=%u   \n",
            conf->feature, conf->version, conf->daemon, conf->users, conf->nTotalFeatureCount);
```

```
}
```

See Also
`lmclient.h` for the `CONFIG` struct definition
lc_get_attr
lc_checkout
LM_A_MULTIPLE_CHECKOUT_DATA
LM_A_CHECKOUTFILTERLAST_EX
LM_A_PORT_HOST_PLUS

lc_checkin

Checks in (releases) the licenses of the specified feature. When a license server supplied the licenses, the licenses are returned to the license pool.

Syntax

```
void lc_checkin(
    LM_HANDLE *job,
    LM_CONST_CHAR_PTR feature,
    int keep);
```

Parameters

The **lc_checkin** function uses the following parameters:

Table 2-19 • lc_checkin Parameters

Parameter	Description
job	From **lc_new_job**.
feature	The feature name to be checked in, or `LM_CI_ALL_FEATURES`.
keep	If the license was obtained from a license server, specifies subsequent action: ● Zero—Drop connection to license server. ● Non-zero—Keep connection to license server. The time period that the connection is kept open is specified using LM_A_TCP_TIMEOUT.

Return Values

There are no values returned by **lc_checkin.**

Discussion

Checks in the licenses of the specified feature. It is recommended that your code always calls **lc_checkin** and **lc_free_job** before the application exits. When the license was obtained from a license server, the connection to the license server can be held open so that subsequent requests for licenses are processed more quickly.

Important • *If your application code uses Secure Data Types (SDTs), ensure that* **lc_checkin** *is used correctly and consistently. See the Programming Reference for Trusted Storage–Based Licensing for more details about SDTs.*

When a license server supplied the licenses, the vendor daemon will detect the fact that the FlexEnabled application exited, and return any licenses that were checked out back to the available pool. The return of a license when the application exits rather than by calling **lc_checkin** is recorded differently in the report log file. For reporting purposes, it is preferable to check in a license with **lc_checkin** rather than simply exiting.

lc_checkout

Requests a license for a feature.

Syntax

```
int lc_checkout(
    LM_HANDLE_PTR job,
    LM_CONST_CHAR_PTR feature,
    LM_CONST_CHAR_PTR version,
    int nlic,
    int flag,
    const VENDORCODE *key,
    int dup_group);
```

Parameters

The **lc_checkout** function uses the following parameters:

Table 2-20 • lc_checkout Parameters

Parameter	Description
job	From **lc_new_job**.
feature	A Null terminated ASCII feature name desired.
version	The version of the feature desired in floating-point format, with a maximum of ten characters (for example, “12345.123” or “123.456789”). This value must be less than or equal to the version number in the license file for the checkout to succeed. The version value cannot contain letters.
nlic	A positive integer specifying the number of licenses to check out.
flag	The Checkout Option Flag sets the response when a checkout request is denied.
key	From **lc_new_job**.
dup_group	Duplicate Grouping Mask for this feature.

Return

The following values are returned:

Table 2-21 • Registry and $HOME/.flexlmrc Returns

Return Value	Description
0	OK, license checked out
<> 0	Error, see Error Returns for details.

Error Returns

These are the error returns:

Table 2-22 • Registry and $HOME/.flexlmrc Error Returns

Error Return	Description
LM_BADCODE	Signature in license file does not match other data in file.
LM_BADHANDSHAKE	Authentication handshake with vendor daemon failed.
LM_BADPARAM	Flexible API function argument is invalid or there is an invalid setting in `lm_code.h`.
LM_BADSYSDATE	System clock has been set back. This error can occur only when the `FEATURE` line contains an expiration date.
LM_BAD_VERSION	Version argument is invalid floating-point format.
LM_BORROW_TOOLONG	Cannot borrow a license for that many hours. The user has specified a borrow period that is greater than the license allows.
LM_BUSYNEWSERV	License server busy starting another copy of itself; retry.
LM_CANTCONNECT	Cannot establish a connection with a license server.
LM_FEATQUEUE	Feature is queued. **lc_status** will indicate when it is available.
LM_FUNCNOTAVAIL	Vendor keys do not support this function.
LM_LOCALFILTER	Checkout request filtered by the publisher-defined filter routine.
LM_MAXLIMIT	Checkout exceeds `MAX` specified in the administrative options file, which limits license usage for a specified user or group.
LM_MAXUSERS	All licenses in use. Applications usually need to test for both `LM_MAXUSERS` and `LM_USERSQUEUED` instead of only `LM_MAXUSERS`.

Table 2-22 • Registry and $HOME/.flexlmrc Error Returns

Error Return	Description
LM_ONE_TSOK_ERR	Only one terminal server remote client checkout is allowed for this feature. The license includes the keyword ONE_TS_OK.
LM_NO_SERVER_IN_FILE	No license server specified for counted license.
LM_NOBORROW_SUPP	License `BORROW` support not enabled, but borrowing was requested by the user.
LM_NOFEATURE	Cannot find requested feature in the license file.
LM_NOSERVSUPP	License server has different license file than the FlexEnabled application; FlexEnabled application's license has feature, but license server's does not.
LM_OLDVER	License file does not support a version this new.
LM_PLATNOTLIC	This platform is not authorized by the license; running on a platform not included in `PLATFORMS="..."` list.
LM_SERVBUSY	License server busy—the request should be retried. (This is a rare occurrence.)
LM_SERVOLDVER	LM_SERVOLDVER error has the following limitations ● It is returned only from lc_checkout ● It only compares major and minor versions, not sub minor versions and patches ● It is supported from server version >= 11.12.1.
LM_SERVLONGGONE	The version number of the license to be checked out is greater than the version number on the INCREMENT line.
LM_SOCKETFAIL	The call to **socket** made by **lc_checkout** has failed. This error can occur when the UNIX OS runs out of system resources.
LM_TSOK_ERR	The feature line does not contain TS_OK to allow Terminal Server Client usage.
LM_USERSQUEUED	Similar to `LM_MAXUSERS`, but also indicates that there are already some users queued. Applications usually need to test for both `LM_MAXUSERS` and `LM_USERSQUEUED` instead of only `LM_MAXUSERS`.
LM_VD_MAX_CLIENTS_REACHED	The maximum number of connections reached for this vendor daemon.
LM_COAVAIL_RECONN_NOT_ALL_CNT_AVAIL	Failed to get all requested license(s) in reconnection.
LM_COAVAIL_FLAG_NOT_SUPPORTED	The supplied checkout option flag is not supported.

Table 2-22 • Registry and $HOME/.flexlmrc Error Returns

Error Return	Description
LM_COAVAIL_PACKAGE_NOT _SUPPORTED	The checkout-avail flag is not supported with PACKAGE licenses.
LM_COAVAIL_UNSUPPORTED	The checkout-avail (COAVAIL) checkout is not supported.

Discussion

Before calling **lc_checkout,** it is recommended that the application first indicate the expected license file location by using **lc_set_attr** to specify a license search path using the attribute LM_A_LICENSE_DEFAULT:

```
lc_set_attr(job, LM_A_LICENSE_DEFAULT, (LM_A_VAL_TYPE)license_file_list)
```

The *Programming Reference for License File–Based Licensing* provides further information on this and the order in which license files are chosen. For applications that use licenses in trusted storage, a license file must be accessible by the application, however it does not need to contain any valid licenses.

lc_checkout attempts to check out one (or more) licenses of the specified feature. Place the call to **lc_checkout** in an executable so that the license is available whenever the user is using the licensed feature. When Secure Data Types (SDTs) are in use (see *Programming Reference for Trusted Storage–Based Licensing* for details), **lc_checkout** must be called before any SDTs are used. It is recommended that when the license is no longer required it is checked in using **lc_checkin**. If the process that calls **lc_checkout** exits in any manner, then the checked-out license will be checked in; however this is not recommended.

If the license file is *counted*, that is, if the number of users specified on the FEATURE line is non-zero, **lc_checkout** will request the license from a license server. If the number of users on the FEATURE line is *uncounted*, it will grant permission based on the contents of the license file only—hostid, version, expiration date, and so on.

The license must specify a version that is greater than or equal to the version in the call to **lc_checkout**.

For improved security, it is recommended that the parameters `feature`, `version`, and so on be hidden, meaning that the string should not be directly declared in source code. It should be built up chars or smaller strings, and then created using **sprintf**. That way, it is more difficult for users to change the license being checked out by altering the string in the binary.

The behavior when checkout is denied is defined by the Checkout Option Flag.

Note • *After successful checkout the location of the license file is saved in the registry. This is a known limitation of setting LM_LICENSE_FILE environment variable for Trusted Storage.*

Using Information in the License For Checkout

Use a checkout filter before calling **lc_checkout** if you want a checkout to depend on information in the license. Call **lc_set_attr** specifying one of the checkout filter routines (LM_A_MULTIPLE_CHECKOUT_DATA, LM_A_CHECKOUTFILTERLAST_EX)before the call to **lc_checkout**.

Using Information in the License After Checkout

If you want to display information to the user from a license line, call **lc_auth_data** after a successful checkout. Because **lc_auth_data** returns only features which have been successfully checked out, the returned data is authenticated. If you call **lc_checkout** with the `LM_CO_LOCALTEST` flag, use the alternate function **lc_test_conf** to retrieve the desired license line instead of **lc_auth_data**.

Checkout Option Flag

The checkout option flag, the parameter `flag` in **lc_checkout**, controls the response to a checkout denial. The following table describes the `flag` values.

Table 2-23 • lc_checkout flag Parameter Values

flag Parameter Value	Description
LM_CO_NOWAIT	Do not wait—non-blocking. If the requested license count for a feature exceeds the available license count, the checkout is denied.
LM_CO_AVAIL_NOWAIT	Do not wait—non-blocking. If the number of requested licenses for a feature exceeds the number of available licenses in a single satisfying license pool on the server, the available number of licenses is checked out from the first satisfying license pool. For more information about this flag, see Using the LM_CO_AVAIL_NOWAIT Flag.
LM_CO_WAIT	Wait, return when license is granted—*blocking*.
LM_CO_QUEUE	Queue request, return immediately. This request grants the license if it is available. If the number of requested licenses exceeds the number of available licenses, the request is queued. If the requested licenses exceed the number of licenses permitted by the `MAX` keyword, the request is denied. You can find out if you hold the license by calling **lc_status**. If there are multiple license pools, this queues from only the first license pool in the list.

Table 2-23 • lc_checkout flag Parameter Values

flag Parameter Value	Description
LM_CO_LOCALTEST	With this flag, the license is tested for errors that would prevent a license checkout; a license is not consumed when **lc_checkout** is used with this flag. In particular, `LM_MAXUSERS` and `LM_USERSQUEUED` are not detected, but other checkout errors are detected. For example, you might want to disable a menu item if a particular feature is not present in the license. If you call **lc_checkout** with the `LM_CO_LOCALTEST` flag, use the function **lc_test_conf** to retrieve the license line for the tested feature (**lc_auth_data** does not work with the `LM_CO_LOCALTEST` flag). This can be done only after the most recent call to **lc_checkout**. Testing a license with this flag causes a connection to the license server that lasts as long as the current job. If a copy of a server license file is used on the client to provide the server address, then `USE_SERVER` should be used in the license file in order for `LM_CO_LOCALTEST` to connect to the license server. Therefore, the job in which the test is performed is irrevocably connected to a specific license server, which may not have any count available for subsequent license requests. Consider testing this in a separate job.

Using the LM_CO_AVAIL_NOWAIT Flag

The `LM_CO_AVAIL_NOWAIT` flag can help to avoid checkout denials that can occur despite a sufficient number of licenses being available.

A client example for the `LM_CO_AVAIL_NOWAIT` flag is available in the FlexNet Publisher Licensing Toolkit (`<platform_dir>\examples\advanced\client\lmflex_coavail.c`).

Behavior for Multiple Checkouts

The behavior of the `LM_CO_AVAIL_NOWAIT` flag in scenarios of multiple calls to **lc_checkout** is best described by contrasting it with its sibling flag, `LM_CO_NOWAIT`.

LM_CO_NOWAIT

When using `LM_CO_NOWAIT`, multiple checkout requests of the same feature in the same license job do not result in additional licenses being checked out, unless a new request specifies more licenses than the previously checked-out licenses. For example, two calls to `lc_checkout(lm_job,"f1",..,1,..)` result in only one license for feature f1 being checked out.

In scenarios with multiple calls to **lc_checkout**, where the number of requested licenses vary, all subsequent checkout requests are treated as upgrades that replace the initially requested license count. The following example demonstrates this behavior.

Let's assume that an application contains two calls to **lc_checkout** as in the following code snippet:

```
...
if(lc_checkout(lm_job, feature, "1.0", 1, LM_CO_NOWAIT, &code, LM_DUP_NONE))
...
if(lc_checkout(lm_job, feature, "1.0", 3, LM_CO_NOWAIT, &code, LM_DUP_NONE))
```

```
...
```

In this example, the first checkout consumes one license. The second checkout consumes two licenses, bringing the total license consumption for this client after two checkouts to three (as requested by the second call to **lc_checkout**).

The corresponding debug log snippet might look like this:

```
13:37:16 (demo) OUT: "f1" User1@Host1
13:37:16 (demo) UPGRADE: "f1" User1@Host1 (1->3 licenses)
```

LM_CO_AVAIL_NOWAIT

When using `LM_CO_AVAIL_NOWAIT`, the `nlic` parameter of **lc_checkout** is an incremental license count, that is, multiple checkout requests from the same license job of the same feature result in additional licenses being checked out. For example, two calls to `lc_checkout(lm_job,"f1",..,1,..)` consume two licenses for feature f1, one from each checkout. The following example demonstrates this behavior.

Let's look at the following sequence of calls to **lc_checkout**:

```
...
if(lc_checkout(lm_job, feature, "1.0", 1, LM_CO_AVAIL_NOWAIT, &code, LM_DUP_NONE))
...
if(lc_checkout(lm_job, feature, "1.0", 3, LM_CO_AVAIL_NOWAIT, &code, LM_DUP_NONE))
...
```

Here, the first COAVAIL checkout consumes one license, and the second COAVAIL checkout consumes three additional licenses. The total license consumption for this client after two checkouts is four (compare this to the license consumption with the `LM_CO_NOWAIT` flag, where the total license consumption is three).

The corresponding debug log snippet might look like this:

```
13:35:55 (demo) COAVAIL: (A:4,1=>1)
13:35:55 (demo) OUT: "f1" User1@Host1
13:35:55 (demo) COAVAIL:UPG2 (A:3,3=>3)
13:35:55 (demo) UPGRADE: "f1" User1@Host1 (1->4 licenses)
```

To consume licenses from multiple license pools, more than one checkout request using `LM_CO_AVAIL_NOWAIT` must be issued (one for each license pool). A client example for the `LM_CO_AVAIL_NOWAIT` flag is available in the FlexNet Publisher Licensing Toolkit (`<platform_dir>\examples\advanced\client\lmflex_coavail.c`).

Combining LM_CO_NOWAIT and LM_CO_AVAIL_NOWAIT in the Same License Job

The `LM_CO_AVAIL_NOWAIT` flag can be combined only with the `LM_CO_NOWAIT` flag in any given license job. The order of the checkout flags determines the number of licenses being checked out.

Example 1

```
if(lc_checkout(lm_job, feature, "1.0", 1, LM_CO_AVAIL_NOWAIT, &code, LM_DUP_NONE))
if(lc_checkout(lm_job, feature, "1.0", 3, LM_CO_NOWAIT, &code, LM_DUP_NONE))
```

This would result in a total of three licenses being checked out: one license would be checked out by the first call to **lc_checkout**, and two additional licenses would be checked out by the second call to **lc_checkout**.

Example 2

```
if(lc_checkout(lm_job, feature, "1.0", 1, LM_CO_NOWAIT, &code, LM_DUP_NONE))
if(lc_checkout(lm_job, feature, "1.0", 3, LM_CO_AVAIL_NOWAIT, &code, LM_DUP_NONE))
```

This would result in a total of four licenses being checked out: one license would be checked out in the first call to **lc_checkout**, and three licenses would be checked out in the second call to **lc_checkout**.

Retrieving the Actual License Consumption

In a checkout using the `LM_CO_AVAIL_NOWAIT` flag, if a license pool does not contain the requested number of licenses for the requested feature, the checkout request is partially granted, and the available number of licenses in that license pool is checked out. In this scenario, the client must keep a record of the number of licenses that have been checked out. Hence, the client needs to know how many licenses have been granted by the license server. To retrieve the number of granted licenses, use the function **lc_auth_data**. It retrieves the CONFIG that is associated with the last successful checkout and gets the actual license consumption from the `nCOAvailLastLicCount` member.

Note the following when using the CONFIG member `nCOAvailLastLicCount`:

- The `nCOAvailLastLicCount` member reflects the actual license count that is consumed on the license server with the last successful checkout.
- On a successful checkout, `nCOAvailLastLicCount` is `1 <= nCOAvailLastLicCount <= nlic,` where `nlic` is the requested license count of **lc_checkout**.
- The `nCOAvailLastLicCount` member is applicable to served CONFIGS only (`config type` is `FNP_CONFIG_TYPE_SERVED_CERT` or `FNP_CONFIG_TYPE_SERVED_TS`).
- When multiple checkouts of the same feature from the same license job have been issued, the `nCOAvailLastLicCount` member reflects the actual license consumption of the last successful checkout and not the cumulative license consumption.

  Example for multiple checkouts

  Let's assume there are six licenses available on the license server for feature f1 (two licenses for v1.0, four licenses for v2.0). The following two checkouts are performed with the `LM_CO_AVAIL_NOWAIT` flag from the same license job:

  - First checkout request: two licenses (`nlic=2`) for feature f1, v1.0.
    - The license server grants two licenses from the v1.0 license pool (the total license consumption for this client on the server is two).
    - The client calls **lc_auth_data** to get the CONFIG associated with this checkout; the CONFIG member `nCOAvailLastLicCount` reflects 2. The actual license consumption for this checkout is two.
  - Second checkout request: ten licenses (`nlic=10`) for feature f1, v1.0.
    - The license server grants four licenses from the v2.0 license pool (hence the total license consumption for this client on the server is six).
    - The client calls **lc_auth_data** to get the CONFIG associated with this checkout; the CONFIG member `nCOAvailLastLicCount` reflects 4. The actual license consumption for this checkout is four.

See Also
LM_A_USER_RECONNECT_V1_DATA
LM_A_USER_RECONNECT_V1
LM_A_USER_RECONNECT_V1_DONE
Information about the COAVAIL checkout in the *Programming Reference for License File–Based Licensing*

Duplicate Grouping Mask

Requests for licenses from “duplicates” can either be “grouped” or “not grouped.” Grouping duplicates allows license requests from separate processes to use a single license if the process’s USER, HOST, DISPLAY, and/or VENDOR_DEFINED field are the same. Duplicate grouping is valid only with counted licenses.

The dup_group parameter enables you to select what to compare to constitute a group from the set {USER HOST DISPLAY VENDOR}. Any of the four fields that are not set to compare will automatically match; thus, not setting any of the four fields yields a site license because all users on all hosts on all displays are the same as far as the comparison is concerned.

The following examples illustrate the use of the duplicate grouping capability:

Table 2-24 • lc_checkout Duplicate Grouping Examples

Value	Meaning
LM_DUP_NONE	Every job handle gets a new license.
LM_DUP_USER	All requests from this user name share the same license.
LM_DUP_HOST	All requests from this host name share the same licenses. This is a floating node-locked license.
LM_DUP_DISP	All requests from this display share the same license. (Useful for display or GUI-based products, like a window system.)
LM_DUP_VENDOR	All requests with the same publisher-defined data, use the same license. (Useful for sharing licenses among otherwise unrelated processes.) If LM_DUP_VENDOR is used, LM_A_CHECKOUT_DATA must be set.
LM_DUP_USER | LM_DUP_HOST	All requests from this user name on this host name use the same license.
LM_DUP_USER | LM_DUP_DISP	All requests from this user name on this display use the same license. (One user, displaying on a single node, using several nodes to run the software.)
LM_DUP_USER | LM_DUP_HOST | LM_DUP_DISP	All requests from this user name on this host name using this display use the same license.
LM_DUP_USER | LM_DUP_VENDOR	All requests from this user name with the same vendor data use the same license. If LM_DUP_VENDOR is used, LM_A_CHECKOUT_DATA must be set.
LM_DUP_SITE	All requests from any user on any node on any display with any vendor data use the same license (site license).

RESERVE and DUP_GROUP Interaction During Checkout

There is an important interaction between RESERVE and the duplicate grouping mask. A license reservation for an entity not contained in the duplicate grouping mask in the call to **lc_checkout** (for example, a USER reservation) when the duplicate grouping mask is set to LM_DUP_HOST | LM_DUP_DISP can cause an interesting interaction at run time.

To understand why this is the case, consider the following examples:

- Your software groups duplicates based on `USER` and `DISPLAY`.
- Your end user has a license file with a single license.
- Your end user reserves this license for `HOST` "nodea."
- User "joe" on display "displaya" on `HOST` "nodea" checks out a license. He gets the license because his `HOST` matches the reservation.
- User "joe" on display "displaya" on `HOST` "nodeb" checks out a license. He also gets a license, because he is grouped with the first license as a duplicate.
- The first user (joe/displaya/nodea) checks in his license.

At this point in the example, the situation appears to be inconsistent. The second user continues to hold the reservation, which means that a user on "nodeb" is using a license reserved for "nodea." Once this second user checks in the license, the reservation will return to the pool of reservations to be used by anyone on "nodea."

FlexNet Licensing was designed to allow this potential temporary inconsistency rather than the alternative, which is to have an unusable reservation.

See Also
`machind/lmflex.c`
lc_auth_data
LM_A_CHECKOUT_DATA
lc_set_attr
LM_A_MULTIPLE_CHECKOUT_DATA
LM_A_CHECKOUTFILTERLAST_EX
LM_A_LICENSE_DEFAULT
LM_A_HOST_OVERRIDE
LM_A_DISPLAY_OVERRIDE
LM_A_USER_OVERRIDE
lc_checkin
lc_status
lc_new_job

lc_cryptstr

Generates a valid signature in a feature line.

Syntax

```
int lc_cryptstr(
    LM_HANDLE *job,
    char *str,
    char **return_str,
    VENDORCODE *code,
    int flag,
    char *filename,
    char **errors);
```

Parameters

The **lc_cryptstr** function uses the following parameters:

Table 2-25 • lc_crypstr Parameters

Parameter	Description
lm_job	From **lc_init**.
str	Set `str` to a complete valid license file, where the signatures are replaced with `SIGN=0`.
return_str	Resulting license file string. Memory is allocated by **lc_cryptstr** and freed by the calling program, use **lc_free_mem** on Windows. Pass the address of a `char` pointer.
code	From `LM_CODE` macro.
flag	Mask that can be binary OR'd (|) using the following flags: ● `LM_CRYPT_ONLY`—If true, only return signature for first `FEATURE` in `str`. ● `LM_CRYPT_FORCE`—If set, recompute the signature for *every* line, even if the signature is already present on the line. ● `LM_CRYPT_IGNORE_FEATNAME_ERRS`—If set, no warnings returned about invalid feature names.
filename	For error reporting, or `(char *)`0. This name will appear in the error message as the file name.
errors	For error reporting, or `(char **)`0. If there are errors, the return value is non-zero and `errors` is set to an explanatory string. Memory is allocated by **lc_cryptstr**, and freed by the calling program (use **lc_free_mem** on Windows). Pass the address of a `char` pointer. If a warning occurs, `errors` is set to a warning string, but the return value is 0 (success).

Note • *It is recommended to use separate job-handle for every license generation request.*

Note • *The lc_cryptstr API is not multithread safe, hence, synchronization between threads is the responsibility of the application code.*

Return

The following values are returned by **lc_cryptstr:**

Table 2-26 • lc_cryptstr Returns

Return Values	Description
0	Success

Table 2-26 • lc_cryptstr Returns

Return Values	Description
<> 0	Error, see Error Returns for details.

Error Returns

Because different errors can occur on every line of the input `str`, **lc_cryptstr** must be able to report all these errors independently, and does so using the `errors` parameter. The `errors` parameter is used for both errors and warnings. If there is an error, **lc_cryptstr** returns non-zero, and no signatures are generated in `return_str`. If there are only warnings, the return value from **lc_cryptstr** is success (0), but `errors` is set to a warning message.

Here is an example of error reporting:

Input

```
FEATURE f1 demo 1.a50 01-jan-2016 uncounted HOSTID=08002b32b161 SIGN=0
```

Error Reported

```
stdin:line 1:Bad version number - must be floating point number,
             with no letters
```

With this error, no signature is generated, and `return_str` will be the same as the input `str`.

Discussion

Signs the given license file. This function is used by license-file generation utilities such as `lmcrypt` and, as such, is not intended to be incorporated into your application. You pass, via the `str` parameter, a license-file template, which is a complete, valid unsigned (`SIGN=0`) license file containing one or more `FEATURE`, `INCREMENT`, `PACKAGE`, or `UPGRADE` lines. A signed license file is returned, via `return_str`. For example, consider the following license-file template as input through `str`:

```
FEATURE f1 demo 1.5 01-jan-2016 uncounted HOSTID=08002b32b161 SIGN=0
```

lc_cryptstr calculates the signature and returns the complete, valid license file in `return_str`:

```
FEATURE f1 demo 1.5 01-jan-2016 uncounted HOSTID=08002b32b161 SIGN=2C817A5100D8
```

If *flag* has `LM_CRYPT_ONLY` set, then the function returns the signature for the first `FEATURE`, `INCREMENT`, `PACKAGE`, or `UPGRADE` line in the license file. If `LM_CRYPT_ONLY` is cleared, then the entire license file is signed and returned as a string. If *flag* has `LM_CRYPT_FORCE` set, then the signature is recomputed for every line, even if the signature is defined.

Comment lines are retained in the `return_str` output.

If you want to specify a start date, then add the following keyword to the license line template using the following syntax:

```
START=dd-mmm-yyyy
```

Example:

```
START=1-jan-2017
```

The output is compatible with the `LM_A_LICENSE_FMT_VER` setting, which defaults to the current FlexNet Publisher version.

The complete source for the `lmcrypt` command is provided in the Licensing toolkit package in `machind/lmcrypt.c`. This as well as the coding example below provide usage information for **lc_crypstr**.

Coding Example

```
[global include and variable info:]
#include "lmprikey.h"
#include "lmclient.h"
#include "lm_code.h"
#include "lmseeds.h"
#include "lm_attr.h"

LM_HANDLE *lm_job;
LM_CODE(code, ENCRYPTION_SEED1, ENCRYPTION_SEED2, VENDOR_KEY1,
        VENDOR_KEY2, VENDOR_KEY3, VENDOR_KEY4, VENDOR_KEY5);
[...]

[C code:]
char *errors;
char *return_str;
int flag = LM_CRYPT_FORCE;
char *filename = "myfile.lic";
char str[MAX_CONFIG_LINE *100]; /* if maximum license is 100 lines */
[...]
[set up str variable with valid license syntax]
[...]

/* License Generator initialization */

/* The LM_CODE_GEN_INIT() is called once during license generator
initialization. Do not call this macro more than once during
the execution of the license generator */

LM_CODE_GEN_INIT(&code);
if (lc_init(0, VENDOR_NAME, &code, &lm_job))
{
    /* present error */
}
/* Initialization complete */
[...]
if (lc_cryptstr(lm_job, str, &return_str, &code, flag, filename, &errors))
{
    /* present error, and if non-null, print it out */
}
if (return_str)
{
     /* return_str is the correct license-file string */
}
```

See Also

lc_check_key
lc_get_attr
lc_init
LM_A_LICENSE_FMT_VER
machind/lmcrypt.c

Linking your License Generator Utility

Note • *Your license generator must be linked statically to the Licensing toolkit client library.*

For Windows Platforms

When you have successfully compiled your license generator utility, link with the libraries listed in Table 2-27. These libraries are located in the `<platform_dir>` directory. Additionally, refer to the rules for building the `lmcrypt` license generator in `<platform_dir>/makefile`.

Table 2-27 • License Generator Utility Libraries

Licensing toolkit Client Libraries	
/MT or /MD	**Windows System Libraries**
• lmgr_trl.lib ***Note*** • *Use a non-TRL library (lmgr.lib) only if you have active legacy clients in production older than v10.8.6 that will be served by the same license server as your new clients. For more information about TRL, see the Programming Reference for License File–Based Licensing, chapter 9, "License Signature Configurations".* • libcrvs.lib • libsb.lib	• libcmt.lib (for MT) or msvcrt.lib (for /MD) • Additional required libraries as listed in the `<platform_dir>/makefile` file

For UNIX Platforms

When you have successfully compiled your license generator, link with a command similar to the following—

```
cc -o lic_gen lic_gen.o -L$platform liblmgr_trl.a libcrvs.a libsb.a $(OSLIBS)
```

—with the following notes:

- `lic_gen` is your license generator utility that incorporates **lcryptstr**.
- `$platform` is the `<platform_dir>` directory in the production FlexNet Publisher Licensing Toolkit that contains the necessary Licensing toolkit client libraries.
- `$(OSLIBS)` is a list of platform-specific libraries. Refer to the rules for building the `lmcrypt` license generator in `<platform_dir>/makefile` for the list specific to your platform.

lc_err_info

Returns a pointer to an `LM_ERR_INFO` structure that contains error information.

Syntax

```
LM_ERR_INFO_PTR lc_err_info(LM_HANDLE *job);
```

Parameters

The **lm_err_info** function uses the following parameters:

Table 2-28 • lm_err_info Parameters

Parameter	Description
job	From **lc_new_job**.

Return

Pointer to the LM_ERR_INFO struct.

Discussion

Returns a pointer to a `LM_ERR_INFO` struct, which contains all necessary information to present an error message to the user. This is the supported method for internationalization and localization of Licensing toolkit error messages.

This information allows applications to present error messages in any language and in any desired format. The Licensing toolkit provides the English text of all messages. The three types of messages that need to be translated include:

- short—located in `machind/lmerrors.h`
- long—located in `machind/lm_lerr.h`
- context—located in `machind/lcontext.h`

Use the value `err_info.maj_errno` as an index into these files in order to retrieve the corresponding message.

See Also
lc_perror
lc_get_errno

LM_ERR_INFO

The format of `LM_ERR_INFO` is:

Table 2-29 • LM_ERR_INFO Format

Format	Description
maj_errno	The Flexible API error number. See `lmerrors.h` and `lm_lerr.h` in the `machind` directory for English versions of the error messages.
min_errno	The minor error number. This enables technical support to better identify the error.
sys_errno	The most recent system `errno` (or Winsock error on Windows).

Table 2-29 • LM_ERR_INFO Format

Format	Description
feature	The name of the feature that the error applies to.
lic_files	A null-terminated array of `char` pointers of the license files used when the error occurred.
context	A string that provides additional information about the error. Its contents depend on the type of error, but is not language dependent. Refer to `machind/lcontext.h` for information needed for translation.

lc_errstring

Returns the Flexible API error string for the most recent Flexible API error.

Syntax

```
LM_CHAR_PTR lc_errstring(LM_HANDLE *job);
```

Parameters

The **lc_errstring** function uses the following parameters:

Table 2-30 • lc_errstring Parameters

Parameter	Description
job	From **lc_new_job**.

Return

The Flexible API error string text.

Discussion

Returns the Flexible API error string for the most recent Licensing error, along with the major and minor error number. If a UNIX error is involved, the UNIX error description will also be included in the message, along with the UNIX `errno`. For internationalization of error messages, use **lc_err_info**.

This memory is managed by the Licensing toolkit library. Do not attempt to free it. This string is freed and reset when another Licensing error occurs, so it is only valid between calls to Flexible API functions. Check that the call to the previous Licensing function has returned an error before calling **lc_errstring**.

Examples

```
No such feature exists (-5,116)
Cannot find license file, (-1,73:2), No such file or directory
```

See Also
lc_perror
lc_err_info
lc_get_errno

lc_expire_days

Returns the number of days until a license expires.

Syntax

```
long lc_expire_days(
    LM_HANDLE *job,
    CONFIG *conf);
```

Parameters

The **lc_expire_days** function uses the following parameters:

Table 2-31 • lc_expire_days Parameters

Parameter	Description
job	From **lc_new_job**.
conf	A FEATURE line from the license file. If security is important and the FEATURE line information should be authenticated, use **lc_auth_data** to obtain the CONFIG pointer after a successful checkout. If the FEATURE line information needs to be authenticated before a checkout is done, use **lc_checkout** with the LM_CO_LOCALTEST flag, then **lc_test_conf** can be used to obtain the CONFIG pointer. If data authentication is not needed, **lc_next_conf** can be used to get the CONFIG pointer.

Return

The following values are returned by **lc_expire_days**:

Table 2-32 • lc_expire_days Returns

Return Value	Description
> 0	Number of days until expiration.
==0	The license will expire tonight at midnight.
< 0	Flexible API errno.

Table 2-32 • lc_expire_days Returns

Return Value	Description
LM_FOREVER	Unexpiring license.

Error Returns

The following values are error returns for the function **lc_expire_days**:

Table 2-33 • lc_expire_days Error Returns

Return Value	Description
LM_BADPARAM	conf is 0.
LM_LONGGONE	The feature has already expired.

See Also
lc_checkout
lc_next_conf

lc_feat_list

Gets the list of all features in the license file

Syntax

```
char **lc_feat_list(
    LM_HANDLE *job,
    int flags,
    void (*dupaction)(void));
```

Parameters

The **lc_feat_list** function uses the following parameters:

Table 2-34 • lc_feat_list Parameters

Parameter	Description
job	From **lc_new_job**.
flags	LM_FLIST_ALL_FILES for all license files. If 0, only the first license in the license search path is used. LM_FILTERLIST_ALL_FILES will only return the valid features that are served by the server (Vendor Specific).
dupaction	Action routine called when a duplicate feature is found. This routine is called upon the second occurrence of any feature name. If specified as NULL, no call is made.

Return

List of features. `list` is a pointer to a NULL-terminated array of feature string pointers. Both the pointers and the string data are malloc'd; this memory is freed upon a subsequent call to **lc_feat_list**. Do not free this data. If NULL, an error has occurred.

Error Returns

The following values are error returns for the function **lc_feat_list**:

Table 2-35 • lc_feat_list Error Returns

Parameter	Description
LM_CANTMALLOC	The call to **malloc** failed.
LM_NOFEATURE	Specified feature not found.

Discussion

Gets the list of all features in the license file. The feature list is not authenticated and includes package names and components as well as feature names.

lc_first_job

lc_first_job is used to find the first job in a list of jobs.

Syntax

```
LM_HANDLE *lc_first_job(LM_HANDLE *job);
```

Parameter

The **lc_first_job** function uses the following parameters:

Table 2-36 • lc_first_job Parameters

Parameter	Description
job	A pointer to the first job is returned into this parameter.

Discussion

Before calling this function, you must set up a list of jobs by linking jobs using the *oldjob* parameter in **lc_new_job**.

Coding Example

```
LM_HANDLE *job
job = lc_first_job(job);
while (job)
{
    /*processing*/
```

```
    job = lc_next_job(job);
}
```

See Also
lc_free_job
lc_new_job
lc_next_job

lc_flexinit

This function loads and initializes the activation library component.

Syntax

```
int lc_flexinit(struct flexinit_property_handle *handle);
```

Parameters

The **lc_flexinit** function uses the following parameter values:

Table 2-37 • lc_flexinit Parameters

Parameter	Description
handle	A property handle created with **lc_flexinit_property_handle_create**. See **lc_flexinit_property_handle_set** for a description of available properties.

Return Values

The **lc_flexinit** function returns the following values

Table 2-38 • lc_flexinit Return Values

Return Value	Description
0	Success.
-42	Invalid or missing property handle.
1	Activation library component not found.
2	Unable to load activation library component.
3	OS version too old: unable to load activation library component.
4	Initialization of activation library failed.
5	Insufficient resources to load activation library component.

Table 2-38 • lc_flexinit Return Values

Return Value	Description
6	Activation library component has not been created using the `preptool`. For standard toolkits, the activation library component must be created specifically for the FlexEnabled application using the `preptool`. ***Note*** *• For license-file-based licensing, this component, also known as the run-time component, handles Secure Data Types (SDTs) in applications.*
7	Activation library component has been unloaded. Once unloaded it cannot be reloaded until the FlexEnabled application is restarted.
8	Mac-specific error. Path to activation library component not specified in correct format, UTF-8.
9	Mac-specific error. Error creating URL for activation library component.
10	Mac-specific error. Error creating path from URL for activation library component.
11	Mac-specific error. Framework specified by bundle ID not loaded.
12	Mac-specific error. Invalid bundle ID.
13	Mac-specific error. Computed path to activation library component is too long.
20	The FlexNet Licensing Service is not installed.
21	Not enough rights to talk to service. Set service to start automatically to resolve this.

Discussion

This function will be successful only if the FlexNet Licensing Service has previously been installed. On Windows, the FlexNet Licensing Service API includes functions that install and uninstall the service. On other platforms, a script is provided. See "Distributing the FlexEnabled Application for Trusted Storage-based Licensing" in the *Development Environment Guide* for details.

Before calling this function, the property handle must be created with **lc_flexinit_property_handle_create** and a property value set into the handle with **lc_flexinit_property_handle_set**.

Place this function in your code before the first call to **lc_new_job**. This ensures that activation support is enabled and that the tamper-resistant runtime is running when required: code that uses Secure Data Types (SDTs) does not operate if the tamper-resistant runtime is not operating. This also ensures that any antidebugger technology checks are operational all the time that licensing functions are being used. It must be paired with a matching call to **lc_flexinit_cleanup** and called before the application exits.

This function must be called once and only once by the application at startup. If you are using the Activation API and the Flexible API within a single executable, you should ensure that you call either this function or **flxActCommonLibraryInit**, but not both. flxActCommonLibraryInit also loads the activation library and additionally allows you to specify the location of the activation library.

This function fails if the activation library is not correctly located. The location of the activation library component is defined when the executable that uses it is processed with the `preptool`. For details of about the `preptool` utility, see the *Programming Reference for Trusted Storage–Based Licensing*.

Caution • *This function is not multi-thread safe: synchronization between threads is the responsibility of the application code. There is an additional restriction on platforms other than Windows and Mac: this function must be called in the main thread of any multi-threaded applications.*

Note • *If you want to use the Activation API and the Flexible API within a single executable, ensure that you call either flxActCommonLibraryInit or lc_flexinit, but not both.*

lc_flexinit_cleanup

This function unloads the activation library and performs any cleanup functions required.

Syntax

```
int lc_flexinit_cleanup(struct flexinit_property_handle *handle);
```

Parameters

The **lc_flexinit_cleanup** function uses the following parameters:

Table 2-39 • lc_flexinit_cleanup Parameters

Parameter	Description
handle	The property handle used in the matching call to **lc_flexinit**. See **lc_flexinit_property_handle_set** for a description of available properties.

Return Values

The **lc_flexinit_cleanup** function returns the following values:

Table 2-40 • lc_flexinit_cleanup Return Values

Return Value	Description
LM_NOERROR	Success.
LM_BADPARAM	Invalid or missing property handle.

Discussion

There must be a matching **lc_flexinit** call in your code that was invoked prior to calling this function. After calling **lc_flexinit_cleanup,** free the property handle using **lc_flexinit_property_handle_free**.

When using the tamper-resistant runtime or activation, you must ensure that **lc_flexinit_cleanup** is called only after all licensing functions have been completed. The tamper-resistant runtime is required for ADT checks and Secure Data Types (SDTs) to operate. All features that are linked to SDTs must be checked in with **lc_checkin** before the call to **lc_flexinit_cleanup**.

lc_flexinit_cleanup should be called only as a matching call to lc_flexinit. If flxActCommonLibraryInit was used instead of lc_flexinit, then call flxActCommonLibraryCleanup instead of lc_flexinit_cleanup.

lc_flexinit_property_handle_create

This function creates a property handle.

Syntax

```
int lc_flexinit_property_handle_create(struct flexinit_property_handle **handle);
```

Parameters

The **lc_flexinit_property_handle_create** function uses the following parameters:

Table 2-41 • lc_flexinit_property_handle_create Parameters

Parameter	Description
handle	The property handle used in the following **lc_flexinit** calls: • **lc_flexinit** • **lc_flexinit_cleanup** • **lc_flexinit_property_handle_set** • **lc_flexinit_property_handle_get**

Return Values

The **lc_flexinit_property_handle_create** function returns the following values:

Table 2-42 • lc_flexinit_property_handle_create Return Values

Return Value	Description
LM_NOERROR	Success.
LM_CANTMALLOC	Could not allocate sufficient memory.

Discussion

Call this function to create a property handle for use to initialize and cleanup a FlexEnabled application. After calling **lc_flexinit_property_handle_create**, call **lc_flexinit_property_handle_set** to define the initialization and cleanup functionality required.

lc_flexinit_property_handle_free

This function frees an initialization property handle.

Syntax

```
int lc_flexinit_property_handle_free(struct flexinit_property_handle *handle);
```

Parameters

The **lc_flexinit_property_handle_free** function uses the following parameters:

Table 2-43 • lc_flexinit_property_handle_free Parameters

Parameter	Description
handle	A property handle created with **lc_flexinit_property_handle_create**.

Return Values

The **lc_flexinit_property_handle_free** function returns the following values:

Table 2-44 • lc_flexinit_property_handle_free Return Values

Return Values	Description
LM_NOERROR	Success.

Discussion

Call this function to free a property handle after calling **lc_flexinit_cleanup**.

lc_flexinit_property_handle_get

This function returns the value for a specified property within the supplied initialization property handle.

Syntax

```
int lc_flexinit_property_handle_get(
    struct flexinit_property_handle *handle,
    FLEXINIT_PROPERTY_TYPE property,
    FLEXINIT_VALUE_TYPE *value);
```

Parameters

The **lc_flexinit_property_handle_get** function uses the following parameters:

Table 2-45 • lc_flexinit_property_handle_get Parameters

Parameter	Description
handle	A property handle created with **lc_flexinit_property_handle_create**.
property	Defines the initialization and cleanup functions. The only available property is `INIT_PROPERTY_USE_TRUSTED_STORAGE`—Perform initialization and cleanup for applications using activation and tamper resistant runtime.
value	The value setting for *property*

Return Values

The **lc_flexinit_property_handle_get** function returns the following values:

Table 2-46 • lc_flexinit_property_handle_get Return Values

Return Value	Description
LM_NOERROR	Success.
LM_BADPARAM	Invalid *property*.

Discussion

Use this function to obtain the current property value setting in the property handle.

lc_flexinit_property_handle_set

This function sets a type and value into the initialization property handle that defines the initialization and cleanup functionality required.

Syntax

```
int lc_flexinit_property_handle_set(
    struct flexinit_property_handle *handle,
    FLEXINIT_PROPERTY_TYPE property,
    FLEXINIT_VALUE_TYPE value);
```

Parameters

The **lc_flexinit_property_handle_set** function uses the following parameters:

Table 2-47 • lc_flexinit_property_handle_set Parameters

Parameter	Description
handle	A property handle created with **lc_flexinit_property_handle_create**.
property	The property to set. For possible property types see Discussion.
value	The value to assign to *property*. For possible settings, see Discussion.

Return Values

The **lc_flexinit_property_handle_set** function returns the following values:

Table 2-48 • lc_flexinit_property_handle_set Return Values

Return Value	Description
LM_NOERROR	Success. Invalid property.
LM_BADPARAM	Invalid *property*.

Discussion

Before calling this function, the property handle must be created using **lc_flexinit_property_handle_create**.

Call this function to define the property to which initialization and cleanup functionality is required, where `property` can be:

`INIT_PROPERTY_USE_TRUSTED_STORAGE`

If `value=1`, specifies initialization and cleanup for applications using activation and tamper-resistant runtime.

If `value=0`, the property is undefined.

lc_fnpservice_present

lc_fnpservice_present provides the status of whether the FlexNet Licensing Service is present.

Syntax

```
int lc_fnpservice_present(LM_HANDLE_PTR *job);
```

Parameters

The **lc_fnpservice_present** function uses the following parameters:

Table 2-49 • lc_fnpservice_present Parameters

Parameter	Description
job	From **lc_new_job**.

Return

The **lc_fnpservice_present** function returns the following values:

Table 2-50 • lc_fnpservice_present Return Values

Return Value	Desciption
> 0	FlexNet Licensing Service is present.
0	FlexNet Licensing Service is not present.

Discussion

This function is only supported for Windows and Linux platforms. For all other platforms, it will always return 0.

lc_free_job

lc_free_job frees the memory associated with a job that has been allocated by **lc_new_job**.

Syntax

```
void lc_free_job(LM_HANDLE *job);
```

Parameters

The **lc_free_job** function uses the following parameters:

Table 2-51 • lc_free_job Parameters

Parameter	Description
job	From **lc_new_job**.

Return

There are no values are returned by **lc_free_job**.

Discussion

lc_free_job removes everything associated with *job*—this includes any timers, memory, and licenses. It is recommended that you call **lc_checkin** first.

For dynamically linked Windows applications or FlexEnabled DLLs, call **lc_cleanup** after the last call to **lc_free_job.** Any features still checked out will be checked in; however, it is recommended that all features are checked in using **lc_checkin** before calling **lc_free_job.**

Note • *Do not call this API more than once from an application (for example, following a checkin and then later during the destructor, or exit, routine). Multiple* lc_free_job *calls can cause the application to crash.*

See Also
lc_init
lc_new_job
lc_set_attr
lc_cleanup (Windows only)

lc_get_attr

Retrieves a Flexible API attribute.

Syntax

```
int lc_get_attr(
    LM_HANDLE *job,
    int attr,
    short *value);
```

Parameters

The **lc_get_attr** function uses the following parameters:

Table 2-52 • lc_get_attr Parameters

Parameter	Description
job	From **lc_new_job**.
attr	Which attribute to get.
value	The value of the requested attribute. Must be a pointer to the correct attribute type and should be cast to a `short *`.

Return

The following values are returned by **lc_get_attr**:

Table 2-53 • lc_get_attr Returns

Return Value	Description
0	Success
Any other value	Error.

Error Returns

These are the error returns for the function **lc_get_attr**:

Table 2-54 • lc_get_attr Error Returns

Error Return	Description
LM_NOSUCHATTR	No such attribute exists.
LM_NOADMINAPI	`LM_A_VD_GENERIC_INFO` or `LM_A_VD_FEATURE_INFO` only—request was made to other company's vendor daemon.
LM_NOSERVSUPP	`LM_A_VD_GENERIC_INFO` or `LM_A_VD_FEATURE_INFO` only—pre-v4.0 server does not support these requests.
LM_NOSERV	Returns an error when an uncounted license is called.

Discussion

To use this function include `lm_attr.h`.

Retrieves a Flexible API attribute. `attr` describes which attribute to retrieve, and the value is a pointer to the value for the attribute. See `lm_attr.h` for `attr` constants and value types. Attribute descriptions can also be found in Flexible API Attributes and Advanced Flexible API Attributes.

Types of `char *` are handled a little differently than other types. Types of `int` or `short` are declared, and a pointer to the declared variable is passed as an argument. Types of `char *` are declared as `char *`, and the variable itself is passed.

See Also
lc_set_attr

lc_get_errno

Returns the most recent Flexible API error.

Syntax

```
int lc_get_errno(LM_HANDLE *job);
```

Parameters

The **lc_get_errno** function uses the following parameters:

Table 2-55 • lc_get_errno Parameters

Parameter	Description
job	From **lc_new_job**.

Return

The following values are returned by **lc_get_errno**:

Table 2-56 • lc_get_errno Returns

Return Value	Description
errorno	See `lmclient.h`, `lm_lerr.h`, and `lmerrors.h` for a list of possible Flexible API errors and associated English descriptions.

Discussion

This function returns the most recently set Flexible API error number. This value is available after the return of any Flexible API function.

For more detailed error information that includes the error number, use **lc_err_info** as an alternative to this function.

See Also
lc_err_info
lc_perror
`machind/lmclient.h`

lc_get_version

Returns the version of the Licensing toolkit client library.

Syntax

```
void lc_get_version(FLEX_VERSION *ver);
```

Parameters

The **lc_get_version** function uses the following parameters:

Table 2-57 • lc_get_version Parameters

Parameter	Description
ver	The FLEX_VERSION structure is filled in with the version of the Licensing toolkit client library. This structure contains the following data: • major version • minor version • sub minor version • patch number • build number • beta build indicator (reserved for future use) • patch character (reserved for future use)

Return

There are no values returned by **lc_get_version**.

Error Returns

There are no error returns for the function **lc_get_version**.

See Also
lc_get_server_version

lc_get_server_version

Returns the version of the Licensing toolkit server library.

Syntax

```
int lc_get_server_version(
    LM_HANDLE *job,
    FNP_SERVER_TYPE type,
    FLEX_VERSION *ver);
```

Parameters

The **lc_get_server_version** function uses the following parameters:

Table 2-58 • lc_get_version Parameters

Parameter	Description
job	From **lc_new_job** handle
type	The type can be `lmgrd` or vendor daemon server version as they are currently supported. FNP_SERVER_TYPE_LIC_MANAGER—supports `lmgrd` and `lmadmin` FNP_SERVER_TYPE_VENDOR_DAEMON
ver	The FLEX_VERSION structure is filled in with the version of the Licensing server toolkit. This structure contains the following data: • major version • minor version • sub minor version • patch number

Return

The following values are returned by **lc_get_server_version**:

Table 2-59 • lc_get_server_version Returns

Return Values	Description
0	Success.
Any other value	Non-success.

Discussion

FlexNet Publisher checks for major and minor versions and will not compare the version of sub minor and patch number. As a result, when a v11.13.1.0 client tries to checkout a license from v11.13.0.0 server, LM_SERVOLDVER error is not returned.

See Also
lc_checkout

lc_heartbeat

Exchanges heartbeat messages with the license server.

Syntax

```
int lc_heartbeat(
    LM_HANDLE *job,
    int *ret_num_reconnects,
    int minutes);
```

Parameters

The **lc_heartbeat** function uses the following parameters:

Table 2-60 • lc_heartbeat Parameters

Parameter	Description
job	From **lc_new_job**.
ret_num_reconnects	If null, this argument is ignored. If non-null, and the FlexEnabled application has just successfully reconnected to the server, the return value will be 0 (success), and `ret_num_reconnects` is set to the number of times the FlexEnabled application has reconnected in the last minutes. If this is a large number, it may indicate attempted theft.
minutes	If 0, this argument is ignored. If non-zero, it is used to detect when a server is being started and stopped many times in a short period, which can indicate attempted theft. The reporting period is set with `minutes`.

Return

The following values are returned by **lc_heartbeat**:

Table 2-61 • lc_heartbeat Returns

Return Values	Description
0	Success.
Any other value	The license server is currently down. The number of failed attempts to reconnect to the license server.

Discussion

Before calling **lc_heartbeat**, you must first turn off the automatic heartbeat mechanism before the first license checkout:

```
lc_set_attr(lm_job, LM_A_CHECK_INTERVAL, (LM_A_VAL_TYPE) -1);
lc_set_attr(lm_job, LM_A_RETRY_INTERVAL, (LM_A_VAL_TYPE) -1);
```

lc_heartbeat sends heartbeat messages to and receives acknowledgments from the license server. By default, these activities are handled automatically by FlexNet Licensing via a separate, dedicated application thread. This function provides manual control of heartbeat messages, thereby overriding the automatic mechanism.

The purpose of heartbeat messages is to:

- **Keep license server informed that the application is still using its license**—otherwise the license server may timeout and check in the license.
- **Keep the application informed that the license server is continually running**—unchecked license server stops and starts may indicate unauthorized license usage.

lc_heartbeat performs the following activities:

- It attempts to read the acknowledgment from the previous heartbeat.
- It sends out a heartbeat message, or, if there is no acknowledgment from the previous one, it makes one attempt to reconnect to the license server.
- If reconnection is successful, checks out the original complement of licenses again from the license server.
- It informs the application of a number of states that may indicate attempted tampering with the license server.

Heartbeat messages ensure that licenses are checked out again from a restarted server. Heartbeats are not needed for the server to retain a FlexEnabled application's license—the server retains the license until the FlexEnabled application exits. If **lc_heartbeat** is called, the FlexEnabled application automatically reconnects and checks out again from a license server that has restarted. It also informs the application of a number of states that may indicate attempted tampering with the license server.

- The return value, if non-zero, indicates that the license server is down, and how many reconnect attempts have been made. This information can be used, for example, to inform the end user that the license server is down and possibly to deny use after a specified number of failures.
- The arguments `num_reconnects` and `num_minutes`, if used, can indicate the rate a server has been stopped and started, possibly signifying attempted theft.

How lc_heartbeat Works

lc_heartbeat sends a heartbeat message to the license server. It then reads the acknowledgment from the previously sent heartbeat. The first heartbeat is sent after the application first connects to the license server via **lc_checkout**. In this manner, there is normally no delay in **lc_heartbeat**.

If **lc_heartbeat** is unable to read an acknowledgment from the license server, it makes one attempt to reconnect to the license server according to the following parameters:

- Each subsequent call to **lc_heartbeat** makes one reconnection attempt for a total of five attempts (default). The `LM_A_RETRY_COUNT` attribute is used to override the total number of attempts.
- Calls to **lc_heartbeat** more frequent than 30 seconds are ignored. That is, a reconnection is attempted at most every 30 seconds even when **lc_heartbeat** is called more frequently.
- The application can optionally define a pre-reconnection callback function via the `LM_A_USER_RECONNECT` attribute.
- If a reconnection occurs before five attempts and the application defines a post-reconnection callback function via the `LM_A_USER_RECONNECT_DONE` attribute, it is called.
- If a reconnection fails to occur after five attempts, then a callback exit handler, if specified with the `LM_A_USER_EXITCALL` attribute, is called. Otherwise, the application exits with the error message, "Lost License, cannot reconnect" (UNIX: stderr, WINDOWS: dialog box).

lc_heartbeat and User TIMEOUT Option

If **lc_heartbeat** is not called for an extended period, then the application may lose its license. This can happen for two reasons: the timeout, set with the `LM_A_TCP_TIMEOUT` attribute, has expired, or the end user has set a `TIMEOUT` for this feature in the end-user options file. In both cases, the license server has a timeout associated with the license which gets invoked if **lc_heartbeat** is not called within the timeout interval. Make sure that `LM_A_TCP_TIMEOUT` is large enough to accommodate your usage of **lc_heartbeat**. Similarly, make sure `ls_minimum_user_timeout` in `lsvendor.c` is large enough so that users will not timeout applications that are in use.

If the license is inadvertently released, the next **lc_heartbeat** will automatically reacquire the license if there is still a license available.

For more information about heartbeats or setting vendor variables, see the *Programming Reference for License File–Based Licensing*.

See Also
LM_A_CHECK_INTERVAL
LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL
LM_A_TCP_TIMEOUT
LM_A_USER_EXITCALL, LM_A_USER_EXITCALL_EX
LM_A_USER_RECONNECT, LM_A_USER_RECONNECT_EX
LM_A_USER_RECONNECT_DONE, LM_A_USER_RECONNECT_DONE_EX
TIMEOUT options file keyword in the *License Administration Guide*

lc_hostid

Provides the hostid in string format.

Syntax

```
int lc_hostid(
    LM_HANDLE_PTR job,
    int type,
    char *buf);
```

Parameters

The **lc_hostid** function uses the following parameters:

Table 2-62 • lc_hostid Parameters

Parameter	Description
job	From **lc_new_job**.
type	The requested hostid type. For details of available values see Hostid Types. For backward compatibility, other choices exist in `lmclient.h`, but are not applicable in this version of the Licensing toolkit.
buf	A pointer to a char array of length `MAX_CONFIG_LINE`. If successful, the hostid string is returned here. This string may contain multiple hostids separated by spaces.

Return

The following values are returned by **lc_hostid**:

Table 2-63 • lc_hostid Returns

Return Values	Description
0	Success.
Any other value	Error—Flexible API errno.

Error Returns

These are the error returns for the function **lc_hostid**:

Table 2-64 • lc_hostid Error Returns

Error Return	Description
LM_FUNCNOTAVAIL	Vendor keys do not support this `id_type`.
LM_LICSRVC_FAILED	License service failed to return VM attributes.
LM_NO_VMATTR_PHY_SYS	VM attributes not available on physical machine.

Discussion

Fills in `buf` with a hostid string specified by `id_type`. If `id_type` is `HOSTID_DEFAULT`, you get the default `id_type` on the system.

This function allows developers access to hostid information in string format. When multiple hostids are detected on the system, **lc_hostid** returns a space-separated list of hostids. Typically this occurs when the hostid is derived from hardware attached to the system, for example Ethernet cards or FlexNet ID dongles.

The following examples show typical output when multiple hostids are available.

- **Composite hostid including HOSTID_ETHER**—This shows the output when two Ethernet cards are attached to a system.

  `"COMPOSITE=27bd96850a29 COMPOSITE=9684d0f2a114"`

- **FlexNet ID dongles**—This shows the output when two dongles are attached to a system.

  `"FLEXID=9-6b3366b2 FLEXIDID=9-58a35210"`

If a request is made for an invalid hostid type for a platform, **lc_hostid** returns the value of the default hostid type for that platform. If a request is made for a valid type that is not currently present on the machine, such as `HOSTID_FLEXID9` and `HOSTID_FLEXID10` on Windows, **lc_hostid** returns an error. The *License Administration Guide* contains a list of currently supported platforms and their default hostid type.

Windows Ethernet Hostid

The operation of **lc_hostid** when used on Windows platforms with the HOSTID_ETHER hostid type is controlled by the LM_A_PHYSICAL_ETHERNETID attribute. The default when this attribute is not set is that all Ethernet adapters, both physical and virtual, are reported.

See Also
`machind/lmclient.h` for valid hostid type definitions
"Hostids" in *Programming Reference for License File–Based Licensing* for instructions on choosing hostids
License Administration Guide for the default hostids
Release Notes for platforms on which multiple Ethernet hostids are supported

Hostid Types

There are two groups of hostid types: those valid for a specific platform and those valid on all platforms. Table 2-65 lists the valid hostid types for a specific platform.

Obtaining any TPM, virtualization, or cloud-licensing hostid requires that the FlexNet Licensing Service be installed.

Table 2-65 • Hostid Types for specific Platforms

Operating System	Supported Hostid Types (id_type)	Description
AIX	HOSTID_LONG	32-bit hostid
HP-UX	HOSTID_ID_STRING	String ID, maximum length is `MAX HOSTID_LEN`
Mac OS	HOSTID_ETHER	Ethernet address
	HOSTID_FLEXID9 HOSTID_FLEXID10	FlexNet ID dongles: `FLEXID=9`, `FLEXID=10`. See the *Driver Installation Guide for FlexNet ID Dongles* for manufacturer and device information.
Microsoft Windows	AMZN_AMI	Amazon template ID for Amazon EC2 environments
	AMZN_EIP	Elastic IP address for Amazon EC2 environments
	HOSTID_DISK_SERIAL_NUM	Windows volume serial number
	HOSTID_ETHER	Ethernet address
	HOSTID_FLEXID9 HOSTID_FLEXID10	FlexNet ID dongles: `FLEXID=9`, `FLEXID=10`, `PHY_FLEXID=9`, `PHY_FLEXID=10` See the *Driver Installation Guide for FlexNet ID Dongles* for manufacturer and device information.
	HOSTID_INTERNET	Internet v4 or v6 address, Amazon Instance ID, and Amazon template ID in Amazon EC2 environments

Table 2-65 • Hostid Types for specific Platforms

Operating System	Supported Hostid Types (id_type)	Description
	HOSTID_TPM_ID1	TPM (Trusted Platform Module) version 2.0 (version 1.2 is not supported) ***Note*** • *Obtaining this hostid requires that the FlexNet Licensing Service be installed.*
	PHY_*	Enforced physical machine
	VM_GENID	Generation ID of the virtual machine ***Note*** • *This identifier is available only in Windows VMs.* GenerationID cannot be used together with the HOSTID keyword. Refer to the *Understanding Virtualization Features in FlexNet Publisher* White Paper for examples of how to use GenerationID.
	VM_UUID	UUID support on virtual platforms
Linux	AMZN_AMI	Amazon template ID for Amazon EC2 environments
	AMZN_EIP	Elastic IP address for Amazon EC2 environments
	HOSTID_ETHER	Ethernet address
	HOSTID_INTERNET	Internet v4 or v6 address, Amazon Instance ID, and Amazon template ID in Amazon EC2 environments
	HOSTID_FLEXID9 HOSTID_FLEXID10	FlexNet ID dongles: `FLEXID=9`, `FLEXID=10`, `PHY_FLEXID=9`, `PHY_FLEXID=10` See *Driver Installation Guide for FlexNet ID Dongles* for manufacturer and device information.
	PHY_*	Enforced physical machine
	VM_UUID	UUID support on virtual platforms
Solaris	HOSTID_LONG	32-bit hostid
	HOSTID_ETHER	Ethernet address

The following table lists the hostid types common to all platforms.

Table 2-66 • Hostid Types Supported on All Platforms

id_type	Description
HOSTID_COMPOSITE	Composite hostid, if defined. This may return a space separated list of hostids. For more information see Multiple Composite Hostids for HOSTID_ETHER in *Programming Reference for License File–Based Licensing*. See also lc_init_simple_composite.
HOSTID_DEFAULT	Default hostid type on the system.
HOSTID_DISPLAY	Display name.
HOSTID_HOSTNAME	Node name.
FNP_HOSTID_INTERNET_V4	Returns IPv4 address only.
FNP_HOSTID_INTERNET_V6	Returns IPv6 address only.
HOSTID_INTERNET	Internet IP address. Fetches IPv4 or IPv6 address based on the environment variable `FNP_IP_Priority`. ***Note*** *• On Windows and Linux, HOSTID_INTERNET also fetches the Amazon Instance ID and Amazon template ID when in an Amazon EC2 environment.*
HOSTID_USER	User name. User names that contain spaces are not supported in license files or in the options file.
HOSTID_VENDOR	Vendor-defined hostid, if defined. For more information see Vendor-Defined Hostid Types in *Programming Reference for License File–Based Licensing*.

lc_idle

Informs the license server when the FlexEnabled application is idle.

Syntax

```
void lc_idle(
    LM_HANDLE *job,
    int flag);
```

Parameters

The **lc_idle** function uses the following parameters:

Table 2-67 • lc_idle Parameters

Parameter	Description
job	From **lc_new_job**.
flag	0 if process is not idle, non-zero if process is idle.

Return

There are no values returned by **lc_idle**.

Discussion

Informs the license server when the process is idle. **lc_idle** enables the end-user feature inactivity `TIMEOUT` to allow idle licenses to be reclaimed. Use of **lc_idle** is recommended for end users to take advantage of the `TIMEOUT` option. **lc_idle** also affects vendor daemon timeout due to `LM_A_TCP_TIMEOUT`.

lc_idle can be used to bracket a portion of the application code that prompts for user input, so that when the user is not using the application, the vendor daemon can detect the fact that the application is idle. **lc_idle** only sets a flag internally in the application; it is therefore safe to call as often as necessary. When lc_idle is enabled, the client application stops sending automatic heartbeat to the license server. Note that lc_idle is applicable to automatic heartbeats only.

Example

A typical use would be:

```
lc_idle(job, 1); /* Process is idle now */
... wait for input from user...
lc_idle(job, 0); /* Process is no longer idle */
```

See Also
lc_heartbeat
LM_A_TCP_TIMEOUT
"ls_minimum_user_timeout" in "Basic Vendor Variables" in *Programming Reference for License File–Based Licensing*

lc_init

Creates a job handle for use by **lc_cryptstr**.

Syntax

```
int lc_init(
    LM_HANDLE *oldjob,
    char *vendor_id,
    VENDORCODE *vendor_key,
    LM_HANDLE **job_id);
```

Parameters

The **lc_init** function uses the following parameters:

Table 2-68 • lc_init Parameters

Parameter	Description
job	Must be NULL on first call to **lc_init**. Otherwise, the current `job` on all subsequent calls.
vendor_id	Vendor name as defined with `VENDOR_NAME` in `machind/lm_code.h`.
vendor_key	From **LM_CODE** macro.
job_id	Set to job for the current process. This is used as the first argument to subsequent **lc_cryptstr** call.

Return

The following values are returned by **lc_init**:

Table 2-69 • lc_init Returns

Return Values	Description
0	Success
Any other value	Value of **lc_get_errno** after initialization is complete.

Error Returns

These are the error returns for the function **lc_init**:

Table 2-70 • lc_init Error Returns

Error Return	Description
LM_BAD_TZ	Time zone offset from GMT is greater than 24 hours, which may imply that a user is attempting to bypass an expiration date.
LM_BADPLATFORM	Vendor keys do not support this platform.
LM_BADKEYDATA	Bad vendor keys.
LM_BADVENDORDATA	Unknown vendor key type.
LM_CANTMALLOC	The call to **malloc** failed.
LM_DEFAULT_SEEDS	Encryption seeds contain default values, but the vendor daemon name is not `demo`.
LM_EXPIRED_KEYS	Vendor keys have expired.

Table 2-70 • lc_init Error Returns

Error Return	Description
LM_NOKEYDATA	Vendor key data not supplied.
LM_LIBRARYMISMATCH	`lmclient.h/liblmgr.a` version mismatch.
LM_NONETWORK	Networking software not available on this machine.
LM_OLDVENDORDATA	Old vendor keys supplied.

Discussion

lc_init initializes FlexNet Licensing and creates a license job. Subsequent calls to **lc_init** create new license jobs; each job is independent. **lc_init** is only to be used with license generators and not in FlexEnabled applications shipped to end users. FlexEnabled applications use **lc_new_job,** instead, for initialization and job creation because it offers enhanced security.

See Also
lc_cryptstr
lc_new_job

lc_log

Sends a message to the license server, which the license server places in the debug log.

Syntax

```
void lc_log(
    LM_HANDLE_PTR job,
    LM_CHAR_PTR msg);
```

Parameters

The **lc_log**function uses the following parameters:

Table 2-71 • lc_log Parameters

Parameter	Description
job	From **lc_new_job**.
msg	The message to be logged. The maximum length of `msg` is `LM_LOG_MAX_LEN`.

Return

There are no values returned by **lc_log**.

Error Returns

The following errors are set in the job handle used when calling **lc_log**:

Table 2-72 • lc_log Error Returns

Error Returns	Description
LM_NOSOCKET	Communications failure to daemon.
LM_CANTWRITE	Write error sending message to daemon.

lc_new_job

Initializes FlexNet Licensing and creates a new job.

Syntax

```
int lc_new_job(
    LM_HANDLE *oldjob,
    lc_new_job_cb l_new_job,
    VENDORCODE *vcode,
    LM_HANDLE **newjobp);
```

Parameters

The **lc_new_job** function uses the following parameters:

Table 2-73 • lc_new_job Parameters

Parameter	Description
oldjob	Must be NULL on first call to **lc_new_job**. On subsequent calls, use any existing job previously initialized with **lc_new_job**.
l_new_job	When linking your application with a DLL model on Windows platforms, this parameter must be NULL. Otherwise, for Windows static models and all UNIX applications, set it to the variable name, *lc_new_job_arg2*.
vcode	Pointer to VENDORCODE struct. Initialized by this function and used later as argument to **lc_checkout**.
newjobp	Set to job for the current process. This is used as the first argument to all subsequent **lc_xxx** functions.

Return

The following values are returned by **lc_new_job**:

Table 2-74 • lc_new_job Returns

Return	Description
0	Success.
Any other value	Value of **lc_get_errno** after initialization is complete.

Error Returns

These are the error returns for the function **lc_new_job**:

Table 2-75 • lc_new_job Error Returns

Error Return	Description
LM_BAD_TZ	Time zone offset from GMT is greater than 24 hours, which may imply that a user is attempting to bypass an expiration date.
LM_BADPLATFORM	Vendor keys do not support this platform.
LM_BADKEYDATA	Bad vendor keys.
LM_BADVENDORDATA	Unknown vendor key type.
LM_CANTMALLOC	The call to **malloc** failed.
LM_DEFAULT_SEEDS	Encryption seeds contain default values, but the vendor daemon name is not `demo`.
LM_EXPIRED_KEYS	Vendor keys have expired.
LM_NOKEYDATA	Vendor key data not supplied.
LM_LIBRARYMISMATCH	`lmclient.h/liblmgr.a` version mismatch.
LM_NONETWORK	Networking software not available on this machine.
LM_OLDVENDORDATA	Old vendor keys supplied.

Discussion

This function enables applications to support more than one FlexNet Licensing job in a single binary. Each job has a separate connection to a license server, as well as a independent set of job attributes. When a new job is created with **lc_new_job**, all the Licensing attributes are set to defaults, and attributes can be set completely independently for this new job. **lc_new_job** is used with FlexEnabled applications. For license generators (like `lmcrypt`) use **lc_init** instead.

All applications must link `lm_new.o` (`lm_new.obj` on Windows) into the application executable. Failing to link `lm_new.o` into the executable will result in unresolved references to `l_n36_buf`.

Note • *The application must call* **lc_new_job** *before calling any other Flexible API function, including* **lc_set_attr** *and* **lc_get_attr**.

Multiple Jobs

Multiple jobs may be desirable for the following reasons:

- If `LM_LICENSE_FILE` is a license search path (colon separated on UNIX and semicolon separated on Windows) with more than one license server supporting features for the FlexEnabled application, and if the application needs to check out more than one feature, it may be necessary to communicate with two license servers to check out the necessary licenses. This can be done only with multiple jobs because a separate connection is required for each server.
- It may be convenient to have a single process manage licenses for other processes. It is usually convenient to manage each process's license as a separate job.
- In multiple license pools, checkout upgrade within a job at sometimes doesn't get its requested number of licenses from vendor daemon despite the license availability in server and in such situations, multiple jobs are required. In these cases, partial checkout feature (aka COAVAIL) comes handy and its worth exploring.
    - Single job handle can perform checkouts from only one license server (one server connection per job handle). Therefore, if checkouts are expected to span across multiple license servers in the same client, it would require more than one job handle.
    - **lc_checkin** signature is (job, feature, disconnect-flag) where the partial feature-checkin cannot be done. But, the checkins that are grouped with vendor checkout data could be returned based on the checkout data that's used at the time of checkout. If checkout data is not used, the partial feature return need a separate job handle.

The first item can be important as an alternative way of supporting license server redundancy. What follows is a program excerpt that illustrates how to support this:

```
LM_HANDLE *job1 = 0, *job2 = 0;
VENDORCODE code;
if (lc_new_job((LM_HANDLE *)0, lc_new_job_arg2, &code, &job1))
       /* error processing */ ;
set_all_my_attr(job1); /* do all necessary lc_set_attr() calls */
if (lc_checkout(job1, "f1", "1.0", 1, LM_CO_NOWAIT, &code, LM_DUP_NONE))
       /* error processing */ ;
/* We checkout out one feature successfully, so we're
 * connected to a server already. In order to connect to
 * another server, we would need another job */
if (lc_checkout(job1, "f2", "1.0", 1, LM_CO_NOWAIT, &code, LM_DUP_NONE))
{
       if (lc_new_job(job1, lc_new_job_arg2, &code, &job2))
       {
           /* error processing */
           job2 = 0;
       }
       else
       {
           set_all_my_attr(job2); /* Reset attributes */
           if (lc_checkout(job2, "f2", "1.0", 1,
                LM_CO_NOWAIT, &code, LM_DUP_NONE))
           {
```

```
                /* error processing */ ;
            }
        }
}
/* application code here */
lc_checkin(job1, LM_CI_ALL_FEATURES, 0);
lc_free_job(job1);
if (job2 && job2 != job1)
{
        lc_checkin(job2, LM_CI_ALL_FEATURES, 0);
        lc_free_job(job2);
}
```

If the application is managing many jobs, you may want to free jobs with **lc_free_job** to save memory.

Jobs can be found and managed using **lc_first_job** and **lc_next_job**, which are used to walk the list of jobs. Attributes for jobs are set and retrieved with **lc_set_attr** and **lc_get_attr**.

See Also
lc_free_job
lc_get_attr
lc_new_job
lc_first_job
lc_set_attr
lc_init

lc_next_job

Walks the list of jobs. This works properly only if all calls to **lc_new_job** have a pointer to the current job as the first parameter.

Syntax

```
LM_HANDLE *lc_next_job(LM_HANDLE *job);
```

Parameters

The **lc_next_job** function uses the following parameters:

Table 2-76 • lc_next_job Parameters

Parameter	Description
job	Current job.

Return

The following values are returned by **lc_next_job**:

Table 2-77 • lc_next_job Returns

Return	Description
job	Next currently active job, or `(LM_HANDLE *)0` if end.

Error Returns

There are no error returns associated with **lc_next_job**.

See Also
lc_first_job
lc_free_job
lc_new_job

lc_perror

Prints a Flexible API error message.

Syntax

```
void lc_perror(
    LM_HANDLE *job,
    char *string);
```

Parameters

The **lc_perror** function uses the following parameters:

Table 2-78 • lc_next_job Parameters

Parameter	Description
job	From **lc_new_job**.
string	The first part of the error message.

Return

None.

Discussion

Prints a Flexible API error message, in the same format as the UNIX function **perror**, for example:

```
"string": net Licensing error-string
```

If a system error has also occurred, it will be included in the message.

On Windows systems, a dialog of type MB_OK will be displayed with the Flexible API error message. The Flexible API error messages are available by calling **lc_errstring**.

See Also
lc_err_info
lc_get_errno
lc_errstring
LM_A_LONG_ERRMSG

lc_set_attr

Sets a Flexible API attribute.

Syntax

```
int lc_set_attr(
    LM_HANDLE *job,
    INT key,
    LM_A_VAL_TYPE value);
```

Parameters

The **lc_set_attr** function uses the following parameters:

Table 2-79 • lc_set_attr Parameters

Parameter	Description
job	From **lc_new_job**.
key	Which attribute to set.
value	Value to set it to. Values should be of the appropriate type for the particular attribute, but should be cast to LM_A_VAL_TYPE.

Return

The following values are returned by **lc_set_attr**:

Table 2-80 • lc_set_attr Returns

Return	Description
0	Success.
Any other value	Error.

Error Returns

These are the error returns for the function **lc_set_attr**:

Table 2-81 • lc_set_attr Error Returns

Error Return	Description
LM_FUNCNOTAVAIL	Vendor keys do not support this function.
LM_BADPARAM	Specified parameter is incorrect.
LM_NOCONFFILE	Specified license file cannot be found (`LM_A_LICENSE_FILE` or `LM_A_LICENSE_FILE_PTR`).
LM_NOSUCHATTR	Specified attribute does not exist.
LM_NOT_ALLOWED	Specified operation is not allowed. The requested function is not allowed. For example calling lc_set_attr after a feature has already been checked out from the license server.

Discussion

Flexible API attributes provide you with control over licensing policy, internal operations of FlexNet Licensing (for example, automatic heartbeat mechanism, and so on), and control of the licensing parameters of your process (defining how FlexNet Licensing will define user name, host name, and display name, and so on for managed license distribution).

For details of attributes, including their type, see:

- Flexible API Attributes
- Advanced Flexible API Attributes
- Obsolete Flexible API Attributes

This function sets the specified attribute.

When your license server contains a Common Vendor Daemon, this function sets the specified attribute for all vendors within the Common Vendor Daemon. To set an attribute for a single vendor within a Common Vendor Daemon use `lc_set_attr_vendor`.

Note • *`lc_set_attr` should not be used in a loop. Attribute must be set only once before license checkout.*

Automatic Update of License File Location

After a successful checkout, the location of the license files used for the checkout will be saved in the registry on Windows or `$HOME/.flexlmrc` on UNIX. This way, all subsequent checkouts for features from this vendor will automatically use the license that worked previously. Note that this location is added to all other locations the application may look for the license. The name of the variable used to save the license file location is *<vendor>*`_LICENSE_FILE` where *<vendor>* is the name of the vendor daemon.

Note • *After successful checkout the location of the license file is saved in the registry. This is a known limitation of setting LM_LICENSE_FILE environment variable for Trusted Storage.*

This automatic update can be turned off with:

```
lc_set_attr(job, LM_A_CKOUT_INSTALL_LIC, (LM_A_VAL_TYPE)0);
```

lc_status

Returns the status of the requested feature.

Syntax

```
int lc_status(
    LM_HANDLE *job,
    char *feature);
```

Parameters

The **lc_status** function uses the following parameters:

Table 2-82 • lc_status Parameters

Parameter	Description
job	From **lc_new_job**.
feature	The feature name.

Return

The following values are returned by **lc_status**:

Table 2-83 • lc_status Returns

Return	Description
0	Feature is checked out by this process.
<0	Error.

Error Returns

These are the error returns for the function **lc_status**:

Table 2-84 • lc_status Error Returns

Error Return	Description
LM_CANTCONNECT	Feature was checked out, but lost connection to the daemon.

Table 2-84 • lc_status Error Returns

Error Return	Description
LM_FEATQUEUE	This process is in the queue for this feature.
LM_NEVERCHECKOUT	Feature was never checked out by this process, or was checked back in after a checkout.

Discussion

A call to this function is made when QUEUEing for a license. Normally QUEUEing is done in the following manner:

```
status = lc_checkout(....LM_CO_NOWAIT,...);
if (status == LM_MAXUSERS || status == LM_USERSQUEUED)
{
    printf("Waiting for license...");
    status = lc_checkout(....LM_CO_WAIT,...);
}
```

However, in the above example, the application must pend on the call to **lc_checkout**. If the application needs to continue doing processing, use `LM_CO_QUEUE` in the call to **lc_checkout**. Call **lc_status** immediately after the call to **lc_checkout** and any other **lc_xxx** function until the license is granted or denied. This might be coded in the following manner:

```
status = lc_checkout(...,LM_CO_QUEUE,...)
switch (status)
{
       case 0:
               break; /* got the license */
       case LM_MAXUSERS:
       case LM_USERSQUEUED:
       case LM_FEATQUEUE:
               printf("Waiting for license...");
               while (lc_status(job, feature))
               {
                      /* processing */
               }
               break; /* got the license */
       default:
               lc_perror(job, "Checkout for license failed");
}
```

See Also
lc_checkout

lc_tpmstatusget (Windows only)

Gets the Trusted Platform Module (TPM) status of the system.

Operates for both physical and virtual TPM 2.0. TPM 1.2 is not supported.

Syntax

```
int lc_tpmstatusget(
    LM_HANDLE_PTR job);
```

Parameters

The **lc_tpmstatusget** function uses the following parameter:

Table 2-85 • lc_tpmstatusget Parameter

Parameter	Description
job	From **lc_new_job**.

Return

The following values are returned:

Table 2-86 • lc_tpmstatusget Returns

Return	Description
LM_NOERROR	Success. TPM 2.0 has been detected.
Any other value	The TPM status could not be obtained. See Error Returns for possible errors.

Error Returns

These are the error returns for the function **lc_tpmstatusget**:

Table 2-87 • lc_tpmstatusget Error Returns

Error Return	Description
LM_BADPARAM	Invalid parameter.
LM_LICSRVC_FAILED	The FlexNet Licensing Service failed to return VM attributes.
LM_LIC_SVC_COMMS_ERROR	Error in communication with the FlexNet Licensing Service.
LM_LIC_SVC_NOTINSTALLED	The FlexNet Licensing Service is not installed.
LM_LIC_SVC_REINSTALL_SVC	The FlexNet Licensing Service should be reinstalled.
LM_LIC_SVC_VER_ERR	The FlexNet Licensing Service version is not as expected.
LM_TPM_DISABLED	TPM 2.0 is available, but disabled. To use TPM, it must be enabled in the machine's BIOS (Security settings).
LM_TPM_LICSRVC_FAILED	The FlexNet Licensing Service failed to return TPM attributes.

Table 2-87 • lc_tpmstatusget Error Returns

Error Return	Description
LM_TPM_PROPS_NOT_AVAILABLE	TPM properties are not available.
LM_TPM_VERSION_UNSUPPORTED	The TPM version is not supported.

See Also
`examples\hostid\tpmutil.c`

lc_userlist

Provides a list of who is using a feature, including information about the users of the license. This output is used by `lmstat`. See the *License Administration Guide* for the behavior of `lmstat`.

Important • *The call to* **lc_userlist** *is potentially expensive (it may cause a lot of network traffic), depending on the number of users of `feature`. Therefore this function must be used with caution. In particular, it is a good idea to call* **lc_userlist** *when a checkout fails with `LM_MAXUSERS`/`LM_USERSQUEUED` error, to inform who is using the feature. However, do not call* **lc_userlist** *before every call to* **lc_checkout** *because this will be guaranteed to cause network load problems when a large number of licenses are checked out.*

Syntax

```
LM_USERS *lc_userlist(
    LM_HANDLE *job,
    char *feature);
```

Parameters

The **lc_userlist** function uses the following parameters:

Table 2-88 • lc_userlist Parameters

Parameter	Description
job	From **lc_new_job**.
feature	The feature name.

Return

If successful, **lc_userlist** returns a pointer to a linked list of structures, one for each user of the license. This data should not be modified by the caller. It will be freed on the next call to **lc_userlist**.

See `lmclient.h` for a description of the `LM_USERS` struct.

The list of users returned by **lc_userlist** includes a special record, indicated by an empty user name (`name[0]==0`), and consists of one empty user list element for the number of feature, INCREMENT/UPGRADE lines in a license file.

If there is an error, **lc_userlist** returns NULL and sets the job error status.

lc_userlist returns only information about users the server knows about; therefore, it will not return any information about users of node-locked uncounted or DEMO licenses, unless the server's license file includes the node-locked licenses and the FlexEnabled application is not reading the license file (via `@host`, `port@host` or `USE_SERVER`). Licenses shared due to duplicate grouping are also not returned by **lc_userlist**.

Reserved licenses are indicated by the **lm_isres** macro (defined in `lmclient.h`). In this case, the `name` contains the entity that the reservation is for.

Error Returns

These are the error returns for the function **lc_userlist**:

Table 2-89 • lc_userlist Error Returns

Error Return	Description
LM_BADCOMM	Communications error with license server.
LM_CANTMALLOC	The call to **malloc** failed.
LM_FUNCNOTAVAIL	Vendor keys do not support this function.
LM_NOFEATURE	Specified feature cannot be found.

See Also
`machind/lmclient.h` for `LM_USERS` structure definition.

lc_virtualstatusget

Provides information on the virtual status of the system. This is determined by the return value.

Syntax

```
int lc_virtualstatusget(LM_HANDLE_PTR job);
```

Parameters

The **lc_virtualstatusget()** API uses the following parameters

Table 2-90 • lc_virtualstatusget() Parameters

Parameter	Description
job	From **lc_new_job**.

Return

The following values are returned by **lc_virtualstatusget**:

Table 2-91 • lc_virtualstatusget Returns

Parameter	Description
0	Machine is physical.
1	Machine is virtual.
-1	Error

lc_vsend

Sends a message to the vendor daemon and returns a result string.

Syntax

```
char *lc_vsend(
    LM_HANDLE *job,
    char *send_str);
```

Parameters

The **lc_vsend** function uses the following parameters:

Table 2-92 • lc_vsend Parameters

Parameter	Description
job	From **lc_new_job**.
send_str	Contains the message—up to 140 bytes in length— to be sent to your vendor daemon.

Return

The following values are returned by **lc_vsend**:

Table 2-93 • lc_vsend Returns

Return Values	Description
char*	String returned by **ls_vendor_msg** in your vendor daemon; 0 if unsuccessful.

Error Returns

These are the error returns for the function **lc_vsend**:

Table 2-94 • lc_vsend Error Returns

Error Return	Description
LM_BADCOMM	Communications problem with the vendor daemon.
LM_CANTREAD	Cannot read data from license server.
LM_NOSERVSUPP	Your vendor daemon does not support this function. You customize your vendor daemon when using lc_vsend.

When LM_A_VSEND_NOWAIT is set, the client must poll the vendor daemon by calling lc_vsend with the message parameter set to the macro LM_VSEND_STATUS. Example usage can be found in examples/lc_vsend/lmvsend.c.

When LM_A_VSEND_NOWAIT is set and lc_vsend is called with a license client message string, then if the return value is non-NULL and the message has been accepted.

If the return value is NULL, one of the following error status values will be set in the job:

Table 2-95 • lc_vsend Error Returns

Error Return	Description
LM_VSEND_ASYNC_NOT_AVAILABLE	The vendor daemon is unable to process vsend messages asynchronously.
LM_VSEND_ASYNC_BUSY	The vendor daemon already has a message for this job, which has not yet been completed.
LM_VSEND_TOO_MANY	The vendor daemon is currently too busy to accept any more asynchronous messages. (See also Vendor Variable ls_vendor_msg_list_size).
LM_VSEND_INVALID_MESSSAGE	The supplied message length exceeds LM_VSEND_MAX_LEN.

When LM_A_VSEND_NOWAIT is set and lc_vsend is called with the license client message set to the macro LM_VSEND_STATUS, then if the return value is non-NULL, it points to the response string from the vendor daemon.

If the return value is NULL, one of the following error status values will be set in the job:

Table 2-96 • lc_vsend Error Returns

Error Return	Description
LM_VSEND_ASYNC_NOT_AVAILABLE	The vendor daemon is unable to process vsend messages asynchronously.

Table 2-96 • lc_vsend Error Returns

Error Return	Description
LM_VSEND_ASYNC_BUSY	The Vendor Daemon is still processing the message for this job.
LM_VSEND_NO_MESSSAGE	The Vendor Daemon does not have any outstanding message for this job.

Note • *If a job gets closed, any pending asynchronous message in the Vendor Daemon will be lost.*

Discussion

Sends a message contained in `send_str`, which can be up to 140 bytes in length, to the vendor daemon and returns a result string in `rcv_str`. (The result string can also be up to 140 bytes in length.) If the FlexEnabled application is not already connected to a server, this function tries each server in a list of servers until it is able to connect.

You must set up a processing routine in your vendor daemon to receive the message from **lc_vsend** and send the reply. This routine is specified in `lsvendor.c` with the variable *`ls_vendor_msg`*.

Important • *Using lc_vsend with a local license causes unpredictable behavior. lc_vsend must only be used for checkouts that use licenses held on a license server.*

See Also
ls_vendor_msg in “Basic Vendor Variables” in *Programming Reference for License File–Based Licensing*

3

Advanced Flexible API Functions

This chapter lists functions that provide advanced Flexible API functionality. Basic functionality is covered by the material in Flexible API.

Summary

The following Flexible API functions are advanced and used for functionality not found in the basic set of functions. They should be used with care, and questions are welcomed before their use. Table 3-1 summarizes these functions.

Table 3-1 • Advanced Flexible API Functions

Function	Description
l_new_hostid	Allocates a hostid structure.
lc_borrow_return	Returns a borrowed license early.
lc_check_key	Validates a license signature.
lc_cleanup (Windows only)	Cleans up Flexible API resources after they are no longer needed.
lc_diagnostics_end	Ends a section of code in a lmflex client to identify as a session.
lc_diagnostics_start	Starts a section of code in a lmflex client to identify as a session.
lc_free_hostid	Frees the memory associated with a hostid structure.
lc_get_config	Returns license file data for a given feature.
lc_init_simple_composite	Initializes a composite hostid.
lc_next_conf	Returns the next line in the license file matching the given feature.

Table 3-1 • Advanced Flexible API Functions

Function	Description
lc_remove	Removes a given feature for the specified user.
lc_set_attr_vendor	Sets an attribute for a specific vendor when a Common Vendor Daemon is being used.
lc_shutdown	Shuts down the FlexNet license servers.
lc_test_conf	Returns license file data for the most recently tested feature.

l_new_hostid

Allocates a hostid structure. Returns a malloc'd and zeroed hostid. Use **lc_free_hostid** to free this memory. This may be needed when doing vendor-defined hostids.

Syntax

```
HOSTID_PTR l_new_hostid(void);
```

Parameters

The **l_new_hostid** function uses no parameters.

Return

The following values are returned by **l_new_hostid**:

Table 3-2 • l_new_hostid Returns

Return	Description
hostid	A `HOSTID` struct, or null.

Error Returns

These are the error returns for **l_new_hostid**:

Table 3-3 • l_new_hostid Error Returns

Error Return	Description
LM_CANTMALLOC	**malloc** call failed.

See Also

LM_A_VENDOR_ID_DECLARE

Information about Vendor-Defined Hostid Types in *Programming Reference for License File–Based Licensing*

lc_borrow_return

Returns a borrowed license early.

Syntax

```
int lc_borrow_return(
    LM_HANDLE *job,
    char *feature,
    char *display);
```

Parameters

The **lc_borrow_return** function uses the following parameters:

Table 3-4 • lc_borrow_return Parameters

Parameter	Description
job	From **lc_new_job**.
feature	The name of the borrowed feature to be returned early.
display	The display from which the borrow was initiated. Required if the current display is different than what was used to initiate the borrow. On Windows, it is the system name or, in the case of a terminal server environment, the terminal server client name. On UNIX, it is in the form `/dev/ttyxx` or the X-Display name. If NULL, Flexible API uses the current display name, determined via the standard method appropriate to the platform.

Return

The following values are returned by **lc_borrow_return**:

Table 3-5 • lc_borrow_return Returns

Return Values	Description
0	Success
Any other value	The Flexible API error code

Error Returns

These are the error returns for function **lc_borrow_return:**

Table 3-6 • lc_borrow_return Error Returns

Error Return	Description
LM_BADPARAM	Problem with *job* or *feature* argument.
LM_BORROW_ERROR	Cannot read the local borrow info.
LM_BORROW_RETURN_SERVER_ERR	The server generated an error during a license return attempt.
LM_CANTCONNECT	Cannot connect to license server to return the license.
LM_CANTWRITE	Cannot send data to license server
LM_NOFEATURE	The feature specified doesn't match any borrowed items on the FlexEnabled application's side.

Description

lc_borrow_return is used to return a borrowed license to the pool of available licenses before the borrow period expires. The application that calls this routine must be running on the same machine from which the license was originally borrowed.

In order for this routine to have an effect, the license server must be configured to allow early return of borrowed licenses using the `ls_borrow_return_early` vendor daemon variable.

See Also
Information about ls_borrow_return_early in the *Programming Reference for License File–Based Licensing*

lc_check_key

Validates a license signature.

Syntax

```
int lc_check_key(
    LM_HANDLE *job,
    CONFIG *conf,
    const VENDORCODE_PTR key);
```

Parameters

The **lc_check_key** function uses the following parameters:

Table 3-7 • lc_check_key Parameters

Parameter	Description
job	From **lc_new_job**.
conf	From **lc_next_conf, lc_get_config**.
key	From **lc_new_job**.

Return

The following values are returned by **lc_check_key**:

Table 3-8 • lc_check_key Returns

Return Values	Description
0	Success
Any other value	The Flexible API error code

Error Returns

These are the error returns for function **lc_check_key**:

Table 3-9 • lc_check_key Error Returns

Error Return	Description
LM_BADCODE	Signature is invalid—license has been typed incorrectly, or altered in some way.
LM_BADPARAM	Problem with `conf` argument.
LM_FUTURE_FILE	License format is invalid and may be from a future FlexNet Publisher Licensing Toolkit version.

Description

lc_check_key determines if the signature in the `CONFIG` structure, pointed to by `conf`, is valid. This function is optional and only for use during license installation; it is not used in an application that also calls **lc_checkout**. If authenticated license information is needed without checking out a license, use **lc_checkout** with the `LM_CO_LOCALTEST` flag.

To verify a license file upon installation, use code similar to the following example:

```
VENDORCODE code;
lc_new_job(..., &code, ...);
```

```
feats = lc_feat_list(...);
while (*feats)
{
    pos = 0;
    while (conf = lc_next_conf(job, *feats, &pos))
    {
         if (lc_check_key(job, conf, &code))
            /*error*/
    }
    feats++;
}
```

See Also
examples\advanced\exinstal.c
lc_next_conf
lc_feat_list
lc_checkout

lc_cleanup (Windows only)

Cleans up Flexible API resources after they are no longer needed.

Syntax

```
void lc_cleanup();
```

Parameters

The **lc_cleanup** function uses no parameters.

Return

No values are returned by **lc_cleanup**.

Description

This routine is only required on Windows platforms under the following conditions:

- Your FlexEnabled application dynamically links to the FlexNet Publisher Licensing Toolkit client library

  Call **lc_cleanup** before the DLL is unmapped from the application's address space.

- You distribute your product as a DLL

  To ensure all Flexible API resources are properly cleaned up after your end user's application unmaps your FlexEnabled DLL, make sure you call **lc_cleanup** in the logical clean up point in your DLL.

This function frees all allocated memory, releases all handles, and kills all threads that are created by FlexNet Publisher Licensing Toolkit library routines. **lc_cleanup** must be the last client library routine that is called by your FlexEnabled application. Do not call any other client library routines after calling this routine.

lc_diagnostics_end

Ends a section of code in a lmflex client to identify as a session.

Syntax

```
void lc_diagnostics_end();
```

Parameters

The **lc_diagnostics_end** function uses no parameters.

Return

No values are returned by **lc_diagnostics_end**.

Description

A single row of diagnostics data corresponds to one message, which is sent to the license server. However, many client API calls, such as lc_checkout, comprise several messages. In order to be able to trace all the messages from a particular `lc_*` call instance, each diagnostics row has a SessionID. By default, a session exists for the duration of any one lc_* call. However, it can be defined as encompassing multiple `lc_*` calls. The `lc_diagnostics_end` marks an end of a section of code in a lmflex client to identify as a session.

lc_diagnostics_start

Starts a section of code in a lmflex client to identify as a session.

Syntax

```
void lc_diagnostics_start(unsigned int value);
```

Parameters

The **lc_diagnostics_start** function uses no parameters.

Return

No values are returned by **lc_diagnostics_start**.

Description

A single row of diagnostics data corresponds to one message, which is sent to the license server. However, many client API calls, such as lc_checkout, comprise several messages. In order to be able to trace all the messages from a particular `lc_*` call instance, each diagnostics row has a SessionID. By default, a session exists for the duration of any one `lc_*` call. However, it can be defined as encompassing multiple lc_* calls. The `lc_diagnostics_start` marks the start of a section of code in a lmflex client to identify as a session.

lc_free_hostid

Frees the memory associated with a hostid structure.

Syntax

```
void lc_free_hostid(
    LM_HANDLE *job,
    HOSTID *id);
```

Parameters

The **lc_free_hostid** function uses the following parameters:

Table 3-10 • lc_free_hostid Parameters

Parameter	Description
job	From **lc_new_job**.
id	From **l_new_hostid**.

Return

No values are returned by **lc_free_hostid**.

Error Returns

These are the error returns associated with **lc_free_hostid**:

Table 3-11 • lc_free_hostid Error Returns

Error Return	Description
LM_BADPARAM	No such job.

Description

lc_free_hostid frees the memory associated with a hostid that has been allocated with **l_new_hostid**. If passed a hostid list, **lc_free_hostid** frees the whole list.

This function does not use or reference the job parameter.

It can also be used with the obsolete function **lc_copy_hostid**.

See Also
l_new_hostid

lc_get_config

Returns license file data for a given feature.

Syntax

```
CONFIG *lc_get_config(
    LM_HANDLE *job,
    char *feature);
```

Parameters

The **lc_get_config** function uses the following parameters:

Table 3-12 • lc_get_config Parameters

Parameter	Description
job	From **lc_new_job**.
feature	The desired feature.

Return

The following values are returned by **lc_get_config**:

Table 3-13 • lc_get_config Returns

Return Values	Description
CONFIG struct	A CONFIG struct that contains the license file data for the specified feature. The `CONFIG` struct is defined in the header file `lmclient.h`.
NULL	The specified feature was not found.

Error Returns

These are the error returns for function **lc_get_config**:

Table 3-14 • lc_get_config Error Returns

Error Return	Description
LM_NOFEATURE	Specified feature does not exist.
LM_NOCONFFILE	License file does not exist.
LM_BADFILE	License file corrupted.
LM_NOREADLIC	Cannot read license file.
LM_SERVNOREADLIC	Cannot read license data from license server.

Description

Gets the license file data for a given feature. FlexNet licensing allows multiple valid `FEATURE` and `INCREMENT` lines (of the same feature name) in a license file. **lc_get_config** will return the first `CONFIG` struct, and **lc_next_conf** retrieves the next (**lc_next_conf** can also find the first). **lc_get_config** does not authenticate feature lines. That is, a user can type in a `FEATURE` line with an invalid signature, and **lc_get_config** will still return it. **lc_get_config** and **lc_next_conf** are usually needed only with `LM_A_VD_GENERIC_INFO` or `LM_A_VD_FEATURE_INFO`.

To get authenticated information from a `FEATURE` line, you must first check out the feature, and then use **lc_auth_data**.

See Also
lc_auth_data
LM_A_MULTIPLE_CHECKOUT_DATA
lc_checkout
lc_next_conf
LM_A_VD_GENERIC_INFO, LM_A_VD_FEATURE_INFO

lc_init_simple_composite

Initializes a composite hostid.

Syntax

```
int lc_init_simple_composite(
    LM_HANDLE_PTR job,
    int hostid_list[],
    int num_ids);
```

Parameters

The **lc_init_simple_composite** function uses the following parameters:

Table 3-15 • lc_init_simple_composite Parameters

Parameter	Description
job	From **lc_new_job**.
hostid_list	A pointer to an integer array of one or more simple hostid types to be included in the composite hostid. See lc_hostid for a list of valid hostid types.
num_ids	The number of hostid types specified in `hostid_list`.

Return

The following values are returned by **lc_init_simple_composite**:

Table 3-16 • lc_init_simple_composite Returns

Return Values	Description
0	Success
Any other value	The Flexible API errno.

Error Returns

These are the error returns for function **lc_init_simple_composite**:

Table 3-17 • lc_init_simple_composite Error Returns

Error Return	Description
LM_COMPOSITEID_INIT_ERR	Error initializing the composite hostid.
LM_COMPOSITEID_ITEM_ERR	An item needed for composite hostid is missing or invalid.

Description

This function initializes a vendor-defined composite hostid. Invoke this function right after the call to **lc_new_job**. The composite hostid is valid for the life of the license job and is accessed by specifying the `HOSTID_COMPOSITE` hostid type to **lc_hostid.**

A composite hostid is a 12-byte hashed hexidecimal value formed by combining the values of one or more simple hostids types, as specified by `hostid_list`. If an invalid hostid type for the platform appears in `hostid_list`, **lc_init_simple_composite** returns the value of the default hostid type for that platform.

For more information about composite hostids, see *Programming Reference for License File–Based Licensing*

Example

The following example demonstrates the definition and use of a composite hostid made up from `HOSTID_ETHER` and `HOSTID_DISPLAY`.

```
#include lmclient.h
LM_HANDLE *job
char buf[MAX_CONFIG_LINE];
/* Set up the list of hostid types which comprise the
   composite hostid, hostids must appear in the same order
   as those in hostid lists specified in other components.
*/
int hostid_list[]={HOSTID_ETHER, HOSTID_DISPLAY};
int num_ids = 2;
   /*...*/
lc_new_job(..., &job);
/* Register the composite hostid */
ret = lc_init_simple_composite(job, hostid_list, num_ids);
```

```
if (ret != 0)
{
   /* error processing */
}
/* ... */
/* Now, access the composite hostid value */
if (lc_hostid(job, HOSTID_COMPOSITE, buf))
{
   /* error processing */
}
```

See Also
lc_hostid
machind/lmclient.h

lc_next_conf

Returns the next line in the license file matching the given feature.

Syntax

```
CONFIG *lc_next_conf(
    LM_HANDLE *job,
    char *feature,
    CONFIG **pos);
```

Parameters

The **lc_next_conf** function uses the following parameters:

Table 3-18 • lc_next_conf Parameters

Parameter	Description
job	From **lc_new_job**.
feature	The desired feature line.
pos	Declare `CONFIG *pos = 0`; use &pos for argument. Internally updated by **lc_next_conf** to next license file entry.

Return

The following values are returned by **lc_next_conf**:

Table 3-19 • lc_next_conf Returns

Return	Description
CONFIG struct	A CONFIG struct that contains the license file data for the specified feature. The `CONFIG` struct is defined in the header file `lmclient.h`.

Table 3-19 • lc_next_conf Returns

Return	Description
NULL	The specified feature was not found.

Error Returns

See error returns for lc_get_config.

Description

Returns the next line in the license file, as a pointer to a `CONFIG` structure, matching `feature`. The search is started from pos, where `pos = 0` indicates the first line. **lc_next_conf** automatically updates pos, so it is ready to retrieve the next feature line on a subsequent call.

lc_next_conf does not authenticate `FEATURE` lines. That is, a user can type in a `FEATURE` line with an invalid signature, and **lc_next_conf** will still return it. **lc_get_config** and **lc_next_conf** are usually needed only with `LM_A_VD_GENERIC_INFO` or `LM_A_ VD_FEATURE_INFO`.

To get authenticated information from a `FEATURE` line, you must first check out the feature, and then use **lc_auth_data**.

Example

```
CONFIG *pos = 0, *conf;
while (conf = lc_next_conf(job, "myfeature", &pos))
{
    /* ... */
}
```

See Also
lc_auth_data
LM_A_MULTIPLE_CHECKOUT_DATA
lc_checkout
lc_get_config
LM_A_VD_GENERIC_INFO, LM_A_VD_FEATURE_INFO

lc_remove

Removes a given feature for the specified user.

Syntax

```
int lc_remove(
    LM_HANDLE *job,
    char *feature,
    char *user,
    char *host,
    char *display);
```

Parameters

The **lc_remove** function uses the following parameters:

Table 3-20 • lc_remove Parameters

Parameter	Description
job	From **lc_new_job**.
feature	Remove the license for this feature.
user	User name of license to remove.
host	Host name of license to remove.
display	Display name of license to remove.

Return

The following values are returned by **lc_remove**:

Table 3-21 • lc_remove Returns

Return Values	Description
0	Success
Any other value	The Flexible API error code

Error Returns

These are the error returns for function **lc_remove**:

Table 3-22 • lc_remove Error Returns

Error Return	Description
LM_BADCOMM	Communications error.
LM_BADPARAM	No licenses issued to this user.
LM_CANTCONNECT	Cannot connect to license server.
LM_CANTREAD	Cannot read from license server.
LM_CANTWRITE	Cannot write to license server.
LM_NOFEATURE	Feature not found in license file data.
LM_NOTLICADMIN	Failed because user is not in “lmadmin” group.

Table 3-22 • lc_remove Error Returns

Error Return	Description
LM_REMOVETOOSOON	Failed because `ls_min_lmremove` time has not elapsed.

Description

Removes the specified user's license for `feature`. This is used by the `lmremove` command and has the same restrictions regarding the "lmadmin" group. **lc_remove** normally is only used when the FlexEnabled application's system has had a hard crash and the server does not detect the FlexEnabled application's machine failure. If **lc_remove** is called on a healthy machine, the license will be checked out again by the FlexEnabled application with its next heartbeat.

The license server can be configured to disable or enable the operation of **lc_remove**:

- **lmadmin** - default configuration disables the operation of **lc_remove**. To enable **lc_remove**, start `lmadmin` from the command line with the `-allowLicenseReclaim` argument set.
- **lmgrd** - default configuration allows the operation of **lc_remove**. To disable **lc_remove**, start `lmgrd` from the command line with the `-x lmremove` argument set.

See Also
Information about the license server manager in *Programming Reference for License File–Based Licensing*
Information about ls_min_lmremove in *Programming Reference for License File–Based Licensing*
lc_heartbeat

lc_set_attr_vendor

Sets an attribute for a specific vendor when a Common Vendor Daemon is being used.

Syntax

```
int status = lc_set_attr_vendor(
    LM_HANDLE_PTR job,
    LM_CHAR_PTR vendorName,
    int attribute,
    LM_A_VAL_TYPE value);
```

Parameters

The **lc_set_attr_vendor** function uses the following parameters:

Table 3-23 • lc_set_attr_vendor Parameters

Parameter	Description
job	From **lc_new_job**.
vendorName	The secondary vendor name, i.e. the vendor name of a vendor daemon that has been combined into a Common Vendor Daemon. Specifies that licensing functionality for this vendor is to be set by this function call.

Table 3-23 • lc_set_attr_vendor Parameters

Parameter	Description
attribute	The attribute to be set.
value	The value used to set the attribute. The type must be appropriate for the particular attribute (see Chapter 4, Flexible API Attributes and Chapter 5, Advanced Flexible API Attributes), and then be cast to `LM_A_VAL_TYPE`.

Return

The following values are returned by **lc_set_attr_vendor**:

Table 3-24 • lc_set_attr_vendor Returns

Return	Description
0	Success
Any other value	Failure, see Error Returns

Error Returns

These are the error returns for function **lc_set_attr_vendor**:

Table 3-25 • lc_set_attr_vendor Error Returns

Error Return	Description
LM_FUNCNOTAVAIL	Vendor keys do not support this function.
LM_BADPARAM	Specified parameter is incorrect.
LM_NOCONFFILE	Specified license file cannot be found (`LM_A_LICENSE_FILE` or `LM_A_LICENSE_FILE_PTR`).
LM_NOSUCHATTR	Specified attribute does not exist.

Description

This function can only be used when a Common Vendor Daemon is being used. It sets the specified attribute for the specified vendor name. To set attributes for all vendors within a Common Vendor Daemon use `lc_set_attr`.

See Also
lc_set_attr
Flexible API Attributes
Advanced Flexible API Attributes
Obsolete Flexible API Attributes
Information about the common vendor daemon in *Programming Reference for License File–Based Licensing*

lc_shutdown

Shuts down the FlexNet license servers. This is used by `lmdown`.

Syntax

```
int lc_shutdown(
    LM_HANDLE *job,
    int prompt,
    int flags);
```

Parameters

The **lc_shutdown** function uses the following parameters:

Table 3-26 • lc_shutdown Parameters

Parameter	Description
job	From **lc_new_job**.
prompt	Unused.
flags	Unused.

Return

The following values are returned by **lc_shutdown**:

Table 3-27 • lc_shutdown Returns

Return	Description
0	Server shuts down.
!	Server does not shut down.

Error Returns

These are the error returns for function **lc_shutdown**:

Table 3-28 • lc_shutdown Error Returns

Error Return	Description
LM_FUNCNOTAVAIL	Vendor keys do not support this function.
LM_NOTLICADMIN	You are not an authorized license administrator.
LM_CANTREAD	Cannot read data from license server.

See Also
Information about the license server manager in *Programming Reference for License File–Based Licensing*

lc_test_conf

Returns license file data for the most recently tested feature.

Syntax

```
CONFIG *lc_test_conf(LM_HANDLE *job);
```

Parameters

The **lc_test_conf** function uses the following parameters:

Table 3-29 • lc_test_conf Parameters

Parameter	Description
job	From **lc_new_job.**

Return

The following values are returned by **lc_test_conf**:

Table 3-30 • lc_test_conf Returns

Return Values	Description
CONFIG struct	A CONFIG struct that contains the license file data for the specified feature. The `CONFIG` struct is defined in the header file `lmclient.h`.
NULL	The specified feature was not found.

Error Returns

There are no error returns for function **lc_test_conf**.

Description

Retrieves the license file line for the most recently tested feature. A feature is tested, but not checked out, when the `LM_CO_LOCALTEST` flag is set in the call to **lc_checkout**. Use **lc_test_conf** to retrieve information from a license file for tested features. For checked out features, use **lc_auth_data**.

The behavior of **lc_test_conf** is undefined if called in any other context other than after a tested feature.

Example

The following code excerpt demonstrates using this function. In this example, the application obtains information for the feature just tested in order to display various fields. For this example, consider the following feature line from the license file:

```
FEATURE f1 demo 1.0 permanent 4 NOTICE="ab" SIGN=xxxxxxxxxxxx
```

The code appears as follows in the application:

```
VENDORCODE code;
Fnp_config_type config_type;
LM_HANDLE *lm_job;
CONFIG *conf;

lc_new_job(0, lc_new_job_arg2, &code, &lm_job)

/* Now, perform the test checkout*/
if (lc_checkout(lm_job, "f1",...,...,LM_CO_LOCALTEST,...))
{
    /* error processing
}
else
{
    /* Retrieve information for feature “f1” */

    conf = lc_test_conf(lm_job);

    if (conf)
    {
        config_type = conf->nConfigType;
        if (config_type == FNP_CONFIG_TYPE_CLIENT_TS)
        {
            /* CONFIG from client trusted storage */
            ...
        }
        else if (config_type == FNP_CONFIG_TYPE_UNSERVED_CERT)
        {
            /* CONFIG from unserved license file */
            ...
        }
        else if (config_type == FNP_CONFIG_TYPE_SERVED_TS)
        {
            /* CONFIG from server trusted storage */
            ...
        }
        else if (config_type == FNP_CONFIG_TYPE_SERVED_CERT)
        {
            /* CONFIG from served license file */
            ...
        }
        ...
        /* Display information for the given feature  */
        printf("feature=%s version=%s daemon=%s, feature count = %d, total feature count=%u \n",
            conf->feature, conf->version, conf->daemon, conf->users, conf->nTotalFeatureCount);
    }
}
```

See Also

`lmclient.h` for the `CONFIG` struct definition
lc_auth_data
lc_checkout
lc_get_config
lc_next_conf

4

Flexible API Attributes

Licensing behavior is controlled from within the FlexEnabled application by setting license *policies* and *attributes*. The Flexible API allows you to control the licensing behavior of your application by setting attributes with the **lc_set_attr** function.

Flexible API attributes allow you control over licensing policy, internal operations of FlexNet licensing (for example, automatic heartbeat mechanism, and so on), and control of the licensing parameters of your process (defining how FlexNet licensing will define user name, host name, and display name, and so on for managed license distribution).

To set Flexible API attributes, call the lc_set_attr function. To query the setting of an attribute, call the lc_get_attr function.

When using these attributes with **lc_set_attr**, the argument must be of the correct type (each attribute includes details of its type) and must then be cast to `LM_A_VAL_TYPE`. Arguments of type `char` must use UTF-8 encoding for non ASCII characters, for example oxE9 for the character é. When using these attributes with **lc_get_attr**, the pointer argument should point to a value of the correct type (noting that `short` and `int` are different in this case) and must be cast to a `short *`. All attribute definitions are in `lm_attr.h`.

The essential Flexible API attribute that should be set by every FlexEnabled application is LM_A_LICENSE_DEFAULT. This attribute defines the default license file location.

The following attributes are often useful:

- Information useful for error, or informational, reporting:
  - LM_A_BORROW_STAT
  - LM_A_LF_LIST
  - LM_A_VD_GENERIC_INFO, LM_A_VD_FEATURE_INFO
  - LM_A_VM_FAMILY
  - LM_A_VM_NAME
- Vendor-defined Hostid
  - LM_A_VENDOR_ID_DECLARE
- Customized checkout

- LM_A_BORROW_EXPIRE

See Also
Advanced Flexible API Attributes
Obsolete Flexible API Attributes

Summary of Flexible API Attributes

The following table lists all the Flexible API attributes described in this chapter.

Table 4-1 • Flexible API attributes

Attribute	Description
LM_A_APP_DISABLE_CACHE_READ	Use to turn off automatic search for licenses from the previously successful location.
LM_A_BORROW_EXPIRE	Sets the date and time when a borrowed license expires.
LM_A_BORROW_RETURN_BY_VERSION	Controls whether the version of a feature can be specified when returning a borrowed feature.
LM_A_BORROW_STAT	Gets details of borrowed licenses on the local machine.
LM_A_CHECK_BADDATE	Checks whether the system date has been turned back: Use for license file-based licenses only.
LM_A_CHECK_INTERVAL	Turns off automatic heartbeats or defines the interval between automatic heartbeats.
LM_A_CKOUT_INSTALL_LIC	Use to turn off automatic storing of the license location used for a successful checkout.
LM_A_CHECK_LOCAL_TS_SIGNATURE	Verifies the signature of feature line in local trusted storage.
LM_A_CONN_TIMEOUT	Sets the connection time (in seconds) where the client would wait to receive a response from license server.
LM_A_LF_LIST	Lists all license files searched.
LM_A_LICENSE_DEFAULT	Sets the directory that will be searched first for licenses.
LM_A_LICENSE_FMT_VER	Use with lc_cryptstr to set the format of the license created.
LM_A_LINGER	Sets the time between the call to lc_checkin and when the license is checked in.
LM_A_LONG_ERRMSG	Use to turn off the display of long format error messages.
LM_A_PERROR_MSGBOX (Windows Only)	Use to turn off the display of errors in an error dialog.

Table 4-1 • Flexible API attributes

Attribute	Description
LM_A_PHYSICAL_ETHERNETID (Windows Only)	Use with lc_hostid to return only Ethernet hostids that are identified as belonging to physical devices.
LM_A_PORT_HOST_PLUS	Retrieves information for all licenses on the server without requiring the licenses to be checked out first.
LM_A_PROMPT_FOR_FILE (Windows Only)	Use to enable user input of the location and name of a license file.
LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL	Use to define the settings for automatic reconnection to a license server.
LM_A_SORT_TS_FIRST	Use to turn off the sorting of license rights so that licenses from trusted storage are searched first.
LM_A_TCP_TIMEOUT	Sets the time that the license server will wait for a heartbeat before checking in all licenses issued to a client.
LM_A_TS_CHECK_BADDATE	Checks whether the system date as recorded by trusted storage has been turned back.
LM_A_TS_ONLY	Use to checkout client and server based trusted storage licenses
LM_A_USE_FQDN	Use fully qualified domain name for hostid matching.
LM_A_USER_EXITCALL, LM_A_USER_EXITCALL_EX	Use to specify a callback routine used when reconnection to a license server fails.
LM_A_USER_RECONNECT, LM_A_USER_RECONNECT_EX	Use to specify a callback routine used before an attempt to reconnect to a license server.
LM_A_USER_RECONNECT_DONE, LM_A_USER_RECONNECT_DONE_EX	Use to specify a callback routine used after successfully reconnecting to a license server.
LM_A_USER_RECONNECT_V1_DATA	Use to specify a callback routine to pass reconnection data.
LM_A_USER_RECONNECT_V1	Use to specify a callback routine, called before every reconnection.
LM_A_USER_RECONNECT_V1_DONE	Use to specify a callback routine used after successfully reconnecting to a license server.
LM_A_VD_GENERIC_INFO, LM_A_VD_FEATURE_INFO	Use to request information from the vendor daemon.
LM_A_VENDOR_ID_DECLARE	Required when using a vendor-defined hostid: Identifies and provides details of a vendor-defined hostid.
LM_A_VM_FAMILY	Gets the Family details of the Virtualized Machine.

Table 4-1 • Flexible API attributes

Attribute	Description
LM_A_VM_NAME	Gets the Name details of the Virtualized Machine.
LM_A_WINDOWS_MODULE_HANDLE (Windows only)	Used only for licensed applications that are dynamic linked to the client library using a DLL.

LM_A_APP_DISABLE_CACHE_READ

Type

`int`

Default

`0`

After a successful checkout, the license file location used for the feature is by default cached in the <vendor>*_LICENSE_FILE* variable in the registry (Windows) or *$HOME/.flexlmrc* (UNIX). All subsequent checkouts for features from this vendor will read the cache first to determine the license file location.

To disable reading the cache, set this attribute to 1. Set this attribute immediately after the call to **lc_new_job** and before any other code in the application.

Note • *This attribute affects only how the FlexEnabled application locates the license file. If you set this attribute, your end users can set the* `LM_APP_DISABLE_CACHE_READ` *environment variable so that the Licensing toolkit utilities are disabled from reading the cache, as well. Make your end user aware of this environment variable.*

See Also
Automatic Update of License File Location
LM_A_CKOUT_INSTALL_LIC

LM_A_BORROW_EXPIRE

Type

`char *`

Default

None

Date and optional time when a borrowed license expires.

If you want your end users to request license borrowing from within your application, issue a license with the `BORROW` keyword on a feature line and write an interface in which the user specifies the end date (and optionally time) when the borrowed license will be returned. Before the call to **lc_checkout**, call:

```
lc_set_attr(job, LM_A_BORROW_EXPIRE, (LM_A_VAL_TYPE)datestring);
```

where `datestring` is the date (and optionally time) when the borrowed license expires (provided by the user through the interface). The date in `datestring` is in `dd-mmm-yyyy[:hh:mm]` format and the time is according to a 24-hour clock (for example, *13-dec-2016:15:00*). All subsequent license checkouts in this job will attempt to borrow licenses.

When querying using **lc_get_attr**, specify a valid `char` buffer as the `value` argument.

See Also
BORROW
Information about License Borrowing in the *Programming Reference for License File–Based Licensing*

LM_A_BORROW_RETURN_BY_VERSION

Type

`int`

Default

`1`

Controls whether the version of a feature can be specified when returning a borrowed feature. Client may require to choose one of the options while borrowing the feature licenses from the Vendor Daemon. Attribute LM_A_BORROW_RETURN_BY_VERSION can be either set as 1 or 0 for the client.

`LM_A_BORROW_RETURN_BY_VERSION=1`

- This is the default and recommended setting.
- The version of the borrowed feature can be specified when returning it.
- The feature name may not contain a hyphen.
- If using a utility to return the feature instead of **lc_borrow_return**, then `lmborrow` is used, with the `-bv` option.
- A borrow cache created by a client from version 11.14.0 or newer will be treated as if `LM_A_BORROW_RETURN_BY_VERSION=1`.

`LM_A_BORROW_RETURN_BY_VERSION=0`

- The feature name may contain a hyphen. This setting is intended to support legacy licensing models where a hyphen was introduced into the feature name.
- The version of the borrowed feature will be ignored when returning it.
- If using a utility to return the feature instead of **lc_borrow_return**, then `lmborrowl` should be used, not the original `lmborrow` utility.
- A borrow cache created by a client older than version 11.14.0 will be treated as if `LM_A_BORROW_RETURN_BY_VERSION=0`.

See Also
BORROW
Information about License Borrowing in the *Programming Reference for License File–Based Licensing*

LM_A_BORROW_STAT

Type

`LM_BORROW_STAT *`

Default

None

Use this attribute to retrieve information about features borrowed on the machine where `lmborrow` was run. This is useful for finding out which features have been borrowed while disconnected from the network. After the call to **lc_checkout** that activates borrowing, call:

```
LM_BORROW_STAT *s;
lc_get_attr(job, LM_A_BORROW_STAT, (LM_SHORT_PTR)&s);
for (; s; s = s->next)
{
    /* s contains local borrow status */
}
```

The same information is provided by running the Licensing utility `lmborrow -status`.

See Also
Information about License Borrowing in the *Programming Reference for License File–Based Licensing*

LM_A_CHECK_BADDATE

Type

`int`

Default

False

If True, and the license that authorizes the application has an expiration date, a check is made to see if the system date has been set back on the machine. When the license files are held on a network drive, this check may result in an appreciably slower checkout time. If the checkout fails for this reason, the checkout error is `LM_BADSYSDATE`.

This attribute should be set for file-based licensing only. When using trusted storage licensing, use the LM_A_TS_CHECK_BADDATE attribute instead.

Important • *If you set this attribute, you are advised to tell your end-users that installing license files on a network server may result in a degradation in the time for license checkout.*

See Also
Information about Limited Functionality Demos in the *Programming Reference for License File–Based Licensing*
Information about the ls_a_check_baddate Vendor Variable in the *Programming Reference for License File–Based Licensing*
LM_A_TS_CHECK_BADDATE

LM_A_CHECK_INTERVAL

Type

`int`

Default

120-second interval

`LM_A_CHECK_INTERVAL` controls the rate at which the FlexEnabled application sends automatic heartbeat messages to the license server. To disable automatic heartbeat messages, set `LM_A_CHECK_INTERVAL` to `-1`, (as well as setting `LM_A_RETRY_INTERVAL` to `-1`). The minimum value for `LM_A_CHECK_INTERVAL` is 30 seconds.

The results of possible settings of this variable are:

Table 4-2 • Variable Setting Results

Variable	Setting	Result
check_interval	-1	Automatic heartbeats are disabled.
check_interval	>= 0, < 30	Causes a failure due to invalid function parameters (LM_BADPARAM).
check_interval	>= 30	Timer interval.

If automatic heartbeats are disabled, you must call **lc_heartbeat** periodically to check the status of the license server.

See Also
lc_heartbeat
LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL
Information about Heartbeats in the *Programming Reference for License File–Based Licensing*

LM_A_CKOUT_INSTALL_LIC

Type

`int`

Default

None

By default, a successful checkout automatically updates the registry `VENDOR_LICENSE_FILE` setting (where `VENDOR` is your vendor name) to include the license file location that was used for the checkout. This can be disabled by setting this attribute to zero.

Note • *As an alternative to disabling this attribute, your end user can set the* `FLEXLM_NO_CKOUT_INSTALL_LIC` *environment variable. Care must be taken when setting this environment variable, as it affects all FlexNet licensing components and FlexEnabled applications running in the same environment and may undermine other publisher's methods for locating their license file.*

See Also
Automatic Update of License File Location

LM_A_CHECK_LOCAL_TS_SIGNATURE

Type

`int` which returns 0 or 1 when queried.

Default

1—signature check enabled

To verify the signature of the feature line in local trusted storage by default. The client side attribute is introduced to enable or disable the signature check of the local trusted storage license. lc_check_key now reports invalid license signature from a feature in local Trusted Storage.

Table 4-3 • LM_A_CHECK_LOCAL_TS_SIGNATURE attributes

Value	Description
0	To disable the signature check for local TS during checkout or lc_check_key
1	To enable signature check for local TS during checkout or lc_check_key

See Also
lc_check_key

LM_A_CONN_TIMEOUT

Type

`int`

Default

10 seconds

Range

Any positive integer. Negative values cause failure due to invalid function parameters (LM_BADPARAM).

LM_A_CONN_TIMEOUT attribute defines the maximum time (in seconds) the client would wait to receive a response after sending a single request to the license server. This attribute allows publishers to change the default timeout value, if needed.

Note • *The timeout should not be increased too much from its default value as this will affect client performance in bad network conditions, and in such scenarios, publisher may wish to perform some experiments to determine a suitable timeout value. The trade-off is small values results in better client performance but large values result in decreased probability of clients being timed out.*

LM_A_FLEXID9_HASP4_SUPPORT

Type

`unsigned int`

Default

`0`

Use this variable to turn the HASP4 detection on or off.

When set to `1`, `LM_A_FLEXID9_HASP4_SUPPORT` executes all HASP4 Safenet API calls. When set to `0`, no HASP4 APIs are executed.

LM_A_LF_LIST

Type

Pointer to `char **`

List of all license files searched for features during the last call to `lc_checkout`. Useful for failure messages for debugging. For example:

```
#include "lm_attr.h"
/*...*/
char **cp;
lc_get_attr(job, LM_A_LF_LIST, (short *)&cp);
if (cp)
{
        puts("files searched are: ");
        while (*cp) printf("\t%s\n",(short *)*cp++);
}
```

LM_A_LICENSE_DEFAULT

Type

`char *`

Note • *It is strongly recommended that this attribute be set in all FlexEnabled applications.*

The expected location of the application's *license file directory.* This is a directory that contains one or more license files with a `.lic` extension. This location can be hard-coded at compile time or determined dynamically at run time. However, it is recommended to determine this location dynamically at run time. For information about identifying the location of the license file, see license search path and locating the license file in the *Programming Reference for License File–Based Licensing*.

If `LM_A_LICENSE_DEFAULT` is set, FlexNet licensing still honors the `VENDOR_LICENSE_FILE` and `LM_LICENSE_FILE` environment variables and/or the license finder first.

Note • *On systems where directory names use non-ASCII characters, these characters must be encoded in UTF-8, for example oxE9 for the character é.*

LM_A_LICENSE_FMT_VER

Type

`char *`

Default

`LM_BEHAVIOR_V11`

Licenses generated by **lc_cryptstr** will be compatible with the version specified. Valid arguments are `LM_BEHAVIOR_Vx`, where x is `2, 3, 4, 5, 5_1, 6, 7, 7_1, 8, 8_1, 8_2, 8_3, 9, 9_3, 10 or 11`. If the license compatible with the desired version cannot be generated:

- The error `LM_LGEN_VER` (-94) will be generated: `Attempt to generate license with incompatible attributes`.
- The `FEATURE` line will be left as is, without replacing the signature with a correct one.

See Also
lc_cryptstr

LM_A_LINGER

Type

long

Default

0 (no linger)

This attribute controls the license linger time for your application. A lingering license allows you to specify how long a license will remain checked out beyond either the call to **lc_checkin** or program exit (whichever comes first). To use this feature, call **lc_set_attr** before checking out the feature that should linger:

```
lc_set_attr(job, LM_A_LINGER, (LM_A_VAL_TYPE)x)
```

where x is the number of seconds to make the license linger.

In addition, the end user can specify a longer linger interval in the vendor daemon's options file, as such:

```
LINGER f1 100
```

The longer of the developer-specified and user-specified times will be used. The actual time of checkin will vary somewhat since the vendor daemon checks all lingering licenses once per minute. If, however, a new license request is made that would otherwise be denied, a check of the lingering licenses is made immediately to attempt to satisfy the new request. Linger is useful for programs that normally take under a minute to complete and when either DUP_GROUP is set in the license file or dup_group is set in the call to **lc_checkout**.

See Also
LM_A_LICENSE_FMT_VER
Information about FEATURE and INCREMENT lines in the *Programming Reference for License File–Based Licensing*
lc_checkout

LM_A_LONG_ERRMSG

Type

int

Default

True

The default is long error messages. Error messages can be presented in a long, more descriptive format. This format contains embedded newline characters, which some applications may not be able to handle or may need special handling.

Applications will often find it useful to present the short error message first, and then long error message upon user request. This can be done thus:

```
lc_set_attr(job, LM_A_LONG_ERRMSG, (LM_A_VAL_TYPE)0);
....
/*error occurs*/
lc_perror(job);
/* user requests long error message */
```

```
lc_set_attr(job, LM_A_LONG_ERRMSG, (LM_A_VAL_TYPE)1);
lc_perror(job);
```

Note that this works only if another Flexible API error does not occur in between, which would change the error condition and message. Not all error conditions have long explanations or context-sensitive information.

Example

```
Invalid host
  The hostid of this system does not match the hostid
  specified in the license file
Hostid:           12345678
License path:     ./file1.lic:./file2.lic:./file3.lic
net Licensing error:     -9,9
```

The format is:

```
short-error-description
optional-long-explanation [1-3 lines]
optional-context-information
License path:   path1:...:pathn
net Licensing error:   major, minor
```

LM_A_PERROR_MSGBOX (Windows Only)

Type

`int`

Default

False

If True, **lc_perror** presents the error message in an error dialog. Also turned off when `FLEXLM_BATCH` is set.

LM_A_PHYSICAL_ETHERNETID (Windows Only)

Type

`short`

Default

False

If True, lc_hostid returns only Ethernet hostids that are identified as belonging to physical devices and filters out virtual adapter values (such as those installed by VPN software) on Windows platforms. This filtering option is provided because using a virtual adapter as a hostid can cause license failure when the Ethernet identity created by the virtual adapter changes. The default is False, which turns the filtering off for backward compatibility.

Note • *This option will not filter out virtual adapters installed by virtual machines. This attribute is only effective on the hostid type **ETHER** or **PHY_ETHER**.*

LM_A_PORT_HOST_PLUS

Type

`short`

Default

1

Discussion

When set to `1` or greater, this attribute makes no request to retrieve authenticated information about a feature from the server.

When set to `0`, it retrieves authenticated information (from FEATURE or INCREMENT lines) on the license server and stores this information in the job handle on the client side. This attribute is useful when the customer wants to retrieve authenticated information about licenses *without* checking out the licenses first. You can use this attribute in conjunction with other attributes such as LM_A_LICENSE_FILE_PTR to designate the specific license file whose information you want to retrieve.

Compare this option with the **lc_auth_data** function, which also obtains authenticated information for one or more features. However, a license must be checked out before the client application can call **lc_auth_data** to retrieve information about the license. Additionally, the application must call **lc_auth_data** for each checked-out license.

Important • *The LM_A_PORT_HOST_PLUS attribute requires that the license file use USE_SERVER.*

See Also
lc_auth_data

LM_A_PROMPT_FOR_FILE (Windows Only)

Type

`int`

Default

False

When True, the user is prompted for the license file path or server name or IP address, if needed. Also turned off when `FLEXLM_BATCH` is set.

LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL

Type

`int`

Default

Five for `LM_A_RETRY_COUNT`, 60 for `LM_A_RETRY_INTERVAL`.

Together, `LM_A_RETRY_COUNT` and `LM_A_RETRY_INTERVAL` are used for automatic reconnection to a license server. When license server failure is detected, the automatic heartbeat mechanism attempts to reconnect to the license server. If reconnection fails, then the reconnect will be attempted `LM_A_RETRY_COUNT` times at intervals of `LM_A_RETRY_INTERVAL`. If the application wants manual control over this activity, enable manual heartbeats by setting `LM_A_RETRY_INTERVAL` to -1 and `LM_A_CHECK_INTERVAL` to -1. The minimum value for `LM_A_RETRY_INTERVAL` is 30 seconds.

If `LM_A_RETRY_COUNT` is set to -1, the application will attempt retrying forever. For applications desiring a more lenient policy, this is recommended.

See Also
lc_heartbeat
LM_A_CHECK_INTERVAL
LM_A_USER_EXITCALL, LM_A_USER_EXITCALL_EX

LM_A_SORT_TS_FIRST

Type

`int`

Default

One—sort so that licenses from trusted storage are used first

This attribute allows you to change how licenses are sorted when loaded into memory. By default, the sort rules are that FEATURE and INCREMENT lines from trusted storage appear at the beginning of the sort order. FEATURE and INCREMENT lines from license files appear later. Within these two groups, features are sorted by the order that each line is listed.

When the attribute is set to zero (0), the pre-11.4 release sorting rules are used.

See Also
Information about adding FlexNet Licensing function calls to an application in the *Programming Reference for Trusted Storage–Based Licensing*
Information about the SORT keyword in the *Programming Reference for License File–Based Licensing*

LM_A_TS_ONLY

Type

`int`

Default

0

This attribute allows you to check out only server or client-based trusted storage licenses. This is to avoid using license file–based licenses. To use `LM_A_TS_ONLY` attribute set the `LM_A_SORT_TS_FIRST`, this helps in sorting the FEATURE and INCREMENT lines from trusted storage appear at the beginning.

Illustration for client-side and server-side behavior

If a feature ADD is present in server trusted storage and served certificate and if the feature is in the same pool, then the feature gets rejected.

Table 4-4 • An illustration of server-side behavior

Feature	Served Trusted storage	Served Certificate	Result
ADD	Yes (nlic = 2)	Yes (nlic = 3)	Feature will be in same pool. Hence will not checkout.
ADD	Yes (1.0)	Yes (2.0)	Feature will be in different pool. Hence checks out only ADD 1.0 feature, but not ADD 2.0.

If the same feature ADD is present in client trusted storage and unserved certificate. On trying to checkout uncounted feature from the client with the `LM_A_TS_ONLY` attribute being set "`No such feature exists`" error is seen.

Table 4-5 • An illustration of client-side behavior

Feature	Client Trusted storage	Unserved Certificate	Result
ADD	Yes (nlic=3)	Yes (uncounted)	Since the features are in different pools, we can checkout feature ADD in client trusted storage, but not from uncounted feature.

See Also
LM_A_SORT_TS_FIRST

LM_A_TCP_TIMEOUT

Type

`int`

Default

7200 seconds (2 hours)

Maximum

15300 seconds (4 hours 15 minutes).

During normal operation, the license server exchanges heartbeats with FlexEnabled clients to ensure that they are connected. If a FlexEnabled client disconnects from the license server, the license server will check in all licenses that are associated with the client application.

This attribute is used in situations where the license server stops receiving heartbeats from a client that has not closed the socket connection. This value is used to specify how long the license server waits since it received the last heart beat before it determines that the client is disconnected and checks in all licenses associated with that client.

The best-practice formula for setting the value of `LM_A_TCP_TIMEOUT` when using automatic heartbeats is as follows:

`LM_A_TCP_TIMEOUT` = (`LM_A_CHECK_INTERVAL` x 2) + `LM_A_RETRY_COUNT` x `LM_A_RETRY_INTERVAL` + 60.

This formula optimizes the tension between the following cases:

- The license server reclaims a license from a disconnected client in reasonable time (earlier than the 2 hour default).
- A client does not lose its license simply due to a temporary network outage.

Setting the value to 0 (zero) turns off the timeout (that is, if the heartbeats stop), but the socket connection is not closed. The license server does not check in licenses that are associated with the client application.

See Also
lc_heartbeat
lc_checkin
LM_A_CHECK_INTERVAL
LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL

LM_A_TS_CHECK_BADDATE

Type

`short`

Default

False

If True, and the license that authorizes the application has an expiration date, a check is made to see if the timestamps in trusted storage indicate that the clock on the machine has been wound back. When windback is detected, the checkout fails with the checkout error `LM_TS_BADDATE`.

This attribute is used to check for windback when the license rights are held in trusted storage. When using file-based licensing, use the `LM_A_CHECK_BADDATE` attribute instead. Both attributes can be set when a FlexEnabled application can use license rights either in license files or in trusted storage.

See Also
LM_A_CHECK_BADDATE

LM_A_USE_FQDN

Type

`int`

Default

0

This attribute specifies which host-naming convention the FlexEnabled application uses for the local host name.

Table 4-6 • Attribute Host-Naming Convention

Value	Description
0	Only the relative distinguished name is used. This is the name to the left of the left-most dot in a fully qualified domain name (FQDN). For example, given an FQDN of **myhost.engr.flexerasoftware.com**, only the component, `myhost`, is used. This is the default setting.
1	The fully qualified domain name is used. The FQDN is composed of the relative distinguished name and the DNS hierarchy name. Consider, for example, the FQDN **myhost.engr.flexerasoftware.com**—`myhost` is the relative distinguished name and **engr.flexerasoftware.com** is the DNS hierarchy name.

This attribute must be set to 1, use the fully qualified domain name, when wildcards are being used in domain names within license files. For example when using the HOSTID keyword:

```
HOSTID=HOSTNAME=*.flexerasoftware.com.
```

It is recommended that you notify your end user if you set this attribute to 1.

For served licenses, the host name is sent to the license server preserving the requested naming convention.

This setting affects the way host names are matched in each of the following contexts:

- `SERVER` (for served licenses), `FEATURE`, and `INCREMENT` lines

  The FlexEnabled application or—in the case of served licenses—the license server, validates as much information about the host name as is specified in the feature line. Consider the following examples:

  ```
  FEATURE f1 ... HOSTID=HOSTNAME=myhost.abc.com ...
  FEATURE f2 ... HOSTID=HOSTNAME=myhost ...
  FEATURE f3 ... HOSTID=HOSTNAME=*.abc.com ...
  ```

  In this example, a request for:

  - Feature f1 is restrictive. It is granted only to a system with the exact FQDN of `myhost.abc.com`.

  - Feature f2 is less restrictive. Any request originating from `myhost` is granted regardless of its DNS hierarchy name.
  - A request for feature f3 is the least restrictive. Any request originating from any hostname in the domain is granted. Note that you must set `LM_A_USE_FQDN` to 1 if you are using wildcards in domain names.

- Duplicate grouping

  Host-based duplicate group processing is based on as much of the host name as is used by the FlexEnabled application. For applications using FQDNs, only a system with an exact FQDN match is counted as a duplicate.

- Host names in an end-user options file

  Host names in the end-user options file are validated with respect to the format used by the FlexEnabled application. For FQDNs, an exact match is required by default; however, the matching criteria can be configured by the end user via the `FQDN_MATCHING` options file keyword.

See Also
DUP_GROUP in *Programming Reference for License File–Based Licensing*
LM_A_HOST_OVERRIDE
FDQN_MATCHING options file keyword in the *License Administration Guide*

LM_A_USER_EXITCALL, LM_A_USER_EXITCALL_EX

Type

Pointer to a function returning `int`. Return value unused.

Default

No user exit handler (FlexEnabled application exits)

Discussion

The function pointer `LM_A_USER_EXITCALL` (or the extended version, `LM_A_USER_EXITCALL_EX`) is set to point to the callback routine that is to receive control if reconnection fails after `LM_A_RETRY_COUNT` attempts. If no routine is specified, then **lc_perror** is called, and the FlexEnabled application will exit. The `LM_A_USER_EXITCALL` routine is called as follows:

```
(*exitcall)(feature);
```

or, with `LM_A_USER_EXITCALL_EX`:

```
(*exitcallEx) (job, feature, vendor_data);
```

This callback function is an exit handler; it should not return. Behavior is undefined if any Flexible API client routine is called after this exit handler is called. If you want to do anything more subtle than exiting, do not specify an exit handler function. Instead, set `LM_A_RETRY_COUNT` to -1 and specify an `LM_A_USER_RECONNECT` function that sets a global variable; monitor this variable in the mainline code (not in a callback function). Take appropriate action when the global variable reaches your defined limit.

Note • `LM_A_USER_EXITCALL_EX` *provides a way to pass vendor-defined data to this callback function. Behavior is undefined if both* `LM_A_USER_EXITCALL_EX` *and* `LM_A_USER_EXITCALL` *are set.*

Callback Parameters

`LM_A_USER_EXITCALL` or the extended version, `LM_A_USER_EXITCALL_EX` uses the following parameters:

Table 4-7 • LM_A_USER_EXITCALL, LM_A_USER_EXITCALL_EX Callback Parameters

Callback Parameter	Description
job	`LM_A_USER_EXITCALL_EX` only: A pointer to the current license job.
feature	A pointer to the feature name.
user_data	`LM_A_USER_EXITCALL_EX` only: A pointer to user-defined data set via the `LM_A_VENDOR_CALLBACK_DATA` attribute.

See Also
lc_heartbeat
LM_A_RETRY_COUNT, LM_A_RETRY_INTERVAL
LM_A_VENDOR_CALLBACK_DATA

LM_A_USER_RECONNECT, LM_A_USER_RECONNECT_EX

Type

Pointer to a function returning `int`. Return value unused.

Default

No user reconnection handler

Discussion

The function pointer `LM_A_USER_RECONNECT` (or the extended version, `LM_A_USER_RECONNECT_EX`) is set to point to a reconnection callback routine. This routine is called each time just before a reconnection is attempted, either by the automatic heartbeat mechanism, or as a result of the application program calling **lc_heartbeat**.

The `LM_A_USER_RECONNECT` routine is called as follows:

```
(*reconnect)(feature, pass, total_attempts, interval)
```

or, with `LM_A_USER_RECONNECT_EX`:

```
(*reconnectEx)(job, feature, pass, total_attempts, interval, vendor_data)
```

Note • LM_A_USER_RECONNECT_EX *provides a way to pass vendor-defined data to this callback function. Behavior is undefined if both* LM_A_USER_RECONNECT_EX *and* LM_A_USER_RECONNECT *are set.*

Callback Parameters

LM_A_USER_RECONNECT or the extended version, LM_A_USER_RECONNECT_EX uses the following callback parameters:

Table 4-8 • LM_A_USER_RECONNECT, LM_A_USER_RECONNECT_EX Callback Parameters

Callback Parameter	Description
job	LM_A_USER_RECONNECT_EX only: A pointer to the current license job.
feature	Feature name.
pass	Current attempt number.
total_attempts	Maximum number of passes that will be attempted.
interval	Time in seconds between reconnection attempts.
vendor_data	LM_A_USER_RECONNECT_EX only: A pointer to vendor-defined data set via the LM_A_VENDOR_CALLBACK_DATA attribute.

See Also
lc_heartbeat
LM_A_CHECK_INTERVAL
LM_A_USER_EXITCALL, LM_A_USER_EXITCALL_EX
LM_A_VENDOR_CALLBACK_DATA

LM_A_USER_RECONNECT_DONE, LM_A_USER_RECONNECT_DONE_EX

Type

Pointer to a function returning `int`. Return value unused.

Default

None.

Discussion

The function pointer LM_A_USER_RECONNECT_DONE (or the extended version, LM_A_USER_RECONNECT_DONE_EX) is set to point to a callback routine. This routine is called when reconnection is successfully completed.

The LM_A_USER_RECONNECT_DONE handler is called as follows:

```
(*reconnect_done)(feature, tries, total_attempts, interval)
```

or, for LM_A_USER_RECONNECT_DONE_EX:

```
(*reconnect_doneEx)(job, feature, tries, total_attempts, interval, vendor_data)
```

Note • LM_A_USER_RECONNECT_DONE_EX *provides a way to pass vendor-defined data to this callback function. Behavior is undefined if both* LM_A_USER_RECONNECT_DONE_EX *and* LM_A_USER_RECONNECT_DONE *are set.*

Callback Parameters

LM_A_USER_RECONNECT_DONE (or the extended version, LM_A_USER_RECONNECT_DONE_EX) uses the following callback parameters:

Table 4-9 • LM_A_USER_RECONNECT_DONE, LM_A_USER_RECONNECT_DONE_EX Callback Parameters

Callback Parameter	Description
job	LM_A_USER_RECONNECT_DONE_EX only: A pointer to the current license job.
feature	Feature name.
tries	Number of attempts that were required to re-connect for this feature.
total_attempts	Maximum number of retry attempts that would be made.
interval	Interval in seconds between reconnection attempts.
vendor_data	LM_A_USER_RECONNECT_DONE_EX only: A pointer to vendor-defined data set via the LM_A_VENDOR_CALLBACK_DATA attribute.

See Also
LM_A_USER_RECONNECT, LM_A_USER_RECONNECT_EX
LM_A_VENDOR_CALLBACK_DATA

LM_A_USER_RECONNECT_V1*

The optional client attributes LM_A_USER_RECONNECT_V1, LM_A_USER_RECONNECT_V1_DATA, and LM_A_USER_RECONNECT_V1_DONE are specific to client reconnections that use the LM_CO_AVAIL_NOWAIT checkout option flag.

LM_A_USER_RECONNECT_V1_DATA

Type

Pointer to a function returning int. Return value unused.

Default

NONE.

Discussion

This attribute allows you to set a pointer to reconnect data which would be used by LM_A_USER_RECONNECT_V1 and LM_A_USER_RECONNECT_V1_DONE callbacks to pass reconnection data to the callback routine.

Callback Parameters

LM_A_USER_RECONNECT_V1_DATA uses the following callback parameters:

Table 4-10 • LM_A_USER_RECONNECT_V1_DATA Parameters

Callback Parameter	Description
userReconnectData	A pointer to FNP_RECONNECT_DATA type.

The FNP_RECONNECT_DATA structure is defined in the header file lmclient.h.

See Also
LM_A_USER_RECONNECT_V1
LM_A_USER_RECONNECT_V1_DONE
..\examples\advanced\client\lmflex_coavail.c

LM_A_USER_RECONNECT_V1

Type

Pointer to a function returning int. Return value unused.

Default

No user reconnection handler.

Discussion

The function pointer LM_A_USER_RECONNECT_V1 is set to point to a reconnection callback routine. This routine is called each time just before a reconnection is attempted, either by automatic heartbeat mechanism or as a result of the client application calling lc_heartbeat.

The LM_A_USER_RECONNECT_V1 routine is called as follows:

```
(*sUserReconnectV1) (FNP_RECONNECT_DATA *reconnect_v1_data)
```

The attribute LM_A_USER_RECONNECT_V1 works in a similar way to the existing attribute LM_A_USER_RECONNECT with the variation in the callback parameter. The callback routine of LM_A_USER_RECONNECT_V1 takes FNP_RECONNECT_DATA pointer as the callback parameter which has to be set through LM_A_USER_RECONNECT_V1_DATA attribute. The callback routine set by the attribute LM_A_USER_RECONNECT_V1 could pass additional reconnection data for partial available checkouts (aka COAVAIL checkouts). Refer the structure definition of FNP_RECONNECT_DATA type in lmclient.h

Callback Parameters

LM_A_USER_RECONNECT_V1 uses the following callback parameter:

Table 4-11 • LM_A_USER_RECONNECT_V1 Parameters

Callback Parameter	Description
reconnect_v1_data	A pointer to FNP_RECONNECT_DATA structure set using LM_A_USER_RECONNECT_V1_DATA attribute.

Dependency

The LM_A_USER_RECONNECT_V1 requires the attribute LM_A_USER_RECONNECT_V1_DATA to set.

See Also
LM_A_USER_RECONNECT_V1_DATA
LM_A_USER_RECONNECT_V1_DONE
`..\examples\advanced\client\lmflex_coavail.c`

LM_A_USER_RECONNECT_V1_DONE

Type

Pointer to a function returning `int`. Return value unused.

Default

No user reconnection handler.

Discussion

The function pointer LM_A_USER_RECONNECT_V1_DONE is set to point to a reconnection done callback routine. This routine is called when reconnection is successfully completed.

The LM_A_USER_RECONNECT_V1_DONE routine is called as follows:

```
(*sUserReconnectV1Done) (FNP_RECONNECT_DATA *reconnect_v1_data)
```

The attribute LM_A_USER_RECONNECT_V1_DONE works in a similar way to the existing attribute LM_A_USER_RECONNECT_DONE with the variation in callback parameter. The callback routine of LM_A_USER_RECONNECT_V1_DONE takes the FNP_RECONNECT_DATA pointer as the callback parameter which has to be set through LM_A_USER_RECONNECT_V1_DATA attribute. The callback routine set by the attribute LM_A_USER_RECONNECT_V1_DONE could pass additional reconnection data for partial available checkouts (aka COAVAIL checkouts). Refer the structure definition of FNP_RECONNECT_DATA type in `lmclient.h`.

Callback Parameters

LM_A_USER_RECONNECT_V1_DONE uses the following callback parameter:

Table 4-12 • LM_A_USER_RECONNECT_V1_DATA Parameters

Callback Parameter	Description
reconnect_v1_data	A pointer to FNP_RECONNECT_DATA type set using LM_A_USER_RECONNECT_V1_DATA attribute.

Dependency

The LM_A_USER_RECONNECT_V1_DONE requires the attribute LM_A_USER_RECONNECT_V1_DATA to set.

See Also
LM_A_USER_RECONNECT_V1_DATA
LM_A_USER_RECONNECT_V1
..\examples\advanced\client\lmflex_coavail.c

LM_A_VD_GENERIC_INFO, LM_A_VD_FEATURE_INFO

Type

Pointer to LM_VD_GENERIC_INFO or pointer to LM_VD_FEATURE_INFO

Both attributes get information from your vendor daemon. LM_A_VD_GENERIC_INFO gets information that is not specific to a feature and is mostly found in lsvendor.c.

LM_A_VD_FEATURE_INFO gets information about a particular feature and provides an accurate count of licenses used, users queued, and so on, and works correctly when a license file has more than one FEATURE or INCREMENT line for the same feature name. This will result in a LM_NOSERVSUPP error if the particular CONFIG struct has been merged with another CONFIG in the vendor daemon.

These attributes will work only on your vendor daemon. If a request is made for a feature served only by a different vendor daemon, then the LM_NOADMINAPI error results.

A pointer to a struct is given as an argument to **lc_get_attr**. This structure must have its feat member set. Upon successful return, this struct is filled with the appropriate information. lc_get_attr returns an error when an uncounted license is called. The following example illustrates the use of both attributes. Although **lc_get_config** and **lc_next_conf** are described in Summary, one of their legitimate uses is with LM_A_VD_GENERIC_INFO and LM_A_VD_FEATURE_INFO.

Note • *If you are reporting on a feature that has been checked out successfully, use* **lc_auth_data** *instead of* **lc_next_conf.**

```
#include "lmclient.h"
#include "lm_code.h"
#include "lm_attr.h"
/* ... */
/*
```

```
 * Print out GENERIC and FEATURE information for every
 * license file line for a given feature name
 */
void
vendor_daemon_info(LM_HANDLE *job, char *feature)
{
  CONFIG *conf, *c;
  LM_VD_GENERIC_INFO gi;
  LM_VD_FEATURE_INFO fi;
  int first = 1;

    c = (CONFIG *)0;

    for (conf = lc_next_conf(job, feature, &c);conf;
                         conf=lc_next_conf(job, feature, &c))
    {
        if (first)

        {
/*
 *           get generic daemon info
 */
             gi.feat = conf;
             if (lc_get_attr(job, LM_A_VD_GENERIC_INFO, (short *)&gi))
             {
                   lc_perror(job, "LM_A_VD_GENERIC_INFO");
             }
             else
             {
                   printf(" conn-timeout %d\n", gi.conn_timeout);
                   printf(" normal_hostid %d\n", gi.normal_hostid);
                   printf(" minimum_user_timeout %d\n",
                           gi.minimum_user_timeout);
                   printf(" min_lmremove %d\n", gi.min_lmremove);
                   printf(" use_featset %d\n", gi.use_featset);
                   printf(" dup_sel 0x%x\n", gi.dup_sel);
                   printf(" use_all_feature_lines %d\n",
                                   gi.use_all_feature_lines);
                   printf(" do_checkroot %d\n", gi.do_checkroot);
                   printf(" show_vendor_def %d\n", gi.show_vendor_def);
            }
             first = 0;
        }
/*
 *      get specific feature info
 */
        fi.feat = conf;
        if (lc_get_attr(job, LM_A_VD_FEATURE_INFO, (short *)&fi))
        {
            lc_perror(job, "LM_A_VD_FEATURE_INFO");
        }
        else
        {
             printf("\nfeature %s\n", conf->feature);
             printf("code %s\n", conf->code);
             printf("rev %d\n", fi.rev);
```

```
            printf("timeout %d\n", fi.timeout);
            printf("linger %d\n", fi.linger);
            printf("res %d\n", fi.res);
            printf("tot_lic_in_use %d\n", fi.tot_lic_in_use);
            printf("float_in_use %d\n", fi.float_in_use);
            printf("user_cnt %d\n", fi.user_cnt);
            printf("num_lic %d\n", fi.num_lic);
            printf("queue_cnt %d\n", fi.queue_cnt);
            printf("overdraft %d\n", fi.overdraft);
        }
    }
}
```

Detecting OVERDRAFT for Suites

This is a special case for `OVERDRAFT`. With suites, when you check out a feature, you also silently check out a token for the suite. Both the suite and feature token may be in the `OVERDRAFT` state, or only one, or neither. To detect suite overdraft, the code must get the parent/suite feature name, and then check for overdraft for this feature. Use **lc_auth_data** to ensure that you get the `CONFIG` struct for the license that has been checked out.

```
conf=lc_auth_data();
if ((conf->package_mask & LM_LICENSE_PKG_COMPONENT)
    && (conf->package_mask & LM_LICENSE_PKG_SUITE))
{
    fi.feat = conf->parent_feat;
    if (lc_get_attr(job, LM_A_VD_FEATURE_INFO, (short *)&fi))
            lc_perror(job, "LM_A_VD_FEATURE_INFO");
    else
            printf("suite overdraft is %d\n", fi.overdraft);
}
```

See Also
lc_get_config
lc_next_conf
lc_auth_data

LM_A_VENDOR_ID_DECLARE

Type

Pointer to `LM_VENDOR_HOSTID` struct.

Default

None

This is for supporting vendor-defined hostid. The `struct` defines and declares the hostid to FlexNet licensing.

See Also
Information about Vendor-Defined Hostid Types in *Programming Reference for License File–Based Licensing*
`machind/lmclient.h` for `LM_VENDOR_HOSTID` definition
`examples/vendor_hostid` directory

LM_A_VM_FAMILY

Displays the virtualization family details. This attribute is read-only.

Type

char*

Default

None

Here the Family denotes the manufacturer or the publisher of the detected hypervisor software where the Virtualized machine (VM) is running.

Currently supported Family names are:

- “MICROSOFT”
- “UNKNOWNVM”
- “VIRTUALBOX”
- “VMWARE”
- “XEN”
- “AMAZON”
- “PARALLELS”
  - “QEMU”
- “GOOGLE”

For example:

```
char *vm_family;

if (LM_NOERROR == lc_get_attr(lm_job, LM_A_VM_FAMILY, (LM_SHORT_PTR)&vm_family))
{
    printf("VM Family: %s\n", vm_family );
}
```

LM_A_VM_NAME

Displays the name of the virtual machine. This attribute is read-only.

Type

char*

Default

None

The Name specifies the detected hypervisor product that the Virtualized machine (VM) is running on. The Name is a sub-set of the Family.

Currently supported Names are:

- “EC2”
- “HYPERV”
- “UNKNOWNVM”
- “VIRTUALBOX”
- “SERVER, Previously VMWARE”
- “DESKTOP, Previously VMWARE”
- “XEN”
- “PARALLELS”
- “QEMU-KVM”
- “COMPUTE”
- “AZURE”

For example:

```
char *vm_name;

if (LM_NOERROR == lc_get_attr(lm_job, LM_A_VM_NAME, (LM_SHORT_PTR)&vm_name))
{
    printf("VM Name: %s\n", vm_name );
}
```

LM_A_WINDOWS_MODULE_HANDLE (Windows only)

Type

`long`

Default

0 (zero)

This is only needed for a specific situation on Windows: You are building a DLL, and the Licensing toolkit client library (`lmgr_trl.lib`) is linked into your DLL. Put another way, the Licensing functions are not in a static binary, but only in a DLL. In this case, the DLL makes calls to the following functions before calling **lc_checkout**—

```
lc_set_attr(job, LM_A_WINDOWS_MODULE_HANDLE,
         (LM_A_VAL_TYPE)GetModuleHandle(dllname));
```

—where *dllname* is the name of the DLL. If these calls are not made, Windows dialogs and error messages do not work properly.

Determining License Source and Count of a Feature

For CONFIG member definitions refer to `machind/lmclient.h`.

users

Licensed number of users on the feature definition line

nConfigType

Indicates the license source and the value of this member takes one of the following:

- FNP_CONFIG_TYPE_UNKNOWN(0)
- FNP_CONFIG_TYPE_CLIENT_TS(1)—License is from client trusted storage.
- FNP_CONFIG_TYPE_UNSERVED_CERT(2)—License is from unserved certificate.
- FNP_CONFIG_TYPE_SERVED_TS(3)—License is from served trusted storage.
- FNP_CONFIG_TYPE_SERVED_CERT(4)—License is from served certificate.

nTSFeatFlag

Indicates whether the feature is from license file or trusted storage.

The values of “nTSFeatFlag” member is one of following,

- 0-- Indicates the feature is from license file.
- 1--Indicates the feature is from trusted storage.

nTotalFeatureCount

Holds the total feature count for a feature obtained from trusted storage.

Table 4-13 • values of CONFIG member in different scenarios:

nConfigType	lc_from_server	nTSFeatFlag	users, nTotalFeaturecount
FNP_CONFIG_TYPE_CLIENT_TS	0	1	nTotalFeatureCount= users
FNP_CONFIG_TYPE_UNSERVED_CERT	0	0	
FNP_CONFIG_TYPE_SERVED_CERT	1	0	
FNP_CONFIG_TYPE_SERVED_TS	1	1	nTotalFeatureCount = user * fulfillment record count

5

Advanced Flexible API Attributes

This chapter lists attributes that provide advanced FlexNet licensing functionality. Basic functionality is covered by the material in Flexible API Attributes.

Summary

These attributes provide advanced capabilities beyond what is provided by the basic attributes in Flexible API Attributes. They should be used with care, and questions are welcomed before their use. These attributes are summarized in Table 5-1.v

Table 5-1 • Advanced Flexible API Attributes

Attribute	Description
LM_A_CHECKOUT_DATA	Allows labeling of a given checkout request.
LM_A_MULTIPLE_CHECKOUT_DATA	Defines a pointer to a checkout filter function for feature pre-processing.
LM_A_CHECKOUTFILTERLAST_EX	Defines a pointer to a checkout filter function for feature post-processing.
LM_A_CROSS_TZ_OK	Enables a 24-hour tolerance for licenses to accommodate time-zone differences.
LM_A_DIAGS_ENABLED	Controls the output of detailed diagnostic data.
LM_A_DISABLE_ENV	Controls the use of the LM_LICENSE_FILE environment variable.
LM_A_DISPLAY_OVERRIDE	Overrides the display name.
LM_A_HOST_OVERRIDE	Overrides the host name.

Table 5-1 • Advanced Flexible API Attributes

Attribute	Description
LM_A_IGNORE_CACHED_SERVER_DATA	Use to turn off use of stored license server data when reconnecting, which may result in multiple licenses being checked out to a client.
LM_A_IGNORE_FILE_CHECK_IN_HEARTBEAT (Linux Only)	Forces the client to ignore the check for the borrow-cache file.
LM_A_INTERNET_OVERRIDE	Override the IP address.
LM_A_LICENSE_CASE_SENSITIVE	Controls license file case-sensitivity.
LM_A_LKEY_CASE_SENSITIVE	Controls license file case-sensitivity in the context of license key authentication.
LM_A_MT_HEARTBEAT (UNIX Only)	Controls the use of multithreaded heartbeats.
LM_A_MULTIPLE_CHECKOUT_DATA	Allows labeling a checkout request. For each label in request a count of license will be consumed.
LM_A_PERIODIC_CALL	Defines a pointer to a function to be called after the specified number of heartbeats.
LM_A_PERIODIC_COUNT	Specifies the number of heartbeats between calls to function specified by LM_A_PERIODIC_CALL.
LM_A_PLATFORM_OVERRIDE	Overrides the platform name.
LM_A_RETRY_CHECKOUT	Controls checkout retries.
LM_A_TZ_OVERRIDE	Platform-independent attribute used to override the client time zone and time. For testing purposes only, `LM_A_TZ_OVERRIDE` can be used to test time zone licensing.
LM_A_HEARTBEAT_LOCK (Windows only)	Introduced a mutex lock between main and timer thread to access the global timer data structures.
LM_A_USER_OVERRIDE	Overrides the user name.
LM_A_VENDOR_CALLBACK_DATA	This attribute allows you to set a pointer to vendor-defined data.
LM_A_VSEND_NOWAIT	Sends an asynchronous message to vendor daemon.

LM_A_CHECKOUT_DATA

Type

char*

Default

None

The LM_A_CHECKOUT_DATA attribute allows you to set a checkout-data string. It is used to label the next successful checked out feature or to qualify a feature check in request with the checkout-data label.

The checkout-data string is a character string, with a maximum size of MAX_VENDOR_CHECKOUT_DATA bytes (1024 bytes). The default value is the NULL string. Each unique value of LM_A_CHECKOUT_DATA represents a unique license group; the NULL string can be one of those unique values. Like all other attributes, set this before the checkout or checkin request; it takes effect for all subsequent calls to **lc_checkout** or **lc_checkin** until it is changed.

When querying using **lc_get_attr**, specify a valid char buffer as the value argument.

There are two scenarios where a checkout-data string, defined via the LM_A_CHECKOUT_DATA attribute setting, can be used:

- To force the license count to be accumulated incrementally across multiple checkouts in the same license job rather than as an aggregate.
- To create a custom duplicate grouping criteria.

These scenarios are described in the following sections.

Incremental License Count in a Single Job

By default, checkout requests for the same feature and version from the same license job have an aggregate affect on the number of licenses consumed. That is, the Licensing toolkit client library checks out only as many additional licenses for the feature as necessary to reach the count specified in the checkout request. For example:

If an initial checkout request which asks for two licenses for feature “f1”

```
lc_checkout(lm_job, f1, "1.0", 2,...,LM_DUP_NONE);
```

is followed by a second call which asks for five licenses for the same feature,

```
lc_checkout(lm_job, f1, "1.0", 5,...,LM_DUP_NONE);
```

the second call checks out only three additional licenses (5 minus 2). If the second call asks for the same or fewer number of licenses than were already checked out, the second call does not check out additional licenses.

To force identical checkout requests, instead, to have an incremental effect on the number of consumed licenses, include each call to **lc_checkout** in its own license job. Alternatively, use the LM_A_CHECKOUT_DATA attribute to set a unique checkout-data string to the job just prior to the call to **lc_checkout**. This causes the second or subsequent request to check out exactly the number of licenses specified in the call to **lc_checkout**, regardless of the number asked for in previous requests. For example:

```
/* Check out 2 licenses for f1,
   label them "first checkout" */
lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, "first checkout");
lc_checkout(lm_job, f1, "1.0", 2,...,LM_DUP_NONE)
```

```
/* Check out 5 more licenses for f1,
   label them "second checkout" */
lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, "second checkout");
lc_checkout(lm_job, f1, "1.0", 5,...,LM_DUP_NONE)

/* Check in 2 licenses for f1 labeled "first checkout " */
lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, "first checkout");
lc_checkin(lm_job, f1, 0);

/* Check in 5 licenses for f1 labeled "second checkout" */
lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, "second checkout");
lc_checkin(lm_job, f1, 0);
```

This makes it possible to have different sets of licenses for a given feature without having to create a separate license job. Each set is labeled with a different checkout-data string, and is tracked separately by the license server. Subsequent check in requests then can qualify the feature with the checkout-data string by setting it with the `LM_A_CHECKOUT_DATA` attribute before calling **lc_checkin**. Each checkout or checkin request uses the value of the checkout-data string from the last call to **lc_set_attr**.

Custom Duplicate Grouping

Duplicate grouping based on `USER`, `HOST`, or `DISPLAY` may not represent the duplicate grouping criteria you need. The checkout-data string can be used to group duplicates via a custom vendor criteria, in addition to the `USER/HOST/DISPLAY` duplicate grouping criteria. Set the checkout-data string using the `LM_A_CHECKOUT_DATA` attribute and then, in the subsequent call to **lc_checkout**, set the `LM_DUP_VENDOR` bit in the duplicate grouping bitmask. For example, if User A's application performs the following checkout request:

```
lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, "dup1");
lc_checkout(lm_job, f1, "1.0", 1,...,LM_DUP_VENDOR)
```

and user B's application makes a similar request:

```
lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, "dup1");
lc_checkout(lm_job, f1, "1.0", 1,...,LM_DUP_VENDOR)
```

then the two checkout requests are considered duplicates, based on the vendor duplicate grouping criteria of `dup1`, and only one license is consumed.

Note • *Duplicate grouping criteria can also be expressed in the license file via the* `DUP_GROUP` *keyword on* `FEATURE/INCREMENT` *lines. See Programming Reference for License File–Based Licensing for further details.*

If a custom vendor duplicate grouping criteria is specified, via either `LM_DUP_VENDOR` *in the call to* **lc_checkout** *or* `DUP_GROUP` *on the* `FEATURE/INCREMENT` *line,* `LM_A_CHECKOUT_DATA` *must set a checkout-data string.*

User Visibility

You have the option in your vendor daemon of allowing the `LM_A_CHECKOUT_DATA` string to be visible or not. The daemon variable *`ls_show_vendor_def`* controls whether the checkout-data string is visible to your end users via `lmstat` (or any utility that calls **lc_userlist**).

If you are considering using this attribute, we recommend that you contact Technical Support for guidance.

See Also
lc_checkout
lc_set_attr
lc_new_job
LM_A_MULTIPLE_CHECKOUT_DATA
ls_show_vendor_def in *Programming Reference for License File–Based Licensing*

LM_A_MULTIPLE_CHECKOUT_DATA

Type

char*

Default

None

The `LM_A_MULTIPLE_CHECKOUT_DATA` attribute is used for bulk custom dup grouping. It allows you to checkout up to 100 custom dup group in a call.

The checkout-data string is a character string, with a maximum size of `MAX_VENDOR_CHECKOUT_DATA` bytes (1024 bytes). The default value is the NULL string. Each unique value of `LM_A_MULTIPLE_CHECKOUT_DATA` represents a unique license group; the NULL string can be one of those unique values. Like all other attributes, set this before the checkout or checkin request; it takes effect for all subsequent calls to **lc_checkout** or **lc_checkin** until it is changed.

Bulk Custom Duplicate Checkout

Duplicate grouping based on `USER`, `HOST`, or `DISPLAY` may not represent the duplicate grouping criteria you need. The checkout-data string can be used to group duplicates via a custom vendor criteria, in addition to the `USER/HOST/DISPLAY` duplicate grouping criteria. Set the checkout-data string using the `LM_A_MULTIPLE_CHECKOUT_DATA` attribute and then, in the subsequent call to **lc_checkout**, set the `LM_DUP_VENDOR` bit in the duplicate grouping bitmask.

The following scenarios explain the consumption of licenses:

Scenario 1

License checkout for more than a user in a single call.

For example, the checkout license can be consumed for user abc and cde in a single call as follows:

lc_set_attr(job1, LM_A_MULTIPLE_CHECKOUT_DATA, "abc" "\f" "cde");

lc_checkout(job1, "f1", "1.0", 1, LM_CO_NOWAIT, &code, LM_DUP_VENDOR);

The licenses consumed are 2 (1 each for user abc and cde).

Note • *The "\f" is the delimiter character to be used with checkout data string supplied to LM_A_MULTIPLE_CHECKOUT_DATA.*

Scenario 2

The use of overlapping checkout data strings results the same whether one job or different jobs are used for the checkout calls.

For example,

lc_set_attr(job1, LM_A_MULTIPLE_CHECKOUT_DATA, "abc" "\f" "cde");

lc_checkout(job1, "f1", "1.0", 1, LM_CO_NOWAIT, &code, LM_DUP_VENDOR);

The licenses consumed are 2 (1 each for user abc and cde)

lc_set_attr(job2, LM_A_MULTIPLE_CHECKOUT_DATA, "fgh" "\f" "abc");

lc_checkout(job2, "f1", "1.0", 1, LM_CO_NOWAIT, &code, LM_DUP_VENDOR);

The license consumed is 1 (1 for user fgh and abc is duped).

Result: The 3 licenses are consumed on license server.

Note • *The "\f" is the delimiter character to be used with checkout data string supplied to LM_A_MULTIPLE_CHECKOUT_DATA.*

Scenario 3

Multiple checkout count with mixed attributes (LM_A_CHECKOUT_DATA and LM_A_MULTIPLE_CHECKOUT_DATA).

For example:

lc_set_attr(job1, LM_A_CHECKOUT_DATA, "abc");

lc_checkout(job1, "f1", "1.0", 2, LM_CO_NOWAIT, &code, LM_DUP_VENDOR);

The 2 licenses are consumed for user abc.

lc_set_attr(job2, LM_A_MULTIPLE_CHECKOUT_DATA, "ijk" "\f" "abc");

lc_checkout(job2, "f1", "1.0", 3, LM_CO_NOWAIT, &code, LM_DUP_VENDOR);

The 2 licenses are duped for user abc, 1 additional license is consumed for user abc, and 3 licenses are consumed for user ijk.

Result: The 6 licenses are consumed on license server.

Note • *The "\f" is the delimiter character to be used with checkout data string supplied to LM_A_MULTIPLE_CHECKOUT_DATA.*

LM_A_CHECKOUTFILTER, LM_A_CHECKOUTFILTER_EX

Type

Pointer to a function returning `int`.

Default

None

Discussion

The function pointer `LM_A_CHECKOUTFILTER` (or the extended version, `LM_A_CHECKOUTFILTER_EX`) is set to point to a checkout filter callback function. This filter function is invoked each time **lc_checkout** finds a `FEATURE/INCREMENT` line that is a candidate for fulfilling the license request. Candidates are selected based on the feature name.

This filter provides the application the opportunity to examine the `FEATURE/INCREMENT` line before **lc_checkout** processes it, and either allows **lc_checkout** to proceed processing the line or rejects this particular line. **lc_checkout** may still reject the line even though the filter function allows processing to proceed.

The `LM_A_CHECKOUTFILTER` routine is called as follows:

```
status = (*myCheckoutFilter)(config);
```

or, with `LM_A_CHECKOUTFILTER_EX`:

```
status = (*myCheckoutFilterEx)(job, config, vendor_data);
```

Checkout Filter Callback Function Parameters

The `LM_A_CHECKOUTFILTER` or the extended version, `LM_A_CHECKOUTFILTER_EX` callback function uses the following parameters:

Table 5-2 • LM_A_CHECKOUTFILTER or LM_A_CHECKOUTFILTER_EX Callback Function Parameters

Callback Parameter	Description
job	`LM_A_CHECKOUTFILTER_EX` only: A pointer to the license job that was passed to the call to **lc_checkout**.
config	A pointer to the `CONFIG` struct representing the candidate feature.
vendor_data	`LM_A_CHECKOUTFILTER_EX` only: A pointer to either vendor-defined data set via the `LM_A_VENDOR_CALLBACK_DATA` attribute or NULL.

Note *• When you use the callback function, disable automatic heartbeat and execute with manual heartbeat to avoid deadlocks.*

Checkout Filter Callback Function Return Values

The return value from this function affects the outcome of the current checkout request in the following ways

- **If 0 is returned**—**lc_checkout** continues to evaluate the `FEATURE` line in an attempt to satisfy the license request.
- **If a non-zero value is returned**—**lc_checkout** does not continue to evaluate the current `FEATURE` line but, instead, takes one of the following actions:
  - Invokes this filter function, again, with a subsequent candidate `FEATURE` line for the license request.

- If there are no more candidate FEATURE lines, **lc_checkout** fails and sets the Flexible API error number to LM_LOCALFILTER; the license is not checked out.

Note • LM_A_CHECKOUTFILTER_EX *provides a way to pass vendor-defined data to this callback function. Behavior is undefined if both* LM_A_CHECKOUTFILTER_EX *and* LM_A_CHECKOUTFILTER *are set.*

See Also
lc_checkout
LM_A_CHECKOUTFILTERLAST_EX
LM_A_VENDOR_CALLBACK_DATA

LM_A_CHECKOUTFILTERLAST_EX

Type

Pointer to a function returning int.

Default

None

Discussion

The function pointer LM_A_CHECKOUTFILTERLAST_EX is set to point to a checkout filter callback function. This filter function is invoked after **lc_checkout** authenticates a FEATURE/INCREMENT line locally as being valid for fulfilling the license request.

This filter provides the application the opportunity to examine the FEATURE/INCREMENT line after **lc_checkout** authenticates it locally, and either allows **lc_checkout** to proceed processing the line or rejects this particular line. If the filter function allows processing to proceed, **lc_checkout** passes a served license request to the license server where it still may get denied. The checkout filter is called on each heartbeat.

The LM_A_CHECKOUTFILTER_EX routine is called as follows:

```
status = (*myCheckoutFilterLastEx)(job, config, vendor_data);
```

Checkout Filter Callback Parameters

The LM_A_CHECKOUTFILTERLAST_EX function uses the following callback parameters:

Table 5-3 • LM_A_CHECKOUTFILTERLAST_EX Callback Parameters

Callback Parameter	Description
job	A pointer to the license job that was passed to the call to **lc_checkout**.
config	A pointer to the CONFIG struct representing the authenticated feature line.
vendor_data	A pointer to either vendor-defined data set via the LM_A_VENDOR_CALLBACK_DATA attribute or NULL.

Return Values

The return value from this function affects the outcome of the current checkout request in the following ways

- **If 0 is returned**—For served licenses, **lc_checkout** passes the license request to the license server for further processing. For unserved licenses, **lc_checkout** checks out the license.
- **If a non-zero value is returned**—**lc_checkout** does not continue to evaluate the current `FEATURE` line but, instead, takes one of the following actions:
  - Invokes this filter function, again, with a subsequent authenticated `FEATURE` line for the license request.
  - If there are no more candidate `FEATURE` lines, **lc_checkout** fails and sets the Flexible API error number to `LM_LOCALFILTER`; the license is not checked out.

See Also
lc_checkout
LM_A_MULTIPLE_CHECKOUT_DATA
LM_A_VENDOR_CALLBACK_DATA

LM_A_CROSS_TZ_OK

Type

`short`

Default

`0` (False)

Set this variable to **1** (True) to enable a 24-hour client-side tolerance to accommodate time-zone differences between the license server and the FlexEnabled client. This tolerance applies to the start date and the expiry date of the license.

Start date example

For example, a license server in Japan contains license rights with a start date valid from January 10 (in Japanese Standard Time). A client located in the US Pacific time zone (where it is still January 9) attempts to check out a license. The server issues a valid license, but the client invalidates it because in the US Pacific time zone the license is not valid yet.

However, in this same scenario, if `LM_A_CROSS_TZ_OK` is set to **1**, the client applies a 24-hour tolerance to the license's start date, validates the license, and checks out the requested feature.

Expiry date example

Assume that a license server is started in a US Pacific time zone. The license rights installed on the server contain multiple `INCREMENT` lines for feature `F1`. Assume that the first `INCREMENT` line expires at midnight in the US Pacific time zone, while the remaining are valid for another six months. However, the date has already changed in other time zones such as in Asia. When the FlexEnabled application is invoked in one of these other time zones, the license server issues a still-valid (not expired) license for `F1` from the first `INCREMENT` line, but the client invalidates the license because it has already expired.

Again, if `LM_A_CROSS_TZ_OK` is set to **1**, the client applies a 24-hour tolerance to the license's expiration date, validates the license, and checks out `F1`.

LM_A_DIAGS_ENABLED

Type

short

Default

1—Diagnostic output enabled

This attribute allows FlexNet licensing to produce some diagnostic output for failures of the **lc_checkout** call if the environment variable FLEXLM_DIAGNOSTICS is set. If LM_A_DIAGS_ENABLED is set to 0, this diagnostic information is unconditionally disabled.

The FLEXLM_DIAGNOSTICS environment variable can be used by your end users to obtain more information if a checkout fails. If FLEXLM_DIAGNOSTICS is set, an **lc_perror** call is made. If FLEXLM_DIAGNOSTICS is set to "2," then in addition to the **lc_perror** call, the arguments to **lc_checkout** (except for the KEY information) are printed to stderr, and also, on Windows, this is logged to flex_err.log.

The diagnostics are enabled by default. It is recommended that this be left enabled. This will allow us to help you debug your end-users' problems with error messages more explicit than, "can't get license." In these situations, we are unable to help. We developed and distributed the FLEXLM_DIAGNOSTICS to enable us (and your support people) to help your end users more effectively.

LM_A_DISABLE_ENV

Type

short

Default

LM_LICENSE_FILE environment variable enabled

If set to a non-zero value, LM_A_DISABLE_ENV will force the Licensing toolkit client functions to disregard the setting of the LM_LICENSE_FILE environment variable. It's rare that there's a legitimate reason to use this, but it does come up with certain utilities that may explicitly need to ignore the LM_LICENSE_FILE environment variable. It is strongly discouraged that this be used in your applications, as many end user sites are familiar with FlexNet licensing, and need to assume that LM_LICENSE_FILE will be effective.

Note • *This must be set before* LM_A_LICENSE_DEFAULT *to be effective.*

LM_A_DISPLAY_OVERRIDE

Type

char*

Default

No override of display name

Note • *This value cannot be changed for a job after the initial connection to the vendor daemon.*

Windows Platforms

This string, if specified, is used to override the display name as derived from the **WTSQuerySessionInformation** call from the Platform SDK Terminal Services API. The most common use of this attribute is for setting the display to the remote host name in a Terminal Server environment.

When querying using **lc_get_attr**, specify a valid `char` buffer as the `value` argument.

UNIX Platforms

This string, if specified, is used to override the display name as derived from the UNIX **ttyname** system call.

The most common use of this attribute is for setting the display to the X-Display name. Unfortunately, the only reliable way of obtaining the name of the X-Display is via a call to an X-based routine. Therefore, this can only be done by the X-based application, after **XOpenDisplay** (or **XtAppInitialize**) has been called.

The Display name is available via the X macro `DisplayString(display)`.

In addition, it is essential to note that there are at least three possible aliases for using the monitor attached to the computer in use: `localhost:0`, `unix:0`, and `:0`. If any of these are used, `LM_A_DISPLAY_OVERRIDE` should use the result of **gethostname** instead. Finally, it may be safest to use the IP address as a string to avoid the problem of aliases for a particular display host.

When querying using **lc_get_attr**, specify a valid `char` buffer as the `value` argument.

LM_A_HOST_OVERRIDE

Type

char*

Default

No override of host name

This string, if specified, will be used to override the host name as derived from the UNIX **gethostname** system call.

When querying using **lc_get_attr**, specify a valid `char` buffer as the `value` argument.

Note • *This value has the following restrictions:*

- *The string cannot contain a colon (:).*
- *The value cannot be changed for a job after the initial connection to the vendor daemon.*

LM_A_IGNORE_CACHED_SERVER_DATA

Type

`int`

Default

False

When the license server is repeatedly restarted, the FlexEnabled application uses stored information about previous connections to the license server. This can result in multiple licenses being checked out for the same client. This eventually leads to a denial of service, when no more licenses are available from the license server. To avoid this error, set LM_A_IGNORE_CACHED_SERVER_DATA to True.

Note • *Use this attribute only when also using a DUP_GROUP specification or when frequent connection interruptions with the license server are occurring.*

LM_A_IGNORE_FILE_CHECK_IN_HEARTBEAT (Linux Only)

Type

`int`

Default

`0`

(Linux platforms only) By default, the FlexEnabled client always checks for the existence of the borrow-cache file (used by FlexNet Publisher's `BORROW` feature). Setting this attribute to **1** forces the client to ignore this check. Best practice is to set this attribute to **1** when the client application is *not* using the `BORROW` feature.

Define this attribute before a checkout request is made.

Important • *Do not set this attribute to* `1` *if the client application is using the* `BORROW` *feature.*

LM_A_INTERNET_OVERRIDE

Type

char*

Default

No override of IP address.

Discussion

This string specifies the IP address with which you want to override the IP address of the client requesting the license for a specific feature. In this way, the client can still check out a license should the licensed feature be bound to an IP address different from the client's. (A licensed feature is bound to an IP address either by `INTERNET` keyword used in the license's `FEATURE` line or through an entry for the licensed feature in the `INCLUDE` line of the options file.)

For example, suppose the license file (or fulfillment record) contains a `FEATURE` line bound to `HOSTID=INTERNET=172.88.22.111`, while the FlexEnabled client has a different IP address. To enable the client to check out this feature, use this attribute to set an IP-address override value to **`172.88.22.111`**. Then, during checkout, the client's IP address is overridden with the same IP address as the one used in the `FEATURE` line, enabling a successful checkout.

This feature is also useful when a proxy client is used to send checkout requests to the license server on behalf of a FlexEnabled client. The override enables the proxy client to check out licensed features successfully.

When querying using **lc_get_attr**, specify a valid `char` buffer as the `value` argument.

Note • *Do not use wild card characters in the value for this attribute.*

If the value entered for this attribute is not a valid IP address, the error -201 (`Invalid IP address used while overriding`) is reported.

Support for IPV4 and IPV6

This attribute supports IP addresses in either IPv4 or IPv6 format.

For example, in the IPv4 format, the attribute syntax looks similar to this:

```
(void)lc_set_attr(lm_job,LM_A_INTERNET_OVERRIDE,(LM_A_VAL_TYPE)”172.84.24.109”)
```

In the IPv6 format, the attribute syntax looks similar to this:

```
(void)lc_set_attr(lm_job,LM_A_INTERNET_OVERRIDE,(LM_A_VAL_TYPE)”
fec0:dead:7:0:20f:feff:fe7f:e866”)
```

API for Java-Based Applications

To support IP-address-override functionality in a Java-based application, add the following API to the application code:

```
lic.setipaddress("<IP_address>");
```

where **`<IP address>`** is the value present in the `FEATURE` line or options file.

Note • *Add this API to the sample application,* `BasicExample.java`*, which ships with the Licensing toolkit for Java, if you want to test the IP-address-override functionality with the sample.*

LM_A_LICENSE_CASE_SENSITIVE

Type

`int`

Default

False

This attribute is used in the context of license generators and FlexEnabled applications. If `True`, text fields in license file `FEATURE/INCREMENT/PACKAGE/UPGRADE` lines are case sensitive. To enable case sensitivity, set this attribute both in the FlexEnabled application (before the call to **lc_checkout**) and the license generator (before the call to **lc_cryptstr**). If licenses are served by the FlexNet license server, then set the corresponding vendor daemon variable, *`ls_a_license_case_sensitive`*; otherwise, the served licenses do not authenticate. LM_A_LICENSE_CASE_SENSITIVE attribute has no impact while checking out a borrowed license from the system registry as this flag is applicable only during check-out from license server. Check-out from borrow registry is always case-sensitive.

For compatibility with Licensing toolkit versions 5.1 or earlier, set this attribute to `True`. This attribute is automatically set to `True` if the `LM_A_LICENSE_FMT_VER` attribute is set to `LM_BEHAVIOR_V5_1` or less.

See Also
`ls_a_license_case_sensitive` in *Programming Reference for License File–Based Licensing*
`ls_hud_hosid_case_sensitive` in *Programming Reference for License File–Based Licensing*

LM_A_LKEY_CASE_SENSITIVE

Type

`int`

Default

False

This attribute is used in the context of license key generation; license signature (SIGN=) generation is not affected. Set this attribute in the license generator, before the call to **lc_cryptstr**.

If `True`, text fields in license file `FEATURE/INCREMENT/PACKAGE/UPGRADE` lines are treated as case sensitive for the purpose of generating license keys. This attribute is provided in order to maintain compatibility with FlexEnabled applications using Licensing toolkit, versions 5.1 or earlier, which treated these fields as case sensitive by default.

LM_A_MT_HEARTBEAT (UNIX Only)

Type

`int`

Default

True

This flag applies to UNIX platforms only. If `True`, the automatic heartbeat mechanism is controlled via a dedicated thread in the FlexEnabled application. The platform must support pthreads in order to implement a dedicated thread for heartbeat messages.

If `False`, automatic heartbeats are controlled in the application's main thread using the SIGALRM signal.

See Also
Information about Heartbeats in *Programming Reference for License File–Based Licensing*

LM_A_PERIODIC_CALL

Type

Pointer to a function returning `int`. Return value not used.

Default

No periodic call

This function, if specified, will be called each `LM_A_PERIODIC_COUNT` times that **lc_heartbeat** is called. **lc_heartbeat** is called directly or automatically depending on the value of `LM_A_CHECK_INTERVAL`.

See Also
LM_A_CHECK_INTERVAL
lc_heartbeat

LM_A_PERIODIC_COUNT

Type

`int`

Default

0 (no PERIODIC_CALL)

This is the count of how many times **lc_heartbeat** must be called before the function specified by `LM_A_PERIODIC_CALL` is called. **lc_heartbeat** is called directly or automatically depending on the value of `LM_A_CHECK_INTERVAL`.

See Also
LM_A_CHECK_INTERVAL
lc_heartbeat

LM_A_PLATFORM_OVERRIDE

Type

char*

Default

No override of the FlexNet licensing platform name

This string, if specified, overrides the current platform name. The name can be either a FlexNet licensing platform name or a vendor-defined name. The *Release Notes* contain the currently supported platforms and their associated Licensing platform names.

It accepts the following new platform values: i86_lsb and x64_lsb.

When querying using **lc_get_attr**, specify a valid `char` buffer as the `value` argument.

LM_A_RETRY_CHECKOUT

Type

`int`

Default

True

When True, checkouts that fail due to communications errors are automatically retried once. Sometimes this second attempt will succeed on networks with poor communications, but this makes failure take twice as long. This is default behavior in the Flexible API.

LM_A_TZ_OVERRIDE

Note • *Use this attribute in your test environment only.*

Platform-independent attribute used to override the client time zone and time. This attribute is used only for testing purposes—it can be used to test time zone licensing. `LM_A_TZ_OVERRIDE` is most useful for testing scenarios that simulate a license client running in various time zones.

Type

char*

Default

No override of client time zone or time.

This string overrides the current client time zone and time with the specified time zone value and time. This attribute is used in conjunction with the `TZ` keyword on `FEATURE`/`INCREMENT` lines. Valid values include hour, half-hour (.30), and quarter-hour (.45) time zones, which are specified relative to Greenwich Mean Time (GMT).

When querying using **lc_get_attr**, specify a valid `char` buffer as the `value` argument.

Syntax and Examples

```
“<[+-]hh<.30|.45><;hh:mm>>”
```

You can specify a value for time zone or a value for both time zone and time.

Table 5-4 •

Example	Syntax	Behavior
Override client time zone	"+05.30"	Client time zone would be overridden with +05.30 Client time is not overridden; the current client machine time would be taken into account.
Override client time zone and time	"+05.30;20:33"	Client time zone would be overridden with +05.30 Client time would be overridden with 8:33 P.M.

For example, to specify that the client time is 1:45 P.M. in India (GMT+05:30), use the following:

```
lc_set_attr(lm_job, LM_A_TZ_OVERRIDE, "+05.30;13:45");
```

For example, to specify only that the client time zone is GMT–08:00, use the following:

```
lc_set_attr(lm_job, LM_A_TZ_OVERRIDE, "-08");
```

Important • *You cannot specify a value for time by itself. To specify a time override, you must also specify a time zone override value. The error code `LM_TZ_INVALID_TZONE_SPEC` is returned if only a value for time is specified.*

LM_A_HEARTBEAT_LOCK (Windows only)

Type

short

Default

1

When this attribute is set to 1 or greater, the checkout and heartbeat is synchronized by a mutex lock. This is applicable to automatic heartbeat only. The client attribute LM_A_HEARTBEAT_LOCK can be set to 0, to disable the mutex lock.

See Also
lc_checkout

LM_A_USER_OVERRIDE

Type

char*

Default

No override of user name

This string, if specified, overrides the user name as derived from the UNIX password file. On Windows, the user name is set to the host name, but can be overridden with this attribute.

When querying using **lc_get_attr**, specify a valid `char` buffer as the `value` argument.

Note • *Do not include a colon (:) in this string.*

LM_A_VENDOR_CALLBACK_DATA

Type

`void *`

Default

`NULL`

This attribute allows you to set a pointer to vendor-defined data. The pointer is of type `void` regardless of the data type. This pointer is passed, as the last argument, to the following callback functions, thus making this data available to those functions.

- LM_A_CHECKOUTFILTER_EX
- LM_A_USER_EXITCALL_EX
- LM_A_USER_RECONNECT_EX
- LM_A_USER_RECONNECT_DONE_EX

LM_A_VSEND_NOWAIT

Type

`short`

Default

`NULL`

If this attribute is set to 1, it requires ls_vendor_msg_async sets to 1. However, setting ls_vendor_msg callback to 1 does not require LM_A_VSEND_NOWAIT to be set.

Setting to 1 allows polling for response from lc_vsend API. If not set, lc_vsend blocks until a response is received. Example usage can be found in examples/lc_vsend/lmvsend.c.

6

Activation Borrow API

The Activation Borrow API is a simple set of functions that enable a FlexEnabled application to request a temporary license from license rights held in trusted storage on a license server. It is a more secure replacement for the license file based BORROW functionality. For information about using the Activation Borrow API, see the *Programming Reference for Trusted Storage–Based Licensing*.

The functions within the Activation API can be used in conjunction with, or to replace, functions within the Activation Borrow API.

Activation Borrow Functions

The following table lists the activation borrow functions that are defined in `FlxActBorrow.h`.

Table 6-1 • Activation Borrow API Functions

Function	Description
flxActBorrowActivate	Uses activation to 'borrow' a license from the specified license server.
flxActBorrowReturn	Returns a borrowed license.
flxActBorrowTSViewCreate	Creates the object required to view the contents of trusted storage.
flxActBorrowTSViewFRAttributeGet	Gets the value of a specific attribute from a fulfillment record.
flxActBorrowTSViewDelete	Deletes the object used to view the contents of trusted storage.

flxActBorrowActivate

Uses activation to *borrow* a license from the specified license server.

Syntax

```
int flxActBorrowActivate(
    const char *pszEntitlementId,
    const char *pszProductId,
    const char *pszExpiration,
    const char *pszServer,
    char *pszFulfillmentId,
    int  length,
    FlxActError *pError);
```

Parameters

The **flxActBorrowActivate** function uses the following parameters:

Table 6-2 • flxActBorrowActivate Parameters

Parameter	Description
pszEntitlementId	A pointer to the variable storing the value of the entitlement ID to be included in the activation request. This parameter can be NULL if the product ID is specified.
pszProductId	A pointer to the variable storing the value of the product ID which will be included in the activation request. This parameter can be NULL if the entitlement ID is specified.
pszExpiration	A pointer to the variable storing the requested expiration date. The format is dd-mmm-yyyy.
pszServer	A pointer to the variable storing the address of the license server from which the license is requested. The following formats can be used for this parameter: ● port@host ● @host
pszFulfillmentId	Buffer for the ID of the fulfillment record added to trusted storage as a result of this function call. If the fulfillment ID is longer than the supplied buffer then it will be truncated and null terminated.
length	The length of the buffer supplied for the fulfillment ID.
pError	Resulting error code, if any. If this function returns LM_ACTIVATION then check pError for details of the error. For an explanation of the contents of pError see FlxActError in the Activation API chapter.

Return Values

The following values are returned by **flxActBorrowActivate**

Table 6-3 • flxActBorrowActivate Return Values

Return Value	Description
LM_NOERROR	Success.
LM_ACTIVATION	Failure due to activation failure. Further information on the error is returned in pError.
LM_BADPARAM	Failure due to invalid function parameters.

This function will fail when there are no suitable licenses in trusted storage on the license server. The following are possible cases where a failure will occur:

- There are no fulfillment records with the requested entitlement ID.
- There are no fulfillment records with the requested product ID.
- There are no fulfillment records with both the requested entitlement and product IDs.
- There are matching fulfillment records, but they have expiration dates that are prior to the requested expiration date.
- There are matching fulfillment records, but none of them have available hybrid licenses. This may be because the hybrid license count is zero. It may also be because the hybrid license count is positive, but all hybrid licenses are in use as concurrent licenses.

Note • *When using activation to replace BORROW, there should be no matching fulfillment records that have a positive activatable count.*

Discussion

This function can only run if the activation library is accessible within the product package. The activation library provides access to trusted storage.

Before calling this function the activation library must have been initialized successfully—when using this function from within a FlexEnabled application, the Flexible API function, lc_flexinit, must have completed successfully before this function is called.

flxActBorrowActivate creates an activation request using the following input parameters:

- **Entitlement ID**—used to identify the fulfillment record that contains the feature licenses required.
- **Product ID**—used to identify the fulfillment record that contains the feature licenses required.
- **Expiration date**—specifies the required expiration date for the license.

The activation request is sent to the specified license server. The license server searches its trusted storage for a fulfillment record that contains available licenses that match the activation request.

When a **product ID** is specified but no entitlement ID is specified, then this search is based on the product ID. The result of this search is a *product license.* When fulfillment records with this product ID contain all the feature licenses required for the FlexEnabled application then the fulfillment record constitutes a product license.

When an **entitlement ID** is specified, but no product ID is specified, then this search is based on the entitlement ID.

When both a **product ID** and an **entitlement ID** are specified, then only fulfillment records that match both values are considered.

The license server looks first for available activatable licenses and if none are available then looks for hybrid licenses. The license server functionality cannot be altered by this function: to ensure that only hybrid licenses are used, trusted storage on the license server must contain no matching activatable licenses.

When an available matching license is found, the license server creates a fulfillment record with the requested expiration date and returns it to the client. This function automatically creates trusted storage on the client machine if necessary.

The fulfillment ID of the fulfillment record provided by the license server is placed in the supplied buffer. This value should be stored so that it can be used as an input parameter to flxActBorrowReturn for the return of this borrowed license to the license server.

Fulfillment records added to trusted storage using this function can be managed either using activation borrow functions or using the Activation API. They can be returned or viewed using the Activation Borrow API or Activation API. To repair the fulfillment records, the Activation API must be used.

flxActBorrowReturn

Returns a borrowed license before it has expired.

Syntax

```
int flxActBorrowReturn(
    const char *pszFulfillmentId,
    const char *pszServer,
    FlxActError *pError);
```

Parameters

The **flxActBorrowReturn** function uses the following parameters:

Table 6-4 • flxActBorrowReturn Parameters

Parameter	Description
pszFulfillmentId	A pointer to the variable storing the fulfillment ID of the fulfillment record that is to be returned. This fulfillment ID is returned in the `pszFulfillmentId` buffer when the license is borrowed using flxActBorrowActivate.
pszServer	A pointer to the variable storing the address of the license server to which the fulfillment record is to be returned. The following formats can be used for this parameter: • port@host • @host

Table 6-4 • flxActBorrowReturn Parameters

Parameter	Description
pError	A pointer to the variable used to store a resulting error code, if any. If this function returns LM_ACTIVATION, then check the variable define by pError for details of the error. For an explanation of the contents of pError, see FlxActError in the Activation API chapter.

Return Values

The following values are returned by **flxActBorrowReturn**

Table 6-5 • flxActBorrowReturn Return Values

Return Value	Description
LM_NOERROR	Success.
LM_ACTIVATION	Failure due to activation failure. Further information on the error is returned in pError.
LM_BADPARAM	Failure due to invalid function parameters.

Discussion

This function can only run if the activation library is accessible within the product package. The activation library provides access to trusted storage.

Before calling this function the activation library must have been initialized successfully. When using this function from within a FlexEnabled application, the Flexible API function, lc_flexinit, must have completed successfully before this function is called.

flxActBorrowReturn creates a return request that specifies the fulfillment record to be returned using the fulfillment ID. This fulfillment ID was obtained when the license was borrowed using flxActBorrowActivate. You can also retrieve the fulfillment record using the TSView functions to examine each fulfillment record in trusted storage in turn until the desired fulfillment record is located and its fulfillment ID ascertained (see flxActBorrowTSViewCreate for details).

The return request is sent to the specified license server. The fulfillment record is only accepted by a license server if it originally issued the fulfillment record.

flxActBorrowTSViewCreate

Creates the object required to view the contents of trusted storage.

Syntax

```
int flxActBorrowTSViewCreate(
    FlxActBorrowContext *pContext,
    int *pFulfillmentCount);
```

Parameters

The **flxActBorrowTSViewCreate** function uses the following parameters:

Table 6-6 • flxActBorrowTSViewCreate Parameters

Parameter	Description
pContext	A pointer to the variable where the context object, created by this function, will be stored. This object is used by flxActBorrowTSViewFRAttributeGet.
pFulfillmentCount	A pointer to a variable storing the number of fulfillment records currently in trusted storage.

Return Values

The following values are returned by **flxActBorrowTSViewCreate**

Table 6-7 • flxActBorrowTSViewCreate Return Values

Return Value	Description
LM_NOERROR	Success.
LM_ACTIVATION	Failure due to activation failure. No recovery possible.
LM_BADPARAM	Failure due to invalid function parameters.

Discussion

This function can only run if the activation library is accessible within the product package. The activation library provides access to trusted storage.

Before calling this function, the activation library must have been initialized successfully. When using this function from within a FlexEnabled application, the Flexible API function, lc_flexinit, must have completed successfully before this function is called. When using this function from within a stand alone utility, it is recommended that the utility uses the Activation API initialization function flxActCommonLibraryInit.

This function creates the object that is required to view the contents of trusted storage. This object, `pContext`, stores all fulfillment records found. You can then use flxActBorrowTSViewFRAttributeGet to loop through the contents of the object and get a specific fulfillment record. The number of fulfillment records in trusted storage, `pFulfillmentCount`, is used to control the loop.

Note that this function provides access to **all** the contents of trusted storage: If fulfillment records have been added to trusted storage using the Activation API functions, these fulfillment records will also be available.

Use flxActBorrowTSViewDelete to delete the object and free all the associated resources.

flxActBorrowTSViewFRAttributeGet

Gets the value of a specific attribute from a fulfillment record. This function can be used to loop through multiple records in the context object to find the fulfillment record of interest.

Syntax

```
int flxActBorrowTSViewFRAttributeGet(
    FlxActBorrowContext  context,
    int  index,
    FlxActBorrowStatusAttribute  attribute,
    char *pszBuffer,
    int *pLen);
```

Parameters

The **flxActBorrowTSViewFRAttributeGet** function uses the following parameters:

Table 6-8 • flxActBorrowTSViewFRAttributeGetParameters

Parameter	Description
context	Object used to hold fulfillment records. The object was previously created using flxActBorrowTSViewCreate.
index	The position of the specified fulfillment record where index is in the range 0 through FulfillmentCount-1, with 0 being the first fulfillment record. FulfillmentCount is returned by flxActBorrowTSViewCreate.
attribute	Specifies the item to be extracted from the fulfillment record and returned in the buffer, pszBuffer. See FlxActBorrowStatusAttribute for valid values.
pszBuffer	A pointer to the variable where the 'attribute' value, requested from the fulfillment record, will be stored.
pLen	A pointer to the variable that specifies the character length of pszBuffer. Set this value to zero to obtain the length of buffer required for the requested item.

Return Values

The following values are returned by **flxActBorrowTSViewFRAttributeGet**

Table 6-9 • flxActBorrowTSViewFRAttributeGet Return Values

Return Value	Description
LM_NOERROR	Success.
LM_BADPARAM	Failure due to invalid function parameters.

Discussion

Before calling this function, call flxActBorrowTSViewCreate to create the required context object and ascertain the number of fulfillment records in trusted storage.

Use this function in a loop to examine each fulfillment record in turn and extract the required items from it. This method can be used to:

- Search for a fulfillment record with a specific attribute—may be required to identify which fulfillment record to return.
- Obtain and display information about fulfillment records in trusted storage.

When access to fulfillment records is no longer required, release the context object and its associated resources by calling flxActBorrowTSViewDelete.

FlxActBorrowStatusAttribute

This enumerated type defines which attribute is extracted from a fulfillment record.

Table 6-10 • FlxActBorrowStatusAttribute

Type	Description
kActBorrowStatusAttributeUnknown	Not used.
kActBorrowStatusAttributeEntitlementId	Entitlement ID—an identity that defines what license rights the user is entitled to and used when requesting license rights from a publisher server.
kActBorrowStatusAttributeProductId	Product ID—identifies the product that uses the license rights in the fulfillment record.
kActBorrowStatusAttributeFulfillmentId	Fulfilllment ID—a unique identity for this fulfillment record. Used to specify the fulfillment record to be returned.
kActBorrowStatusAttributeExpiration	Expiration date—the expiration date for this fulfillment record.

flxActBorrowTSViewDelete

Deletes the context object used to view the contents of trusted storage.

Syntax

```
void flxActBorrowTSViewDelete(FlxActBorrowContext context);
```

Parameters

The **flxActBorrowTSViewDelete** function uses the following parameters:

Table 6-11 • flxActBorrowTSViewDelete Parameters

Parameter	Description
context	Object used to hold fulfillment records. The object was previously created using flxActBorrowTSViewCreate.

Return Values

There is no return value.

Discussion

This function should be paired with a matching successful call to flxActBorrowTSViewCreate. All resources associated with the object, including the fulfillment record it may contain, are freed.

7

FlexNet Licensing Service API

The FlexNet Licensing Service is an administrative proxy that runs on behalf of the user to perform system tasks where elevated privilege is required. It should be installed to enable the FlexEnabled application, the activation utilities (for trusted storage–based licensing), and the vendor daemon to operate correctly.

For trusted storage–based licensing, the FlexNet Licensing Service is mandatory.

In addition, the FlexNet Licensing Service is required to enable the FlexEnabled application and the vendor daemon to access virtualization data (on Windows and Linux platforms only). The FlexNet Licensing Service regularly polls the system to update virtualization data. Therefore, the FlexNet Licensing Service runs continuously. The virtualization poll interval is not configurable.

The FlexNet Licensing Service is optional for certificate-only applications that do not use virtualization features.

Installing the FlexNet Licensing Service on Windows

On Windows platforms, the FlexNet Licensing Service API is used to install and uninstall the service and register that an application requires the service. It is strongly recommended that the API is used within your product installer. The Windows operating system requires that a user have administrator privileges to install and uninstall services. Additionally, the installer must be run with elevated privileges.

Two types of functions are provided:

- **MSI-compatible Installer**—Use **fnpActSvcInstallForMSI** and **fnpActSvcUninstallForMSI**.
- **C/C++ Applications**—Use **fnpActSvcInstallWin** and **fnpActSvcUninstallWin** within an application.

There are example files included in the licensing toolkit that demonstrate the use of these functions.

The recommended method of installing the FlexNet Licensing Service on Windows is from the MSI-compatible installer used to install the FlexEnabled application.

Task ***To install/uninstall the FlexNet Licensing Service using an MSI-based installer***

1. Use **fnpActSvcInstallForMSI** in the MSI to install the FlexNet Licensing Service and **fnpActSvcUninstallForMSI** to uninstall the service.
2. Set the required execution level to Administrator in the MSI.
3. Package the MSI as an executable.

   This is required because Windows will not allow an MSI to call out to a DLL (in this case `FNP_Act_Installer.dll`) that executes code requiring Administrative privileges.
4. Sign the MSI executable package.

   Windows requires executables that call out to DLLs that execute administrative-privilege code to be signed. Note that InstallShield 12 or later has settings in the Release Wizard that enable you to sign the generated installer executable.

Task ***To install/uninstall the FlexNet Licensing Service using an application or C-based installer***

1. Use **fnpActSvcInstallWin**, **fnpActSvcUninstallWin**, and **fnpActSvcGetLastErrorWin** in your application.
2. Run the application/installer with elevated privileges.

   This requires the user to log on with administrator privilege and then use the 'run-as' option set to administrator. Alternatively, signing the executable with an Authenticode license file will ensure that elevated privileges are used.

Installing the FlexNet Licensing Service on Linux

On Linux, the FlexNet Licensing Service comprises the FlexNet Licensing Service executable and the FlexNet Licensing Service daemon. The daemon is launched by the FlexNet Licensing Service executable.

The following command-line options are available when running `install_fnp.sh`.

Table 7-1 • `install_fnp.sh` options

Option	Description
`--cert`	Restricts the installation to only those components that are required for license file–based licensing.
`--nolsb`	Creates a symlink to the native loader to enable the FlexNet Publisher binaries to run without full lsb support on the target platform.
`--nodaemon`	Prevents the installation script from automatically starting the FlexNet Licensing Service daemon. This can be useful when using systemd.

Task ***To install the FlexNet Licensing Service***

1. Depending on your application, do one of the following:
   - Trusted storage applications: Run `install_fnp.sh`.
   - Certificate applications: Run `install_fnp.sh --cert`.

   The `install_fnp.sh` shell script must be run with root privilege. The script configures the FlexNet Licensing Service executable to run as a root-privilege setuid process.

2. The FlexNet Licensing Service daemon needs to run continuously. Therefore, a further installation step is required to ensure it is started a boot time. Because the FlexNet Licensing Service daemon does not need to run with root privilege, it can be started by adding the following line to a nominated user's crontab:

   ```
   @reboot /usr/local/share/FNP/service64/<version>/FNPLicensingService -r 2>&1 >/tmp/fnpd.log
   ```

Notes

- For the i86_lsb FlexNet Licensing Service daemon, the path is `/usr/local/share/FNP/service/<version>/FNPLicensingService`.
- The machine's administrator is free to redirect output as required.
- The machine's administrator may want to add an additional line to the crontab to run the above command at regular intervals to ensure that if the FlexNet Licensing Service daemon gets stopped for some unexpected reason, it will be restarted by the *crond*. The FlexNet Licensing Service daemon itself ensures there is never more than one instance running.
- Typical messages output on running the startup command are `Licensing Service daemon activated` and `Licensing Service daemon already active`.
- Rather than using cron, administrators may want to add a boot-script for the FlexNet Licensing Service daemon or optionally use systemd. The following commands are required:

  To start the FlexNet Licensing Service daemon

  ```
  /usr/local/share/FNP/service64/<version>/FNPLicensingService -r
  ```

  Note • *The process ID of the daemon will be written to `/var/run/FNPLicensingService64.pid` (`/var/run/FNPLicensingService.pid` on 32-bit).*

  To stop the FlexNet Licensing Service daemon

  ```
  /usr/local/share/FNP/service64/<version>/FNPLicensingService -k
  ```

- When the Linux FlexNet Licensing Service daemon is not installed and an operation occurs that requires it, a descriptive error is displayed. For example:

  ```
  ./lmhostid -ptype VM -uuid
  ```

  results in

  ```
  lmhostid: The VM Host ID is not available. (-215,14704) - The FlexNet Licensing Service does not
  appear to be running.
  ```

Installing the FlexNet Licensing Service on OS X

On OS X, a shell script is used to configure the FlexNet Licensing Service executable to run as a root-privilege setuid process.

Task ***To install the FlexNet Licensing Service***

Run `install_fnp.sh`.

FlexNet Licensing Service Functions

The following table lists the FlexNet Licensing Service functions that are defined in `wininstaller.h` and are available to C/C++ applications. Applications that use the FlexNet Licensing Service API functions must include `wininstaller.h` and link to `FNP_Act_Installer.lib`. They use the installer DLL, `FNP_Act_Installer.dll`, and this must be in the same directory as the application that calls these functions.

Table 7-2 • FlexNet Licensing Service API Functions

Function	Description
fnpActSvcInstallWin	Installs the FlexNet Licensing Service when it is not already installed and registers the application as using the service. If the FlexNet Licensing Service is already installed, this function registers each application that requires the service.
fnpActSvcUninstallWin	Removes the FlexEnabled application from the register of applications requiring the FlexNet Licensing Service. If no other applications are registered, this function uninstalls the service.
fnpActSvcGetLastErrorWin	Returns additional information about the last function called.

The following table lists the functions that are available to MSI-compatible installers only. These functions are in `FNP_Act_Installer.lib` but not in the `wininstaller.h` file. They are not available to C/C++ applications. They use the installer DLL, `FNP_Act_Installer.dll`, and this must be included in the installation package.

Table 7-3 • FlexNet Licensing Service API Functions for MSI-compatible Installers

Function	Description
fnpActSvcInstallForMSI	For MSI-compatible installers to install the FlexNet Licensing Service.
fnpActSvcUninstallForMSI	For MSI-compatible installers to uninstall the FlexNet Licensing Service.

fnpActSvcInstallWin

Installs the FlexNet Licensing Service and registers that the application requiring the service.

Syntax

```
FNPINS_BOOL fnpActSvcInstallWin(
    const char *publisher,
    const char *product);
```

Parameters

The **fnpActSvcInstallWin** function uses the following parameters:

Table 7-4 • fnpActSvcInstallWin Parameters

Parameter	Description
publisher	The name you assign to yourself as a publisher. This string can be anything meaningful to you and does not need to match your vendor name as registered with Flexera. This string cannot contain the comma character.
product	The name of the application. This name is used to identify that the application is using the FlexNet Licensing Service. You should ensure that this value and the publisher name uniquely identify the application. This string can be anything meaningful to you. This string cannot contain the comma character.

Return Values

The following values are returned by **fnpActSvcInstallWin**

Table 7-5 • fnpActSvcInstallWin Return Values

Return Value	Description
TRUE	Success. Call fnpActSvcGetLastErrorWin to obtain further information.
FALSE	Failure. Call fnpActSvcGetLastErrorWin to obtain further information

The following values are returned by **fnpActSvcGetLastErrorWin** when it is returning additional information about a call to **fnpActSvcInstallWin** that was successful:

- `FNP_INSTALL_NOEXTENDEDINFO`
- `FNP_INSTALL_UPGRADED`
- `FNP_INSTALL_NOUPGRADEREQ`
- `FNP_INSTALL_INSTALLED`
- `FNP_INSTALL_ACEPTABLEINSTALLED`

The following values are returned by **fnpActSvcGetLastErrorWin** when it is returning additional information about a call to **fnpActSvcInstallWin** that failed:

- `FNP_INSTALL_REBOOTREQ`
- `FNP_INSTALL_INUSE`
- `FNP_INSTALL_CANNOTCOPYFILE`
- `FNP_INSTALL_CANNOTSTART`
- `FNP_INSTALL_CANNOTSTOP`
- `FNP_INSTALL_NORIGHTS`
- `FNP_INSTALL_INVALIDPATH`
- `FNP_INSTALL_UNSUPPORTEDOS`
- `FNP_INSTALL_EARLYREBOOT`

For an explanation of all values returned by **fnpActSvcGetLastErrorWin,** see Return Values.

Discussion

This function can run only if the FlexNet Licensing Service installation component, `FNP_Act_Installer.dll`, is in the same directory.

The function determines whether the FlexNet Licensing Service is installed and attempts to install it if it is not. If the service is installed, the function updates it (if required) and attempts to start it (if it is not running). It then registers the current FlexEnabled application as requiring the FlexNet Licensing Service. This registration ensures that the FlexNet Licensing Service is removed from the machine only when no FlexEnabled applications require it.

fnpActSvcUninstallWin

Uninstalls the FlexNet Licensing Service if no applications require it.

Syntax

```
FNPINS_BOOL fnpActSvcUninstallWin(
    const char *publisher,
    const char *product);
```

Parameters

The **fnpActSvcUninstalWin** function uses the following parameters:

Table 7-6 • fnpActSvcUninstallWin Parameters

Parameter	Description
publisher	The name you assign to yourself as a publisher. This string can be anything meaningful to you and does not need to match your vendor name as registered with Flexera. This string cannot contain the comma character and must match the value used when the FlexNet Licensing Service was installed for this product.

Table 7-6 • fnpActSvcUninstallWin Parameters

Parameter	Description
product	The name of the application. This name is used to identify that the application that was using the FlexNet Licensing Service. You should ensure that this value and the publisher name uniquely identify the application. This string can be anything meaningful to you. This string cannot contain the comma character. Must match the value used when the FlexNet Licensing Service was installed for this product.

Return Values

The following values are returned by **fnpAcSvcUninstallWin**

Table 7-7 • fnpActSvcUninstallWin Return Values

Return Value	Description
TRUE	Success. Call fnpActSvcGetLastErrorWin to obtain further information.
FALSE	Failure. Call fnpActSvcGetLastErrorWin to obtain further information

The following values are returned by **fnpActSvcGetLastErrorWin** when it is returning additional information about a call to **fnpActSvcUninstallWin** that was successful:

- `FNP_INSTALL_NOEXTENDEDINFO`
- `FNP_INSTALL_UNINSTALLED`

The following values are returned by **fnpActSvcGetLastErrorWin** when it is returning additional information about a call to **fnpActSvcInstallWin** that failed:

- `FNP_INSTALL_REBOOTREQ`
- `FNP_INSTALL_INUSE`
- `FNP_INSTALL_CANNOTSTOP`
- `FNP_INSTALL_NORIGHTS`
- `FNP_INSTALL_INVALIDPATH`
- `FNP_INSTALL_UNSUPPORTEDOS`
- `FNP_INSTALL_EARLYREBOOT`

For an explanation of all values returned by **fnpActSvcGetLastErrorWin,** see Return Values.

Discussion

This function can only run if the FlexNet Licensing Service installation component, `FNP_Act_Installer.dll`, is in the same directory.

The function removes the specified application from the registry of applications that require the FlexNet Licensing Service. If there are no other applications registered as using the FlexNet Licensing Service, the function removes the service from the machine.

fnpActSvcGetLastErrorWin

Returns additional information about the last function called.

Syntax

```
unsigned int fnpActSvcGetLastErrorWin(void);
```

Parameters

The **fnpActSvcGetLastErrorWin** function has no parameters.

Return Values

The following values are returned by **fnpActSvcGetLastErrorWin** when reporting on a successful function call**:**

Table 7-8 • fnpActSvcGetLastErrorWin Return Values for Successful Function Calls

Return Value	Description
0	FNP_INSTALL_NOEXTENDEDINFO There is no additional information. If the **fnpActSvcUninstallWin** returns TRUE and **fnpActSvcGetLastErrorWin** returns this value, it indicates that the FlexNet Licensing Service is required by other applications and has not been uninstalled.
2001	FNP_INSTALL_UPGRADED The existing FlexNet Licensing Service driver was upgraded.
2002	FNP_INSTALL_NOUPGRADEREQ: The existing FlexNet Licensing Service driver was not upgraded.
2003	FNP_INSTALL_INSTALLED The FlexNet Licensing Service was installed on the machine.
2004	FNP_INSTALL_UNINSTALLED The FlexNet Licensing Service driver was not required by other applications and was successfully removed.
2006	FNP_INSTALL_ACCEPTABLEINSTALLED The driver or FlexNet Licensing Service is in use and cannot be updated until a reboot, but an acceptable version is installed.
2007	FNP_INSTALL_STILLREFERENCED The FlexNet Licensing Service is still being referenced by another application and therefore was not removed.

The following values are returned by **fnpActSvcGetLastErrorWin** when it is returning additional information about a call that failed.

Table 7-9 • fnpActSvcGetLastErrorWin Return Values for Failed Function Calls

Return Value	Description
1001	FNP_INSTALL_REBOOTREQ A system reboot is required to complete the install or uninstall.
1002	FNP_INSTALL_INUSE The FlexNet Licensing Service driver is in use. Request the end-user to shut down all other applications. Repeat the call that failed. If this results in another failure, request the end-user to reboot the system.
1003	FNP_INSTALL_CANNOTCOPYFILE Failed to copy the FlexNet Licensing Service driver to the system.
1005	FNP_INSTALL_CANNOTSTART Failed to start the FlexNet Licensing Service or driver.
1006	FNP_INSTALL_CANNOTSTOP Failed to stop the FlexNet Licensing Service. Request the end-user to shut down all other applications. Repeat the call that failed. If this results in another failure, request the end-user to reboot the system.
1007	FNP_INSTALL_NORIGHTS Administrator privileges are required to install the FlexNet Licensing Service.
1008	FNP_INSTALL_INVALIDPATH Invalid path to source or destination.
1010	FNP_INSTALL_EARLYREBOOT Reboot of system required to recover from a previous failed request. Request the end-user to reboot the system and then repeat the call that failed.
1100	FNP_INSTALL_UNSUPPORTEDOS The operating system version on the machine is not supported.

Discussion

This function can only run if the FlexNet Licensing Service installation component, `FNP_Act_Installer.dll`, is in the same directory.

The function returns additional information on the operation of the last FlexNet Licensing Service API function called. Additional information is available both when the call succeeded and when it failed so this function does not, as it name implies, return only errors.

fnpActSvcInstallForMSI

This function wraps **fnpActSvcInstallWin** and allows you to install the FlexNet Licensing Service from an MSI-compatible installer. This is not available to C/C++ applications.

Syntax

```
unsigned int fnpActSvcInstallForMSI(MSIHANDLE hInstall);
```

Parameters

The **fnpActSvcInstallForMSI** function uses the following parameters:

Table 7-10 • fnpActSvcInstallForMSI Parameters

Parameter	Description
hInstall	This is the MSIHANDLE for the running installation.

Return Values

The following values are returned by **fnpActSvcInstallForMSI.**

Table 7-11 • fnpActSvcInstallForMSI Return Values

Return Value	Description
0	Indicates that the transaction was successful.
1002	FNP_INSTALL_INUSE The FlexNet Licensing Service driver is in use. Request the end-user to shut down all other applications. Repeat the call that failed. If this results in another failure, request the end-user to reboot the system.
1003	FNP_INSTALL_CANNOTCOPYFILE Failed to copy the FlexNet Licensing Service driver to the system.
1005	FNP_INSTALL_CANNOTSTART Failed to start the FlexNet Licensing Service or driver.
1006	FNP_INSTALL_CANNOTSTOP Failed to stop the FlexNet Licensing Service. Request the end-user to shut down all other applications. Repeat the call that failed. If this results in another failure, request the end-user to reboot the system.
1007	FNP_INSTALL_NORIGHTS Administrator privileges are required to install the FlexNet Licensing Service.

Table 7-11 • fnpActSvcInstallForMSI Return Values

Return Value	Description
1008	FNP_INSTALL_INVALIDPATH Invalid path to source or destination.
1010	FNP_INSTALL_EARLYREBOOT Reboot of system required to recover from a previous failed request. Request the end-user to reboot the system and then repeat the call that failed.
1100	**FNP_INSTALL_UNSUPPORTEDOS** The operating system version on the machine is not supported.
ERROR_BAD_CONFIGURATION	This error occurs if you have not have not set the FNPPUBLISHER or FNPPRODUCT.

Discussion

This function requires that FNP_Act_Installer.dll is available within the installer package.

This is a wrapper function for use by MSI-compatible installation applications and perfoms the same capability as **fnpActSvcInstallWin**. When creating the installation program, you must set the following properties:

- FNPPUBLISHER
- FNPPRODUCT

You set these properties and their values in the Property table of your MSI database.

Schedule this function for deferred execution in the `Execute` sequence somewhere after the `InstallValidate` action. To ensure that it runs only during the initial installation, give it the condition `Not Installed`.

See Also
fnpActSvcInstallWin
fnpActSvcUninstallForMSI

fnpActSvcUninstallForMSI

This function wraps fnpActSvcUninstallWin and allows you to uninstall the FlexNet Licensing Service from an MSI-compatible installer. This is not available to C/C++ applications.

Syntax

```
unsigned int fnpActSvcUninstallForMSI(MSIHANDLE hInstall);
```

Parameters

The **fnpActSvcUninstallForMSI** function uses the following parameters:

Table 7-12 • fnpActSvcUninstallForMSI Parameters

Parameter	Description
hInstall	This is the MSIHANDLE for the running installation.

Return Values

The following values are returned by **fnpActSvcUninstallForMSI.**

Table 7-13 • fnpActSvcUninstallForMSI Return Values

Return Value	Description
0	Indicates that the transaction was successful.
1002	FNP_INSTALL_INUSE The FlexNet Licensing Service driver is in use. Request the end-user to shut down all other applications. Repeat the call that failed. If this results in another failure, request the end-user to reboot the system.
1003	FNP_INSTALL_CANNOTCOPYFILE Failed to copy the FlexNet Licensing Service driver to the system.
1005	FNP_INSTALL_CANNOTSTART Failed to start the Licensing server or driver.
1006	FNP_INSTALL_CANNOTSTOP Failed to stop the FlexNet Licensing Service. Request the end-user to shut down all other applications. Repeat the call that failed. If this results in another failure, request the end-user to reboot the system.
1007	FNP_INSTALL_NORIGHTS Administrator privileges are required to install the FlexNet Licensing Service.
1008	FNP_INSTALL_INVALIDPATH Invalid path to source or destination.

Table 7-13 • fnpActSvcUninstallForMSI Return Values

Return Value	Description
1010	FNP_INSTALL_EARLYREBOOT Reboot of system required to recover from a previous failed request. Request the end-user to reboot the system and then repeat the call that failed.
1100	**FNP_INSTALL_UNSUPPORTEDOS** The operating system version on the machine is not supported.
ERROR_BAD_CONFIGURATION	This error occurs if you have not have not set the FNPPUBLISHER or FNPPRODUCT.

Discussion

This function requires that FNP_Act_Installer.dll is available within the installer package.

This is a wrapper function for use by MSI-compatible installation applications that perfoms the same capability as **fnpActSvcUninstallWin**. When creating the installation program, you must set the following properties:

- FNPPUBLISHER
- FNPPRODUCT

You set these properties and their values in the Property table of your MSI database.

Schedule this function for deferred execution in the `Execute` sequence somewhere after the `InstallValidate` action. To ensure that it runs only during a complete uninstallation, give it a condition such as `REMOVE="ALL"`.

See Also
fnpActSvcUninstallWin
fnpActSvcInstallForMSI

8

Activation API

Activation API is used to manipulate license rights held in trusted storage. These functions are normally invoked within an activation utility—either an *application* activation utility (which handles license rights in trusted storage on the FlexEnabled client) or a *server* activation utility (which handles license rights in trusted storage on the FlexNet license server).

An activation utility is built to support one of these activation transaction types:

- Composite transactions, which perform multiple actions per transaction. Composite transactions are supported for life-cycle operations between the publisher's activation server and the license server, as well as between the license server and FlexEnabled clients. From FlexNet Publisher 11.14.0, composite transactions are fully supported for all servers. From this version onwards, it is best practice to use composite transactions for all interactions with servers.
- Single-action transactions, which perform one action, such as an activation or return, per transaction. Licensing life-cycle operations performed between the FlexNet license server and FlexEnabled clients can use either single-action transactions or composite transactions. However, from FlexNet Publisher version 11.14.0, single-action transactions have been deprecated. It is now best practice to use composite transactions for all interactions with servers.

API Categories in This Chapter

Read this chapter in conjunction with the information in the *Programming Reference for Trusted Storage–Based Licensing*, which describes how to combine the API functions to create activation utilities. This chapter organizes the API function descriptions in the following categories:

- Common Library Functions—Functions used by both application and server activation utilities, including access to trusted storage and local activation. This includes API initialization and the creation of the activation handle.
- Composite Transaction API Functions—Functions used by application or server activation utilities that handle composite transactions to manipulate fulfillment records in trusted storage.
- Short Code Transaction APIs—Functions used to send request and response messages as numeric codes (called request codes and response codes), rather than XML messages.
- Common Data—Descriptions of data used by multiple functions.

The functions within the Activation Borrow API can be used in conjunction with or to replace functions within the Activation API.

API Structure—Objects

The following diagram illustrates the structures that are used by the API and how they are related.

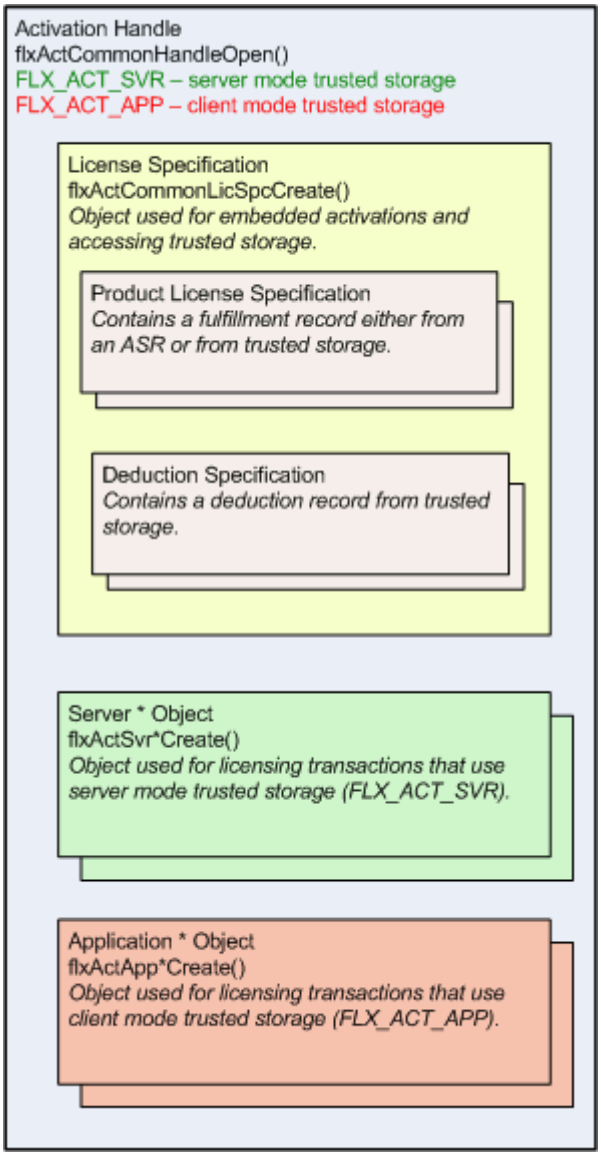

Figure 8-1: API Structures

There are two basic sequences in which objects are created, used and then deleted using functions within the API:

- *Create –> *Set –> execute –> *Delete
- *Create –> execute –> *Get –>*Delete

An example of the first sequence is provided by an activation transaction where the following sequence of API functions are typically used:

1. flxActAppActivationCreate—create an activation object
2. flxActAppActivationEntitlementIdSet—set entitlement ID for activation
3. flxActAppActivationExpDateSet—set required expiry date
4. flxActAppActivationSend—send activation request and process response
5. flxActAppActivationDelete—delete activation object

Objects maintain their own memory resources so creating an object allocates memory and deleting an object frees the memory resources used by that object. In a multithreaded environment create an object per thread. If an object must be shared between threads, you must provide your own synchronization between the threads.

Common Library Functions

This section contains common Activation API functions used when accessing trusted storage. It also includes functions required to implement local activations--that is, the addition of license rights from ASRs (activation specification records) to trusted storage.

The basic record in trusted storage is a fulfillment record. A fulfillment record contains details of license rights. The common library functions include functions for creating and manipulating a structure that contains a single fulfillment record. This structure is a *product license specification*. One or more product license specifications (fulfillment records) are held in a *license specification*. License specifications are used to read and write fulfillment records to and from trusted storage.

The following table lists the functions included in this section and defined in `FlxActCommon.h`.

Table 8-1 • Common Activation API Functions

Function	Description
flxActCommonDedSpcCountGet	Gets the count from the deduction record
flxActCommonDedSpcDestinationFulfillmentIdGet	Gets the destination fulfillment ID for the deduction record
flxActCommonDedSpcDestinationSystemNameGet	Gets the destination system name for the deduction record
flxActCommonDedSpcExpDateGet	Gets the expiry date of the deduction record
flxActCommonDedSpcTypeGet	Gets the type of the deduction record
flxActCommonGetFRCountFromServerTS	Remotely retrieves fulfillment-record counts from server-side trusted storage (per an activation-client request)
flxActCommonGetProtectionMode	Gets the application's protection mode
flxActCommonHandleClose	Deletes an Activation API handle
flxActCommonHandleDeleteProduct	Deletes all fulfillment records for specified product

Table 8-1 • Common Activation API Functions

Function	Description
flxActCommonHandleGetBuildInfo	Returns the build information of the FlexNet Publisher Licensing Toolkit used to prep the application.
flxActCommonHandleGetCommType	Gets the type of communication
flxActCommonHandleGetError	Gets the latest error code
flxActCommonHandleGetLastOpsError	Gets additional error information about transaction with a FlexNet Operations server
flxActCommonHandleGetLastOpsErrorString	Gets additional error information about transaction with a FlexNet Operations server
flxActCommonHandleGetLastResponseError	Gets the “error code” from the ErrorCode element in the last failure response received
flxActCommonHandleGetLastResponseErrorReason	Gets the reason from the last failure response received
flxActCommonHandleGetLastResponseErrorString	Gets the “error text” from the ErrorCode element in the last failure response received
flxActCommonHandleGetRemoteServer	Gets the address of the activation server
flxActCommonHandleGetServiceBuildInfo	Returns the build information of the FlexNet Licensing Service running on this machine.
flxActCommonHandleGetServiceVersion	Returns the version number of the FlexNet Licensing Service running on this machine.
flxActCommonHandleGetUniqueMachineNumber	Gets the unique machine number
flxActCommonHandleGetVersion	Returns the version number of the FlexNet Publisher Licensing Toolkit used to prep the application.
flxActCommonHandleOpen	Creates an Activation API handle
flxActCommonHandleSetCommType	Sets the type of communication used
flxActCommonHandleSetPollInterval	Sets response polling interval for SOAP
flxActCommonHandleSetProxyDetails	Sets the proxy server details when using SOAP
flxActCommonHandleSetRemoteServer	Stores the address of the activation server
flxActCommonHandleSetSSLDetails	Sets details of the certificate used for SSL

Table 8-1 • Common Activation API Functions

Function	Description
flxActCommonHandleSetStatusCallback	Sets a callback that reports changes in status of transactions
flxActCommonHandleSetTimeout	Sets timeout for SOAP communications
flxActCommonLibraryCleanup	Releases the activation library
flxActCommonLibraryInit	Initialization required to use the activation library and the Activation API
flxActCommonLicSpcAddASRFromBuffer	Adds fulfillment record from ASR in a buffer to trusted storage
flxActCommonLicSpcAddASRs	Adds fulfillment records from valid ASRs to trusted storage
flxActCommonLicSpcCreate	Creates a license specification
flxActCommonLicSpcDelete	Deletes a license specification
flxActCommonLicSpcGet	Gets the fulfillment record from the license specification
flxActCommonLicSpcGetNumberProducts	Number of fulfillment records in the license specification
flxActCommonLicSpcGetNumProdEntries	Number of fulfillment records for the product
flxActCommonLicSpcGetProd	Gets the fulfillment record for the product from the license specification
flxActCommonLicSpcPopulateAllFromServerTS	Retrieves all (expired, non-expired, trusted, not trusted) fulfillment records from remote server-side trusted storage into a license specification structure
flxActCommonLicSpcPopulateAllFromTS	Loads all fulfillment records from trusted storage
flxActCommonLicSpcPopulateFromServerTS	Retrieves trusted fulfillment records (expired or non-expired) from remote server-side trusted storage into a license specification structure
flxActCommonLicSpcPopulateFromTS	Loads valid fulfillment records from trusted storage
flxActCommonLicSpcProductDelete	Deletes the fulfillment record from trusted storage
flxActCommonProdLicSpcActServerChainGet	Gets details of the route by which the fulfillment record was served
flxActCommonProdLicSpcCountGet	Reads the number of licenses from a fulfillment record

Table 8-1 • Common Activation API Functions

Function	Description
flxActCommonProdLicSpcDedSpcGet	Gets the specified deduction record from the product license specification
flxActCommonProdLicSpcEntitlementIdGet	Gets the entitlement ID from a fulfillment record
flxActCommonProdLicSpcExpDateGet	Gets the expiration date of a fulfillment record
flxActCommonProdLicSpcFeatureLineGet	Gets the feature definition lines from a fulfillment record
flxActCommonProdLicSpcFulfillmentIdGet	Gets the ID of the fulfillment record
flxActCommonProdLicSpcFulfillmentTypeGet	Gets the fulfillment type from the fulfillment record
flxActCommonProdLicSpcIsDisabled	Returns the status of the specified fulfillment record
flxActCommonProdLicSpcNumberDedSpcGet	Number of deduction records for the fulfillment record
flxActCommonProdLicSpcNumberValidDedSpcGet	Number of valid deduction records for the fulfillment record
flxActCommonProdLicSpcPopulateFRFromServerTS	Retrieves detailed information for the fulfillment record obtained from **flxActCommonLicSpc PopulateFromServerTS** or **flxActCommon LicSpcPopulateAllFromServerTS** API
flxActCommonProdLicSpcProductIdGet	Gets the product ID from a fulfillment record
flxActCommonProdLicSpcSuiteIdGet	Gets the suite ID from a fulfillment record
flxActCommonProdLicSpcTrustFlagsGet	Gets the trust status of a fulfillment record
flxActCommonProdLicSpcUniqueIdGet	Gets the unique ID from the fulfilment record in the supplied product license specification
flxActCommonProdLicSpcVendorDataGetByIndex	Gets the vendor data dictionary key-value pair for the specified index
flxActCommonProdLicSpcVendorDataGetByKey	Gets specified data value from the vendor data dictionary in the fulfillment record
flxActCommonProdLicSpcVendorDataGetCount	Number of vendor data dictionary entries in the fulfillment record
flxActCommonRepairLocalTrustedStorage	Repairs broken segments of trusted storage so that fulfillment records can be added to them
flxActCommonResetTrialASRFromBuffer	Resets trial anchor using ASR in a buffer

Table 8-1 • Common Activation API Functions

Function	Description
flxActCommonResetTrialASRs	Resets trial anchors
flxActCommonRespProdLicSpcGet	Obtains information about the fulfilment record affected by processing a v1 (single-action) transaction response.
flxActDiagTrustedStorageHealthCheck	Performs a series of checks related to trusted-storage health and records the result as event log entries.
flxActCommonValidateDateString	Validates the supplied date string.
flxActCommonVirtualFamilyGet	Returns the current setting of the Virtual Machine Family attribute.
flxActCommonVirtualGenidGet	Returns the current setting of the Virtual Machine GenId attribute.
flxActCommonVirtualNameGet	Returns the current setting of the Virtual Machine Name attribute.
flxActCommonVirtualStatusGet	Gets the platform status of the requesting machine—virtual or physical
flxActCommonVirtualUuidGet	Returns the current setting of the Virtual Machine UUID attribute.

flxActCommonDedSpcCountGet

Get the count from the supplied deduction record.

Syntax

```
uint32_t flxActCommonDedSpcCountGet(
    FlxActDedSpc dedSpc,
    FlxActCountType countType);
```

Parameters

The **flxActCommonDedSpcCountGet** function uses the following parameters:

Table 8-2 • flxActCommonDedSpcCountGet Parameters

Parameter	Description
dedSpc	A deduction specification that contains a deduction record.

Table 8-2 • flxActCommonDedSpcCountGet Parameters

Parameter	Description
countType	The type of count required. For information about these license types, see the *Programming Reference for Trusted Storage–Based Licensing*. The following values (FlxActCountType) can be used for this parameter: ● `CONCURRENT`—Concurrent licenses ● `CONCURRENT_OD`—Concurrent overdraft licenses ● `ACTIVATABLE`—Activatable licenses ● `ACTIVATABLE_OD`—Activatable overdraft licenses ● `HYBRID`—Hybrid licenses ● `HYBRID_OD`—Hybrid overdraft licenses ● `REPAIR`—The number of repairs that the remote license server can provide for activatable licenses included in the remote fulfillment record

Return Values

The specified count in the deduction record or NULL if not set.

- **Activation deduction records**—the only counts set are activatable or hybrid
- **Transfer deduction records**—all counts may be set
- **Repair deduction records**—no counts are set.

Discussion

Before calling this routine access the deduction record for which the counts are required: read fulfillment records from trusted storage and select the required fulfillment record. Then select the required deduction record associated with the selected fulfillment record. For example, flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, flxActCommonLicSpcGet, flxActCommonProdLicSpcNumberDedSpcGet, and flxActCommonProdLicSpcDedSpcGet.

This function returns the value of the specified count in the deduction record supplied in the deduction specification.

See Also
flxActCommonDedSpcDestinationFulfillmentIdGet
flxActCommonDedSpcDestinationSystemNameGet
flxActCommonDedSpcExpDateGet
flxActCommonDedSpcTypeGet

flxActCommonDedSpcDestinationFulfillmentIdGet

Get the destination fulfillment ID from the deduction record in the supplied deduction specification.

Syntax

```
const char *flxActCommonDedSpcDestinationFulfillmentIdGet(FlxActDedSpc dedSpc);
```

Parameters

The **flxActCommonDedSpcDestinationFulfillmentIdGet** function uses the following parameters:

Table 8-3 • flxActCommonDedSpcDestinationFulfillmentIdGet Parameters

Parameter	Description
dedSpc	A deduction specification that contains a deduction record.

Return Values

The fulfillment ID from the deduction record or NULL if not set.

Discussion

Before calling this routine access the deduction record that contains the destination fulfillment ID that is required: read fulfillment records from trusted storage and select the required fulfillment record. Then select the required deduction record associated with the selected fulfillment record. For example, flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, flxActCommonLicSpcGet, flxActCommonProdLicSpcNumberDedSpcGet, and flxActCommonProdLicSpcDedSpcGet.

This function returns the *destination fulfillment ID* from the specified deduction record. For an explanation of destination fulfillment ID, see the information about deduction records in the *Programming Reference for Trusted Storage–Based Licensing*.

The destination fulfillment ID together with the destination system name, obtained using flxActCommonDedSpcDestinationSystemNameGet, enable the remote fulfillment record to be located. This information could potentially be used to request the return of license rights to the server that issued them.

See Also
flxActCommonDedSpcDestinationSystemNameGet
flxActCommonDedSpcExpDateGet
flxActCommonDedSpcTypeGet
flxActCommonValidateDateString

flxActCommonDedSpcDestinationSystemNameGet

Get the destination system name from the deduction record in the supplied deduction specification.

Syntax

```
const char *flxActCommonDedSpcDestinationSystemNameGet(FlxActDedSpc dedSpc);
```

Parameters

The **flxActCommonDedSpcDestinationSystemNameGet** function uses the following parameters:

Table 8-4 • flxActCommonDedSpcDestinationSystemNameGet Parameters

Parameter	Description
dedSpc	A deduction specification that contains a deduction record.

Return Values

The destination system name from the deduction record or NULL if not set.

Discussion

Before calling this routine access the deduction record that contains the destination system name that is required: read fulfillment records from trusted storage and select the required fulfillment record. Then select the required deduction record associated with the selected fulfillment record. For example, flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, flxActCommonLicSpcGet, flxActCommonProdLicSpcNumberDedSpcGet, and flxActCommonProdLicSpcDedSpcGet.

This function returns the *destination system name* from the specified deduction record. For an explanation of destination system name, see the information about deduction records in the *Programming Reference for Trusted Storage–Based Licensing*.

The destination system name together with the destination fulfillment ID, obtained using flxActCommonDedSpcDestinationFulfillmentIdGet, enable the remote fulfillment record to be located. This information could potentially be used to request the return of license rights to the server that issued them.

See Also
flxActCommonDedSpcDestinationFulfillmentIdGet
flxActCommonDedSpcExpDateGet
flxActCommonDedSpcTypeGet
flxActCommonValidateDateString

flxActCommonDedSpcExpDateGet

Get the expiry date of the deduction record in the supplied deduction specification.

Syntax

```
const char *flxActCommonDedSpcExpDateGet(FlxActDedSpc dedSpc);
```

Parameters

The **flxActCommonDedSpcExpDateGet** function uses the following parameters:

Table 8-5 • flxActCommonDedSpcExpDateGet Parameters

Parameter	Description
dedSpc	A deduction specification that contains a deduction record.

Return Values

The expiry date of the deduction record or NULL if not set. Expiry date is not set in repair deduction records.

Discussion

Before calling this routine access the deduction record whose expiry date is required: read fulfillment records from trusted storage and select the required fulfillment record. Then select the required deduction record associated with the selected fulfillment record. For example, flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, flxActCommonLicSpcGet, flxActCommonProdLicSpcNumberDedSpcGet, and flxActCommonProdLicSpcDedSpcGet.

This function returns the expiry date of the specified deduction record. This is the expiry date of the remote fulfillment record whose details are recorded in the specified deduction record. For a description of deduction records, see the *Programming Reference for Trusted Storage–Based Licensing*.

See Also
flxActCommonDedSpcDestinationFulfillmentIdGet
flxActCommonDedSpcDestinationSystemNameGet
flxActCommonDedSpcTypeGet
flxActCommonValidateDateString

flxActCommonDedSpcTypeGet

Get the type of transaction that created the deduction record in the supplied deduction specification.

Syntax

```
FlxActDeductionType flxActCommonDedSpcTypeGet(FlxActDedSpc dedSpc);
```

Parameters

The **flxActCommonDedSpcTypeGet** function uses the following parameters:

Table 8-6 • flxActCommonDedSpcTypeGet Parameters

Parameter	Description
dedSpc	A deduction specification that contains a deduction record.

Return Values

The type of transaction that created the deduction record or NULL if not set. For details of valid values see FlxActDeductionType.

Discussion

Before calling this routine access the deduction record that contains the fulfillment ID that is required: read fulfillment records from trusted storage and select the required fulfillment record. Then select the required deduction record associated with the selected fulfillment record. For example, flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, flxActCommonLicSpcGet, flxActCommonProdLicSpcNumberDedSpcGet, and flxActCommonProdLicSpcDedSpcGet.

This function returns the type of transaction that created the specified deduction record as recorded in the deduction record.

See Also
flxActCommonDedSpcDestinationFulfillmentIdGet
flxActCommonDedSpcDestinationSystemNameGet
flxActCommonDedSpcExpDateGet
flxActCommonValidateDateString

flxActCommonGetFRCountFromServerTS

Retrieves total counts of fulfillment records from remote server-side trusted storage:

- Total count of all fulfillment records (trusted, non-trusted, expired, non-expired)
- Total count of trusted fulfillment records (expired and non-expired)

Note • *The use of this function is limited to the retrieval of fulfillment information in server-side trusted-storage only; it cannot be used to retrieve fulfillment information stored in the client trusted storage.*

Syntax

```
FlxActBool   flxActCommonGetFRCountFromServerTS(
    FlxActHandle handle,
    const char *pszEntitlementID,
    const char *pszProductID,
    const char *pszServer,
    uint32_t *pnTrustedFRCount,
    uint32_t *pnTotalFRCount);
```

Parameters

The **flxActCommonGetFRCountFromServerTS** function uses the following parameters:

Table 8-7 • flxActCommonGetFRCountFromServerTS Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.

Table 8-7 • flxActCommonGetFRCountFromServerTS Parameters

Parameter	Description
pszEntitlementID	The pointer variable storing the value of provided entitlement ID. ● If this parameter is specified, only fulfillment records belonging to the specified entitlement ID are taken into account when retrieving the total fulfillment-record and total trusted fulfillment-record counts. ● If this parameter is `NULL` or an empty string, all fulfillment records from server-trusted storage will be taken into account for these totals. ● If parameters are specified for both the entitlement ID and product ID, fulfillment records that match both filters are taken into account for the totals.
pszProductID	The pointer variable storing the value of provided product ID. ● If this parameter is specified, only fulfillment records belonging to the specified product ID are taken into account when retrieving the total fulfillment-record and total trusted fulfillment-record counts. ● If this parameter is `NULL` or an empty string, all fulfillment records from server-side trusted storage are taken into account for these totals. ● If parameters are specified for both the entitlement ID and product ID, fulfillment records that match both filters are taken into account for the totals.
pszServer	A pointer to the variable storing the address of the license server from which the counts are retrieved. The following formats can be used for this parameter: ● *port@host* ● *@host*
pnTrustedFRCount	The pointer variable used to hold the total count of trusted fulfillment records (expired and non-expired) in server-side trusted storage.
pnTotalFRCount	This pointer variable used to hold the total count of all (including trusted, non- trusted, expired, non-expired) fulfillment records in server-side trusted storage.

Return Values

The following values are returned by **flxActCommonGetFRCountFromServerTS**:

Table 8-8 • flxActCommonGetFRCountFromServerTS Return Values

Return Value	Description
FLX_ACT_TRUE	Success

Table 8-8 • flxActCommonGetFRCountFromServerTS Return Values

Return Value	Description
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to determine the specific error. Possible errors include: • `LM_TS_BADPARAM`—One or more of the parameters passed in are invalid. • `LM_TS_CANTMALLOC`—Insufficient memory has caused an error. • `LM_TS_INVALID_REQUEST_TYPE`—The activation server does not support this request type. (This error normally occurs if the request is corrupt.) • `LM_TS_SERVERQUERY_GEN_REQ`—The request to query remote server-side trusted storage cannot be generated. • `LM_TS_UNKNOWN_RESP`—The response is corrupt. • `LM_TS_SERVERQUERY_RESPONSE_FAILED`—The response to query remote server-side trusted storage cannot be processed.

Discussion

Use this function along with the following APIs to retrieve fulfillment-record information from server-side trusted storage:

- flxActCommonLicSpcPopulateFromServerTS
- flxActCommonLicSpcPopulateAllFromServerTS
- flxActCommonProdLicSpcPopulateFRFromServerTS

Before calling this function, create an activation handle using **flxActCommonHandleOpen**. See flxActCommonHandleOpen for information about this process. (**flxActCommonGetFRCountFromServerTS** can use a handle created in either `FLX_ACT_APP` or `FLX_ACT_SVR` mode.)

Important • *The remote license server whose trusted storage you want to read must be running.*

flxActCommonGetProtectionMode

Returns the application's protection mode; that is, whether the preptool XML parameters contain `isserver="true"` or `isserver="false"`.

Note • *This function is used by the sample composite-transaction activation utility, but is useful in general to developers who use common source code to create their application and server activation utilities.*

Syntax

```
FlxActMode flxActCommonGetProtectionMode();
```

Return Values

The following values are returned by **flxActCommonGetProtectionMode**:

Table 8-9 • flxActCommonGetProtectionMode Return Values

Return Value	Description
FLX_ACT_SVR	Protection mode is set to FLX_ACT_SVR.
FLX_ACT_APP	Protection mode is set to FLX_ACT_APP.

flxActCommonHandleClose

Closes an activation handle.

Syntax

```
FlxActBool flxActCommonHandleClose(FlxActHandle handle);
```

Parameters

The **flxActCommonHandleClose** function uses the following parameters:

Table 8-10 • flxActCommonServiceHandleClose Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen

Return Values

The following values are returned by **flxActCommonHandleClose:**

Table 8-11 • flxActCommonHandleClose Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success

Discussion

This function closes the activation handle and releases all resources associated with it.

flxActCommonHandleDeleteProduct

Deletes all fulfillment records held in trusted storage that are for the specified product.

Syntax

```
FlxActBool flxActCommonHandleDeleteProduct(
    FlxActHandle handle,
    const char *pszProductId);
```

Parameters

The **flxActCommonHandleDeleteProduct** function uses the following parameters:

Table 8-12 • flxActCommonHandleDeleteProduct Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
pszProductId	Deletes fulfillment records that contain this product ID.

Return Values

The following values are returned by **flxActCommonHandleDeleteProduct:**

Table 8-13 • flxActCommonHandleDeleteProduct Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function deletes all the fulfillment records in trusted storage for a specified product, identified by the product ID. All fulfillment records in trusted storage with this product ID are deleted. Any outstanding transactions that include the specified product ID are canceled.

A standard activation request, that is not one using short codes, does not contain a product ID. Thus a fulfillment record with the specified product ID is added to trusted storage under the following circumstances:

- A standard activation request has been sent and the response not processed.
- The activation response contains the specified product ID.
- The activation response is processed after **flxActCommonHandleDeleteProduct** has completed.

It is recommended that this function is used to delete fulfillment records only from client-mode trusted storage. Before calling this function, open an activation handle using flxActCommonHandleOpen **with** the parameter `mode` set to `FLX_ACT_APP`. If fulfillment records are deleted from a license server, it will leave any fulfillment records supplied from the deleted record orphaned. These orphaned records cannot be either returned or repaired.

For instructions about deleting fulfillment records for a product, see information about fulfillment record manipulation in the *Programming Reference for Trusted Storage–Based Licensing*. For example code that uses this function, see **sHandleDeleteProductFromTS** in the example application activation utility, `appActUtil.c`.

flxActCommonHandleGetBuildInfo

Returns the build information of the FlexNet Publisher Licensing Toolkit that was used to prep the application.

For release versions the version number returned by flxActCommonHandleGetVersion is sufficient to identify the build. This additional information is used to identify pre-release, Beta, and diagnostic builds.

Syntax

```
FlxActBool flxActCommonHandleGetBuildInfo(
    FlxActHandle handle,
    uint32_t *pBuildNo,
    uint32_t *pBuildYear,
    uint32_t *pBuildMonth,
    uint32_t *pBuildDay);
```

Parameters

The **flxActCommonHandleGetBuildInfo** function uses the following parameters:

Table 8-14 • flxActCommonHandleGetBuildInfo Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
pBuildNo	For return of the build number
pBuildYear	For return of the build year
pBuildMonth	For return of the build month (1–12)
pBuildDay	For return of the build day (1–31)

Return Values

The following values are returned by **flxActCommonHandleGetBuildInfo:**

Table 8-15 • flxActCommonHandleGetBuildInfo Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Handle is NULL

See Also
flxActCommonHandleGetVersion

flxActCommonHandleGetCommType

Gets the type of communication used for the connection to the activation server.

Syntax

```
FlxActCommType flxActCommonHandleGetCommType(FlxActHandle handle);
```

Parameters

The **flxActCommonHandleGetCommType** function uses the following parameters:

Table 8-16 • flxActCommonHandleGetCommType Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.

Return Values

The type of communication set for the connection to the activation server:

- flxCommsNone = 0, no communications set
- flxCommsMvsnFlex = 1, used for connections to FlexNet license server
- flxCommsMvsnSoap = 4, used for connections to FlexNet Operations server

Discussion

This function returns the type of communication that has been set to be used for connections to the activation server. The type of communication defines the type of activation server, either a FlexNet license server or a FlexNet Operations server.

flxActCommonHandleGetError

Gets the latest error code for the last operation performed using the specified handle.

Syntax

```
void flxActCommonHandleGetError(
    FlxActHandle handle,
    FlxActError *err);
```

Parameters

The **flxActCommonHandleClose** function uses the following parameters:

Table 8-17 • flxActCommonServiceHandleClose Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen

Table 8-17 • flxActCommonServiceHandleClose Parameters

Parameter	Description
err	Resulting error code, if any. For an explanation of the contents of `err`, see FlxActError.

Return Values

This function returns the last error code as the parameter `err`.

Discussion

Call this function when any of the Activation API functions returns an error to obtain more information about the cause of the error. For detailed information about the error codes, see FlxActError.

flxActCommonHandleGetLastOpsError

Gets the last error code returned by the FlexNet Operations server.

Syntax

```
uint32_t flxActCommonHandleGetLastOpsError(FlxActHandle handle);
```

Parameters

The **flxActCommonHandleGetLastOpsError** function uses the following parameters:

Table 8-18 • flxActCommonGetLastOpsError Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.

Return Values

The FlexNet Operations server error code or zero if no error code is available, that is if there has been no FlexNet Operations server error returned.

Discussion

Before calling this function, call flxActCommonHandleGetError. This function will only return a useful value when **flxActCommonHandleGetError** returns major error code `LM_TS_OPERATIONS`, 50040.

For information about FlexNet Operations server errors, see the *FlexNet Operations Implementation Guide*.

flxActCommonHandleGetLastOpsErrorString

Gets text that explains the last error code returned by the FlexNet Operations server.

Syntax

```
const char *flxActCommonHandleGetLastOpsErrorString(FlxActHandle handle);
```

Parameters

The **flxActCommonHandleGetLastOpsErrorString** function uses the following parameters:

Table 8-19 • flxActCommonGetLastOpsErrorString Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.

Return Values

The FlexNet Operations server error text or NULL if no error text is available.

Discussion

Before calling this function call flxActCommonHandleGetError. This function will only return a useful value when **flxActCommonHandleGetError** returns major error code `LM_TS_OPERATIONS`, 50040.

For information about FlexNet Operations server errors, see the *FlexNet Operations Implementation Guide*.

flxActCommonHandleGetLastResponseError

Gets the error code from the ErrorCode element in the last failure response received.

Syntax

```
uint32_t flxActCommonHandleGetLastResponseError(FlxActHandle handle);
```

Parameters

The **flxActCommonHandleGetLastResponseError** function uses the following parameters:

Table 8-20 • flxActCommonGetLastResponseError Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.

Return Values

The reason for the failure of an activation transaction from the last failure response or NULL if no error code is available.

Discussion

All types of activation transactions (activate, return or repair) either return a response XML document that completes the transaction or a failure response that reports the reason for the failure of the transaction and optionally an error string.

The reason is one of the following:

- ERROR—Failure is due to an error in processing the request
- DENIED—Failure is due to the request being denied

The error string has no defined format. This function assumes that it has the following format and extracts the error_code from the error string:

error_code;error_text

When the error string does not have this format this function will return NULL and the complete error string is returned by flxActCommonHandleGetLastResponseErrorString.

See Also
flxActCommonHandleGetLastResponseErrorString

flxActCommonHandleGetLastResponseErrorReason

Gets the reason from the last failure response received.

Syntax

```
FlxErrorResponseReason flxActCommonHandleGetLastResponseErrorReason(FlxActHandle handle);
```

Parameters

The **flxActCommonHandleGetLastResponseErrorReason** function uses the following parameters:

Table 8-21 • flxActCommonHandleGetLastResponseErrorReason Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.

Return Values

The reason for the error as defined by the value of **FlxErrorResponseReason** as given in the following table:

Table 8-22 • flxActCommonHandleGetLastResponseErrorReason Return Values

Value	Type	Description
0	`flxErrorResponseReasonUnknown`	An unknown error reason was received
1	`flxErrorResponseReasonError`	Failure is due to an error in processing the request
2	`flxErrorResponseReasonDenied`	Failure is due to the request being denied

Discussion

All types of activation transactions (activate, return, or repair) either return a response XML document that completes the transaction or a failure response that reports the reason for the failure of the transaction and optionally an error string.

This function returns the reason for the failure, which is one of the following:

- ERROR—An error in processing the request
- DENIED—The request being denied

See Also
flxActCommonHandleGetLastResponseError
flxActCommonHandleGetLastResponseErrorString

flxActCommonHandleGetLastResponseErrorString

Gets the error text from the last failure response received.

Syntax

```
const char *flxActCommonHandleGetLastResponseErrorString(FlxActHandle handle);
```

Parameters

The **flxActCommonHandleGetLastResponseErrorString** function uses the following parameters:

Table 8-23 • flxActCommonHandleGetLastResponseErrorString Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.

Return Values

The error text from the last failure response or NULL if no error text is available.

Discussion

All types of activation transactions (activate, return, or repair) either return a response XML document that completes the transaction or a failure response that reports the reason for the failure of the transaction and optionally an error string.

The failure is due to one of the following:

- ERROR—An error in processing the request
- DENIED—The request being denied

The error string has no defined format. This function extract the error_text from the error string when it has the following format:

error_code;error_text

When the error string has any other format, this function returns the complete error string.

See Also
flxActCommonHandleGetLastResponseError

flxActCommonHandleGetRemoteServer

Returns the address of the activation server to which activation transactions will be sent.

Syntax

```
const char *flxActCommonHandleGetRemoteServer(FlxActHandle handle);
```

Parameters

The **flxActCommonHandleGetRemoteServer** function uses the following parameters:

Table 8-24 • flxActCommonHandleGetRemoteServer Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.

Return Values

The activation server address or NULL if it has not been previously set.

Discussion

This function is used in conjunction with flxActCommonHandleSetRemoteServer and flxActCommonHandleSetCommType that define the address and type of activation server.

flxActCommonHandleGetServiceBuildInfo

Returns the build information of the FlexNet Licensing Service that is running on the machine.

For release versions the version number returned by flxActCommonHandleGetServiceVersion is sufficient to identify the build. This additional information is used to identify pre-release, Beta, and diagnostic builds.

Syntax

```
FlxActBool flxActCommonHandleGetServiceBuildInfo(
    FlxActHandle handle,
    uint32_t *pBuildNo,
    uint32_t *pBuildYear,
    uint32_t *pBuildMonth,
    uint32_t *pBuildDay);
```

Parameters

The **flxActCommonHandleGetServiceBuildInfo** function uses the following parameters:

Table 8-25 • flxActCommonHandleGetServiceBuildInfo Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.

Table 8-25 • flxActCommonHandleGetServiceBuildInfo Parameters

Parameter	Description
pBuildNo	For return of the build number
pBuildYear	For return of the build year
pBuildMonth	For return of the build month (1–12)
pBuildDay	For return of the build day (1–31)

Return Values

The following values are returned by **flxActCommonHandleGetServiceBuildInfo:**

Table 8-26 • flxActCommonHandleGetServiceBuildInfo Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

If the function is called on a machine on which the FlexNet Licensing Service is not present, major error `LM_TS_SERVICE_INFO` is returned. The system error indicates why the information could not be obtained from the service. For more information on possible system errors, see FlxActSystemError.

See Also
flxActCommonHandleGetServiceVersion

flxActCommonHandleGetServiceVersion

Returns the version of the FlexNet Licensing Service running on this machine.

Syntax

```
FlxActBool flxActCommonHandleGetServiceVersion(
    FlxActHandle handle,
    uint32_t *pMajor,
    uint32_t *pMinor,
    uint32_t *pMaint,
    uint32_t *pHotfix);
```

Parameters

The **flxActCommonHandleGetServiceVersion** function uses the following parameters:

Table 8-27 • flxActCommonHandleGetServiceVersion Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
pMajor	For return of the major version number
pMinor	For return of the minor version number
pMaint	For return of the maintenance version number
pHotfix	For return of the hotfix version number

Return Values

The following values are returned by **flxActCommonHandleGetServiceVersion:**

Table 8-28 • flxActCommonHandleGetServiceVersion Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

If the function is called on a machine on which the FlexNet Licensing Service is not present, major error `LM_TS_SERVICE_INFO` is returned. The system error indicates why the information could not be obtained from the service. For more information on possible system errors, see FlxActSystemError.

See Also
flxActCommonHandleGetServiceBuildInfo

flxActCommonHandleGetUniqueMachineNumber

Get the Unique Machine Number (UMN).

Syntax

```
const char *flxActCommonHandleGetUniqueMachineNumber(
    FlxActHandle handle,
    FlxUMNType type);
```

Parameters

The **flxActCommonHandleGetUniqueMachineNumber** function uses the following parameters:

Table 8-29 • flxActCommonHandleGetUniqueMachineNumber Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
type	The type of unique machine number requested: `flxUMNTypeOne`, `flxUMNTypeTwo`, `flxUMNTypeThree`, or `flxUMNTypeFour`. For details of valid values see "Information Used to Identify the System Initiating the Request" in the *Programming Reference for Trusted Storage–Based Licensing*. ***Note*** • *These types are also referred to by their common names UMN1, UMN2, UMN3, or UMN4, respectively, in the FlexNet Publisher documentation.*

Return Values

If an error occurs, it will return NULL. If NULL is returned, call **flxActCommonHandleGetError** to determine additional error information. Possible errors include:

- `LM_TS_CANT_FIND`—The UMN could not be found.
- `LM_TS_INTERNAL_ERROR`—An unexpected internal error occurred.
- `LM_TS_BAD_PARAM`—One or more of the parameters passed in were invalid.
- `LM_TS_UNSUPPORTED`—The UMN is not supported.

Discussion

See Unique Machine Number (UMN) for an explanation of UMNs.

For toolkit versions 11.5 and later, the UMN is automatically included in all requests to an activation server. Thus, this function is required only for activities such as client-side reporting of UMNs.

See Also
flxActCommonHandleGetError
flxActAppActivationVendorDataSet
flxActAppReturnVendorDataSet
flxActAppRepairVendorDataSet

flxActCommonHandleGetVersion

Returns the version number of the FlexNet Publisher Licensing Toolkit that was used to prep the application.

Syntax

```
FlxActBool flxActCommonHandleGetVersion(
    FlxActHandle handle,
```

```
    uint32_t *pMajor,
    uint32_t *pMinor,
    uint32_t *pMaint,
    uint32_t *pHotfix);
```

Parameters

The **flxActCommonHandleGetVersion** function uses the following parameters:

Table 8-30 • flxActCommonHandleGetVersion Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
pMajor	For return of the major version number
pMinor	For return of the minor version number
pMaint	For return of the maintenance version number
pHotfix	For return of the hotfix version number

Return Values

The following values are returned by **flxActCommonHandleGetVersion:**

Table 8-31 • flxActCommonHandleGetVersion Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Handle is NULL

See Also
flxActCommonHandleGetBuildInfo

flxActCommonHandleOpen

Creates an Activation API handle.

Syntax

```
FlxActBool flxActCommonHandleOpen(
    FlxActHandle *handle,
    FlxActMode mode,
    FlxActError *err);
```

Parameters

The **flxActCommonHandleOpen** function uses the following parameters:

Table 8-32 • flxActCommonServiceHandleOpen Parameters

Parameter	Description
handle	The resulting pointer to the newly opened Activation API handle.
mode	Defines the mode in which the activation utility using this handle executes. Possible modes are: ● FLX_ACT_SVR—Enables activation transactions with server mode trusted storage. Use this mode when creating a handle for a server activation utility. Subsequent code should use Common Library Functions or Deprecated License Server Activation Functions. ● FLX_ACT_APP—Enables activation transactions with client mode trusted storage. Use this mode when creating a handle for an application activation utility. Subsequent code should use Common Library Functions or Deprecated Single-Action Application Activation Functions. ● FLX_ACT_PREP— Enables composite activation transactions. Subsequent code should use Common Library Functions or Deprecated Single-Action Application Activation Functions.
err	Resulting error code, if any. If this method returns FLX_ACT_FALSE, then check err for the current error. For an explanation of the contents of err, see FlxActError.

Return Values

The following values are returned by **flxActCommonHandleOpen:**

Table 8-33 • flxActCommonHandleOpen Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success

Discussion

Before calling this function, the activation library must have been initialized successfully (flxActCommonLibraryInit).

This function creates an activation handle that is required to use all the other Activation API functions.

All open handles should be closed using flxActCommonHandleClose before calling flxActCommonLibraryCleanup when the Activation API is no longer required.

flxActCommonHandleSetCommType

Sets the type of communication used for the connection to the activation server used for transactions.

Syntax

```
FlxActBool flxActCommonHandleSetCommType(
    FlxActHandle handle,
    FlxActCommType commType);
```

Parameters

The **flxActCommonHandleSetCommType** function uses the following parameters:

Table 8-34 • flxActCommonHandleSetCommType Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
commType	The type of comms required. Set this in relation to the type of activation server: ● FlexNet license server—set **flxCommsMvsnFlex** ● FlexNet Operations server—set **flxCommsMvsnSoap** The full set of values for FlxActCommType are: ● flxCommsNone = 0 ● flxCommsMvsnFlex = 1 ● flxCommsMvsnSoap = 4

Return Values

The following values are returned by **flxActCommonHandleSetCommType:**

Table 8-35 •

Return Value	Description
FLX_ACT_TRUE	Success
FLEX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function sets the type of communication used for the connection to the activation server. It must be set in relation to the type of activation server: defined when setting the address of the activation server using flxActCommonHandleSetRemoteServer.

Use flxActCommonHandleGetCommType to retrieve the type of communication currently set.

For connections to FlexNet license servers the built in FLEX comms is used. This is completely transparent when using the Activation API. No other comms functions are required for transactions with a FlexNet license server.

For connections to FlexNet Operations servers a SOAP connection is used for the transport layer. It is implemented by the FnpCommsSoap module that is included in the toolkit. If your application utility uses SOAP, you must ensure that this module is available:

- **Windows**—Put `FnpCommsSoap.dll` in the same directory as your application utility.
- **Mac**—When your product is packaged in a bundle, put `FnpCommsSoap.dylib` with your application utility.
- **UNIX**—Put `FnpCommsSoap.*` in the normal location for libraries (typically, `/usr/local/lib`).

The Windows Soap module, `FnpCommsSoap.dll`, uses WinINet.

The following additional comms functions may optionally be used when using a SOAP connection to control or monitor a transaction:

- flxActCommonHandleSetTimeout
- flxActCommonHandleSetProxyDetails
- flxActCommonHandleSetSSLDetails
- flxActCommonHandleSetStatusCallback

When this function is not used, or commType is set to NULL, the default is flxCommsMvsnFlex, that is, connection to a license server is assumed. If flxActCommonHandleSetRemoteServer is also not used, or set to NULL, then the default license server address is @localhost.

flxActCommonHandleSetPollInterval

Sets polling interval for responses when using Soap communications.

Syntax

```
FlxActBool flxActCommonHandleSetPollInterval(
    FlxActHandle handle,
    uint32_t pollInterval);
```

Parameters

The **flxActCommonHandleSetPollInterval** function uses the following parameters:

Table 8-36 • flxActCommonHandleSetPollInterval Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
pollInterval	Interval in seconds between successive polls for a response. The default value when this function is not used is to poll every second. When this value is set to 0, zero, the default value of polling every second is used.

Return Values

The following values are returned by **flxActCommonHandleSetPollInterval:**

Table 8-37 • flxActCommonHandleSetPollInterval Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before using this function call flxActCommonHandleSetCommType with commType set to **flxCommsMvsnSoap**. This sets the use of SOAP for connections to the activation server.

Use this optional function to set the interval between polls for responses on the SOAP connection to the activation server. The default when this function is not used is to poll every second.

flxActCommonHandleSetProxyDetails

Sets the proxy server details when using a SOAP connection.

Syntax

```
FlxActBool flxActCommonHandleSetProxyDetails(
    FlxActHandle handle,
    uint32_t port,
    const char *pHost,
    const char *pUserid,
    const char *pPasswd);
```

Parameters

The **flxActCommonHandleSetProxyDetails** function uses the following parameters:

Table 8-38 • flxActCommonHandleSetProxyDetails Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
port	The proxy server port number.
pHost	The name of the proxy server.
pUserid	Optionally, when the proxy server requires authentication, a user name. Set this to NULL when authentication is not required.
pPasswd	Optionally, when the proxy server requires authentication, a password. Set this to NULL when authentication is not required.

Return Values

The following values are returned by **flxActCommonHandleSetProxyDetails:**

Table 8-39 • flxActCommonHandleSetProxyDetails Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before using this function call flxActCommonHandleSetCommType with commType set to flxCommsMvsnSoap. This sets the use of SOAP for connections to the activation server.

Use this optional function to set a proxy server for SOAP communications for platforms other than Windows. On Windows platforms this functionality is performed by WinINet (see WinINet documentation for details).

This function is required, on platforms other than Windows, when an enterprise uses a proxy server to handle external communications. This is typically used when a firewall is in use. The proxy server must support either HTTP version 1.0 or 1.1.

Use flxActCommonHandleSetRemoteServer to set the actual address of the FlexNet Operations server. Then use **flxActCommonHandleSetProxyDetails** to direct the connection initially to the proxy server. Typically, the proxy server details are held in a file that can be edited by the enterprise, as the proxy server is under the control of the enterprise.

Optionally the proxy server details can include a user name and password. Some enterprises only allow external access via a proxy server to authenticated users.

Note • *It is recommended that the address of the FlexNet Operations server is* ***not*** *included in any file that contains the proxy server details.*

flxActCommonHandleSetRemoteServer

Stores the address of the activation server to be used for subsequent activation transactions that require a server.

Syntax

```
FlxActBool flxActCommonHandleSetRemoteServer(
    FlxActHandle handle,
    const char *targetServer);
```

Parameters

The **flxActCommonHandleSetRemoteServer** function uses the following parameters:

Table 8-40 • flxActCommonHandleSetRemoteServer Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
targetServer	The address of the activation server. For activation servers that are part of a FlexNet license server the following formats can be used for this parameter: ● port@host ● @host For FlexNet Operations servers the format of this parameter is a full URL, for example http://servername/page.

Return Values

The following values are returned by **flxActCommonHandleSetRemoteServer:**

Table 8-41 • flxActCommonHandleSetRemoteServer Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function stores the address of the activation server to be used for activation transactions that require a server. Use flxActCommonHandleGetRemoteServer to retrieve this address.

This function is used in conjunction with flxActCommonHandleSetCommType, which specifies the type of activation server: FlexNet license server or FlexNet Operations server.

It is recommended that this function is used to set the address of the activation server to be used. When this function is not used or set to NULL, the following defaults addresses are used:

- *FlexNet license server*—@localhost
- *FlexNet Operations server*—http://localhost:8888/flexnet/services/ActivationService?wsdl

See Also
When a proxy server is in use—flxActCommonHandleSetProxyDetails

flxActCommonHandleSetSSLDetails

Sets details of the certificate used for the SSL, (Secure Sockets Layer), protocol used by the SOAP connection.

Syntax

```
FlxActBool flxActCommonHandleSetSSLDetails(
    FlxActHandle handle,
    const char *pCacert,
    const char *pCapath);
```

Parameters

The **flxActCommonHandleSetSSLDetails** function uses the following parameters:

Table 8-42 • flxActCommonHandleSetSSLDetails Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
pCacert	Name of the CA certificate for the open source SSL.
pCapath	Path to the CA certificate for the open source SSL.

Return Values

The following values are returned by **flxActCommonHandleSetSSLDetails:**

Table 8-43 • flxActCommonHandleSetSSLDetails Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before using this function, call flxActCommonHandleSetCommType with commType set to **flxCommsMvsnSoap**. This sets the use of SOAP for connections to the activation server.

The SOAP implementation used by FlexNet Licensing uses OpenSSL on all platforms except Windows. On Windows platforms WinINet is used for SSL and this function is therefore not required on Windows.

On platforms other than Windows, the default is to use HTTP rather than HTTPS. To use HTTPS and add security to the transactions, you must purchase and install an SSL certificate. A client certificate may be required on the machine where the activation utility is running. This is required only when an SSL implementation that requires a client certificate is used on the Operations server. Then the details of this certificate are set up using this function.

For further information, see the Operations server documentation, specifically the *FlexNet Operations Implementation Guide* as it describes how to configure it for SSL.

flxActCommonHandleSetStatusCallback

Sets a callback that reports changes in status of transactions in progress on a SOAP connection.

Syntax

```
FlxActBool flxActCommonHandleSetStatusCallback(
    FlxActHandle handle,
    FLX_STATUS_CALLBACK callback,
    const void *pCallerData);
```

Parameters

The **flxActCommonHandleSetStatusCallback** function uses the following parameters:

Table 8-44 • flxActCommonHandleSetStatusCallback Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
callback	Pointer to the callback function. See FLX_STATUS_CALLBACK for a typedef of this function.
pCallerData	Optional data passed to the callback function. This is input to the callback function as pUserData, see FLX_STATUS_CALLBACK for details.

Return Values

The following values are returned by **flxActCommonHandleSetStatusCallback:**

Table 8-45 • flxActCommonHandleSetStatusCallback Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before using this function call flxActCommonHandleSetCommType with commType set to **flxCommsMvsnSoap**. This sets the use of SOAP for connections to the activation server.

This function is used to set a callback. The callback reports status changes of transactions in progress to a callback function. For information on coding the callback function, see FLX_STATUS_CALLBACK.

Additionally the use of a callback function allows the activation utility to cancel the transaction while it is in progress. The transaction can only be cancelled when a change in status is reported and the callback function is run. For more details, see FLX_STATUS_CALLBACK.

Note • *Read and implement the sequence of operations in Canceling a Return Transaction to ensure that license rights are correctly returned.*

FLX_STATUS_CALLBACK

When **flxActCommonHandleSetStatusCallback** is used within the activation utility code, a callback function must also be provided with the activation utility. This function must be declared to match the definition of **FLX_STATUS_CALLBACK** that follows.

Syntax

```
typedef uint32_t(*FLX_STATUS_CALLBACK)(
    const void *pUserData,
    uint32_t statusOld,
    uint32_t statusNew);
```

Parameters

The **FLX_STATUS_CALLBACK** function uses the following parameters:

Table 8-46 • FLX_STATUS_CALLBACK Parameters

Parameter	Description
pUserData	Optional data passed to the callback function. This is input to **flxActCommonHandleSetStatusCallback** as `pCallerData`.
statusOld	The previous status of the transaction. See flxCommsStatusCodes.
statusNew	The previous status of the transaction. See flxCommsStatusCodes.

Return Values

The return values for the callback function are defined in **flxActCallbackReturn** and are as follows**:**

Table 8-47 • FLX_STATUS_CALLBACK Return Values

Return Value	Description
flxCallbackReturnContinue	Continue transaction
flxCallbackReturnCancelRequest	Cancel transaction

The callback function is called whenever a change in the status of the transaction is detected. Use the callback function to display the status of the transaction to the end user and perform any actions that may be required during a transaction. This can include offering the option to the end user of canceling the transaction.

flxCommsStatusCodes

This defines the status of the transaction as reported by the callback function. Normal transactions have a sequence of callbacks as the request is created, sent and the response returned. The following table lists the status in sequential order where appropriate. The SOAP connection is asynchronous so a response is polled for at intervals. A change of status is indicated when a poll is sent and if the server is still processing the request the status is set to “waiting for response” until the next poll is sent. Thus, items 6 and 7 may be repeated several times.

Table 8-48 • flxCommsStatusCodes

Item	Status Code	Description
1	`flxCommsStatusCreatingRequest`	Initiation of the transaction
2	`flxCommsStatusRequestCreated`	Transaction request created
3	`flxCommsStatusContextCreated`	SOAP context created
4	`flxCommsStatusConnected`	SOAP connection to server established
5	`flxCommsStatusRequestSent`	Transaction request sent
6	`flxCommsStatusPolingForResponse`	Polling server for response
7	`flxCommsStatusWaitingForResponse`	Server response to poll that it is still processing the request
8	`flxCommsStatusDone`	Transaction completed
n/a	`flxCommsStatusError`	An error has occurred during the transaction request
n/a	`flxCommsStatusSuccess`	For internal use only
n/a	`flxCommsStatusNotSet`	For internal use only
n/a	`flxCommsStatusCancelledByUser`	Transaction has been cancelled by the end user

Canceling a Return Transaction

This section describes the sequence of operations in a return transaction and indicates when canceling the transaction must be disabled to ensure that the license rights are returned correctly.

The end-user is allowed to cancel the transaction at any point before the transaction request is reported as sent to the FlexNet Operations server. After this point the license rights may have been written into the FlexNet Operations server so the end-user must not be allowed to cancel the transaction. The license rights are only deleted from the local trusted storage when a success response is received from the FlexNet Operations server. The following table indicates when the end-user cancel, **flxCallbackReturnCancelRequest**, must not be used.

Table 8-49 • Using flxCallbackReturnCancelRequest during Return Transactions

Status Code	Allow flxCallbackReturnCancelRequest
flxCommsStatusCreatingRequest	Yes
flxCommsStatusRequestCreated	Yes
flxCommsStatusContextCreated	Yes
flxCommsStatusConnected	Yes
flxCommsStatusRequestSent	No
flxCommsStatusPolingForResponse	No
flxCommsStatusWaitingForResponse	No
flxCommsStatusDone	No
flxCommsStatusError	Yes
flxCommsStatusSuccess	No
flxCommsStatusNotSet	No
flxCommsStatusCancelledByUser	Not applicable - already cancelled

flxActCommonHandleSetTimeout

Sets timeout for Soap communications.

Syntax

```
FlxActBool flxActCommonHandleSetTimeout(
    FlxActHandle handle,
    uint32_t timeout);
```

Parameters

The **flxActCommonHandleSetTimeout** function uses the following parameters:

Table 8-50 • flxActCommonHandleSetTimeout Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
timeout	Timeout in seconds for the server to wait before closing an idle connection. When this value is set to 0, zero, then the activation server uses its own timeout.

Return Values

The following values are returned by **flxActCommonHandleSetTimeout:**

Table 8-51 • flxActCommonHandleSetTimeout Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before using this function call flxActCommonHandleSetCommType with commType set to **flxCommsMvsnSoap**. This sets the use of SOAP for connections to the activation server.

Use this optional function to set the timeout on the SOAP connection to the activation server. The default when this function is not used is to set the timeout to 0, zero, when the activation server uses its own timeout.

flxActCommonLibraryCleanup

Releases the activation library.

Syntax

```
FlxActBool flxActCommonLibraryCleanup();
```

Return Values

The following values are returned by **flxActCommonLibraryCleanup:**

Table 8-52 • flxActCommonLibraryCleanup Return Values

Return Value	Description
FLX_ACT_TRUE	Success

Table 8-52 • flxActCommonLibraryCleanup Return Values

Return Value	Description
FLX_ACT_FALSE	Non-success

Discussion

This function should be called only if a previous call to flxActCommonLibraryInit succeeded.

Call **flxActCommonLibraryCleanup** to shutdown the Activation Library. No other Activation API routines can be called after this point.

Caution • *This function is not multi-thread safe: synchronization between threads is the responsibility of the application code. There is an additional restriction on platforms other than Windows and Mac: this function must be called in the main thread of any multi-threaded applications.*

Note • *On Windows platforms, do not call* **flxActCommonLibraryCleanup** *in* `DllMain`*. This API results in calls to the Microsoft Win32 FreeLibrary and LoadLibrary APIs. Calls to these Win32 APIs in* `DllMain` *are widely known to be associated with issues.*

See Also
flxActCommonLibraryInit
lc_flexinit_cleanup

flxActCommonLibraryInit

Initialization required to use the activation library and the Activation API.

Note • *The activation library is known as the* run-time component *enabled by* preptool *for an application that uses license-file licensing and contains Secure Data Types (SDTs). If you want to use the Activation API and the Flexible API within a single executable, ensure that you call either flxActCommonLibraryInit or lc_flexinit, but not both.*

Syntax

```
int flxActCommonLibraryInit(const char *runtimepath);
```

Parameters

The **flxActCommonLibraryInit** function uses the following parameters.

Table 8-53 • flxActCommonLibraryInit Parameters

Parameter	Description
runtimepath	The path and file name for the activation library. Specify a fixed or relative path to override the default path. When set to `NULL`, the default value is used. ● Standard toolkits—The default path is specified in the `preptool` configuration file as `runtimeloadpath`. (For details about the preptool configuration file, see the following Discussion section and the *Programming Reference for Trusted Storage–Based Licensing*.) ● notr toolkits—The default path is specified using the *`Libpath`* environment variable. **Mac OS X–Specific Formats** The following are Mac OS X–specific formats for `runtimePath`: ● *`@pathname/../`*`MacOS`—Load tamper-resistant run-time executable from the /Contents/MacOS directory of a standard OS X application package where *`pathname`* is the path and name of the application package. ● *`@pathname/./`*`Resources`—Load tamper-resistant run-time executable from the /Contents/Resources directory of a standard OS X application package where *`pathname`* is the path and name of the application package. ● *`@com.mycompanyname.mybundle/./`*`MacOS`—Load tamper-resistant run-time executable from the `/Contents/Resources` directory of a loaded OS X bundle, where *`com.mycompanyname.mybundle`* specifies the OS X bundle.

Return Values

The following values are returned by **flxActCommonLibraryInit**:

Table 8-54 • flxActCommonLibraryInit Return Values

Return Value	Description
0	Success
1	Activation library was not found at the path and name specified. This is likely to be because the path has been incorrectly specified either in the call to **flxActCommonLibraryInit**, in the preptool configuration file, or in the *`Libpath`* environment variable.
2	Activation library could not be loaded. This may be because the file could not be opened, or is corrupted in some way.
3	The product has been installed on a platform that is not supported.
4	Initialization of activation library failed.

Table 8-54 • flxActCommonLibraryInit Return Values

Return Value	Description
5	Unable to allocate resources.
6	Activation application has not been processed using the preptool, or the activation component library (*`application`*_libFNP) for the activation application cannot be found.
7	Activation library has already been loaded.
8	Mac specific error: path string was not in UTF-8 format.
9	Mac specific error: a URL could not be created from the path string. Probably caused by an invalid *pathname*.
10	Mac specific error: a path could not be created from the URL. Probably caused by an invalid *pathname*.
11	Mac-specific error: Incorrect bundle identifier: *`com.mycompanyname.mybundle`*
12	Mac-specific error: Invalid bundle identifier: *`com.mycompanyname.mybundle`*
13	Mac-specific error: The computed path exceeds the maximum allowed on this platform—1024 characters on most platforms.
20	The FlexNet Licensing Service is not installed.
21	Not enough rights to talk to the FlexNet Licensing Service. Set the service to start automatically to resolve this.
22	The FlexNet Licensing Service configuration or dependency issue.
23	Attempt to access server TS from client app, or vice-versa.
24	Insufficient privilege to complete initialization.

Discussion

Before an application can use this function, the FlexNet Licensing Service must be installed (see the following table). Normally, installing this service is part of the activation utility installation on the machine. This combined installation ensures that the activation utility can load the activation library the first time it is run. See Chapter 7, FlexNet Licensing Service API for details. On other platforms, a script is provided, see the *Development Environment Guide* for details.

Table 8-55 • Prerequisites for Loading the Activation Library

Platform	Install	Description
Windows	`installanchorservice.exe`	Example application provided in the FlexNet Publisher Licensing Toolkit that installs the FlexNet Licensing Service. Create your own application or include code within your installer using the FlexNet Licensing Service API.
Mac	`install_fnp.sh`	Script that installs the FlexNet Licensing Service.
Linux	`install_fnp.sh`	Script that installs the FlexNet Licensing Service.

This function can run only if the activation library is accessible within the product package. The activation library provides access to trusted storage. For standard toolkits, the activation library provides tamper resistance in conjunction with the use of Secure Data Types (SDTs).

flxActCommonLibraryInit loads the activation library. This function must be called once and only once by the activation utility at startup.

The Flexible API function lc_flexinit can also be used to load the activation library and is provided for use within a FlexEnabled application. If you are using the Activation API and the Flexible API within a single executable, ensure that you call either **flxActCommonLibraryInit** or **lc_flexinit,** but not both.

This function should be paired with a call to flxActCommonLibraryCleanup to shut down the activation library.

For standard toolkits, this function uses the following settings in the preptool configuration file:

- `runtimeloadname`—mandatory entry that defines the name of the activation library.
- `runtimeloadpath`—optional entry that defines the path to the activation library. This parameter is used by **flxActCommonLibraryInit** when `runtimepath` is set to NULL.
- `installpath`—optional entry that defines the path to the directory where the activation library is created when the project is built.

Ensure that where a fixed path is specified for `runtimeloadpath`, it is accurate for all machines and installations.

Caution • *This function is not multithread safe; synchronization between threads is the responsibility of the application code.*

Note • *On Windows platforms, it is recommended that* **flxActCommonLibraryInit** *not be called from DllMain. Additionally, do not call the matching uninitialize call,* *flxActCommonLibraryCleanup***,** *from DllMain.*

See Also
flxActCommonLibraryCleanup

flxActCommonLicSpcAddASRFromBuffer

Adds the fulfillment record from the supplied ASR to trusted storage.

Syntax

```
FlxActBool flxActCommonLicSpcAddASRFromBuffer(
    FlxActLicSpc licSpc,
    const char *pszBuffer);
```

Parameters

The **flxActCommonLicSpcAddASRFromBuffer** function uses the following parameters:

Table 8-56 • flxActCommonLicSpcAddASRFromBuffer Parameters

Parameter	Description
licSpc	A license specification creating using flxActCommonLicSpcCreate.
pszBuffer	A buffer that contains the ASR that is to be loaded into the license specification and then written to trusted storage.

Return Values

The following values are returned by **flxActCommonLicSpcAddASRFromBuffer:**

Table 8-57 • flxActCommonLicSpcAddASRFromBuffer Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function create a license specification. See flxActCommonLicSpcCreate for a description of a license specification and how its creation process defines how it operates. Then add the fulfillment records in trusted storage by calling flxActCommonLicSpcPopulateFromTS or flxActCommonLicSpcPopulateAllFromTS.

This functions reads the ASR in the specified buffer and the following steps are carried out:

- The ASR is examined to determine if it is for use with short code transactions. If it is for a short code transaction, then no further processing is performed and an error is returned.
- Entries in the `DataDictionary` element of the ASR are read.

- If publisher rules for local activation are defined, the rules are applied. See the *Programming Reference for Trusted Storage–Based Licensing* for general information about activation specification records (ASRs). See Trial Licensing for information on how these publisher rules are applied by this function. When the ASR does not comply with the rules, the fulfillment record is not added to the license specification.
- If a duration is defined, the expiry date is calculated and the fulfillment record is modified.

When the ASR has been processed, the fulfillment records in the license specification are written to trusted storage.

Note • *This function only processes a single ASR: do not put multiple ASRs in the buffer.*

Note • *A short code ASR as defined by the operation of this function is one with a* `ShortCodeData` *element that has at the minimum a SchemeId element. Ensure that ASRs used for local activation do not include a* `ShortCodeData` *element.*

For example code that uses this function, see **sHandleLocalActivation** in the example activation-utility code files, `appActUtil.c`. and `serverActUtil.c`, included in the toolkit.

flxActCommonLicSpcAddASRs

Adds the fulfillment records from valid ASRs to trusted storage.

Syntax

```
FlxActBool flxActCommonLicSpcAddASRs(
    FlxActLicSpc licSpc,
    const char *filePath);
```

Parameters

The **flxActCommonLicSpcAddASRs** function uses the following parameters:

Table 8-58 • flxActCommonLicSpcAddASRs Parameters

Parameter	Description
licSpc	A license specification creating using flxActCommonLicSpcCreate.
filePath	The location of the ASRs that are to be loaded into the license specification and then written to trusted storage. This can be either of the following: • The path to a directory that contains the ASRs • The path and file name of an ASR

Return Values

The following values are returned by **flxActCommonLicSpcAddASRs:**

Table 8-59 • flxActCommonLicSpcAddASRs Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function create a license specification. See flxActCommonLicSpcCreate for a description of a license specification and how its creation process defines how it operates. Then add the fulfillment records in trusted storage by calling flxActCommonLicSpcPopulateFromTS or flxActCommonLicSpcPopulateAllFromTS.

This function is used for local activation; it cannot be used with ASRs used for short code activation.

This functions reads each of the ASRs in the specified directory, or a single ASR file if a filename is included in `filePath`, and the following steps are carried out:

1. The ASR is examined to determine if it is for use with short code transactions. If it is for a short code transaction, then no further processing of ASRs is performed and an error is returned. The fulfillment records in this and any other ASRs are not written to trusted storage.
2. Entries in the `DataDictionary` element of the ASR are read.
3. If publisher rules for local activation are defined, the rules are applied. See information about activation specification records (ASR) in the *Programming Reference for Trusted Storage–Based Licensing*. See also Trial Licensing. When the ASR does not comply with the rules, the fulfillment record is not added to the license specification.
4. If a duration is defined, the expiry date is calculated and added to the fulfillment record.

When all the ASRs have been processed, the fulfillment records in the license specification are written to trusted storage.

Note • *A short code ASR as defined by the operation of this function is one with a* `ShortCodeData` *element that has at the minimum a SchemeId element. Ensure that ASRs used for local activation do not include a* `ShortCodeData` *element.*

Trial Licensing

This function is commonly used to allow an end user to try out a product for a limited time period before purchasing a full license. This scenario is referred to as *trial licensing*. In this case the ASR has the following characteristics:

- **Can only be used once to install a Fulfillment Record** - `DataDictionary` element PUBLISHER_DEFINED_CRITERION set to LOCAL_ONCE.
- **Limited duration license** - `DataDictionary` element PUBLISHER_DEFINED_DURATION set to a positive value.

When using an ASR for trial licensing, it is recommended that a trial anchor is used. When a trial anchor is not specified in the ASR a publisher anchor will be used. Using a publisher anchor to anchor trial licenses results in further trials being disallowed when trusted storage has been deleted and recreated. The following sections describe in detail the functionality provided by Trial Anchors and Publisher Anchors when used by this function in the context of trial licensing.

Trial Anchor

A trial anchor stores information on a machine about a trial. This information remains after trusted storage is deleted. It therefore provides for:

- Installation of subsequent trial licenses after trusted storage is deleted.
- Recalculation of trial duration during re installation of a trial product after deletion of trusted storage. For example, a product is installed with a trial license of 30 days and then trusted storage is mistakenly deleted. When the user reinstalls the product 10 days after the initial installation, the remaining license duration (20 days) is calculated and used when processing the ASR (see step 4).

Publisher Anchor

It is recommended that you do not use a publisher anchor for trial licensing: use a trial anchor instead.

The publisher anchor stores information about the creation and recreation of trusted storage. When the publisher anchor records that trusted storage has been deleted or recreated, there are no complete records of previous licenses available. Therefore, when the publisher rules within an ASR specify that the license is to be installed only once, and the publisher anchor records that trusted storage has been deleted, the fulfillment record within the ASR is not written to trusted storage.

flxActCommonLicSpcCheckASR

Checks whether an ASR has already been added to trusted storage.

Syntax

```
FlxActBool flxActCommonLicSpcCheckASR(
    FlxActLicSpc licSpc,
    const char *pszFilePath,
    uint32_t *pTrialId,
    time_t *pStartTime);
```

Parameters

The **flxActCommonLicSpcCheckASR** function uses the following parameters:

Table 8-60 • flxActCommonLicSpcCheckASR Parameters

Parameter	Description
licSpc	A license specification creating using flxActCommonLicSpcCreate.
filePath	The location of the ASRs that are to be loaded into the license specification and then written to trusted storage. The path of the file containing an ASR.
pTrialId	Returns the TrialId value of the ASR if successful. If there is no TrialId, 0 (zero) is returned.
pStartTime	If the ASR specifies a Trial Id and it has been loaded previously, the Universal Coordinated Time (UTC) of midnight of the day on which the anchor associated with the Trial Id was written is returned. If the anchor is not present, 0 (zero) is returned.

Return Values

The following values are returned by **flxActCommonLicSpcCheckASR:**

Table 8-61 • flxActCommonLicSpcCheckASR Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error. The possible errors returns include: `LM_TSSE_ASR_CHECK_IS_DIR`—pszFilePath is a directory; must specify a file `LM_TSSE_ASR_PATH`—Can't open pszFilePath `LM_TSSE_ASR_NOT_XML`—Not an ASR (not valid XML) `LM_TSSE_ASR_NOT_ASR`—Not an ASR (wrong XML root name) `LM_TSSE_ASR_XML_ELEMENT_MISSING`—Not a valid ASR (element missing) `LM_TSSE_ASR_SIGNATURE`—ASR signature is invalid `LM_TSSE_ASR_WRONG_SERVER_MODE`—ASR has wrong server mode

Discussion

Enables to check if a trial ASR had already been loaded and exists in the trusted storage, a **FlxActProdLicSpec** containing its fulfillment information is added to the supplied `FlxActLicSpc`, other than by attempting to load the ASR and receiving the `LM_TS_ASR_LOAD_ONCE` error. It can be confusing for an end-user if the producer application offers a trial ASR and then tells the end-user that the trial ASR has already been consumed. It is better not to offer the trial ASR.

flxActCommonLicSpcCheckASRFromBuffer

Checks whether an ASR has already been added to trusted storage.

Syntax

```
FlxActBool flxActCommonLicSpcCheckASRFromBuffer(
    FlxActLicSpc licSpc,
    const char *pszAsrxml,
    uint32_t *pTrialId,
    time_t *pStartTime);
```

Parameters

The **flxActCommonLicSpcCheckASRFromBuffer** function uses the following parameters:

Table 8-62 • flxActCommonLicSpcAddASRFromBuffer Parameters

Parameter	Description
licSpc	A license specification creating using flxActCommonLicSpcCreate.
Asrxml	The buffer which stores the ASR.
pTrialId	Returns the TrialId value of the ASR if successful. If there is no TrialId, 0 (zero) is returned..
pStartTime	If the ASR specifies a Trial Id and it has been loaded previously, the Universal Coordinated Time (UTC) of midnight of the day on which the anchor associated with the Trial Id was written is returned. If the anchor is not present, 0 (zero) is returned.

Return Values

The following values are returned by **flxActCommonLicSpcCheckASRFromBuffer:**

Table 8-63 • flxActCommonLicSpcCheckASRFromBuffer Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error. The Possible error returns include: `LM_TSSE_ASR_NOT_XML`—Not an ASR (not valid XML) `LM_TSSE_ASR_NOT_ASR`—Not an ASR (wrong XML root name) `LM_TSSE_ASR_XML_ELEMENT_MISSING`—Not a valid ASR (element missing) `LM_TSSE_ASR_SIGNATURE`—ASR signature is invalid `LM_TSSE_ASR_WRONG_SERVER_MODE`—ASR has wrong server mode

Discussion

Enables to check if a trial ASR had already been loaded, other than by attempting to load the ASR and receiving the `LM_TS_ASR_LOAD_ONCE` error. It can be confusing for an end-user if the producer application offers a trial ASR and then tells the end-user that the trial ASR has already been consumed. It is better not to offer the trial ASR.

flxActCommonLicSpcCreate

Creates a license specification. This is a structure that is used to contain and manipulate fulfillment records. A license specification may hold multiple fulfillment records: an individual fulfillment record is contained in a product license specification.

Syntax

```
FlxActBool flxActCommonLicSpcCreate(
    FlxActHandle handle,
    FlxActLicSpc *licSpc);
```

Parameters

The **flxActCommonLicSpcCreate** function uses the following parameters:

Table 8-64 • flxActCommonLicSpcCreate Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
licSpc	The resulting handle to the license specification.

Return Values

The following values are returned by **flxActCommonLicSpcCreate:**

Table 8-65 • flxActCommonLicSpcCreate Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function the following is required:

- Load the Activation Library (flxActCommonLibraryInit)
- Open an activation handle (flxActCommonHandleOpen)

The *mode* used when creating this handle defines whether fulfillment records in the license specification are held in client or server mode trusted storage.

This function creates a license specification. This license specification is then used with various functions that are used to load and manipulate fulfillment records.

Use the following functions to read and write fulfillment records to and from local trusted storage:

- flxActCommonLicSpcPopulateFromTS
- flxActCommonLicSpcPopulateAllFromTS
- flxActCommonLicSpcAddASRs
- flxActCommonLicSpcAddASRFromBuffer
- flxActCommonLicSpcProductDelete

Use the following functions to read fulfillment-records from remote server-side trusted storage:

- flxActCommonLicSpcPopulateFromServerTS
- flxActCommonLicSpcPopulateAllFromServerTS
- flxActCommonProdLicSpcPopulateFRFromServerTS

Use the following functions to access the fulfillment records (product license specifications) in the license specification:

- flxActCommonLicSpcGetNumberProducts
- flxActCommonLicSpcGet
- flxActCommonLicSpcGetNumProdEntries
- flxActCommonLicSpcGetProd

Note • *Any changes to fulfillment records in trusted storage as the result of using the Activation API are not reflected in any pre-existing license specification objects. After updating trusted storage, delete an existing license specification object and create a new license specification object to obtain up to date data from trusted storage.*

Use flxActCommonLicSpcDelete to remove the license specification and free all the resources associated with it.

flxActCommonLicSpcDelete

Deletes the specified license specification. All resources associated with the license specification, including any fulfillment records it contains, are freed.

Syntax

```
void flxActCommonLicSpcDelete(FlxActLicSpc licSpc);
```

Parameters

The **flxActCommonLicSpcDelete** function uses the following parameters:

Table 8-66 • flxActCommonLicSpcDelete Parameters

Parameter	Description
licSpc	The resulting handle to the license specification.

Return Values

There is no return value.

Discussion

This function should be paired with a matching successful call to flxActCommonLicSpcCreate. All resources associated with the license specification, including any fulfillment records it contains, are freed.

flxActCommonLicSpcGet

Gets the fulfillment record at the specified index from the license specification.

Syntax

```
FlxActProdLicSpc flxActCommonLicSpcGet(
    FlxActLicSpc licSpc,
    uint32_t index);
```

Parameters

The **flxActCommonLicSpcGet** function uses the following parameters:

Table 8-67 • flxActCommonLicSpcGet Parameters

Parameter	Description
licSpc	A license specification creating using flxActCommonLicSpcCreate.
index	The position of the desired fulfillment record where index is in the range 0 through count-1, with 0 being the first fulfillment record. Use flxActCommonLicSpcGetNumberProducts to determine count

Return Values

If successful, the fulfillment record is returned in a product license specification object. If not successful, NULL is returned.

Discussion

This function is used in conjunction with flxActCommonLicSpcGetNumProducts. For a full description, see flxActCommonLicSpcGetNumberProducts.

flxActCommonLicSpcGetNumberProducts

Returns the number of fulfillment records in the license specification.

Syntax

```
uint32_t flxActCommonLicSpcGetNumberProducts(FlxActLicSpc licSpc);
```

Parameters

The **flxActCommonLicSpcGetNumberProducts** function uses the following parameters:

Table 8-68 • flxActCommonLicSpcGetNumberProducts Parameters

Parameter	Description
licSpc	A license specification creating using flxActCommonLicSpcCreate.

Return Values

The number of fulfillment records in the license specification.

Discussion

This function is used together with flxActCommonLicSpcGet to access the fulfillment records in the license specification. Then, the **flxActCommonProdLicSpc*** functions are used to access data inside a fulfillment record. The basic procedure is as follows:

1. Obtain the number of fulfillment records—flxActCommonLicSpcGetNumberProducts.
2. Examine a fulfillment record in the license specification—use flxActCommonLicSpcGet to load a fulfillment record from the license specification.
3. Use **flxActCommonProdLicSpc*Get** to obtain the value of a parameter in the fulfillment record.
4. Repeat Step 2 and Step 3 for subsequent fulfillment records in the license specification—loop based on the number of fulfillment records returned using flxActCommonLicSpcGetNumberProducts.

flxActCommonLicSpcGetNumProdEntries

Returns the number of fulfillment records in the license specification container for the specified product.

Syntax

```
uint32_t flxActCommonLicSpcGetNumProdEntries(
    FlxActLicSpc licSpc,
    const char *productId);
```

Parameters

The **flxActCommonLicSpcGetNumProdEntries** function uses the following parameters:

Table 8-69 • flxActCommonLicSpcGetNumProdEntries Parameters

Parameter	Description
licSpc	A license specification creating using flxActCommonLicSpcCreate.
productId	A product ID that identifies the product for which the number of fulfillment records is required.

Return Values

The number of fulfillment records in the license specification that contain a matching product ID (that are for the specified product).

Discussion

This function is used together with flxActCommonLicSpcGetProd to access the fulfillment records in the license specification that have a product ID that matches the product ID specified in the call to the function. Then the **flxActCommonProdLicSpc*** functions are used to access data inside a fulfillment record. The basic procedure is as follows.

1. Obtain the number of fulfillment records—use flxActCommonLicSpcGetNumProdEntries.
2. Examine a fulfillment record in the license specification—use flxActCommonLicSpcGetProd to load a fulfillment record from the license specification.
3. Use **flxActCommonProdLicSpc*Get** to obtain the value of a parameter in the fulfillment record.
4. Repeat Step 2 and Step 3 for subsequent fulfillment records in the license specification—loop based on the number of fulfillment records returned using flxActCommonLicSpcGetNumProdEntries.

flxActCommonLicSpcGetProd

Get the fulfillment record at the specified index for the specified product from the license specification.

Syntax

```
FlxProdLicSpc flxActCommonLicSpcGetProd(
    FlxActActLicSpc licSpc,
    const char *productId,
    uint32_t index);
```

Parameters

The **flxActCommonLicSpcGetProd** function uses the following parameters:

Table 8-70 • flxActCommonLicSpcGetProd Parameters

Parameter	Description
licSpc	A license specification creating using flxActCommonLicSpcCreate.
productId	A product ID that identifies the product for which the fulfillment record is required.
index	The position of the desired fulfillment record, where index is in the range 0 through count-1, with 0 being the first object for productId. Use flxActCommonLicSpcGetNumProdEntries to determine count.

Return Values

If successful, the fulfillment record is returned in a product license specification object. If not successful, NULL is returned.

Discussion

This function is used in conjunction with **flxActCommonLicSpcGetNumProducts**. For a full description, see flxActCommonLicSpcGetNumProdEntries.

flxActCommonLicSpcPopulateAllFromServerTS

Retrieves all fulfillment records (trusted, non-trusted, expired, non-expired) from server-side trusted storage into a license specification structure.

Note • *The use of this function is limited to the retrieval of fulfillment information in server-side trusted-storage only; it cannot be used to retrieve fulfillment information stored in the client trusted storage.*

Syntax

```
FlxActBool flxActCommonLicSpcPopulateAllFromServerTS(
    FlxActLicSpc licSpc,
    const char *pszEntitlementID,
    const char *pszProductID,
    const char *pszServer);
```

Parameters

The **flxActCommonLicSpcPopulateAllFromServerTS** function uses the following parameters:

Table 8-71 • flxActCommonLicSpcPopulateAllFromServerTS Parameters

Parameter	Description
licSpc	The license specification object that is created using flxActCommonLicSpcCreate.
pszEntitlementID	The pointer variable storing the value of provided entitlement ID. • If this parameter is specified, only fulfillment records belonging to the specified entitlement ID in server-side trusted storage are retrieved into the license specification structure. • If this parameter is `NULL` or an empty string, all fulfillment records from server-trusted storage are retrieved into license specification structure. • If parameters are specified for both the entitlement ID and product ID, fulfillment records that match both filters are retrieved into the license specification structure.
pszProductID	The pointer variable storing the value of provided product ID. • If this parameter is specified, only fulfillment records belonging to the specified product ID in server-side trusted storage are retrieved into the license specification structure. • If this parameter is `NULL` or an empty string, all fulfillment records from server-side trusted storage are retrieved into license specification structure. • If parameters are specified for both the entitlement ID and product ID, fulfillment records that match both filters are retrieved into the license specification structure.

Table 8-71 • flxActCommonLicSpcPopulateAllFromServerTS Parameters

Parameter	Description
pszServer	A pointer to the variable storing the address of the license server from which the counts are retrieved. The following formats can be used for this parameter: ● *port@host* ● *@host*

Return Values

The following values are returned by **flxActCommonLicSpcPopulateAllFromServerTS**:

Table 8-72 • flxActCommonLicSpcPopulateAllFromServerTS Parameters Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to determine the specific error. Possible errors include: ● `LM_TS_BADPARAM`—One or more of the parameters passed in are invalid. ● `LM_TS_CANTMALLOC`—Insufficient memory has caused an error. ● `LM_TS_INVALID_REQUEST_TYPE`—The activation server does not support this request type. (This error normally occurs if the request is corrupt.) ● `LM_TS_SERVERQUERY_GEN_REQ`—The request to query remote server-side trusted storage cannot be generated. ● `LM_TS_UNKNOWN_RESP`—The response is corrupt. ● `LM_TS_SERVERQUERY_RESPONSE_FAILED`—The response to query remote server-side trusted storage cannot be processed.

Discussion

flxActCommonLicSpcCreate must be called before this function to create a license specification. See flxActCommonLicSpcCreate for information about this process. (**flxActCommonLicSpcPopulateAllFromServerTS** can use a handle created in either `FLX_ACT_APP` or `FLX_ACT_SVR` mode.)

This function can be used in conjunction with these other APIs to retrieve additional information about fulfillments in server-side trusted storage:

- flxActCommonGetFRCountFromServerTS
- flxActCommonProdLicSpcPopulateFRFromServerTS

Important • *The remote license server whose trusted storage you want to read must be running.*

Extracted Fulfillment Information

When this API is used, the following fields in the product license specification are populated:

- `EntitlementId`
- `ProductId`
- `ActServerChainGet`
- `Expiry`
- `FulfillmentType`
- `CONCURRENT`, `CONCURRENT_OD`, `ACTIVATABLE`, `ACTIVATABLE_OD`, `HYBRID`, `HYBRID_OD`, and `REPAIR` total counts

The count information retrieved with this API reflects the total count for the given count type. To obtain the actual *available* counts (as well as additional fulfillment-record information, such as FEATURE lines), use the flxActCommonProdLicSpcPopulateFRFromServerTS API.

Performance

The frequent use of this API can cause client connections to experience timeouts due to the license server's being busy in processing this request. Therefore, use this API sparingly. The API response time mainly depends on the total number of fulfillment records for the specified entitlement ID and productID in the server-side trusted storage.

As a best practice, do the following:

- Always use the filter conditions *`entitlementID`* or *`productID`* or both to obtain the desired fulfillment records.
- Use the flxActCommonLicSpcPopulateFromServerTS API instead of this API (flxActCommonLicSpcPopulateAllFromServerTS) for the retrieval of trusted fulfillment records.

flxActCommonLicSpcPopulateAllFromTS

Loads all fulfillment records; expired and non-expired; valid and not valid, from trusted storage into the specified license specification.

Syntax

```
FlxActBool flxActCommonLicSpcPopulateAllFromTS(FlxActLicSpc licSpc);
```

Parameters

The **flxActCommonLicSpcPopulateAllFromTS** function uses the following parameters:

Table 8-73 • flxActCommonLicSpcPopulateAllFromTS Parameters

Parameter	Description
licSpc	A license specification creating using flxActCommonLicSpcCreate.

Return Values

The following values are returned by **flxActCommonLicSpcPopulateAllFromTS:**

Table 8-74 • flxActCommonLicSpcPopulateAllFromTS Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function a license specification must be created using flxActCommonLicSpcCreate. The parameters used in this process define from which trusted storage the fulfillment records are read.

This function loads fulfillment records from trusted storage into the license specification supplied.

flxActCommonLicSpcPopulateFromServerTS

Retrieves valid fulfillment records (expired and non-expired) from remote server-side trusted storage into a license specification structure. (A fulfillment record is valid if its trust flags are set as trusted.)

Note • *The use of this function is limited to the retrieval of fulfillment information in server-side trusted-storage only; it cannot be used to retrieve fulfillment information stored in the client trusted storage.*

Syntax

```
FlxActBool flxActCommonLicSpcPopulateFromServerTS (
    FlxActLicSpc licSpc,
    const char *pszEntitlementID,
    const char *pszProductID,
    const char *pszServer);
```

Parameters

The **flxActCommonLicSpcPopulateFromServerTS** function uses the following parameters:

Table 8-75 • flxActCommonLicSpcPopulateFromServerTS Parameters

Parameter	Description
licSpc	The license specification object that is created using flxActCommonLicSpcCreate.

Table 8-75 • flxActCommonLicSpcPopulateFromServerTS Parameters

Parameter	Description
pszEntitlementID	The pointer variable storing the value of provided entitlement ID. ● If this parameter is specified, only fulfillment records belonging to the specified entitlement ID in server-side trusted storage are retrieved into the license specification structure. ● If this parameter is `NULL` or an empty string, all fulfillment records from server-trusted storage are retrieved into license specification structure. ● If parameters are specified for both the entitlement ID and product ID, fulfillment records that match both filters are retrieved into the license specification structure.
pszProductID	The pointer variable storing the value of provided product ID. ● If this parameter is specified, only fulfillment records belonging to the specified product ID in server-side trusted storage are retrieved into the license specification structure. ● If this parameter is `NULL` or an empty string, all fulfillment records from server-side trusted storage are retrieved into license specification structure. ● If parameters are specified for both the entitlement ID and product ID, fulfillment records that match both filters are retrieved into the license specification structure.
pszServer	A pointer to the variable storing the address of the license server from which the counts are retrieved. The following formats can be used for this parameter: ● *port@host* ● *@host*

Return Values

The following values are returned by **flxActCommonLicSpcPopulateFromServerTS**:

Table 8-76 • flxActCommonLicSpcPopulateFromServerTS Parameters Return Values

Return Value	Description
FLX_ACT_TRUE	Success

Table 8-76 • flxActCommonLicSpcPopulateFromServerTS Parameters Return Values

Return Value	Description
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to determine the specific error. Possible errors include: • `LM_TS_BADPARAM`—One or more of the parameters passed in are invalid. • `LM_TS_CANTMALLOC`—Insufficient memory has caused an error. • `LM_TS_INVALID_REQUEST_TYPE`—The activation server does not support this request type. (This error normally occurs if the request is corrupt.) • `LM_TS_SERVERQUERY_GEN_REQ`—The request to query remote server-side trusted storage cannot be generated. • `LM_TS_UNKNOWN_RESP`—The response is corrupt. • `LM_TS_SERVERQUERY_RESPONSE_FAILED`—The response to query remote server-side trusted storage cannot be processed.

Discussion

flxActCommonLicSpcPopulateFromServerTS loads valid fulfillment records from server-side trusted storage into the license specification supplied. A fulfillment record is valid if all its trust flags are set as trusted. For details of the trust flags, see flxActCommonProdLicSpcTrustFlagsGet.

flxActCommonLicSpcCreate must be called before this function to create a license specification. See flxActCommonLicSpcCreate for information about this process. (**flxActCommonPopulateAllFromServerTS** can use a handle created in either `FLX_ACT_APP` or `FLX_ACT_SVR` mode.)

This function can be used in conjunction with these other APIs to retrieve additional information about fulfillments in server-side trusted storage:

- flxActCommonGetFRCountFromServerTS
- flxActCommonProdLicSpcPopulateFRFromServerTS

Important • *The remote license server whose trusted storage you want to read must be running.*

Extracted Fulfillment Information

When this API is used, the following fields in the product license specification are populated:

- `EntitlementId`
- `ProductId`
- `ActServerChainGet`
- `Expiry`
- `FulfillmentType`
- `CONCURRENT`, `CONCURRENT_OD`, `ACTIVATABLE`, `ACTIVATABLE_OD`, `HYBRID`, `HYBRID_OD`, and `REPAIR` total counts

The count information retrieved with this API reflects the total count for the given count type. To obtain the actual *available* counts (as well as additional fulfillment-record information, such as FEATURE lines), use the flxActCommonProdLicSpcPopulateFRFromServerTS API.

Performance

The frequent use of this API can cause client connections to experience timeouts due to the license server's being busy in processing this request. Therefore, use this API sparingly. The API response time mainly depends on the total number of fulfillment records for the specified entitlement ID and productID in the server-side trusted storage.

As a best practice, do the following:

- Always use the filter conditions *`entitlementID`* or *`productID`* or both to obtain the desired fulfillment records.
- Use this API (flxActCommonLicSpcPopulateFromServerTS) instead of the flxActCommonLicSpcPopulateAllFromServerTS API for the retrieval of trusted fulfillment records.

flxActCommonLicSpcPopulateFromTS

Loads valid fulfillment records from trusted storage into the license specification.

Syntax

```
FlxActBool flxActCommonLicSpcPopulateFromTS(FlxActLicSpc licSpc);
```

Parameters

The **flxActCommonLicSpcPopulateFromTS** function uses the following parameters:

Table 8-77 • flxActCommonLicSpcPopulateFromTS Parameters

Parameter	Description
licSpc	A license specification creating using flxActCommonLicSpcCreate.

Return Values

The following values are returned by **flxActCommonLicSpcPopulateFromTS:**

Table 8-78 • flxActCommonLicSpcPopulateFromTS Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function a license specification must be created using **flxActCommonLicSpcCreate**. See flxActCommonLicSpcCreate for information about this process. The parameters used in this process define from which trusted storage the fulfillment records are read.

This function loads valid fulfillment records from trusted storage into the license specification supplied. A fulfillment record is valid if all its trust flags are set as trusted. For details of the trust flags see flxActCommonProdLicSpcTrustFlagsGet.

flxActCommonLicSpcProductDelete

Deletes the specified fulfillment record from trusted storage.

Syntax

```
FlxActBool flxActCommonLicSpcProductDelete(
    FlxActLicSpc licSpc,
    FlxActProdLicSpc product);
```

Parameters

The **flxActCommonLicSpcProductDelete** function uses the following parameters:

Table 8-79 • flxActCommonLicSpcProductDelete Parameters

Parameter	Description
licSpc	A license specification created with **flxActCommonLicSpcCreate** that contains fulfillment records written from trusted storage by calling flxActCommonLicSpcPopulateFromTS or flxActCommonLicSpcPopulateAllFromTS.
product	The fulfillment record to be deleted from trusted storage.

Return Values

The following values are returned by **flxActCommonLicSpcProductDelete:**

Table 8-80 • flxActCommonLicSpcProductDelete Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function deletes a fulfillment record from trusted storage. Before calling this function:

- Create a license specification object with flxActCommonLicSpcCreate.
- Write fulfillment records to the license specification using flxActCommonLicSpcPopulateFromTS or flxActCommonLicSpcPopulateAllFromTS.
- Use the appropriate **flxActCommon** functions to access the fulfillment records in the license specification and determine the one for deletion:

- flxActCommonLicSpcGetNumberProducts and flxActCommonLicSpcGet
- flxActCommonLicSpcGetNumProdEntries and flxActCommonLicSpcGetProd

This function deletes the specified fulfillment record from trusted storage. You can continue to iterate through the fulfillment records in the license specification and delete other fulfillment records as required. To refresh your copy of the fulfillment records in trusted storage, call flxActCommonLicSpcPopulateFromTS or flxActCommonLicSpcPopulateAllFromTS. Any fulfillment records you have deleted from trusted storage will no longer be included in the license specification object.

It is recommended that this function is only used to delete fulfillment records from client mode trusted storage. If fulfillment records are deleted from a license server, it will leave any fulfillment records supplied from the deleted record orphaned. These orphaned records cannot be either returned or repaired.

For outline instructions for deleting a fulfillment record from trusted storage, see the *Programming Reference for Trusted Storage–Based Licensing*. For example code that uses this function, see `sHandleDelete` in the example application activation utility, `appActUtil.c`.

flxActCommonProdLicSpcActServerChainGet

Get details of the route by which the fulfillment record in the supplied product license specification was served.

Syntax

```
const char *flxActCommonProdLicSpcActServerChainGet(FlxActProdLicSpc prodSpc);
```

Parameters

The **flxActCommonProdLicSpcActServerChainGet** function uses the following parameters:

Table 8-81 • flxActCommonProdLicSpcActServerChainGet Parameters

Parameter	Description
prodSpc	A product license specification that contains a fulfillment record.

Return Values

The chain of servers through which this fulfillment record was supplied or NULL if not set.

This is given in the form of a list of the names of the servers through which this fulfillment record has been supplied. The last server through which it passed is first in the list. Each entry is separated by a vertical bar, |, so an example server chain is:

`adam|nexus|london`

Discussion

Before calling this routine access the fulfillment record. Possible methods of accessing fulfillment records are:

- Reading fulfillment records from trusted storage and selecting the fulfillment record to be read—flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, and flxActCommonLicSpcGet.

- Reading the fulfillment record returned by the activation server in an activation transaction—flxActAppActivationRespProdLicSpcGet.

flxActCommonProdLicSpcCountGet

Get the value of the specified count from the fulfillment record in the supplied product license specification.

Syntax

```
uint32_t flxActCommonProdLicSpcCountGet(
    FlxActProdLicSpc prodSpc,
    FlxActCountType countType);
```

Parameters

The **flxActCommonProdLicSpcCountGet** function uses the following parameters:

Table 8-82 • flxActCommonProdLicSpcCountGet Parameters

Parameter	Description
prodSpc	A product license specification that contains a fulfillment record.
countType	The type of count required. For more information about these license types, see the *Programming Reference for Trusted Storage–Based Licensing*. The following values (FlxActCountType) can be used for this parameter: • `CONCURRENT`—Concurrent licenses in the fulfillment record • `CONCURRENT_OD`—Concurrent overdraft licenses in the fulfillment record • `ACTIVATABLE`—Activatable licenses in the fulfillment record • `ACTIVATABLE_OD`—Activatable overdraft licenses in the fulfillment record • `HYBRID`—Hybrid licenses in the fulfillment record • `HYBRID_OD`—Hybrid overdraft licenses in the fulfillment record • `REPAIR`—The number of times this fulfillment record can be repaired

Return Values

The value of the count in the fulfillment record.

Discussion

Before calling this routine access the fulfillment record that contains the count that is required. Possible methods of accessing fulfillment records are:

- Reading fulfillment records from trusted storage and selecting the fulfillment record to be read—flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, and flxActCommonLicSpcGet.
- Reading the fulfillment record returned by the activation server in an activation transaction—flxActAppActivationRespProdLicSpcGet.

The following counts are used for license groups: CONCURRENT, CONCURRENT_OD, ACTIVATABLE, ACTIVATABLE_OD, HYBRID, and HYBRID_OD. In these cases, this function returns the current number of licenses of the specified type in the fulfillment record. These counts are only effective for fulfillment records on a license server.

The REPAIRS count is only used for fulfillment records on a license server. It defines the number of times that this fulfillment record can be repaired.

See Also
Information about how license groups are used in the *Programming Reference for Trusted Storage–Based Licensing*

flxActCommonProdLicSpcDedSpcGet

Get the deduction record at the specified index from the product license specification.

Syntax

```
FlxActDedSpc flxActCommonProdLicSpcDedSpcGet(
    FlxActProdLicSpc prodSpc,
    uint32_t index);
```

Parameters

The **flxActCommonProdLicSpcDedSpcGet** function uses the following parameters:

Table 8-83 • flxActCommonProdLicSpcDedSpcGet Parameters

Parameter	Description
prodSpc	A product license specification that contains the fulfillment record for which the deduction record is required.
index	The position of the desired deduction record where index is in the range 0–count-1, with 0 being the first deduction record. Use flxActCommonProdLicSpcNumberDedSpcGet to determine count.

Return Values

If successful, the deduction record is returned. If not successful, NULL is returned.

Discussion

This function is used in conjunction with flxActCommonProdLicSpcNumberDedSpcGet.

flxActCommonProdLicSpcEntitlementIdGet

Get the entitlement ID from the fulfillment record in the supplied product license specification.

Syntax

```
const char *flxActCommonProdLicSpcEntitlementIdGet(FlxActProdLicSpc prodSpc);
```

Parameters

The **flxActCommonProdLicSpcEntitlementIdGet** function uses the following parameters:

Table 8-84 • flxActCommonProdLicSpcEntitlementIdGet Parameters

Parameter	Description
prodSpc	A product license specification that contains a fulfillment record.

Return Values

The entitlement ID from the fulfillment record or NULL if not set.

Discussion

Before calling this routine access the fulfillment record that contains the entitlement ID that is required. Possible methods of accessing fulfillment records are:

- Reading fulfillment records from trusted storage and selecting the fulfillment record to be read—flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, and flxActCommonLicSpcGet.
- Reading the fulfillment record returned by the activation server in an activation transaction—flxActAppActivationRespProdLicSpcGet.

This function returns the entitlement ID in the fulfillment record.

flxActCommonProdLicSpcExpDateGet

Get the expiration date from the fulfillment record in the supplied product license specification.

Syntax

```
const char *flxActCommonProdLicSpcExpDateGet(FlxActProdLicSpc prodSpc);
```

Parameters

The **flxActCommonProdLicSpcExpDateGet** function uses the following parameters:

Table 8-85 • flxActCommonProdLicSpcExpDateGet Parameters

Parameter	Description
prodSpc	A product license specification that contains a fulfillment record.

Return Values

The expiration date from the fulfillment record or NULL if not set. The format is dd-mmm-yyyy.

Discussion

Before calling this routine access the fulfillment record that contains the expiration date that is required. Possible methods of accessing fulfillment records are:

- Reading fulfillment records from trusted storage and selecting the fulfillment record to be read—flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, and flxActCommonLicSpcGet.
- Reading the fulfillment record returned by the activation server in an activation transaction—flxActAppActivationRespProdLicSpcGet.

This function returns the expiration date in the fulfillment record.

flxActCommonProdLicSpcFeatureLineGet

Get the feature definition lines from the fulfillment record in the supplied product license specification.

Syntax

```
const char *flxActCommonProdLicSpcFeatureLineGet(FlxActProdLicSpc prodSpc);
```

Parameters

The **flxActCommonProdLicSpcFeatureLineGet** function uses the following parameters:

Table 8-86 • flxActCommonProdLicSpcFeatureLineGet Parameters

Parameter	Description
prodSpc	A product license specification that contains a fulfillment record.

Return Values

The feature definition lines from the fulfillment record, or NULL if not set.

Discussion

Before calling this routine access the fulfillment record that contains the feature definition lines that are required. Possible methods of accessing fulfillment records are:

- Reading fulfillment records from trusted storage and selecting the fulfillment record to be read—flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, and flxActCommonLicSpcGet.
- Reading the fulfillment record returned by the activation server in an activation transaction—flxActAppActivationRespProdLicSpcGet.

This function returns the feature definition lines in the fulfillment record.

flxActCommonProdLicSpcFulfillmentIdGet

Get the fulfillment ID from the fulfillment record in the supplied product license specification.

Syntax

```
const char *flxActCommonProdLicSpcFulfillmentIdGet(FlxActProdLicSpc prodSpc);
```

Parameters

The **flxActCommonProdLicSpcFulfillmentIdGet** function uses the following parameters:

Table 8-87 • flxActCommonProdLicSpcFulfillmentIdGet Parameters

Parameter	Description
prodSpc	A product license specification that contains a fulfillment record.

Return Values

The fulfillment ID from the fulfillment record or NULL if not set.

Discussion

Before calling this routine access the fulfillment record that contains the fulfillment ID that is required. Possible methods of accessing fulfillment records are:

- Reading fulfillment records from trusted storage and selecting the fulfillment record to be read—flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, and flxActCommonLicSpcGet.
- Reading the fulfillment record returned by the activation server in an activation transaction—flxActAppActivationRespProdLicSpcGet.

This function returns the fulfillment ID of the supplied fulfillment record.

flxActCommonProdLicSpcFulfillmentTypeGet

Get the fulfillment type from the fulfillment record in the supplied product license specification.

Syntax

```
FlxFulfillmentType flxActCommonProdLicSpcFulfillmentTypeGet(FlxActProdLicSpc prodSpc);
```

Parameters

The **flxActCommonProdLicSpcFulfillmentTypeGet** function uses the following parameters:

Table 8-88 • flxActCommonProdLicSpcFulfillmentTypeGet Parameters

Parameter	Description
prodSpc	A product license specification that contains a fulfillment record.

Return Values

The type of “transaction” used to place this fulfillment record in trusted storage. For details of valid values, see flxFulfillmentType.

Discussion

Before calling this routine access the fulfillment record. Possible methods of accessing fulfillment records are:

- Reading fulfillment records from trusted storage and selecting the fulfillment record to be read—flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, and flxActCommonLicSpcGet.
- Reading the fulfillment record returned by the activation server in an activation transaction—flxActAppActivationRespProdLicSpcGet.

This function returns the fulfillment type from the supplied fulfillment record. The fulfillment type is included in the fulfillment record in one of the following ways:

- Added to the fulfillment record when an ASR is created by `asrgen`. This value may be subsequently edited before the ASR is used.
- Added to the fulfillment record when an activation response is supplied (served) by a license server.
- Added to the fulfillment record when an activation response is supplied (served) by FlexNet Operations.
- Added to the fulfillment record when an activation response is supplied (served) by a License Generator.
- Included in a fulfillment record within an activation response file used by the Manual Response Generator during a test transaction.

For information on the structure of fulfillment records, see the *Programming Reference for Trusted Storage–Based Licensing*.

The fulfillment type records details of the transaction that will be, or was, used to add the fulfillment record to trusted storage. As can be seen, this information is added to the fulfillment record in various ways. You should be aware that a consistent meaning can only be ensured if the 'correct' fulfillment type is added to all manually edited fulfillment records. Also, when an ASR is used for local activation, that the fulfillment types are used correctly and consistently. For details about fulfillment types, see the following section, flxFulfillmentType.

flxFulfillmentType

This enumerated type provides a mapping from the **fulfillmentType** key-value pair in a fulfillment record. The numeric values in the key-value pair are mapped as followed:

Table 8-89 • flxFulfillmentType

Value	Type	Description
0	`FULFILLMENT_TYPE_UNKNOWN`	The type of transaction that added this fulfillment record to trusted storage has not been recorded.
1	`FULFILLMENT_TYPE_TRIAL`	This fulfillment record was added to trusted storage as a result of a local activation that was used to install a 'trial' license. See Note in this section.
2	`FULFILLMENT_TYPE_SHORTCODE`	This fulfillment record was added to trusted storage as a result of a short code transaction.

Table 8-89 • flxFulfillmentType

Value	Type	Description
3	`FULFILLMENT_TYPE_EMERGENCY`	This fulfillment record was added to trusted storage as a result of a local activation that was used to install an 'emergency' license. See Note in this section.
4	`FULFILLMENT_TYPE_SERVED_ACTIVATION`	This fulfillment record was added to trusted storage as a result of an activation transaction with a license server. The license was served from the normal pool of licenses.
5	`FULFILLMENT_TYPE_SERVED_OVERDRAFT_ACTIVATION`	This fulfillment record was added to trusted storage as a result of an activation transaction with a license server. The license was served from the overdraft pool of licenses.
6	`FULFILLMENT_TYPE_PUBLISHER_ACTIVATION`	This fulfillment record was added to trusted storage as a result of an activation transaction with a publisher's activation server.
7	`FULFILLMENT_TYPE_PUBLISHER_OVERDRAFT_ACTIVATION`	This fulfillment type is not currently used.

Note *• An ASR used for local activation can be used to implement various licensing models. The two main ones are trial and emergency:*

- **Trial** *is a limited duration license that allows a user to trial a product. To continue using the product when this license expires, the user must initiate a licensing transaction with an activation server.*
- **Emergency** *is a limited duration license that allows a user to use a product when a license is not available from an activation server (for example, when the activation server is down).*

These are identified as separate fulfillment types. You must ensure that these fulfillment types are used consistently: the Activation API does not enforce consistent usage of these fulfillment types.

flxActCommonProdLicSpcIsDisabled

Returns the status of the specified fulfillment record.

Syntax

```
FlxActBool flxActCommonProdLicSpcIsDisabled(FlxActProdLicSpc prodSpc);
```

Parameters

The **flxActCommonProdLicSpcIsDisabled** function uses the following parameters:

Table 8-90 • flxActCommonProdLicSpcIsDisabled Parameters

Parameter	Description
prodSpc	A product license specification that contains a fulfillment record.

Return Values

The status of the fulfillment record:

- TRUE = disabled
- FALSE = not disabled

Discussion

Before calling this routine access the fulfillment record by reading fulfillment records from trusted storage and selecting the fulfillment record to be read. (For example, flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, and flxActCommonLicSpcGet.)

This function returns the status of the specified fulfillment record. Fulfillment records can be disabled during a return transaction. When the return request is sent, the fulfillment record that is being returned is disabled. If the return response is success, then the fulfillment record is deleted from trusted storage. If the return response is non-success, then the disablement of the fulfillment record is removed.

The license rights contained in a disabled fulfillment record cannot be used.

flxActCommonProdLicSpcNumberDedSpcGet

Returns the number of deduction records in trusted storage for the fulfillment record in the supplied product license specification.

Syntax

```
uint32_t flxActCommonProdLicSpcNumberDedSpcGet(FlxActProdLicSpc prodSpc);
```

Parameters

The **flxActCommonProdLicSpcNumberDedSpcGet** function uses the following parameters:

Table 8-91 • flxActCommonProdLicSpcNumberDedSpcGet Parameters

Parameter	Description
prodSpc	A product license specification that contains the fulfillment record for which the deduction records are required.

Return Values

The number of deduction records in this product license specification.

Discussion

This function is used together with flxActCommonProdLicSpcDedSpcGet to access the deduction records for the specified fulfillment record. Then the **flxActCommonDedSpc*** functions are used to read the deduction records. The basic procedure is:

1. Identify the fulfillment record—use flxActCommonLicSpcGetNumberProducts or flxActCommonLicSpcGetNumProdEntries to determine the number of fulfillment records to be searched and then examine each fulfillment record in turn (using flxActCommonLicSpcGet or flxActCommonLicSpcGetProd.
2. Obtain the number of deduction records for the selected fulfillment record—**flxActCommonProdLicSpcNumberDedSpcGet**.
3. Examine a deduction record in the product license specification—use flxActCommonProdLicSpcNumberDedSpcGet to load a deduction record from the product license specification.
4. Use **flxActCommonDedSpc*Get** to obtain the value of a parameter in the deduction record.
5. Repeat Step 3 and Step 4 for subsequent deduction records in the product license specification—loop based on the number of deduction records returned using **flxActCommonProdLicSpcNumberDedSpcGet**.

For information about the format and use of deduction records, see the *Programming Reference for Trusted Storage–Based Licensing*.

flxActCommonProdLicSpcNumberValidDedSpcGet

Returns the number of valid activation and transfer deduction records in trusted storage for the fulfillment record in the supplied product license specification.

Syntax

```
uint32_t flxActCommonProdLicSpcNumberValidDedSpcGet(FlxActProdLicSpc prodSpc);
```

Parameters

The **flxActCommonProdLicSpcDedSpcGet** function uses the following parameters:

Table 8-92 • flxActCommonProdLicSpcDedSpcGet Parameters

Parameter	Description
prodSpc	A product license specification that contains the fulfillment record for which the valid activation and transfer deduction records are required.

Return Values

The number of valid activation and transfer deduction records in this product license specification.

Discussion

This function is used to determine the number of valid activation and transfer deduction records in trusted storage associated with the fulfillment record in the supplied product license specification. Deduction records are *valid* if they reference fulfillment records on remote machines that contain unexpired license rights. An *activation* deduction record is created when an activation request is received from an end-user machine and license rights are supplied by the license server. A *transfer* deduction record is created when an activation request is received from a license server and license rights are supplied by this license server. For a full list of types of deduction records, see FlxActDeductionType.

For information about the format and use of deduction records, see the *Programming Reference for Trusted Storage–Based Licensing*.

flxActCommonProdLicSpcPopulateFRFromServerTS

Retrieves *all* information for a specific fulfillment record stored in remote server-side trusted storage. The information includes deduction records, in addition to fulfillment record details.

The product license specification for the specific fulfillment is obtained from the license specification container generated from a call to flxActCommonLicSpcPopulateFromServerTS or flxActCommonLicSpcPopulateAllFromServerTS.

Note • *The use of this function is limited to the retrieval of fulfillment information in server-side trusted-storage only; it cannot be used to retrieve fulfillment information stored in the client trusted storage.*

Syntax

```
FlxActBool flxActCommonProdLicSpcPopulateFRFromServerTS(
    FlxActProdLicSpc prodLicSpc,
    const char *pszServer);
```

Parameters

The **flxActCommonProdLicSpcPopulateFRFromServerTS** function uses the following parameters:

Table 8-93 • flxActCommonProdLicSpcPopulateFRFromServerTS Parameters

Parameter	Description
prodLicSpc	The product license specification used as input and output for this function: ● **Input**—The product license specification object initially populated by a call to flxActCommonLicSpcPopulateFromServerTS or flxActCommonLicSpcPopulateAllFromServerTS supplies the fulfillmentID. ● **Output**—The product license specification is populated with the requested fulfillment record and its associated deduction records.
pszServer	A pointer to the variable storing the address of the license server from which the counts are retrieved. The following formats can be used for this parameter: ● *port@host* ● *@host*

Return Values

The following values are returned by **flxActCommonProdLicSpcPopulateFRFromServerTS**:

Table 8-94 • flxActCommonProdLicSpcPopulateFRFromServerTS Parameters Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to determine the specific error. Possible errors include: ● `LM_TS_BADPARAM`—One or more of the parameters passed in are invalid. ● `LM_TS_CANTMALLOC`—Insufficient memory has caused an error. ● `LM_TS_INVALID_REQUEST_TYPE`—The activation server does not support this request type. (This error normally occurs if the request is corrupt.) ● `LM_TS_SERVERQUERY_GEN_REQ`—The request to query remote server-side trusted storage cannot be generated. ● `LM_TS_UNKNOWN_RESP`—The response is corrupt. ● `LM_TS_SERVERQUERY_RESPONSE_FAILED`—The response to query remote server-side trusted storage cannot be processed. ● `LM_TS_SERVERQUERY_FR_NOT_FOUND`—The fulfillment record in remote server-side trusted storage does not exist.

Discussion

Either flxActCommonLicSpcPopulateFromServerTS or flxActCommonLicSpcPopulateAllFromServerTS must be called before this function to populate a license specification container with product license specification objects. Use the existing product license specification APIs to retrieve the fulfillment information from product license specification object.

Important • *The remote license server whose trusted storage you want to read must be running.*

Performance

The frequent use of this API can cause activation-client connections to experience timeouts due to the license server's being busy in processing this request. Therefore, use this API sparingly. In addition to network factors, the API response time depends on factors such as the number of feature lines in a fulfillment record, the length of each feature line, the number of deduction records for the fulfillment record, and the number of clients making simultaneous requests to the license server.

flxActCommonProdLicSpcProductIdGet

Get the product ID from the fulfillment record in the supplied product license specification.

Syntax

```
const char *flxActCommonProdLicSpcProductIdGet(FlxActProdLicSpc prodSpc);
```

Parameters

The **flxActCommonProdLicSpcProductIdGet** function uses the following parameters:

Table 8-95 • flxActCommonProdLicSpcProductIdGet Parameters

Parameter	Description
prodSpc	A product license specification that contains a fulfillment record.

Return Values

The product ID from the fulfillment record or NULL if not set.

Discussion

Before calling this routine, access the fulfillment record that contains the product ID that is required.

Possible methods of accessing fulfillment records are:

- Reading fulfillment records from trusted storage and selecting the fulfillment record to be read—flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, and flxActCommonLicSpcGet.
- Reading the fulfillment record returned by the activation server in an activation transaction—flxActAppActivationRespProdLicSpcGet.

This function returns the product ID in the fulfillment record.

flxActCommonProdLicSpcSuiteIdGet

Get the suite ID from the fulfillment record in the supplied product license specification.

Syntax

```
const char *flxActCommonProdLicSpcSuiteIdGet(FlxActProdLicSpc prodSpc);
```

Parameters

The **flxActCommonProdLicSpcSuiteIdGet** function uses the following parameters:

Table 8-96 • flxActCommonProdLicSpcSuiteIdGet Parameters

Parameter	Description
prodSpc	A product license specification that contains a fulfillment record.

Return Values

The suite ID from the fulfillment record or NULL if not set.

Discussion

Before calling this routine access the fulfillment record that contains the suite ID that is required. Possible methods of accessing fulfillment records are:

- Reading fulfillment records from trusted storage and selecting the fulfillment record to be read—flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, and flxActCommonLicSpcGet.
- Reading the fulfillment record returned by the activation server in an activation transaction—flxActAppActivationRespProdLicSpcGet.

This function returns the suite ID in the fulfillment record.

flxActCommonProdLicSpcTrustFlagsGet

Get the trust status of the fulfillment record in the supplied product license specification.

Syntax

```
uint32_t flxActCommonProdLicSpcTrustFlagsGet(FlxActProdLicSpc prodSpc);
```

Parameters

The **flxActCommonProdLicSpcTrustFlagsGet** function uses the following parameters:

Table 8-97 • flxActCommonProdLicSpcTrustFlagsGet Parameters

Parameter	Description
prodSpc	A product license specification that contains a fulfillment record.

Return Values

The trust status from the fulfillment record or NULL if not set. See Discussion for an explanation of this value.

Discussion

Before calling this routine access the fulfillment record whose trust status is required. Possible methods of accessing fulfillment records are:

- Reading fulfillment records from trusted storage and selecting the fulfillment record to be read—flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, and flxActCommonLicSpcGet.
- Reading the fulfillment record returned by the activation server in an activation transaction—flxActAppActivationRespProdLicSpcGet.

The security of fulfillment records within trusted storage is ensured in three ways:

- **Anchor(s)**—record the deletion and restoration of a publisher trusted storage file.
- **Binding**—locks a trusted storage segment to the machine on which it is located.
- **Time stamp**—records when the fulfillment record was last accessed.

Fulfillment records can be *trusted* or *untrusted* based on whether the security is intact. This is reported by trusted storage as three trust flags, which can be set either to `trusted` or `untrusted`:

- **Restore trust flag**—reports the state of anchors required for the fulfillment record
- **Host trust flag**—reports the binding state for the fulfillment record
- **Time trust flag**—reports whether the last time the fulfillment record was accessed was earlier than the current time (trusted) or the last time the fulfillment record was accessed was later than the current time (untrusted).

This function returns the trust flags as an integer where the flags are represented by bits as follows:

- Restore trust flag—bit 0
- Host trust flag—bit 1
- Time trust flag—bit 2

  where trusted = 1 (bit set) and untrusted = 0 (bit not set).

The following values are defined for trust flags:

- `FLAGS_FULLY_TRUSTED` = all the trust flags are set to trusted.
- `FLAGS_TRUST_RESTORE` = restore flag set to trusted
- `FLAGS_TRUST_HOST` = host flag set to trusted
- `FLAGS_TRUST_TIME` = time trust flag set to trusted

flxActCommonProdLicSpcUniqueIdGet

Gets the unique ID from the fulfillment record in the supplied product license specification.

Syntax

```
const char *flxActCommonProdLicSpcUniqueIdGet(FlxActProdLicSpc prodSpc);
```

Parameters

The **flxActCommonProdLicSpcUniqueIdGet** function uses the following parameters:

Table 8-98 • flxActCommonProdLicSpcUniqueIdGet Parameters

Parameter	Description
prodSpc	A product license specification that contains a fulfillment record.

Return Values

The unique ID from the fulfillment record or NULL if not set.

Discussion

Before calling this routine access, the fulfillment record that contains the unique ID that is required. Possible methods of accessing fulfillment records are:

- Reading fulfillment records from trusted storage and selecting the fulfillment record to be read—flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, and flxActCommonLicSpcGet.
- Reading the fulfillment record returned by the activation server in an activation transaction—flxActAppActivationRespProdLicSpcGet.

This function returns the unique ID in the fulfillment record.

flxActCommonProdLicSpcVendorDataGetByIndex

Gets the unique ID from the fulfillment record in the supplied product license specification.

Syntax

```
FlxActBool flxActCommonProdLicSpcVendorDataGetByIndex(
    FlxActProdLicSpc prodSpc,
    uint32_t index,
    const char **ppKey,
    const char **ppValue);
```

Parameters

The **flxActCommonProdLicSpcVendorDataGetByIndex** function uses the following parameters:

Table 8-99 • flxActCommonProdLicSpcVendorDataGetByIndex Parameters

Parameter	Description
prodSpc	A product license specification that contains a fulfillment record.

Table 8-99 • flxActCommonProdLicSpcVendorDataGetByIndex Parameters

Parameter	Description
index	Specifies the key-value entry required.
ppKey	A pointer for the return of a pointer to the key for the specified index as held in the fulfillment record provided.
ppValue	A pointer for the return of a pointer to the value for the specified index as held in the fulfillment record provided.

Return Values

The following values are returned by **flxActCommonProdLicSpcVendorDataGetByIndex:**

Table 8-100 • flxActCommonProdLicSpcVendorDataGetByIndex Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this routine access the fulfillment record that contains the vendor data dictionary that you require. Possible methods of accessing fulfillment records are:

- Reading the fulfillment record returned by the activation server in an activation transaction—flxActAppActivationRespProdLicSpcGet.
- Reading fulfillment records from trusted storage and selecting the fulfillment record to be read—flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, and flxActCommonLicSpcGet**.**

This function gets the key-value pair from the vendor data dictionary at the specified index. See flxActCommonProdLicSpcVendorDataGetCount for a detailed description of how to use this function.

flxActCommonProdLicSpcVendorDataGetByKey

Get data value for the key specified from the vendor data dictionary in the fulfillment record.

Syntax

```
FlxActBool flxActCommonProdLicSpcVendorDataGetByKey(
    FlxActProdLicSpc prodSpc,
    const char *pKey,
    const char **ppValue);
```

Parameters

The **flxActCommonProdLicSpcVendorDataGetByKey** function uses the following parameters:

Table 8-101 • flxActCommonProdLicSpcVendorDataGetByKey Parameters

Parameter	Description
prodSpc	A product license specification that contains a fulfillment record.
pKey	The specified key for which the data value is required.
ppValue	A pointer for the return of a pointer to the value for the specified vendor data key as held in the fulfillment record provided.

Return Values

The following values are returned by **flxActCommonProdLicSpcVendorDataGetByKey:**

Table 8-102 • flxActCommonProdLicSpcVendorDataGetByKey Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this routine access the fulfillment record that contains the vendor data that you require. Possible methods of accessing fulfillment records are:

- Reading the fulfillment record returned by the activation server in an activation transaction—flxActAppActivationRespProdLicSpcGet.
- Reading fulfillment records from trusted storage and selecting the fulfillment record to be read—flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, and flxActCommonLicSpcGet**.**

This function gets data from the vendor data dictionary that is part of the specified fulfillment record. For information about using the Vendor Dictionary (vendor data) for custom data, see the *Programming Reference for Trusted Storage–Based Licensing*.

The data value for the specified key is returned by this function.

flxActCommonProdLicSpcVendorDataGetCount

Returns the number of vendor data dictionary entries in the fulfillment record.

Syntax

```
FlxActBool flxActCommonProdLicSpcVendorDataGetCount(
    FlxActProdLicSpc prodSpc,
```

```
uint32_t *pCount);
```

Parameters

The **flxActCommonProdLicSpcVendorDataGetCount** function uses the following parameters:

Table 8-103 • flxActCommonProdLicSpcVendorDataGetCount Parameters

Parameter	Description
prodSpc	A product license specification that contains a fulfillment record.
pCount	A pointer for the return of the number of key-value pairs in the fulfillment record.

Return Values

The following values are returned by **flxActCommonProdLicSpcVendorDataGetCount:**

Table 8-104 • flxActCommonProdLicSpcVendorDataGetCount Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this routine access the fulfillment record that contains the vendor data dictionary that you require. Possible methods of accessing fulfillment records are:

- Reading the fulfillment record returned by the activation server in an activation transaction—flxActAppActivationRespProdLicSpcGet.
- Reading fulfillment records from trusted storage and selecting the fulfillment record to be read—flxActCommonLicSpcPopulateFromTS, flxActCommonLicSpcGetNumberProducts, and flxActCommonLicSpcGet**.**

This function is used together with flxActCommonProdLicSpcVendorDataGetByIndex to access the dictionary entries in the vendor dictionary. Use these two functions when you do not know the structure of the vendor dictionary. The basic procedure is as follows:

1. Obtain the number of dictionary entries—flxActCommonProdLicSpcVendorDataGetCount.
2. Use flxActCommonProdLicSpcVendorDataGetByIndex to get the first key-value pair from the data dictionary (first entry index=0).
3. Repeat Step 2—loop based on the number of data dictionary entries returned using flxActCommonProdLicSpcVendorDataGetCount (loop until pCount-1).

flxActCommonRepairLocalTrustedStorage

Partially, repairs broken segments of trusted storage so that new fulfillment records can be added. Existing fulfillment records in broken segments of trusted storage will remain untrusted until they are repaired using a repair transaction. This function may be used with any of local activation, server activation, and short code activation.

Syntax

```
FlxActBool flxActCommonRepairLocalTrustedStorage(FlxActHandle handle);
```

Parameters

The **flxActCommonRepairLocalTrustedStorage** function uses the following parameters:

Table 8-105 • flxActCommonRepairLocalTrustedStorage Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.

Return Values

The following values are returned by **flxActCommonRepairLocalTrustedStorage:**

Table 8-106 • flxActCommonRepairLocalTrustedStorage Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function repairs broken segments of trusted storage to the point where new fulfillment records can be added. For a description of local activation, see the *Programming Reference for Trusted Storage–Based Licensing*.

Before calling this function create an activation handle using flxActCommonHandleOpen with `mode` set to `FLX_ACT_APP`. This function will fail if called after a call to **flxActCommonLicSpcCreate**.

This function does not repair broken fulfillment records. See the *Programming Reference for Trusted Storage–Based Licensing* for descriptions of the functions required to repair broken fulfillment records (for example, a repair from a license server).

For an example of the use of this function, see **sHandleLocalRepair** in the example application activation utility, `appActUtil.c`.

Note • *As a best practice follow the points mentioned below:*

1. *This function must be called before performing a local activation (***flxActCommonLicSpcAddASRs, flxActCommonResetTrialASRs, flxActCommonLicSpcAddASRFromBuffer, flxActCommonResetTrialASRFromBuffer***)*
2. *The single action and shortcode request generation functions call this function internally so there is no need to call it before using them.*
3. *Composite transaction functions do not call it internally; the sample utility* `compTranUtil` *adopts the policy of only calling it on error* `FLX_ACT_TRAN_ERR_TS_LOAD` *and this is best practice (see sample file* `compTranUtil.c`*)*

flxActCommonResetTrialASRFromBuffer

Uses the ASR in the buffer to reset the trial anchor and remove the Fulfillment Record from trusted storage.

Syntax

```
FlxActBool flxActCommonResetTrialASRFromBuffer(
    FlxActLicSpc licSpc,
    const char *pszBuffer);
```

Parameters

The **flxActCommonResetTrialASRFromBuffer** function uses the following parameters:

Table 8-107 • flxActCommonResetTrialASRFromBuffer Parameters

Parameter	Description
licSpc	A license specification creating using flxActCommonLicSpcCreate.
pszBuffer	A buffer that contains the ASR that specifies: ● the trial anchor to be reset ● the Fulfillment Record that is to be deleted from trusted storage

Return Values

The following values are returned by **flxActCommonResetTrialASRFromBuffer:**

Table 8-108 • flxActCommonResetTrialASRFromBuffer Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function:

1. Open a client mode handle—flxActCommonHandleOpen
2. Repair trusted storage anchoring—flxActCommonRepairLocalTrustedStorage (required when trusted storage has been deleted)
3. Create a license specification—flxActCommonLicSpcCreate
4. Add the fulfillment records in trusted storage— flxActCommonLicSpcPopulateFromTS

This functions reads the ASRs in the buffer and the following steps are carried out:

1. The ASR signature is validated.
2. The DataDictionary within the ASR is read. The following values must be present in the ASR for the trial anchor to be reset:
   - PUBLISHER_DEFINED_CRITERION with value LOCAL_ONCE
   - PUBLISHER_TRIAL_ID
   - PUBLISHER_TRIAL_RESET with a numeric value that is not zero

   For a description of an ASR, see the *Programming Reference for Trusted Storage–Based Licensing*.
3. The trial anchor specified in the ASR is reset so that it can be reused.
4. The Fulfillment Record specified in the ASR is deleted from trusted storage.

For example code for this function, see **sHandleTrialReset** in the example activation-utility code files, `appActUtil.c`. and `serverActUtil.c`, included in the toolkit.

See Also
flxActCommonLicSpcAddASRs
flxActCommonLicSpcAddASRFromBuffer
Programming Reference for Trusted Storage–Based Licensing

flxActCommonResetTrialASRs

Resets the trial anchors and removes Fulfillment Records from trusted storage as specified in the ASRs.

Syntax

```
FlxActBool flxActCommonResetTrialASRs(
    FlxActLicSpc licSpc,
    const char *filePath);
```

Parameters

The **flxActCommonResetTrialASRs** function uses the following parameters:

Table 8-109 • flxActCommonResetTrialASRs Parameters

Parameter	Description
licSpc	A license specification creating using flxActCommonLicSpcCreate.
filePath	The location of the ASRs that specify the trial anchors to be reset and the FRs to be deleted. This can be either of the following: ● The path to a directory that contains the ASRs ● The path and file name of an ASR

Return Values

The following values are returned by **flxActCommonResetTrialASRs:**

Table 8-110 • flxActCommonResetTrialASRs Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function:

1. Open a handle in the appropriate mode—flxActCommonHandleOpen
2. Repair trusted storage anchoring—flxActCommonRepairLocalTrustedStorage (required when trusted storage has been deleted)
3. Create a license specification—flxActCommonLicSpcCreate
4. Add the fulfillment records in trusted storage— flxActCommonLicSpcPopulateFromTS

This function reads each of the ASRs in the specified directory, or a single ASR file if a filename is included in `filePath`, and the following steps are carried out:

1. The ASR signature is validated.
2. The DataDictionary within the ASR is read. The following values must be present in the ASR for the trial anchor to be reset:
   - PUBLISHER_DEFINED_CRITERION with value LOCAL_ONCE
   - PUBLISHER_TRIAL_ID
   - PUBLISHER_TRIAL_RESET with a numeric value that is not zero

   For a description of an ASR, see the *Programming Reference for Trusted Storage–Based Licensing*.

3. The trial anchor specified in the ASR is reset so that it can be reused.
4. The Fulfillment Record specified in the ASR is deleted from trusted storage.

For example code for this function, see **sHandleTrialReset** in the example activation-utility code files, `appActUtil.c`. and `serverActUtil.c`, included in the toolkit.

See Also
flxActCommonLicSpcAddASRs
flxActCommonLicSpcAddASRFromBuffer
Programming Reference for Trusted Storage–Based Licensing

flxActCommonRespProdLicSpcGet

Obtains information about the fulfilment record affected by processing a v1 (single-action) transaction response.

For activations it is the fulfillment record created, for returns it is the fulfillment record deleted, and for repairs it is the fulfillment record repaired. The fulfillment record details are returned in a `FlxActProdLicSpc` object.

Note that for v2 (composite) transactions, the response details are obtained using the **flxActTranResponseXXX** functions.

Syntax

```
FlxActBool flxActCommonRespProdLicSpcGet(
    FlxActHandle handle,
    FlxActProdLicSpc *pProdSpc);
```

Parameters

The **flxActCommonRespProdLicSpcGet** function uses the following parameters:

Table 8-111 • flxActCommonRespProdLicSpcGet Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
prodSpc	A pointer to the `FlxActProdLicSpc` object to be populated with the details of the fulfillment record created, deleted or repaired.

Return Values

The following values are returned by **flxActCommonRespProdLicSpcGet**:

Table 8-112 • flxActCommonRespProdLicSpcGet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success

FlxActDeductionType

This enumerated type is used to identify the type of transaction that created a deduction record.

Table 8-113 • FlxActDeductionType

Value	Type	Description
0	DEDUCTION_TYPE_UNKNOWN	The type of transaction that added this deduction record to trusted storage has not been recorded.
1	DEDUCTION_TYPE_ACTIVATION	This deduction record was added to trusted storage as a result of an activation transaction requested by an end-user machine.
2	DEDUCTION_TYPE_TRANSFER	This deduction record was added to trusted storage as a result of an activation transaction requested by a license server.
3	DEDUCTION_TYPE_REPAIR	This deduction record was added to trusted storage as a result of a repair transaction.

flxActDiagTrustedStorageHealthCheck

Performs a series of checks related to trusted-storage health and records the results as event log entries.

Syntax

```
FlxActBool flxActDiagTrustedStorageHealthCheck(
    uint32_t *pResultArray,
    uint32_t resultArrayLength);
```

Parameters

The **flxActDiagTrustedStorageHealthCheck** function uses the following parameters:

Table 8-114 • flxActDiagTrustedStorageHealthCheck Parameters

Parameter	Description
pResultArray	For the return of the result of each check (see Table 8-116). For example, `pResultArray[0]` is the result of check number 0. Returns `LM_TSSE_NO_ERROR` if the check passed. Otherwise, a system error code defined in `FlxActSystemError.h` is returned. For information on possible errors, see FlxActSystemError. The length of the array should be set to `flxTsHcCount`, which is defined in `FlxActDiagTypes.h`. If it is less, the results of higher-numbered checks will not be returned. If NULL is passed, no results are returned.
resultArrayLength	If `pResultArray` is NULL then zero, otherwise the length of `pResultArray` (number of elements).

Return Values

The following values are returned by **flxActDiagTrustedStorageHealthCheck:**

Table 8-115 • flxActDiagTrustedStorageHealthCheck Return Values

Return Value	Description
FLX_ACT_TRUE	All trusted-storage diagnostic checks passed, or checks are not implemented for this platform. See the event log for more information.
FLX_ACT_FALSE	One or more trusted-storage diagnostic checks failed. See the event log for more information.

Discussion

The function **flxActDiagTrustedStorageHealthCheck** performs the following checks:

Table 8-116 • Health Checks

Number	Name	Check
0	flxTsHcEventLog	Event log check
1	flxTsHcLicensingService	FlexNet Licensing Service version and installation check.
2	flxTsHcAnchoring	Anchor check
3	flxTsHcBinding	Binding check
4	flxTsHcFile	Trusted-storage file check
5	flxTsHcSection	Trusted-storage section check

For an implementation example, see the function **flxCtuDoCommandHealthCheck** in the sample files `compTranUtil.c` and `compTranUtilMisc.c` (in folder `<install_dir>/examples/activation/comptranutil`).

Example

The toolkit contains the sample file `comptranUtilMisc.c` (in folder `<install_dir>/examples/activation/comptranutil`) which demonstrates the implementation of function **flxActDiagTrustedStorageHealthCheck**.

The health check can be demonstrated using the executables `appcomptranutil` and `servercomptranutil`. Use the following commands:

- For a client-side trusted-storage health check: **`appcomptranutil -healthCheck`**
- For a server-side trusted-storage health check: **`servercomptranutil -healthCheck`**

flxActCommonValidateDateString

Validates the supplied date string.

Syntax

```
FlxActBool flxActCommonValidateDateString(
    const char *pszDate,
    FlxActBool *pbIsValid);
```

Parameters

The **flxActCommonValidateDateString** function uses the following parameters:

Table 8-117 • flxActCommonValidateDateString Parameters

Parameter	Description
pszDate	Pointer to a null terminated string containing the date.
pbIsValid	Pointer for returning whether the date is valid.

Return Values

The following values are returned by **flxActCommonValidateDateString**:

Table 8-118 • flxActCommonValidateDateString Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success

Discussion

Valid date strings are "d-mmm-yyyy", "dd-mmm-yyyy", and "permanent" (for example, "1-jan-2018", "09-dec-2019"). If the date is valid, **FLX_ACT_TRUE** is returned via pointer `pbIsvalid`, otherwise **FLX_ACT_FALSE**.

flxActCommonVirtualFamilyGet

Returns the current setting of the Virtual Machine Family attribute.

Syntax

```
FlxActBool flxActCommonVirtualFamilyGet(
    FlxActHandle handle,
    const char** pszFamily);
```

Parameters

The **flxActCommonVirtualFamilyGet** function uses the following parameters:

Table 8-119 • flxActCommonVirtualFamilyGet Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
pszFamily	Pointer to a char* variable into which a pointer to the family string will be written.

Return Values

The following values are returned by **flxActCommonVirtualFamilyGet**:

Table 8-120 • flxActCommonVirtualFamilyGet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success

Discussion

If the Family attribute is not supported on the current platform, the function will return FLX_ACT_FALSE and set major error `LM_TS_NOT_THIS_PLATFORM`.

The caller is responsible for freeing the returned pointer.

Virtualization is only supported for Windows and Linux platforms. **flzActCommonVirtualFamilyGet** will always return FLX_ACT_FALSE for other platforms.

flxActCommonVirtualGenidGet

Returns the current setting of the Virtual Machine Genid attribute.

Syntax

```
FlxActBool flxActCommonVirtualGenidGet(
    FlxActHandle handle,
    const char** pszGenid);
```

Parameters

The **flxActCommonVirtualGenidGet** function uses the following parameters:

Table 8-121 • flxActCommonVirtualGenidGet Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
pszGenid	Pointer to a char* variable into which a pointer to the Genid string will be written.

Return Values

The following values are returned by **flxActCommonVirtualGenidGet**:

Table 8-122 • flxActCommonVirtualGenidGet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success

Discussion

If the Genid attribute is not supported on the current platform, the function will return FLX_ACT_FALSE and set major error `LM_TS_NOT_THIS_PLATFORM`.

The caller is responsible for freeing the returned pointer.

Virtualization is only supported for Windows and Linux platforms. **flxActCommonVirtualGenidGet** will always return FLX_ACT_FALSE for other platforms.

flxActCommonVirtualNameGet

Returns the current setting of the Virtual Machine Name attribute.

Syntax

```
FlxActBool flxActCommonVirtualNameGet(
    FlxActHandle handle,
    const char** pszName);
```

Parameters

The **flxActCommonVirtualNameGet** function uses the following parameters:

Table 8-123 • flxActCommonVirtualNameGet Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
pszName	Pointer to a char* variable into which a pointer to the name string will be written.

Return Values

The following values are returned by **flxActCommonVirtualNameGet**:

Table 8-124 • flxActCommonVirtualNameGet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success

Discussion

If the Name attribute is not supported on the current platform, the function will return FLX_ACT_FALSE and set major error `LM_TS_NOT_THIS_PLATFORM`.

The caller is responsible for freeing the returned pointer.

Virtualization is only supported for Windows and Linux platforms. **flxActCommonVirtualNameGet** will always return FLX_ACT_FALSE for other platforms.

flxActCommonVirtualStatusGet

Gets the platform status of the requesting machine—virtual or physical.

Syntax

```
FlxActBool flxActCommonVirtualStatusGet(
    FlxActHandle handle,
    FlxActBool *pbIsVirtual);
```

Parameters

The **flxActCommonVirtualStatusGet** function uses the following parameters:

Table 8-125 • flxActCommonVirtualStatusGet Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
***pbIsVirtual**	Boolean output that indicates whether the machine initiating the request is virtual: ● TRUE = The machine runs on a virtual platform. ● FALSE = The machine runs on a physical platform.

Return Values

The following values are returned by **flxActCommonVirtualStatusGet**:

Table 8-126 • flxActCommonVirtualStatusGet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function detects whether a hypervisor is present in the requesting machine's environment. If a hypervisor is present, the function returns the `TRUE` value to specify that the machine is running in a virtual environment. If the value is `TRUE`, you can expect that the subsequent composite request from the activation application will include a `<Virtualization>` section containing details of the virtual machine and the hypervisor.

The information returned by this function enables the activation application to make business decisions (for example, to inform the user that a different pricing model applies to virtual machines) before a request for a license is generated.

Virtualization is only supported for Windows and Linux platforms. **flxActCommonVirtualStatusGet** will always return FLX_ACT_FALSE and set major error `LM_TS_NOT_THIS_PLATFORM` for other platforms.

See Also
"Additional Composite Request Data Items for Virtual Environments" in the *Programming Reference for Trusted Storage-Based Licensing*

flxActCommonVirtualUuidGet

Returns the current setting of the Virtual Machine UUID attribute.

Syntax

```
FlxActBool flxActCommonVirtualUuidGet(
    FlxActHandle handle,
```

```
    const char** pszUuid);
```

Parameters

The **flxActCommonVirtualUuidGet** function uses the following parameters:

Table 8-127 • flxActCommonVirtualUuidGet Parameters

Parameter	Description
handle	An activation handle opened with flxActCommonHandleOpen.
pszUuid	Pointer to a char* variable into which a pointer to the uuid string will be written.

Return Values

The following values are returned by **flxActCommonVirtualUuidGet**:

Table 8-128 • flxActCommonVirtualUuidGet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success

Discussion

If the Uuid attribute is not supported on the current platform, the function will return FLX_ACT_FALSE and set major error `LM_TS_NOT_THIS_PLATFORM`.

The caller is responsible for freeing the returned pointer.

Virtualization is only supported for Windows and Linux platforms. **flxActCommonVirtualUuidGet** will always return FLX_ACT_FALSE for other platforms.

Error and Warning Reporting

If a **flxActCommonXXX** function fails it returns **FLX_ACT_FALSE** and sets error information (defined in `FlxActSystemError.h`).

FlxActError

The structure of an Activation API error is as follows:

```
typedef struct(
    uint32_t majorErrorNo;
    uint32_t minorErrorNo;
    uint32_t sysErrorNo;
) FlxActError;
```

The minor and system error codes are for use by Flexera to diagnose faults. The major error codes returned by the Activation API are defined in `FlxActError.h` and detailed in the following table.

Table 8-129 • Activation API Error Codes

Number	Define	Explanation
50001	`LM_TS_BADPARAM`	Error caused by an invalid parameter in the function. Recovery: verify parameters
50002	`LM_TS_CANTMALLOC`	Insufficient memory. Recovery: none
50003	`LM_TS_OPEN_ERROR`	This is an internal error. Recovery: none
50004	`LM_TS_CLOSE_ERROR`	This is an internal error. Recovery: none
50005	`LM_TS_INIT`	Initialization of API failed. Recovery: use **flxActCommonLibraryInit**. Note that this error is not set by flxActCommonLibraryInit.
50006	`LM_TS_ACT_GEN_REQ`	Generate activation request failed. Recovery: check parameters used for activation request
50007	`LM_TS_RETURN_GEN_REQ`	Generate return request failed. Recovery: check parameters used for return request
50008	`LM_TS_REPAIR_GEN_REQ`	Generate repair request failed. Recovery: check parameters used for repair request
50010	`LM_TS_DELETE`	Error when deleting a fulfillment record. Recovery: none
50011	`LM_TS_CANT_FIND`	No matching fulfillment record. Recovery: check fulfillment ID of fulfillment to be returned or repaired
50012	`LM_TS_INTERNAL_ERROR`	Corrupt or incomplete trusted storage or ASR. Recovery: none

Table 8-129 • Activation API Error Codes

Number	Define	Explanation
50014	LM_TS_INVALID_INDEX	Invalid index parameter in API function that requires an index value. Recovery: ensure index value is within the range returned by the relevant API function
50016	LM_TS_NO_ASR_FOUND	Specified ASR not found. Recovery: ensure specified ASR and or path exist
50017	LM_TS_UPDATING_TS	Failed to add ASR to trusted storage. Recovery: none
50018	LM_TS_SEND_RECEIVE	Failure to send request or receive a response for an unspecified reason. Recovery: check parameters used for the request
50019	LM_TS_PROCESS_RESP	Failure to process response for an unspecified reason. This error may be returned when a valid failure response is received, for example when the activation server denies a request. Recovery: check parameters used for the request
50020	LM_TS_UNKNOWN_RESP	Response corrupted. Recovery: none
50021	LM_TS_ASR_LOAD_ONCE	This ASR has been activated. It can only be activated once so this request has been refused. Recovery: none as this is the configured behavior
50023	LM_TS_SHORT_CODE_PENDING	[Legacy] Cannot perform requested action because there is an outstanding short code transaction. Only one short code transaction for a given alias can be in progress at any one time. Recovery: either wait for transaction to complete or use **flxActApp*ShortCodeCancel** to cancel existing transaction
50024	LM_TS_SHORT_CODE_ACT_CREATE	[Legacy] Create a short code activation failure. Recovery: check parameters used for short code activation request
50025	LM_TS_SHORT_CODE_REPAIR_CREATE	[Legacy] Create a short code repair failure. Recovery: check parameters used for short code repair request

Table 8-129 • Activation API Error Codes

Number	Define	Explanation
50026	LM_TS_SHORT_CODE_RETURN_CREATE	[Legacy] Create a short code return failure. Recovery: check parameters used for short code return request
50027	LM_TS_SHORT_CODE_CANCEL	[Legacy] Cancel a short code transaction failure. Recovery: check parameters used for short code transaction cancel request
50028	LM_TS_SHORT_CODE_PROCESS	[Legacy] Error while processing a short code response. Recovery: none
50029	LM_TS_SHORT_CODE_UNSUPPORTED	[Legacy] Either: ASR specified in short code function does not support short codes or Short code ASR specified for use in local activation. Recovery: specify correct ASR
50030	LM_TS_LOAD	Failed to load trusted storage or specified ASR. Recovery: none
50031	LM_TS_FR_DISABLE	Failed to disable fulfillment record in trusted storage. Recovery: none
50032	LM_TS_TIMEDOUT	The connection to the license server or Operations server timed out. Recovery: check connection
50033	LM_TS_INSUFFICIENT_RESOURCE	The license server or Operations server did not have enough licenses to satisfy the activation request. Note that the license server will only provide licenses that can be served from a single fulfillment record. Recovery: submit request with a smaller number of licenses
50034	LM_TS_INVALID_REQUEST_TYPE	Server does not support this request type. Normally this occurs when the request is corrupted. Recovery: resubmit request

Table 8-129 • Activation API Error Codes

Number	Define	Explanation
50035	LM_TS_NO_MATCHING_FULFILLMENT	The license server has no fulfillment records that match this request. Recovery: submit request for a different entitlement
50036	LM_TS_INVALID_REQUEST	The request is invalid. Recovery: check parameters in request
50037	LM_TS_RETURN_OUT_OF_CHAIN	A return request has been sent to a license server that is not the one that served this fulfillment record. Recovery: examine fulfillment record to determine the correct license server and resubmit
50038	LM_TS_MAX_COUNT_EXCEEDED	Return request rejected because maximum count will be exceeded. The maximum count records the original number of licenses in a fulfillment record on a license server. Recovery: none
50039	LM_TS_INSUFFICIENT_REPAIR_COUNT	Repair request rejected because the specified number of repairs have already been completed for this fulfillment record. Recovery: none
50040	LM_TS_OPERATIONS	Error returned by the Operations server. See flxActCommonHandleGetLastOpsError and flxActCommonHandleGetLastOpsErrorString for details of how to retrieve details of the error. Recovery: see FlexNet Operations server documentation
50041	LM_TS_CONNECTION_FAILED	Failed to connect to the license server or Operations server. Recovery: check connection and that server is operational
50042	LM_TS_SSL_ERROR	Error with SSL certificate provided. See flxActCommonHandleSetSSLDetails for more details. Recovery: verify SSL settings
50043	LM_TS_RETURN_INCOMPLETE	Return not allowed because it would orphan fulfillment records created from the fulfillment to be returned. See flxActSvrReturnSend. Recovery: either ensure all fulfillment records created from this fulfillment are returned or use flxActSvrReturnForceIncompleteSet

Table 8-129 • Activation API Error Codes

Number	Define	Explanation
50044	`LM_TS_LOCAL_REPAIR`	Error occurred during call to flxActCommonRepairLocalTrustedStorage. Recovery: none
50045	`LM_TS_UNSUPPORTED_REQUEST_VERSION`	A FlexEnabled application has requested activation from a license server that uses an earlier version of fulfillment records. See information about flexFulfillmentVersion in the *Programming Reference for Trusted Storage–Based Licensing* for more details. Recovery: connect to a license server that is using the correct version of fulfillment records
50046	`LM_TS_UNSUPPORTED`	A FlexEnabled application has received a response from a license server that contains a fulfillment record that it cannot process. See information about flexFulfillmentVersion in the *Programming Reference for Trusted Storage–Based Licensing* for more details. Recovery: connect to a license server that is using the correct version of fulfillment records
50047	`LM_TS_CONFIGURATION`	The requested operation cannot be completed because of an invalid configuration. For example, a valid anchor is required in an ASR that specifies the use of a trial anchor. Recovery: ensure that a valid configuration is specified
50048	`LM_TS_NO_PRODUCT_SET`	The requested operation cannot be completed because the fulfillment record has not been specified. Recovery: ensure that the fulfillment record is specified using the appropriate function call before calling the function that resulted in this error
50049	`LM_TS_TRIALPACKS_ASR_ENTRY`	An error occurred while attempting to load or validate the TRIAL_PACK_IDS entry in an ASR. Recovery: ensure that you are using the XML snippet supplied by Flexera and that it has not been modified
50050	`LM_TS_VIRTUALIZATION_POLICY_MISMATCH`	The ASR being used for local activation has a virtualization policy that excludes the current environment. Recovery: load the ASR in the environment type (physical or virtual) specified by the ASR

Table 8-129 • Activation API Error Codes

Number	Define	Explanation
50051	`LM_TS_VIRTUAL_INTERFACE`	The attempt to retrieve Virtual Machine attributes failed. Recovery: ensure that the FlexNet Licensing Service has been properly installed and is running correctly
50052	`LM_TS_SERVERQUERY_GEN_REQ`	The request to query remote server-side trusted storage could not be generated. Recovery: this is a low memory error; take the action appropriate for the environment, for example, restart
50053	`LM_TS_SERVERQUERY_RESPONSE_FAILED`	The response to a remote server-side trusted-storage query could not be processed. The system error code provides more detailed information. Recovery: as appropriate for the system error code
50054	`LM_TS_SERVERQUERY_FR_NOT_FOUND`	The fulfillment record intended to be queried in remote server-side trusted storage does not exist. Recovery: none
50055	`LM_TS_VM_ATTRIBUTE_PRIVILEGE`	The requested attribute could not be retrieved due to insufficient user privilege. Recovery: ensure that the FlexNet Licensing Service has been properly installed and is running correctly
50056	`LM_TS_VM_ATTRIBUTE_PROTECTED`	The requested attribute could not be retrieved because a protected resource was encountered. Recovery: system-level security policies (for example, SELinux) are preventing a VM attribute from being accessed. Avoid using this VM attribute.
50058	`LM_TS_SHORTCODE`	flxActShortCodeXXX functions—short code action failed, see system error code. Recovery: as appropriate for the system error code
50059	`LM_TS_BADEXP_DATE`	Invalid expiration date passed. Recovery: modify the invalid date string to comply with the date format mandated by Flexera. For information on the valid format, see the *Programming Reference for Trusted Storage–Based Licensing*, section *FulfillmentDictionary Data in Client-side Trusted Storage*, key `expiry`.

Table 8-129 • Activation API Error Codes

Number	Define	Explanation
50060	`LM_TS_TRANSFER_TO_SELF`	Transfer requests to a license server on the same machine are not allowed. Recovery: none
50061	`LM_TS_SERVICE_INFO`	Unable to get information from the service; see system error code.
50062	`LM_TS_NOT_THIS_PLATFORM`	The action or information requested is not supported on this platform.

Note • *Major error codes in the 53000-53007 range are reserved for limited availability functionality.*

See Also
flxActCommonHandleGetLastOpsError
flxActCommonHandleGetLastOpsErrorString

FlxActSystemError

Defines system error codes. These are returned by API calls in the `sysErrorNo` member of the `FlxActError` structure. The `flxActSystemError` adds additional information to the `majorErrorNo`. System error codes are defined in `FlxActSystemError.h` and detailed in the following table.

Table 8-130 • Activation API Error Codes

Number	Define	Explanation
0	`LM_TSSE_NO_ERROR`	No system error code was set by the last API function called.
10101	`LM_TSSE_SERVICE_NOT_PRESENT`	The FlexNet Licensing Service is not present on the system.
10102	`LM_TSSE_SERVICE_NOT_ENOUGH_RIGHTS`	Insufficient rights to connect to the FlexNet Licensing Service.
10103	`LM_TSSE_SERVICE_TOO_OLD`	FlexNet Licensing Service is too old; its version must be the same or greater than the kit used to build this application.
10104	`LM_TSSE_SERVICE_DISABLED`	The FlexNet Licensing Service has been disabled.
10105	`LM_TSSE_SERVICE_MARKED_FOR_DELETE`	The FlexNet Licensing Service is marked for deletion.

Table 8-130 • Activation API Error Codes

Number	Define	Explanation
10106	LM_TSSE_SERVICE_DATABASE_LOCKED	The service control manager database is locked.
10107	LM_TSSE_SERVICE_CONFIGURATION	The FlexNet Licensing Service or its dependencies are configured incorrectly. The event log may contain more information.
10108	LM_TSSE_SERVICE_DID_NOT_START	The FlexNet Licensing Service did not start. The event log may contain more information.
10109	LM_TSSE_SERVICE_READ_ONLY_PORTAL	The call to the FlexNet Licensing Service has failed. The event log may contain more information.
10110	LM_TSSE_SERVICE_NONE	This platform does not run a FlexNet Licensing Service.
10111	LM_TSSE_SERVICE_INFO_NOT_AVAILABLE	The requested information is not available from the FlexNet Licensing Service on this platform.
10121	LM_TSSE_SERVICE_NOT_PRESENT_MAC	The FlexNet Licensing Service is not present on the system (OS X only).
10122	LM_TSSE_SERVICE_FAILED_MAC	The FlexNet Licensing Service failed (OS X only).
10201	LM_TSSE_ASR_PATH	The ASR path supplied cannot be opened as a file or directory.
10202	LM_TSSE_ASR_NOT_XML	The ASR supplied is not well-formed XML.
10203	LM_TSSE_ASR_NOT_ASR	The ASR supplied is not an ASR (XML root element not "ActivationSpecificationRecord").
10204	LM_TSSE_ASR_XML_ELEMENT_MISSING	The ASR supplied has a missing XML element.
10205	LM_TSSE_ASR_SIGNATURE	The ASR signature does not verify.
10206	LM_TSSE_ASR_LOADED_ALREADY	The ASR has been loaded already; its fulfillment exists in Trusted Storage.
10207	LM_TSSE_ASR_LOADED_PREVIOUSLY	The ASR has been loaded previously; its fulfillment (or Trusted Storage) has been deleted at some time.
10208	LM_TSSE_ASR_NONE_IN_DIRECTORY	No ASRs were found in the specified directory.
10209	LM_TSSE_ASR_IS_SHORTCODE	The ASR cannot be loaded because it is for use only in shortcode transactions.

Table 8-130 • Activation API Error Codes

Number	Define	Explanation
10210	LM_TSSE_ASR_HAS_SERVER_CONFIG	The ASR is for server TS and it contains a Trusted Configuration - this is not allowed.
10211	LM_TSSE_ASR_HAS_NO_CONFIG	The ASR has no config - it is mandatory for VersionNumber 5 and earlier.
10212	LM_TSSE_ASR_WRONG_SERVER_MODE	The ASR is for server TS and is being loaded to application TS, or vice versa.
10213	LM_TSSE_ASR_NO_TRUSTED_SECTION	The ASR has a TrustedId for a section that does not exist (trusted config not included in ASR and section not previously created).
10214	LM_TSSE_ASR_NO_RESET_ATTRIBUTE	The ASR cannot be used to reset a trial because it does not have the reset attribute.
10215	LM_TSSE_ASR_ALL_IN_DIRECTORY	All the ASRs in the directory failed to load.
10216	LM_TSSE_ASR_CHECK_IS_DIR	Path for flxActCommonLicSpcCheckASR must be a file not a directory.
10241	LM_TSSE_RESP_NOT_XML_OR_SHORTCODE	Response supplied is not an expected format (not well-formed XML or shortcode).
10242	LM_TSSE_RESP_NOT_XML_RESPONSE	Response supplied is XML but not a response (does not contain a ResponseHeader).
10243	LM_TSSE_RESP_UNKNOWN_TYPE	XML response is of an unknown type (not Activate, Return, Repair or Error).
10244	LM_TSSE_RESP_XML_ELEMENT_MISSING	XML response has a missing element.
10245	LM_TSSE_RESP_SIGNATURE	XML response signature does not verify.
10246	LM_TSSE_RESP_NO_REQUEST	No valid outstanding request for this XML response. (Response already processed, response is for a different machine, request has been cancelled or trust has been lost since the request was created).
10247	LM_TSSE_RESP_NOT_XML	Response supplied is not XML or XML is not well-formed.
10248	LM_TSSE_RESP_SERVER_DENY	XML response is that the request is denied by the server.
10249	LM_TSSE_RESP_SERVER_ERROR	XML response is that the request is an error by the server.

Table 8-130 • Activation API Error Codes

Number	Define	Explanation
10250	LM_TSSE_RESP_NO_KEY	XML response targets a trusted section that does not exist or is untrusted or does not contain the verification key required.
10281	LM_TSSE_SHORTCODE_NO_REQUEST	No valid outstanding request for this shortcode response. (Response already processed, response entered incorrectly, response is for a different machine, request has been cancelled or trust has been lost since the request was created).
10282	LM_TSSE_SHORTCODE_TYPE	Shortcode response type unknown (response code entered incorrectly).
10283	LM_TSSE_SHORTCODE_SIGNATURE	Shortcode response signature did not verify (response entered incorrectly, response is for a different machine, key management issue).
10284	LM_TSSE_SHORTCODE_LOCAL_ASR	The ASR cannot be loaded because it is for local activations only.
10285	LM_TSSE_SHORTCODE_WARN_OVERWRITE	(Warning) activation response replaced an existing fulfillment with the same ID.
10291	LM_TSSE_SHORTCODE_FID_NOT_FOR_ASR	The FID specified for return or repair was not created using the specified ASR.
10292	LM_TSSE_SHORTCODE_FID_NOT_FOUND	No fulfillment record with the FID specified exists.
10293	LM_TSSE_SHORTCODE_NO_FR_FOR_ASR	Cannot return or repair because no FR activated with the specified ASR exits.
10294	LM_TSSE_SHORTCODE_FR_DISABLED	Cannot return because there is a pending return request for the FR
10295	LM_TSSE_SHORTCODE_PENDING	Cannot create request because there is one pending for the ASR
10296	LM_TSSE_SHORTCODE_INVALID_CHAR	Invalid character(s) in shortcode.
10297	LM_TSSE_SHORTCODE_INVALID_CHECK	Short code check digit verification failed (code entered incorrectly).
10298	LM_TSSE_SHORTCODE_BAD_ASR_CDT	Short code ASR 'CheckDigiting' property not 'none', 'all' or 'group'.
10299	LM_TSSE_SHORTCODE_TOO_BIG	Short code exceeds 1024 bits (check ASR overrides).

Table 8-130 • Activation API Error Codes

Number	Define	Explanation
10300	`LM_TSSE_SHORTCODE_OVERRIDE_SIZE`	Short code ASR override size is invalid (non-numeric or greater than UINT32_T_MAX).
10301	`LM_TSSE_SHORTCODE_UNDERFLOW`	Short code does not contain enough bits.
11001	`LM_TSSE_PARAM_POINTER_IS_NULL`	Pointer parameter is NULL.
11002	`LM_TSSE_PARAM_BUFFER_TOO_SMALL`	The buffer supplied is too small.
11101	LM_TSSE_CHECK_EVENT_LOG	Cannot log events. Possible cause: Insufficient permissions of trusted-storage folder or event log file.
11102	LM_TSSE_CHECK_NOT_RUN	Diagnostic check does not run. Advisory: Check not implemented for this platform.
11103	LM_TSSE_CHECK_NO_INVALID	Check number invalid. Possible cause: The `resultArrayLength` passed to the **flxActDiagTrustedStorageHealthCheck** function is greater than `flxTsHcCount`.
11104	LM_TSSE_CHECK_ANCHOR_WRITE	Anchor write error. Possible cause: System, security, or anti-virus software blocked access to the anchor location.
11105	LM_TSSE_CHECK_ANCHOR_READ	Anchor read error. Possible cause: System, security, or anti-virus software blocked the anchor create and did not report an error.
11106	LM_TSSE_CHECK_ANCHOR_DATA	Anchor does not exist. Possible cause: System, security, or anti-virus software blocked the anchor write and did not report an error.
11107	LM_TSSE_CHECK_ANCHOR_REMOVE	Anchor remove error. Possible cause: Unexpected (given write and read worked)
11108	LM_TSSE_CHECK_BINDING_NO_VALUE	Cannot get a value for any binding entity. Possible cause: Unexpected

Table 8-130 • Activation API Error Codes

Number	Define	Explanation
11109	LM_TSSE_CHECK_EXCEPTION	Exception; see event log. Possible cause: Unexpected. Exception thrown by FlexNet Publisher code called by the health check.
11110	LM_TSSE_CHECK_TS_CREATE	Cannot create trusted-storage file. Possible cause: Insufficient permissions of trusted-storage folder or file.
11111	LM_TSSE_CHECK_GET_COMMON	Cannot get common section. Possible cause: Trusted-storage integrity compromised.
11112	LM_TSSE_CHECK_CREATE_SECTION	Cannot create Health Check section. Possible cause: Trusted-storage integrity compromised.
11113	LM_TSSE_CHECK_SET_VALUE	Cannot get set value in Health Check section. Possible cause: Trusted-storage integrity compromised.
11114	LM_TSSE_CHECK_GET_VALUE	Cannot get get value from Health Check section. Possible cause: Trusted-storage integrity compromised.
11115	LM_TSSE_CHECK_WRONG_VALUE	Value read is not value written. Possible cause: Trusted-storage integrity compromised.
11116	LM_TSSE_CHECK_TS_SAVE	Cannot save trusted storage. Possible cause: Insufficient permissions of trusted-storage folder or file. Trusted strorage integrity compromised.
11117	LM_TSSE_CHECK_DELETE_SECTION	Cannot delete Health Check section. Possible cause: Trusted strorage integrity compromised.
11118	LM_TSSE_CHECK_PREPPED_CONFIG	Processing prepped trusted config failed. Possible cause: Config used when prepping application was invalid. Trusted strorage integrity compromised.

Table 8-130 • Activation API Error Codes

Number	Define	Explanation
11119	LM_TSSE_CHECK_OPEN_UNTRUSTED	Cannot open trusted section. Possible cause: Trusted strorage integrity compromised.
11120	LM_TSSE_CHECK_OPEN_TRUSTED	Section is not trusted. Possible cause: Section has lost trust (anchoring and/or binding).
16901	LM_TSSE_REQ_V_XML	Unexpected error creating internal request XML; could be low memory.
17001	LM_TSSE_SHORTCODE_SCHEME_UNKNOWN	The shortcode SchemeId in the ASR is not valid.
17002	LM_TSSE_SHORTCODE_VENDOR_DATA	The vendor data item being added to the shortcode is not valid.
17003	LM_TSSE_SHORTCODE_KEY_MISSING	The shortcode key required is not present in the trusted configuration.
17004	LM_TSSE_RESP_FAILURE_REASON	The ResponseReason in a failure response is not DENIED or ERROR. Validate against XML schema 'T'.
17005	LM_TSSE_SHORTCODE_NO_TS_SECTION	The trusted section needed by the short code ASR does not exist; this is usually because the prepped trusted config of a server activation application does not have the same trusted id as the ASR.
19001	LM_TSSE_LAST_ERROR_NOT_SET	fnpActManGetLastError() failed and no System Error Code was set.
19002	LM_TSSE_LAST_ERROR_STD_EXCEPTION	fnpActManGetLastError() threw a std:exception.
19203	LM_TSSE_NOT_XML	Generic, not XML when XML expected.
19204	LM_TSSE_NOT_SHORTCODE	Generic, not a shortcode.
19205	LM_TSSE_SHORTCODE_ASR_LOAD	Failed to load stored shortcode ASR when processing response.
19206	LM_TSSE_SHORTCODE_ASR_LOAD_GET	Failed to get stored shortcode request after loading.
19207	LM_TSSE_SHORTCODE_NO_FR	FR targeted by shortcode response does not exist.
19208	LM_TSSE_SHORTCODE_REQ_TYPE	Stored shortcode request type unknown.

Table 8-130 • Activation API Error Codes

Number	Define	Explanation
19209	LM_TSSE_SHORTCODE_ALIAS_MISMATCH	Shortcode alias in response does not match that in stored ASR.
19210	LM_TSSE_SERVICE_REGISTER	Could not register the FlexNet Licensing Service.
19211	LM_TSSE_ASR_NOT_DIRECTORY	Converted to LM_TSSE_ASR_PATH.
19212	LM_TSSE_SERVICE_CONNECT_EXCEPTION	Service connect threw unexpected exception.
19213	LM_TSSE_ASR_HAS_MULTIPLE_IDS	Multiple trusted ids in an ASR.
19214	LM_TSSE_ASR_SECTION_NOT_TRUSTED	The target section for an ASR is not fully trusted.
19215	LM_TSSE_ASR_HAS_SERVER_CONFIG	The ASR is for server TS and it contains a Trusted Configuration.
19216	LM_TSSE_CTXN_ELS_UNEXPECTED	Error when performing composite transaction with the FlexNet Operations server. See event log for code location.
19217	LM_TSSE_STD_EXCEPTION	Unexpected std::exception. See trusted storage event log.
19218	LM_TSSE_INTEGER_EXCEPTION	Unexpected integer exception. The event log may contain more information.
19219	LM_TSSE_STRING_EXCEPTION	Unexpected const char* exception. The event log may contain more information.
19220	LM_TSSE_DEFAULT_EXCEPTION	Unexpected exception of unknown type.
19800	LM_TSSE_EXCEPTION_ID_BASE	System error codes 19801-19998 are mapped to exception ids 1 - 198.
19999	LM_TSSE_EXCEPTION_ID_FUTURE	Reserved for (unlikely) exception ids greater than 198.

See Also

FlxActError

Composite Transaction API Functions

Composite transactions are used by publisher activation servers to activate, return, or repair licenses on license servers or FlexEnabled clients. They can include combinations of activations, returns, and repairs in a single multiple-action transaction—a single request from the activation client and a single response from the activation server. Composite transactions can also be used for licensing life-cycle operations between the FlexNet license server and FlexEnabled clients.

The Activation API includes a set of functions used exclusively in the creation of both server and application activation utilities that handle composite transactions. For information on how to create and deploy this type of activation utility, see the chapter "Building an Activation Utility for Composite Transactions" in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*.

Additional Information

Refer to the following for additional information:

- For an overview of license server activation transactions, see the chapter "Transactions to a License Server" in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*.
- For an overview of application activation transactions, see the chapter "Transactions with a FlexEnabled Client" in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*.

Function Summary

The following table provides a summary of the functions described in this section and defined in `FlxActTransaction.h`.

Note • *Most functions return a boolean value to indicate whether they succeeded (TRUE) or failed (FALSE).*

Table 8-131 • Composite Transaction API Functions

Function	Description
flxActTranDictionaryGetByIndex	Gets the corresponding dictionary entry by index.
flxActTranDictionaryGetByKey	Gets the corresponding dictionary entry by key.
flxActTranDictionaryGetCount	Gets the count of entries in the specified dictionary.
flxActTranDictionarySetByKey	Sets the value of the dictionary entry specified by key.
flxActTranReqActionGetAttribute	Gets the value of the specified request attribute.
flxActTranReqActionGetType	Gets the type of request action.
flxActTranReqActionGetVendorDictionary	Gets the vendor dictionary handle of a request action.
flxActTranReqActionSetAttribute	Sets the value of a request action attribute.

Table 8-131 • Composite Transaction API Functions

Function	Description
flxActTranRequestAddAction	Adds actions to a request.
flxActTranRequestAddExistingFulfillment	Add a specific fulfillment to a request.
flxActTranRequestClearExistingFulfillments	Clears all existing fulfillments from a request.
flxActTranRequestGetAction	Gets an action in a response.
flxActTranRequestGetActionCount	Gets the count of actions in a request transaction.
flxActTranRequestGetAttribute	Gets the value of the specified request action attribute.
flxActTranRequestGetStatus	Gets the status of a request.
flxActTranRequestGetVendorDictionary	Gets the vendor dictionary for the request.
flxActTranRequestGetXML	Gets the XML for a request.
flxActTranRequestMatchStored	Checks to see if there is a stored request that exactly matches the current request.
flxActTranRequestSave	Saves the request in trusted storage.
flxActTranRequestSetAttribute	Sets a request attribute.
flxActTranRequestSetExistingFulfillmentDetails	Adds or removes full details of each fulfillment currently in the request.
flxActTranResponseGetAction	Gets an action in a response transaction.
flxActTranResponseGetActionCount	Gets the count of actions in a response.
flxActTranResponseGetVendorDictionary	Gets the vendor dictionary in a response.
flxActTranRspActionGetAttribute	Gets a response action attribute.
flxActTranRspActionGetVendorDictionary	Gets a response action vendor dictionary.
flxActTranRspActionGetResult	Returns the result of performing the action during response processing.
flxActTranRspActionGetType	Gets the response action type.
flxActTransactionCreate	Creates a composite transaction object for use in all other calls.
flxActTransactionDestroy	Destroys a composite transaction object.

Table 8-131 • Composite Transaction API Functions

Function	Description
flxActTransactionCreate	Used when response processing has failed and the server should register this fact.
flxActTransactionCreateRequest	Creates a new request.
flxActTransactionDeleteRequestBySeqNo	Deletes a specified request.
flxActTransactionDeleteStoredRequest	Deletes a stored request.
flxActTransactionGetCommsAttribute	Gets server communications attributes.
flxActTransactionGetRequest	Gets the current request for the transaction.
flxActTransactionGetResponse	Gets the current response from the transaction.
flxActTransactionGetResponseXML	Gets the response XML last processed using the specified transaction handle.
flxActTransactionLoadStoredRequest	Loads a stored request.
flxActTransactionProcessResponse	Processes the response from the server.
flxActTransactionSend	Sends the request to the server, gets and processes the response.
flxActTransactionSetCommsAttribute	Sets server communications attributes.
flxActTransactionGetStoredRequestCount	Gets the count of requests in trusted storage.

flxActTranDictionaryGetByIndex

Returns the dictionary entry corresponding to the specified index (0 = first, 1 = second, ... , (count-1) = last).

Syntax

```
FlxActBool flxActTranDictionaryGetByIndex(
    FlxActTranDictionay hDictionary,
    uint32_t index,
    const char **ppKey,
    const char **ppValue);
```

Parameters

The **flxActTranDictionaryGetByIndex** function uses the following parameters:

Table 8-132 • flxActTranDictionaryGetByIndex Parameters

Parameter	Description
hDictionary	Dictionary handle.
index	Index for which to retrieve the dictionary data.
ppKey	For return of the dictionary entry key.
ppValue	For return of the dictionary entry value.

Return Values

The following values are returned by **flxActTranDictionaryGetByIndex:**

Table 8-133 • flxActTranDictionaryGetByIndex Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. If the index is invalid (greater than count-1), calling **flxActTransactionGetResult** will return **FNP_TRAN_ERROR_INVALID_DICT_INDEX**.

Discussion

You can enumerate the dictionary entries by getting the count using **flxActTranDictionaryGetCount** and calling this function repeatedly for index 0 to (count-1)

flxActTranDictionaryGetByKey

Returns the dictionary entry corresponding to the specified key.

Syntax

```
FlxActBool flxActTranDictionaryGetByKey(
    FlxActTranDictionary hDictionary,
    const char *pKey,
    const char **ppValue);
```

Parameters

The **flxActTranDictionaryGetByKey** function uses the following parameters:

Table 8-134 • flxActTranDictionaryGetByKey Parameters

Parameter	Description
hDictionary	Dictionary handle.
pKey	The key of the entry required.
ppValue	For return of the entry value.

Return Values

The following values are returned by **flxActTranDictionaryGetByKey:**

Table 8-135 • flxActTranDictionaryGetByKey Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. If the key does not exist, calling **flxActTransactionGetResult** will return **FLX_ACT_TRAN_ERR_UNKNOWN_DICT_KEY**.

flxActTranDictionaryGetCount

Returns the count of entries in the specified dictionary.

Syntax

```
FlxActBool flxActTranDictionaryGetCount(
    FlxActTranDictionary hDictionary,
    uint32_t *pCount);
```

Parameters

The **flxActTranDictionaryGetCount** function uses the following parameters:

Table 8-136 • flxActTranDictionaryGetCount Parameters

Parameter	Description
hDictionary	Dictionary handle.
pCount	For return of the dictionary count.

Return Values

The following values are returned by **flxActTranDictionaryGetCount:**

Table 8-137 • flxActTranDictionaryGetCount Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranDictionarySetByKey

Sets the value of the dictionary entry corresponding to the specified key. If an entry with the same key exists its value is overwritten. Otherwise, a new entry is created in the dictionary.

Syntax

```
FlxActBool flxActTranDictionarySetByKey(
    FlxActTranDictionary hDictionary,
    const char *pKey,
    const char *pValue);
```

Parameters

The **flxActTranDictionarySetByKey** function uses the following parameters:

Table 8-138 • flxActTranDictionarySetByKey Parameters

Parameter	Description
hDictionary	Dictionary handle.
pKey	The entry key.
pValue	The entry value.

Return Values

The following values are returned by **flxActTranDictionarySetByKey:**

Table 8-139 • flxActTranDictionarySetByKey Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranReqActionGetAttribute

Gets the value of the specified request action attribute.

Syntax

```
FlxActBool flxActTranReqActionGetAttribute(
    FlxActTranReqAction hTranReqAction,
    FlxActTranReqActAttributeType attributeType,
    const char **ppAttributeValue);
```

Parameters

The **flxActTranReqActionGetAttribute** function uses the following parameters:

Table 8-140 • flxActTranReqActionGetAttribute Parameters

Parameter	Description
hTranReqAction	Handle for transaction request action object.
attributeType	The type of the requested attribute.
ppAttributeValue	For return of the value of the attribute.

Return Values

The following values are returned by **flxActTranReqActionGetAttribute:**

Table 8-141 • flxActTranReqActionGetAttribute Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranReqActionGetType

Gets the type of request action.

Syntax

```
FlxActBool flxActTranReqActionGetType(
    FlxActTranReqAction hTranReqAction,
    FlxActTranReqActionType *pActionType);
```

Parameters

The **flxActTranReqActionGetType** function uses the following parameters:

Table 8-142 • flxActTranReqActionGetType Parameters

Parameter	Description
hTranReqAction	Handle for request action object.
pActionType	For return of the type of request action.

Return Values

The following values are returned by **flxActTranReqActionGetType:**

Table 8-143 • flxActTranReqActionGetType Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranReqActionGetVendorDictionary

Gets the vendor dictionary handle of a request action.

An empty dictionary is created when this function is called for the first time; entries are added using **flxActTranDictionarySetByKey**.

Syntax

```
FlxActBool flxActTranReqActionGetVendorDictionary(
    FlxActTranReqAction hTranReqAction,
    FlxActTranDictionary *phDictionary);
```

Parameters

The **flxActTranReqActionGetVendorDictionary** function uses the following parameters:

Table 8-144 • flxActTranReqActionGetVendorDictionary Parameters

Parameter	Description
hTranReqAction	Handle for request action object.
phDictionary	For return of the vendor dictionary handle.

Return Values

The following values are returned by **flxActTranReqActionGetVendorDictionary:**

Table 8-145 • flxActTranReqActionGetVendorDictionary Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranReqActionSetAttribute

Sets the value of a request action attribute.

Syntax

```
FlxActBool flxActTranReqActionSetAttribute(
    FlxActTranReqAction hTranReqAction,
    FlxActTranReqActAttributeType attributeType,
    const char *pAttributeValue);
```

Parameters

The **flxActTranReqActionSetAttribute** function uses the following parameters:

Table 8-146 • flxActTranReqActionSetAttribute Parameters

Parameter	Description
hTranReqAction	Handle for request action object.
attributeType	Type of the attribute being set.
pAttributeValue	Attribute value to set.

Return Values

The following values are returned by **flxActTranReqActionSetAttribute:**

Table 8-147 • flxActTranReqActionSetAttribute Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranRequestAddAction

A request may contain zero or more actions. Actions may be added to a request object until it is saved. The action object handle returned must be used to set the action properties.

Syntax

```
FlxActBool flxActTranRequestAddAction(
    FlxActTranRequest hTranRequest,
    FlxActTranReqActionType actionType,
    FlxActTranReqAction *phAction);
```

Parameters

The **flxActTranRequestAddAction** function uses the following parameters:

Table 8-148 • flxActTranRequestAddAction Parameters

Parameter	Description
hTranRequest	Handle for request object.
actionType	Type of the action to be added to the request.
phAction	For return of the action.

Return Values

The following values are returned by **flxActTranRequestAddAction:**

Table 8-149 • flxActTranRequestAddAction Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranRequestAddExistingFulfillment

Adds a specific fulfillment to the request. The request can contain details of the fulfillment records that are currently in trusted storage. By default, summary details of all fulfillments are included (FulfillmentId, FulfillmentType, IsTrusted, IsEnabled, TrustedId).

Syntax

```
FlxActBool flxActTranRequestAddExistingFulfillment(
    FlxActTranRequest hTranRequest,
    const char *pFulfillmentId,
    FlxActBool bFullDetails);
```

Parameters

The **flxActTranRequestAddExistingFulfillment** function uses the following parameters:

Table 8-150 • flxActReqAddExistingFulfillment Parameters

Parameter	Description
hTranRequest	Handle for request object.
pFulfillmentId	Id of the fulfillment to be added.
bFullDetails	**FLX_ACT_TRUE**: Complete fulfillment details of the added fulfillment. Full details can be sent - this is intended for use in diagnostic or support situations as it may make the request very large. **FLX_ACT_FALSE**: Summary details.

Return Values

The following values are returned by **flxActTranRequestAddExistingFulfillment:**

Table 8-151 • flxActReqAddExistingFulfillment Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranRequestClearExistingFulfillments

Clears all existing fulfillments from the request.

Syntax

```
FlxActBool flxActTranRequestClearExistingFulfillments(FlxActTranRequest hTranRequest);
```

Parameters

The **flxActTranRequestClearExistingFulfillments** function uses the following parameters:

Table 8-152 • flxActTranRequestClearExistingFulfillments Parameters

Parameter	Description
hTranRequest	Handle for request object.

Return Values

The following values are returned by **flxActTranRequestClearExistingFulfillments:**

Table 8-153 • flxActTranRequestClearExistingFulfillments Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranRequestGetAction

Gets an action in a request.

Syntax

```
FlxActBool flxActTranRequestGetAction(
    FlxActTranRequest hTranRequest,
    uint32_t index,
    FlxActTranReqAction *phTranReqAction);
```

Parameters

The **flxActTranRequestGetAction** function uses the following parameters:

Table 8-154 • flxActTranRequestGetAction Parameters

Parameter	Description
hTranRequest	Handle for request object.
index	Actions are held in a vector in the order they were added; the first has index = 0, the last has index = (count -1).
phTranReqAction	For return of the request action.

Return Values

The following values are returned by **flxActTranRequestGetAction:**

Table 8-155 • flxActTranRequestGetAction Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

Discussion

You can enumerate the actions by getting the count using flxActTranRequestGetActionCount and calling this function repeatedly for index 0 to (count-1).

flxActTranRequestGetActionCount

Returns the number of actions in a request.

Syntax

```
FlxActBool flxActTranRequestGetActionCount(
    FlxActTranRequest hTranRequest,
    uint32_t *pCount);
```

Parameters

The **flxActTranRequestGetActionCount** function uses the following parameters:

Table 8-156 • flxActTranRequestGetActionCount Parameters

Parameter	Description
hTranRequest	Handle for request object.
pCount	For return of the count.

Return Values

The following values are returned by **flxActTranRequestGetActionCount:**

Table 8-157 • flxActTranRequestGetActionCount Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranRequestGetAttribute

Gets the value of a specified request attribute.

Syntax

```
FlxActBool flxActTranRequestGetAttribute(
    FlxActTranRequest hTranRequest,
    FlxActTranReqAttributeType attributeType,
    const char **ppAttributeValue);
```

Parameters

The **flxActTranRequestGetAttribute** function uses the following parameters:

Table 8-158 • flxActTranRequestGetAttribute Parameters

Parameter	Description
hTranRequest	Handle for request object.
attributeType	Type of attribute in the request.
ppAttributeValue	For return of the value of the attribute.

Return Values

The following values are returned by **flxActTranRequestGetAttribute:**

Table 8-159 • flxActTranRequestGetAttribute Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranRequestGetStatus

Requests with status PENDING should normally be retained until the response is received; if necessary they can be resent to the server using **flxActTransactionSend** or **flxActTranRequestGetXML**. However, if there is no prospect of a response they should be deleted using **flxActTransactionDeleteStoredRequest**.

Requests with status UNTRUSTED should also be retained until the response is received. Again, if there is no prospect of a response they should be deleted.

Requests with status COMPLETE should be deleted using **flxActTransactionDeleteStoredRequest** once any response vendor data has been processed.

Syntax

```
FlxActBool flxActTranRequestGetStatus(
    FlxActTranRequest hTranRequest,
    FlxActTranRequestStatus *pStatus);
```

Parameters

The **flxActTranRequestGetStatus** function uses the following parameters:

Table 8-160 • flxActTranRequestGetStatus Parameters

Parameter	Description
hTranRequest	Handle for request object.
pStatus	For return of the status.

Return Values

The following values are returned by **flxActTranRequestGetStatus:**

Table 8-161 • flxActTranRequestGetStatus Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranRequestGetVendorDictionary

Request vendor dictionary; for items defined by the publisher Read/write until the request is saved, then read-only.

Entries are added using **flxActTranDictionarySetByKey**.

Syntax

```
FlxActBool flxActTranRequestGetVendorDictionary(
    FlxActTranRequest hTranRequest,
    FlxActTranDictionary *phDictionary);
```

Parameters

The **flxActTranRequestGetVendorDictionary** function uses the following parameters:

Table 8-162 • flxActTranRequestGetVendorDictionary Parameters

Parameter	Description
hTranRequest	Handle for request object.
phDictionary	For return of the vendor dictionary handle.

Return Values

The following values are returned by **flxActTranRequestGetVendorDictionary:**

Table 8-163 • flxActTranRequestGetVendorDictionary Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranRequestGetXML

Returns the request XML that is to be sent to the server.

Note • *You can manage the server communications yourself by using this function to get the request XML; if the request has not already been saved,* ***flxActTranRequestSave()*** *will be called automatically.*

Syntax

```
FlxActBool flxActTranRequestGetXML(
    FlxActTranRequest hTranRequest,
    const char **ppRequestXML);
```

Parameters

The **flxActTranRequestGetXML** function uses the following parameters:

Table 8-164 • flxActTranRequestGetXML Parameters

Parameter	Description
hTranRequest	Handle for request object.
ppRequestXML	For return of the request XML to be sent to the server.

Return Values

The following values are returned by **flxActTranRequestGetXML:**

Table 8-165 • flxActTranRequestGetXML Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranRequestMatchStored

This function checks to see if there is a stored request that matches the current request exactly. This usually means that the user is regenerating the request for recovery purposes. They could do this by enumerating stored requests and selecting the appropriate one but this function provides the more usable option of simply repeating the request generation with the same actions and attributes.

To support this behavior, call this function immediately before **flxActTranRequestSave**. If a match is found, use the **FLX_ACT_TRAN_REQ_SEQUENCE_NO** attribute of hTranRequestMatch to load the matching stored request.

Matching is optional; you can go ahead and save the new request. Be aware, however, that you cannot include a return action for a fulfillment record that has already been returned by a stored request (because the fulfillment record was disabled when the earlier request was created.

Syntax

```
FlxActBool flxActTranRequestMatchStored(
    FlxActTranRequest hTranRequest,
    FlxActTranRequest *hTranRequestMatch);
```

Parameters

The **flxActTranRequestMatchStored** function uses the following parameters:

Table 8-166 • flxActTranRequestMatchStored Parameters

Parameter	Description
hTranRequest	Handle for request object.
hTranRequestMatch	For return of the handle for request match object.

Return Values

The following values are returned by **flxActTranRequestMatchStored:**

Table 8-167 • flxActTranRequestMatchStored Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranRequestSave

This function saves the request in trusted storage so that it is available when the response is processed. This makes the request object read-only.

Syntax

```
FlxActBool flxActTranRequestSave(FlxActTranRequest hTranRequest);
```

Parameters

The **flxActTranRequestSave** function uses the following parameters:

Table 8-168 • flxActTranRequestSave Parameters

Parameter	Description
hTranRequest	Handle for request object.

Return Values

The following values are returned by **flxActTranRequestSave:**

Table 8-169 • flxActTranRequestSave Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranRequestSetAttribute

Sets a request attribute.

Syntax

```
FlxActBool flxActTranRequestSetAttribute(
    FlxActTranRequest hTranRequest,
    FlxActTranReqAttributeType attributeType,
    const char *pAttributeValue);
```

Parameters

The **flxActTranRequestSetAttribute** function uses the following parameters:

Table 8-170 • flxActTranRequestSetAttribute Parameters

Parameter	Description
hTranRequest	Handle for request object.
attributeType	Type of attribute in the request.

Table 8-170 • flxActTranRequestSetAttribute Parameters

Parameter	Description
pAttributeValue	Value of the specified attribute. ***Note*** • *Null is processed as empty string*

Return Values

The following values are returned by **flxActTranRequestSetAttribute:**

Table 8-171 • flxActTranRequestSetAttribute Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranRequestSetExistingFulfillmentDetails

Adds or remove full details of each fulfillment currently in the request.

Syntax

```
FlxActBool flxActTranRequestSetExistingFulfillmentDetails(
    FlxActTranRequest hTranRequest,
    FlxActBool bFullDetails);
```

Parameters

The **flxActTranRequestSetExistingFulfillmentDetails** function uses the following parameters:

Table 8-172 • flxActTranRequestSetExistingFulfillmentDetails Parameters

Parameter	Description
hTranRequest	Handle for request object.
bFullDetails	**FLX_ACT_TRUE**: Complete fulfillment details of the fulfillment. Full details can be sent - this is intended for use in diagnostic or support situations as it may make the request very large. **FLX_ACT_FALSE**: Summary details.

Return Values

The following values are returned by **flxActTranRequestSetExistingFulfillmentDetails:**

Table 8-173 • flxActTranRequestSetExistingFulfillmentDetails Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

Discussion

This function updates the details of all existing fulfillments in this request. Use **flxActTranRequestAddExistingFulfillment** for individual fulfillments.

flxActTranResponseGetAction

Gets an action in a response transaction.

Syntax

```
FlxActBool flxActTranResponseGetAction(
    FlxActTranResponse hTranResponse,
    uint32_t index,
    FlxActTranRspAction *phTranRspAction);
```

Parameters

The **flxActTranResponseGetAction** function uses the following parameters:

Table 8-174 • flxActTranResponseGetAction Parameters

Parameter	Description
hTranResponse	Handle for response object.
index	Actions are held in a vector in the order they were added; the first has index = 0, the last has index = (count -1).
phTranRspAction	For return of the response action handle.

Return Values

The following values are returned by **flxActTranResponseGetAction:**

Table 8-175 • flxActTranResponseGetAction Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranResponseGetActionCount

Returns the number of actions in a response.

Syntax

```
FlxActBool flxActTranResponseGetActionCount(
    FlxActTranResponse hTranResponse,
    uint32_t *pCount);
```

Parameters

The **flxActTranRequestGetActionCount** function uses the following parameters:

Table 8-176 • flxActTranRequestGetActionCount Parameters

Parameter	Description
hTranResponse	Handle for response object.
pCount	For return of the count.

Return Values

The following values are returned by **flxActTranRequestGetActionCount:**

Table 8-177 • flxActTranRequestGetActionCount Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranResponseGetVendorDictionary

Response vendor dictionary; for items defined by the publisher. Read-only.

Syntax

```
FlxActBool flxActTranResponseGetVendorDictionary(
    FlxActTranResponse hTranResponse,
    FlxActTranDictionary *phDictionary);
```

Parameters

The **flxActTranResponseVendorDictionary** function uses the following parameters:

Table 8-178 • flxActTranRequestGetVendorDictionary Parameters

Parameter	Description
hTranResponse	Handle for response object.
phDictionary	For return of the vendor dictionary.

Return Values

The following values are returned by **flxActTranResponseVendorDictionary:**

Table 8-179 • flxActTranResponseVendorDictionary Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranRspActionGetAttribute

Gets a response action attribute.

Syntax

```
FlxActBool flxActTranRspActionGetAttribute(
    FlxActTranRspAction hTranRspAction,
    FlxActTranRspActAttributeType attributeType,
    const char **ppAttributeValue);
```

Parameters

The **flxActTranRspActionGetAttribute** function uses the following parameters:

Table 8-180 • flxActTranRspActionGetAttribute Parameters

Parameter	Description
hTranRspAction	Handle for response action object.

Table 8-180 • flxActTranRspActionGetAttribute Parameters

Parameter	Description
attributeType	Type of attribute.
ppAttributeValue	For return of the value of the attribute.

Return Values

The following values are returned by **flxActTranRspActionGetAttribute:**

Table 8-181 • flxActTranRspActionGetAttribute Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranRspActionGetVendorDictionary

Gets the vendor dictionary for the response action. Read-only.

Syntax

```
FlxActBool flxActTranRspActionGetVendorDictionary(
    FlxActTranRspAction hTranRspAction,
    FlxActTranDictionary *phDictionary);
```

Parameters

The **flxActTranRspActionGetVendorDictionary** function uses the following parameters:

Table 8-182 • flxActTranRequestGetVendorDictionary Parameters

Parameter	Description
hTranRspAction	Handle for response action object.
phDictionary	For return of the vendor dictionary.

Return Values

The following values are returned by **flxActTranRspActionGetVendorDictionary:**

Table 8-183 • flxActTranRequestGetVendorDictionary Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTranRspActionGetResult

This returns the result of performing the action during response processing.

Syntax

```
FlxActBool flxActTranRspActionGetResult(
    FlxActTranRspAction hTranRspAction,
    FlxActTranRspActionResult *pActionResult);
```

Parameters

The **flxActTranRspActionGetResult** function uses the following parameters:

Table 8-184 • flxActTranRspActionGetResult Parameters

Parameter	Description
hTranRspAction	Handle for response action object.
pActionResult	For return of the result of performing the action

Return Values

The following values are returned by **flxActTranRspActionGetResult:**

Table 8-185 • flxActTranRspActionGetResult Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

Results

On success, **pActionResult** will be set to one of the following values:

Table 8-186 • flxActTranRspActionGetResult Results

Parameter	Description
FLX_ACT_TRAN_RSP_ACT_SUCCESS	Action performed successfully.
FLX_ACT_TRAN_RSP_ACT_WARN_DENY	This is a deny action.
FLX_ACT_TRAN_RSP_ACT_ERR_NA	Not attempted because of another error.
FLX_ACT_TRAN_RSP_ACT_ERR_SKIP	Valid but not performed because of another error.
FLX_ACT_TRAN_RSP_ACT_WARN_FR_OVERWRITE	Fulfillment record already exists.
FLX_ACT_TRAN_RSP_ACT_ERR_NO_FR	Fulfillment record does not exist.
FLX_ACT_TRAN_RSP_ACT_ERR_CONFIG	Unable to import trusted configuration.
FLX_ACT_TRAN_RSP_ACT_WARN_NO_FR	Fulfillment record does not exist (warning on Delete).

flxActTranRspActionGetType

Gets the response action type.

Syntax

```
FlxActBool flxActTranRspActionGetType(
    FlxActTranRspAction hTranRspAction,
    FlxActTranRspActionType *pActionType);
```

Parameters

The **flxActTranRspActionGetType** function uses the following parameters:

Table 8-187 • flxActTranRequestGetType Parameters

Parameter	Description
hTranRspAction	Handle for response action object.
pActionType	For return of the action type.

Return Values

The following values are returned by **flxActTranRspActionGetType:**

Table 8-188 • flxActTranRequestGetType Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTransactionCreate

Creates a composite transaction object for use in all other calls. The **FlxActHandle** must have been obtained from **flxActCommonHandleOpen()**.

This function will succeed provided that the **FlxActHandle** is valid and memory is available.

Syntax

```
FlxActBool flxActTransactionCreate(
    FlxActHandle handle,
    FlxActTransaction *phTransaction,
    FlxActTranResult *pErr);
```

Parameters

The **flxActTransactionCreate** function uses the following parameters:

Table 8-189 • flxActTransactionCreate Parameters

Parameter	Description
handle	Handle for transaction object.
phTransaction	For return of the transaction handle.
pErr	For return of the error code, if any.

Return Values

The following values are returned by **flxActTransactionCreate:**

Table 8-190 • flxActTransactionCreate Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTransactionDestroy

Destroys the object returned by **flxActTransactionCreate** and all its subordinate objects. Also invalidates all handles and pointers returned by functions that use the **FlxActTransaction** handle and the handles of all attached objects.

Syntax

```
FlxActBool flxActTransactionDestroy(FlxActTransaction hTransaction);
```

Parameters

The **flxActTransactionDestroy** function uses the following parameters:

Table 8-191 • flxActTransactionDestroy Parameters

Parameter	Description
hTransaction	Handle for transaction object.

Return Values

The following values are returned by **flxActTransactionDestroy:**

Table 8-192 • flxActTransactionDestroy Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTransactionCreateRequest

This creates a new request object with default attributes and attaches it to the transaction object, replacing any existing request object and making the current response object null.

The reference string supplied with the **pReference** parameter is used to initialize the **FLX_ACT_TRAN_REQ_REFERENCE** attribute. This can be used to identify the request later when enumerating stored requests, for example “Activate Product A”.

The request handle is returned as a convenience; it can also be obtained from the transaction object using **flxActTransactionGetRequest**. The handle can be used to set and get request attributes and actions.

The request object is automatically destroyed with the transaction object. This function will succeed provided that the parameters are valid and memory is available.

Syntax

```
FlxActBool flxActTransactionCreateRequest(
    FlxActTransaction hTransaction,
    const char *pReference,
    FlxActTranRequest *phTranRequest);
```

Parameters

The **flxActTransactionCreateRequest** function uses the following parameters:

Table 8-193 • flxActTransactionCreateRequest Parameters

Parameter	Description
hTransaction	Handle for transaction object.
pReference	The request reference.
phTranRequest	For return of the request.

Return Values

The following values are returned by **flxActTransactionCreateRequest:**

Table 8-194 • flxActTransactionCreateRequest Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTransactionDeleteRequestBySeqNo

The following function can be used to delete any stored request using its sequence number attribute **FLX_ACT_TRAN_REQ_SEQUENCE_NO**. This is useful in situations where you don't want to change the current request, for example when deleting the original request after creating a recovery request.

Syntax

```
FlxActBool flxActTransactionDeleteRequestBySeqNo(
    FlxActTransaction hTransaction,
    const char *seqNo);
```

Parameters

The **flxActTransactionDeleteRequestBySeqNo** function uses the following parameters:

Table 8-195 • flxActTransactionDeleteRequestBySeqNo Parameters

Parameter	Description
hTransaction	Handle for transaction object.
seqNo	Sequence number.

Return Values

The following values are returned by **flxActTransactionDeleteRequestBySeqNo:**

Table 8-196 • flxActTransactionDeleteRequestBySeqNo Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTransactionDeleteStoredRequest

This function deletes the current request (the one that would be returned by flxActTransactionGetRequest) from trusted storage, together with the corresponding response. Call this function after the response has been successfully processed and you have obtained any additional information from it, such as vendor dictionaries.

If this call is not made, for example if there is a crash, the request and response can be loaded in a subsequent session to complete the processing of vendor data.

Syntax

```
FlxActBool flxActTransactionDeleteStoredRequest(FlxActTransaction hTransaction);
```

Parameters

The **flxActTransactionDeleteStoredRequest** function uses the following parameters:

Table 8-197 • flxActTransactionDeleteStoredRequest Parameters

Parameter	Description
hTransaction	Handle for transaction object.

Return Values

The following values are returned by **flxActTransactionDeleteStoredRequest:**

Table 8-198 • flxActTransactionDeleteStoredRequest Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTransactionGetCommsAttribute

Gets the server communication attributes.

Syntax

```
FlxActBool flxActTransactionGetCommsAttribute(
    FlxActTransaction hTransaction,
    FlxActTranServerCommsType attributeType,
    const char **ppAttributeValue);
```

Parameters

The **flxActTransactionGetCommsAttribute** function uses the following parameters:

Table 8-199 • flxActTransactionGetCommsAttribute Parameters

Parameter	Description
hTransaction	Handle for transaction object.
attributeType	Type of server communication attribute, defined by **FlxActTranServerCommsType**.
ppAttributeValue	For return of the value of the attribute.

Return Values

The following values are returned by **flxActTransactionGetCommsAttribute:**

Table 8-200 • flxActTransactionGetCommsAttribute Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTransactionGetRequest

Gets the current request for the transaction.

Given valid parameters, this function will only fail if a request object has not been created or loaded.

Syntax

```
FlxActBool flxActTransactionGetRequest(
    FlxActTransaction hTransaction,
    FlxActTranRequest *phTranRequest);
```

Parameters

The **flxActTransactionGetRequest** function uses the following parameters:

Table 8-201 • flxActTransactionGetRequest Parameters

Parameter	Description
hTransaction	Handle for transaction object.
phTranRequest	For return of the request.

Return Values

The following values are returned by **flxActTransactionGetRequest:**

Table 8-202 • flxActTransactionGetRequest Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTransactionGetResponse

This returns the response object currently attached to the transaction by the most recent **flxActTransactionProcessResponse** or **flxActTransactionLoadStoredRequest** call. It can be used to get response properties such as vendor dictionary entries.

Syntax

```
FlxActBool flxActTransactionGetResponse(
    FlxActTransaction hTransaction,
    FlxActTranResponse *phTranResponse);
```

Parameters

The **flxActTransactionGetResponse** function uses the following parameters:

Table 8-203 • flxActTransactionGetResponse Parameters

Parameter	Description
hTransaction	Handle for transaction object.
phTranResponse	For return of the response object handle.

Return Values

The following values are returned by **flxActTransactionGetResponse:**

Table 8-204 • flxActTransactionGetResponse Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTransactionGetResponseXML

This function returns the response XML that was last processed using the specified transaction handle. The response XML is set by calls to the functions **flxActTransactionSend** and **flxActTransactionProcessResponse**. It is intended mainly for use when **flxActTransactionSend** has failed and the response XML sent by the server needs to be obtained.

Syntax

```
FlxActBool flxActTransactionGetResponseXML(
    FlxActTransaction hTransaction,
    const char *ppResponseXML);
```

Parameters

The **flxActTransactionGetResponseXML** function uses the following parameters:

Table 8-205 • flxActTransactionGetResponseXML Parameters

Parameter	Description
hTransaction	Handle for the transaction object.
ppResponseXML	For the return of a pointer to the response XML. The pointer is valid until the next call to **flxActTransactionSend** or **flxActTransactionProcessResponse** or until the transaction handle is destroyed.

Return Values

The following values are returned by **flxActTransactionGetResponseXML:**

Table 8-206 • flxActTransactionGetResponseXML Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTransactionLoadStoredRequest

When a request is loaded, it becomes the current request for the transaction, replacing any existing request. The request handle is returned as a convenience, it can always be obtained from the transaction object using **flxActTransactionGetRequest**. The request can then be identified from its attributes (**flxActTranRequestGetAttribute**).

If a response to the request has already been processed (status **COMPLETE** or **RECOVER**) it will become the current response. The response handle (from **flxActTransactionGetResponse**) can be used to get attributes, for example vendor dictionary entries.

Syntax

```
FlxActBool flxActTransactionLoadStoredRequest(
    FlxActTransaction hTransaction,
    uint32_t index,
    FlxActTranRequest *phTranRequest);
```

Parameters

The **flxActTransactionLoadStoredRequest** function uses the following parameters:

Table 8-207 • flxActTransactionLoadStoredRequest Parameters

Parameter	Description
hTransaction	Handle for transaction object.
index	Index value of the stored request.
phTranRequest	For return of the handle of the stored request.

Return Values

The following values are returned by **flxActTransactionLoadStoredRequest:**

Table 8-208 • flxActTransactionLoadStoredRequest Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTransactionProcessResponse

This function processes a composite response received from the server.

1. The response XML is validated for:
   - Correct XML header format and namespace.
   - Root element is <Response>

- Mandatory elements are present
- Element <ResponseType> has value COMPOSITE.
- XML version is correct

2. The first signature is validated (if no trusted sections currently exist in trusted storage, this will be the common signature, otherwise the first or only section signature).
3. Requester verification is performed, if specified by the server. This confirms that the response is being processed on the machine that generated the request.
4. The corresponding request is loaded from trusted storage (if it is not already the current request); its status must be PENDING.
5. A response object is created, attached to the transaction object (replacing any existing one) and stored with the request in trusted storage.
6. Any Config actions are performed, creating (or modifying the trusted configuration of) the trusted sections needed to validate other signatures and perform other actions.
7. Any other signatures are validated.
8. The remaining actions are performed, in the following order.
   - Repair
   - Delete
   - Create
   - Modify
   - Deny
9. If all actions succeed, the changes are committed to trusted storage; otherwise all changes are discarded. This means that either all actions succeed or none. Config actions are an exception; any trusted sections that have been created will not be deleted.
10. If all steps are successful the function will return **FNP_ACT_TRUE** and the current response object will have been set. As well as details of the actions that have been performed during processing, the response object contains any vendor defined data sent by the server in the vendor dictionaries; this should be read and acted upon now.
11. If the function returns **FNP_ACT_FALSE**, call **flxActTransactionGetResult** to get error information.
12. Depending on the type of error, a response object may still have been created; you can call flxActTransactionGetResponse to find this out.

Because of the complex nature of response processing, this function has its own category of result codes (**FLX_ACT_TRAN_ERR_RSP_XXX**), though it can set other result codes as well.

Note • *Successful processing does not guarantee that all the actions requested have been granted by the server; if not, the server should have included "deny" response actions. You can check for this by enumerating the response actions.*

All objects created by this function are automatically destroyed with the transaction object.

Syntax

```
FlxActBool flxActTransactionProcessResponse(
    FlxActTransaction hTransaction,
    const char *pResponseXml);
```

Parameters

The **flxActTransactionProcessResponse** function uses the following parameters:

Table 8-209 • flxActTransactionProcessResponse Parameters

Parameter	Description
hTransaction	Handle for transaction object.
pResponseXml	For return of the response XML.

Return Values

The following values are returned by **flxActTransactionProcessResponse:**

Table 8-210 • flxActTransactionProcessResponse Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTransactionSend

Sends the request to the server. This function performs the entire transaction by

1. Getting the request XML using **flxActTranRequestGetXML()**.
2. Sending it to the server and obtaining the response.
3. Calling **flxActTransactionProcessResponse()**—see notes for that function for error handling and how to proceed next.

You must set the server communications attributes first.

The callback function is called with status information as the transaction proceeds; its return value determines whether the transaction should proceed or be canceled (see **flxCommsStatusCodes** and **flxActCallbackReturn** in `flxActCommon.h`).

Syntax

```
FlxActBool flxActTransactionSend(
    FlxActTransaction hTransaction,
    FlxActTranStatusCallback pfnStatus,
    const void *pCallerData);
```

Parameters

The **flxActTransactionSend** function uses the following parameters:

Table 8-211 • flxActTransactionSend Parameters

Parameter	Description
hTransaction	Handle for transaction object.
pfnStatus	For return of the function status.
pCallerData	A pointer to the callback function.

Return Values

The following values are returned by **flxActTransactionSend:**

Table 8-212 • flxActTransactionSend Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

flxActTransactionSetCommsAttribute

Sets the server communications attribute.

Syntax

```
FlxActBool flxActTransactionSetCommsAttribute(
    FlxActTransaction hTransaction,
    FlxActTranServerCommsType attributeType,
    const char *pAttributeValue);
```

Parameters

The **flxActTransactionSetCommsAttribute** function uses the following parameters:

Table 8-213 • flxActTransactionSetCommsAttribute Parameters

Parameter	Description
hTransaction	Handle for transaction object.
attributeType	Type of the server communications attribute.
pAttributeValue	Attribute value.

Table 8-214 • flxActTransactionSetCommsAttribute Attribute type

Parameter	Number	Description
FLX_ACT_TRAN_SVR_COMMS_TYPE	1	The server communication attributes FLEX or SOAP. Default “SOAP”. ● FLX_ACT_TRAN_SVR_COMMS_TYPE_FLEX - FLEX is for communication with the Enterprise License Server ● FLX_ACT_TRAN_SVR_COMMS_TYPE_SOAP - SOAP is for communication with the publisher server.
FLX_ACT_TRAN_SVR_URI	10	The server communications URI attribute. (Mandatory)
Following attributes apply only to “SOAP” communications		
FLX_ACT_TRAN_SVR_PROXY_HOST	21	The server communications proxy server URI. (Optional).
FLX_ACT_TRAN_SVR_PROXY_PORT	22	The server communications proxy server port. (Optional)
FLX_ACT_TRAN_SVR_PROXY_USER	23	The server communications proxy server username. (Optional)
FLX_ACT_TRAN_SVR_PROXY_PASSWORD	24	The server communications proxy server password. (Optional)
FLX_ACT_TRAN_SVR_SSL_CACERT	30	The server communications name of the CA certificate for the open source SSL. (Optional)
FLX_ACT_TRAN_SVR_SSL_CAPATH	31	The server communications path to the CA certificate for the open source SSL. (Optional)
FLX_ACT_TRAN_SVR_TIMEOUT	40	The server communications timeout value. Default 10 seconds.
FLX_ACT_TRAN_SVR_POLL_INTERVAL	41	The server communications poll interval. Default 1 seconds.

Return Values

The following values are returned by **flxActTransactionSetCommsAttribute:**

Table 8-215 • flxActTransactionSetCommsAttribute Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

Discussion

You must set the FLX_ACT_TRAN_SVR_URI attribute and any other optional attributes you require before calling flxActTransactionSend.

flxActTransactionGetStoredRequestCount

You do not need to enumerate stored requests when processing a response, as the correct request will be loaded automatically by the transaction object and set as the current request.

Enumeration is provided so that users can be prompted for outstanding responses and for recovery from error conditions where the requests needs to be re-sent to the server.

Enumerate the stored requests by getting the count then loading each in turn by index (0 to (count-1)) using **flxActTransactionLoadStoredRequest**.

Syntax

```
FlxActBool flxActTransactionGetStoredRequestCount(
    FlxActTransaction hTransaction,
    uint32_t *pCount);
```

Parameters

The **flxActTransactionGetStoredRequestCount** function uses the following parameters:

Table 8-216 • flxActTransactionGetStoredRequestCount Parameters

Parameter	Description
hTransaction	Handle for transaction object.
pCount	For return of the count.

Return Values

The following values are returned by **flxActTransactionGetStoredRequestCount:**

Table 8-217 • flxActTransactionGetStoredRequestCount Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActTransactionGetResult to find out the specific error.

Error and Warning Reporting

If a **flxActTranXXX** function fails it returns **FLX_ACT_FALSE** and sets error information (defined in `FlxActTransaction.h`).

Error information is held per thread. The error information is cleared when the next **flxActTranXXX** function (other than the error reporting function) is called.

flxActTransactionGetResult

Returns the result code defining the error when a **flxActTranXXX** function failed.

Syntax

```
FlxActTranResult flxActTransactionGetResult(void);
```

Parameters

The **flxActTransactionGetResult** function uses no parameters.

Return Values

The function **flxActTransactionGetResult** returns the following values:

Table 8-218 • flxActTransactionGetResult Return Values

Parameter	Description
Response processing errors	
FLX_ACT_TRAN_ERR_RSP_XML_MISSING	Mandatory XML element missing
FLX_ACT_TRAN_ERR_RSP_NOT_COMPOSITE	Not a composite response
FLX_ACT_TRAN_ERR_RSP_XML_VERSION	Response XML version number not supported
FLX_ACT_TRAN_ERR_RSP_XML_OTHER	XML invalid in some other way
FLX_ACT_TRAN_ERR_RSP_SIGNATURE_1	First (or only) signature invalid

Table 8-218 • flxActTransactionGetResult Return Values

Parameter	Description
FLX_ACT_TRAN_ERR_RSP_NO_REQUEST	No stored request for this response
FLX_ACT_TRAN_ERR_RSP_NOT_REQUESTER	Request generated on a different machine
FLX_ACT_TRAN_ERR_RSP_SIGNATURE_2	Second or subsequent signature invalid
FLX_ACT_TRAN_ERR_RSP_ACTION_INFO	Invalid or missing response action attribute
FLX_ACT_TRAN_ERR_RSP_ACTION_FAILED	One or more response actions failed
FLX_ACT_TRAN_ERR_RSP_SIG_VERSION	Response XML signature version number not supported
FLX_ACT_TRAN_ERR_RSP_NOT_XML	Response is not an XML document
FLX_ACT_TRAN_ERR_RSP_REQUEST_NOT_PENDING	Can't invoke response actions when matching request not PENDING
FLX_ACT_TRAN_ERR_RSP_REQUEST_UNTRUSTED	Can't invoke non-config response actions when matching request UNTRUSTED
FLX_ACT_TRAN_ERR_RSP_XML_CONFIG	Config response action XML invalid
FLX_ACT_TRAN_ERR_RSP_SVR_CONFIG	Unable to action server trusted configuration
FLX_ACT_TRAN_ERR_RSP_APP_CONFIG	Unable to action application trusted configuration
FLX_ACT_TRAN_ERR_RSP_XML_CREATE	Create response action XML invalid
FLX_ACT_TRAN_ERR_RSP_XML_DELETE	Delete response action XML invalid
FLX_ACT_TRAN_ERR_RSP_XML_MODIFY	Modify response action XML invalid
FLX_ACT_TRAN_ERR_RSP_XML_REPAIR	Repair response action XML invalid
FLX_ACT_TRAN_ERR_RSP_ELS_MEMORY	Enterprise License Server can't process request—no memory
FLX_ACT_TRAN_ERR_RSP_ELS_REQ_XML	Enterprise License Server can't process request—invalid request XML
FLX_ACT_TRAN_ERR_RSP_ELS_REQ_ACTION	Enterprise License Server can't process request—invalid action in request
General errors	
FLX_ACT_TRAN_ERR_TS_LOAD	Trusted storage load failed

Table 8-218 • flxActTransactionGetResult Return Values

Parameter	Description
FLX_ACT_TRAN_ERR_TS_SAVE	Trusted storage save failed
FLX_ACT_TRAN_ERR_COMMS_INIT	Failed to initialise comms
FLX_ACT_TRAN_ERR_COMMS_SEND	Failed to send request
FLX_ACT_TRAN_ERR_ATTR_UNAVAILABLE	Failed to access attribute
FLX_ACT_TRAN_ERR_FR_DISABLED	Fulfillment record is disabled
FLX_ACT_TRAN_ERR_FR_TRUSTED	Fulfillment record is trusted
FLX_ACT_TRAN_ERR_FR_UNTRUSTED	Fulfillment record is untrusted
FLX_ACT_TRAN_ERR_RESPONSE_NOT_FOUND	No response available
FLX_ACT_TRAN_ERR_REQUEST_NOT_FOUND	No request available
FLX_ACT_TRAN_ERR_WRONG_REQUEST_STATUS	Request not in correct state for attempted operation
FLX_ACT_TRAN_ERR_TS_SECTION_FAILED	Trusted storage section does not exist
FLX_ACT_TRAN_ERR_ELS_CONNECT	Can't connect to the Enterprise License Server
FLX_ACT_TRAN_ERR_ELS_SEND_RECEIVE	Can't communicate with Enterprise License Server
Invalid parameters	
FLX_ACT_TRAN_ERR_FR_NOT_FOUND	Invalid FulfillmentId
FLX_ACT_TRAN_ERR_UNKNOWN_DICT_KEY	Unknown dictionary key
FLX_ACT_TRAN_ERR_INVALID_DICT_INDEX	Invalid dictionary index
FLX_ACT_TRAN_ERR_RESULT_PTR	Bad pointer parameter
FLX_ACT_TRAN_ERR_INVALID_ACTION_INDEX	Invalid action index
FLX_ACT_TRAN_ERR_INVALID_REQUEST_INDEX	Invalid request index
FLX_ACT_TRAN_ERR_REQUEST_READONLY	Request is read-only and cannot be changed
FLX_ACT_TRAN_ERR_PARAM_XML	Invalid XML
FLX_ACT_TRAN_ERR_PARAM_DATE	Invalid date. Use `dd-mmm-yyyy`, `d-mmm-yyyy`, or `permanent`.

See also
flxActTransactionGetFunctionId
flxActTransactionLogError
flxActTransactionGetLogString

flxActTransactionGetFunctionId

Returns the API function ID. This function is provided for cases where the last function called is not known explicitly, for example common error reporting code.

Syntax

```
FlxActTranFunctionId flxActTransactionGetFunctionId(void);
```

Parameters

The **flxActTransactionGetFunctionId** function uses no parameters.

Return

Returns the identity of the API function that failed—a **flxActTransactionXXX_ID** value (see `flxActTransaction.h`).

See also
flxActTransactionGetResult
flxActTransactionLogError
flxActTransactionGetLogString

flxActTransactionLogError

Creates an error log string and writes it to the FlexNet Publisher event file.

The error log string is a printable diagnostic string that contains the result and function ID of the last **FlxActTranXXX** function that was called, as well as other information about the error. Call this function if an unexpected error occurs that may require assistance from Flexera support.

The error log string is encoded and can only be interpreted by Flexera support.

Syntax

```
FlxActBool flxActTransactionLogError(void);
```

Parameters

The **flxActTransactionLogError** function uses no parameters.

Return

The following values are returned by **flxActTransactionLogError:**

Table 8-219 • flxActTransactionLogError Return Values

Return Value	Description
FLX_ACT_TRUE	Success. Error log string for was written to the FlexNet Publisher event file.
FLX_ACT_FALSE	Non-success. No log string created.

See also
flxActTransactionGetFunctionId
flxActTransactionGetResult
flxActTransactionGetLogString

flxActTransactionGetLogString

Returns the error log string, which is a printable diagnostic string that contains the result and function ID of the last **FlxActTranXXX** function that was called (excluding **flxActTransactionGetFunctionId** and **flxActTransactionGetResult**), as well as other information about the error that may be useful in support situations.

The error log string is encoded and can only be interpreted by Flexera support.

Syntax

```
const char* flxActTransactionGetLogString(void);
```

Parameters

The **flxActTransactionGetLogString** function uses no parameters.

Return

Returns the error log string.

See also
flxActTransactionGetFunctionId
flxActTransactionGetResult
flxActTransactionLogError

Short Code Transaction APIs

Function Summary

The following table provides a summary of the functions described in this section and defined in `FlxActShortCode.h`

Table 8-220 • Short Code activation API functions

Function	Description
flxActShortCodeCreate	Creates a short code API object from a short code ASR file.
flxActShortCodeCreateFromBuffer	Creates a short code API object from a short code ASR file from buffer.
flxActShortCodeCancelRequest	Cancels a pending request code
flxActShortCodeDestroy	Destroys the short code context
flxActShortCodeProcessResponse	Processes the available response.
flxActShortCodeGetActivationRequest	Creates an activation request
flxActShortCodeGetDenyReason	Gets the reason sent by the server when it denied a request
flxActShortCodeGetErrorDetail	Returns additional error details.
flxActShortCodeGetPendingRequest	Obtains a pending request code
flxActShortCodeGetPendingRequestProdSpc	Returns details of the fulfillment that is the target of the current return or repair request
flxActShortCodeGetRepairRequest	Generates a short code for a repair
flxActShortCodeGetReturnRequest	Generates a shortcode for a return request
flxActShortCodeGetWarning	Checks for existing fulfillment being overwritten
flxActShortCodeSetVendorData	Sets the vendor data to be included in a request.

flxActShortCodeCreate

Creates a short code object used to create and manage short code requests.

Syntax

```
FlxActBool flxActShortCodeCreate(
    FlxActHandle handle,
    const char *pszAsrFile,
```

```
    FlxActShortCode *phShortCode);
```

Parameters

The flxActShortCodeCreate function uses the following parameters:

Table 8-221 • flxActShortCodeCreate Parameters

Parameter	Description
handle	An activation handle created with flxActCommonHandleOpen with the mode parameter set to FLX_ACT_PREP.
pszAsrFile	Path of short code ASR XML
phShortCode	Used for return of the short code handle

Return Values

The following values are returned by **flxActShortCodeCreate:**

Table 8-222 • flxActShortCodeCreate Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function creates an object, which is required when creating and managing short codes

See Also
flxActShortCodeCreateFromBuffer

flxActShortCodeCreateFromBuffer

Creates a short code object used to create and manage short code requests.

Syntax

```
FlxActBool flxActShortCodeCreateFromBuffer(
    FlxActHandle handle,
    const char *pszAsrBuffer,
    FlxActShortCode *phShortCode);
```

Parameters

The flxActShortCodeCreateFromBuffer function uses the following parameters:

Table 8-223 • flxActShortCodeCreateFromBuffer Parameters

Parameter	Description
handle	An activation handle created with flxActCommonHandleOpen with the mode parameter set to FLX_ACT_P.REP
pszAsrBuffer	Buffer containing short code ASR XML
phShortCode	Used for return of the short code handle

Return Values

The following values are returned by **flxActShortCodeCreateFromBuffer**:

Table 8-224 • flxActShortCodeCreateFromBuffer Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function creates an object, which is required when creating and managing short code requests.

flxActShortCodeCancelRequest

Cancels a pending request.

Syntax

```
FlxActBool flxActShortCodeCancelRequest(FlxActShortCode hShortCode);
```

Parameters

The **flxActShortCodeCancelRequest** function uses the following parameters:

Table 8-225 • flxActShortCodeCancelRequest Parameters

Parameter	Description
hShortCode	Destroys a short code context.

Return Values

The following values are returned by **flxActShortCodeCancelRequest**:

Table 8-226 • flxActShortCodeCancelRequest Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Cancelling a return request re-enables the fulfillment making it usable again. You need to trust that the user has not completed the return on the server, otherwise it's an exploit. The alternative is to delete the fulfillment with the request (user can re-install if necessary).

flxActShortCodeDestroy

Destroys a short code object.

Syntax

```
void flxActShortCodeDestroy(FlxActShortCode hShortCode);
```

Parameters

The **flxActShortCodeDestroy** function uses the following parameters:

Table 8-227 • flxActShortCodeDestroy Parameters

Parameter	Description
hShortCode	A short code handle created with flxActShortCodeCreate or flxActShortCodeCreateFromBuffer.

Return Values

There is no return value.

Discussion

This function destroys the short code object created by flxActShortCodeCreate or flxActShortCodeCreateFromBuffer.

flxActShortCodeProcessResponse

Processes a short code response, when the response is available.

Syntax

```
FlxActBool flxActShortCodeProcessResponse(
    FlxActShortCode handle,
    const char *pszResponse,
    FlxShortCodeType *pResponseType);
```

Parameters

The flxActShortCodeProcessResponse function uses the following parameters:

Table 8-228 • flxActShortCodeProcessResponse Parameters

Parameter	Description
handle	An activation handle created with flxActCommonHandleOpen with the mode parameter set to FLX_ACT_APP.
pszResponse	The response code from the server
pResponseType	For return of the response type

Return Values

The following values are returned by **flxActShortCodeProcessResponse:**

Table 8-229 • flxActShortCodeProcessResponse Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

When the response is available, process it using **flxActShortCodeProcessResponse**.

No context is needed as the ASR is cached in trusted storage when the request is created and it can be identified from the content of the response code.

These functions can be called to get more information about the response processing

- flxActShortCodeGetDenyReason
- flxActShortCodeGetWarning
- flxActShortCodeGetErrorDetail

flxActShortCodeGetActivationRequest

Generates an activation request code based on the ASR in the specified file.

Syntax

```
FlxActBool flxActShortCodeGetActivationRequest(
    FlxActShortCode ShortCode,
    const char **ppszRequest);
```

Parameters

The **flxActShortCodeGetActivationRequest** function uses the following parameters:

Table 8-230 • flxActShortCodeGetActivationRequest Parameters

Parameter	Description
hShortCode	A short code handle created with flxActShortCodeCreate or flxActShortCodeCreateFromBuffer.
ppszRequest	For return of a pointer to the short code request.

Return Values

The following values are returned by **flxActShortCodeGetActivationRequest:**

Table 8-231 • flxActShortCodeGetActivationRequest Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function reads the short code alias from the ASR. It determines whether there is any outstanding transaction (activation, return, or repair) for this short code alias.

If there is an outstanding transaction, it terminates with an error.

If there is no outstanding transaction, it:

- Uses the Trusted Configuration in the ASR to create or update a segment of trusted storage as required.
- Generates an activation request code with a format as defined by the short code scheme ID in the ASR.

When the response code is received, it is processed using **flxActShortCodeProcessResponse** and this completes the activation transaction.

See Also
flxActShortCodeGetDenyReason
flxActAppActivationRespProcess

flxActShortCodeGetDenyReason

Gets the reason sent by the server when it denied a request.

Syntax

```
uint32_t flxActShortCodeGetDenyReason(FlxActHandle handle);
```

Parameters

The **flxActShortCodeGetDenyReason** function uses the following parameters:

Table 8-232 • flxActShortCodeGetDenyReason Parameters

Parameter	Description
handle	An activation handle created with flxActCommonHandleOpen with the mode parameter set to FLX_ACT_PREP.

Return Values

The value returned by flxActShortCodeGetDenyReason is the reason included in the deny response received from the activation server. It is an integer.

Discussion

If the request type returned by **flxActShortCodeProcessResponse** is **flxShortCodeTypeError** then this function will return the reason given by the server for denying the request. If the response was of a different type or the server did not set a reason, it will return zero.

Note • *The* reason *values are defined by the activation server, not FlexNet Publisher.*

flxActShortCodeGetErrorDetail

Returns any additional error details.

Syntax

```
const char *flxActShortCodeGetErrorDetail(FlxActHandle handle);
```

Parameters

The **flxActShortCodeGetErrorDetail** function uses the following parameters:

Table 8-233 • flxActShortCodeGetErrorDetail Parameters

Parameter	Description
handle	As returned by flxActCommonHandleOpen.

Return Values

flxActShortCodeGetErrorDetail returns an error string indicating invalid characters.

Discussion

If the system error returned by **flxActCommonHandleGetError** is `LM_TSSE_SHORTCODE_INVALID_CHAR` or `LM_TSSE_SHORTCODE_INVALID_CHECK`, **flxActShortCodeGetErrorDetail** will return an error string that contains a '?' for each possibly invalid character. The error string should be displayed in a monospaced font underneath the response code to indicate the invalid characters, for example:

```
Processing response: 455976-326180-807501-999982-850286-42#479-264703
                     455976-326180-807501-??????-850286-42?479-264703
ERROR: Response entered incorrectly - check the character(s) replaced by '?' above.
```

Note • *Note that each 6-character-group has a check digit. If any character is incorrect, the entire group is flagged. Characters like '#' which are never valid can be flagged individually.*

flxActShortCodeGetPendingRequest

Returns the request code for an outstanding short code transaction based on the ASR in the specified file.

Syntax

```
FlxActBool flxActShortCodeGetPendingRequest(
    FlxActShortCode hShortCode,
    const char **ppszRequest,
    FlxShortCodeType *pRequestType);
```

Parameters

The **flxActShortCodeGetPendingRequest** function uses the following parameters:

Table 8-234 • flxActShortCodeGetPendingRequest Parameters

Parameter	Description
hShortCode	A short code handle created with flxActShortCodeCreate or flxActShortCodeCreateFromBuffer.
ppszRequest	For returning of a pointer to the short code request
pRequestType	For return of the request type.

Return Values

The following values are returned by **flxActShortCodeGetPendingRequest:**

Table 8-235 • flxActShortCodeGetPendingRequest Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success: either a short code transaction is not in progress or an error has occurred. Call flxActCommonHandleGetError to determine if this was actually an error return or that there is no short code transaction in progress.

Discussion

This function reads the short code alias from the ASR. It determines whether there is a short code transaction (activation, return, or repair) in progress for this short code alias.

If there is a short code transaction in progress, the request code is written to *ppszPendingShortCode*. The function returns success. If there is no short code transaction in progress, the function returns non-success.

See Also
flxActShortCodeCancelRequest

flxActShortCodeGetPendingRequestProdSpc

Returns details of the fulfillment that is the target of the current return or repair request

Syntax

```
FlxActBool flxActShortCodeGetPendingRequestProdSpc(
    FlxActShortCode pShortCode,
    FlxActLicSpc licSpc);
```

Parameters

The **flxActShortCodeGetPendingRequestProdSpc** function uses the following parameters:

Table 8-236 • flxActShortCodeGetPendingRequestProdSpc Parameters

Parameter	Description
pShortCode	A short code handle created with flxActShortCodeCreate or flxActShortCodeCreateFromBuffer
licSpc	For RETURN and REPAIR, FlxActProdSpc of target fulfillment is added to this licSpc

Return Values

The following values are returned by **flxActShortCodeGetPendingRequestProdSpc:**

Table 8-237 • flxActShortCodeGetPendingRequestProdSpc Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Return details of the fulfillment that is the target of the current return or repair request.

flxActShortCodeGetRepairRequest

Generates a short code for a repair based on the ASR in the specified file.

Syntax

```
FlxActBool flxActShortCodeGetRepairRequest(
    FlxActShortCode hShortCode,
    const char *pszFulfillmentId,
    const char **ppszRequest);
```

Parameters

The **flxActShortCodeGetRepairRequest** function uses the following parameters:

Table 8-238 • flxActShortCodeGetRepairRequest Parameters

Parameter	Description
hShortCode	A repair short code object created with flxActShortCodeCreate or flxActShortCodeCreateFromBuffer.
pszFulfillmentId	The fulfillment to be returned
ppszRequest	The returned repair request code.

Return Values

The following values are returned by **flxActShortCodeGetRepairRequest:**

Table 8-239 • flxActShortCodeGetRepairRequest Return Values

Return Value	Description
FLX_ACT_TRUE	Success

Table 8-239 • flxActShortCodeGetRepairRequest Return Values

Return Value	Description
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function determines whether there is any outstanding transaction (activation, return or repair) for this short code alias.

If there is an outstanding transaction, it terminates with an error.

If there is no outstanding transaction, it:

- Generates a repair request code for *PZFulfillmentId* with a format as defined by the short code scheme ID in the ASR.

When the response code is received, it is processed using flxActShortCodeProcessResponse. This completes the repair transaction.

See Also
flxActShortCodeCancelRequest
flxActShortCodeGetDenyReason

flxActShortCodeGetReturnRequest

Generates a short code for a return based on the ASR in the specified file.

Syntax

```
FlxActBool flxActShortCodeGetReturnRequest(
    FlxActShortCode hShortCode,
    const char *pszFulfillmentId,
    const char **ppszRequest);
```

Parameters

The **flxActShortCodeGetReturnRequest** function uses the following parameters:

Table 8-240 • flxActShortCodeGetReturnRequest Parameters

Parameter	Description
hShortCode	A short code handle created with flxActShortCodeCreate or flxActShortCodeCreateFromBuffer.
pszFulfillmentId	A fulfillment to be returned.
ppszRequest	For return of a pointer to the short code request

Return Values

The following values are returned by **flxActShortCodeGetReturnRequest:**

Table 8-241 • flxActShortCodeGetReturnRequest Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function reads the short code alias from the ASR. It determines whether there is any outstanding transaction (activation, return, or repair) for this short code alias.

If an outstanding transaction exists, it is terminated with an error. If no outstanding transaction exists, it does the following:

- Uses the fulfillment record ID in the ASR to determine the unique ID for this fulfillment record within trusted storage. This identifies the fulfillment record that is to be returned.
- Generates a return request code with a format as defined by the short code scheme ID in the ASR.

When the response code is received, it is processed using flxActAppRepairRespProcess and this completes the return transaction.

See Also
flxActAppRepairRespProcess
flxActShortCodeCancelRequest
flxActShortCodeGetDenyReason

flxActShortCodeGetWarning

Returns any warning that was set when processing a response.

Syntax

```
flxShortCodeWarning flxActShortCodeGetWarning(FlxActHandle handle);
```

Parameters

The **flxActShortCodeGetWarning** function uses the following parameters:

Table 8-242 • flxActShortCodeGetWarning Parameters

Parameter	Description
handle	An activation handle created with flxActCommonHandleOpen with the mode parameter set to FLX_ACT_PREP.

Return Values

The following values are returned by **flxActShortCodeGetWarning:**

Table 8-243 • flxActShortCodeGetWarning Return Values

Return Value	Description
flxShortCodeWarningNone	No warning.
flxShortCodeWarningFulfillmentOverwritten	The fulfillment ID (a combination of the ID from the short code ASR and the `SCI_ID_SUFFEX` returned by the server) was the same as an existing fulfillment, which was therefore overwritten.
flxShortCodeWarningInvalidHandle	Handle is invalid (NULL).

flxActShortCodeSetVendorData

Sets the vendor data that is to be included in a request.

Syntax

```
FlxActBool flxActShortCodeSetVendorData(
    FlxActShortCode hShortCode,
    const char *pszItemName,
    const char *pszItemValue);
```

Parameters

The **flxActShortCodeGetWarning** function uses the following parameters:

Table 8-244 • flxActShortCodeSetVendorData Parameters

Parameter	Description
hShortCode	A short code handle created with flxActShortCodeCreate or flxActShortCodeCreateFromBuffer.
pszItemName	The name as defined in the short code ASR
pszItemValue	A value valid for the type specified in the ASR, For example "123" for DEC, "F12A" for HEX, "MyValue" for ASCII96

Return Values

The following values are returned by **flxActShortCodeSetVendorData:**

Table 8-245 • flxActShortCodeSetVendorData Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Vendor data items must be pre-defined in the ASR - name, type and length. This function must be call for each item before calling flxActShortCodeGetActivationRequest, flxActShortCodeGetReturnRequest or flxActShortCodeGetRepairRequest to create the request.

flxShortCodeType

This defines the type of the short code transaction that is reported as being in progress (pending). The following table lists the valid values for flxShortCodeType:

Table 8-246 • flxShortCodeType

Value	Type Code	Description
0	`flxShortCodeTypeNotSet`	The request type was not recorded so cannot be given. The request was generated by a previous version of the API.
1	`flxShortCodeTypeActivation`	Activation short code request.
2	`flxShortCodeTypeReturn`	Return short code request.
3	`flxShortCodeTypeRepair`	Repair short code request.

Deprecated Single-Action Application Activation Functions

In general, application activation functions are used by an application activation utility to manage license rights on the FlexEnabled client. The functions described in this section specifically support *single-action transactions*, which can be used to perform client-side activations from the license server. However, from FlexNet Publisher 11.14.0 it is best practice to use composite transactions for all interactions with servers. Composite transactions perform multiple actions in one transaction.

For information on how to create and deploy an application activation utility that supports single-action transactions, see the chapter “Building an Application Activation Utility for Single-Action Transactions” in FlexNet Publisher’s *Programming Reference for Trusted Storage–Based Licensing*.

The format of this chapter outlines how to use the Activation API to initiate the following activation transactions (which implement the various license lifecycle stages):

- Activation
- Return
- Repair

Additional Information

Refer to the following for additional information:

- For a description of functions used for local activation, see Common Library Functions.
- For a list of deprecated application activation functions, see Deprecated Activation Functions.
- For an overview of application activation transactions, see the chapter "Transactions with a FlexEnabled Client" in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*.
- For a description of functions used by license server activation utilities that handle single-action transactions, see Deprecated License Server Activation Functions. For instructions on how to build these utilities, see the chapter "Building a Server Activation Utility for Single-Action Transactions" in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*.
- For a description of functions used by both application and server activation utilities that handle composite transactions, see Composite Transaction API Functions. For instructions on how to build these activation utilities, see the chapter "Building an Activation Utility for Composite Transactions" in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*.

Function Summary

The following table provides a summary of the functions described in this section and defined in `FlxActApp.h`.

Table 8-247 • Application Activation API Functions

Function	Description
flxActAppActivationCreate	Creates an application activation object.
flxActAppActivationDelete	Deletes an application activation object.
flxActAppActivationSend	Creates and sends an activation request to an activation server and processes the response.
flxActAppActivationReqSet	Creates an activation request. (Replaces the deprecated flxActAppActivationReqCreate function.)
flxActAppActivationRespProcess	Processes an activation response received from an activation server.
flxActAppActivationRespProdLicSpcGet	Gets the FR created in response to an activation request.

Table 8-247 • Application Activation API Functions

Function	Description
flxActAppActivationEntitlementIdGet	Reads the Entitlement ID previously set into the activation object.
flxActAppActivationEntitlementIdSet	Sets the Entitlement ID to be included in an activation request.
flxActAppActivationDurationGet	Reads the license rights duration previously set into the activation object.
flxActAppActivationDurationSet	Sets the license rights duration to be included in an activation request.
flxActAppActivationExpDateGet	Reads the license rights expiry date previously set into the activation object.
flxActAppActivationExpDateSet	Sets the license rights expiry date to be included in an activation request.
flxActAppActivationReasonGet	Reads the activation reason previously set into the activation object.
flxActAppActivationReasonSet	Sets the activation reason to be included in an activation request.
flxActAppActivationProductIdGet	Gets the ProductID from an activation object.
flxActAppActivationProductIdSet	Sets the ProductID for an activation object that will be used in an activation request.
flxActAppActivationVendorDataGet	Reads the vendor data previously set into the activation object.
flxActAppActivationVendorDataSet	Sets the vendor data to be included in an activation request.
flxActAppReturnCreate	Creates an application return object.
flxActAppReturnDelete	Deletes an application return object.
flxActAppReturnSend	Creates and sends a return request to an activation server and processes the response.
flxActAppReturnReqSet	Creates a return request. (Replaces the deprecated flxActAppReturnReqCreate function.)
flxActAppReturnRespProcess	Processes a return response from an activation server
flxActAppReturnCancel	Enables a fulfillment record that has been disabled when a return was requested.

Table 8-247 • Application Activation API Functions

Function	Description
flxActAppReturnProdLicSpcSet	Sets the fulfillment record to be returned
flxActAppReturnProductIdGet	Reads the product ID of the fulfillment record previously set into the return object
flxActAppReturnSuiteIdGet	Reads the suite ID of the fulfillment record previously set into the return object
flxActAppReturnEntitlementIdGet	Reads the Entitlement ID of the fulfillment record previously set into the return object
flxActAppReturnFRIdSet	Sets the fulfillment ID into the return object
flxActAppReturnFRIdGet	Reads the ID of the fulfillment record previously included in the return object
flxActAppReturnReasonGet	Reads the return reason previously set into the return object.
flxActAppReturnReasonSet	Sets the return reason to be included in a return request.
flxActAppReturnVendorDataGet	Reads the vendor data previously set into the return object
flxActAppReturnVendorDataSet	Sets the vendor data to be included in a return request
flxActAppRepairDelete	Deletes an application repair object
flxActAppRepairSend	Creates and sends a repair request to an activation server and processes the response
flxActAppRepairReqSet	Creates a repair request. (Replaces the deprecated function flxActAppRepairReqCreate.)
flxActAppRepairRespProcess	Processes a repair response from an activation server
flxActAppRepairProdLicSpcSet	Sets the fulfillment record to be repaired
flxActAppRepairFRIdSet	Sets the fulfillment ID into the repair object
flxActAppRepairProductIdGet	Reads the product ID of the fulfillment record previously set into the repair object
flxActAppRepairSuiteIdGet	Reads the suite ID of the fulfillment record previously set into the repair object
flxActAppRepairEntitlementIdGet	Reads the entitlement ID of the fulfillment record previously set into the repair object

Table 8-247 • Application Activation API Functions

Function	Description
flxActAppRepairFRIdGet	Reads the ID of the fulfillment record previously set into the repair object
flxActAppRepairVendorDataGet	Reads the vendor data previously set into the repair object
flxActAppRepairVendorDataSet	Sets the vendor data to be included in a repair request
flxActAppActivationDelete	Creates a short code API object from a short code ASR file.

flxActAppActivationCreate

Creates an application activation object. This object is used when creating an activation request and processing an activation response.

Syntax

```
FlxActBool flxActAppActivationCreate(
    FlxActHandle handle,
    FlxActAppActivation *appAct);
```

Parameters

The **flxActAppActivationCreate** function uses the following parameters:

Table 8-248 • flxActAppActivationCreate Parameters

Parameter	Description
handle	An activation handle created with flxActCommonHandleOpen with the mode parameter set to FLX_ACT_APP.
appAct	The resulting pointer to the activation object.

Return Values

The following values are returned by **flxActAppActivationCreate:**

Table 8-249 • flxActAppActivationCreate Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function creates an object, which is required when using any of the following activation functions:

- flxActAppActivationSend
- flxActAppActivationReqSet
- flxActAppActivationRespProcess
- flxActAppActivationEntitlementIdGet
- flxActAppActivationEntitlementIdSet
- flxActAppActivationDurationGet
- flxActAppActivationDurationSet
- flxActAppActivationExpDateGet
- flxActAppActivationExpDateSet
- flxActAppActivationVendorDataGet
- flxActAppActivationVendorDataSet

This function should be paired with a call to **flxActAppActivationDelete**, which releases the object and frees all the associated resources.

flxActAppActivationDelete

Deletes an application activation object.

Syntax

```
void flxActAppActivationDelete(FlxActAppActivation appAct);
```

Parameters

The **flxActAppActivationDelete** function uses the following parameters:

Table 8-250 • flxActAppActivationDelete Parameters

Parameter	Description
appAct	The resulting pointer to the activation object.

Return Values

There are no values returned by **flxActAppActivationDelete**.

Discussion

This function should only be called if a previous call to **flxActAppActivationCreate** succeeded. It deletes the application activation object, including any activation transaction data that it may contain, and frees all the associated resources. Thus, it must not be called while an activation transaction is in progress.

flxActAppActivationSend

Sends an activation request to a remote activation server and process the resulting response.

Syntax

```
FlxActBool flxActAppActivationSend(
    FlxActAppActivation appAct,
    FlxActError *err);
```

Parameters

The **flxActAppActivationSend** function uses the following parameters:

Table 8-251 • flxActAppActivationSend Parameters

Parameter	Description
appAct	An activation object created with flxActAppActivationCreate to which the required parameter values have been added using the **flxActAppActivation*Set** functions.
err	A pointer to the resulting error, if any, see flxActCommonHandleGetError for further details.

Return Values

The following values are returned by **flxActAppActivationSend:**

Table 8-252 • flxActAppActivationSend Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success.

Discussion

Before calling this function you must set the mandatory and optional request parameters into the activation object. The following are request parameters in an activation request:

- **Entitlement ID**—Use flxActAppActivationEntitlementIdSet to set the entitlement ID. This identifies the entitlement for which the license rights are being requested.
- **License rights expiration**—Use either flxActAppActivationExpDateSet to enter a specific expiration date or flxActAppActivationDurationSet to enter the relative duration.
- **Activation reason**—Optionally use flxActAppActivationReasonSet to specify the reason for the activation. When this is not set, the default value of zero is used.
- **Vendor data**—Optionally use flxActAppActivationVendorDataSet to add a key-value pair to the PublisherDictionary in the activation request.

Optionally, the identity and location of the activation server can be set into the activation object using the following functions:

- **flxActCommonHandleSetCommType**—Defines the type of comms used for transactions and therefore the type of activation server. `flxCommsMvsnFlex` is used for connections to a FlexNet license server and `flxCommsMvsnFlexSoap` is used for connections to a FlexNet Operations server or to a License Generator.
- **flxActCommonHandleSetRemoteServer**—Sets the address of the activation server used for transactions.

This function formats an activation request and sends it to an activation server. This function is used for programmatic activation.

The following data is automatically added to the activation request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the activation. See "Information Used to Identify the System Initiating the Request" in the *Programming Reference for Trusted Storage–Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.
- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

The activation server is identified before this function is called, see previous paragraphs, or the following defaults are used:

- When **flxActCommonHandleSetCommType** is used to set `flxCommsMvsnFlex`, the default address of the FlexNet license server is @localhost.
- When **flxActCommonHandleSetCommType** is used to set `flxCommsMvsnSoap`, the default address of the FlexNet Operations server is http://localhost:8888/flexnet/services/ActivationService?wsdl.
- The default setting for the `commType` is `flxCommsMvsnFlex`; the default address of the FlexNet license server is @localhost.

When this process is successful a fulfillment record for the requested license rights is placed in trusted storage. If there is an existing fulfillment record with the same fulfillment ID, then it is deleted.

Non-success is reported in the return value.

flxActAppActivationReqSet

Creates an activation request as a signed XML document for use in manual activation transactions.

Note • *flxActAppActivationReqSet is replaces the deprecated function flxActAppActivationReqCreate.*

Syntax

```
FlxActBool flxActAppActivationReqSet(
    FlxActAppActivation appAct,
    const char **xmlString);
```

Parameters

The **flxActAppActivationReqSet** function uses the following parameters:

Table 8-253 • flxActAppActivationReqSet Parameters

Parameter	Description
appAct	An activation object created with flxActAppActivationCreate to which the required parameter values have been added using the **flxActAppActivation*Set** functions.
xmlString	The resulting request as a signed XML document.

Return Values

The following values are returned by **flxActAppActivationReqSet:**

Table 8-254 • flxActAppActivationReqSet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function you must set the required request parameters into the activation object. The following are request parameters in an activation request:

- **Entitlement ID**—use flxActAppActivationEntitlementIdSet to set the entitlement ID. This identifies the entitlement for which the license rights are being requested.
- **License rights expiration**—use either flxActAppActivationExpDateSet to enter a specific expiration date or flxActAppActivationDurationSet to enter the relative duration.
- **Activation reason**—optionally use flxActAppActivationReasonSet to specify the reason for the activation. When this is not set, the default value of zero is used.
- **Vendor data**—optionally use flxActAppActivationVendorDataSet to add a key-value pair to the PublisherDictionary in the activation request.

This function returns an activation request. The activation request is a signed XML document that must be transmitted without any modification. This method of activation is used for manual activation (that is, where the activation request is transmitted to the activation server in a file). If your code is writing the signed XML document to a file, ensure that it is not modified in any way.

The following data is automatically added to the activation request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the activation. See “Information Used to Identify the System Initiating the Request” in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.

- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.
- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

flxActAppActivationRespProcess

Processes the activation response. Depending on the contents of the activation response it creates a segment of trusted storage or adds a fulfillment record to trusted storage.

Syntax

```
FlxActBool flxActAppActivationRespProcess(
    FlxActAppActivation appAct,
    char *xmlString,
    FlxActBool *pbIsConfig);
```

Parameters

The **flxActAppActivationRespProcess** function uses the following parameters:

Table 8-255 • flxActAppActivationRespProcess Parameters

Parameter	Description
appAct	An activation object created with flxActAppActivationCreate.
xmlString	A pointer to the response to be processed. The response can be either: ● A signed XML document ● A response code
pbIsConfig	Boolean output that indicates whether the activation response contains a Trusted Configuration: ● TRUE = activation response contains a Trusted Configuration. ● FALSE = activation response contains a fulfillment record.

Return Values

The following values are returned by **flxActAppActivationRespProcess:**

Table 8-256 • flxActAppActivationRespProcess Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function processes an activation response. This is sent by an activation server in response to an activation request. See flxActAppActivationReqSet for information on how to create an activation request as an XML document (manual activation), Note that for backward compatibility this function can also be used to process legacy short code responses; this functionality is replaced by **flxActShortCodeOProcessResponse**..

XML Activation Response

This function takes the signed XML response, authenticates it, and if it passes authentication, uses the information it contains to update trusted storage. The activation response can contain either of the following:

- **A fulfillment record**—this is written to trusted storage
- **A Trusted Configuration**—used to create, identify and configure a segment of trusted storage

A fulfillment record can only be written to an existing segment of trusted storage. Thus a manual activation can consist of two activation transactions: the first to create the segment of trusted storage, and the second to write the required fulfillment record to trusted storage. See the *Programming Reference for Trusted Storage–Based Licensing* for a description of the activation process.

If the response contains a fulfillment record with the same fulfillment ID as an existing record in trusted storage, then the existing fulfillment record is deleted. The net effect is that the existing fulfillment record is overwritten.

flxActAppActivationRespProdLicSpcGet

Gets the Fulfillment Record created as a result of the current activation request.

Syntax

```
FlxActBool flxActAppActivationRespProdLicSpcGet(
    FlxActAppActivation act,
    FlxActProdLicSpc *pProdSpc);
```

Parameters

The **flxActAppActivationRespProdLicSpcGet** function uses the following parameters:

Table 8-257 • flxActAppActivationRespProdLicSpcGet Parameters

Parameter	Description
act	An activation object created with flxActAppActivationCreate.
pProdSpc	A pointer to the product license specification returned that contains the Fulfillment Record created as a result of the current activation request.

Return Values

The following values are returned by **flxActAppActivationRespProdLicSpcGet:**

Table 8-258 • flxActAppActivationRespProdLicSpcGet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Call this function after an activation request has been successfully processed and before the activation object is released using **flxActCommonHandleClose**. Thus this function can be called after:

- flxActAppActivationSend
- flxActAppActivationRespProcess

This function returns the product license specification that contains the Fulfillment Record created as a result of the current activation request. You can then query the contents of this structure using any of the **flxActCommonProdLicSpc*Get** functions. The specification remains valid until the next response is processed or the activation object is released.

See Also
flxActCommonProdLicSpc*Get functions

flxActAppActivationEntitlementIdGet

Gets the entitlement ID from the activation object.

Syntax

```
const char *flxActAppActivationEntitlementIdGet(FlxActAppActivation appAct);
```

Parameters

The **flxActAppActivationEntitlementIdGet** function uses the following parameters:

Table 8-259 • flxActAppActivationEntitlementIdGet Parameters

Parameter	Description
appAct	An activation object created with flxActAppActivationCreate.

Return Values

The entitlement ID, or NULL if not set.

Discussion

This function returns the entitlement ID that was previously set in the activation object, or NULL if it has not been set.

See Also
flxActAppActivationEntitlementIdSet

flxActAppActivationEntitlementIdSet

Writes the specified entitlement ID into the activation object.

Syntax

```
void flxActAppActivationEntitlementIdSet(
    FlxActAppActivation appAct,
    const char *entitlementId);
```

Parameters

The **flxActAppActivationEntitlementIdSet** function uses the following parameters:

Table 8-260 • flxActAppActivationEntitlementIdSet Parameters

Parameter	Description
appAct	An activation object created with flxActAppActivationCreate.
entitlementId	The value of the entitlement ID to be included in the activation request.

Return Values

There is no return value.

Discussion

This function writes the specified entitlement ID into the activation object. When this activation object is subsequently used to create an activation request, this entitlement ID is included in the activation request.

See Also
flxActAppActivationSend
flxActAppActivationReqSet

flxActAppActivationDurationGet

Gets the license rights duration value from the activation object.

Syntax

```
uint32_t flxActAppActivationDurationGet(FlxActAppActivation appAct);
```

Parameters

The **flxActAppActivationDurationGet** function uses the following parameters:

Table 8-261 • flxActAppActivationDurationGet Parameters

Parameter	Description
appAct	An activation object created with flxActAppActivationCreate.

Return Values

The license rights duration, or 0 if not set or appAct not valid.

Discussion

This function returns the license rights duration that was previously set in the activation object.

See Also
flxActAppActivationDurationSet

flxActAppActivationDurationSet

Writes the specified duration into the activation object.

Syntax

```
void flxActAppActivationDurationSet(
    FlxActAppActivation appAct,
    uint32_t duration);
```

Parameters

The **flxActAppActivationDurationSet** function uses the following parameters:

Table 8-262 • flxActAppActivationDurationSet Parameters

Parameter	Description
appAct	An activation object created with flxActAppActivationCreate.
duration	The duration value, in seconds, to be included in the activation request. The duration is rounded up to the next whole day worth of seconds before being used.

Return Values

There is no return value.

Discussion

This function writes the specified duration into the activation object. When this activation object is subsequently used to create an activation request, this duration is included in the activation request.

See Also
flxActAppActivationSend
flxActAppActivationReqSet

flxActAppActivationExpDateGet

Gets the license rights expiration date from the activation object.

Syntax

```
const char *flxActAppActivationExpDateGet(FlxActAppActivation appAct);
```

Parameters

The **flxActAppActivationExpDateGet** function uses the following parameters:

Table 8-263 • flxActAppActivationExpDateGet Parameters

Parameter	Description
appAct	An activation object created with flxActAppActivationCreate.

Return Values

The license rights expiration date, or NULL if not set. The format is dd-mmm-yyyy.

Discussion

This function returns the license rights expiration date that was previously set in the activation object, or NULL if it has not been set.

See Also
flxActAppActivationExpDateSet

flxActAppActivationExpDateSet

Writes the specified expiration date into the activation object.

Syntax

```
void flxActAppActivationExpDateSet(
    FlxActAppActivation appAct,
    const char *expDate);
```

Parameters

The **flxActAppActivationExpDateSet** function uses the following parameters:

Table 8-264 • flxActAppActivationExpDateSet Parameters

Parameter	Description
appAct	An activation object created with flxActAppActivationCreate.
expDate	The value of the expiration date to be included in the activation request. The format is `dd-mmm-yyyy` or `permanent`.

Return Values

There is no return value.

Discussion

This function writes the specified expiration date into the activation object. When this activation object is subsequently used to create an activation request, this expiration date is included in the activation request.

See Also
flxActAppActivationSend
flxActAppActivationReqSet

flxActAppActivationReasonGet

Gets the reason for the activation from the specified activation object.

Syntax

```
const char *flxActAppActivationReasonGet(FlxActAppActivation appAct);
```

Parameters

The **flxActAppActivationReasonGet** function uses the following parameters:

Table 8-265 • flxActAppActivationReasonGet Parameters

Parameter	Description
appAct	An activation object created with flxActAppActivationCreate.

Return Values

The value of the activation reason or NULL if not set.

Discussion

This function returns the activation reason that was previously set in the activation object or NULL if it has not been set.

See Also
flxActAppActivationReasonSet

flxActAppActivationReasonSet

Writes the specified reason for the activation into the activation object.

Syntax

```
FlxActBool flxActAppActivationReasonSet(
    FlxActAppActivation appAct,
    const char *pszReason);
```

Parameters

The **flxActAppActivationReasonSet** function uses the following parameters:

Table 8-266 • flxActAppActivationReasonSet Parameters

Parameter	Description
appAct	An activation object created with flxActAppActivationCreate.
pszReason	A pointer to the variable storing the numeric string that will be included in the activation request. This is a publisher defined coded value that indicates the reason that the activation is requested. Note that decimal numbers 0 to 9 are the only valid characters in this string. Alpha characters cannot be used.

Return Values

The following values are returned by **flxActAppActivationReasonSet**:

Table 8-267 • flxActAppActivationReasonSet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function writes the specified numeric code for the reason for the activation request into the activation object. When this activation object is subsequently used to create an activation request, this reason code is included in the activation request.

See Also
flxActAppActivationReasonGet

flxActAppActivationProductIdGet

Gets the ProductID from the specified activation object.

Syntax

```
const char *flxActAppActivationProductIdGet(FlxActAppActivation appAct);
```

Parameters

The **flxActAppActivationProductIdGet** function uses the following parameters:

Table 8-268 • flxActAppActivationProductIdGet Parameters

Parameter	Description
appAct	An activation object created with flxActAppActivationCreate.

Return Values

The value of the ProductID or NULL if not set.

Discussion

This function returns the ProductID that was previously set in the activation object or NULL if it has not been set.

See Also
flxActAppActivationProductIdSet

flxActAppActivationProductIdSet

Writes the specified ProductID into the activation object.

Syntax

```
void flxActAppActivationProductIdSet(
    FlxActAppActivation appAct,
    const char *productId);
```

Parameters

The **flxActAppActivationProductIdSet** function uses the following parameters:

Table 8-269 • flxActAppActivationProductIdSet Parameters

Parameter	Description
appAct	An activation object created with flxActAppActivationCreate.
productId	A pointer to the variable storing the value of the ProductID that will be included in the activation request.

Return Values

There is no return value.

Discussion

This function writes the specified ProductID into the activation object. When this activation object is subsequently used to create an activation request, this ProductID is included in the activation request.

See Also
flxActAppActivationProductIdGet

flxActAppActivationVendorDataGet

Gets the vendor data for the specified key from the activation object.

Syntax

```
const char *flxActAppActivationVendorDataGet(
    FlxActAppActivation appAct
    const char *pszKey);
```

Parameters

The **flxActAppActivationVendorDataGet** function uses the following parameters:

Table 8-270 • flxActAppActivationVendorDataGet Parameters

Parameter	Description
appAct	An activation object created with flxActAppActivationCreate.
pszKey	The key that identifies the vendor data required.

Return Values

The value of the vendor data for the specified key or NULL if not set.

Discussion

This function returns the vendor data that was previously set in the activation object for the specified key, or NULL if it has not been set.

This function does a linear search of all the vendor data key-value pairs in the activation object.

See Also
flxActAppActivationVendorDataSet

flxActAppActivationVendorDataSet

Writes the specified vendor data into the activation object.

Syntax

```
void flxActAppActivationVendorDataSet(
    FlxActAppActivation appAct,
    const char *key,
    const char *value);
```

Parameters

The **flxActAppActivationVendorDataSet** function uses the following parameters:

Table 8-271 • flxActAppActivationVendorDataSet Parameters

Parameter	Description
appAct	An activation object created with flxActAppActivationCreate.
key	This identifies the vendor data: vendor data is added as a key-value pair to the activation request in the vendor dictionary (XML element VendorDictionary).
value	The value of the vendor data to be included in the activation request. This is treated as a null terminated string and no further processing is done to it before placing it in the activation request.

Return Values

There is no return value.

Discussion

This function writes the specified vendor data into the activation object. When this activation object is subsequently used to create an activation request, this vendor data is included in the activation request.

The vendor data is added to the VendorDictionary element in the activation request XML document. All data dictionaries contain a set of entries. The first element in the entry is the Key element that identifies the type of data in this entry. The second element of the entry, the Value element, contains the data for the entry. This type of entry is referred to here and elsewhere as a *key-value pair*.

Note • *If the key or value contain any of the five standard XML reserved characters <, >, ", ', and &, these must be encoded (as: `<`, `>`, `"`, `'`, `&`, respectively) before calling this function.*

This function adds a single key-value pair to the activation object. To add multiple key-value pairs, call this function multiple times.

In a test environment, the vendor data contained within an activation request can be viewed using the GUI version of the Manual Response Generator, responsegenUI.exe. If you are using the command-line version, `responsegen.*`, you will have to examine the activation request before using the Manual Response Generator to generate the appropriate response. For details about the Manual Response Generator, see the *Programming Reference for Trusted Storage–Based Licensing*.

See the FlexNet Operations documentation for information about how vendor data is used by the Operations Server.

When vendor data is added to a request code, it will be modified if the supplied string is too long. The size of the item in the request code is defined by a request override in the ASR (see references to RequestOverrides element in the *Programming Reference for Trusted Storage–Based Licensing* for details). The required number of least significant characters are extracted from the supplied string.

For example if 5 digits are defined for the Unique Machine Number in the ASR then the following 40 digit UMN, 7851233534523409852359823610638450975643 is added to the request code as 75643.

Note • *Activation from a license server—if this function is used to add vendor data to an activation request sent to a license server, the vendor data is ignored by the license server.*

See Also
flxActAppActivationSend
flxActAppActivationReqSet

flxActAppReturnCreate

Creates an application return object. This object is used when creating a return request and processing a return response.

Syntax

```
FlxActBool flxActAppReturnCreate(
    FlxActHandle handle,
    FlxActAppReturn *appRet);
```

Parameters

The **flxActAppReturnCreate** function uses the following parameters:

Table 8-272 • flxActAppReturnCreate Parameters

Parameter	Description
handle	An activation return handle created with flxActCommonHandleOpen with the mode parameter set to FLX_ACT_APP.
appRet	The resulting pointer to the application return object.

Return Values

The following values are returned by **flxActAppReturnCreate:**

Table 8-273 • flxActAppReturnCreate Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success: Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function creates an object, which is required when using any of the following activation functions:

- flxActAppReturnSend
- flxActAppReturnReqSet
- flxActAppReturnRespProcess
- flxActAppReturnProdLicSpcSet
- flxActAppReturnProductIdGet
- flxActAppReturnSuiteIdGet
- flxActAppReturnEntitlementIdGet
- flxActAppReturnFRIdGet

This function should be paired with a call to **flxActAppReturnDelete**, which releases the object and frees all the associated resources.

flxActAppReturnDelete

Delete an application return object.

Syntax

```
void flxActAppReturnDelete(FlxActAppReturn appRet);
```

Parameters

The **flxActAppReturnDelete** function uses the following parameters:

Table 8-274 • flxActAppReturnDelete Parameters

Parameter	Description
appRet	A return object created with flxActAppReturnCreate.

Return Values

There is no return value.

Discussion

This function should only be called if a previous call to **flxActAppReturnCreate** succeeded.

It deletes the application return object, including any return transaction data that it may contain, and frees all the associated resources. Thus, it must not be called while a return transaction is in progress.

flxActAppReturnSend

Send a return request to a remote activation server and automatically process the response.

Syntax

```
FlxActBool flxActAppReturnSend(
    FlxActAppReturn appRet,
    FlxActError *err);
```

Parameters

The **flxActAppReturnSend** function uses the following parameters:

Table 8-275 • flxActAppReturnSend Parameters

Parameter	Description
appRet	A return object created with flxActAppReturnCreate to which the required parameter values have been added using the flxActAppReturnProdLicSpcSet function.
err	A pointer to the resulting error, if any, see flxActCommonHandleGetError for further details.

Return Values

The following values are returned by **flxActAppReturnSend:**

Table 8-276 • flxActAppReturnSend Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success.

Discussion

Before calling this function you must set the fulfillment record that is to be returned into the return object using flxActAppReturnProdLicSpcSet.

Optionally use flxActAppReturnReasonSet to specify the reason for the return. When this is not set, the default value of zero is used.

Optionally, the identity and location of the activation server can be set into the return object using the following functions:

- flxActCommonHandleSetCommType—defines the type of comms used for transactions and therefore the type of activation server. **flxCommsMvsnFlex** is used for connections to a FlexNet license server and **flxCommsMvsnFlexSoap** is used for connections to a FlexNet Operations server or to a License Generator.
- flxActCommonHandleSetRemoteServer—sets the address of the activation server used for transactions.

This function formats a return request and sends it to an activation server.

The following data is automatically added to the return request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the return. See "Information Used to Identify the System Initiating the Request" in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.
- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

The activation server is identified before this function is called, see previous paragraphs, or the following defaults are used:

- When **flxActCommonHandleSetCommType** is used to set **flxCommsMvsnFlex**, the default address of the FlexNet license server is @localhost.
- When **flxActCommonHandleSetCommType** is used to set **flxCommsMvsnSoap**, the default address of the FlexNet Operations server is http://localhost:8888/flexnet/services/ActivationService?wsdl.
- The default setting for the `commType` is **flxCommsMvsnFlex**, and then the default address of the FlexNet license server is @localhost.

A license server only accepts the return of the fulfillment record when it originally supplied it. When the license server accepts the return of the fulfillment record, it replies success and the fulfillment record is deleted from the application's trusted storage. When the license server returns non-success, the fulfillment record is not deleted from the application's trusted storage and this function returns non-success.

The Operations server or License Generator processes the return request according to its internal logic. When it replies success the fulfillment record is deleted from the application's trusted storage. When it returns non-success, the fulfillment record is not deleted from the applications trusted storage and this function returns non-success.

A processing failure may result in no response or an invalid response being received from the activation server. In this case the fulfillment record will remain disabled and the license rights it contains unavailable unless it is enabled using flxActAppReturnCancel.

flxActAppReturnReqSet

Create a return request as a signed XML document for use in manual return transactions.

Note • *flxActAppReturnReqSet replaces the deprecated function flxActAppReturnReqCreate.*

Syntax

```
FlxActBool flxActAppReturnReqSet(
    FlxActAppReturn appRet,
    const char **xmlString);
```

Parameters

The **flxActAppReturnReqSet** function uses the following parameters:

Table 8-277 • flxActAppReturnReqSet Parameters

Parameter	Description
appRet	A return object created with flxActAppReturnCreate to which the required parameter values have been added using the **flxActAppReturnProdLicSpcSet** function.
xmlString	The resulting request as a signed XML document.

Return Values

The following values are returned by **flxActAppReturnReqSet:**

Table 8-278 • flxActAppReturnReqSet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function you must set the fulfillment record that is to be returned into the return object using flxActAppReturnProdLicSpcSet.

Optionally use flxActAppReturnReasonSet to specify the reason for the return. When this is not set, the default value of zero is used.

This function returns a return request. The return request is a signed XML document that must be transmitted without any modification. This transaction method is normally used for manual transfers, that is, where the return request is transmitted to the activation server in a file. If your code is writing the signed XML document to a file, ensure that it is not modified in any way.

The following data is automatically added to the return request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the return. See “Information Used to Identify the System Initiating the Request” in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.
- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

flxActAppReturnRespProcess

Process the return response. Depending on the contents of the return response it either deletes the returned fulfillment record from the application's trusted storage or leaves it in trusted storage and reports an error.

Syntax

```
FlxActBool flxActAppReturnRespProcess(
    FlxActAppReturn appRet,
    char *xmlString,
    FlxActBool *pbIsConfig);
```

Parameters

The **flxActAppReturnRespProcess** function uses the following parameters:

Table 8-279 • flxActAppReturnRespProcess Parameters

Parameter	Description
appRet	A return object created with flxActAppReturnCreate.
xmlString	A pointer to the response to be processed. The response can be either of: ● A signed XML document ● A response code
pbIsConfig	Boolean output that indicates whether the return response contains a Trusted Configuration: ● FLX_ACT_TRUE = The response contains a Trusted Configuration. Trusted storage has been updated with the new Trusted Configuration. The server determined that this was a pre-requisite to processing the request. The original request should be sent to the server again. ● FLX_ACT_FALSE = The response does not contain a Trusted Configuration. The transaction is complete, no further action is required.

Return Values

The following values are returned by **flxActAppReturnRespProcess:**

Table 8-280 • flxActAppReturnRespProcess Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function processes a return response. This is sent by an activation server in response to a return request. See flxActAppReturnReqSet for information on how to create a return request as an XML document, and flxActShortCodeGetReturnRequest for information on how to create a request code from an ASR held either in a file or memory buffer, respectively.

XML Activation Response

The return response can contain either of the following:

- A response to the return request
- A Trusted Configuration—used to create a segment of trusted storage

The license server only accepts the return of the fulfillment record when it originally supplied it. When the license server accepts the return of the fulfillment record, it replies success and this function deletes the fulfillment record from the application's trusted storage. When the license server returns non-success, the fulfillment record is not deleted from the application's trusted storage and this function returns non-success.

If the response contained a Trusted Configuration (`pbIsConfig = TRUE`), it is used to do one of the following:

- Update the configuration for an existing segment of trusted storage. This is the more probable scenario for a Trusted Configuration in a return response.
- Create a segment of trusted storage.

Response Code

This function takes the response code and validates it against the outstanding short code transaction for the short code alias. If the response code is valid and does not contain an error response, then this function deletes the fulfillment record from the application's trusted storage.

Failure Response or No Response Received

When a failure response is processed, the fulfillment record is re-enabled and the license rights it contains can once again be used. The failure response can be because the return of the fulfillment record was denied by the activation server or because there was a failure processing the return request.

A processing failure may result in no response or an invalid response being received from the activation server. In this case the fulfillment record will remain disabled and the license rights it contains unavailable unless it is enabled using flxActAppReturnCancel.

flxActAppReturnCancel

Enable a fulfillment record that has been disabled when a return was requested.

Syntax

```
FlxActBool flxActAppReturnCancel(FlxActAppReturn appRet);
```

Parameters

The **flxActAppReturnCancel** function uses the following parameters:

Table 8-281 • flxActAppReturnCancel Parameters

Parameter	Description
appRet	A return object created with flxActAppReturnCreate that has the fulfillment record whose return was requested set into it.

Return Values

The following values are returned by **flxActAppReturnCancel:**

Table 8-282 • flxActAppReturnCancel Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function create the return object and set the fulfillment record whose return was requested into the return object using flxActAppReturnProdLicSpcSet.

This function is used to enable a fulfillment record when a return was requested and no response has been received. When a return request is sent the fulfillment record is disabled. When a response is received the fulfillment record is either deleted (when the return is allowed by the activation server) or enabled (when the return is denied by the activation server or the return processing fails). When no response is received, the fulfillment record will remain disabled and the license rights unavailable indefinitely.

You have three options at this point:

- **Do nothing** - This leaves the fulfillment record disabled and the license rights unavailable.
- **Delete the fulfillment record** - See “Delete a Single Fulfillment Record from Trusted Storage” in the *Programming Reference for Trusted Storage–Based Licensing* for detailed instructions. This might remove the fulfillment record when the activation server and associated systems record that the customer still has these license rights.
- **Enable the fulfillment record** - This may give the customer extra license rights if the activation server and associated systems have recorded that the license rights were returned.

For example code that demonstrates the use of this function, see the example activation utility, `appactutil`. The `appactutil` command line syntax to demonstrate the use of **flxActAppReturnCancel** is:

```
appactutil -return <fulfillmentID> -cancel
```

flxActAppReturnProdLicSpcSet

Sets the fulfillment record to be returned into the return object.

Syntax

```
void flxActAppReturnProdLicSpcSet(
    FlxActAppReturn appRet,
    FlxActProdLicSpc product);
```

Parameters

The **flxActAppReturnProdLicSpcSet** function uses the following parameters:

Table 8-283 • flxActAppReturnProdLicSpcSet Parameters

Parameter	Description
appRet	A return object created with flxActAppReturnCreate.
product	A product license specification object that contains the fulfillment record that is to be returned.

Return Values

There is no return value.

Discussion

Before calling this function select the fulfillment record that is to be returned. Use the **flxActCommonLicSpc*** and **flxActCommonProdLicSpc*** functions to read the fulfillment records from trusted storage and determine which record is to be returned.

This function sets the required request parameters into the return object. It extracts the required parameters from the fulfillment record in the product license specification.

flxActAppReturnProductIdGet

Get the product ID from the return object.

Syntax

```
const char *flxActAppReturnProductIdGet(FlxActAppReturn appRet);
```

Parameters

The **flxActAppReturnProductIdGet** function uses the following parameters:

Table 8-284 • flxActAppReturnProductIdGet Parameters

Parameter	Description
appRet	A return object created with flxActAppReturnCreate.

Return Values

The product ID, or NULL if not set.

Discussion

This function returns the product ID from the fulfillment record that was previously set in the return object, or NULL if it has not been set.

See Also
flxActAppReturnProdLicSpcSet

flxActAppReturnSuiteIdGet

Get the suite ID from the return object.

Syntax

```
const char *flxActAppReturnSuiteIdGet(FlxActAppReturn appRet);
```

Parameters

The **flxActAppReturnSuiteIdGet** function uses the following parameters:

Table 8-285 • flxActAppReturnSuiteIdGet Parameters

Parameter	Description
appRet	A return object created with flxActAppReturnCreate.

Return Values

The suite ID, or NULL if not set.

Discussion

This function returns the suite ID from the fulfillment record from the fulfillment record that was previously set in the return object, or NULL if it has not been set.

See Also
flxActAppReturnProdLicSpcSet

flxActAppReturnEntitlementIdGet

Get the entitlement ID from the return object.

Syntax

```
const char *flxActAppReturnEntitlementIdGet(FlxActAppReturn appRet);
```

Parameters

The **flxActAppReturnEntitlementIdGet** function uses the following parameters:

Table 8-286 • flxActAppReturnEntitlementIdGet Parameters

Parameter	Description
appRet	A return object created with flxActAppReturnCreate.

Return Values

The entitlement ID, or NULL if not set.

Discussion

This function returns the entitlement ID from the fulfillment record that was previously set in the return object, or NULL if it has not been set.

See Also
flxActAppReturnProdLicSpcSet

flxActAppReturnFRIdSet

Set the fulfillment ID into the return object. This function is **only** required when the fulfillment record was not activated using a short code transaction but is to be returned using a short code transaction.

Syntax

```
void flxActAppReturnFRIdSet(
    FlxActAppReturn appRet,
    const char *pszID);
```

Parameters

The **flxActAppReturnFRIdSet** function uses the following parameters:

Table 8-287 • flxActAppReturnFRIdSet Parameters

Parameter	Description
appRet	A return object created with flxActAppReturnCreate.
pszID	The fulfillment ID of the record that is to be returned.

Return Values

There is no return value.

Discussion

Read "Programmatic Activation Followed by Short Code Return or Repair" in the *Programming Reference for Trusted Storage–Based Licensing* for an explanation of the sequence of transactions that require the use of this function.

This function sets the specified fulfillment ID into the return object. The return object can then be used to return the fulfillment record using a short code transaction.

See Also
flxActAppRepairCreate

flxActAppReturnFRIdGet

Get the fulfillment ID from the return object.

Syntax

```
const char *flxActAppReturnFRIdGet(FlxActAppReturn appRet);
```

Parameters

The **flxActAppReturnFRIdGet** function uses the following parameters:

Table 8-288 • flxActAppReturnFRIdGet Parameters

Parameter	Description
appRet	A return object created with flxActAppReturnCreate.

Return Values

The fulfillment ID, or NULL if not set.

Discussion

This function returns the fulfillment ID from the fulfillment record that was previously set in the return object, or NULL if it has not been set.

See Also
flxActAppReturnProdLicSpcSet

flxActAppReturnReasonGet

Get the reason for the return from the specified return object.

Syntax

```
const char *flxActAppReturnReasonGet(FlxActAppReturn ret);
```

Parameters

The **flxActAppReturnReasonGet** function uses the following parameters:

Table 8-289 • flxActAppReturnReasonGet Parameters

Parameter	Description
ret	A return object created with flxActAppReturnCreate.

Return Values

The value of the return reason, or NULL if not set.

Discussion

This function returns the reason for the return request that was previously set in the return object or NULL if it has not been set.

See Also
flxActAppReturnReasonSet

flxActAppReturnReasonSet

Write the specified reason for the return into the return object.

Syntax

```
FlxActBool flxActAppReturnReasonSet(
    FlxActAppActivation appAct,
    const char *pszReason);
```

Parameters

The **flxActAppReturnReasonSet** function uses the following parameters:

Table 8-290 • flxActAppReturnReasonSet Parameters

Parameter	Description
appAct	An activation object created with flxActAppReturnCreate.
pszReason	A pointer to the variable storing the numeric string that will be included in the return request. This is a publisher defined coded value that indicates the reason that the return is requested.

Return Values

The following values are returned by **flxActAppReturnReasonSet**:

Table 8-291 • flxActAppReturnReasonSet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function writes the specified numeric code for the reason for the return request into the return object. When this return object is subsequently used to create a return request, this reason code is included in the return request.

See Also
flxActAppReturnReasonGet

flxActAppReturnVendorDataGet

Get the vendor data for the specified key from the return object.

Syntax

```
const char *flxActAppReturnVendorDataGet(
    FlxActAppReturn ret,
    const char *key);
```

Parameters

The **flxActAppReturnVendorDataGet** function uses the following parameters:

Table 8-292 • flxActAppReturnVendorDataGet Parameters

Parameter	Description
ret	A return object created with flxActAppReturnCreate.
key	The key that identifies the vendor data required.

Return Values

The value of the vendor data for the specified key, or NULL if not set.

Discussion

This function returns the vendor data that was previously set in the return object for the specified key, or NULL if it has not been set.

This function does a linear search of all the vendor data key-value pairs in the return object.

See Also
flxActAppReturnVendorDataSet

flxActAppReturnVendorDataSet

Write the specified vendor data into the return object.

Syntax

```
void flxActAppReturnVendorDataSet(
    FlxActAppReturn ret,
    const char *key,
    const char *value);
```

Parameters

The **flxActAppReturnVendorDataSet** function uses the following parameters:

Table 8-293 • flxActAppReturnVendorDataSet Parameters

Parameter	Description
ret	A activation object created with flxActAppReturnCreate.
key	This identifies the vendor data: vendor data is added as a key-value pair to the activation request in the vendor dictionary (XML element VendorDictionary).

Table 8-293 • flxActAppReturnVendorDataSet Parameters

Parameter	Description
value	The value of the vendor data to be included in the activation request. This is treated as a null terminated string and no further processing is done to it before placing it in the activation request.

Return Values

There is no return value.

Discussion

This function writes the specified vendor data into the return object. When this return object is subsequently used to create a return request, this vendor data is included in the return request.

The vendor data is added to the VendorDictionary element in the return request XML document. All data dictionaries contain a set of entries. The first element in the entry is the Key element that identifies the type of data in this entry. The second element of the entry, the Value element, contains the data for the entry. This type of entry is referred to here and elsewhere as a *key-value pair*.

This function adds a single key-value pair to the return object. To add multiple key-value pairs, call this function multiple times.

In a test environment, the vendor data contained within a return request can be viewed using the GUI version of the Manual Response Generator, responsegenUI.exe. If you are using the command-line version, `responsegen.*`, you will have to examine the return request before using the Manual Response Generator to generate the appropriate response. For details of the Manual Response Generator, see the *Programming Reference for Trusted Storage–Based Licensing*.

See the FlexNet Operations documentation for information about how vendor data is used by the Operations Server.

When vendor data is added to a request code, it will be modified if the supplied string is too long. The size of the item in the request code is defined by a request override in the ASR (see references to `RequestOverrides` element in the *Programming Reference for Trusted Storage–Based Licensing* for details). The required number of least significant characters are extracted from the supplied string.

For example if 5 digits are defined for the Unique Machine Number in the ASR, then the following 40-digit UMN, 7851233534523409852359823610638450975643 is added to the request code as 75643.

Note • *If this function is used to add vendor data to a return request sent to a license server, the vendor data is ignored by the license server.*

See Also
flxActAppReturnSend
flxActAppReturnReqSet

flxActAppRepairCreate

Creates an application repair object. This object is used when creating a repair request and processing a repair response.

Syntax

```
FlxActBool flxActAppRepairCreate(
    FlxActHandle handle,
    FlxActAppRepair *appRep);
```

Parameters

The **flxActAppRepairCreate** function uses the following parameters:

Table 8-294 • flxActAppRepairCreate Parameters

Parameter	Description
handle	An activation handle created with flxActCommonHandleOpen with the mode parameter set to FLX_ACT_APP.
appRep	The resulting pointer to the application repair object.

Return Values

The following values are returned by **flxActAppRepairCreate:**

Table 8-295 • flxActAppRepairCreate Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function creates an object, which is required when using any of the following repair functions:

- flxActAppRepairSend
- flxActAppRepairReqSet
- flxActAppRepairRespProcess
- flxActAppRepairProdLicSpcSet
- flxActAppRepairProductIdGet
- flxActAppRepairSuiteIdGet
- flxActAppRepairEntitlementIdGet
- flxActAppRepairFRIdGet
- flxActAppRepairFRIdSet

This function should be paired with a call to **flxActAppRepairDelete**, which releases the object and frees all the associated resources.

flxActAppRepairDelete

Delete an application repair object.

Syntax

```
void flxActAppRepairDelete(FlxActAppRepair appRep);
```

Parameters

The **flxActAppRepairDelete** function uses the following parameters:

Table 8-296 • flxActAppRepairDelete Parameters

Parameter	Description
appRep	A repair object created with flxActAppRepairCreate.

Return Values

There is no return value.

Discussion

This function should only be called if a previous call to **flxActAppRepairCreate** succeeded.

It deletes the application repair object, including any repair transaction data that it may contain, and frees all the associated resources. Thus, it must not be called while a repair transaction is in progress.

flxActAppRepairSend

Send a repair request to a remote activation server and automatically process the response.

Syntax

```
FlxActBool flxActAppRepairSend(
    FlxActAppRepair appRep,
    FlxActError *err);
```

Parameters

The **flxActAppRepairSend** function uses the following parameters:

Table 8-297 • flxActAppRepairSend Parameters

Parameter	Description
appRep	A repair object created with flxActAppRepairCreate to which the required parameter values have been added using the **flxActAppRepairProdLicSpcSet** function.
err	A pointer to the resulting error, if any, see flxActCommonHandleGetError for further details.

Return Values

The following values are returned by **flxActAppRepairSend:**

Table 8-298 • flxActAppRepairSend Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success.

Discussion

Before calling this function you must set the fulfillment record that is to be repaired into the repair object using flxActAppRepairProdLicSpcSet.

Optionally, the identity and location of the activation server can be set into the repair object using the following functions:

- flxActCommonHandleSetCommType—defines the type of comms used for transactions and therefore the type of activation server. **flxCommsMvsnFlex** is used for connections to a FlexNet license server and **flxCommsMvsnFlexSoap** is used for connections to a FlexNet Operations server or License Generator.
- flxActCommonHandleSetRemoteServer—sets the address of the activation server used for transactions.

This function formats a repair request and sends it to an activation server.

The following data is automatically added to the repair request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the repair. See “Information Used to Identify the System Initiating the Request” in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.
- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

The activation server is identified before this function is called, see previous paragraphs, or the following defaults are used:

- When **flxActCommonHandleSetCommType** is used to set **flxCommsMvsnFlex**, the default address of the FlexNet license server is @localhost.
- When **flxActCommonHandleSetCommType** is used to set `flxCommsMvsnSoap`, the default address of the FlexNet Operations server is http://localhost:8888/flexnet/services/ActivationService?wsdl.
- The default setting for the `commType` is **flxCommsMvsnFlex** and then the default address of the FlexNet license server is @localhost.

If the activation server allows the repair, the fulfillment record is repaired and this function returns success. If the activation server does not allow the repair, or any other error occurs then the fulfillment record is not repaired and **flxActAppRepairSend** returns non-success.

flxActAppRepairReqSet

Create a repair request as a signed XML document for use in manual repair transactions.

Note • **flxActAppRepairReqSet** *replaces the deprecated function flxActAppRepairReqCreate.*

Syntax

```
FlxActBool flxActAppRepairReqSet(
    FlxActAppRepair appRep,
    const char **xmlString);
```

Parameters

The **flxActAppRepairReqSet** function uses the following parameters:

Table 8-299 • flxActAppRepairReqSet Parameters

Parameter	Description
appRep	A repair object created with flxActAppRepairCreate to which the required parameter values have been added using the **flxActAppRepairProdLicSpcSet** function.
xmlString	The resulting request as a signed XML document.

Return Values

The following values are returned by **flxActAppRepairReqSet:**

Table 8-300 • flxActAppRepairReqSet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function you must set the fulfillment record that is to be repaired into the repair object using flxActAppRepairProdLicSpcSet.

This function returns a repair request. The repair request is a signed XML document that must be transmitted without any modification. This transaction method is normally used for manual transfers, that is, where the repair request is transmitted to the activation server in a file. If your code is writing the signed XML document to a file, ensure that it is not modified in any way.

The following data is automatically added to the repair request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the repair. See "Information Used to Identify the System Initiating the Request" in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.
- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

flxActAppRepairRespProcess

Process the repair response. Depending on the contents of the repair response it either repairs the fulfillment record or leaves it unrepaired and reports an error.

Syntax

```
FlxActBool flxActAppRepairRespProcess(
    FlxActAppRepair appRep,
    char *xmlString,
    FlxActBool *pbIsConfig);
```

Parameters

The **flxActAppRepairRespProcess** function uses the following parameters:

Table 8-301 • flxActAppRepairRespProcess Parameters

Parameter	Description
appRep	A repair object created with flxActAppRepairCreate.
xmlString	A pointer to the response to be processed. The response can be either: ● A signed XML document ● A response code
pbIsConfig	Boolean output that indicates whether the return response contains a Trusted Configuration: ● FLX_ACT_TRUE = The response contains a Trusted Configuration. Trusted storage has been updated with the new Trusted Configuration. The server determined that this was a pre-requisite to processing the request. The original request should be sent to the server again. ● FLX_ACT_FALSE = The response does not contain a Trusted Configuration. The transaction is complete, no further action is required.

Return Values

The following values are returned by **flxActAppRepairRespProcess:**

Table 8-302 • flxActAppRepairRespProcess Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function processes a repair response. This is sent by an activation server in response to a repair request. See flxActAppRepairReqSet for information on how to create a repair request as an XML document, and flxActShortCodeGetDenyReason for information on how to create a request code from an ASR held either in a file or memory buffer, respectively, that contains a repair.

XML Activation Response

The repair response can contain either of the following:

- A repair response
- A Trusted Configuration

The license server determines whether the repair is to be allowed. If the repair is allowed, then this function repairs the fulfillment record and reports success. If the repair is not allowed or some other error occurs, then the fulfillment record is not repaired and this function reports non-success.

If the response contained a Trusted Configuration (pbIsConfig = TRUE), it is used to do one of the following:

- Update the configuration of an existing segment of trusted storage. This is the more probable scenario for a Trusted Configuration in a repair response.
- Create a segment of trusted storage.

Response Code

This function takes the response code and validates it against the outstanding short code transaction for the short code alias. If the response code is valid and the repair is allowed, then this function repairs the fulfillment record and reports success. If the repair is not allowed or some other error occurs, then the fulfillment record is not repaired and this function reports non-success.

The response code contains in an encrypted format a value, SCI_REPAIR_SCOPE, that defines which fulfillment record(s) are to be repaired. The following table lists the values that can be set by the activation server:

Table 8-303 • SCI_REPAIR_SCOPE Values

Value	Description
0	Repair the fulfillment record specified in the request.
1	Repair all fulfillment records in the section of trusted storage specified in the request.

Table 8-303 • SCI_REPAIR_SCOPE Values

Value	Description
2	Repair all fulfillment records.

flxActAppRepairProdLicSpcSet

Sets the fulfillment record to be repaired into the return object.

Syntax

```
void flxActAppRepairProdLicSpcSet(
    FlxActAppRepair appRep,
    FlxActProdLicSpc product);
```

Parameters

The **flxActAppRepairProdLicSpcSet** function uses the following parameters:

Table 8-304 • flxActAppRepairProdLicSpcSet Parameters

Parameter	Description
appRep	A repair object created with flxActAppRepairCreate.
product	A product license specification object that contains the fulfillment record that is to be repaired.

Return Values

There is no return value.

Discussion

Before calling this function select the fulfillment record that is to be repaired. Use the **flxActCommonLicSpc*** and **flxActCommonProdLicSpc*** functions to read the fulfillment records from trusted storage and determine which record is to be repaired.

This function sets the required request parameters into the repair object. It extracts the required parameters from the fulfillment record in the product license specification.

flxActAppRepairProductIdGet

Get the product ID from the repair object.

Syntax

```
const char *flxActAppRepairProductIdGet(FlxActAppRepair appRep);
```

Parameters

The **flxActAppRepairProductIdGet** function uses the following parameters:

Table 8-305 • flxActAppRepairProductIdGet Parameters

Parameter	Description
appRep	A repair object created with flxActAppRepairCreate.

Return Values

The product ID, or NULL if not set.

Discussion

This function returns the product ID from the fulfillment record in the repair object, or NULL if it has not been set.

See Also
flxActAppRepairProdLicSpcSet

flxActAppRepairSuiteIdGet

Get the suite ID from the repair object.

Syntax

```
const char *flxActAppRepairSuiteIdGet(FlxActAppRepair appRep);
```

Parameters

The **flxActAppRepairSuiteIdGet** function uses the following parameters:

Table 8-306 • flxActAppRepairSuiteIdGet Parameters

Parameter	Description
appRep	A repair object created with flxActAppRepairCreate.

Return Values

The suite ID, or NULL if not set.

Discussion

This function returns the suite ID from the fulfillment record that was previously set in the repair object, or NULL if it has not been set.

See Also
flxActAppRepairProdLicSpcSet

flxActAppRepairEntitlementIdGet

Get the entitlement ID from the repair object.

Syntax

```
const char *flxActAppRepairEntitlementIdGet(FlxActAppRepair appRep);
```

Parameters

The **flxActAppRepairEntitlementIdGet** function uses the following parameters:

Table 8-307 • flxActAppRepairEntitlementIdGet Parameters

Parameter	Description
appRep	A repair object created with flxActAppRepairCreate.

Return Values

The entitlement ID, or NULL if not set.

Discussion

This function returns the entitlement ID from the fulfillment record that was previously set in the repair object, or NULL if it has not been set.

See Also
flxActAppRepairProdLicSpcSet

flxActAppRepairFRIdSet

Set the fulfillment ID into the repair object. This function is **only** required when the fulfillment record was not activated using a short code transaction but is to be repaired using a short code transaction.

Syntax

```
void flxActAppRepairFRIdSet(
    FlxActAppRepair appRep,
    const char *pszID);
```

Parameters

The **flxActAppRepairFRIdSet** function uses the following parameters:

Table 8-308 • flxActAppRepairFRIdSet Parameters

Parameter	Description
appRep	A repair object created with flxActAppRepairCreate.
pszID	The fulfillment ID of the records that is to be repaired.

Return Values

There is no return value.

Discussion

Read "Programmatic Activation Followed by Short Code Return or Repair" in the *Programming Reference for Trusted Storage–Based Licensing* for an explanation of the sequence of transactions that require the use of this function.

This function sets the specified fulfillment ID into the repair object. The repair object can then be used to repair the fulfillment record using a short code transaction.

See Also
flxActShortCodeGetDenyReason

flxActAppRepairFRIdGet

Get the fulfillment ID from the repair object.

Syntax

```
const char *flxActAppRepairFRIdGet(FlxActAppRepair appRep);
```

Parameters

The **flxActAppRepairFRIdGet** function uses the following parameters:

Table 8-309 • flxActAppRepairFRIdGet Parameters

Parameter	Description
appRep	A repair object created with flxActAppRepairCreate.

Return Values

The fulfillment ID, or NULL if not set.

Discussion

This function returns the fulfillment ID from the fulfillment record that was previously set in the repair object, or NULL if it has not been set.

See Also
flxActAppRepairProdLicSpcSet

flxActAppRepairVendorDataGet

Get the vendor data for the specified key from the repair object.

Syntax

```
const char *flxActAppRepairVendorDataGet(
    FlxActAppRepair repair,
    const char *key);
```

Parameters

The **flxActAppRepairVendorDataGet** function uses the following parameters:

Table 8-310 • flxActAppRepairVendorDataGet Parameters

Parameter	Description
repair	A repair object created with flxActAppRepairCreate.
key	The key that identifies the vendor data required.

Return Values

The value of the vendor data for the specified key or NULL if not set.

Discussion

This function returns the vendor data that was previously set in the repair object for the specified key, or NULL if it has not been set.

This function does a linear search of all the vendor data key-value pairs in the repair object.

See Also
flxActAppRepairVendorDataSet

flxActAppRepairVendorDataSet

Write the specified vendor data into the repair object.

Syntax

```
void flxActAppRepairVendorDataSet(
    FlxActAppRepair repair,
    const char *key,
    const char *value);
```

Parameters

The **flxActAppRepairVendorDataSet** function uses the following parameters:

Table 8-311 • flxActAppRepairVendorDataSet Parameters

Parameter	Description
repair	A repair object created with flxActAppRepairCreate.
key	This identifies the vendor data: vendor data is added as a key-value pair to the repair request in the vendor dictionary (XML element VendorDictionary).
value	The value of the vendor data to be included in the repair request. This is treated as a null terminated string and no further processing is done to it before placing it in the activation request.

Return Values

There is no return value.

Discussion

This function writes the specified vendor data into the repair object. When this repair object is subsequently used to create a repair request, this vendor data is included in the repair request.

The vendor data is added to the VendorDictionary element in the repair request XML document. All data dictionaries contain a set of entries. The first element in the entry is the Key element that identifies the type of data in this entry. The second element of the entry, the Value element, contains the data for the entry. This type of entry is referred to here and elsewhere as a *key-value pair*.

This function adds a single key-value pair to the repair object. To add multiple key-value pairs, call this function multiple times.

In a test environment, the vendor data contained within a repair request can be viewed using the GUI version of the Manual Response Generator, responsegenUI.exe. If you are using the command-line version, `responsegen.*`, you will have to examine the repair request before using the Manual Response Generator to generate the appropriate response. For details of the Manual Response Generator, see the *Programming Reference for Trusted Storage–Based Licensing*.

See the FlexNet Operations documentation for information about how vendor data is used by the Operations Server.

When vendor data is added to a request code, it will be modified if the supplied string is too long. The size of the item in the request code is defined by an request override in the ASR (see references to RequestOverrides element in the *Programming Reference for Trusted Storage–Based Licensing*). The required number of least significant characters are extracted from the supplied string.

For example if 5 digits are defined for the Unique Machine Number in the ASR then the following 40 digit UMN, 7851233534523409852359823610638450975643 is added to the request code as 75643.

Note • *Repair from a license server—if this function is used to add vendor data to a repair request sent to a license server, the vendor data is ignored by the license server.*

See Also
flxActAppRepairSend
flxActAppRepairReqSet

Deprecated License Server Activation Functions

In general, the license server activation functions are used by a server activation utility to perform any of the following:

- Activate license rights on a license server
- Activate license rights from the license server to a FlexEnabled client
- Transfer licenses to another license server

The functions described in this section specifically support single-action transactions, which can be used to perform client-side activations from the license server. However, from FlexNet Publisher 11.14.0 it is best practice to use composite transactions for all interactions with servers. Composite transactions perform multiple actions in one transaction.

For information on how to create and deploy a server activation utility that supports single-action transactions, see the chapter "Building a Server Activation Utility for Single-Action Transactions" in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*.

This chapter outlines how to use the Activation API to initiate the following activation transactions (which implement the various license lifecycle stages):

- Activation (activations from one license server to another is also referred to as a *transfer*)
- Return
- Repair

Additional Information

Refer to the following for additional information:

- For a description of functions used for local activation, see *Common Library Functions*.
- For a list of deprecated application activation functions, see Deprecated Activation Functions.
- For an overview of license server activation transactions, see the chapter "Transactions to a License Server" in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*.
- For a description of functions used by both application and server activation utilities that handle composite transactions, see Composite Transaction API Functions. For instructions on how to build these activation utilities, see the chapter "Building an Activation Utility for Composite Transactions" in FlexNet Publisher's *Programming Reference for Trusted Storage–Based Licensing*.

Function Summary

The following table provides a summary of the functions described in this section and defined in `FlxActSvr.h`.

Table 8-312 • License Server Activation API Functions

Function	Description
flxActSvrActivationCreate	Create a server activation object.
flxActSvrActivationDelete	Delete a server activation object.
flxActSvrActivationSend	Create and send an activation request to an activation server and process the response.
flxActSvrActivationReqSet	Create an activation request. (Replaces the deprecated function flxActSvrActivationReqCreate.)
flxActSvrActivationRespProcess	Process an activation response received from an activation server.
flxActSvrActivationCountGet	Read the license rights count previously set into the server activation object.
flxActSvrActivationCountSet	Set the number of licenses of the specified type to be set into the server activation object.
flxActSvrActivationEntitlementIdGet	Read the Entitlement ID previously set into the server activation object.
flxActSvrActivationEntitlementIdSet	Set the Entitlement ID to be included in a server activation request.
flxActSvrActivationExpDateGet	Read the license rights expiry date previously set into the server activation object.
flxActSvrActivationExpDateSet	Set the license rights expiry date to be included in a server activation request.
flxActSvrActivationReasonGet	Read the activation reason previously set into the server activation object.
flxActSvrActivationReasonSet	Set the activation reason to be included in a server activation request.
flxActSvrActivationUsernameGet	Read the user name previously set into the server activation object.
flxActSvrActivationUsernameSet	Set the user name to be included in a server activation object.

Table 8-312 • License Server Activation API Functions

Function	Description
flxActSvrActivationHostnameGet	Read the host name previously set into the server activation object.
flxActSvrActivationHostnameSet	Set the host name to be included in a server activation object.
flxActSvrActivationIncludeEnterpriseData	Specify whether user name and/or host name previously set into the activation object are to be included in the activation request.
flxActSvrActivationProductIdGet	Gets the ProductID from an activation object.
flxActSvrActivationProductIdSet	Sets the ProductID for an activation object that will be used in an activation request.
flxActSvrActivationVendorDataGet	Read the vendor data previously set into the activation object.
flxActSvrActivationVendorDataSet	Set the vendor data to be included in a activation request.
flxActSvrReturnCreate	Create a server return object.
flxActSvrReturnDelete	Delete a server return object.
flxActSvrReturnSend	Create and send a return request to an activation server and process the response.
flxActSvrReturnReqSet	Create a return request. (Replaces the deprecated function flxActSvrReturnReqCreate.)
flxActSvrReturnRespProcess	Process a return response from an activation server.
flxActSvrReturnCancel	Enable a fulfillment record that has been disabled when a return was requested.
flxActSvrReturnProdLicSpcSet	Set the fulfillment record to be returned.
flxActSvrReturnReasonGet	Read the return reason previously set into the return object.
flxActSvrReturnReasonSet	Set the return reason to be included in a server return request.
flxActSvrReturnNumberValidDedSpcGet	Get the number of valid activation and transfer deduction records associated with the fulfillment record that is to be returned
flxActSvrReturnForceIncompleteSet	Force the return of a fulfillment record

Table 8-312 • License Server Activation API Functions

Function	Description
flxActSvrReturnForceIncompleteGet	Read whether this return object has been set for a forced return of a fulfillment record
flxActSvrReturnProductIdGet	Read the product ID previously set into the server return object
flxActSvrReturnSuiteIdGet	Read the suite ID previously set into the server return object
flxActSvrReturnEntitlementIdGet	Read the Entitlement ID previously set into the server return object
flxActSvrReturnFRIdGet	Read the fulfillment record ID previously included in the server return object
flxActSvrReturnVendorDataGet	Read the vendor data previously set into the return object
flxActSvrReturnVendorDataSet	Set the vendor data to be included in a return request
flxActSvrRepairCreate	Create a repair object
flxActSvrRepairDelete	Delete a repair object
flxActSvrRepairSend	Create and send a repair request to an activation server and process the response
flxActSvrRepairReqSet	Create a repair request. (Replaces the deprecated function flxActSvrRepairReqCreate.)
flxActSvrRepairRespProcess	Process a repair response from an activation server
flxActSvrRepairProdLicSpcSet	Set the fulfillment record to be repaired
flxActSvrRepairProductIdGet	Read the product ID previously set into the repair object
flxActSvrRepairSuiteIdGet	Read the suite ID previously set into the repair object
flxActSvrRepairEntitlementIdGet	Read the entitlement ID previously set into the repair object
flxActSvrRepairFRIdGet	Read the fulfillment record ID previously set into the repair object
flxActSvrRepairVendorDataGet	Read the vendor data previously set into the repair object
flxActSvrRepairVendorDataSet	Set the vendor data to be included in a repair request

flxActSvrActivationCreate

Create a server activation object. This object is used when creating an activation request and processing an activation response.

Syntax

```
FlxActBool flxActSvrActivationCreate(
    FlxActHandle handle,
    FlxActSvrActivation *serverAct);
```

Parameters

The **flxActSvrActivationCreate** function uses the following parameters:

Table 8-313 • flxActSvrActivationCreate Parameters

Parameter	Description
handle	An activation handle created with flxActCommonHandleOpen with the mode parameter set to FLX_ACT_SVR.
serverAct	The resulting pointer to the server activation object.

Return Values

The following values are returned by **flxActSvrActivationCreate:**

Table 8-314 • flxActSvrActivationCreate Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function creates an object, which is required when using any of the following functions:

- flxActSvrActivationSend
- flxActSvrActivationReqSet
- flxActSvrActivationRespProcess
- flxActSvrActivationCountGet
- flxActSvrActivationCountSet
- flxActSvrActivationEntitlementIdGet
- flxActSvrActivationEntitlementIdSet
- flxActSvrActivationExpDateGet

- flxActSvrActivationExpDateSet
- flxActSvrActivationIncludeEnterpriseData
- flxActSvrActivationUsernameGet
- flxActSvrActivationUsernameSet
- flxActSvrActivationHostnameGet
- flxActSvrActivationHostnameSet

This function should be paired with a call to **flxActSvrActivationDelete**, which releases the object and frees all the associated resources.

flxActSvrActivationDelete

Delete a server activation object.

Syntax

```
void flxActSvrActivationDelete(FlxActSvrActivation serverAct);
```

Parameters

The **flxActSvrActivationDelete** function uses the following parameters:

Table 8-315 • flxActSvrActivationDelete Parameters

Parameter	Description
serverAct	An activation object created with flxActSvrActivationCreate.

Return Values

There is no return value.

Discussion

This function should only be called if a previous call to **flxActSvrActivationCreate** succeeded.

It deletes the server activation object, including any activation transaction data that it may contain, and frees all the associated resources. Thus it must not be called while an activation transaction is in progress.

flxActSvrActivationSend

Send an activation request to a remote activation server and process the resulting response.

Syntax

```
FlxActBool flxActSvrActivationSend(
    FlxActSvrActivation serverAct,
    FlxActError *err);
```

Parameters

The **flxActSvrActivationSend** function uses the following parameters:

Table 8-316 • flxActSvrActivationSend Parameters

Parameter	Description
serverAct	An activation object.
err	A pointer to the resulting error, if any, see flxActCommonHandleGetError for further details.

Return Values

The following values are returned by **flxActSvrActivationSend:**

Table 8-317 • flxActSvrActivationSend Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success.

Discussion

Before calling this function you must set the mandatory and optional request parameters into the server activation object. The following are request parameters in an activation request from a license server:

- **Entitlement ID**—Use flxActAppActivationEntitlementIdSet to set the entitlement ID. This identifies the entitlement for which the license rights are being requested.
- **License rights expiration**—Use flxActAppActivationExpDateSet to enter a specific expiration date.
- **Activatable Count**—Use flxActSvrActivationCountSet for `ACTIVATABLE` license group to request the number of activations of the requested license rights.
- **Activatable Overdraft Count**—Use flxActSvrActivationCountSet for `ACTIVATABLE_OD` license group to request the number of activations of the requested license rights.
- **Concurrent Count**—Use flxActSvrActivationCountSet for `CONCURRENT` license group to request the number of activations of the requested license rights.
- **Concurrent Overdraft Count**—Use flxActSvrActivationCountSet for `CONCURRENT_OD` license group to request the number of activations of the requested license rights.
- **Hybrid Count**—Use flxActSvrActivationCountSet for `HYBRID` license group to request the number of activations of the requested license rights.
- **Hybrid Overdraft Count**—Use flxActSvrActivationCountSet for `HYBRID_OD` license group to request the number of activations of the requested license rights.
- **Repair count**—Use flxActSvrActivationCountSet for

- **Activation reason**—Optionally use flxActSvrActivationReasonSet to specify the reason for the activation. When this is not set, the default value of zero is used.

Optionally, the identity and location of the activation server can be set into the activation object using the following functions:

- flxActCommonHandleSetCommType—defines the type of comms used for transactions and therefore the type of activation server. **flxCommsMvsnFlex** is used for connections to a FlexNet license server and **flxCommsMvsnFlexSoap** is used for connections to a FlexNet Operations server or a License Generator.
- flxActCommonHandleSetRemoteServer—sets the address of the activation server used for transactions.

This function formats an activation request and sends it to an activation server. It is used for programmatic activation.

The following data is automatically added to the activation request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the activation. See "Information Used to Identify the System Initiating the Request" in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.
- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

The activation server is identified before this function is called, see previous paragraphs, or the following defaults are used:

- When **flxActCommonHandleSetCommType** is used to set **flxCommsMvsnFlex**, the default address of the FlexNet license server is @localhost.
- When **flxActCommonHandleSetCommType** is used to set **flxCommsMvsnSoap**, the default address of the FlexNet Operations server is http://localhost:8888/flexnet/services/ActivationService?wsdl.
- The default setting for the `commType` is **flxCommsMvsnFlex**, and then the default address of the FlexNet license server is @localhost.

When this process is successful a fulfillment record for the requested license rights is placed in trusted storage. If there is an existing fulfillment record with the same fulfillment ID, then it is deleted.

Non-success is reported in the return value.

flxActSvrActivationReqSet

Create an activation request as a signed XML document for use in manual activation transactions.

Note • *flxActSvrActivationReqSet replaces the deprecated function flxActSvrActivationReqCreate.*

Syntax

```
FlxActBool flxActSvrActivationReqSet(
    FlxActSvrActivation serverAct,
    const char **xmlString);
```

Parameters

The **flxActSvrActivationReqSet** function uses the following parameters:

Table 8-318 • flxActSvrActivationReqSet Parameters

Parameter	Description
serverAct	A server activation object created with flxActSvrActivationCreate to which the required parameters values have been added using the **flxActSvrActivation*Set** functions.
xmlString	The resulting request as a signed XML document.

Return Values

The following values are returned by **flxActSvrActivationReqSet:**

Table 8-319 • flxActSvrActivationReqSet Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function you must set the mandatory and optional request parameters into the server activation object. The following are request parameters in an activation request from a license server:

- **Entitlement ID**—Use flxActAppActivationEntitlementIdSet to set the entitlement ID. This identifies the entitlement for which the license rights are being requested.
- **License rights expiration**—Use flxActAppActivationExpDateSet to enter a specific expiration date.
- **Activatable Count**—Use flxActSvrActivationCountSet for `ACTIVATABLE` license group to request the number of activations of the requested license rights.
- **Activatable Overdraft Count**—Use flxActSvrActivationCountSet for `ACTIVATABLE_OD` license group to request the number of activations of the requested license rights.
- **Concurrent Count**—Use flxActSvrActivationCountSet for `CONCURRENT` license group to request the number of activations of the requested license rights.
- **Concurrent Overdraft Count**—Use flxActSvrActivationCountSet for `CONCURRENT_OD` license group to request the number of activations of the requested license rights.
- **Hybrid Count**—Use flxActSvrActivationCountSet for `HYBRID` license group to request the number of activations of the requested license rights.
- **Hybrid Overdraft Count**—Use flxActSvrActivationCountSet for `HYBRID_OD` license group to request the number of activations of the requested license rights.
- **Activation reason**—Optionally use flxActSvrActivationReasonSet to specify the reason for the activation. When this is not set, the default value of zero is used.

This function returns an activation request. The activation request is a signed XML document that must be transmitted without any modification. This function is used for manual activation, that is, where the activation request is transmitted to the activation server in a file. If your code is writing the signed XML document to a file, ensure that it is not modified in any way.

The following data is automatically added to the activation request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the activation. See "Information Used to Identify the System Initiating the Request" in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.
- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

flxActSvrActivationRespProcess

Process the activation response. Depending on the contents of the activation response it creates a segment of trusted storage or adds a fulfillment record to trusted storage.

Syntax

```
FlxActBool flxActSvrActivationRespProcess(
    FlxActSvrActivation serverAct,
    char *xmlString
    FlxActBool *pbIsConfig);
```

Parameters

The **flxActSvrActivationRespProcess** function uses the following parameters:

Table 8-320 • flxActSvrActivationRespProcess Parameters

Parameter	Description
serverAct	An activation object created with flxActSvrActivationReqSet.
xmlString	A pointer to a signed XML document containing the response to be processed.
pbIsConfig	Boolean output that indicates whether the return response contains a Trusted Configuration: ● FLX_ACT_TRUE = The response contains a Trusted Configuration. Trusted storage has been updated with the new Trusted Configuration. The server determined that this was a pre-requisite to processing the request. The original request should be sent to the server again. ● FLX_ACT_FALSE = The response does not contain a Trusted Configuration. The transaction is complete, no further action is required.

Return Values

The following values are returned by **flxActSvrActivationRespProcess:**

Table 8-321 • flxActSvrActivationRespProcess Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function is used in manual activation. It processes an activation response, which is sent by an activation server in response to an activation request. See flxActSvrActivationReqSet for information on how to create a server activation request.

This function takes the signed XML response, authenticates it, and if it passes authentication, uses the information it contains to update trusted storage. The activation response can contain either of the following:

- A fulfillment record—Written to trusted storage
- A Trusted Configuration—Used to create a segment of trusted storage

A fulfillment record can only be written to an existing segment of trusted storage. Thus a manual activation can consist of two activation transactions: the first to create the segment of trusted storage and the second to write the required fulfillment record to trusted storage. See the *Programming Reference for Trusted Storage–Based Licensing* for a description of the activation process.

If the response contains a fulfillment record with the same fulfillment ID as an existing record in trusted storage, then the existing fulfillment record is deleted. The net effect is that the existing fulfillment record is overwritten.

flxActSvrActivationCountGet

Get the specified count from the server activation object.

Syntax

```
uint32_t flxActSvrActivationCountGet(
    FlxActSvrActivation *serverAct,
    FlxActCountType licGroup);
```

Parameters

The **flxActSvrActivationCountGet** function uses the following parameters:

Table 8-322 • flxActSvrActivationCountGet Parameters

Parameter	Description
serverAct	A server activation object created with flxActSvrActivationCreate.

Table 8-322 • flxActSvrActivationCountGet Parameters

Parameter	Description
licGroup	The type of count required. For information about these license types, see the *Programming Reference for Trusted Storage–Based Licensing*. The following values (FlxActCountType) can be used for this parameter: • `CONCURRENT`—Concurrent licenses • `CONCURRENT_OD`—Concurrent overdraft licenses • `ACTIVATABLE`—Activatable licenses • `ACTIVATABLE_OD`—Activatable overdraft licenses • `HYBRID`—Hybrid licenses • `HYBRID_OD`—Hybrid overdraft licenses • `REPAIR`—Number of times the fulfillment records can be repaired.

Return Values

The count as set in the server activation object.

Discussion

This function returns the specified count.

See Also
flxActSvrActivationCountSet

flxActSvrActivationCountSet

Write the specified count into the server activation object.

Syntax

```
void flxActSvrActivationCountSet(
    FlxActSvrActivation serverAct,
    FlxActCountType licGroup,
    uint32_t count);
```

Parameters

The **flxActSvrActivationCountSet** function uses the following parameters:

Table 8-323 • flxActSvrActivationCountSet Parameters

Parameter	Description
serverAct	A server activation object created with flxActSvrActivationCreate.

Table 8-323 • flxActSvrActivationCountSet Parameters

Parameter	Description
licGroup	The type of count required. For information about these license types, see the *Programming Reference for Trusted Storage–Based Licensing*. The following values (FlxActCountType) can be used for this parameter: • `CONCURRENT`—Concurrent licenses • `CONCURRENT_OD`—Concurrent overdraft licenses • `ACTIVATABLE`—Activatable licenses • `ACTIVATABLE_OD`—Activatable overdraft licenses • `HYBRID`—Hybrid licenses • `HYBRID_OD`—Hybrid overdraft licenses • `REPAIR`—The requested number of times the fulfillment record can be repaired.
count	The value of the count required.

Return Values

The following values are returned by **flxActSvrActivationCountSet:**

Table 8-324 • flxActSvrActivationCountSet Values

Return Value	Description
void	No return value

Discussion

This function writes the required number of a specified license type or the number of times the fulfillment record can be repaired into the server activation object. When this activation object is subsequently used to create an activation request, this count is included in the activation request. An activation request can contain multiple license group counts and in this case this function would be called once for each license group required.

See Also
flxActSvrActivationSend
flxActSvrActivationReqSet

flxActSvrActivationEntitlementIdGet

Get the entitlement ID from the server activation object.

Syntax

```
const char *flxActSvrActivationEntitlementIdGet(FlxActSvrActivation serverAct);
```

Parameters

The **flxActSvrActivationEntitlementIdGet** function uses the following parameters:

Table 8-325 • flxActSvrActivationEntitlementIdGet Parameters

Parameter	Description
serverAct	An activation object created with flxActSvrActivationCreate.

Return Values

The entitlement ID, or NULL if not set.

Discussion

This function returns the entitlement ID that was previously set in the activation object, or NULL if it has not been set.

See Also
flxActSvrActivationEntitlementIdSet

flxActSvrActivationEntitlementIdSet

Write the specified entitlement ID into the server activation object.

Syntax

```
void flxActSvrActivationEntitlementIdSet(
    FlxActSvrActivation serverAct,
    const char *entitlementId);
```

Parameters

The **flxActSvrActivationEntitlementIdSet** function uses the following parameters:

Table 8-326 • flxActSvrActivationEntitlementIdSet Parameters

Parameter	Description
serverAct	An activation object created with flxActSvrActivationCreate.
entitlementId	The value of the entitlement ID to be included in the activation request.

Return Values

There is no return value.

Discussion

This function writes the specified entitlement ID into the activation object. When this activation object is subsequently used to create an activation request, this entitlement ID is included in the activation request.

See Also
flxActSvrActivationSend
flxActSvrActivationReqSet

flxActSvrActivationExpDateGet

Get the license rights expiration date from the server activation object.

Syntax

```
const char *flxActSvrActivationExpDateGet(FlxActSvrActivation serverAct);
```

Parameters

The **flxActSvrActivationExpDateGet** function uses the following parameters:

Table 8-327 • flxActSvrActivationExpDateGet Parameters

Parameter	Description
serverAct	An activation object created with flxActSvrActivationCreate.

Return Values

The license rights expiration date, or NULL if not set. The format is dd-mmm-yyyy.

Discussion

This function returns the license rights expiration date that was previously set in the activation object, or NULL if it has not been set.

See Also
flxActSvrActivationExpDateSet

flxActSvrActivationExpDateSet

Write the specified license rights expiration date into the server activation object.

Syntax

```
void flxActSvrActivationExpDateSet(
    FlxActSvrActivation *serverAct,
    const char *expDate);
```

Parameters

The **flxActSvrActivationExpDateSet** function uses the following parameters:

Table 8-328 • flxActSvrActivationExpDateSet Parameters

Parameter	Description
serverAct	An activation object created with flxActSvrActivationCreate.
expDate	The value of the expiration date to be included in the activation request. The format is dd-mmm-yyyy.

Return Values

There is no return value.

Discussion

This function writes the specified license rights expiration date into the activation object. When this activation object is subsequently used to create an activation request, this expiration date is included in the activation request.

See Also
flxActSvrActivationSend
flxActSvrActivationReqSet

flxActSvrActivationReasonGet

Get the reason for the activation from the specified activation object.

Syntax

```
const char *flxActSvrActivationReasonGet(FlxActSvrActivation appAct);
```

Parameters

The **flxActSvrActivationReasonGet** function uses the following parameters:

Table 8-329 • flxActSvrActivationReasonGet Parameters

Parameter	Description
appAct	An activation object created with flxActSvrActivationCreate.

Return Values

The value of the activation reason or NULL if not set.

Discussion

This function returns the activation reason that was previously set in the activation object or NULL if it has not been set.

See Also
flxActSvrActivationReasonSet

flxActSvrActivationReasonSet

Write the specified reason for the activation into the activation object.

Syntax

```
FlxActBool flxActSvrActivationReasonSet(
    FlxActSvrActivation appAct,
    const char *pszReason);
```

Parameters

The **flxActSvrActivationReasonSet** function uses the following parameters:

Table 8-330 • flxActSvrActivationReasonSet Parameters

Parameter	Description
appAct	An activation object created with flxActSvrActivationCreate.
pszReason	A pointer to the variable storing the numeric string that will be included in the activation request. This is a publisher defined coded value that indicates the reason that the activation is requested.

Return Values

The following values are returned by **flxActSvrActivationReasonSet**:

Table 8-331 • flxActSvrActivationReasonSet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function writes the specified numeric code for the reason for the activation request into the activation object. When this activation object is subsequently used to create an activation request, this reason code is included in the activation request.

See Also
flxActSvrActivationReasonGet

flxActSvrActivationUsernameGet

Get the user name from the activation object.

Syntax

```
const char *flxActSvrActivationUsernameGet(FlxActSvrActivation serverAct);
```

Parameters

The **flxActSvrActivationUsernameGet** function uses the following parameters:

Table 8-332 • flxActSvrActivationUsernameGet Parameters

Parameter	Description
serverAct	An activation object created with flxActSvrActivationCreate.

Return Values

The user name, or NULL if not set.

Discussion

This function returns the user name that was previously set in the activation object, or NULL if it has not been set.

See Also
flxActSvrActivationUsernameSet

flxActSvrActivationUsernameSet

Write the specified user name into the activation object.

Syntax

```
void flxActSvrActivationUsernameSet(
    FlxActSvrActivation serverAct,
    const char *username);
```

Parameters

The **flxActSvrActivationUsernameSet** function uses the following parameters:

Table 8-333 • flxActSvrActivationUsernameSet Parameters

Parameter	Description
serverAct	An activation object created with flxActSvrActivationCreate.
username	The user name to be included in the activation request.

Return Values

There is no return value.

Discussion

This function writes the specified user name into the activation object. After using this function you must call flxActSvrActivationIncludeEnterpriseData to specify whether this user name is to be included in the activation request.

See Also
flxActSvrActivationIncludeEnterpriseData
flxActAppActivationSend
flxActAppActivationReqSet

flxActSvrActivationHostnameGet

Get the host name from the activation object.

Syntax

```
const char *flxActSvrActivationHostnameGet(FlxActSvrActivation serverAct);
```

Parameters

The **flxActSvrActivationHostnameGet** function uses the following parameters:

Table 8-334 • flxActSvrActivationHostnameGet Parameters

Parameter	Description
serverAct	An activation object created with flxActSvrActivationCreate.

Return Values

The host name, or NULL if not set.

Discussion

This function returns the host name that was previously set in the activation object, or NULL if it has not been set.

See Also
flxActSvrActivationHostnameSet

flxActSvrActivationHostnameSet

Write the specified host name into the activation object.

Syntax

```
void flxActSvrActivationHostnameSet(
    FlxActSvrActivation serverAct,
```

```
    const char *hostname);
```

Parameters

The **flxActSvrActivationHostnameSet** function uses the following parameters:

Table 8-335 • flxActSvrActivationHostnameSet Parameters

Parameter	Description
serverAct	An activation object created with flxActSvrActivationCreate.
hostname	The host name to be included in the activation request.

Return Values

There is no return value.

Discussion

This function writes the specified host name into the activation object. After using this function you must call flxActSvrActivationIncludeEnterpriseData to specify whether this user name is to be included in the activation request.

See Also
flxActSvrActivationIncludeEnterpriseData
flxActAppActivationSend
flxActAppActivationReqSet

flxActSvrActivationIncludeEnterpriseData

Specify whether details of the user and host requesting this activation are to be included in the activation request.

Syntax

```
FlxActBool flxActSvrActivationIncludeEnterpriseData(
    FlxActSvrActivation serverAct,
    FlxActBool bInclude);
```

Parameters

The **flxActSvrActivationIncludeEnterpriseData** function uses the following parameters:

Table 8-336 • flxActSvrActivationIncludeEnterpriseData Parameters

Parameter	Description
serverAct	An activation object created with flxActSvrActivationCreate.

Table 8-336 • flxActSvrActivationIncludeEnterpriseData Parameters

Parameter	Description
bInclude	Indicates whether the user name and host name are to be included in the activation request. • FLX_ACT_TRUE = Include user name and host name, if they have been specified using flxActSvrActivationUsernameSet and flxActSvrActivationHostnameSet. • FLX_ACT_FALSE = Do not include user name and host name.

Return Values

The following values are returned by **flxActSvrActivationIncludeEnterpriseData:**

Table 8-337 • flxActSvrActivationIncludeEnterpriseData Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function is used to include or exclude the user name and host name from an activation request. Exclusion may be required to protect the security of the user making the request.

Before calling this function you must call either or both flxActSvrActivationHostnameSet and flxActSvrActivationUsernameSet to set the host name and/or user name into the server activation object.

flxActSvrActivationProductIdGet

Gets the ProductID from an activation object.

Syntax

```
const char *flxActSvrActivationProductIdGet(FlxActSvrActivation serverAct);
```

Parameters

The **flxActSvrActivationProductIdGet** function uses the following parameters:

Table 8-338 • flxActSvrActivationProductIdGet Parameters

Parameter	Description
serverAct	An activation object.

Return Values

The value set for the ProductID or NULL if it is not set.

Discussion

This function returns the ProductID that was previously set in the activation object or NULL if it has not been set.

See Also
flxActSvrActivationProductIdSet

flxActSvrActivationProductIdSet

Sets the ProductId into an activation object used for an activation or transfer request.

Syntax

```
void flxActSvrActivationProductIdSet(
    FlxActSvrActivation serverAct,
    const char *productId);
```

Parameters

The **flxActSvrActivationProductIdSet** function uses the following parameters:

Table 8-339 • flxActSvrActivationProductIdSet Parameters

Parameter	Description
serverAct	An activation object created with flxActSvrActivationCreate.
productId	A pointer to the variable storing the value of the product ID that will be included in the activation request.

Return Values

There is no return value.

Discussion

This function writes the specified ProductID into the activation object. When this activation object is subsequently used to create an activation request, this ProductID is included in the activation request.

See Also
flxActSvrActivationProductIdGet

flxActSvrActivationVendorDataGet

Get the vendor data for the specified key from the activation object.

Syntax

```
const char *flxActSvrActivationVendorDataGet(
    FlxActSvrActivation svr
    const char *key);
```

Parameters

The **flxActSvrActivationVendorDataGet** function uses the following parameters:

Table 8-340 • flxActSvrActivationVendorDataGet Parameters

Parameter	Description
svr	An activation object created with flxActSvrActivationCreate.
key	The key that identifies the vendor data required.

Return Values

The value of the vendor data for the specified key or NULL if not set.

Discussion

This function returns the vendor data that was previously set in the activation object for the specified key, or NULL if it has not been set.

This function does a linear search of all the vendor data key-value pairs in the activation object.

See Also
flxActSvrActivationVendorDataSet

flxActSvrActivationVendorDataSet

Write the specified vendor data into the activation object.

Syntax

```
void flxActSvrActivationVendorDataSet(
    FlxActSvrActivation svr,
    const char *key,
    const char *value);
```

Parameters

The **flxActSvrActivationVendorDataSet** function uses the following parameters:

Table 8-341 • flxActSvrActivationVendorDataSet Parameters

Parameter	Description
svr	An activation object created with flxActSvrActivationCreate.
key	This identifies the vendor data: vendor data is added as a key-value pair to the activation request in the vendor dictionary (XML element VendorDictionary).

Table 8-341 • flxActSvrActivationVendorDataSet Parameters

Parameter	Description
value	The value of the vendor data to be included in the activation request. This is treated as a null terminated string and no further processing is done to it before placing it in the activation request.

Return Values

There is no return value.

Discussion

This function writes the specified vendor data into the activation object. When this activation object is subsequently used to create an activation request, this vendor data is included in the activation request.

The vendor data is added to the VendorDictionary element in the activation request XML document. All data dictionaries contain a set of entries. The first element in the entry is the Key element that identifies the type of data in this entry. The second element of the entry, the Value element, contains the data for the entry. This type of entry is referred to here and elsewhere as a *key-value pair*.

This function adds a single key-value pair to the activation object. To add multiple key-value pairs, call this function multiple times.

In a test environment, the vendor data contained within an activation request can be viewed using the GUI version of the Manual Response Generator, responsegenUI.exe. If you are using the command-line version, `responsegen.*`, you will have to examine the activation request before using the Manual Response Generator to generate the appropriate response. For details about the Manual Response Generator, see the *Programming Reference for Trusted Storage–Based Licensing*.

See the FlexNet Operations documentation for information about how vendor data is used by the Operations Server.

Note *• Activation from a license server—if this function is used to add vendor data to an activation request sent to a license server, the vendor data is ignored by the license server.*

See Also
flxActSvrActivationSend
flxActSvrActivationReqSet

flxActSvrReturnCreate

Create a server return object. This object is used when creating a return request and processing a return response.

Syntax

```
FlxActBool flxActSvrReturnCreate(
    FlxActHandle handle,
    FlxActSvrReturn *serverRet);
```

Parameters

The **flxActSvrReturnCreate** function uses the following parameters:

Table 8-342 • flxActSvrReturnCreate Parameters

Parameter	Description
handle	An activation handle created with flxActCommonHandleOpen with the mode parameter set to FLX_ACT_SVR.
serverRet	The resulting pointer to the server return object

Return Values

The following values are returned by **flxActSvrReturnCreate:**

Table 8-343 • flxActSvrReturnCreate Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function creates an object, which is required when using any of the following return functions:

- flxActSvrReturnSend
- flxActSvrReturnReqSet
- flxActSvrReturnRespProcess
- flxActSvrReturnProdLicSpcSet
- flxActSvrReturnNumberValidDedSpcGet
- flxActSvrReturnForceIncompleteSet
- flxActSvrReturnForceIncompleteGet
- flxActSvrReturnProductIdGet
- flxActSvrReturnSuiteIdGet
- flxActSvrReturnEntitlementIdGet
- flxActSvrReturnFRIdGet

This function should be paired with a call to **flxActSvrReturnDelete**, which releases the object and frees all the associated resources.

Note • *If the returned fulfillment record contains concurrent or hybrid licenses, the counts in memory require updating after a successful return of a fulfillment record. Use `Lmreread.exe` as described in the Programming Reference for License File–Based Licensing. This updates the license group counts held in memory. See the Programming Reference for Trusted Storage–Based Licensing for information about these license groups.*

flxActSvrReturnDelete

Delete a server return object.

Syntax

```
void flxActSvrReturnDelete(FlxActSvrReturn serverRet);
```

Parameters

The **flxActSvrReturnDelete** function uses the following parameters:

Table 8-344 • flxActSvrReturnDelete Parameters

Parameter	Description
serverRet	A return object created with flxActSvrReturnCreate.

Return Values

There is no return value.

Discussion

This function should only be called if a previous call to flxActSvrReturnCreate succeeded.

It deletes the return object, including any return transaction data that it may contain, and frees all the associated resources. Thus, it must not be called while a return transaction is in progress.

flxActSvrReturnSend

Send a return request to a remote activation server and automatically process the response.

Syntax

```
FlxActBool flxActSvrReturnSend(
    FlxActSvrReturn serverRet,
    FlxActError *err);
```

Parameters

The **flxActSvrReturnSend** function uses the following parameters:

Table 8-345 • flxActSvrReturnSend Parameters

Parameter	Description
serverRet	A server return object created with flxActSvrReturnCreate to which the required parameter values have been added using the flxActSvrReturnProdLicSpcSet function.
err	A pointer to the resulting error, if any, see flxActCommonHandleGetError for further details of general errors. See LM_TS_RETURN_INCOMPLETE for a description of this error and how to use other functions to override this error.

Return Values

The following values are returned by **flxActSvrReturnSend:**

Table 8-346 • flxActSvrReturnSend Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success.

Discussion

Before calling this function you must set the fulfillment record that is to be returned into the return object using flxActSvrReturnProdLicSpcSet.

Optionally use flxActSvrReturnReasonSet to specify the reason for the return. When this is not set, the default value of zero is used.

Optionally, the identity and location of the activation server can be set into the return object using the following functions:

- flxActCommonHandleSetCommType—defines the type of comms used for transactions and therefore the type of activation server. **flxCommsMvsnFlex** is used for connections to a FlexNet license server and **flxCommsMvsnFlexSoap** is used for connections to a FlexNet Operations server or License Generator.
- flxActCommonHandleSetRemoteServer—sets the address of the activation server used for transactions.

This function formats a return request and sends it to an activation server.

The following data is automatically added to the return request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the return. See “Information Used to Identify the System Initiating the Request” in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.

- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

The activation server is identified before this function is called, see previous paragraphs, or the following defaults are used:

- When **flxActCommonHandleSetCommType** is used to set **flxCommsMvsnFlex**, the default address of the FlexNet license server is @localhost.
- When **flxActCommonHandleSetCommType** is used to set **flxCommsMvsnSoap**, the default address of the FlexNet Operations server is http://localhost:8888/flexnet/services/ActivationService?wsdl.
- The default setting for the `commType` is **flxCommsMvsnFlex**, and then the default address of the FlexNet license server is @localhost.

A remote license server only accepts this as a return of the fulfillment record when it originally supplied it. When the remote license server accepts the return of the fulfillment record it:

- Adds it to its trusted storage
- Replies success

The Operations server processes the return request according to its internal logic and sends the appropriate return response. On receipt of the return response reporting success the fulfillment record is deleted from the local server mode trusted storage. On receipt of the return response reporting non-success, the fulfillment record is not deleted from the local server mode trusted storage and this function returns non-success.

A processing failure may result in no response or an invalid response being received from the activation server. In this case the fulfillment record will remain disabled and the license rights it contains unavailable unless it is enabled using flxActSvrReturnCancel.

LM_TS_RETURN_INCOMPLETE

This error is reported under the following circumstances:

1. A fulfillment record is placed in trusted storage on a license server.
2. Activation requests result in license rights from the fulfillment record being available on other machines.

   The activation requests can be from end-user machines or from other license servers. The first case uses activatable or hybrid counts within the fulfillment record. The second case can use any of activatable, hybrid or concurrent counts within the fulfillment record.

   The license server generates a fulfillment record, which contains the license rights requested, and sends it to the requesting machine. It also generates a *deduction record* that is stored in its trusted storage in association with the original fulfillment record. The deduction record records details of the license rights, which originated in the fulfillment record, and that have been issued to another machine.
3. A call to **flxActSvrReturnSend** is made when there are valid associated deductions records.

   When there are valid deduction records, there are license rights that are part of this fulfillment still in use on other machines. Therefore, a complete return of the license rights held within the fulfillment record is impossible.

   Additionally the fulfillment records on other machines that have been supplied from the original fulfillment record will be orphaned if the original fulfillment record is returned. If an attempt is made to return them to the license server after the original fulfillment record is returned, this return request will fail.
4. The call to **flxActSvrReturnSend** fails with error `LM_TS_RETURN_INCOMPLETE`.

Additional functions are provided to allow for the return of a fulfillment record in the following cases:

- When the FlexEnabled client machine has been destroyed and its licenses cannot be returned.
- When one of the license servers in the chain of servers has been destroyed and so licenses cannot be returned to the originating license server.

The following functions are provided:

- **flxActSvrReturnNumberValidDedSpcGet**—Get the number of valid deduction records associated with the fulfillment record in the return object.
- **flxActSvrReturnForceIncompleteSet**—Force a return request to proceed even when there are valid deduction records associated with the fulfillment record.

By examining the contents of deduction records, the following functions allow you to ascertain details of the license rights that have been issued and to determine where they are being used:

- flxActCommonProdLicSpcNumberValidDedSpcGet
- flxActCommonDedSpcDestinationFulfillmentIdGet
- flxActCommonDedSpcDestinationSystemNameGet
- flxActCommonDedSpcExpDateGet
- flxActCommonDedSpcTypeGet
- flxActCommonValidateDateString

For an example of the use of these functions, see the *Programming Reference for Trusted Storage–Based Licensing*.

flxActSvrReturnReqSet

Create a return request as a signed XML document for use in manual return transactions.

Note • *flxActSvrReturnReqSet replace the deprecated function flxActSvrReturnReqCreate.*

Syntax

```
FlxActBool flxActSvrReturnReqSet(
    FlxActSvrReturn serverRet,
    const char **xmlString);
```

Parameters

The **flxActSvrReturnReqSet** function uses the following parameters:

Table 8-347 • flxActSvrReturnReqSet Parameters

Parameter	Description
serverRet	A server return object created with flxActSvrReturnCreate to which the required parameter values have been added using the **flxActSvrReturnProdLicSpcSet** function.

Table 8-347 • flxActSvrReturnReqSet Parameters

Parameter	Description
xmlString	The resulting request as a signed XML document.

Return Values

The following values are returned by **flxActSvrReturnReqSet:**

Table 8-348 • flxActSvrReturnReqSet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out details of general errors. For information about `LM_TS_RETURN_INCOMPLETE`, see **flxActSvrReturnSend** LM_TS_RETURN_INCOMPLETE.

Discussion

Before calling this function you must set the fulfillment record that is to be returned into the return object using flxActSvrReturnProdLicSpcSet.

Optionally use flxActSvrReturnReasonSet to specify the reason for the return. When this is not set, the default value of zero is used.

This function returns a return request. The following data is automatically added to the return request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the return. See “Information Used to Identify the System Initiating the Request” in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.
- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

The return request is a signed XML document that must be transmitted without any modification. This transaction method is normally used for manual transfers, that is, where the return request is transmitted to the activation server in a file. If your code is writing the signed XML document to a file, ensure that it is not modified in any way.

flxActSvrReturnRespProcess

Process the return response. Depending on the contents of the return response it either deletes the returned fulfillment record from the server mode trusted storage or leaves it in trusted storage and reports an error.

Syntax

```
FlxActBool flxActSvrReturnRespProcess(
    FlxActSvrReturn serverRet,
```

```
    char *xmlString,
    FlxActBool *pbIsConfig);
```

Parameters

The **flxActSvrReturnRespProcess** function uses the following parameters:

Table 8-349 • flxActSvrReturnRespProcess Parameters

Parameter	Description
serverRet	A return object created with flxActSvrReturnCreate.
xmlString	A pointer to a signed XML document containing the response to be processed.
pbIsConfig	Boolean output that indicates whether the activation response contains a Trusted Configuration: ● FLX_ACT_TRUE = activation response contains a Trusted Configuration. ● FLX_ACT_FALSE = activation response contains a fulfillment record.

Return Values

The following values are returned by **flxActSvrReturnRespProcess:**

Table 8-350 • flxActSvrReturnRespProcess Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function processes a return response. This is sent by an activation server in response to a return request. See flxActAppReturnReqSet for information on how to create a return request.

The return response can contain either of the following:

- A response to the return request
- A Trusted Configuration—used to create a segment of trusted storage

The remote license server only accepts this as a return of the fulfillment record when it originally supplied it. When the remote license server accepts the return of the fulfillment record it:

- Adds it to its trusted storage
- Sends a return response that indicates success

On receipt of the return response reporting success the fulfillment record is deleted from the local server mode trusted storage.

When the remote license server sends a failure response, the fulfillment record is re-enabled and the license rights it contains can once again be used. This function returns non-success. The failure response can be because the return of the fulfillment record was denied by the activation server or because there was a failure processing the return request.

A processing failure may result in no response or an invalid response being received from the activation server. In this case the fulfillment record will remain disabled and the license rights it contains unavailable unless it is enabled using flxActSvrReturnCancel.

flxActSvrReturnCancel

Enable a fulfillment record that has been disabled when a return was requested.

Syntax

```
FlxActBool flxActSvrReturnCancel(FlxActAppReturn serverRet);
```

Parameters

The **flxActSvrReturnCancel** function uses the following parameters:

Table 8-351 • flxActSvrReturnCancel Parameters

Parameter	Description
serverRet	A return object created with flxActSvrReturnCreate that has the fulfillment record whose return was requested set into it.

Return Values

The following values are returned by **flxActSvrReturnCancel:**

Table 8-352 • flxActSvrReturnCancel Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function create the return object and set the fulfillment record whose return was requested into the return object using flxActSvrReturnProdLicSpcSet.

This function is used to enable a fulfillment record when a return was requested and no response has been received. When a return request is sent the fulfillment record is disabled. When a response is received the fulfillment record is either deleted (when the return is allowed by the activation server) or enabled (when the return is denied by the activation server or the return processing fails). When no response is received the fulfillment record will remain disabled and the license rights unavailable indefinitely. You have three options at this point:

- **Do nothing** - This leaves the fulfillment record disabled and the license rights unavailable.

- **Delete the fulfillment record** - See "Delete a Single Fulfillment Record from Trusted Storage" in the *Programming Reference for Trusted Storage–Based Licensing* for detailed instructions. This may remove the fulfillment record when the activation server and associated systems record that the customer still has these license rights.
- **Enable the fulfillment record** - This may give the customer extra license rights if the activation server and associated systems have recorded that the license rights were returned.

For example code that demonstrates the use of this function, see the example activation utility, `serveractutil`. The `serveractutil` command line syntax to demonstrate the use of **flxActSvrReturnCancel** is:

```
serveractutil -return <fulfillmentID> -cancel
```

flxActSvrReturnProdLicSpcSet

Sets the fulfillment record to be returned into the return object.

Syntax

```
void flxActSvrReturnProdLicSpcSet(
    FlxActSvrReturn ret,
    FlxActProdLicSpc product);
```

Parameters

The **flxActSvrReturnProdLicSpcSet** function uses the following parameters:

Table 8-353 • flxActSvrReturnProdLicSpcSet Parameters

Parameter	Description
ret	A return object created with flxActSvrReturnCreate.
product	A product license specification object that contains the fulfillment record that is to be returned.

Return Values

There is no return value.

Discussion

Before calling this function select the fulfillment record that is to be returned. Use the **flxActCommonLicSpc*** and **flxActCommonProdLicSpc*** functions to read the fulfillment records from trusted storage and determine which record is to be returned.

This function sets the required request parameters into the return object. It extracts the required parameters from the fulfillment record in the product license specification.

flxActSvrReturnReasonGet

Get the reason for the return from the specified return object.

Syntax

```
const char *flxActSvrReturnReasonGet(FlxActSvrReturn ret);
```

Parameters

The **flxActSvrReturnReasonGet** function uses the following parameters:

Table 8-354 • flxActSvrReturnReasonGet Parameters

Parameter	Description
ret	A return object created with flxActSvrReturnCreate.

Return Values

The value of the return reason, or NULL if not set.

Discussion

This function returns the reason for the return request that was previously set in the return object or NULL if it has not been set.

See Also
flxActSvrReturnReasonSet

flxActSvrReturnReasonSet

Write the specified reason for the return into the return object.

Syntax

```
FlxActBool flxActSvrReturnReasonSet(
    FlxActSvrActivation appAct,
    const char *pszReason);
```

Parameters

The **flxActSvrReturnReasonSet** function uses the following parameters:

Table 8-355 • flxActSvrReturnReasonSet Parameters

Parameter	Description
appAct	An activation object created with flxActSvrReturnCreate.
pszReason	A pointer to the variable storing the numeric string that will be included in the return request. This is a publisher defined coded value that indicates the reason that the return is requested.

Return Values

The following values are returned by **flxActSvrReturnReasonSet**:

Table 8-356 • flxActAppReturnReasonSet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function writes the specified numeric code for the reason for the return request into the return object. When this return object is subsequently used to create a return request, this reason code is included in the return request.

See Also
flxActSvrReturnReasonGet

flxActSvrReturnNumberValidDedSpcGet

Get the number of valid activation and transfer deduction records associated with the fulfillment record that is to be returned.

Syntax

```
uint32_t flxActSvrReturnNumberValidDedSpcGet(FlxActSvrReturn serverRet);
```

Parameters

The **flxActSvrReturnNumberValidDedSpcGet** function uses the following parameters:

Table 8-357 • flxActSvrReturnNumberValidDedSpcGet Parameters

Parameter	Description
serverRet	A return object created with flxActSvrReturnCreate that contains a fulfillment record that is to be returned.

Return Values

The number of valid activation and transfer deduction records associated with the fulfillment record that is to be returned.

Discussion

Before calling this function select the fulfillment record that is to be returned. Use the **flxActCommonLicSpc*** and **flxActCommonProdLicSpc*** functions to read the fulfillment records from trusted storage and determine which record is to be returned. Then use **flxActSvrReturnProductLicSpcSet** to set the selected fulfillment record into the return object.

This function returns the number of valid *activation* and *transfer* deduction records associated with the fulfillment record that is to be returned. Deduction records are *valid* if they reference fulfillment records on remote machines that contain unexpired license rights. An *activation* deduction record is created when an activation request is received from an end-user machine and license rights are supplied by the license server. A *transfer* deduction record is created when an activation request is received from a license server and license rights are supplied by this license server. For a full list of types of deduction records, see FlxActDeductionType.

It can be used to avoid sending a return request that will result in an LM_TS_RETURN_INCOMPLETE.

See Also
flxActSvrReturnSend
flxActSvrReturnReqSet
flxActSvrReturnForceIncompleteSet

flxActSvrReturnForceIncompleteGet

Get the conditions set for the return of a fulfillment record from the return object.

Syntax

```
FlxActBool flxActSvrReturnForceIncompleteGet(FlxActSvrReturn serverRet);
```

Parameters

The **flxActSvrReturnForceIncompleteGet** function uses the following parameters:

Table 8-358 • flxActSvrReturnForceIncompleteGet Parameters

Parameter	Description
serverRet	A return object created with flxActSvrReturnCreate that contains a fulfillment record that is to be returned.

Return Values

Indicates the conditions for the return of the fulfillment record.

Table 8-359 • flxActSvrReturnForceIncompleteGet Return Values

Return Value	Description
FLX_ACT_TRUE	Return fulfillment record if there are associated valid deduction records.
FLX_ACT_FALSE	Do not return fulfillment record if there are associated valid deduction records.

Discussion

This function returns the conditions that have been set in the return object for the return of a fulfillment record.

See Also
flxActSvrReturnForceIncompleteSet

flxActSvrReturnForceIncompleteSet

Force the return of a fulfillment record when license rights will be left orphaned on other machines by this action.

Syntax

```
FlxActBool flxActSvrReturnForceIncompleteSet(
    FlxActSvrReturn serverRet,
    FlxActBool bForceIncomplete);
```

Parameters

The **flxActSvrReturnForceIncompleteSet** function uses the following parameters:

Table 8-360 • flxActSvrReturnForceIncompleteSet Parameters

Parameter	Description
serverRet	A return object created with flxActSvrReturnCreate that contains a fulfillment record that is to be returned.
bForceIncomplete	Indicates the conditions for the return of the fulfillment record. ● FLX_ACT_TRUE = return fulfillment record even if there are associated valid deduction records. ● FLX_ACT_FALSE = do not return fulfillment record if there are associated valid deduction records.

Return Values

The following values are returned by **flxActSvrReturnForceIncompleteSet:**

Table 8-361 • flxActSvrReturnForceIncompleteSet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function select the fulfillment record that is to be returned. Use the **flxActCommonLicSpc*** and **flxActCommonProdLicSpc*** functions to read the fulfillment records from trusted storage and determine which record is to be returned. Then use **flxActSvrReturnProductLicSpcSet** to set the selected fulfillment record into the return object.

Use this function when a forced return of a fulfillment record is required. Any fulfillment records that have been created from this fulfillment record as the result of an activation request and are still valid will be orphaned. For more details, see LM_TS_RETURN_INCOMPLETE in flxActSvrReturnSend.

This function can be used for the return of a fulfillment record in the following cases:

- When the FlexEnabled client machine has been destroyed and its licenses cannot be returned.

- When one of the license servers in the chain of servers has been destroyed and so licenses cannot be returned to the originating license server.

Use this function with care: it can result in extra license rights being available to the enterprise when a return is followed by the licenses being re-issued.

See Also
flxActSvrReturnForceIncompleteGet
flxActSvrReturnNumberValidDedSpcGet
flxActSvrReturnSend
flxActSvrReturnReqSet

flxActSvrReturnProductIdGet

Get the product ID from the server return object.

Syntax

```
const char *flxActSvrReturnProductIdGet(FlxActSvrReturn serverRet);
```

Parameters

The **flxActSvrReturnProductIdGet** function uses the following parameters:

Table 8-362 • flxActSvrReturnProductIdGet Parameters

Parameter	Description
serverRet	A server return object created with flxActSvrReturnCreate.

Return Values

The product ID, or NULL if not set.

Discussion

This function returns the product ID that was previously set in the return object, or NULL if it has not been set.

See Also
flxActSvrReturnProdLicSpcSet

flxActSvrReturnSuiteIdGet

Get the suite ID from the server return object.

Syntax

```
const char *flxActSvrReturnSuiteIdGet(FlxActSvrReturn serverRet);
```

Parameters

The **flxActSvrReturnSuiteIdGet** function uses the following parameters:

Table 8-363 • flxActSvrReturnSuiteIdGet Parameters

Parameter	Description
serverRet	A return object created with flxActSvrReturnCreate.

Return Values

The suite ID, or NULL if not set.

Discussion

This function returns the suite ID that was previously set in the return object, or NULL if it has not been set.

See Also
flxActSvrReturnProdLicSpcSet

flxActSvrReturnEntitlementIdGet

Get the entitlement ID from the server return object.

Syntax

```
const char *flxActSvrReturnEntitlementIdGet(FlxActSvrReturn serverRet);
```

Parameters

The **flxActSvrReturnEntitlementIdGet** function uses the following parameters:

Table 8-364 • flxActSvrReturnEntitlementIdGet Parameters

Parameter	Description
serverRet	A return object created with flxActSvrReturnCreate.

Return Values

The entitlement ID, or NULL if not set.

Discussion

This function returns the entitlement ID that was previously set in the server return object, or NULL if it has not been set.

See Also
flxActSvrReturnProdLicSpcSet

flxActSvrReturnFRIdGet

Get the fulfillment ID from the server return object.

Syntax

```
const char *flxActSvrReturnFRIdGet(FlxActSvrActivation serverRet);
```

Parameters

The **flxActSvrReturnFRIdGet** function uses the following parameters:

Table 8-365 • flxActSvrReturnFRIdGet Parameters

Parameter	Description
serverRet	A return object created with flxActSvrReturnCreate.

Return Values

The fulfillment ID, or NULL if not set.

Discussion

This function returns the fulfillment ID that was previously set in the server return object, or NULL if it has not been set.

See Also
flxActSvrReturnProdLicSpcSet

flxActSvrReturnVendorDataGet

Get the vendor data for the specified key from the return object.

Syntax

```
const char *flxActSvrReturnVendorDataGet(
    FlxActSvrReturn ret,
    const char *key);
```

Parameters

The **flxActSvrReturnVendorDataGet** function uses the following parameters:

Table 8-366 • flxActSvrReturnVendorDataGet Parameters

Parameter	Description
ret	A return object created with flxActSvrReturnCreate.

Table 8-366 • flxActSvrReturnVendorDataGet Parameters

Parameter	Description
key	The key that identifies the vendor data required.

Return Values

The value of the vendor data for the specified key or NULL if not set.

Discussion

This function returns the vendor data that was previously set in the return object for the specified key, or NULL if it has not been set.

This function does a linear search of all the vendor data key-value pairs in the return object.

See Also
flxActSvrReturnVendorDataSet

flxActSvrReturnVendorDataSet

Write the specified vendor data into the return object.

Syntax

```
void flxActSvrReturnVendorDataSet(
    FlxActSvrReturn ret,
    const char *key,
    const char *value);
```

Parameters

The **flxActSvrReturnVendorDataSet** function uses the following parameters:

Table 8-367 • flxActSvrReturnVendorDataSet Parameters

Parameter	Description
ret	A return object created with flxActSvrReturnCreate.
key	This identifies the vendor data: vendor data is added as a key-value pair to the return request in the vendor dictionary (XML element VendorDictionary).
value	The value of the vendor data to be included in the return request. This is treated as a null terminated string and no further processing is done to it before placing it in the activation request.

Return Values

There is no return value.

Discussion

This function writes the specified vendor data into the return object. When this return object is subsequently used to create a return request, this vendor data is included in the return request.

The vendor data is added to the VendorDictionary element in the return request XML document. All data dictionaries contain a set of entries. The first element in the entry is the Key element that identifies the type of data in this entry. The second element of the entry, the Value element, contains the data for the entry. This type of entry is referred to here and elsewhere as a *key-value pair*.

This function adds a single key-value pair to the return object. To add multiple key-value pairs, call this function multiple times.

In a test environment, the vendor data contained within a return request can be viewed using the GUI version of the Manual Response Generator, responsegenUI.exe. If you are using the command-line version, `responsegen.*`, you will have to examine the return request before using the Manual Response Generator to generate the appropriate response. For details about the Manual Response Generator, see the *Programming Reference for Trusted Storage–Based Licensing*.

See the FlexNet Operations documentation for information about how vendor data is used by the Operations Server.

Note • *If this function is used to add vendor data to a return request sent to a license server, the vendor data is ignored by the license server.*

See Also
flxActSvrReturnSend
flxActSvrReturnReqSet

flxActSvrRepairCreate

Creates a repair object. This object is used when creating a repair request and processing a repair response.

Syntax

```
FlxActBool flxActSvrRepairCreate(
    FlxActHandle handle,
    FlxActSvrRepair *serverRep);
```

Parameters

The **flxActSvrRepairCreate** function uses the following parameters:

Table 8-368 • flxActSvrRepairCreate Parameters

Parameter	Description
handle	An activation handle created with flxActCommonHandleOpen with the mode parameter set to FLX_ACT_APP.
serverRep	The resulting pointer to the repair object

Return Values

The following values are returned by **flxActSvrRepairCreate**:

Table 8-369 • flxActSvrRepairCreate Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function creates an object, which is required when using any of the following repair functions:

- flxActSvrRepairSend
- flxActSvrRepairReqSet
- flxActSvrRepairRespProcess
- flxActSvrRepairProdLicSpcSet
- flxActSvrRepairProductIdGet
- flxActSvrRepairSuiteIdGet
- flxActSvrRepairEntitlementIdGet
- flxActSvrRepairFRIdGet

This function should be paired with a call to **flxActSvrRepairDelete**, which releases the object and frees all the associated resources.

flxActSvrRepairDelete

Delete a repair object.

Syntax

```
void flxActSvrRepairDelete(FlxActSvrRepair serverRep);
```

Parameters

The **flxActSvrRepairDelete** function uses the following parameters:

Table 8-370 • flxActSvrRepairDelete Parameters

Parameter	Description
serverRep	A repair object created with flxActSvrReturnCreate.

Return Values

There is no return value.

Discussion

This function should only be called if a previous call to **flxActSvrRepairCreate** succeeded.

It deletes the repair object, including any repair transaction data that it may contain, and frees all the associated resources. Thus, it must not be called while a repair transaction is in progress.

flxActSvrRepairSend

Send a repair request to a remote activation server and automatically process the response.

Syntax

```
FlxActBool flxActSvrRepairSend(
    FlxActSvrRepair serverRep,
    FlxActError *err);
```

Parameters

The **flxActSvrRepairSend** function uses the following parameters:

Table 8-371 • flxActSvrRepairSend Parameters

Parameter	Description
serverRep	A repair object created with flxActSvrReturnCreate to which the required parameter values have been added using the **flxActSvrRepairProdLicSpcSet** function.
err	A pointer to the resulting error, if any, see flxActCommonHandleGetError for further details.

Return Values

The following values are returned by **flxActSvrRepairSend**:

Table 8-372 • flxActSvrRepairSend Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success.

Discussion

Before calling this function you must set the fulfillment record that is to be repaired into the repair object using flxActSvrRepairProdLicSpcSet.

Optionally, the identity and location of the activation server can be set into the activation object using the following functions:

- **flxActCommonHandleSetCommType**—Defines the type of comms used for transactions and therefore the type of activation server. **flxCommsMvsnFlex** is used for connections to a FlexNet license server and **flxCommsMvsnFlexSoap** is used for connections to a FlexNet Operations server or License Generator.
- **flxActCommonHandleSetRemoteServer**—Sets the address of the activation server used for transactions.

This function formats a repair request and sends it to an activation server.

The following data is automatically added to the repair request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the repair. See "Information Used to Identify the System Initiating the Request" in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.
- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

The activation server is identified before this function is called, see previous paragraphs, or the following defaults are used:

- When **flxActCommonHandleSetCommType** is used to set **flxCommsMvsnFlex**, the default address of the FlexNet license server is @localhost.
- When **flxActCommonHandleSetCommType** is used to set **flxCommsMvsnSoap**, the default address of the FlexNet Operations server is http://localhost:8888/flexnet/services/ActivationService?wsdl.
- The default setting for the `commType` is **flxCommsMvsnFlex,** and then the default address of the FlexNet license server is @localhost.

If the activation server allows the repair, the fulfillment record is repaired and this function returns success. If the activation server does not allow the repair, or any other error occurs then the fulfillment record is not repaired and **flxActSvrRepairSend** returns non-success.

flxActSvrRepairReqSet

Create a repair request as a signed XML document for use in manual repair transactions.

Note • *flxActSvrRepairReqSet replaces the deprecated function flxActSvrRepairReqCreate.*

Syntax

```
FlxActBool flxActSvrRepairReqSet(
    FlxActSvrRepair serverRep,
    const char **xmlString);
```

Parameters

The **flxActSvrRepairReqSet** function uses the following parameters:

Table 8-373 • flxActSvrRepairReqSet Parameters

Parameter	Description
serverRep	A repair object created with flxActSvrReturnCreate to which the required parameter values have been added using the **flxActSvrRepairProdLicSpcSet** function.
xmlString	The resulting request as a signed XML document.

Return Values

The following values are returned by **flxActSvrRepairReqSet**:

Table 8-374 • flxActSvrRepairReqSet Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function you must set the fulfillment record that is to be repaired into the repair object using flxActSvrRepairProdLicSpcSet.

This function returns a repair request.

The following data is automatically added to the repair request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the repair. See “Information Used to Identify the System Initiating the Request” in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.
- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

The repair request is a signed XML document that must be transmitted without any modification. This transaction method is normally used for manual transfers, that is, where the repair request is transmitted to the activation server in a file. If your code is writing the signed XML document to a file, ensure that it is not modified in any way.

flxActSvrRepairRespProcess

Process the repair response. Depending on the contents of the repair response it either repairs the fulfillment record or leaves it unrepaired and reports an error.

Syntax

```
FlxActBool flxActSvrRepairRespProcess(
    FlxActSvrRepair serverRep,
    char *xmlString,
    FlxActBool *pbIsConfig);
```

Parameters

The **flxActSvrRepairRespProcess** function uses the following parameters:

Table 8-375 • flxActSvrRepairRespProcess Parameters

Parameter	Description
serverRep	A repair object created with flxActSvrReturnCreate.
xmlString	A pointer to a signed XML document containing the response to be processed.
pbIsConfig	Boolean output that indicates whether the activation response contains a Trusted Configuration: ● FLX_ACT_TRUE = activation response contains a Trusted Configuration. ● FLX_ACT_FALSE = activation response contains a fulfillment record.

Return Values

The following values are returned by **flxActSvrRepairRespProcess**:

Table 8-376 • flxActSvrRepairRespProcess Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

This function processes a repair response. This is sent by an activation server in response to a repair request. See flxActSvrRepairReqSet for information on how to create a repair request.

The repair response can contain either of the following:

- A repair response
- A Trusted Configuration

The activation server determines whether the repair is to be allowed. If the repair is allowed, then this function repairs the fulfillment record and reports success. If the repair is not allowed or some other error occurs, then the fulfillment record is not repaired and this function reports non-success.

If the response contained a Trusted Configuration (`pbIsConfig = TRUE`), it is used to do one of the following:

- Update the settings for an existing segment of trusted storage. This is the more probable scenario for a Trusted Configuration in a repair response.
- Create a segment of trusted storage.

flxActSvrRepairProdLicSpcSet

Sets the fulfillment record to be repaired into the repair object.

Syntax

```
void flxActSvrRepairProdLicSpcSet(
    FlxActSvrRepair serverRep,
    FlxActProdLicSpc product);
```

Parameters

The **flxActSvrRepairProdLicSpcSet** function uses the following parameters:

Table 8-377 • flxActSvrRepairProdLicSpcSet Parameters

Parameter	Description
serverRep	A repair object created with flxActSvrReturnCreate.
product	A product license specification object that contains the fulfillment record that is to be repaired.

Return Values

There is no return value.

Discussion

Before calling this function select the fulfillment record that is to be repaired. Use the **flxActCommonLicSpc*** and **flxActCommonProdLicSpc*** functions to read the fulfillment records from trusted storage and determine which record is to be repaired.

This function sets the required request parameters into the repair object. It extracts the required parameters from the fulfillment record in the product license specification.

flxActSvrRepairProductIdGet

Get the product ID from the repair object.

Syntax

```
const char *flxActSvrRepairProductIdGet(FlxActSvrRepair serverRep);
```

Parameters

The **flxActSvrRepairProductIdGet** function uses the following parameters:

Table 8-378 • flxActSvrRepairProductIdGet Parameters

Parameter	Description
serverRep	A repair object created with flxActSvrReturnCreate.

Return Values

The product ID, or NULL if not set.

Discussion

This function returns the product ID that was previously set in the repair object, or NULL if it has not been set.

See Also
flxActSvrRepairProdLicSpcSet

flxActSvrRepairSuiteIdGet

Get the suite ID from the repair object.

Syntax

```
const char *flxActSvrRepairSuiteIdGet(FlxActSvrRepair serverRep);
```

Parameters

The **flxActSvrRepairSuiteIdGet** function uses the following parameters:

Table 8-379 • flxActSvrRepairSuiteIdGet Parameters

Parameter	Description
serverRep	A repair object created with flxActSvrReturnCreate.

Return Values

The suite ID, or NULL if not set.

Discussion

This function returns the suite ID that was previously set in the repair object, or NULL if it has not been set.

See Also
flxActSvrRepairProdLicSpcSet

flxActSvrRepairEntitlementIdGet

Get the entitlement ID from the repair object.

Syntax

```
const char *flxActSvrRepairEntitlementIdGet(FlxActSvrRepair serverRep);
```

Parameters

The **flxActSvrRepairEntitlementIdGet** function uses the following parameters:

Table 8-380 • flxActSvrRepairEntitlementIdGet Parameters

Parameter	Description
serverRep	A repair object created with flxActSvrReturnCreate.

Return Values

The entitlement ID, or NULL if not set.

Discussion

This function returns the entitlement ID that was previously set in the repair object, or NULL if it has not been set.

See Also
flxActSvrRepairProdLicSpcSet

flxActSvrRepairFRIdGet

Get the fulfillment ID from the repair object.

Syntax

```
const char *flxActSvrRepairFRIdGet(FlxActSvrRepair serverRep);
```

Parameters

The **flxActSvrRepairFRIdGet** function uses the following parameters:

Table 8-381 • flxActSvrRepairFRIdGet Parameters

Parameter	Description
serverRep	A repair object created with flxActSvrReturnCreate.

Return Values

The fulfillment ID, or NULL if not set.

Discussion

This function returns the fulfillment ID that was previously set in the repair object, or NULL if it has not been set.

See Also
flxActSvrRepairProdLicSpcSet

flxActSvrRepairVendorDataGet

Get the vendor data for the specified key from the repair object.

Syntax

```
const char *flxActSvrRepairVendorDataGet(
    FlxActSvrRepair repair,
    const char *key);
```

Parameters

The **flxActSvrRepairVendorDataGet** function uses the following parameters:

Table 8-382 • flxActSvrRepairVendorDataGet Parameters

Parameter	Description
repair	A repair object created with flxActSvrReturnCreate.
key	The key that identifies the vendor data required.

Return Values

The value of the vendor data for the specified key or NULL if not set.

Discussion

This function returns the vendor data that was previously set in the repair object for the specified key, or NULL if it has not been set.

This function does a linear search of all the vendor data key-value pairs in the activation object.

See Also
flxActSvrRepairVendorDataSet

flxActSvrRepairVendorDataSet

Write the specified vendor data into the repair object.

Syntax

```
void flxActSvrRepairVendorDataSet(
    FlxActSvrRepair repair,
    const char *key,
    const char *value);
```

Parameters

The **flxActSvrRepairVendorDataSet** function uses the following parameters:

Table 8-383 • flxActSvrRepairVendorDataSet Parameters

Parameter	Description
repair	A repair object created with flxActSvrReturnCreate.
key	This identifies the vendor data: vendor data is added as a key-value pair to the repair request in the vendor dictionary (XML element VendorDictionary).
value	The value of the vendor data to be included in the repair request. This is treated as a null terminated string and no further processing is done to it before placing it in the repair request.

Return Values

There is no return value.

Discussion

This function writes the specified vendor data into the repair object. When this repair object is subsequently used to create a repair request, this vendor data is included in the repair request.

The vendor data is added to the VendorDictionary element in the repair request XML document. All data dictionaries contain a set of entries. The first element in the entry is the Key element that identifies the type of data in this entry. The second element of the entry, the Value element, contains the data for the entry. This type of entry is referred to here and elsewhere as a *key-value pair*.

This function adds a single key-value pair to the repair object. To add multiple key-value pairs, call this function multiple times.

The vendor data contained within a repair request can be viewed using the GUI version of the Manual Response Generator, `responsegenUI.exe`. If you are using the command-line version, `responsegen.*`, you will have to examine the repair request before using the Manual Response Generator to generate the appropriate response. For details of the Manual Response Generator, see the *Programming Reference for Trusted Storage–Based Licensing*. The FlexNet Operations Server documentation also contains information about how vendor data is used by the Operations Server.

In a test environment, the vendor data contained within a repair request can be viewed using the GUI version of the Manual Response Generator, responsegenUI.exe. If you are using the command-line version, `responsegen.*`, you will have to examine the repair request before using the Manual Response Generator to generate the appropriate response. For details of the Manual Response Generator, see the *Programming Reference for Trusted Storage–Based Licensing*.

See the FlexNet Operations documentation for information about how vendor data is used by the Operations Server.

Note • *If this function is used to add vendor data to a repair request sent to a license server, the vendor data is ignored by the license server.*

See Also
flxActSvrRepairSend
flxActSvrRepairReqSet

Deprecated Activation Functions

The following application and license-server activation functions have been deprecated and replaced by the functions identified in these descriptions.

flxActAppActivationReqCreate

Create an activation request as a signed XML document for use in manual activation transactions.

Note • *flxActAppActivationReqCreate is a deprecated function and may be removed in a future release. flxActAppActivationReqSet replaces flxActAppActivationReqCreate.*

Syntax

```
FlxActBool flxActAppActivationReqCreate(
    FlxActAppActivation appAct,
    char **xmlString);
```

Parameters

The **flxActAppActivationReqCreate** function uses the following parameters:

Table 8-384 • flxActAppActivationReqCreate Parameters

Parameter	Description
appAct	An activation object created with flxActAppActivationCreate to which the required parameter values have been added using the **flxActAppActivation*Set** functions.
xmlString	The resulting request as a signed XML document. Free the memory used for this string after using.

Return Values

The following values are returned by **flxActAppActivationReqCreate:**

Table 8-385 • flxActAppActivationReqCreate Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function you must set the required request parameters into the activation object. The following are request parameters in an activation request:

- **Entitlement ID**—Use flxActAppActivationEntitlementIdSet to set the entitlement ID. This identifies the entitlement for which the license rights are being requested.
- **License rights expiration**—Use either flxActAppActivationExpDateSet to enter a specific expiration date or flxActAppActivationDurationSet to enter the relative duration.
- **Activation reason**—Optionally use flxActAppActivationReasonSet to specify the reason for the activation. When this is not set, the default value of zero is used.
- **Vendor data**—Optionally use flxActAppActivationVendorDataSet to add a key-value pair to the PublisherDictionary in the activation request.

This function returns an activation request. The activation request is a signed XML document that must be transmitted without any modification. This method of activation is used for manual activation, i.e., where the activation request is transmitted to the activation server in a file. If your code is writing the signed XML document to a file, ensure that it is not modified in any way.

The following data is automatically added to the activation request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the activation. See “Information Used to Identify the System Initiating the Request” in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.
- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

After calling this function and using the XML document returned, free the memory used for this string.

flxActAppReturnReqCreate

Create a return request as a signed XML document for use in manual return transactions.

Note • *flxActAppReturnReqCreate is a deprecated function and may be removed in a future release. flxActAppReturnReqSet replaces* **flxActAppReturnReqCreate**.

Syntax

```
FlxActBool flxActAppReturnReqCreate(
    FlxActAppReturn appRet,
    char **xmlString);
```

Parameters

The **flxActAppReturnReqCreate** function uses the following parameters:

Table 8-386 • flxActAppReturnReqCreate Parameters

Parameter	Description
appRet	A return object created with flxActAppReturnCreate to which the required parameter values have been added using the **flxActAppReturnProdLicSpcSet** function.
xmlString	The resulting request as a signed XML document. Free the memory used for this string after using.

Return Values

The following values are returned by **flxActAppReturnReqCreate:**

Table 8-387 • flxActAppReturnReqCreate Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function you must set the fulfillment record that is to be returned into the return object using flxActAppReturnProdLicSpcSet.

Optionally use flxActAppReturnReasonSet to specify the reason for the return. When this is not set, the default value of zero is used.

flxActAppRepairReqCreate

This function returns a return request. The return request is a signed XML document that must be transmitted without any modification. This transaction method is normally used for manual transfers, that is, where the return request is transmitted to the activation server in a file. If your code is writing the signed XML document to a file, ensure that it is not modified in any way.

The following data is automatically added to the return request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the return. See "Information Used to Identify the System Initiating the Request" in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.
- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

After calling this function and using the XML document returned, free the memory used for this string.

Create a repair request as a signed XML document for use in manual repair transactions.

Note • *flxActAppRepairReqCreate is a deprecated function and may be removed in a future release. flxActAppRepairReqSet replaces* **flxActAppRepairReqCreate**.

Syntax

```
FlxActBool flxActAppRepairReqCreate(
    FlxActAppRepair appRep,
    char **xmlString);
```

Parameters

The **flxActAppRepairReqCreate** function uses the following parameters:

Table 8-388 • flxActAppRepairReqCreate Parameters

Parameter	Description
appRep	A repair object created with flxActAppRepairCreate to which the required parameter values have been added using the **flxActAppRepairProdLicSpcSet** function.
xmlString	The resulting request as a signed XML document. Free the memory used for this string after using.

Return Values

The following values are returned by **flxActAppRepairReqCreate:**

Table 8-389 • flxActAppRepairReqCreate Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function you must set the fulfillment record that is to be repaired into the repair object using flxActAppRepairProdLicSpcSet.

This function returns a repair request. The repair request is a signed XML document that must be transmitted without any modification. This transaction method is normally used for manual transfers, that is, where the repair request is transmitted to the activation server in a file. If your code is writing the signed XML document to a file, ensure that it is not modified in any way.

The following data is automatically added to the repair request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the repair. See "Information Used to Identify the System Initiating the Request" in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.
- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

After calling this function and using the XML document returned, free the memory used for this string.

flxActSvrActivationReqCreate

Create an activation request as a signed XML document for use in manual activation transactions.

Note • *flxActSvrActivationReqCreate is a deprecated function and may be removed in a future release. flxActSvrActivationReqSet replaces flxActSvrActivationReqCreate.*

Syntax

```
FlxActBool flxActSvrActivationReqCreate(
    FlxActSvrActivation serverAct,
    char **xmlString);
```

Parameters

The **flxActSvrActivationReqCreate** function uses the following parameters:

Table 8-390 • flxActSvrActivationReqCreate Parameters

Parameter	Description
serverAct	A server activation object created with flxActSvrActivationReqSet to which the required parameters values have been added using the **flxActSvrActivation*Set** functions.
xmlString	The resulting request as a signed XML document.

Return Values

The following values are returned by **flxActSvrActivationReqCreate:**

Table 8-391 • flxActSvrActivationReqCreate Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function you must set the mandatory and optional request parameters into the server activation object. The following are request parameters in an activation request from a license server:

- **Entitlement ID**—Use flxActAppActivationEntitlementIdSet to set the entitlement ID. This identifies the entitlement for which the license rights are being requested.
- **License rights expiration**—Use flxActAppActivationExpDateSet to enter a specific expiration date.
- **Activatable Count**—Use flxActSvrActivationCountSet for `ACTIVATABLE` license group to request the number of activations of the requested license rights.
- **Activatable Overdraft Count**—Use flxActSvrActivationCountSet for `ACTIVATABLE_OD` license group to request the number of activations of the requested license rights.
- **Concurrent Count**—Use flxActSvrActivationCountSet for `CONCURRENT` license group to request the number of activations of the requested license rights.
- **Concurrent Overdraft Count**—Use flxActSvrActivationCountSet for `CONCURRENT_OD` license group to request the number of activations of the requested license rights.
- **Hybrid Count**—Use flxActSvrActivationCountSet for `HYBRID` license group to request the number of activations of the requested license rights.
- **Hybrid Overdraft Count**—Use flxActSvrActivationCountSet for `HYBRID_OD` license group to request the number of activations of the requested license rights.
- **Activation reason**—Optionally use flxActSvrActivationReasonSet to specify the reason for the activation. When this is not set, the default value of zero is used.

This function returns an activation request. The activation request is a signed XML document that must be transmitted without any modification. This function is used for manual activation, that is, where the activation request is transmitted to the activation server in a file. If your code is writing the signed XML document to a file, ensure that it is not modified in any way.

The following data is automatically added to the activation request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the activation. See “Information Used to Identify the System Initiating the Request” in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.

- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

flxActSvrReturnReqCreate

Create a return request as a signed XML document for use in manual return transactions.

Note • *flxActSvrReturnReqCreate is a deprecated function and may be removed in a future release. flxActSvrReturnReqSet replaces* **flxActSvrReturnReqCreate**.

Syntax

```
FlxActBool flxActSvrReturnReqCreate(
    FlxActSvrReturn serverRet,
    char **xmlString);
```

Parameters

The **flxActSvrReturnReqCreate** function uses the following parameters:

Table 8-392 • flxActSvrReturnReqCreate Parameters

Parameter	Description
serverRet	A server return object created with flxActSvrReturnCreate to which the required parameter values have been added using the **flxActSvrReturnProdLicSpcSet** function.
xmlString	The resulting request as a signed XML document.

Return Values

The following values are returned by **flxActSvrReturnReqCreate:**

Table 8-393 • flxActSvrReturnReqCreate Return Values

Return Value	Description
FLX_ACT_TRUE	Success
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out details of general errors. For information about `LM_TS_RETURN_INCOMPLETE`, see **flxActSvrReturnSend** LM_TS_RETURN_INCOMPLETE.

Discussion

Before calling this function you must set the fulfillment record that is to be returned into the return object using flxActSvrReturnProdLicSpcSet.

Optionally use flxActSvrReturnReasonSet to specify the reason for the return. When this is not set, the default value of zero is used.

This function returns a return request. The following data is automatically added to the return request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the return. See "Information Used to Identify the System Initiating the Request" in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.
- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type for details.

The return request is a signed XML document that must be transmitted without any modification. This transaction method is normally used for manual transfers, that is, where the return request is transmitted to the activation server in a file. If your code is writing the signed XML document to a file, ensure that it is not modified in any way.

flxActSvrRepairReqCreate

Create a repair request as a signed XML document for use in manual repair transactions.

Note • *flxActSvrRepairReqCreate is a deprecated function and may be removed in a future release. flxActSvrRepairReqSet replaces flxActSvrRepairReqCreate.*

Syntax

```
FlxActBool flxActSvrRepairReqCreate(
    FlxActSvrRepair serverRep,
    char **xmlString);
```

Parameters

The **flxActSvrRepairReqCreate** function uses the following parameters:

Table 8-394 • flxActSvrRepairReqCreate Parameters

Parameter	Description
serverRep	A repair object created with flxActSvrReturnCreate to which the required parameter values have been added using the **flxActSvrRepairProdLicSpcSet** function.
xmlString	The resulting request as a signed XML document.

Return Values

The following values are returned by **flxActSvrRepairReqCreate**:

Table 8-395 • flxActSvrRepairReqCreate Return Values

Return Value	Description
FLX_ACT_TRUE	Success

Table 8-395 • flxActSvrRepairReqCreate Return Values

Return Value	Description
FLX_ACT_FALSE	Non-success. Call flxActCommonHandleGetError to find out the specific error.

Discussion

Before calling this function you must set the fulfillment record that is to be repaired into the repair object using flxActSvrRepairProdLicSpcSet.

This function returns a repair request.

The following data is automatically added to the repair request:

- **Unique Machine Number, UMN**—All available UMNs are added. The UMNs are used by the activation server to identify the machine requesting the repair. See "Information Used to Identify the System Initiating the Request" in the *Programming Reference for Trusted Storage-Based Licensing* for a description of the types of UMNs available.
- **Trusted Host Information**—Information about currently configured trusted sections. The information includes the trusted ID, revision number and type, and Machine ID for each section.
- **Platform Type**—The platform on which the activation utility is running as identified by the Licensing toolkit used to create the utility. See Platform Type on page 453 for details.

The repair request is a signed XML document that must be transmitted without any modification. This transaction method is normally used for manual transfers, that is, where the repair request is transmitted to the activation server in a file. If your code is writing the signed XML document to a file, ensure that it is not modified in any way.

Deprecated Short Code Functions

In FlexNet Publisher 11.14.0, a new short code API was introduced that supports both server and application ASRs (see Short Code Transaction APIs on page 328). For new implementations of FlexNet Publisher, it is best practice to use this new API for all short codes. The previous API and its functions continue to be supported for legacy implementations. For information about the previous API and its functions, see the product documentation for the relevant FlexNet Publisher release (11.13.0 or earlier, if appropriate).

The following table lists deprecated short code functions and its new equivalents.

Table 8-396 • Changes to the short code API

Description	New Function (Application and Server)	Deprecated Function
Create a new activation request	flxActShortCodeCreate or flxActShortCodeCreateFromBuffer	flxActAppActivationCreate
	flxActShortCodeGetActivationRequest	flxActAppActivationShortCodeGenerate or flxActAppActivationShortCodeGenerateFromBuffer
	flxActShortCodeDestroy	flxActAppActivationDelete

Table 8-396 • Changes to the short code API

Description	New Function (Application and Server)	Deprecated Function
Create a new return request	flxActShortCodeCreate or flxActShortCodeCreateFromBuffer flxActShortCodeGetReturnRequest flxActShortCodeDestroy	flxActAppReturnCreate flxActAppReturnShortCodeGenerate or flxActAppReturnShortCodeGenerateFromBuffer flxActAppReturnDelete
Create a new repair request	flxActShortCodeCreate or flxActShortCodeCreateFromBuffer flxActShortCodeGetRepairRequest flxActShortCodeDestroy	flxActAppRepairCreate flxActAppRepairShortCodeGenerate or flxActAppRepairShortCodeGenerateFromBuffer flxActAppRepairDelete
Process a response	flxActShortCodeProcessResponse	Short codes use the XML processing functions: flxActAppActivationRespProcess flxActAppReturnRespProcess flxActAppRepairRespProcess
Get error tuple	flxActCommonHandleGetError	flxActCommonHandleGetError
Get extended error information	flxActShortCodeGetDenyReason flxActShortCodeGetWarning flxActShortCodeGetErrorDetail	Not available

Table 8-396 • Changes to the short code API

Description	New Function (Application and Server)	Deprecated Function
Get a pending request	**All request types** flxActShortCodeCreate or flxActShortCodeCreateFromBuffer flxActShortCodeGetPendingRequest flxActShortCodeDestroy	**Activation requests** flxActAppActivationCreate flxActAppActivationShortCodePending or flxActAppActivationShortCodePendingFromBuffer flxActAppActivationDelete **Return requests** flxActAppReturnCreate flxActAppReturnShortCodePending or flxActAppReturnShortCodePendingFromBuffer flxActAppReturnDelete **Repair requests** flxActAppRepairCreate flxActAppRepairShortCodePending or flxActAppRepairShortCodePendingFromBuffer flxActAppRepairDelete **All request types (duplicates the above)** flxActAppGetPendingShortCode or flxActAppGetPendingShortCodeFromBuffer
Get target fulfillment record of pending return or repair request	flxActShortCodeCreate or flxActShortCodeCreateFromBuffer flxActShortCodeGetPendingRequestProdSpc flxActShortCodeDestroy	Not available

Table 8-396 • Changes to the short code API

Description	New Function (Application and Server)	Deprecated Function
Cancel a request	**All request types**	**Activation requests**
	flxActShortCodeCreate or flxActShortCodeCreateFromBuffer	flxActAppActivationCreate
	flxActShortCodeCancelRequest	flxActAppActivationShortCodeCancel or flxActAppActivationShortCodeCancelFromBuffer
	flxActShortCodeDestroy	flxActAppActivationDelete
		Return requests
		flxActAppReturnCreate
		flxActAppReturnShortCodeCancel or flxActAppReturnShortCodeCancelFromBuffer
		flxActAppReturnDelete
		Repair requests
		flxActAppRepairCreate
		flxActAppRepairShortCodeCancel or flxActAppRepairShortCodeCancelFromBuffer
		flxActAppRepairDelete

Common Data

The following sections describe data that is used by multiple functions:

- Unique Machine Number (UMN)
- Platform Type

Unique Machine Number (UMN)

Unique Machine Numbers (UMNs) are retrieved by the activation utility and used by the activation server to identify the machine requesting activation activity. For more information about the UMNs supported by FlexNet Publisher, refer to “Information Used to Identify the System Initiating the Request” in the *Programming Reference for Trusted Storage–Based Licensing*.

Platform Type

The platform type is automatically included in all XML requests. The platform-type name is the name of the FlexNet Publisher Licensing Toolkit download file (for example, Windows 32-bit platform-type name is `i86_n3`). For a complete list of platform types, see “Information Used to Identify the System Initiating the Request” in the *Programming Reference for Trusted Storage–Based Licensing*.

9

Obsolete Flexible API Features

The functions and attributes listed in this section are obsolete but are still provided in the Licensing toolkit for backward compatibility. Their functionality has, in most cases, been replaced. The current feature that replaces the obsolete function or attribute is given in the following tables.

If you are implementing FlexNet licensing for the first time into your application, do not use these functions and attributes. Refer to reference material in the following sections:

- Flexible API
- Advanced Flexible API Functions
- Flexible API Attributes
- Advanced Flexible API Attributes

Obsolete Flexible API Functions

The following functions are obsolete, and exist only for compatibility with earlier Licensing toolkit versions. They should be used with care, and questions are welcomed before their use.

Table 9-1 • Obsolete Flexible API Functions

Function	Description
lc_alarm	Sets a timer. For current functionality see: • lc_heartbeat • LM_A_CHECK_INTERVAL

Table 9-1 • Obsolete Flexible API Functions

Function	Description
lc_baddate	Detects if system date has been set back. For current functionality see: ● LM_A_CHECK_BADDATE ● ls_a_check_baddate in the *Programming Reference for License File–Based Licensing*
lc_chk_conf	Validates a config structure. For current functionality see: ● lc_auth_data
lc_ck_feats	Checks the FEATURESET line for a given vendor. FEATURESET is no longer used.
lc_copy_hostid	Returns a copy of the hostid list. For current functionality see: ● lc_hostid
lc_crypt	Computes the license key for a FEATURE line. For current functionality see: ● lc_cryptstr
lc_convert	Converts a license file from one format to the other. Formats include readable and decimal. Decimal format is deprecated and no longer used.
lc_disalarm	Turns off a timer set with **lc_alarm**. For current functionality see: ● lc_heartbeat ● LM_A_CHECK_INTERVAL
lc_disconn	Drops the connection to the server. This functionality is no longer supported by the Flexible API; use functionality provided by the native operating system.
lc_display	Returns environment information about the current display. This functionality is no longer supported by the Flexible API; use functionality provided by the native operating system.
lc_errtext	Returns the English text string corresponding to the Flexible API errno. For current functionality see: ● lc_perror ● lc_err_info
lc_feat_set	Computes the FEATURESET code. FEATURESET is no longer used.

Table 9-1 • Obsolete Flexible API Functions

Function	Description
lc_get_feats	Gets a license key from the `FEATURESET` line. `FEATURESET` is no longer used.
lc_gethostid	Returns the hostid for the local host. For current functionality see: ● lc_hostid
lc_getid_type	Returns the `HOSTID` of the specified type for the local host. For current functionality see: ● lc_hostid ● Vendor-Defined Hostid Types in the *Programming Reference for License File–Based Licensing*
lc_hostname	Returns environment information about the current hostname. This functionality is no longer supported by the Flexible API; use functionality provided by the native operating system.
lc_isadmin	Verifies the specified user is a license administrator. This functionality is no longer supported by the Flexible API; use functionality provided by the native operating system.
lc_set_errno	Sets the Flexible API errno. For current functionality see: ● lc_perror ● lc_err_info ● lc_get_errno
lc_timer	Exchanges heartbeat messages with the license server. For current functionality see: ● lc_heartbeat
lc_username	Returns environment information about the current user name. This functionality is no longer supported by the Flexible API; use functionality provided by the native operating system.

Obsolete Flexible API Attributes

The following attributes are obsolete, and exist only for compatibility with earlier FlexNet Publisher versions. They should be used with care, and questions are welcomed before their use.

Table 9-2 • Obsolete Flexible API Attributes

Attribute	Description
LM_A_BEHAVIOR_VER	Sets the behavior of the FlexEnabled application to the given version of FlexNet Licensing.
LM_A_CRYPT_CASE_SENSITIVE	Controls case sensitivity in license files using license keys. For current functionality see: ● LM_A_LICENSE_CASE_SENSITIVE
LM_A_LKEY_LONG	Enables 64-bit license keys. License keys are obsolete. For current functionality see: ● SIGN in the *Programming Reference for License File–Based Licensing*
LM_A_LKEY_START_DATE	Enables embedded start dates in license keys. License keys are obsolete. For current functionality see: ● START in the *Programming Reference for License File–Based Licensing*
LM_A_MAX_TIMEDIFF	Tests the difference in clock settings between the FlexEnabled application's machine and the license server machine. Now, automatically performed when needed.
LM_A_NO_MT_HEARBEAT_TIMER (UNIX Only)	Stores heartbeat timer information in a separate memory space where all shared objects have access to it. The attribute was used to avoid heartbeat problems that could occur if a FlexEnabled application used multiple shared objects that were built with the non-multithreaded library `liblmgr_nomt_pic.a` (now deprecated).
LM_A_PATCH_NUMBER	Gets the patch number of the Licensing toolkit client library. For current functionality see: ● lc_get_version
LM_A_REVISION	Gets the revision number of the Licensing toolkit client library. For current functionality see: ● lc_get_version

Table 9-2 • Obsolete Flexible API Attributes

Attribute	Description
LM_A_SETITIMER, LM_A_SIGNAL (UNIX Only)	Controls which function is used in place of **settimer.**
LM_A_SUB_MINOR_REVISION	Gets the sub minor revision number of the Licensing toolkit client library. For current functionality see: ● lc_get_version
LM_A_USE_START_DATE	Enforces the start date to be used in checkouts. For current functionality see: ● START in the *Programming Reference for License File–Based Licensing*
LM_A_VERSION	Gets the version number of the Licensing toolkit client library. For current functionality see: ● lc_get_version

10

Flexible API Error Return Values

The table below shows all the possible errors returned from **lc_*** functions. All the positive errors at the start of the table are returned by `lc_flexinit` during initialization required for using trusted storage. These positive errors (names starting with **flxInit***) are from `FlexInit.h`. All other errors (names of the form **LM_***) are defined in `lmclient.h`.

Error strings corresponding to the errors definded in `lmclient.h` can be found in `lmerrors.h` (short form) and `lm_lerr.h` (long form).

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
24	flxInitPrivilegeError	Insufficient Privilege to complete initialization.
23	flxInitTSPrepTypeMismatch	Attempt to access server trusted storage from client app, or vice-versa.
22	flxInitServiceConfigurationError	FlexNet Licensing Service configuration or dependency issue.
21	flxInitServiceNotEnoughRights	lc_flexinit failed because there was insufficient rights to start the FlexNet Licensing Service. Resolve this by setting the service to start automatically.
20	flxInitServiceNotInstalled	FlexNet Licensing Service is not installed.
13	flxInitPathOverflow	Computed path to required file is too long for Mac OS X operating system.
12	flxInitBundleIDError	Invalid bundle ID on Mac OS X operating system.
11	flxInitFrameworkNotLoadedError	Framework specified by bundle ID was not loaded.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
10	flxInitUrlNoPathError	Error creating path from URL.
9	flxInitUrlError	Error creating URL.
8	flxInitEncodingError	Path string not specified in UTF-8 format.
7	flxInitCannotReload	A call to lc_flexinit is not allowed after a call to lc_flexinit_cleanup.
6	flxInitNotProtected	The executable has not been prepped with the preptool. For more information about the preptool, see *Programming Reference for Trusted Storage–Based Licensing* for instructions.
5	flxInitAllocationError	Unable to allocate resources.
4	flxInitInitializationError	Initialization failed.
3	flxInitUnsupportedPlatform	Unsupported version of the operating system.
2	flxInitUnableToLoad	Unable to load activation library.
1	flxInitUnableToLocate	Unable to find activation library.
0	LM_NOERROR	“There was no error. @ref FLEX_ERROR”. The licensed file used to return success.
-1	LM_NOCONFFILE	“Cannot find license file.” The license file cannot be opened.
-2	LM_BADFILE	“Invalid license file syntax.” Feature name is > **MAX_FEATURE_LEN**, or daemon name is > **MAX_DAEMON_LEN**, or server name is > **MAX_SERVER_NAME**, or a feature specifies no hostid and # of licenses is <= 0.
-3	LM_NOSERVER	"No license server system for this feature." The daemon name specified in the license file FEATURE line does not match the vendor daemon name.
-4	LM_MAXUSERS	“Licensed number of users already reached.” The licensed number of users has been reached.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-5	LM_NOFEATURE	“No such feature exists.” The feature could not be found in the license file.
-6	LM_NOSERVICE	"No port number in license file and "FLEXlm" service does not exist." This happens if a SERVER line does not specify a TCP/IP port number, and the TCP/IP license service does not exist in `/etc/services`.
-7	LM_NOSOCKET	"No socket connection to license server manager." **lc_disconn** was called after the process had been disconnected from the socket.
-8	LM_BADCODE	"Invalid (inconsistent) license key." The code in a license file line does not match the other data in the license file. This is usually the result of not building all the Licensing components with the same encryption seeds. Check `lsvendor.c` and your application code carefully to ensure that they are all built with the same encryption seeds.
-9	LM_NOTTHISHOST	"Invalid host." The hostid specified in the license file does not match the node on which the software is running.
-10	LM_LONGGONE	"Feature has expired." The feature has expired, meaning that today’s date is after the expiration date in the license file.
-11	LM_BADDATE	"Invalid date format in license file." The start or expiration date in the license file is invalid.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-12	LM_BADCOMM	"Invalid returned data from license server system." The port number returned from `lmadmin` or `lmgrd` is invalid. An attempted connection to a vendor daemon did not result in a correct acknowledgment from the daemon. The daemon did not send back a message within the timeout interval. A message from the daemon had an invalid checksum. An **lc_userlist** request did not receive the correct data.
-13	LM_NO_SERVER_IN_FILE	"No SERVER lines in license file." There is no SERVER line in the license file. All non-zero license count features need at least one SERVER line.
-14	LM_BADHOST	"Cannot find SERVER hostname in network database." The **gethostbyname** system call failed for the SERVER name in the license file.
-15	LM_CANTCONNECT	"Cannot connect to license server system." The **connect** system call failed, while attempting to connect to the daemon. The attempt to connect to the vendor daemon on all SERVER nodes was unsuccessful. **lc_status** returns **LM_CANTCONNECT** if the feature had been checked out but the program is in the process of reconnecting. If reconnection fails, the final status return is **LM_CANTCONNECT**.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-16	LM_CANTREAD	"Cannot read data from license server system." The process cannot read data from the daemon within the timeout interval. The connection was reset by the daemon (usually because the daemon exited) before the process attempted to read data.
-17	LM_CANTWRITE	"Cannot write data to license server system." The process could not write data to the daemon after the connection was established.
-18	LM_NOSERVSUPP	"License server system does not support this feature." The feature has expired (on the server), or has not yet started, or the version is greater than the highest supported version.
-19	LM_SELECTERR	"Error in select system call." The **select** system call failed.
-20	LM_SERVBUSY	"License server system busy (no majority).", The license server is busy establishing a quorum of server nodes so that licensing can start. This error is very rare, and checkout should be retried if this occurs.
-21	LM_OLDVER	"License file does not support this version." The version requested is greater than the highest version supported in the license file FEATURE line.
-22	LM_CHECKINBAD	"Feature checkin failure detected with license server system." The checkin request did not receive a good reply from the vendor daemon (the license might still be considered in use).
-23	LM_BUSYNEWSERV	"License server system temporarily busy (new server connecting)." The vendor daemon is in the process of establishing a quorum condition. New requests from FlexEnabled applications are deferred during this period. This request should be retried.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-24	LM_USERSQUEUED	"Users are queued for this feature." This error is similar to `MAXUSERS`, but supplies the additional information that there are other users in the queue for this feature.
-25	LM_SERVLONGGONE	"License server system does not support this version of this feature." The version specified in the checkout request is greater than the highest version number the daemon supports.
-26	LM_TOOMANY	"Request for more licenses than this feature supports." A checkout request was made for more licenses than are available. This request will never succeed.
-27	LM_CANTREADKMEM	"Cannot read `/dev/kmem`."
-28	LM_CANTREADVMUNIX	"Cannot read `/vmunix`."
-29	LM_CANTFINDETHER	"Cannot find ethernet device." The ethernet device could not be located on this system.
-30	LM_NOREADLIC	"Cannot read license file." The license file cannot be read (errno == EPERM or EACCES).
-31	LM_TOOEARLY	"Feature start date is in the future." The feature is not enabled yet (current date is before the feature start date).
-32	LM_NOSUCHATTR	"No such attribute." A call to **lc_get_attr** or **lc_set_attr** specified an unknown attribute code.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-33	LM_BADHANDSHAKE	"Bad encryption handshake with vendor daemon." The FlexEnabled application performs an encryption handshake operation with the daemon prior to any licensing operations. This handshake operation failed.
-34	LM_CLOCKBAD	"Clock difference too large between client and license server system." The date on the machine does not agree closely enough with the date on the license server machine. The amount of difference allowed is set by the software vendor with **lc_set_attr(**`LM_A_MAX_TIMEDIFF,...`**)**.
-35	LM_FEATQUEUE	"In the queue for this feature." This checkout request has resulted in the process being placed in the queue for this feature. Subsequent calls to **lc_status** will yield the status of this queued request.
-36	LM_FEATCORRUPT	"Feature database corrupted in vendor daemon." The daemon's run-time feature data structures have become corrupted. This is an internal daemon error.
-37	LM_BADFEATPARAM	"Duplicate selection mismatch for this feature."
-38	LM_FEATEXCLUDE	"User/host on EXCLUDE list for feature." The user/host/display has been excluded from this feature by an end user's vendor daemon options file.
-39	LM_FEATNOTINCLUDE	"User/host not on INCLUDE list for feature." The user/host/display has NOT been included in this feature by an end user's vendor daemon options file.
-40	LM_CANTMALLOC	"Cannot allocate dynamic memory." The **malloc** call failed to return sufficient memory.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-41	LM_NEVERCHECKOUT	"Feature was never checked out." This code is returned by **lc_status** if the feature requested has never been checked out.
-42	LM_BADPARAM	"Invalid parameter." A call to **lc_set_attr** specified an invalid value for its attribute. **lc_get_attr(**`LM_A_MASTER, ...`**)** called without connection already established to server.
-43	LM_NOKEYDATA	"No key data supplied in call to **lc_new_job** or **lc_init**." No key data was supplied to the call to **lc_new_job** or **lc_init**. Some Flexible API functions will be disabled.
-44	LM_BADKEYDATA	"Invalid key data supplied." Invalid key data was supplied to the **lc_new_job** call. Some Flexible API functions will be disabled.
-45	LM_FUNCNOTAVAIL	"Function not available in this version." This Flexible API client function is not available. This could be a result of a `BADKEYDATA`, `NOKEYDATA`, or `DEMOKIT` return from **lc_new_job**.
-47	LM_NOCLOCKCHECK	"Clock setting check not available in vendor daemon." **lc_checkout** returns this code when the `CLOCK SETTING` check between the FlexEnabled application and daemon is not supported in this daemon. To disable the clock check **lc_set_attr(**`LM_A_MAX_TIMEDIFF, (LM_A_VAL_TYPE)-1`**)**
-48	LM_BADPLATFORM	"Platform not enabled." The software is running on a platform which is not supported by the vendor keys you have purchased. To purchase keys for additional platforms, contact Flexera.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-49	LM_DATE_TOOBIG	"Date invalid for binary format." The start date format in FlexNet licenses are good until the year 2027. This is probably a bad date.
-50	LM_EXPIREDKEYS	"Key data has expired." The demo vendor keys have expired. Contact Flexera for new demo keys.
-51	LM_NOFLEXLMINIT	"Not initialized." A Flexible API client function was called before **lc_new_job** was called. Always call **lc_new_job** first.
-52	LM_NOSERVRESP	"Vendor daemon did not respond within timeout interval."
-53	LM_CHECKOUTFILTERED	"Checkout request rejected by vendor-defined filter." **lc_checkout** failed because of the vendor defined routine, which is set in `lsvendor.c`: **ls_outfilter**.
-54	LM_NOFEATSET	"No FEATURESET line present in license file." **lc_ck_feats** called, but no FEATURESET line in license file.
-55	LM_BADFEATSET	"Incorrect FEATURESET line in license file." Error return from **lc_ck_feats**.
-56	LM_CANTCOMPUTEFEATSET	"Cannot compute FEATURESET data from license file." Error return from **lc_ck_feats**, which occurs because **lc_feat_set** cannot compute the FEATURESET line. This can happen because there are no FEATUREs in the file.
-57	LM_SOCKETFAIL	“socket() call failed.” This can occur when the UNIX OS runs out of system resources.
-58	LM_SETSOCKFAIL	"setsockopt() failed." The **setsockopt** call has failed. This is likely due to an OS error.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-59	LM_BADCHECKSUM	"License server system message checksum failure." Communications error—messages between the FlexEnabled application and license server are encrypted and checksummed for security and integrity. The checksum will usually fail because of poor networking communications.
-61	LM_SERVNOREADLIC	"Cannot read license file from license server system." This occurs when the license file, via **LM_LICENSE_FILE**, or **lc_set_attr(LM_A_LICENSE_FILE,** (**LM_AL_VAL_TYPE**)***path*)**, is incorrectly defined. This only occurs in `lmutil` when `LM_LICENSE_FILE` is set to *`port@host`* or *`@host`*.
-62	LM_NONETWORK	"Network software (tcp/ip) not available." This is reported on systems where this is detectable. Some systems may have this problem, but the error will not be reported as **LM_NONETWORK**—system calls will simply fail.
-63	LM_NOTLICADMIN	"You are not a license administrator." Various functions, such as **lc_remove** and **lc_shutdown**, require that the user be a license administrator, depending on how `lmgrd` was started.
-64	LM_REMOVETOOSOON	"lmremove request before the minimum lmremove interval." An **lc_remove** request occurred, but **ls_min_lmremove** (defined in `lsvendor.c`) seconds have not elapsed since the license was checked out. See **ls_vendor**.
-65	LM_BADVENDORDATA	"Unknown `VENDORCODE` struct type passed to **lc_new_job** or **lc_init**." **LM_CODE** macro was not used to define the `VENDORCODE` argument for **lc_new_job** or **lc_init**. See `lmclient.h` and `lmflex.c` for an example of how to use the **LM_CODE** macro.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-66	LM_LIBRARYMISMATCH	"Include file/library mismatch." An attempt was made to create a FlexEnabled application with mismatching source/header files and the client libraries, for example `liblmgr_trl.a`. The source code version must match that of the libraries that are linked to it.
-67	LM_NONETOBORROW	"No licenses available to borrow." All licenses that were available to borrow have already been borrowed.
-68	LM_NOBORROWSUPP	"License `BORROW` support not enabled." The license or the application doesn't support license borrowing, but borrowing was requested by a user.
-69	LM_NOTONSERVER	"`FLOAT_OK` can't run standalone on license server system." If the license server machine hostid is specified after `FLOAT_OK`, only the floating license can be used to run the application on the license server machine.
-71	LM_BAD_TZ	"Invalid TZ environment variable." On some operating systems, the end user can significantly change the date using the `TZ` environment variable. This error detects this type of theft.
-72	LM_OLDVENDORDATA	"Old VENDORCODE(3-word) struct type passed to lc_new_job() or lc_init()." **lm_init** detected that an old **LM_CODE** macro was used.
-73	LM_LOCALFILTER	"Local checkout filter rejected request." Request was denied by filter specified in **lc_set_attr(**`LM_A_CHECKOUTFILTER (LM_A_VAL_TYPE)`*`filter`***)**.
-74	LM_ENDPATH	"Attempt to read beyond end of license file path." An error occurred with the list of license files.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-75	LM_VMS_SETIMR_FAILED	"SYS$SETIMR call failed." SYS$SETIMR is used on VMS to time out certain FlexNet licensing system calls.
-76	LM_INTERNAL_ERROR	“Internal FLEXlm Error - Please report to Flexera."
-77	LM_BAD_VERSION	"Bad version number - must be floating point number, with no letters." A line in the license file has an invalid version number. **lc_checkout** was called with an invalid version character string.
-78	LM_NOADMINAPI	"FLEXadmin API functions not available." An attempt to get information from another company’s vendor daemon was made via **lc_get_attr(**LM_A_VD_*, ...**)**. This function call is only allowed for the vendor’s own vendor daemon.
-82	LM_BADPKG	"Invalid PACKAGE line in license file." PACKAGE line missing or invalid COMPONENTS. A COMPONENT has number of licenses set, with OPTIONS=SUITE. A COMPONENT has number of licenses==0.
-83	LM_SERVOLDVER	"Version of vendor daemon is too old." Vendor daemon version is older than the FlexEnabled application’s Licensing toolkit version. This is only supported with a version 5.0 or later FlexEnabled application.
-84	LM_USER_BASED	"USER_BASED license has no specified users—see license server system log." When a feature has the USER_BASED attribute, this error occurs when there no INCLUDE line in the end-user options file for this feature, or the number of users included exceeds the number authorized. See the *Programming Reference for License File–Based Licensing*, for more information about FEATURE and INCREMENT lines.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-85	LM_NOSERVCAP	"License server system doesn't support this request." This occurs when a vendor daemon with a Licensing toolkit version is older than that in the FlexEnabled application. The daemon didn't understand and respond to the request made by the application.
-87	LM_MAXLIMIT	"Checkout exceeds MAX specified in options file." End-user option MAX has been specified for this feature.
-88	LM_BADSYSDATE	"System clock has been set back." Returned from checkout call.
-89	LM_PLATNOTLIC	"This platform not authorized by license." Returned from checkout call where FEATURE line specifies PLATFORMS="...".
-90	LM_FUTURE_FILE	"Future license file format or misspelling in license file." Returned from checkout call when license file attribute was introduced in a later Licensing toolkit version than that in the FlexEnabled application.
-91	LM_DEFAULT_SEEDS	"Encryption seeds are non-unique." Returned from **lc_new_job** or **lp_checkout** when vendor name is not `demo`, but encryption seeds are default encryption seeds.
-92	LM_SERVER_REMOVED	"Feature removed during lmreread or wrong `SERVER` line hostid." Checkout failure due to two possible causes: ● The feature is removed during lmreread, but the FlexEnabled application is reading an old copy of the license file which still has removed feature. ● The hostid on the SERVER line is for a different host, so all features in this license file were removed.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-93	LM_POOL	"This feature is available in a different license pool." This is a possible response to `LM_A_VD_FEATURE_INFO` request, indicating that this `INCREMENT` line can be ignored, as it has been pooled with another line.
-94	LM_LGEN_VER	"Attempt to generate license with incompatible attributes." Occurs with `lmcrypt -verfmt` or for `lminstall -overfmt`. Also set by **lc_cryptstr** and **lc_chk_conf**.
-95	LM_NOT_THIS_HOST	“Network connect to `THIS_HOST` failed.” Returned by **checkout** when `this_host` is used as a host name. Replace `this_host` with a real host name to resolve this error.
-96	LM_HOSTDOWN	"License server machine is down or not responding." Returned by **checkout**; indicates the whole license server is not up, not just `lmadmin` or `lmgrd`.
-97	LM_VENDOR_DOWN	“The desired vendor daemon is down.” Returned by **checkout**; indicates `lmadmin` or `lmgrd` is running, but not the vendor daemon.
-98	LM_CANT_DECIMAL	“This FEATURE line can’t be converted to decimal format.” Returned by **lc_cryptstr**, `lmcrypt`, or `lminstall`.
-99	LM_BADDECFILE	“The decimal format license is typed incorrectly.” The internal checksum on the decimal line has indicated that the line has been typed incorrectly.
-100	LM_REMOVE_LINGER	“Cannot remove a linger license.” Returned to `lmremove` command. User has already exited, but license is lingering. `lmremove` doesn’t remove the linger time.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-101	LM_RESVFOROTHERS	"All licenses are reserved for others." Checkout return value when a checkout will never succeed, because the end-user options file has all licenses reserved for others.
-102	LM_BORROW_ERROR	"A FLEXid borrow error occurred."
-103	LM_TSOK_ERR	"Terminal Server remote client not allowed." The feature line does not contain TS_OK to allow Terminal Server Client usage.
-104	LM_BORROW_TOOLONG	"Cannot borrow that long." The user has specified a borrow period longer than the license allows.
-105	LM_UNBORROWED_ALREADY	"Feature already returned to license server system."
-106	LM_SERVER_MAXED_OUT	"License server system is out of network connections." The vendor daemon can't handle any more users. See the `lmadmin` or `lmgrd` debug log for further information.
-107	LM_NOBORROWCOMP	"Can't borrow a PACKAGE component."
-110	LM_NODONGLE	"Can't read dongle: check dongle or driver." In order to read the hostid (FlexID) from the dongle, the correct driver must be installed. These drivers are available at www.flexerasoftware.com or from your software vendor.
-112	LM_NODONGLEDRIVER	"Missing Dongle Driver." In order to read the hostid (FlexID) from the dongle, the correct driver must be installed. These drivers are available at www.flexerasoftware.com or from your software vendor.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-114	LM_SIGN_REQ	"SIGN= keyword required but missing from the license certificate." This is probably because the license is older than the application. You need to obtain a SIGN= version of this license from your vendor.
-115	LM_PUBKEY_ERROR	"Error in Public Key package." Rare error.
-116	LM_NOTRLSUPPORT	"TRL not supported for this platform." You are trying to use TRL, but have not installed TRL keys.
-117	LM_BORROW_LINGER_ERR	"BORROW failed." Borrowing information is invalid.
-118	LM_BORROW_EXPIRED	"BORROW period expired." Borrowed license can no longer be checked out because it has expired.
-119	LM_MUST_BE_LOCAL	"lmdown/lmreread must be run on license server machine." When licenses are borrowed, `lmdown` and `lmreread` must be run on the same machine where the license server is running.
-120	LM_BORROW_DOWN	"Cannot lmdown the server when licenses are borrowed." When licenses are borrowed, you must shut down a license server with the `-force` option.
-121	LM_FLOATOK_ONEHOSTID	"FLOAT_OK requires exactly one dongle hostid." The hostid to which the FLOAT_OK feature is node-locked must be a single FlexID, not a list of FlexID hostids.
-122	LM_BORROW_DELETE_ERR	"Unable to delete local borrow info."

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-123	LM_BORROW_RETURN_EARLY_ERR	"Returning a borrowed license early is not supported." This support is enabled in the vendor daemon via the `ls_borrow_return_early` variable.
-124	LM_BORROW_RETURN_SERVER_ERR	"Error returning borrowed license."
-125	LM_CANT_CHECKOUT_JUST_PACKAGE	"A PACKAGE component must be specified."
-126	LM_COMPOSITEID_INIT_ERR	"Composite Hostid not initialized."
-127	LM_COMPOSITEID_ITEM_ERR	"An item needed for composite hostid missing or invalid."
-128	LM_BORROW_MATCH_ERR	"Error, borrowed license doesn't match any known server license."
-135	LM_EVENTLOG_INIT_ERR	"Error enabling event log."
-136	LM_EVENTLOG_DISABLED	"Event logging is disabled."
-137	LM_EVENTLOG_WRITE_ERR	"Error writing to event log."
-139	LM_TIMEOUT	"Timeout."
-140	LM_BADCOMMAND	"Bad message command."
-141	LM_SOCKET_BROKEN_PIPE	"Error writing to socket. Peer has closed socket."
-142	LM_INVALID_SIGNATURE	"Attempting to generate version specific license tied to a single hostid, which is composite."
-143	LM_UNCOUNTED_NOT_SUPPORTED	"Version-specific signatures are not supported for uncounted licenses."
-144	LM_REDUDANT_SIGNATURES	"License template contains redundant signature specifiers."
-145	LM_BADCODE_V71_LK	"Invalid V71_LK signature."
-146	LM_BADCODE_V71_SIGN	"Invalid V71_SIGN signature."
-147	LM_BADCODE_V80_LK	"Invalid V80_LK signature."
-148	LM_BADCODE_V80_SIGN	"Invalid V80_SIGN signature."

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-149	LM_BADCODE_V81_LK	"Invalid V81_LK signature."
-150	LM_BADCODE_V81_SIGN	"Invalid V81_SIGN signature."
-151	LM_BADCODE_V81_SIGN2	"Invalid V81_SIGN2 signature."
-152	LM_BADCODE_V84_LK	"Invalid V84_LK signature."
-153	LM_BADCODE_V84_SIGN	"Invalid V84_SIGN signature."
-154	LM_BADCODE_V84_SIGN2	"Invalid V84_SIGN2 signature."
-155	LM_LQ_REQ	"License key required but missing from the license certificate." The application requires a license key in the license file. You need to obtain a license key version of this license file from your vendor.
-156	LM_BADAUTH	"Invalid AUTH={} signature."
-157	LM_REPAIR_NEEDED	"Trusted Storage compromised; repair needed." For information on trusted storage breaks, see flxActCommonProdLicSpcTrustFlagsGet. Instructions for performing repair transactions can be found in the *Programming Reference for Trusted Storage–Based Licensing*.
-158	LM_TS_OPEN	"Trusted Storage open failure."
-159	LM_BAD_FULFILLMENT	"Invalid fulfillment record."
-160	LM_BAD_ACTREQ	"Invalid activation request received."
-161	LM_TS_NO_FULFILL_MATCH	"No fulfillment exists in trusted storage which matches the request."
-162	LM_BAD_ACT_RESP	"Invalid activation response received."
-163	LM_CANTRETURN	"Cannot return the specified activation."
-164	LM_RETURNEXCEEDMAX	"Return count(s) would exceed max for the fulfillment"
-165	LM_NO_REPAIRS_LEFT	"No repair count left."

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-166	LM_NOT_ALLOWED	"Specified operation is not allowed." The requested function is not allowed. For example calling **lc_set_attr** after a feature has already been checked out from the license server.
-167	LM_ENTLEXCLUDE	The user requesting an activatable license is not allowed a license from the available fulfillment records. For information about EXCLUDE_ENTITLEMENT, see the *License Administration Guide*.
-168	LM_ENTLNOTINCLUDE	The user requesting an activatable license is not allowed a license from the available fulfillment records. For information about INCLUDE_ENTITLEMENT, see the *License Administration Guide*.
-169	LM_ACTIVATION	An error occurred when using the Activation Borrow API functions.
-170	LM_TS_BADDATE	"Invalid date format in trusted storage." The fulfillment record has its time trust flag set to untrusted, therefore the requested was refused. See flxActCommonProdLicSpcTrustFlagsGet for a description of this flag.
-171	LM_ENCRYPTION_FAILED	"Message Encryption failed" Unable to encrypt message sent between the grid filter server and the vendor daemon.
-172	LM_DECRYPTION_FAILED	"Message Decryption failed" Unable to decrypt message sent between the grid filter server and the vendor daemon.
-173	LM_BADCONTEXT	"Bad Filter Context" Internal error.
-174	LM_SUPERSEDE_CONFLICT	"SUPERSEDE feature conflict."
-175	LM_INVALID_SUPERSEDE_SIGN	"Invalid SUPERSEDE_SIGN syntax."
-176	LM_SUPERSEDE_SIGN_EMPTYSTRING	"SUPERSEDE_SIGN does not contain a feature name and license signature."

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-177	LM_ONE_TSOK_PLATFORM_ERROR	"ONE_TS_OK is not supported in this Windows Platform."
-178	LM_ONE_TSOK_MTX_ERR	"Internal error - 178. Please report to Flexera."
-179	LM_ONE_TSOK_ERR	"Only one terminal server remote client checkout is allowed for this feature."
-180	LM_SSIDNULL	"Internal error - 180. Please report to Flexera."
-181	LM_SMTXNOTREL	"Internal error - 181. Please report to Flexera."
-182	LM_MTXNOPERM	"Internal error - 182. Please report to Flexera."
-183	LM_COMPOSITEID_ETHER_ERR	"More than one ethernet hostid not supported in composite hostid definition." The composite hostid definition you are using is invalid. Ensure that it only contains a single Ethernet hostid. See 'Using Composite Hostids' in *Programming Reference for License File–Based Licensing*.
-184	LM_LIC_FILE_CHAR_EXCEED	"The number of characters in the license file paths exceeds the permissible limit." See 'License File Limits' in *Programming Reference for License File–Based Licensing*.
-185	LM_TZ_INVALID_SYNTAX	"Invalid TZ keyword syntax." Returned at license encryption time if the TZ keyword syntax is not valid. Ensure that, if specifying multiple time zones, the space delimiter is used and the values are enclosed in quotation marks. If using the SERVERTZ value, it must be used by itself.
-186	LM_TZ_INVALID_TZONE_SPEC	"Invalid time zone override specification in the client."

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-187	LM_TZ_INVALID_TZONE_INFO	"The time zone information could not be obtained." Returned at run time. The `TZ` keyword is used in the license file, but time zone information could not be obtained for the machine on which the FlexEnabled application is attempting to run. The checkout request is denied.
-188	LM_TZ_UNAUTHORIZED	"License client time zone not authorized for license rights." Returned at run time. The time zone of the FlexEnabled client does not match the time zone specified for the feature using the `TZ` keyword: As a result, license usage for the feature is not authorized and the checkout request is denied.
-189	LM_INVALID_VM_PLATFORMS	"Invalid syntax for `VM_PLATFORMS` keyword." Returned at license encryption time if an invalid keyword is specified for use with `VM_PLATFORMS`.
-190	LM_VM_PHYSICAL_ONLY	"Feature can be checked out from physical machine only." Returned at run time if a checkout request is made from an application on a virtual machine.
-191	LM_VM_VIRTUAL_ONLY	"Feature can be checked out from virtual machine only." Returned at run time if a checkout request is made from an application on a physical machine.
-192	LM_VM_NOT_SUPPORT	"VM platform not autorized by license." Returned at run time when the feature definition line specifies a virtual environment but the client is running in another virtual environment. For example, VM_PLATFORMS=VMW but the client is checking out from HPYPER-V virtual environment.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-193	LM_VM_BAD_KEY	"FNP vendor keys do not support Virtualization feature." Returned in the following scenarios if the your vendor keys do not support virtualization: ● At build time if the optional vendor variable `ls_allow_vm` is set to `PHYSICAL` or `VM_ONLY`. ● At encryption time if the optional license keyword `VM_PLATFORMS` is set to `PHYSICAL` or `VM_ONLY`. ● At run time, this message is displayed to the end user if the FlexEnabled application is built using vendor keys that do not support virtualization and attempts to check out a feature that uses the license keyword `VM_PLATFORMS` set to `PHYSICAL` or `VM_ONLY`. Contact Flexera to obtain vendor keys that support virtualization.
-194	LM_MAXLIMIT_EXCEED	"Checkout request denied as it exceeds the MAX limit specified in the options file."
-198	LM_UNSUPPORTED_FEATURE_HOSTID	Unsupported hostid provided in feature line.
-199	LM_SERVERQUERY_LOAD_REQUEST_FAILED	Failed to load ServerQuery request.
-200	LM_SERVERQUERY_RESPONSE_FAILED	Failed to generate ServerQuery response.
-201	LM_INVALID_IPADDRESS	"Invalid IP address used while overriding." Returned when the value specified for `LM_A_INTERNET_OVERRIDE` is not a valid IP address.
-202	LM_UNRESOLVED_FEATURE	Returning borrowed feature failed as same feature borrowed from different Vendor daemons.
-203	LM_TS_GET_TOTAL_FEAT_COUNT_FAILED	Failed to get the total feature count.
-204	LM_TS_ACT_RECLAIM_NOT_ALLOWED	Activation borrow reclaim operation is not allowed.
-205	LM_TS_ACT_RECLAIM_FAILED	Failed to perform activation borrow reclaim operation.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-206	LM_TS_ACT_RECLAIM_NO_DR_FOUND	No deduction record found for the requested client host.
-207	LM_TS_ACT_RECLAIM_LS_TS_NOT_SUPPORTED	The license server does not support trusted storage.
-208	LM_TS_SAVE_ERROR	Trusted storage could not be saved. Used in vendor daemon for debug log purposes. In FlexNet utilities, the usage of this error code is limited to activation borrow reclaim and in that case, “Trusted storage could not be saved by the license server” is seen as a failure reason. This error code is not used in FlexNet client or an activation client.
-209	LM_MAXSERVERS	Maximum number of servers reached.
-210	LM_LICSRVC_FAILED	License service failed to return VM attributes.
-211	LM_NO_VMATTR_PHY_SYS	VM attributes not available on physical machine.
-212	LM_LIC_SVC_DISABLED	FlexNet Licensing Service was found to be disabled.
-213	LM_LIC_SVC_NOTINSTALLED	FlexNet Licensing Service is not installed.
-214	LM_LIC_SVC_VER_ERR	FlexNet Licensing Service version is not as expected.
-215	LM_VM_HOSTID_NOT_AVAILABLE	The VM Host ID is not available.
-216	LM_COAVAIL_RECONN_NOT_ALL_CNT_AVAIL	Failed to get all requested license(s) in reconnection.
-217	LM_COAVAIL_FLAG_NOT_SUPPORTED	The supplied checkout option flag is not supported.
-218	LM_COAVAIL_PACKAGE_NOT_SUPPORTED	The checkout-avail flag is not supported with PACKAGE licenses.
-219	LM_COAVAIL_UNSUPPORTED	The checkout-avail (COAVAIL) checkout is not supported.

Table 10-1 • Errors Returned from lc_* Functions

Error Number	Symbolic Name	Description
-220	LM_ONLY_TS_ALLOWED	Only trusted storage features are allowed to checkout. Failed to get licenses from trusted storage.
-221	LM_TRANSFER_TO_SELF_NOT_ALLOWED	A transfer from a license server on the same machine is not allowed.
-222	LM_HOSTID_UNSUPPORTED	An unsupported hostid.
-223	LM_LIC_SVC_REINSTALL_SVC	FlexNet Licensing Service requires new install.
-224	LM_LIC_SVC_NOT_AVAILABLE	FlexNet Licensing Service is not available.
-225	LM_LIC_SVC_COMMS_ERROR	Error in communication with the FlexNet Licensing Service.
-226	LM_TPM_LICSRVC_FAILED	FlexNet Licensing Service failed to return TPM attributes.
-227	LM_TPM_VERSION_UNSUPPORTED	TPM version is not supported.
-228	LM_TPM_DISABLED	TPM is disabled.
-229	LM_TPM_PROPS_NOT_AVAILABLE	TPM properties are not available.
-230	LM_TPM_HOSTID_NOT_AVAILABLE	TPM hostid is not available.
-231	LM_TPM_PLATFORM_UNSUPPORTED	TPM is not supported on this platform.
-232	LM_TPM_UNKNOWN_ERROR	TPM attributes could not be obtained.
-237	LM_VD_MAX_CLIENTS_REACHED	Maximum number of connections reached for this vendor daemon.

Glossary

Term	Meaning
activatable license	A license right in trusted storage that can be activated to a FlexEnabled client. See also license group.
activated feature line	See feature definition line.
activation	In the FlexNet Publisher Licensing Toolkit, this is a process used to load license rights into trusted storage. Different activation types include: ● programmatic activation ● manual activation ● short-code activation ● local activation In FlexNet Operations, this is a process used to generate license rights in either trusted storage and in license files.
Activation API	Functions available in the FlexNet Publisher Licensing Toolkit libraries that enable you to initiate transactions and manage license rights in trusted storage. The Activation API is used to develop the functionality in an activation utility.

Term	Meaning
activation client	The system that works with an activation server to receive license rights. This system can be either a license server or a FlexEnabled client. The activation client initiates an activation, repair, or return transaction with an activation server, using an activation utility. The following transaction types can be initiated by an activation client: ● activation transaction ● return transaction ● repair transaction
activation request	A message sent by an activation utility to an activation server that initiates an activation transaction.
activation response	A message sent from an activation server to an activation client, in response to an activation request. This message defines the license rights.
activation server	A system that responds to requests for license rights from an activation client. The activation server can be one of the following: ● a license server ● FlexNet Operations ● in a test environment, the Manual Response Generator (responsegen.exe) An activation server can also perform repair and return transactions.
activation specification record (ASR)	A signed XML file used with local activation and short-code activation that contains information about license rights, configuration settings for the publisher's trusted storage segment, and additional information required by the activation transaction.
activation transaction	A sequence of activation request and activation response messages, and related actions that result in license rights being loaded into trusted storage.
activation utility	A utility that initiates an activation transaction, repair transaction, or return transaction by sending a request to an activation server. This utility also processes the response received from the activation server. It uses the Activation API to perform these capabilities. It is written by the software publisher to meet specific business process requirements. This application can be either separate from or embedded in the FlexEnabled application.
anchor	A link between the system and the trusted storage that is used to detect whether trusted storage has been tampered with, deleted, or restored. Anchors are also used to ensure that information in trusted storage about trial licenses persists beyond the expiration date.
ASR file	See activation specification record (ASR).
binding	A capability used with trusted storage where one or more properties of a system, combined together into a unique identifier, are used to ensure that license rights held in trusted storage are not copied to another system.

Term	Meaning
borrowing	A capability available in FlexEnabled applications that enables a user to temporarily obtain a license in such a way that the license can be used locally for a limited period of time while the user is disconnected from the local network.
bulk entitlement	An entitlement created for an order of multiple copies of a single product or suite. Bulk entitlements are used for sales through reseller channels.
checkin	The transaction that a FlexEnabled application performs to return a license after it has been checked out. This operation is not the same as an activation transaction. FlexEnabled applications must perform a checkout and checkin transaction to use a license.
checkout	The transaction that a FlexEnabled application performs to request a license. This transaction is different than activating a license. FlexEnabled applications must perform a checkout and checkin transaction to use a license.
client system	A general term used to define any system that interacts with a server in a client-server relationship. When using the FlexNet Publisher Licensing Toolkit there are two types of clients: ● activation client ● FlexEnabled client
composite transaction	Available in FlexNet Publisher 11.8, composite transactions allow multiple actions to be combined into a single transaction with FlexNet Operations. Also called multi-action or V2 transactions. From FlexNet Publisher 11.14.0, composite transactions are also supported for all license servers.
concurrent license	A license that can be shared among users by allowing each user to check out and then return the license. These licenses are served to FlexEnabled applications from a license server. If all licenses are being used, an additional user cannot run the FlexEnabled application until one of the other users check in their license. When one user finishes using the license, another user can begin using it. Also called floating license or served license rights.
config response	A response to any request from an activation client that contains information about how the publisher's trusted storage segment must be configured (called the trusted storage segment configuration). This response is processed by the activation utility and results in a segment of trusted storage being created or re-configured for that publisher.
consumer	An individual or small business who purchases a license for their individual use. Compare with enterprise.
counted	A type of license right or license model that defines a multiple quantity of licenses and allows concurrent use of the licenses. This term is taken from the fact that the license server counts the number of licenses that are being used.
debug log file	A file used by the license server to record status and error messages that are useful for debugging the license server. Each license server can have one or more of these files.

Term	Meaning
demo license	See trial license.
duplicate grouping	A capability in the FlexNet Publisher Licensing Toolkit used to uniquely identify a single client (or client checkout request) for the purpose of counting the number of licenses that are used. Duplicate grouping defines a set of rules under which multiple checkout requests can share the same license(s).
embedded activation	See local activation.
encryption seed	A set of values defined by a software publisher that are used when creating a publisher-specific vendor daemon.
end-user	A person who uses a FlexEnabled application. This role is not the same as a license administrator.
end-user system	See client system.
end-user license administrator	See license administrator.
enterprise	An organization that purchases licenses on behalf of a number of individual users. Compare with consumer.
entitlement	A representation of the license model and product that a customer has bought. It is a definition only. The customer will need to fulfill the entitlement so that the license rights can be populated to license files or trusted storage. In FlexNet Operations, there are two types of entitlements: ● bulk entitlement ● simple entitlement Each entitlement is uniquely identified by an entitlement ID.
entitlement ID	In the FlexNet Publisher Licensing Toolkit, an attribute in trusted storage that stores a value used to identify the customer's entitlement. In FlexNet Operations, a value that uniquely identifies the entitlement.
expiring license	License rights granted to a customer for a limited duration. When the license rights have expired, the licenses can no longer be checked out.

Term	Meaning
feature	In the context of a software application, this is a single unit of capability. In the FlexNet Publisher Licensing Toolkit, this is an identifier used to associate a license model to a unit of capability in a software application. The software publisher defines which features in their software application map to a single feature in the FlexNet Publisher Licensing Toolkit. For example, a feature in the FlexNet Publisher Licensing Toolkit could represent any of the following: ● The entire software application. ● A subset of menu items. ● A specific set of data available from the software application. ● A process in the software application. The software publisher also defines the license model for each FlexNet Publisher Licensing Toolkit feature. This gives software publishers the ability to define a different license model for each feature in the software application.
feature bundle	A collection of features in FlexNet Operations. It exists primarily to enable modularity and re-use.
feature definition line	An entry in a license file or fulfillment record that describes a feature. Each line begins with the keyword FEATURE, INCREMENT, or UPGRADE. In license files, feature definition lines are used with SERVER and VENDOR lines to define the license model. In trusted storage, feature definition lines are used with the fulfillment record properties to define the license model.
feature line	See feature definition line.
file-based activation	See manual activation.
FlexEnabled application	A software application that uses the FlexNet Publisher Licensing Toolkit to implement its license models.
FlexEnabled client	A client system where a FlexEnabled application is installed.
Flexible API	Functions available in the FlexNet Publisher Licensing Toolkit libraries that allow FlexEnabled clients to access license rights.
FlexNet Agent	A utility that runs on a license server and allows communication between the license server and other FlexNet products.
FlexNet Manager	This is a license server management and license management decision support solution in the FlexNet product family that allows companies to centrally manage license servers and license usage across the enterprise.
FlexNet Operations	A product in the FlexNet family of products that manages entitlements and generates license rights.

Term	Meaning
floating license	A license that can be shared among users and is served to FlexEnabled applications from a license server. When a user finishes using the license, another user can begin using it. Also called concurrent license or served license rights.
fulfill	The action of getting license rights.
fulfillment record	In the FlexNet Publisher Licensing Toolkit, a data structure specific to trusted storage that defines license rights.
heartbeat	A type of message sent periodically between two systems (for example, a FlexEnabled application and a license server) to identify whether each is running and able to communicate with the other.
hop count	In the FlexNet Publisher Licensing Toolkit, the number of times that license rights in a single fulfillment record have been transferred from one license server to another.
hostid	In the FlexNet Publisher Licensing Toolkit, the value of a specific system attribute that uniquely identifies the host. For example, if IP address is a hostid type, then the hostid might be 129.87.33.101. Hostids are used in node-locked license.
hostid type	A type of attribute that is available to uniquely identify a system. For example, disk serial number, IP address, or MAC address.
hybrid license	In the FlexNet Publisher Licensing Toolkit, a type of license right in the license group on trusted storage that can function as either a concurrent license or as an activatable license.
internet activation	See programmatic activation.
keyboard activation	See short-code activation.
license administrator	A person in an enterprise who is responsible for installing the FlexEnabled application and administering the licenses and license servers.
license file	A text file, usually with the `.lic` extension, that contains license certificates from one or more publishers.
license-file list	See license search path.
license generator	A general term used to describe any mechanism a software publisher uses to generate license rights for their customers. The following FlexNet capabilities and products can function as a license generator: ● FlexNet Operations: generates license rights for trusted storage and license files. ● lmcrypt: generates license rights for license files.

Term	Meaning
license group	A mechanism used to describe licenses rights held in server-side trusted storage. These groups are not used with license rights held in license files. ● **activatable**—A license right in trusted storage that can be activated to a FlexEnabled client. ● **concurrent**—license rights in this group can be shared among users by having a FlexEnabled client to check out, and then return the license. ● **hybrid**—these license rights have the same characteristics as both the activatable license group and concurrent license group. They can be activated to client-side trusted storage on a FlexEnabled client or checked out by a FlexEnabled client. Although you see these group names used to describe license rights in client-side trusted storage, the capabilities of the license rights do not change. Once license rights are in client-side trusted storage, you cannot transfer, serve, or activate them to another system. The FlexEnabled application can only check out and check in these licenses. The group names are used when a license right is returned to the license server, so that it is assigned to the correct license group on the license server.
license model	The set of characteristics that determine the conditions under which (for example, how, when, where, and by whom) a FlexEnabled application can be used.
license rights	Information in trusted storage or license files that defines the license model and controls user access to a specific FlexEnabled application.
license search path	A property that the FlexEnabled application uses to locate where license rights are stored. Sometimes known as a license-file list.
license server	A set of files from the FlexNet Publisher Licensing Toolkit, that work together to manage and serve licenses to FlexEnabled applications. A license server is required to support a concurrent license model. The license server consists of a license server manager, one or more vendor daemons, zero or more options files, zero or more debug logs, and, optionally, a report log. These components are available in the FlexNet Publisher Licensing Toolkit. If the license server supports license rights held in trusted storage, it also includes an activation utility. It may, optionally, act as an activation server.
license server system	See license server.
license server manager (lmadmin or lmgrd)	A license server manager `lmadmin` or `lmgrd` executable that forwards connections from a FlexEnabled application to the correct vendor daemon. Because the vendor daemons are responsible for authenticating requests and serving licenses, a single license server manager can be used with multiple vendor daemons on the license server.

Term	Meaning
license sharing	A capability available in the FlexNet Publisher Licensing Toolkit where multiple license requests from the same user, host, or display results in only one license being used. Also known as duplicate grouping.
licensing client	See FlexEnabled client.
lmadmin	The license server manager introduced in version 11.6 which includes a Web-based GUI.
lmgrd	A command-line-based license server manager.
local license rights	License rights that are stored on the same system where the FlexEnabled application is running, rather than on a license server.
local activation	A type of activation where license rights are loaded using an ASR file. The ASR file must reside with the FlexEnabled application. This does not require user intervention.
machine ID	A value, generated using attributes of the system, that is used to bind trusted storage to that system.
maintenance	A maintenance represents a customer's right to obtain updates and upgrades to a particular licensed product or suite for a specified duration. Used in FlexNet Operations.
manual activation	An type of activation where license rights are transmitted from an activation server to another system using a set of request and response messages that are formatted as XML. These messages are saved to and read from physical XML files. An activation utility loads the license rights from an XML file into trusted storage.
Manual Response Generator (responsegen.exe)	A tool included in the FlexNet Publisher Licensing Toolkit that enables software publishers to test the activation process offline using XML request and response messages stored as files.
node-locked license	A license that can be used only when the FlexEnabled application is run on a specific system or by a specific user (as defined by the hostid).
options file	A configuration file available on the license server that license administrators can use to either change license server behavior or implement certain licensing policies. Each software publisher has its own separate options file that works with a specific vendor daemon.
orderable	Anything that can be purchased by a customer. Orderables can include suites, products, documentation, and maintenances.
package	In the FlexNet Publisher Licensing Toolkit, a set of features that can be grouped and licensed together.
part number	A unique identifier, typically correlated with an order system, assigned by a publisher to a product.

Term	Meaning
permanent license	License rights granted to a customer that do not expire.
product	Anything that can be sold by a publisher and can be licensed.
product ID	In the FlexNet Publisher Licensing Toolkit, one of the values that uniquely identifies the license rights in trusted storage. See also entitlement ID.
programmatic activation	A type of activation where license rights are transmitted from one system to another across a network using a set of request and response messages that are formatted as XML. This process is the identical to manual activation, except that it does not generate a physical XML file. An activation utility loads the license rights from the XML message into trusted storage. This is an automated process and does not require user intervention.
publisher ID	A value used by trusted storage to uniquely identify the software publisher's logical and physical licensing components. Flexera distributes this identifier to software publishers. It is used to prepare certain FlexNet Publisher Licensing Toolkit files that access trusted storage. This value is not the same as the vendor daemon.
quorum	A condition specific to three-server redundancy that is met when at least two of the three license servers are running and can communicate with each other using heartbeats.
rehost	The process of reissuing a license for a different system. This process requires software publishers to perform certain activities to redefine the license rights so that they will work properly on the new system.
repair request	A message sent by an activation utility to an activation server that initiates a repair transaction. The activation server provides the information that repairs the license rights in trusted storage. An activation utility is an application that initiates the repair transaction.
repair response	A message sent from an activation server to an activation client, in response to a repair request. This message contains information that is used to the repair the license rights.
repair transaction	A sequence of repair request and repair response messages, and the related actions that result in license rights being repaired in trusted storage.
report log file	A file that runs on a license server that contains data about the features used by a single vendor daemon. Report logs are encrypted and cannot be read by a person, but are used by FlexNet Manager to produce reports.
responsegen	See Manual Response Generator (responsegen.exe).
return request	A message sent by an activation utility to an activation server that initiates a return transaction. The activation server receives the license rights returned by an activation client. An activation utility is an application that initiates the return transaction.
return response	A message sent from an activation server to an activation client, in response to a return request.

Term	Meaning
return transaction	A sequence of return request and return response messages, and the related actions, that result in license rights being returned from trusted storage to the activation server.
served license rights	License rights that are stored on a license server and are checked out when needed by FlexEnabled applications that run on client systems. These license rights are used for concurrent licensing.
short code	A multi-character code used with a short-code activation transaction, a repair transaction, or a return transaction.
short-code activation	A type of activation where license rights are loaded from an ASR file using an activation transaction that requires a specific code (called a short code). Often the user enters this code into a prompt from the application. This is also is used for licensing via automated telephony.
signature	A secure multi-character string added to the license certificate that ensures it has not been modified.
simple entitlement	An entitlement for a single, specific customer.
single-action transaction	This is the default transaction type for FlexNet Publisher versions 11.7 and earlier. Each transaction can include only a single action. Also known as legacy or V1 transactions. From FlexNet Publisher version 11.14, single actions are deprecated.
suite	A set products grouped together that are functionally complementary and associated with one or more license models.
supersede	A capability in the FlexNet Publisher Licensing Toolkit, used to replace the characteristics of old license rights with new characteristics.
three-server redundancy	A capability in the FlexNet Publisher Licensing Toolkit that enables license administrators to configure three license servers in a failover cluster.
triad	A set of three license servers which operate together to support three-server redundancy.
trial license	License rights granted to a user for a limited duration so that the user can evaluate the software application.
trusted storage	A tamper-proof area on the system that stores license rights. This storage location is different than license files and requires a different mechanism for loading and managing license rights. Compare with license file.
trusted configuration	A set of attributes that define the security configuration of a segment of trusted storage. Each software publisher can configure their segments of trusted storage differently. This configuration defines the settings for binding, anchoring, and windback detection.
trusted configuration file	A file that stores the attributes that define a software publisher's trusted storage segment configuration.

Term	Meaning
trusted storage configuration	See trusted configuration.
uncounted	A license right attribute that allows an unlimited number of concurrent uses of a FlexEnabled application. Compare with counted.
unserved license rights	These are license rights that are not served from a license server. These license rights are located with the FlexEnabled application. Also known as local license rights.
upgrade	A process where a user installs a newer version of a FlexEnabled application and also updates the license rights so that they can use this newer version.
upsell	The process of transitioning from a product of lesser functionality to one of greater functionality in the same product line.
vendor daemon	One of the files that is a part of the license server. This executable is customized and built by the software publisher using files in the FlexNet Publisher Licensing Toolkit. The vendor daemon is responsible for communicating with the FlexEnabled application and issuing licenses.
Vendor Certificate Generator	A publisher-built component, that is a part of FlexNet Operations, that creates license certificates or signed feature lines in fulfillment records. This functions similarly to lmcrypt.
vendor name	A value used to uniquely identify a software publisher's logical and physical licensing components. It is used to prepare certain files in the FlexNet Publisher Licensing Toolkit that manage license rights in both trusted storage and license files. Flexera distributes this identifier to software publishers. This value is not the same as the publisher ID.
windback	A condition where the system clock has been set to an earlier date or time. One reason users change the system clock is to extend the license duration.
windback detection	A capability in the FlexNet Publisher Licensing Toolkit that detects whether a user has reset the system clock. This prevents a user from extending the duration of the license.
windback tolerance	An acceptable amount of windback to a system clock that won't trigger windback detection. A certain tolerance is normally allowed to account for time zone differences or Daylight Saving Time.

Index

Symbols

$HOME/.flexlmrc 108

A

activatable license 485
activated feature line, See feature definition line
activation 485
    short code 494
Activation API 177, 485
    common library functions 179
    error code description 195
    error for last operation 194
    error returned by FlexNet Operations 195
    objects 178
activation application 486
activation client 486
activation library
    initialization 216
    location 217
activation licensing components
    application activation utility 204, 341
    feature definition line 181, 216
    server activation utility 204, 389
    SOAP module 205, 206, 207, 210, 211, 214
activation request 357, 359, 360, 405, 412, 486
activation response 486
activation server 486
    address 208
activation specification record 179, 220, 221, 223, 259, 261, 332, 335, 486
activation transaction 342, 389, 394, 486
    communication link for 204
    duration 354
    entitlement ID 353
    expiration date 355, 405, 411, 422, 429, 439
    file-based 348, 350, 396, 398, 441, 445
    file-based repair 433, 434, 448
    file-based return 364, 366, 417, 418, 442, 447
    repair 342, 376, 378, 389, 430, 432, 436
    return 342, 361, 363, 369, 389, 412, 414, 421
    sending activation request 347
    short code repair 334, 337
    short code, warning 339
    short codes 335, 367, 382
    short codes, generating 338
AMZN_AMI 65, 66
AMZN_EIP 65, 66
anchor 486

B

binding 486
borrowing 487
bulk entitlement 487

C

callback function 211–212
checkin 487
checkin functions 19
checkout 487
checkout functions 19
checkout options flags 33
client system 487
COAVAIL
    behavior for multiple checkouts 36
    LM_A_USER_RECONNECT_V1 125
    LM_A_USER_RECONNECT_V1_DATA 125

LM_A_USER_RECONNECT_V1_DONE 125
retrieving license count 36
COAVAIL checkout 33
Common Vendor Daemon 100
composite hostid, initializing 96
composite transaction 487
concurrent license 487
config response 487
consumer 487
counted license 487

D

daemon death
automatic reconnection 123
detecting 111
daemon, vendor 495
debug log file 487
deduction records 247
demo license, See trial license
Denial of service 143, 146
DisplayString() 145
dup_group
DISPLAY 37
HOST 37
USER 37
VENDOR 37
duplicate grouping 488
and lingering licenses 115
and RESERVE 37
display 37
host 37
mask 37
none 37
site 37
user 37
vendor 37

E

embedded activation, See local activation
encryption seed 488
end user 488
end-user license administrator, See license administrator
end-user machine, See client system
enterprise 488
entitlement 488, 491
entitlement ID 488
error messages
internationalization 43
localization 43
error reporting functions 20
expiration date 242
expiring license 488

F

feature 489
feature bundle 489
feature definition line 485, 489
FEATURE/INCREMENT line, See feature definition line
file
license 490
options 492
report log 493
file-based activation, See manual activation
FlexEnabled application 489
FlexEnabled client 489
Flexible API 489
checkin and checkout functions 19
error and warning reporting functions 20
function summary 24
functions by category 18
heartbeat management functions 21
information functions 22
job handling functions 18
license file generation functions 23
license file installation functions 23
required functions 17
FlexNet Agent 489
FlexNet Licensing Service
installing (Linux) 164
installing (OS X) 166
FlexNet Operations 489
FlexNet Operations Server 209
error codes 195
floating license 490
flxActCommonGetProtectionMode 190
flxActCommonHandleGetUniqueMachineNumber 201
flxActDiagTrustedStorageHealthCheck 263
FNP_HOSTID_INTERNET_V4 67
FNP_HOSTID_INTERNET_V6 67
fulfill 490
fulfillment record 351, 490
definition 179
status 246
fulfillment type 244

G

gethostname()
and LM_A_HOST_OVERRIDE 145

H

heartbeat management functions 21
heartbeats 490
how lc_heartbeat() works 62

hop count 490
hostid 490
hostid type 490
HOSTID_COMPOSITE 67
HOSTID_DEFAULT 67
HOSTID_DISK_SERIAL_NUM 65
HOSTID_DISPLAY 67
HOSTID_ETHER 65, 66
HOSTID_FLEXID10 65, 66
HOSTID_FLEXID9 65, 66
HOSTID_HOSTNAME 67
HOSTID_INTERNET 65, 66, 67
HOSTID_LONG 65, 66
HOSTID_STRING 65
HOSTID_TPM_ID1 66
HOSTID_USER 67
HOSTID_VENDOR 67
HTTPS 210
hybrid license 490

I

informational functions 22
installing
    FlexNet Licensing Service (Linux) 164
    FlexNet Licensing Service (OS X) 166
internet activation, See programmatic activation

K

keyboard activation, See short-code activation

L

l_n36_buf 72
l_new_hostid() 88
    and lc_free_hostid() 93
lc_alarm() 455
lc_auth_data() 25
lc_baddate() 456
lc_borrow_return() 88
lc_check_key() 90
lc_checkin() 28
    and lingering licenses 115
lc_checkout() 29
    and LM_A_CHECKOUT_FILTER 141, 142
lc_chk_conf() 456
lc_ck_feats() 456
lc_cleanup() 92
lc_copy_hostid() 456
    and lc_free_hostid() 93
lc_crypt() 456
lc_cryptstr() 38
lc_disalarm() 456
lc_disconn() 456
lc_display() 456
lc_err_info() 42
lc_errstring() 44
    and lc_perror() 76
lc_errtext() 456
lc_expire_days() 45
lc_feat_list() 46
lc_feat_set() 456
lc_first_job() 47
lc_fnpservice_present() 54
lc_free_hostid() 92, 93
    and l_new_hostid() 88
lc_free_job() 55
lc_free_mem()
    and lc_cryptstr() 39
lc_get_attr() 56
lc_get_config() 93
lc_get_errno() 57
lc_get_feats() 457
lc_get_version() 58, 59
lc_gethostid() 457
lc_getid_type() 457
lc_heartbeat() 60, 123
    and LM_A_PERIODIC_COUNT 149
lc_hostid() 63
lc_hostname() 457
lc_idle() 67
lc_init_simple_composite() 95
lc_init() 68
    and license generators 72
lc_isadmin() 457
lc_log() 70
lc_new_job() 71
    and lc_free_job() 55
lc_next_conf() 97
lc_next_job() 74
lc_perror() 75
    and LM_A_USER_EXITCALL 122
lc_remove() 98
lc_set_attr_vendor 100
lc_set_attr() 76
    and lingering licenses 115
    setting license file location 32
lc_set_errno() 457
lc_shutdown() 101
lc_status() 78
lc_test_conf() 102
lc_timer() 457
lc_tpmstatusget() 79
lc_userlist() 81
    and LM_A_CHECKOUT_DATA 138
lc_username() 457

lc_vsend() 83
libcmt.lib (/MT) 42
license
    counted 487
    served 494
    unserved 495
license administrator 490
license checkin functions 19
license checkout functions 19
license count
    retrieving in COAVAIL checkout 36
license file 490
    setting location 32
license file generation functions 23
license file installation functions 23
license file setting
    $HOME/.flexlmrc 108
    Windows registry 108
license generator 490
license group 491
license model 491
license rights 491
    local 492
license search path 491
License Server 209
license server 491
license server manager 491
license server manager, definition 491
license server system 491
license sharing 492
license-file list, See license search path
licenses
    activatable 395, 397, 446
    concurrent 21, 395, 397, 446
    expiration date 242
    hybrid 395, 397, 446
    overdraft 395, 397, 446
licensing client 492
lingering licenses
    and lmremove 474
    overview 115
LM_A_APP_DISABLE_CACHE_READ 108
LM_A_BEHAVIOR_VER 137, 458
LM_A_BORROW_EXPIRE 108, 109
LM_A_BORROW_STAT 110
LM_A_CHECK_BADDATE 110
LM_A_CHECK_INTERVAL 111
LM_A_CHECKOUT_DATA 137
LM_A_CHECKOUTFILTER_EX 140
LM_A_CHECKOUTFILTERLAST_EX 142
LM_A_CKOUT_INSTALL_LIC 111
LM_A_CRYPT_CASE_SENSITIVE 458
LM_A_DIAGS_ENABLED 144
LM_A_DISABLE_ENV 144
LM_A_DISPLAY_OVERRIDE 144
LM_A_HOST_OVERRIDE 145
LM_A_LF_LIST 113
LM_A_LICENSE_CASE_SENSITIVE 147
LM_A_LICENSE_DEFAULT 114
LM_A_LICENSE_FMT_VER 114
    and lc_cryptstr() 40
LM_A_LINGER 115
LM_A_LKEY_CASE_SENSITIVE 148
LM_A_LKEY_LONG 458
LM_A_LKEY_START_DATE 458
LM_A_LONG_ERRMSG 115
LM_A_MAX_TIMEDIFF 458
LM_A_MT_HEARTBEAT 148
LM_A_MULTIPLE_CHECKOUT_DATA 139
LM_A_NO_MT_HEARBEAT_TIMER 458
LM_A_PATCH_NUMBER 458
LM_A_PERIODIC_CALL 149
LM_A_PERIODIC_COUNT 149
LM_A_PERROR_MSGBOX 116
LM_A_RECONNECT_DONE
    and lc_heartbeat() 62
LM_A_RETRY_CHECKOUT 150
LM_A_RETRY_COUNT 118
    and lc_heartbeat() 62
LM_A_RETRY_INTERVAL 118, 120
LM_A_REVISION 458
LM_A_SETITIMER 459
LM_A_SIGNAL 459
LM_A_SUB_MINOR_REVISION 459
LM_A_TCP_TIMEOUT
    and lc_heartbeat() 63
LM_A_USE_FQDN 121
LM_A_USE_START_DATE 459
LM_A_USER_EXITCALL 122
    and lc_heartbeat() 62
LM_A_USER_OVERRIDE 149, 151
LM_A_USER_RECONNECT 123
    and lc_heartbeat() 62
LM_A_USER_RECONNECT_DONE 124, 125, 126, 127
LM_A_USER_RECONNECT_DONE_EX 124
LM_A_USER_RECONNECT_V1_DATA, client attribute 125
LM_A_USER_RECONNECT_V1_DONE, client attribute 125
LM_A_USER_RECONNECT_V1, client attribute 125
LM_A_VAL_TYPE and lc_set_attr() 76, 100
LM_A_VD_FEATURE_INFO 128
    and lc_get_attr() 57
LM_A_VD_GENERIC_INFO 128
    and lc_get_attr() 57
LM_A_VENDOR_CALLBACK_DATA 152
LM_A_VENDOR_ID_DECLARE 130, 131
LM_A_VERSION 459
LM_A_WINDOWS_MODULE_HANDLE 132
LM_BAD_TZ (-71)

and lc_init() 69
and lc_new_job() 72
LM_BAD_VERSION (-77)
and lc_checkout() 30
LM_BADCODE (-8)
and lc_check_key() 91
and lc_checkout() 30
LM_BADCOMM (-12)
and lc_remove() 99
and lc_userlist() 82
and lc_vsend() 84
LM_BADFILE (-2)
and lc_get_config() 94
LM_BADHANDSHAKE (-33)
and lc_checkout() 30
LM_BADKEYDATA (-44)
and lc_init() 69
and lc_new_job() 72
LM_BADPARAM (-42)
and lc_borrow_return() 89
and lc_check_key() 91
and lc_checkout() 30
and lc_expire_days() 46
and lc_free_hostid() 93
and lc_remove() 99
and lc_set_attr() 77, 101
LM_BADPLATFORM (-48)
and lc_init() 69
and lc_new_job() 72
LM_BADSYSDATE (-88)
and lc_checkout() 30
LM_BADVENDORDATA (-65)
and lc_init() 69
and lc_new_job() 72
LM_BORROW_ERROR (-102)
and lc_borrow_return() 89
LM_BORROW_RETURN_SERVER_ERR (-124)
and lc_borrow_return() 90
LM_BORROW_TOOLONG (-104)
and lc_checkout() 30
LM_BUSYNEWSERV (-23)
and lc_checkout() 30
LM_CANTCONNECT (-15)
and lc_borrow_return() 90
and lc_checkout() 30
and lc_remove() 99
and lc_status() 78
LM_CANTMALLOC (-40)
and l_new_hostid() 88
and lc_feat_list() 47
and lc_init() 69
and lc_new_job() 72
and lc_userlist() 82
LM_CANTREAD (-16)
and lc_remove() 99
and lc_shutdown() 102
and lc_vsend() 84
LM_CANTWRITE (-17)
and lc_borrow_return() 90
and lc_log() 71
and lc_remove() 99
LM_CI_ALL_FEATURES
and lc_checkin() 28
LM_CO_AVAIL_NOWAIT
and lc_auth_data() 34
and lc_checkout() 33
behavior for multiple checkouts 35
combining with LM_CO_NOWAIT 35
LM_CO_LOCALTEST
and lc_checkout() 34
LM_CO_NOWAIT
and lc_checkout() 33
behavior for multiple checkouts 34
combining with LM_CO_AVAIL_NOWAIT 35
lc_status() 79
LM_CO_QUEUE
and lc_checkout() 33
LM_CO_WAIT
and lc_checkout() 33
LM_COMPOSITEID_INIT_ERR (-126)
and lc_init_simple_composite() 96
LM_COMPOSITEID_ITEM_ERR (-127)
and lc_init_simple_composite() 96
LM_CRYPT_DECIMAL 39
LM_CRYPT_FORCE 39
LM_CRYPT_IGNORE_FEATNAME_ERRS 39
LM_CRYPT_ONLY 39
LM_DEFAULT_SEEDS (-91)
and lc_init() 69
and lc_new_job() 72
LM_DUP_DISP 37
LM_DUP_HOST 37
LM_DUP_NONE 37
LM_DUP_SITE 37
LM_DUP_USER 37
LM_DUP_VENDOR 37
and LM_A_CHECKOUT_DATA 138, 139
LM_EXPIRED_KEYS (-50)
and lc_init() 69
and lc_new_job() 72
LM_FEATQUEUE (-35)
and lc_checkout() 30
and lc_status() 79
LM_FUNCNOTAVAIL (-45)
and lc_auth_data() 26
and lc_checkout() 30
and lc_hostid() 64
and lc_set_attr() 77, 101

and lc_shutdown() 102
and lc_userlist() 82
LM_FUTURE_FILE (-90)
and lc_check_key() 91
lm_isres() 82
LM_LIBRARYMISMATCH (-66)
and lc_init() 70
and lc_new_job() 72
LM_LICENSE_FILE
and LM_A_DISABLE_ENV 144
and LM_A_LICENSE_DEFAULT 114
and multiple jobs 73
LM_LOCALFILTER (-73)
and lc_checkout() 30
LM_LONGGONE (-10)
and lc_expire_days() 46
LM_MAXLIMIT (-87)
and lc_checkout() 30
LM_MAXUSERS (-4)
and lc_checkout() 30
LM_NEVERCHECKOUT (-41)
and lc_status() 79
lm_new.o 72
lm_new.obj 72
LM_NO_SERVER_IN_FILE (-13)
and lc_checkout() 31
LM_NOADMINAPI (-78)
and lc_get_attr() 57
and LM_A_VD_*_INFO 128
LM_NOBORROW_SUPP (-68)
and lc_checkout() 31
LM_NOCONFFILE (-1)
and lc_get_config() 94
and lc_set_attr() 77, 101
LM_NOFEATURE (-5)
and lc_auth_data() 26
and lc_borrow_return() 90
and lc_checkout() 31
and lc_feat_list() 47
and lc_get_config() 94
and lc_remove() 99
and lc_userlist() 82
LM_NOKEYDATA (-43)
and lc_init() 70
and lc_new_job() 72
LM_NONETWORK (-62)
and lc_init() 70
and lc_new_job() 72
LM_NOREADLIC (-30)
and lc_get_config() 94
LM_NOSERVSUPP (-18)
and lc_checkout() 31
and lc_get_attr() 57
and lc_vsend() 84, 85
LM_NOSOCKET (-7)
and lc_log() 71
LM_NOSUCHATTR (-32)
and lc_get_attr() 57
and lc_set_attr() 77, 101
LM_NOTLICADMIN (-63)
and lc_remove() 99
and lc_shutdown() 102
LM_OLDVENDORDATA (-72)
and lc_init() 70
and lc_new_job() 72
LM_OLDVER (-21)
and lc_checkout() 31
LM_PLATNOTLIC (-89)
and lc_checkout() 31
LM_REMOVETOOSOON (-64)
and lc_remove() 99
LM_SERVBUSY (-20)
and lc_checkout() 31
LM_SERVNOREADLIC (-61)
and lc_get_config() 94
LM_SOCKETFAIL (-57)
and lc_checkout() 31
LM_USERSQUEUED (-24)
and lc_checkout() 31
LM_VER_BEHAVIOR
and LM_A_LICENSE_CASE_ SENSITIVE 148
lmadmin 492
lmadmin group
lc_remove() 99
lmdown
lc_shutdown() 101
lmgrd 492
automatic reconnection 123
detecting daemon death 111
lmremove
and lingering licenses 474
lmstat
and LM_A_CHECKOUT_DATA 138
lc_userlist() 81
Local Activation 179, 220, 221, 223, 224
local activation 492
local license rights 492
log file
debug 487
report 493
ls_minimum_user_timeout
and lc_heartbeat() 63
ls_show_vendor_def
and LM_A_CHECKOUT_DATA 138
ls_vendor_msg
and lc_vsend() 85
lsvendor.c
and lc_heartbeat() 63

and lc_vsend() 85

M

machineid 492
manual activation 492
Manual Response Generator 492
MAX_VENDOR_CHECKOUT_DATA 137, 139
model, license 491
multiple jobs 73
multiple licenses allocated to a single client 143, 146

N

nCOAvailLastLicCount 36
newlink GlossaryC 487
notr 16

O

options file 492
OVERDRAFT
  and LM_A_VD_FEATURE_INFO 130
  detecting for suites 130

P

package 492
part number 492
permanent license 493
PHY_* 66
preptool 50, 217
product 493
programmatic activation 493
proxy server 207
publisher anchor 223
publisher id 493

Q

quorum 493

R

regular entitlement, See simple entitlement
rehost 493
repair request 493
repair response 493
report log file 493
RESERVE and duplicate grouping 37
responsegen 493
return request 493
return response 493
return transaction 494

S

Secure Sockets Layer (SSL) 210
served license rights 494
server
  license 491
short code 494
short-code activation 494
signature 494
  and lc_check_key() 91
  and LM_BADCODE 30
  and LM_LGEN_VER 114
simple entitlement 494
single-action transaction 494
site license 37
SOAP 205, 210, 211
  proxy server 207
  status of transaction 211
  timeout 206, 214
  transaction status codes 213
SSL Certificate 210
suite 494
suite, detecting OVERDRAFT in 130
supersede 494

T

TCP/IP and LM_A_TCP_TIMEOUT 120
three-server redundancy 494
TIMEOUT
  and lc_heartbeat() 63
  and lc_idle() 68
triad 494
trial anchor 223
trial license 494
trust flags 252
  Host 253
  Restore 253
  Time 253
Trusted Configuration 350, 351, 366, 381, 398, 435
trusted configuration file 494
trusted storage 177, 494
trusted storage configuration 495
type, hostid 490

U

uncounted license
  uncounted 495
Unique Machine Number
  generate 202

unserved license rights 495
unserved licenses 495
upgrade 495
upsell 495

V

Vendor Certificate Generator 495
vendor daemon 495
    automatic reconnection 123
    detecting daemon death 111
vendor data in activation request 360, 376, 388, 412, 430, 440
vendor name 495
VENDOR_LICENSE_FILE
    $HOME/.flexlmrc 108
    and LM_A_LICENSE_DEFAULT 114
    Windows registry 108
vendor-defined hostid 130
verifying license on install 91
VM_GENID 66
VM_UUID 66

W

windback 495
windback detection 495
windback tolerance 495

X

X-Display name 145
XOpenDisplay() 145
XtAppInitialize() 145