Application Note: Using the FlexNet Publisher Sample C# Implementation for Trusted Storage–Based Licensing

May 2023, Revision 00
Version: 11.19.4

FlexNet Publisher Using Sample C# Sharp Impementation 3
- Summary Functionality 3
- Limitations 3
- Packaging 3
- Prerequisites 3
- Supported Platforms 4
- Supported Compiler 4
- Architecture 4
- Support Statement 5

Preparing to Use the C# Toolkit 5
- Download and Install the C# Toolkit 5
- Download, Install, and Build the Native Toolkit 6
- Build Imgract.dll 6
- (Optional) Build the Demo C# Application 7

Understanding the C# Wrapper Classes 7
- Activation API Classes 7
- Flexible API Class 9
- Service Installer Class 9
- Using the C# Wrapper APIs in an Application 9
- Building an Application Activation Utility 10
- Building a FlexEnabled Application 10

Deploying the C# Application 10

Securing the C# Application 11

Appendix A: The Demo C# Application 11
- Activation Functionality 12
- Trial Licensing 13
- Offline Transaction Processing 13
- FlexEnabled Application Functionality 13

Appendix B: Comparison of C# Toolkit APIs with Native Code Toolkit APIs 13
Common Library Functions 13
Application Activation Functions 17
Legal Information 20

FlexNet Publisher Using Sample C# Sharp Impementation

The increasing popularity of the Microsoft .NET platform is providing an impetus for many software publishers to release products based on that platform. The C# programming language provides functionality similar to Java, but is targeted for the .NET platform. To address the growing needs of this space, the FlexNet Publisher Licensing Toolkit now makes available a sample implementation of the Licensing APIs in C#. This Application Note describes the details of this sample C# toolkit.

Summary Functionality

Using the C# toolkit's sample implementation as a guide, you will be able to develop applications that incorporate the following licensing functionality of the FlexNet Publisher Licensing Toolkit:

- A trusted storage-based application activation utility for managing and automating the license lifecycle activities, such as the activation of a license, return of a license to its source, repairing of the trusted storage, trial licensing and querying the contents of a trusted storage.
- A trusted storage-based FlexEnabled application that can enforce a variety of license models.

Since the C# source code is made available as part of this toolkit, you can extend the functionality to customize the wrappers to suit your needs.

Limitations

Note that the following functionality is not supported with the C# toolkit:

- License file-based licensing model (either served or nodelocked)
- A trusted storage-based server activation utility
- Any of the license server components (e.g. the vendor daemon)
- Activation using composite transactions functionality
- Short code transactions
- Use of multiple job handles

Packaging

The FlexNet Publisher C# Toolkit is delivered as a self-contained zip file that can be downloaded from the same location as the other FlexNet Publisher toolkits.

Prerequisites

To be able to use the C# toolkit APIs, you must download and build the FlexNet Publisher Licensing Toolkit for x86 or x64 Windows (`i86_n3` or `x64_n6`).

Supported Platforms

The C# toolkit has been tested with the .NET Framework 4.7 runtime environment.

Refer to the FlexNet Publisher *Release Notes* for versions of currently supported Windows operating systems.

Both 32-bit and 64-bit versions of the above operating systems are supported, with the 64-bit OS running the 32-bit application.

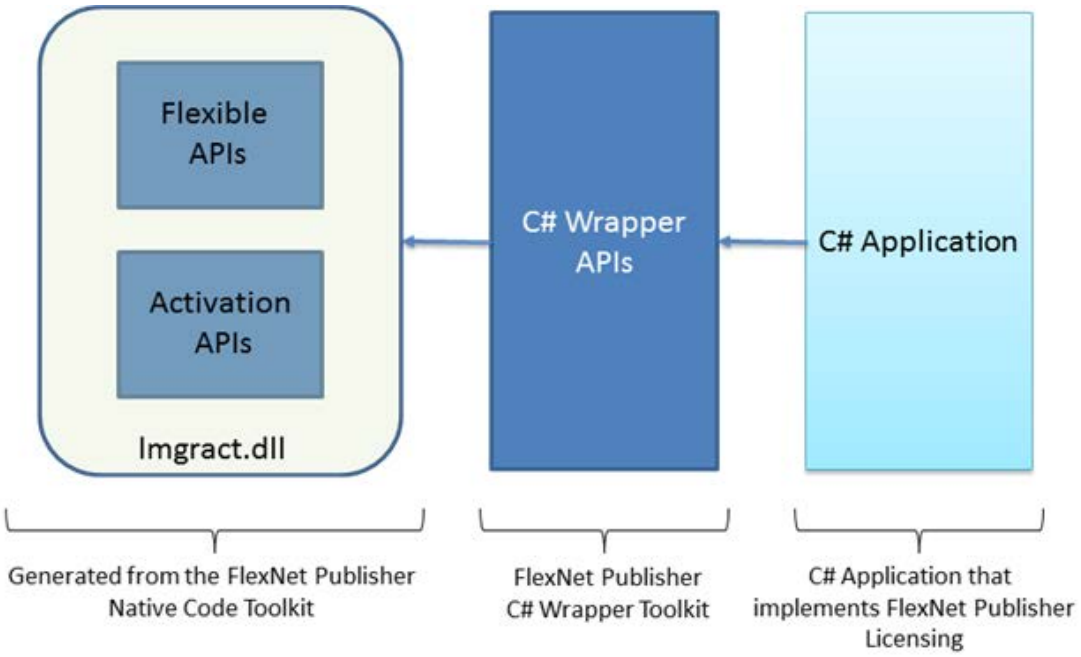
Additionally, this toolkit supports FlexEnabled applications running on virtual machine instances set up in Microsoft Hyper-V, VMware, or Citrix XenServer environments. See the current FlexNet Publisher *Release Notes* for details about the virtualization-platform versions supported in this release.

Supported Compiler

The sample implementation has most recently been tested with Microsoft Visual Studio 2017 (15.7).

Architecture

The C# toolkit provides a C#-based wrapper interface around the native ("unmanaged") code interface of FlexNet Publisher Activation APIs and the Flexible APIs. You will then integrate the C# Wrapper APIs into your C#-based ("managed") application. The high-level architecture of the C# wrapper interface is shown in the following diagram:



Support Statement

Revenera provides technical support for all the components included in the C# toolkit. Technical support for any extensions performed to the source code provided in the toolkit is subject to that issue being reproduced in the standard components of the toolkit.

Preparing to Use the C# Toolkit

Before you can use the C# toolkit APIs, you must perform the following steps:

1. Download and install the FlexNet Publisher C# Toolkit.
2. Download, install, and build the native code FlexNet Publisher Licensing Toolkit for x86 Windows (i86_n3), referred to as the *native toolkit*.
3. Build the `lmagract.dll` from the native toolkit using the sample `makefile` from the C# toolkit.
4. (Optional) Build the demo C# application that provides an example usage of the C# toolkit API calls.

These steps are explained in detail below.

Download and Install the C# Toolkit

The C# toolkit is packaged as a zip file and is downloadable from the same area of the Revenera's web site from which you download the native toolkit. The downloaded file follows the naming convention **fnpcsharpwrapper-i86_n3-<*version*>.zip**. The <*version*> field contains the version number of the FlexNet Publisher toolkit, such as *11.16.1*.

After you download the C# toolkit, unzip the contents into a folder. The folder where you extract the C# toolkit will be referred to as C#_INSTALL_DIR. A brief explanation of the folder contents is as below:

- `C#_INSTALL_DIR\FlexNetActivation`: This folder contains the following main components:
  - Under the `C#_INSTALL_DIR\FlexNetActivation\FlexNet` folder, the source code for the sample C# wrapper APIs.
  - The Visual Studio project file and solution.
  - Under the `C#_INSTALL_DIR\FlexNetActivation\demoApp` folder, the source code for the demo C# application.
  - Under the `C#_INSTALL_DIR\FlexNetActivation\Properties` folder, the assembly and resource information for the C# project.
- `C#_INSTALL_DIR\lmgract`: This folder contains files necessary to build the `lmgract.dll`. Locate the following files in the `C#_INSTALL_DIR\lmgract` folder:
  - `lmgract.c`: A C program wrapper around the Flexible APIs that provides an easy-to-use encapsulation for the C# wrappers.
  - `lmgract.def`: The module definition file required by Visual Studio compiler to create a DLL.

- `lmgract.xml`: Configuration file to be used by the `preptool`. The `preptool` utility is included as part of the native toolkit and is used to create a publisher- and product-specific activation library component for the applications that use it.
- `makefile.csharp`: An example `makefile` that can be run from the command-line on the native toolkit to generate the `lmgract.dll` file.

Download, Install, and Build the Native Toolkit

Download and install the FlexNet Publisher Licensing Toolkit for the Microsoft Windows 32-bit (x86) platform. Follow the instructions from the documentation of the native toolkit to generate the file `lm_new.obj`. The folder where you extract the contents of the native toolkit will be referred to as `FNP_INSTALL_DIR`.

Build lmgract.dll

Before you proceed, make sure you have successfully built the `lm_new.obj` from the native toolkit. Follow these instructions to build `lmgract.dll` from the native toolkit. These instructions are to be executed from a Visual Studio command prompt.

Task ***To build lmgract.dll from the native toolkit***

1. Change directory to the `FNP_INSTALL_DIR\i86_n3` folder.
2. Copy the file `makefile.csharp` from `C#_INSTALL_DIR\lmgract` to `FNP_INSTALL_DIR\i86_n3`.
3. Edit the `makefile.csharp` file to specify the location of the `C#_INSTALL_DIR` by modifying the `makefile` variable `CSHARPDIR`.
4. Issue the following command to build `lmgract.dll`:

   ```
   nmake -f makefile.csharp csharp-all
   ```

5. The above command will not only build `lmgract.dll`, but will also copy the DLLs that you may need to redistribute with your application to your customers. The copied files will be located at the directory `C#_INSTALL_DIR\FlexNetActivation\demoApp\bin\Release`.
6. (Optional) To build a debug version of the `lmgract.dll`, you can modify the `nmake` command as below:

   ```
   nmake -f makefile.csharp csharp-all DEBUG=1
   ```

   The above command will build and copy the DLLs to `C#_INSTALL_DIR\FlexNetActivation\demoApp\bin\Debug`.

Note ▪ *Note: By default, the TRL-only libraries are linked with the FlexNet Publisher toolkits. To use the non-TRL- based libraries, modify the toolkit `makefile` with `lmgr.lib` while building the kit.*

(Optional) Build the Demo C# Application

A sample C# application is included in the C# toolkit that demonstrates the application and usage of the C# Wrapper APIs.

Task ***To build the demo C# application***

1. Launch Visual Studio.
2. Open the solution FlexNetActivation, located under the folder `C#_INSTALL_DIR\FlexNetActivation`.
3. Build the solution FlexNetActivation (default is the Debug configuration). It generates the executable `FlexNetActivation.exe` in the folder `C#_INSTALL_DIR\FlexNetActivation\demoApp\bin\Debug`.

Refer to Appendix A: The Demo C# Application on page 11 for instructions on how to run the demo C# application.

Understanding the C# Wrapper Classes

The C# toolkit encapsulates a subset of the Activation APIs as well as the Flexible APIs. It also includes a utility class that encapsulates the installation and uninstallation of the FlexNet Licensing Service and another class that provides a higher level of abstraction for an application activation utility.

Refer to Appendix B: Comparison of C# Toolkit APIs with Native Code Toolkit APIs on page 13 for more details.

Activation API Classes

The trusted storage activation APIs are broadly divided into two categories:

- The **Application Activation** functions are used by an application activation utility to manage license rights for a FlexEnabled application. Note that the methods from the C# toolkit cannot be used to create a server activation utility. The functions in this category are used to initiate activation transactions of the following types:
  - Activation
  - Return
  - Repair
- The **Common Library** functions include activation APIs used when accessing trusted storage. They also include functions required to implement local activations (such as ASRs). The basic record in trusted storage is a fulfillment record. A fulfillment record contains details of license rights. The Common Library functions include APIs 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 C# toolkit further organizes the above two categories of the FlexNet Publisher APIs into distinct classes for ease of use as described below:

C# Class Name	Description
FlexObjects	This is an abstract base class that encapsulates various FlexNet handles returned from FlexNet Publisher APIs.
ApplicationActivation	This is a wrapper class that encapsulates a subset of the Application Activation category of functions that have the prefix **flxActAppActivation**. For example, this class includes methods that wrap the activation APIs such as **flxActShortCodeCreate()** or **flxActShortCodeCreateFromBuffer()** and **flxActAppActivationEntitlementIdSet()**.
ActivationReturn	This is a wrapper class that encapsulates a subset of the Application Activation category of functions that deal with activation return functionality and have the prefix **flxActAppReturn**. For example, this class includes methods that wrap the activation APIs such as **flxActShortCodeCreate()** and **flxActAppReturnVendorDataSet()**.
ActivationRepair	This is a wrapper class that encapsulates a subset of the Application Activation category of functions that deal with activation repair functionality and have the prefix **flxActAppRepair**. For example, this class includes methods that wrap the activation APIs such as **flxActShortCodeCreate()** and **flxActAppRepairVendorDataSet()**.
CommonLibrary	This is a wrapper class that encapsulates a subset of the Common Library category of functions that have the prefix **flxActCommonLibrary** and **flxActCommonHandle**. For example, this class includes methods that wrap the common library APIs such as **flxActCommonLibraryInit()** and **flxActCommonHandleOpen()**.
License Specification	This is a wrapper class that encapsulates a subset of the Common Library category of functions that deal with license specifications and have the prefix **flxActCommonLicSpc**. For example, this class includes methods that wrap the common library APIs such as **flxActCommonLicSpcCreate()** and **flxActCommonLicSpcPopulateFromTS()**.

C# Class Name	Description
ProductLicenseSpecification	This is a wrapper class that encapsulates a subset of the Common Library category of functions that deal with the product license specifications and have the prefix **flxActCommonProdLicSpc**. For example, this class includes methods that wrap the common library APIs such as **flxActCommonProdLicSpcFulfillmentIdGet()** and **flxActCommonProdLicSpcExpDateGet()**.
TrialLicensing	This is a wrapper class that encapsulates a subset of the Common Library category of functions that deal with trial licensing (such as loading the license rights from an ASR) and have the prefix **flxActCommonLicSpcAddASR**. For example, this class includes methods that wrap the common library APIs such as **flxActCommonLicSpcAddASRs()** and **flxActCommonLicSpcAddASRFromBuffer()**.
VirtualizationLibrary	This is a wrapper class that encapsulates the Common Library category of FlexNet Publisher APIs that deal with virtualization. For example, this class includes methods that wrap **flxActCommonVirtualStatusGet** API.

Flexible API Class

The `Feature` class from the C# toolkit encapsulates the essential Flexible APIs that can perform the following tasks in a FlexEnabled application:

- Activation library initialization and cleanup
- Job handle creation and destruction
- License feature check-out and check-in
- Error string display

Service Installer Class

The `ActivationServiceInstaller` is an internal class from the C# toolkit that encapsulates native method APIs to perform FlexNet Licensing Service installation and uninstallation.

Using the C# Wrapper APIs in an Application

You can invoke the C# Wrapper APIs in your application to build either of the following:

- An application activation utility
- A FlexEnabled application

Building an Application Activation Utility

Use the reference implementation of the C# class `ActivationUtility.cs` to create an activation utility that follows the general flow as below:

1. Instantiate the `ActivationUtility` class. This will perform the following tasks:
   - Install the FlexNet Licensing Service
   - Initialize the activation library
2. Set the `ActivationUtility.CommunicationType` depending on the user choice or your deployment model. The communication type determines whether the license rights are being sourced from an operations server or an enterprise license server.
3. Perform the desired action as chosen by the user: Activate new rights, return existing rights, repair the trusted storage, load an ASR etc.
4. You can also query the license rights held in the trusted storage to display them to the user.
5. Clean up the objects upon closing with **ActivationUtility.Dispose()**.

Refer to the source code `C#_INSTALL_DIR\FlexNetActivation\FlexNet\ActivationUtility.cs` for more details. You can also refer to the demo C# application code in `C#_INSTALL_DIR\FlexNetActivation\demoApp\Form1.cs` for a sample implementation.

Building a FlexEnabled Application

You should modify your C# application at the appropriate place to enforce licensing to make it a FlexEnabled Application. The process of integrating the C# toolkit APIs into your application follows the general flow described below:

1. Use the following methods from the `Feature` class in sequence to initialize the licensing library:
   - **Feature.InitializeHandle()**
   - **Feature.CreateJob()**
   - **Feature.SetAttribute()**
2. Check out or check in a license feature using **Feature.CheckOut()** and **Feature.CheckIn()** methods. Process the error messages with **Feature.GetErrorString()**.
3. Clean up the objects with **Feature.FreeJob()** and **Feature.CleanupHandle()** upon program exit.

Refer to the source code `C#_INSTALL_DIR\FlexNetActivation\FlexNet\Feature.cs` for more details. You can also refer to the demo C# application code in `C#_INSTALL_DIR\FlexNetActivation\demoApp\Form1.cs` for a sample implementation.

Deploying the C# Application

The package of the C# application that you deploy to your customers must include the following DLLs:

- `lmgract.dll`

- `lmgract_libFNP.dll`
- `FNP_Act_Installer.dll`
- `FnpCommsSoap.dll`

Securing the C# Application

Note that using a DLL that contains all the licensing functionality is susceptible to security hacks leading to license leakage. As such, a C# FlexEnabled application using the `lmgract.dll` as described in this document is more suitable to be deployed in a trusted environment. The following security best practices can help lower (but will not eliminate) the vulnerability of the application to DLL spoofing:

- Include several calls to the Flexible API licensing functions within a single exported function from the DLL
- Use external function names that do not reflect the licensing functionality. This makes it harder for the hackers to identify that licensing functions are being used.
- Include other (non-licensing) functionality used by the application in the DLL. This makes it harder for the DLL to be replaced without loss of major functionality.

Appendix A: The Demo C# Application

The demo C# application demonstrates the capabilities of the C# toolkit. The demo application implements the functionality of both an application activation utility and a FlexEnabled application in a single executable. In practice, it is highly recommended that you separate out these two functionalities for maximum flexibility. Launch the application and explore the various functionalities as shown in the picture below.

The title of the demo application window indicates whether you are running the application in a virtual or physical environment. The picture below shows that the application is running in a physical environment.

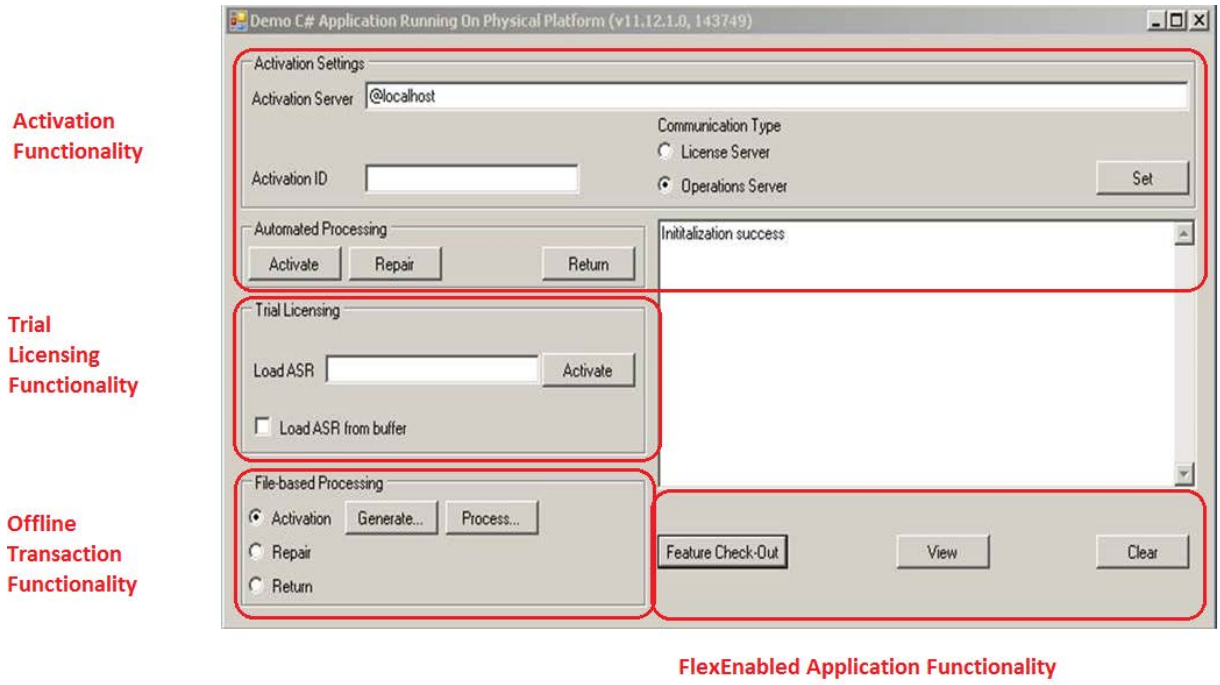


Figure 1: GUI of the demo C# application

Note ▪ *Note: Before you run the Demo C# Application, make sure you have license entitlements available in your activation server, license server and/or a local ASR.*

Activation Functionality

Task ***To explore the online (or real-time) activation functionality from the Form GUI***

1. Choose the Communication Type option—either **License Server** or **Operations Server**.
2. Fill in the **Activation Server** and **Activation ID** fields.
3. Click **Set** to apply the settings.
4. You can click **Activate**, **Repair**, or **Return** to explore the different functionalities.

   Note ▪ *Note: The Repair and Return functionality as demonstrated in this application does not let you choose the fulfillment record if you have a trusted-storage file with multiple fulfillment records.*

Trial Licensing

Task ***To explore Trial Licensing functionality using an ASR***

1. In the **Load ASR** field, enter the file path to an ASR file, and click **Activate**.
2. Optionally, select the **Load ASR from buffer** checkbox, and click **Activate**. This demonstrates loading an ASR that is built into the application rather than being provided as an external file.

Offline Transaction Processing

You can perform offline transactions using the controls in the **File-based Processing** group of the GUI. Choose the transaction type (**Activation**, **Repair**, or **Return**); and click **Generate...** to specify a filename where a request transaction will be saved. You can similarly process the response from an activation server using the **Process...** button.

FlexEnabled Application Functionality

You can perform a license check-out and check-in by clicking **Feature Check-Out**. In the **Feature Check-Out /Check-In** window, specify a feature name in the **Feature** textbox and then click **Check Out** or **Check In**. You can also view your current entitlements by clicking **View**. Click **Clear** to clear out the contents of the status window.

Appendix B: Comparison of C# Toolkit APIs with Native Code Toolkit APIs

This section contains a list of C-based activation APIs that have a corresponding implementation in the C# toolkit.

Common Library Functions

Following are the set of common library category of functions that are implemented as part of the C# toolkit

Native Code Toolkit API	Correspond C# Toolkit API
Class Name: **CommonLibrary**	
flxActCommonGetFRCountFromServerTS Retrieves fulfillment-record counts from remote server-side trusted storage.	**GetFRCountFromServerTS**

Native Code Toolkit API	Correspond C# Toolkit API
flxActCommonHandleClose Closes an activation handle.	**CloseHandle**
flxActCommonHandleGetCommType Gets the type of communication used for the connection to the activation server.	**GetCommunicationType**
flxActCommonHandleGetError Get the latest error code for the last operation performed using the specified handle.	**GetLastError**
flxActCommonHandleGetLastOpsErrorString Gets text that explains the last error code returned by the FlexNet Operations server.	**GetLastOperationsErrorDescription**
flxActCommonHandleGetRemoteServer Returns the address of the activation server to which activation transactions will be sent.	**GetActivationServer**
flxActCommonHandleOpen Creates an Activation API handle.	**OpenHandle**
flxActCommonHandleSetCommType Sets the type of communication for the connection to the activation server used for transactions.	**SetCommunicationType**
flxActCommonHandleSetProxyDetails Sets the proxy server details when using a SOAP connection.	**SetProxy**
flxActCommonHandleSetRemoteServer Stores the address of the activation server to be used for subsequent activation transactions that require a server.	**SetActivationServer**
flxActCommonHandleSetStatusCallback Sets a callback that reports changes in status of transactions in progress on a SOAP connection.	**SetStatusCallback**
flxActCommonLibraryCleanup Release the activation library	**Cleanup**
flxActCommonLibraryInit Initialization required using the activation library and the Activation API.	**InitializeLibrary**
Class Name: **License Specification**	

Native Code Toolkit API	Correspond C# Toolkit API
flxActCommonLicSpcCreate Creates a license specification.	**LicenseSpecification Constructor**
flxActCommonLicSpcDelete Deletes the specified license specification.	**LicenseSpecification Desctructor**
flxActCommonLicSpcGet Get the fulfillment record at the specified index from the license specification.	**ProductLicenseSpecification**
flxActCommonLicSpcGetNumberProducts Returns the number of fulfillment records in the license specification.	**GetNumberProducts**
flxActCommonLicSpcPopulateFromTS Loads valid fulfillment records from trusted storage into the license specification.	**PopulateFromTrustedStorage**
flxActCommonLicSpcPopulateAllFromTS Loads all fulfillment records; expired or non-expired; valid or not valid; from trusted storage into the specified license specification.	**PopulateAllFromTrustedStorage**
flxActCommonLicSpcPopulateFromServerTS Retrieves trusted fulfillment records (expired and non-expired) from remote server-side trusted storage into the specified license specification structure.	**PopulateFromServerTrustedStorage**
flxActCommonLicSpcPopulateAllFromServerTS Retrieves all (expired, non-expired, valid or not valid) fulfillment records from remote server-side trusted storage into the specified license specification structure.	**PopulateAllFromServerTrustedStorage**
Class Name: **TrialLicensing**	
flxActCommonLicSpcAddASRFromBuffer Adds the fulfillment record from the supplied ASR to trusted storage.	**AddASRFromBuffer**
flxActCommonLicSpcAddASRs Adds the fulfillment records from a valid ASR file to trusted storage.	**AddASR**

Native Code Toolkit API	Correspond C# Toolkit API
flxActCommonRepairLocalTrustedStorage Repairs broken segments of trusted storage so that 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.	**RepairLocalTrustedStorage**
Class Name: **ProductLicenseSpecification**	
flxActCommonProdLicSpcEntitlementIdGet Get the entitlement ID from the fulfillment record in the supplied product license specification.	**GetEntitlementId**
flxActCommonProdLicSpcExpDateGet Get the expiration date from the fulfillment record in the supplied product license specification.	**GetExpirationDate**
flxActCommonProdLicSpcFeatureLineGet Get the feature definition lines from the fulfillment record in the supplied product license specification.	**GetFeatureLines**
flxActCommonProdLicSpcFulfillmentIdGet Get the fulfillment ID from the fulfillment record in the supplied product license specification.	**GetFulfillmentId**
flxActCommonProdLicSpcFulfillmentTypeGet Get the fulfillment type from the fulfillment record in the supplied product license specification.	**GetFulfillmentType**
flxActCommonProdLicSpcIsDisabled Returns the status of the specified fulfillment record.	**IsDisabled**
flxActCommonProdLicSpcPopulateFRFromServerTS Retrieves detailed information for the fulfillment record obtained from either: ● flxActCommonLicSpcPopulateFromServerTS ● flxActCommonLicSpcPopulateAllFromServerTS PopulateFRFromServerTS	**PopulateFRFromServerTS**
flxActCommonProdLicSpcProductIdGet Get the product ID from the fulfillment record in the supplied product license specification.	**GetProductId**
flxActCommonProdLicSpcSuiteIdGet Get the suite ID from the fulfillment record in the supplied product license specification.	**GetSuiteId**

Native Code Toolkit API	Correspond C# Toolkit API
flxActCommonProdLicSpcTrustFlagsGet Get the trust status of the fulfillment record in the supplied product license specification.	**GetTrustFlags**
Class Name: **VirtualizationLibrary**	
flxActCommonVirtualStatusGet Get the status of the machine running the FlexEnabled application: virtual or physical (non-virtual).	**GetVirtualMachineStatus**

Application Activation Functions

Following are the set of application activation category of functions that are implemented as part of the C# toolkit.

Native Code Toolkit API	Correspond C# Toolkit API
Class Name: **ApplicationActivation**	
flxActShortCodeCreate or **flxActShortCodeCreateFromBuffer** Creates an application activation object.	**ApplicationActivation Constructor**
flxActShortCodeDestroy Deletes an application activation object.	**ApplicationActivation Destructor**
flxActAppActivationDurationGet Gets the license rights duration value from the activation object.	**GetDuration**
flxActAppActivationDurationSet Writes the specified duration into the activation object.	**SetDuration**
flxActAppActivationEntitlementIdGet Gets the entitlement ID from the activation object.	**GetActivationId**
flxActAppActivationEntitlementIdSet Writes the specified entitlement ID into the activation object.	**SetActivationId**
flxActAppActivationProductIdGet Gets the ProductID from the specified activation object.	**GetProductId**

Native Code Toolkit API	Correspond C# Toolkit API
flxActAppActivationProductIdSet Writes the specified ProductID into the activation object.	**SetProductId**
flxActAppActivationReqSet Creates an activation request as a signed XML document for use in manual return transactions.	**CreateRequest**
flxActShortCodeProcessResponse 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.	**ProcessResponse**
flxActAppActivationSend Sends an activation request to a remote activation server and processes the resulting response.	**SendActivationRequest**
flxActAppActivationExpDateGet Gets the license rights expiration date from the activation object.	**GetExpirationDate**
flxActAppActivationExpDateSet Writes the specified expiration date into the activation object.	**SetExpirationDate**
flxActAppActivationVendorDataGet Gets the vendor data for the specified key from the activation object	**GetVendorData**
flxActAppActivationVendorDataSet Writes the specified vendor data into the activation object.	**SetVendorData**
Class Name: **ActivationRepair**	
flxActShortCodeCreate Creates an application repair object. This object is used when creating a repair request and processing a repair response.	**ActivationRepair Constructor**
flxActShortCodeDestroy Deletes an application repair object.	**ActivationRepair Destructor**
flxActAppRepairProdLicSpcSet Sets the fulfillment record to be repaired into the return object.	**SetProductLicenseSpecification**

Native Code Toolkit API	Correspond C# Toolkit API
flxActShortCodeProcessResponse Processes 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.	**ProcessResponse**
flxActAppRepairSend Sends a repair request to a remote activation server and automatically processes the response.	**SendActivationRepair**
flxActAppRepairVendorDataSet Writes the specified vendor data into the repair object.	**SetVendorData**
flxActAppRepairReqSet Creates a repair request as a signed XML document for use in manual repair transactions	**CreateRequest**
Class Name: **ActivationReturn**	
flxActShortCodeCreate Creates an application return object. This object is used when creating a return request and processing a return Response.	**ActivationReturn Constructor**
flxActShortCodeDestroy Deletes an application return object.	**ActivationReturn Destructor**
flxActAppReturnProdLicSpcSet Sets the fulfillment record to be returned into the return object.	**SetProductLicenseSpecification**
flxActAppReturnReqSet Creates a return request as a signed XML document for use in manual return transactions.	**CreateRequest**
flxActShortCodeProcessResponse Processes 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.	**ProcessResponse**
flxActAppReturnSend Sends a return request to a remote activation server and automatically processes the response.	**SendActivationReturn**

Legal Information

Copyright Notice

Copyright © 2023 Flexera Software

This publication contains proprietary and confidential information and creative works owned by Flexera Software 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 Software is strictly prohibited. Except where expressly provided by Flexera Software in writing, possession of this publication shall not be construed to confer any license or rights under any Flexera Software intellectual property rights, whether by estoppel, implication, or otherwise.

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

Intellectual Property

For a list of trademarks and patents that are owned by Flexera Software, see https://www.revenera.com/legal/intellectual-property.html. All other brand and product names mentioned in Flexera Software 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.