

Downloadable Content Sample

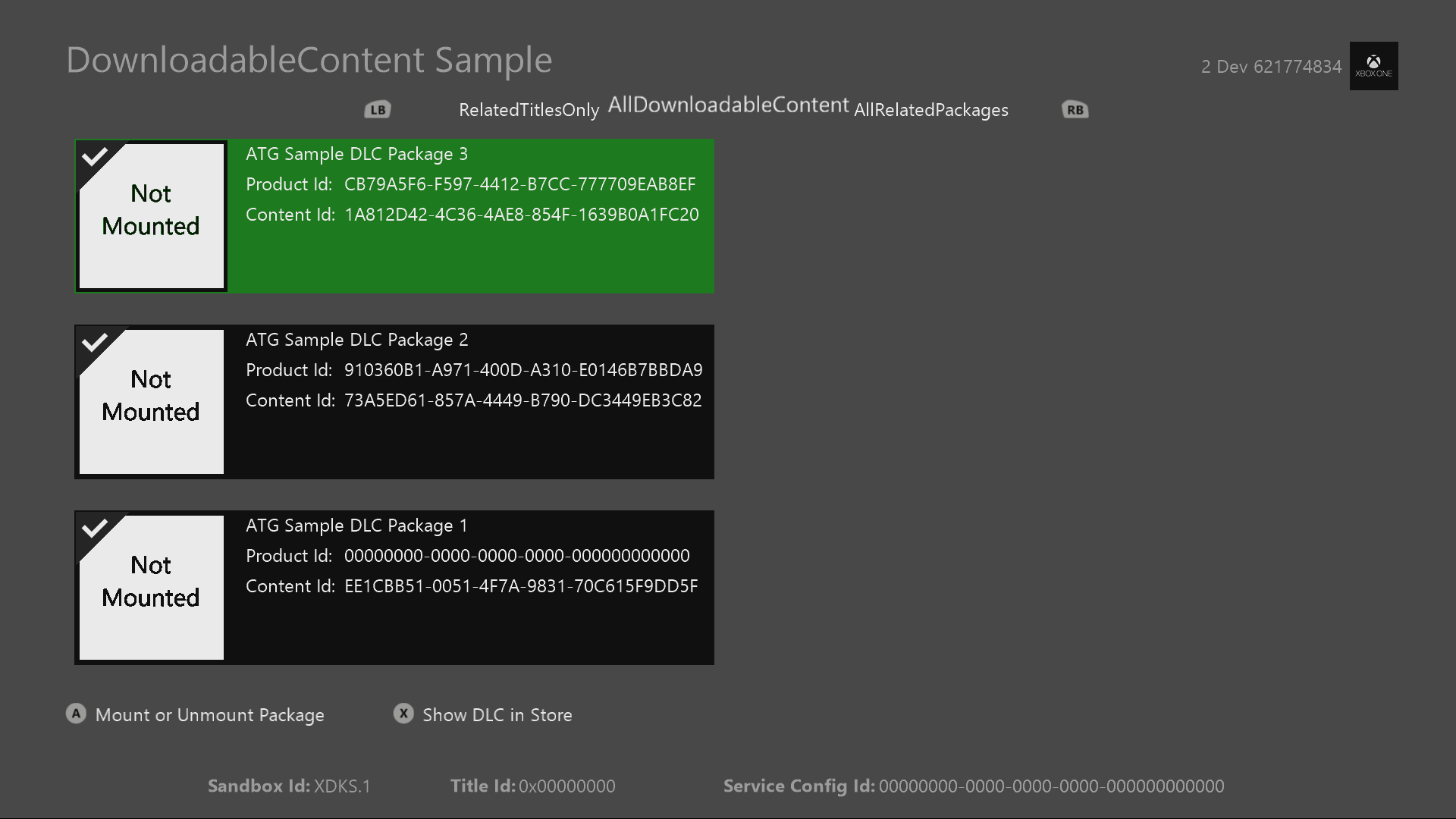
*\* This sample is compatible with the March 2016 Xbox One XDK.*

# Description

This sample demonstrates how to implement enumeration and loading of downloadable content within your Xbox One title.

# Using the sample

This sample is configured to work in the XDKS.1 sandbox.



|  |  |
| --- | --- |
| Action | Gamepad |
| Select DLC Package | D-Pad Up and Down |
| Mount or Unmount Package | A button |
| Go to Marketplace for DLC products | X Button |
| Sign-In | Menu Button |
| Exit | View button |

# Implementation notes

In this version of the XDK, the sample also requires your Xbox One console to use the XDKS.1 **SandboxID**. To switch to this **SandboxID**, in the Xbox One XDK command prompt, use:

xbconfig sandboxid=XDKS.1

# Creating & Deploying Downloadable Content Packages

To create a content package to load as DLC within your title place the raw files in the directory structure as they should be packaged and then run the following tool from an elevated Xbox One XDK Command Prompt:

makepkg.exe pack /lt /v /f "./Chunks.xml" /d ./[RawFilesFolder] /pd ./[DestinationOutputDirectory]

You can find an example of simple Chunks.xml, appxmanifest.xml, and how to create the package in the SamplePackages.zip file provided with the sample. The appxmanifest.xml should be included in the directory of your loose files and updated appropriately for your content. Once you have created the package, you can deploy them to the console with the following from the same command prompt:

xbapp install [DestinationOutputDirectory]\[package name]

When you are ready to publish the content to the marketplace for in-game download testing you will want to include the /l tag instead of /lt in the makepkg command to create a licensed and encrypted version. Also contact your developer account manager for the required submission forms and process. Packages created with /l will not be able to be installed manually with xbapp install unless the package has a ContentID that matches a ContentID already propped and ingested into the Marketplae Catalog. For this reason we recommend using /lt for test content before propping it to the Marketplace.

Debugging and enumerating DLC:  
For enumeration to work properly with your title and DLC packages, the package must have a matching **AllowedProductID** in the appxmaifest.xml to the ProductID of the title. When debugging through Visual Studio or deploying the raw files to a devkit, your title will have a default ProductID of all 0’s. To enable the ability to debug your title and still enumerate DLC set with the actual title in AllowedProductID, add a **RelatedProduct** definition to your Package.appxmanifest of your title’s real ProductID. This marks your title’s ProductID and all of the DLC tied to it as related to the debug version of your title. If you then enumerate DLC with the **AllDownloadableContentPackages** flag, you will get results of your real title’s DLC and not just packages tied to a ProductID of all 0’s. Here is an example of how to define a RelatedProduct and is also in the sample’s Package.appxmanifest for reference (the Id provided is the sample’s ProductID:

<Applications>

…

</Applications>

<Extensions>

<mx:PackageExtension Category="xbox.store">

<mx:XboxStore>

<mx:RelatedProducts>

<mx:RelatedProduct Id="eabe6ccb-48f2-4cd2-832c-1d9753bd707b" />

</mx:RelatedProducts>

</mx:XboxStore>

</mx:PackageExtension>

</Extensions>

# Enumerating and Mounting DLC Packages

To enumerate the locally installed DLC packages use the following API:

IVectorView< IDownloadableContentPackage^ >^

IDownloadableContentPackageManager::FindPackages(InstalledPackagesFilter *filter*)

This will provide to you a vector of objects defining the currently installed DLC packages matching the filter passed in. If you operate on one of the returned objects, your title will need to keep a reference to that specific object as any package operations with that object are not synchronized between other enumeration results for the same package. Example: If you mount package 1, the **IDownloadableContentPackage** for the same package returned in a subsequent enumeration call will not reflect that the package is currently mounted.

To mount a DLC package and access the package’s files use the following API:

Platform::String^ IDownloadableContentPackage::Mount()

This will first verify that the package is licensed to be mounted and if the package is licensed within the current scope of users or console, it will mount the package to a virtual drive. This drive location is returned as a string and can then be used to create a path and open the contents of the package. Once a package is mounted, the system will obtain and keep some file handles open to prevent the package from being deleted off the console. Mounted packages also remain mounted to the same virtual drives when the title is in constrained or suspended mode. Only when the title is closed / terminated, or **Unmount** is called are the mounted packages released.

NOTE:   
There is a limit of how many DLC packages can be mounted at one time across the system. Currently we recommend that an ERA title mount no more than 35 packages at a time. You should be releasing packages when they are not in use to free up slots for more packages. This number is a recommendation at this time and is subject to change. If this is an issue for your title, please talk to your Developer Account Manager.

If the scope of the license changes and your title desires to unload the DLC currently mounted, use the following API:

void IDownloadableContentPackage::Unmount()

This will unmount the package from the virtual drive and release any handles that the system has on the files. Your title is still expected to release handles of any files opened by your title to the package before calling **Unmount**. Once a package has been unmounted, it can be deleted from the user’s console.

When a new package is installed on the system a notification is sent to all apps who have subscribed to the **DownloadableContentPackageManager::DownloadbleContentPackageInstallCompleted** event handler. When titles see this event, they should use **FindPackages** again to discover if there are new packages within their scope. Currently this event will fire twice when a package is installed so you will get two event notifications per package. This was needed for expansion of multiple install packages in future Marketplace scenarios. This sample provides an example of how to re-enumerate for packages with this event as well as proper handling of mounted and unmounted packages.

# Licensing of DLC packages

Your title can validate the current license status for a package with this API:

bool IDownloadableContentPackage::CheckLicense ( bool \*isTrial )

This API will return a boolean value if this package has a license allowing it to be mounted given the current scope of signed-in users or console entitlements. You can also verify if the license available for this package is a preview license by passing in a Boolean and checking its returned value.

The license for a package may change when users sign-in or out on different consoles or purchase an entitlement within the Xbox One Marketplace for the DLC during a game session. If the package had a valid license and is mounted, but now does not have a valid license it is up to the title on when and if it would be appropriate to unmount the DLC package. For example: User 1 owns a DLC Map Pack and is playing with User 2 on User 2’s console. While playing one of the new maps in the DLC, User 1 signs out. The license for the package on the console has now changed to an invalid license as User 2 does not own the content and the console does not have an entitlement to it. The game could continue to allow User 2 to play with the DLC until the game session ends, or the round is over and then unmounts the DLC. The game could also try and upsell user 2 to continue playing the DLC by buying the DLC for their own account and point them at the Product in the Xbox One Store.

A title can create a **TypedEventHandler** for the **LicenseTerminated** event and add it to the **LicenseTerminated** property of the package’s object. This event is fired when a mounted package’s license goes invalid and is no longer within the ownership scope when it was originally mounted. This event does not fire if a package is unmounted and it should be removed from the **LicenseTerminated** property before the package object is released from the title’s scope to avoid memory leaks. This sample demonstrates how to properly subscribe, manage, and remove this event when mounting and unmounting a package.

# Update history

**Initial Release:** August 2017

# Privacy Statement

When compiling and running a sample, the file name of the sample executable will be sent to Microsoft to help track sample usage. To opt-out of this data collection, you can remove the block of code in Main.cpp labeled “Sample Usage Telemetry”.

For more information about Microsoft’s privacy policies in general, see the [Microsoft Privacy Statement](https://privacy.microsoft.com/en-us/privacystatement/).