

Modio C# SDK

4.1.0

Generated by Doxygen 1.9.5

1 LICENSE	1
2 Unity V1 Upgrade Guide	3
2.1 Why Upgrade to the New Plugin?	3
2.2 Core Differences	3
2.2.1 Interface	3
2.2.2 Callbacks	3
2.3 Initializing the plugin	4
2.4 Getting and Synchronizing Subscription Data	4
2.5 Listing the User's Installed Mods	5
2.6 Downloading, Updating and Uninstalling Mods	6
2.7 Authenticating a user	7
2.7.1 Email Authentication	7
2.7.2 Third party Authentication	7
2.8 Need more help?	8
3 README	9
3.1 mod.io Unity Plugin v2023.6.1	9
3.1.1 Watch the video tutorial	9
3.1.2 Platform Support	9
3.1.2.1 Git Repository or .unitypackage	9
3.1.3 Getting started	10
3.1.4 Setting up the Browser UI	10
3.1.5 Authentication	10
3.1.6 Usage	10
3.1.6.1 Initialise the plugin	11
3.1.6.2 Get the user's installed mods	11
3.1.6.3 Enable automatic mod downloads and installs	11
3.1.6.4 Authenticate a user	11
3.1.6.5 Get Mod profiles from the mod.io server	12
3.1.7 Submitting mods	12
3.1.7.1 Adding a mod	12
3.1.7.2 Loading mods	13
3.1.8 Dependencies	13
3.1.9 Benefits	13
3.1.10 Game studios and Publishers	14
3.1.11 Contributions Welcome	14
3.1.12 Other Repositories	14
4 Namespace Index	15
4.1 Package List	15
5 Hierarchical Index	17
5.1 Class Hierarchy	17

6 Class Index	19
6.1 Class List	19
7 Namespace Documentation	23
7.1 ModIO Namespace Reference	23
7.1.1 Enumeration Type Documentation	25
7.1.1.1 SortModsBy	25
7.1.1.2 SubscribedModStatus	25
7.1.1.3 UserPortal	26
7.2 ModIO.Implementation Namespace Reference	26
7.3 ModIO.Implementation.API Namespace Reference	26
7.4 ModIO.Implementation.API.Objects Namespace Reference	26
7.5 ModIO.Util Namespace Reference	26
7.6 Plugins Namespace Reference	27
7.7 Plugins.mod Namespace Reference	27
7.8 Plugins.mod.io Namespace Reference	27
7.9 Plugins.mod.io.Runtime Namespace Reference	27
7.10 Plugins.mod.io.Runtime.Utility Namespace Reference	27
8 Class Documentation	29
8.1 ModIO.AuthenticationServiceProviderExtensions Class Reference	29
8.2 ModIO.BuildSettings Class Reference	29
8.2.1 Detailed Description	30
8.3 CommentDetails Class Reference	30
8.4 ModIO.CommentPage Struct Reference	30
8.4.1 Detailed Description	31
8.4.2 Member Data Documentation	31
8.4.2.1 CommentObjects	31
8.4.2.2 totalSearchResultsFound	31
8.5 Plugins.mod.io.Runtime.Utility.CoroutineRunner Class Reference	31
8.6 ModIO.CreationToken Class Reference	32
8.6.1 Detailed Description	32
8.7 ModIO.DownloadReference Struct Reference	32
8.7.1 Detailed Description	33
8.7.2 Member Function Documentation	33
8.7.2.1 IsValid()	33
8.8 ModIO.IModIoWebRequest Interface Reference	33
8.9 ModIO.InstalledMod Struct Reference	34
8.9.1 Detailed Description	34
8.9.2 Member Data Documentation	35
8.9.2.1 enabled	35
8.10 ModIO.Implementation.InstalledModExtensions Class Reference	35
8.11 ModIO.Util.ISimpleMessage Interface Reference	35

8.12 ModIO.Util.ISimpleMonoSingleton Interface Reference	35
8.13 ModIO.ModComment Struct Reference	36
8.13.1 Member Data Documentation	36
8.13.1.1 threadPosition	36
8.14 ModIO.Implementation.API.Objects.ModDependencies Struct Reference	37
8.14.1 Detailed Description	37
8.15 ModIO.Implementation.API.Objects.ModDependenciesObject Struct Reference	37
8.15.1 Detailed Description	37
8.16 ModIO.ModfileDetails Class Reference	38
8.16.1 Member Data Documentation	38
8.16.1.1 metadata	38
8.17 ModIO.ModId Struct Reference	38
8.17.1 Detailed Description	39
8.18 ModIO.ModIOUnity Class Reference	39
8.18.1 Detailed Description	42
8.18.2 Member Function Documentation	42
8.18.2.1 AddDependenciesToMod()	43
8.18.2.2 AddModComment()	43
8.18.2.3 AddTags()	44
8.18.2.4 ArchiveModProfile()	45
8.18.2.5 AuthenticateUserViaDiscord()	45
8.18.2.6 AuthenticateUserViaEpic()	46
8.18.2.7 AuthenticateUserViaGOG()	47
8.18.2.8 AuthenticateUserViaGoogle()	47
8.18.2.9 AuthenticateUserViaItch()	48
8.18.2.10 AuthenticateUserViaOculus()	49
8.18.2.11 AuthenticateUserViaPlayStation()	50
8.18.2.12 AuthenticateUserViaSteam()	51
8.18.2.13 AuthenticateUserViaSwitch()	52
8.18.2.14 AuthenticateUserViaXbox()	53
8.18.2.15 CreateModProfile()	54
8.18.2.16 DeleteModComment()	55
8.18.2.17 DeleteTags()	55
8.18.2.18 DisableModManagement()	56
8.18.2.19 DownloadImage()	56
8.18.2.20 EditModProfile()	57
8.18.2.21 EnableModManagement()	58
8.18.2.22 FetchUpdates()	58
8.18.2.23 ForceUninstallMod()	59
8.18.2.24 GenerateCreationToken()	59
8.18.2.25 GetCurrentModManagementOperation()	60
8.18.2.26 GetCurrentUploadHandle()	60

8.18.2.27	GetCurrentUser()	61
8.18.2.28	GetCurrentUserRatingFor()	61
8.18.2.29	GetCurrentUserRatings()	62
8.18.2.30	GetInstalledModsForUser()	62
8.18.2.31	GetMod()	63
8.18.2.32	GetModComments()	63
8.18.2.33	GetModDependencies()	64
8.18.2.34	GetMods()	65
8.18.2.35	GetSubscribedMods()	65
8.18.2.36	GetSystemInstalledMods()	66
8.18.2.37	GetTagCategories()	67
8.18.2.38	GetTermsOfUse()	67
8.18.2.39	InitializeForUser() [1/2]	68
8.18.2.40	InitializeForUser() [2/2]	68
8.18.2.41	IsAuthenticated()	69
8.18.2.42	IsInitialized()	70
8.18.2.43	IsModManagementBusy()	70
8.18.2.44	LogOutCurrentUser()	71
8.18.2.45	MuteUser()	71
8.18.2.46	RateMod()	71
8.18.2.47	RemoveDependenciesFromMod()	72
8.18.2.48	Report()	73
8.18.2.49	RequestAuthenticationEmail()	73
8.18.2.50	SetLoggingDelegate()	74
8.18.2.51	Shutdown()	75
8.18.2.52	SubmitEmailSecurityCode()	75
8.18.2.53	SubscribeToMod()	76
8.18.2.54	UnmuteUser()	76
8.18.2.55	UnsubscribeFromMod()	77
8.18.2.56	UpdateModComment()	77
8.18.2.57	UploadModfile()	79
8.18.2.58	UploadModMedia()	80
8.19	ModIO.ModIOUnityAsync Class Reference	80
8.19.1	Detailed Description	83
8.19.2	Member Function Documentation	83
8.19.2.1	AddDependenciesToMod()	83
8.19.2.2	AddModComment()	84
8.19.2.3	AddTags()	84
8.19.2.4	ArchiveModProfile()	86
8.19.2.5	AuthenticateUserViaDiscord()	87
8.19.2.6	AuthenticateUserViaEpic()	87
8.19.2.7	AuthenticateUserViaGOG()	88

8.19.2.8 AuthenticateUserViaGoogle()	89
8.19.2.9 AuthenticateUserViaItch()	90
8.19.2.10 AuthenticateUserViaOculus()	91
8.19.2.11 AuthenticateUserViaPlayStation()	91
8.19.2.12 AuthenticateUserViaSteam()	92
8.19.2.13 AuthenticateUserViaSwitch()	93
8.19.2.14 AuthenticateUserViaXbox()	94
8.19.2.15 CreateModProfile()	95
8.19.2.16 DeleteModComment()	95
8.19.2.17 DeleteTags()	96
8.19.2.18 DownloadImage()	96
8.19.2.19 EditModProfile()	97
8.19.2.20 FetchUpdates()	97
8.19.2.21 GetCurrentUser()	98
8.19.2.22 GetCurrentUserRatingFor()	98
8.19.2.23 GetCurrentUserRatings()	99
8.19.2.24 GetMod()	99
8.19.2.25 GetModComments()	100
8.19.2.26 GetModDependencies()	100
8.19.2.27 GetMods()	101
8.19.2.28 GetTagCategories()	101
8.19.2.29 GetTermsOfUse()	102
8.19.2.30 IsAuthenticated()	102
8.19.2.31 MuteUser()	102
8.19.2.32 RateMod()	103
8.19.2.33 RemoveDependenciesFromMod()	103
8.19.2.34 Report()	104
8.19.2.35 RequestAuthenticationEmail()	105
8.19.2.36 Shutdown()	105
8.19.2.37 SubmitEmailSecurityCode()	105
8.19.2.38 SubscribeToMod()	106
8.19.2.39 UnmuteUser()	106
8.19.2.40 UnsubscribeFromMod()	107
8.19.2.41 UpdateModComment()	107
8.19.2.42 UploadModfile()	108
8.19.2.43 UploadModMedia()	109
8.20 ModIO.ModIoWebRequest Class Reference	109
8.20.1 Member Function Documentation	110
8.20.1.1 GetResponseHeader()	110
8.20.2 Property Documentation	110
8.20.2.1 downloadedBytes	110
8.20.2.2 downloadProgress	110

8.20.2.3 isDone	110
8.20.2.4 uploadedBytes	111
8.20.2.5 uploadProgress	111
8.21 ModIO.ModPage Struct Reference	111
8.21.1 Detailed Description	111
8.21.2 Member Data Documentation	111
8.21.2.1 modProfiles	112
8.21.2.2 totalSearchResultsFound	112
8.22 ModIO.ModProfile Struct Reference	112
8.22.1 Detailed Description	113
8.22.2 Member Data Documentation	113
8.22.2.1 metadata	113
8.23 ModIO.ModProfileDetails Class Reference	114
8.23.1 Detailed Description	114
8.23.2 Member Data Documentation	115
8.23.2.1 communityOptions	115
8.23.2.2 contentWarning	115
8.23.2.3 description	115
8.23.2.4 homepage_url	115
8.23.2.5 images	115
8.23.2.6 logo	116
8.23.2.7 maxSubscribers	116
8.23.2.8 metadata	116
8.23.2.9 modId	116
8.23.2.10 name	117
8.23.2.11 name_id	117
8.23.2.12 summary	117
8.23.2.13 tags	117
8.23.2.14 visible	118
8.24 ModIO.ModStats Struct Reference	118
8.24.1 Detailed Description	118
8.25 ModIO.Util.MonoDispatcher Class Reference	118
8.25.1 Member Function Documentation	119
8.25.1.1 Awake()	119
8.26 ModIO.Util.MonoSingleton< T > Class Template Reference	119
8.26.1 Member Function Documentation	120
8.26.1.1 SetupSingleton()	120
8.27 ModIO.Util.Mutex Class Reference	120
8.27.1 Detailed Description	121
8.28 ModIO.ProgressHandle Class Reference	121
8.28.1 Detailed Description	121
8.28.2 Property Documentation	121

8.28.2.1 BytesPerSecond	121
8.28.2.2 Completed	122
8.29 ModIO.Implementation.API.Objects.Rating Struct Reference	122
8.29.1 Detailed Description	122
8.30 ModIO.Implementation.API.Objects.RatingObject Struct Reference	122
8.30.1 Detailed Description	123
8.31 ModIO.Report Class Reference	123
8.31.1 Detailed Description	123
8.31.2 Constructor & Destructor Documentation	123
8.31.2.1 Report()	123
8.32 ModIO.Result Struct Reference	124
8.32.1 Detailed Description	124
8.32.2 Member Function Documentation	124
8.32.2.1 IsNetworkError()	125
8.33 ModIO.ResultAnd< T > Class Template Reference	125
8.33.1 Detailed Description	125
8.34 ModIO.SearchFilter Class Reference	125
8.34.1 Detailed Description	126
8.34.2 Member Function Documentation	126
8.34.2.1 AddSearchPhrase()	126
8.34.2.2 AddTag()	127
8.34.2.3 AddUser()	127
8.34.2.4 IsSearchFilterValid()	127
8.34.2.5 SetPageIndex()	128
8.34.2.6 SetPageSize()	128
8.34.2.7 SetToAscending()	129
8.34.2.8 SortBy()	129
8.35 ModIO.Util.SelfInstantiatingMonoSingleton< T > Class Template Reference	129
8.35.1 Member Function Documentation	130
8.35.1.1 SetupSingleton()	130
8.36 ModIO.ServerSettings Struct Reference	131
8.36.1 Detailed Description	131
8.37 ModIO.Util.SimpleMessageHub Class Reference	131
8.37.1 Member Function Documentation	132
8.37.1.1 OnDestroy()	132
8.37.1.2 Publish< T >()	132
8.37.1.3 PublishThreadSafe< T >()	133
8.37.1.4 Subscribe< T >()	133
8.37.2 Member Data Documentation	133
8.37.2.1 dictionary	133
8.38 ModIO.Util.SimpleMessageUnsubscribeToken Class Reference	133
8.39 ModIO.Util.SimpleSingleton< T > Class Template Reference	134

8.40 ModIO.SubscribedMod Struct Reference	134
8.40.1 Detailed Description	134
8.40.2 Member Data Documentation	135
8.40.2.1 enabled	135
8.41 ModIO.Tag Struct Reference	135
8.41.1 Detailed Description	135
8.42 ModIO.TagCategory Struct Reference	135
8.42.1 Detailed Description	136
8.43 ModIO.TermsHash Struct Reference	136
8.43.1 Detailed Description	136
8.44 ModIO.TermsOfUse Struct Reference	136
8.44.1 Detailed Description	137
8.45 ModIO.TermsOfUseLink Struct Reference	137
8.45.1 Detailed Description	137
8.46 ModIO.UserInstalledMod Struct Reference	137
8.46.1 Detailed Description	138
8.46.2 Member Data Documentation	138
8.46.2.1 enabled	138
8.47 ModIO.UserProfile Struct Reference	138
8.47.1 Detailed Description	139
8.48 ModIO.Util.Utility Class Reference	139
8.48.1 Member Function Documentation	139
8.48.1.1 EncodeEncryptedSteamAppTicket()	139
8.48.1.2 FindEverythingInScene< T >()	140
8.48.1.3 GenerateHumanReadableNumber()	140

Chapter 1

LICENSE

MIT License

Copyright (c) 2022 mod.io Proprietary Limited

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Chapter 2

Unity V1 Upgrade Guide

Where the [Unity Plugin QuickStart Guide](#) contains information to help you install and setup the new Unity Plugin in your project, this guide exists to assist with upgrading from mod.io Unity V1 to the new mod.io Unity Plugin by offering equivalent examples of the most common functions.

2.1 Why Upgrade to the New Plugin?

mod.io Unity Plugin V1 has been deprecated as structural and design decisions have made it impractical to continue maintaining and enhancing with new features and functionality.

Upgrading to the new mod.io Unity Plugin results in extended support, additional features, and improved user experience. It provides enhanced support for consoles and boasts a new UI, as well as numerous enhancements, enabling you to accomplish more with less code.

2.2 Core Differences

2.2.1 Interface

The new mod.io Unity Plugin has a simplified and clearer interface. All core functionality is accessible via `ModIOUnity.cs`, or the asynchronous version `ModIOUnityAsync.cs`, located within the `ModIO` namespace. Additionally, the browser code is contained in `Browser.cs` in the `ModIOBrowser` namespace. No more calling functions from multiple static classes depending on the context!

2.2.2 Callbacks

All calls into async functions on the new mod.io Unity Plugin provide a single callback parameter with a `Result` or `ResultAnd<T>` parameter that will **always** be called at the conclusion of the operation. (Async methods return the result struct after an `await` instead.) This provides a much clearer calling convention and gives a guarantee of one operation completion - one callback invocation, rather than the two callbacks required by async functions in mod.io Unity V1.

Here is an example of the usage with a callback and an await:

```
{c#}  
void CallbackExample()  
{
```

```

ModIOUnity.GetMods(filter, (Result result, ModPage modPage)=>
{
    if (result.Succeeded())
    {
        // Success
    }
});
// OR
async void AsyncExample()
{
    ResultAnd<ModPage> getMods = await ModIOUnityAsync.GetMods(filter);
    if (getMods.result.Succeeded())
    {
        // Success
    }
}

```

2.3 Initializing the plugin

Initialization in mod.io Unity V1 was handled mostly statically, pulling the details from the `Resources/modio_v1_settings` asset during static initialization. Optionally, a developer could set the user data directory by calling `UserDataStorage.SetActiveUser()` as seen in the sample below.

```

{c#}
void modioUnityV1Example()
{
    // Optional (Sets the user data directory)
    string userProfileIdentifier = "local_game_user_id"; // Local User Account Identifier
    UserDataStorage.SetActiveUser(userProfileIdentifier, setActiveUserCallback);
}

```

Apart from this, there are no initialization possibilities in mod.io Unity V1.

For the new mod.io Unity Plugin, we have kept an automatic initialization option that pulls the data from the `Resources/mod.io/config` asset, similar to the function of mod.io Unity V1. However, there are also explicit initialization methods and shutdown methods that can be utilized if automatic initialization is disabled in the config asset.

```

{c#}
void InitializationExample()
{
    string userProfileIdentifier = "local_game_user_id"; // Local User Account Identifier
    Result result = ModIOUnity.InitializeForUser(userProfileIdentifier, initializationCallback);
    // Do work...
    ModIOUnity.Shutdown(shutdownCallback);
}

```

For further information, see the [initialization documentation](#).

2.4 Getting and Synchronizing Subscription Data

mod.io Unity V1 built the synchronization of an authenticated user's subscription data into the UI code, meaning that a developer not using the sample browser would be responsible for ensuring that the user's subscriptions were kept in agreement with the server. This has not changed in the new mod.io Unity Plugin, but the process of keeping that data synchronized is much easier, along with fetching the data for the subscribed mods.

Adding, synchronizing, and retrieving subscription data in mod.io Unity V1 involves chaining multiple calls together.

```

{c#}
void modioUnityV1Example()
{
    int newSubModId = 123;
    int newUnsubModId = 456;
    // This call adds the sub to the local cache and queues it for synchronization
    UserAccountManagement.SubscribeToMod(newSubModId);
    // This call adds the unsub to the local cache and queues it for synchronization
    UserAccountManagement.UnsubscribeFromMod(newUnsubModId);
    // Push local subscription changes to mod.io servers
    UserAccountManagement.PushSubscriptionChanges(

```

```

    () => { /* chain callback into next section */ },
    (List<WebRequestError> unsortedErrorList) => { /* error callback code */ });
// Fetch remote subscription changes from mod.io servers
UserAccountManagement.PullSubscriptionChanges(
    (List<ModProfile> newRemoteSubscriptions) => { /* chain callback into next section */ },
    (WebRequestError error) => { /* error callback code */ });
// Get Mod Profiles for Subscribed Mods
List<int> subscribedModIds = LocalUser.subscribedModIds;
ModManager.GetModProfiles(modIds,
    (ModProfile[] modProfiles) => { /* success callback code */ },
    (WebRequestError error) => { /* error callback code */ });
}

```

The new mod.io Unity Plugin streamlines this process by reducing the need for callback chaining, synchronizing immediately for local changes, and removing the need to handle the mod ids. `FetchUpdates()` is the single synchronization function on the interface, handling all synchronization actions.

```

{c#}
void Example()
{
    int newSubModId = 123;
    int newUnsubModId = 456;
    // Pushes a subscribe attempt directly to the server, returning an error on failure
    ModIOUnity.SubscribeToMod(newSubModId,
        (Result result) => { /* callback code */ });
    // Pushes an unsubscribe attempt directly to the server, returning an error on failure
    ModIOUnity.UnsubscribeFromMod(newUnsubModId,
        (Result result) => { /* callback code */ });
    // Synchronizes the local and server data
    ModIOUnity.FetchUpdates(
        (Result result) => { /* callback code */ });
    // Get Subscribed mod data
    SubscribedMod[] subscribedMods = ModIOUnity.GetSubscribedMods(out Result result);
}

```

Furthermore, the subscribe and unsubscribe operations automatically flag the mod as requiring installation/uninstallation, a responsibility placed on the consumer in mod.io Unity V1. (See below)

2.5 Listing the User's Installed Mods

Like in mod.io Unity V1, the new mod.io Unity Plugin allows the sharing of installed mods across multiple users to save network traffic and storage space.

mod.io Unity V1 didn't have a direct method of retrieving the mods installed for the current user. There are a variety of different methods that need to be chained together to retrieve a complete picture of the installed mod data.

```

{c#}
void modioUnityV1Example()
{
    // Retrieves a de-identified list of mod directories for mods the user has "enabled"
    bool onlyEnabledMods = true;
    ModManager.QueryInstalledModDirectories(onlyEnabledMods,
        (List<string> installedModDirectories) => { /* callback code */ });
    // Retrieves a mapping of mod directories for mods the user has subscribed to
    List<int> subscriptions = LocalUser.subscribedModIds;
    ModManager.QueryInstalledMod(subscriptions,
        (IList<KeyValuePair<ModfileIdPair, string> modDirectoryMap) => { /* callback code */ });
    // Retrieves the data for the installed mods that the user has "enabled" with no directory
    bool onlyEnabledMods = true;
    ModManager.QueryInstalledModVersions(onlyEnabledMods,
        (List<ModfileIdPair> installedModVersions) =>
        {
            // map the mod ids to a list
            List<int> modIds = installedModVersions.Select(x => x.modId).ToList();
            ModManager.GetModProfiles(modIds,
                (ModProfile[] modProfiles) => { /* success callback code */ },
                (WebRequestError error) => { /* error callback code */ });
        });
}

```

The new mod.io Unity Plugin makes this much simpler, giving you all the information in a single call, returning a `UserInstalledMod` array (and a `Result`).

```

{c#}
void Example()
{
    UserInstalledMod[] mods = ModIOUnity.GetInstalledModsForUser(out Result result);
}

```

2.6 Downloading, Updating and Uninstalling Mods

The new mod.io Unity Plugin has the business rules of "Subscription = install and update" built into it, such that the download, extract, and uninstall processes are managed automatically by the [Mod Management Loop](#), a process that runs asynchronously to detect changes to the subscriptions and automate mod data management.

mod.io Unity V1 handled the installation and uninstallation of mods in the ModBrowser code, but any developer looking to exclude that code or understand the installation process had a more difficult time.

```
{c#}
void modioUnityV1Example()
{
    /// === Add a new subscription and install ===
    int newSubModId = 123;
    // This call adds the sub to the local cache and queues it for synchronization
    UserAccountManagement.SubscribeToMod(newSubModId);
    // Push local subscription changes to mod.io servers
    UserAccountManagement.PushSubscriptionChanges(
        () => { /* chain callback into next section */ },
        (List<WebRequestError> unsortedErrorList) => { /* error callback code */ });
    // Download and Install all mods (equivalent to new mod.io Unity Plugin)
    gameObject.StartCoroutine(ModManager.DownloadAndUpdateMods_Coroutine(LocalUser.subscribedModIds,
        () => { /* callback code */ }));
    /// === Remove a subscription and uninstall ===
    int newUnsubModId = 456;
    // This call adds the unsub to the local cache and queues it for synchronization
    UserAccountManagement.UnsubscribeFromMod(newUnsubModId);
    // Push local subscription changes to mod.io servers
    UserAccountManagement.PushSubscriptionChanges(
        () => { /* chain callback into next section */ },
        (List<WebRequestError> unsortedErrorList) => { /* error callback code */ });
    // Uninstall the mod
    ModManager.UninstallMod(newUnsubModId,
        (bool uninstallSuccess) => { /* callback code */ });
    /// === Fetch remote data and fix installation state ===
    // Fetch remote subscription changes from mod.io servers
    UserAccountManagement.PullSubscriptionChanges(
        (List<ModProfile> newRemoteSubscriptions) => { /* chain callback into next section */ },
        (WebRequestError error) => { /* error callback code */ });
    // Download and Install all mods (equivalent to new mod.io Unity Plugin)
    gameObject.StartCoroutine(ModManager.DownloadAndUpdateMods_Coroutine(LocalUser.subscribedModIds,
        () => { /* chain callback into next section */ }));
    // Calculate uninstall mods
    List<int> modsToUninstall = new List<int>();
    List<int> subscribedModIds = LocalUser.subscribedModIds;
    ModManager.QueryInstalledModVersions(subscribedModIds,
        (IList<KeyValuePair<ModfileIdPair, string>> modDirectoryMap) =>
        {
            if(!subscribedModIds.Contains(modDirectoryMap.Key.modId))
            {
                modsToUninstall.Add(modDirectoryMap.Key.modId);
            }
        })
        /* chain callback into next section */
    );
    // uninstall mods
    foreach(int modId in modsToUninstall)
    {
        ModManager.UninstallMod((bool uninstallSuccess) => { /* callback code */ });
    }
}
```

Of note, the uninstall process above, can't account for mods installed by other users on the system. This is one of the key processes that has been streamlined in the new mod.io Unity Plugin.

A call to [ModIOUnity.EnableModManagement](#) starts the background process of monitoring for subscription changes, and takes a (nullable) callback for mod management events. This can be disabled at any point with a call to [ModIOUnity.DisableModManagement](#). Any changes invoked locally, and any changes retrieved with [ModIOUnity.FetchUpdates](#) are automatically queued and actioned.

```
{c#}
void Example()
{
    ModManagementEventDelegate eventDelegate = (ModManagementEventType eventType, ModId modId, Result
        eventResult) => { /* handle event */ };
    // Begins monitoring for changes and enables downloading/extracting/deleting of mod data
    ModIOUnity.EnableModManagement(eventDelegate);
    /// === Add a new subscription and install ===
    int newSubModId = 123;
    // Pushes the subscription to the server and flags for download and installation
    ModIOUnity.SubscribeToMod(newSubModId, (Result result) => { /* callback code */ });
}
```



```

/// === Remove a subscription and uninstall ===
int newUnsubModId = 456;
// Pushes the unsubscribe action to the server and flags it for uninstallation
ModIOUnity.UnsubscribeFromMod(newUnsubModId, (Result result) => /* callback code */);
// === Fetch remote data and fix installation state ===
// Synchronizes local and server data, flagging install/uninstall operations as required
ModIOUnity.FetchUpdates((Result result) => { /* callback code */ });
// Ends monitoring for changes and disables downloading/extracting/deleting of mod data
ModIOUnity.DisableModManagement();
}

```

Note: The `Result` returned from `ModIOUnity.SubscribeToMod` and `ModIOUnity.UnsubscribeFromMod` indicate the outcome of the subscribe/unsubscribe attempt being sent to the server. For notification of the outcome of an installation/uninstallation operation, the delegate passed to `ModIOUnity.EnableModManagement` will be invoked.

2.7 Authenticating a user

The new Plugin requires a user to be authenticated in order to download, install and manage mods. The plugin offers email authentication or numerous third party methods, such as Steam or Xbox.

2.7.1 Email Authentication

In the V1 Plugin you can authenticate via email in the following way:

```

{c#}
// TODO: EXAMPLE OF EMAIL AUTH IN V1

```

In the new Plugin you can do it like so:

```

{c#}
async void RequestEmailCode()
{
    Result result = await ModIOUnityAsync.RequestAuthenticationEmail("johndoe@gmail.com");
    if (result.Succeeded())
    {
        Debug.Log("Succeeded to send security code");
    }
    else
    {
        Debug.Log("Failed to send security code to that email address");
    }
}
async void SubmitCode(string userSecurityCode)
{
    Result result = await ModIOUnityAsync.SubmitEmailSecurityCode(userSecurityCode);
    if (result.Succeeded())
    {
        Debug.Log("You have successfully authenticated the user");
    }
    else
    {
        Debug.Log("Failed to authenticate the user");
    }
}
}

```

2.7.2 Third party Authentication

In the V1 Plugin you can authenticate a user with a third party service like Steam:

```

{c#}
UserAccountManagement.AuthenticateWithSteamEncryptedAppTicket(pTicket,
                                                                pcbTicket,
                                                                hasUserAcceptedTerms,
                                                                onSuccess,
                                                                onError);

```

In the new Plugin you can use the following:

```

{c#}
ModIOUnity.AuthenticateUserViaSteam(token, email, termsHash, callback);

```

(Note you can use `ModIOUnityAsync` to await instead.)

Be advised that you will need to get the terms of use hash key in order to properly authenticate. This requires the use of `ModIOUnity.GetTermsOfUse`. This is to ensure users view and accept the terms of use.

Here is a complete example getting the TOS hash and Authenticating via Steam:

```
{c#}
// Use this to cache the TOS we receive
TermsOfUse termsOfUse;
// This needs to be displayed to the user before they can authenticate
async void DisplayTOS()
{
    ResultAnd<TermsOfUse> tos = await ModIOUnityAsync.GetTermsOfUse();
    if (tos.result.Succeeded())
    {
        // Display the terms of use via a text field
        TextField.text = tos.value.termsOfUse;

        // cache the Terms of Use
        termsOfUse = tos.value;
    }
}
// If they agree after viewing the TOS you can attempt to authenticate with the TOS hash
async void AgreeAndAuthenticate(string token, string email)
{
    Result result = await ModIOUnityAsync.AuthenticateUserViaSteam(token, email, termsOfUse.hash);

    if (result.Succeeded())
    {
        // Succeeded to authenticate via Steam
    }
}
```

2.8 Need more help?

If you require additional assistance you're welcome to contact us directly via email or on our Discord channel. Our team can help answer specific questions about the plugin and provide support.

Chapter 3

README

3.1 mod.io Unity Plugin v2023.6.1

Welcome to the mod.io Unity Engine plugin repository. It allows game developers to host and automatically install user-created mods in their games which use **Unity 2019.4** or newer. It provides a UI for mod discovery, installation and collection management, and a C# interface which connects to the [mod.io REST API](#).

3.1.1 Watch the video tutorial

3.1.2 Platform Support

Platform	Support
Windows	Yes
Windows (GDK)	Yes Contact us
XBox (GDK)	Yes Contact us
PlayStation 4	Yes Contact us
PlayStation 5	Yes Contact us
Nintendo Switch	Yes Contact us
Linux	Yes
macOS	Yes

3.1.2.1 Git Repository or .unitypackage

You can import the plugin directly from the [Unity Asset Store](#), or by downloading the package directly from the [Releases page](#). If you have any previous versions of the plugin installed, it is highly recommended to delete them before importing a newer version.

Alternatively, you can download an archive of the code using GitHub's download feature and place it in the Assets/Plugins directory within your Unity project.

3.1.3 Getting started

1. Set up your [game profile on mod.io](#) (or our [private test environment](#)) to get your game ID and API key.
2. Add the plugin to your project using the installation instructions above.
3. Ensure you don't have any conflicting libraries by going to Assets/Plugins/mod.io/ThirdParty to remove any libraries you may already have in your project (such as JsonNet).
4. Restart unity to ensure it recognises the new assembly definitions.
5. Go to Tools > mod.io > Edit Settings to locate the config file.
6. Select the config file and use the inspector to assign your game ID and API key in server settings (Make sure to deselect the config file before using playmode in the editor. A known unity bug can cause the editor to crash in 2019-2021).
7. Setup complete! Join us [on Discord](#) if you have any questions or need help.

3.1.4 Setting up the Browser UI

If you do not wish to create your own UI implementation you can use our default UI that comes built in to the plugin. Examples of how the UI looks are provided below. (If you don't wish to use the UI it is safe to delete the UI folder located at Assets/Plugins/mod.io/UI)

1. Follow the steps above to setup the config.
2. Navigate to the ModIOBrowser prefab at Assets/Plugins/mod.io/UI/Examples and drag it into your scene.
3. Use the `ModIOBrowser.Browser.OpenBrowser()` method to open the browser in your scene.

```
ModIOBrowser.Browser.OpenBrowser(null)
```
4. The Browser UI is now setup!

3.1.5 Authentication

In the current version of the plugin it is required that a user session is authenticated. Either via email or through another third party, such as Steam or Google. The process is fairly simply. Examples can be found below.

3.1.6 Usage

below are a couple examples for some of the common usages of the plugin. Such as initialising, authenticating, enabling automatic downloads and installs, and getting a few mods from the mod.io server.

All of the methods required to use the plugin can be found in `ModIOUnity.cs`. If you prefer using async methods over callbacks you can alternatively use `ModIOUnityAsync.cs` to use an async variation of the same methods.

3.1.6.1 Initialise the plugin

```
{c#}
void Example()
{
    Result result = ModIOUnity.InitializeForUser("ExampleUser");
    if (result.Succeeded())
    {
        Debug.Log("Initialised plugin");
    }
    else
    {
        Debug.Log("Failed to initialise plugin");
    }
}
```

3.1.6.2 Get the user's installed mods

```
{c#}
void Example()
{
    UserInstalledMod[] mods = ModIOUnity.GetInstalledModsForUser(out Result result);

    foreach(UserInstalledMod mod in mods)
    {
        // This is the location of the installed mod
        string directory = mod.directory;
    }
}
```

3.1.6.3 Enable automatic mod downloads and installs

```
{c#}
void Example()
{
    Result result = ModIOUnity.EnableModManagement(ModManagementDelegate);
    if (result.Succeeded())
    {
        Debug.Log("Enabled mod management");
    }
    else
    {
        Debug.Log("Failed to enable mod management");
    }
}

// The following method will get invoked whenever an event concerning mod management occurs
void ModManagementDelegate(ModManagementEventType eventType, ModId modId, Result result)
{
    Debug.Log("a mod management event of type " + eventType.ToString() + " has been invoked");
}
```

3.1.6.4 Authenticate a user

In the current version of the plugin it is required that a user session is authenticated in order to subscribe and download mods. You can accomplish this with an email address or through another third party service, such as Steam or Google. Below is an example of how to do this from an email address provided by the user. A security code will be sent to their email account and can be used to authenticate (The plugin will cache the session token to avoid having to re-authenticate every time they run the application).

```
{c#}
async void RequestEmailCode()
{
    Result result = await ModIOUnityAsync.RequestAuthenticationEmail("johndoe@gmail.com");
    if (result.Succeeded())
    {
        Debug.Log("Succeeded to send security code");
    }
    else
    {
        Debug.Log("Failed to send security code to that email address");
    }
}

async void SubmitCode(string userSecurityCode)
{
    Result result = await ModIOUnityAsync.SubmitEmailSecurityCode(userSecurityCode);
    if (result.Succeeded())
    {
    }
}
```

```

    {
        Debug.Log("You have successfully authenticated the user");
    }
    else
    {
        Debug.Log("Failed to authenticate the user");
    }
}

```

3.1.6.5 Get Mod profiles from the mod.io server

```

{c#}
async void Example()
{
    // create a filter to retrieve the first ten mods for your game
    SearchFilter filter = new SearchFilter();
    filter.SetPageIndex(0);
    filter.SetPageSize(10);

    ResultAnd<ModPage> response = await ModIOUnityAsync.GetMods(filter);
    if (response.result.Succeeded())
    {
        Debug.Log("ModPage has " + response.value.modProfiles.Length + " mods");
    }
    else
    {
        Debug.Log("failed to get mods");
    }
}

```

3.1.7 Submitting mods

You can also submit mods directly from the plugin. Refer to the documentation for methods such as `ModIOUnity.CreateModProfile` and `ModIOUnity.UploadModfile`.

Users can also submit mods directly from the mod.io website by going to your game profile page. Simply create an account and upload mods directly.

3.1.7.1 Adding a mod

Here we go through the mod addition flow. Generate Token, Create Mod Profile, and Upload Mod File

```

{c#}
public async void CreateMod()
{
    //token used to create mod profile
    var token = ModIOUnity.GenerateCreationToken();
    //Mod profile specifics
    ModProfileDetails modDetails = new ModProfileDetails
    {
        logo = GetTexture(), //the texture you will use for this mod's logo
        summary = "A brief summary of the mod.",
        name = "Mod Name"
    };
    //create the mod profile
    var createResultAnd = await ModIOUnityAsync.CreateModProfile(token, modDetails);
    if(!createResultAnd.result.Succeeded())
        return; //create mod unsuccessful
    //Points to a folder where all mod files are located (folder cannot be empty)
    ModfileDetails modFile = new ModfileDetails
    {
        modId = createResultAnd.value,
        directory = "files/mods/mod_123"
    };
    //upload the file to the mod profile
    var result = await ModIOUnityAsync.UploadModfile(modFile);
    if(result.Succeeded())
    {
        //Upload file successful!
    }
}

```

3.1.7.2 Loading mods

Here is an example that grabs all mods installed for the current user, finds the png files in the mod's directory if they are tagged as a "Texture" and then loads them into a Texture2D asset.

```
{c#}
public void LoadModExample()
{
    UserInstalledMod[] mods = ModIOUnity.GetInstalledModsForUser(out Result result);
    if (result.Succeeded())
    {
        foreach (var mod in mods)
        {
            //Tags are USER defined strings for a game and are setup in the web portal.
            string textureTag = "Texture";

            string directoryWithInstalledMod = mod.directory;

            //Optionally, you may want to use tags to help you determine the files to look for in an installed
            mod folder
            if (!mod.modProfile.tags.Contains(textureTag))
            {
                //Get all files in a directory
                string[] filePaths = System.IO.Directory.GetFiles(directoryWithInstalledMod);
                foreach (var path in filePaths)
                {
                    //Find .png files so that we can convert them into textures
                    if (path.EndsWith(".png"))
                    {
                        Texture2D tex = new Texture2D(1024, 1024);

                        //Load a texture from directory
                        tex.LoadImage(File.ReadAllBytes(path));

                        //Now you can replace the current texture in your game with the new one
                    }
                }
            }
        }
    }
}
```

3.1.8 Dependencies

The **mod.io** Unity Plugin requires the functionality of two other open-source Unity plugins to run. These are included as libraries in the UnityPackage in the `Assets/Plugins/mod.io/ThirdParty` directory:

- Json.Net for improved Json serialization. ([GitHub Repo](#) || [Unity Asset Store Page](#))
- SharpZipLib to zip and unzip transmitted files. ([GitHub Repo](#))

3.1.9 Benefits

mod.io offers the same core functionality as Steamworks Workshop (1 click mod installs in-game), plus mod hosting, moderation and all of the critical pieces needed. Where we differ is our approach to modding and the flexibility a REST API offers. For example:

- We make mods cross platform accessible. That means users can upload a mod on PC and someone else can play it on the Xbox, for example.
- Our API is not dependent on a client, platform or SDK, allowing you to run mod.io in many places such as your homepage and launchers.
- Designing a good mod browsing UI is hard, our plugin ships with a UI built in to save you a lot of effort and help your mods stand out.
- We don't apply rules globally, so if you want to enable patronage, sales or other experimental features, reach out to discuss.
- Our platform is built by the super experienced ModDB.com team and is continually improving for your benefit.
- Your community can consume the mod.io API to build modding fan sites or discord bots if they want.

3.1.10 Game studios and Publishers

If you need assistance with 1st party approvals, or require a private, white-label UGC solution. [Contact us](#) to discuss.

3.1.11 Contributions Welcome

Our Unity plugin is public and open source. Game developers are welcome to utilize it directly, to add support for mods in their games, or fork it for their games customized use. Want to make changes to our plugin? Submit a pull request with your recommended changes to be reviewed.

3.1.12 Other Repositories

Our aim with [mod.io](#), is to provide an [open modding API](#). You are welcome to [view, fork and contribute to our other codebases](#) in use.

Chapter 4

Namespace Index

4.1 Package List

Here are the packages with brief descriptions (if available):

ModIO	23
ModIO.Implementation	26
ModIO.Implementation.API	26
ModIO.Implementation.API.Objects	26
ModIO.Util	26
Plugins	27
Plugins.mod	27
Plugins.mod.io	27
Plugins.mod.io.Runtime	27
Plugins.mod.io.Runtime.Utility	27

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

ModIO.AuthenticationServiceProviderExtensions	29
ModIO.BuildSettings	29
CommentDetails	30
ModIO.CommentPage	30
ModIO.CreationToken	32
ModIO.DownloadReference	32
ModIO.IModIoWebRequest	33
ModIO.ModIoWebRequest	109
ModIO.InstalledMod	34
ModIO.Implementation.InstalledModExtensions	35
ModIO.Util.ISimpleMessage	35
ModIO.Util.ISimpleMonoSingleton	35
ModIO.Util.MonoSingleton< T >	119
ModIO.Util.SelfInstantiatingMonoSingleton< T >	129
ModIO.ModComment	36
ModIO.Implementation.API.Objects.ModDependencies	37
ModIO.Implementation.API.Objects.ModDependenciesObject	37
ModIO.ModfileDetails	38
ModIO.ModId	38
ModIO.ModIOUnity	39
ModIO.ModIOUnityAsync	80
ModIO.ModPage	111
ModIO.ModProfile	112
ModIO.ModProfileDetails	114
ModIO.ModStats	118
MonoBehaviour	
ModIO.Util.MonoSingleton< T >	119
ModIO.Util.SelfInstantiatingMonoSingleton< T >	129
ModIO.Util.Mutex	120
ModIO.ProgressHandle	121
ModIO.Implementation.API.Objects.Rating	122
ModIO.Implementation.API.Objects.RatingObject	122
ModIO.Report	123
ModIO.Result	124
ModIO.ResultAnd< T >	125

ModIO.SearchFilter	125
ModIO.Util.SelfInstantiatingMonoSingleton< CoroutineRunner >	129
Plugins.mod.io.Runtime.Utility.CoroutineRunner	31
ModIO.Util.SelfInstantiatingMonoSingleton< MonoDispatcher >	129
ModIO.Util.MonoDispatcher	118
ModIO.Util.SelfInstantiatingMonoSingleton< PrefabPool >	129
ModIO.Util.SelfInstantiatingMonoSingleton< SimpleMessageHub >	129
ModIO.Util.SimpleMessageHub	131
ModIO.Util.SelfInstantiatingMonoSingleton< SimpleMessageHubTester >	129
ModIO.ServerSettings	131
ModIO.Util.SimpleMessageUnsubscribeToken	133
ModIO.Util.SimpleSingleton< T >	134
ModIO.SubscribedMod	134
ModIO.Tag	135
ModIO.TagCategory	135
ModIO.TermsHash	136
ModIO.TermsOfUse	136
ModIO.TermsOfUseLink	137
ModIO.UserInstalledMod	137
ModIO.UserProfile	138
ModIO.Util.Utility	139

Chapter 6

Class Index

6.1 Class List

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

ModIO.AuthenticationServiceProviderExtensions	29
ModIO.BuildSettings	
Build-specific configuration values. This can be setup directly from the inspector when editing the config settings file, or you can instantiate and use this at runtime with the Initialize method	29
CommentDetails	30
ModIO.CommentPage	
A struct containing the ModComments and total number of remaining results that can be acquired with the SearchFilter used in the GetMods request.	30
Plugins.mod.io.Runtime.Utility.CoroutineRunner	31
ModIO.CreationToken	
This is used with creating new mod profiles. Using a token ensures you dont create duplicate profiles.	32
ModIO.DownloadReference	
Used in ModIOUnity.DownloadTexture() to get the Texture. (DownloadReference is serializable with Unity's JsonUtility)	32
ModIO.IModIoWebRequest	33
ModIO.InstalledMod	
Struct used to represent a mod that already exists on the current device. You can view the subscribed users to this mod as well as the directory and modprofile associated to it.	34
ModIO.Implementation.InstalledModExtensions	35
ModIO.Util.ISimpleMessage	35
ModIO.Util.ISimpleMonoSingleton	35
ModIO.ModComment	36
ModIO.Implementation.API.Objects.ModDependencies	
A struct representing all of the information available for a Mod's Dependencies.	37
ModIO.Implementation.API.Objects.ModDependenciesObject	
A struct representing all of the information available for a ModDependenciesObject .	37
ModIO.ModfileDetails	38
ModIO.ModId	
A struct representing the globally unique identifier for a specific mod profile.	38
ModIO.ModIOUnity	
Main interface for the mod.io Unity plugin. Every method within ModIOUnity.cs that has a callback can also be found in ModIOUnityAsync with an asynchronous alternative method (if you'd rather not use callbacks)	39

ModIO.ModIOUnityAsync	
Main async interface for the mod.io Unity plugin. Every method within ModIOUnity.cs that has a callback can also be found in ModIOUnityAsync with an asynchronous alternative method (if you'd rather not use callbacks).	80
ModIO.ModIoWebRequest	109
ModIO.ModPage	
A struct containing the ModProfiles and total number of remaining results that can be acquired with the SearchFilter used in the GetMods request.	111
ModIO.ModProfile	
A struct representing all of the information available for a ModProfile .	112
ModIO.ModProfileDetails	
Use this class to fill out the details of a Mod Profile that you'd like to create or edit. If you're submitting this via CreateModProfile you must assign values to logo, name and summary, otherwise the submission will be rejected (All fields except modId are optional if submitting this via EditModProfile)	114
ModIO.ModStats	
Detailed stats about a Mod's ratings, downloads, subscribers, popularity etc	118
ModIO.Util.MonoDispatcher	118
ModIO.Util.MonoSingleton< T >	119
ModIO.Util.Mutex	
This serves only as an abstract handle for using lock(mutex) to synchronize IO operations	120
ModIO.ProgressHandle	
A ProgressHandle can only be used to monitor the progress of an operation and cannot be used to cancel or suspend ongoing operations. The OperationType enum field specifies what type of operation this handle is for. The Progress field can be used to get the percentage (0.0 - 1.0) of the progress. The Completed and Failed fields can be used to determine if the operation is complete and whether or not it failed.	121
ModIO.Implementation.API.Objects.Rating	
A struct representing all of the information available for a Rating .	122
ModIO.Implementation.API.Objects.RatingObject	
A struct representing all of the information available for a ModDependenciesObject .	122
ModIO.Report	
Used in conjunction with ModIOUnity.Report() to send a report to the mod.io server for a specific mod.	123
ModIO.Result	
Struct returned from ModIO callbacks to inform the caller if the operation succeeded.	124
ModIO.ResultAnd< T >	
Convenience wrapper for essentially a Tuple.	125
ModIO.SearchFilter	
Used to build a filter that is sent with requests for retrieving mods.	125
ModIO.Util.SelfInstantiatingMonoSingleton< T >	129
ModIO.ServerSettings	
Describes the server settings to use for the ModIO Plugin. This can be setup directly from the inspector when editing the config settings file, or you can instantiate and use this at runtime with the Initialize method	131
ModIO.Util.SimpleMessageHub	131
ModIO.Util.SimpleMessageUnsubscribeToken	133
ModIO.Util.SimpleSingleton< T >	134
ModIO.SubscribedMod	
Represents the ModProfile of a mod the current user has subscribed to. Contains the status and a directory (if installed) and the associated ModProfile .	134
ModIO.Tag	
Represents a Tag that can be assigned to a mod.	135
ModIO.TagCategory	
Represents a particular category of tags.	135
ModIO.TermsHash	
This is the hash that identifies the TOS. Used to validate the TOS requirement when attempting to authenticate a user.	136

ModIO.TermsOfUse	
TOS object received from a successful use of ModIOUnity.GetTermsOfUse This is used when attempting to authenticate via a third party. You must retrieve the TOS and input it along with an authentication request.	136
ModIO.TermsOfUseLink	
Represents a url as part of the TOS. The 'required' field can be used to determine whether or not it is a TOS requirement to be displayed to the end user when viewing the TOS text.	137
ModIO.UserInstalledMod	
Struct used to represent a mod that already exists on the current device. You can view the subscribed users to this mod as well as the directory and modprofile associated to it.	137
ModIO.UserProfile	
Represents a particular mod.io user with their username, DownloadReferences for getting their avatar, as well as their language and timezone.	138
ModIO.Util.Utility	139

Chapter 7

Namespace Documentation

7.1 ModIO Namespace Reference

Classes

- class [AuthenticationServiceProviderExtensions](#)
- class [BuildSettings](#)

Build-specific configuration values. This can be setup directly from the inspector when editing the config settings file, or you can instantiate and use this at runtime with the Initialize method
- struct [CommentPage](#)

A struct containing the ModComments and total number of remaining results that can be acquired with the [SearchFilter](#) used in the GetMods request.
- class [CreationToken](#)

This is used with creating new mod profiles. Using a token ensures you dont create duplicate profiles.
- struct [DownloadReference](#)

Used in ModIOUnity.DownloadTexture() to get the Texture. ([DownloadReference](#) is serializable with Unity's [JsonUtility](#))
- interface [IModIoWebRequest](#)
- struct [InstalledMod](#)

Struct used to represent a mod that already exists on the current device. You can view the subscribed users to this mod as well as the directory and modprofile associated to it.
- struct [ModComment](#)
- class [ModfileDetails](#)
- struct [ModId](#)

A struct representing the globally unique identifier for a specific mod profile.
- class [ModIOUnity](#)

Main interface for the mod.io Unity plugin. Every method within ModIOUnity.cs that has a callback can also be found in [ModIOUnityAsync](#) with an asynchronous alternative method (if you'd rather not use callbacks)
- class [ModIOUnityAsync](#)

Main async interface for the mod.io Unity plugin. Every method within ModIOUnity.cs that has a callback can also be found in [ModIOUnityAsync](#) with an asynchronous alternative method (if you'd rather not use callbacks).
- class [ModIoWebRequest](#)
- struct [ModPage](#)

A struct containing the ModProfiles and total number of remaining results that can be acquired with the [SearchFilter](#) used in the GetMods request.
- struct [ModProfile](#)

A struct representing all of the information available for a [ModProfile](#).

- class [ModProfileDetails](#)
Use this class to fill out the details of a Mod Profile that you'd like to create or edit. If you're submitting this via `CreateModProfile` you must assign values to logo, name and summary, otherwise the submission will be rejected (All fields except modId are optional if submitting this via `EditModProfile`)
- struct [ModStats](#)
Detailed stats about a Mod's ratings, downloads, subscribers, popularity etc
- class [ProgressHandle](#)
A [ProgressHandle](#) can only be used to monitor the progress of an operation and cannot be used to cancel or suspend ongoing operations. The `OperationType` enum field specifies what type of operation this handle is for. The `Progress` field can be used to get the percentage (0.0 - 1.0) of the progress. The `Completed` and `Failed` fields can be used to determine if the operation is complete and whether or not it failed.
- class [Report](#)
Used in conjunction with [ModIOUnity.Report\(\)](#) to send a report to the mod.io server for a specific mod.
- struct [Result](#)
Struct returned from [ModIO](#) callbacks to inform the caller if the operation succeeded.
- class [ResultAnd](#)
Convenience wrapper for essentially a `Tuple`.
- class [SearchFilter](#)
Used to build a filter that is sent with requests for retrieving mods.
- struct [ServerSettings](#)
Describes the server settings to use for the [ModIO](#) Plugin. This can be setup directly from the inspector when editing the config settings file, or you can instantiate and use this at runtime with the `Initialize` method
- struct [SubscribedMod](#)
Represents the [ModProfile](#) of a mod the current user has subscribed to. Contains the status and a directory (if installed) and the associated [ModProfile](#).
- struct [Tag](#)
Represents a [Tag](#) that can be assigned to a mod.
- struct [TagCategory](#)
Represents a particular category of tags.
- struct [TermsHash](#)
This is the hash that identifies the TOS. Used to validate the TOS requirement when attempting to authenticate a user.
- struct [TermsOfUse](#)
TOS object received from a successful use of [ModIOUnity.GetTermsOfUse](#) This is used when attempting to authenticate via a third party. You must retrieve the TOS and input it along with an authentication request.
- struct [TermsOfUseLink](#)
Represents a url as part of the TOS. The 'required' field can be used to determine whether or not it is a TOS requirement to be displayed to the end user when viewing the TOS text.
- struct [UserInstalledMod](#)
Struct used to represent a mod that already exists on the current device. You can view the subscribed users to this mod as well as the directory and modprofile associated to it.
- struct [UserProfile](#)
Represents a particular mod.io user with their username, `DownloadReferences` for getting their avatar, as well as their language and timezone.

Enumerations

- enum **AuthenticationServiceProvider**
- enum **AvatarSize**
- enum **CommunityOptions**
- enum **ContentWarnings**
- enum [LogLevel](#)
The logging level of the plugin. Used in `BuildSettings` to determine which log messages to ignore or display.

- enum **ModManagementEventType**
- enum **ModManagementOperationType**
- enum **ModRating**
- enum **ModStatus**
- enum **OculusDevice**
- enum **PlayStationEnvironment**
- enum **ReportType**
- enum [SortModsBy](#)
Category to be used in the [SearchFilter](#) for determining how mods should be filtered in a request.
- enum [SubscribedModStatus](#)
The current state of a subscribed mod. Useful for checking whether or not a mod has been installed yet or if there was a problem trying to download/install it.
- enum [UserPortal](#)
Values representing the valid User Portals that mod.io works with. Used when setting up [BuildSettings](#).

Functions

- delegate void **LogMessageDelegate** ([LogLevel](#) logLevel, string logMessage)
Logging delegate that can be assigned via [ModIOUnity.SetLogMessageDelegate](#).
- delegate void **ModManagementEventDelegate** (ModManagementEventType eventType, [ModId](#) modId, [Result](#) eventResult)
A delegate that gets invoked each time a new ModManagement event happens (download, install, subscribe, etc)

7.1.1 Enumeration Type Documentation

7.1.1.1 SortModsBy

enum [ModIO.SortModsBy](#)

Category to be used in the [SearchFilter](#) for determining how mods should be filtered in a request.

See also

[SearchFilter](#), [ModIOUnity.GetMods](#), [ModIOUnityAsync.GetMods](#)

7.1.1.2 SubscribedModStatus

enum [ModIO.SubscribedModStatus](#)

The current state of a subscribed mod. Useful for checking whether or not a mod has been installed yet or if there was a problem trying to download/install it.

See also

[SubscribedMod](#)

7.1.1.3 UserPortal

enum [ModIO.UserPortal](#)

Values representing the valid User Portals that mod.io works with. Used when setting up [BuildSettings](#).

See also

[BuildSettings](#)

7.2 ModIO.Implementation Namespace Reference

Classes

- class [InstalledModExtensions](#)

7.3 ModIO.Implementation.API Namespace Reference

7.4 ModIO.Implementation.API.Objects Namespace Reference

Classes

- struct [ModDependencies](#)
A struct representing all of the information available for a Mod's Dependencies.
- struct [ModDependenciesObject](#)
A struct representing all of the information available for a [ModDependenciesObject](#).
- struct [Rating](#)
A struct representing all of the information available for a [Rating](#).
- struct [RatingObject](#)
A struct representing all of the information available for a [ModDependenciesObject](#).

7.5 ModIO.Util Namespace Reference

Classes

- interface [ISimpleMessage](#)
- interface [ISimpleMonoSingleton](#)
- class **MessagePoke**
- class [MonoDispatcher](#)
- class [MonoSingleton](#)
- class [Mutex](#)
This serves only as an abstract handle for using lock(mutex) to synchronize IO operations
- class **PrefabPool**
- class [SelfInstancingMonoSingleton](#)
- class [SimpleMessageHub](#)
- class **SimpleMessageHubTester**
- class [SimpleMessageUnsubscribeToken](#)
- class [SimpleSingleton](#)
- class [Utility](#)

7.6 Plugins Namespace Reference

7.7 Plugins.mod Namespace Reference

7.8 Plugins.mod.io Namespace Reference

7.9 Plugins.mod.io.Runtime Namespace Reference

7.10 Plugins.mod.io.Runtime.Utility Namespace Reference

Classes

- class [CoroutineRunner](#)

Chapter 8

Class Documentation

8.1 ModIO.AuthenticationServiceProviderExtensions Class Reference

Static Public Member Functions

- static string **GetProviderName** (this AuthenticationServiceProvider provider)
- static string **GetTokenFieldName** (this AuthenticationServiceProvider provider)

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Enums/AuthenticationServiceProvider.cs

8.2 ModIO.BuildSettings Class Reference

Build-specific configuration values. This can be setup directly from the inspector when editing the config settings file, or you can instantiate and use this at runtime with the Initialize method

Public Member Functions

- **BuildSettings** ([BuildSettings](#) buildSettings)

Public Attributes

- [LogLevel](#) **logLevel**
Level to log at.
- [UserPortal](#) **userPortal**
Portal the game will be launched through.
- uint **requestCacheLimitKB**
Size limit for the request cache.

8.2.1 Detailed Description

Build-specific configuration values. This can be setup directly from the inspector when editing the config settings file, or you can instantiate and use this at runtime with the Initialize method

See also

[ServerSettings](#), [ModIOUnity.InitializeForUser](#), [ModIOUnityAsync.InitializeForUser](#)

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/BuildSettings.cs

8.3 CommentDetails Class Reference

Public Member Functions

- **CommentDetails** (long [replyId](#), string [content](#))

Public Attributes

- long **replyId**
Id of the parent comment this comment is replying to (can be 0 if the comment is not a reply).
- string **content**
Contents of the comment.

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/CommentDetails.cs

8.4 ModIO.CommentPage Struct Reference

A struct containing the ModComments and total number of remaining results that can be acquired with the [SearchFilter](#) used in the GetMods request.

Public Attributes

- [ModComment\[\]](#) [CommentObjects](#)
The mod profiles retrieved from this pagination request
- long [totalSearchResultsFound](#)
the total results that could be found. eg there may be a total of 1,000 comments but this [CommentPage](#) may only contain the first 100, depending on the [SearchFilter](#) pagination settings.

8.4.1 Detailed Description

A struct containing the ModComments and total number of remaining results that can be acquired with the [SearchFilter](#) used in the GetMods request.

See also

[ModIOUnity.GetModComments](#), [ModIOUnityAsync.GetModComments](#)

8.4.2 Member Data Documentation

8.4.2.1 CommentObjects

`ModComment [] ModIO.CommentPage.CommentObjects`

The mod profiles retrieved from this pagination request

See also

[ModIOUnity.GetModComments](#), [ModIOUnityAsync.GetModComments](#)

8.4.2.2 totalSearchResultsFound

`long ModIO.CommentPage.totalSearchResultsFound`

the total results that could be found. eg there may be a total of 1,000 comments but this [CommentPage](#) may only contain the first 100, depending on the [SearchFilter](#) pagination settings.

See also

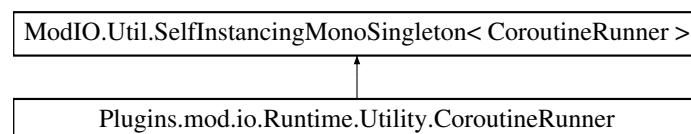
[SearchFilter](#), [SearchFilter.SetPageIndex](#), [SearchFilter.SetPageSize](#), [ModIOUnity.GetModComments](#), [ModIOUnityAsync.GetModComments](#)

The documentation for this struct was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/CommentPage.cs

8.5 Plugins.mod.io.Runtime.Utility.CoroutineRunner Class Reference

Inheritance diagram for Plugins.mod.io.Runtime.Utility.CoroutineRunner:



Public Member Functions

- Coroutine **Run** (IEnumerator coroutine)

Additional Inherited Members

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Utility/CoroutineRunner.cs

8.6 ModIO.CreationToken Class Reference

This is used with creating new mod profiles. Using a token ensures you dont create duplicate profiles.

Private Attributes

- string **creationTokenFileHash**

8.6.1 Detailed Description

This is used with creating new mod profiles. Using a token ensures you dont create duplicate profiles.

See also

[ModIOUnity.GenerateCreationToken](#), [ModIOUnityAsync.CreateModProfile](#), [ModIOUnity.CreateModProfile](#)

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Classes/CreationToken.cs

8.7 ModIO.DownloadReference Struct Reference

Used in `ModIOUnity.DownloadTexture()` to get the Texture. ([DownloadReference](#) is serializable with Unity's `JsonUtility`)

Public Member Functions

- bool [IsValid](#) ()

Check if there is a valid url for this image. You may want to check this before using the `ModIOUnity.DownloadTexture` method.

Public Attributes

- [ModId](#) **modId**
- string **url**
- string **filename**

8.7.1 Detailed Description

Used in `ModIOUnity.DownloadTexture()` to get the Texture. ([DownloadReference](#) is serializable with Unity's `JsonUtility`)

See also

`ModIOUnity.DownloadTexture`, `ModIOUnityAsync.DownloadTexture`

8.7.2 Member Function Documentation

8.7.2.1 IsValid()

```
bool ModIO.DownloadReference.IsValid ( )
```

Check if there is a valid url for this image. You may want to check this before using the `ModIOUnity.DownloadTexture` method.

See also

`ModIOUnity.DownloadTexture`, `ModIOUnityAsync.DownloadTexture`

Returns

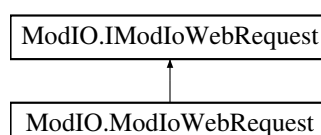
true if the url isn't null

The documentation for this struct was generated from the following file:

- `Assets/Plugins/mod.io/Runtime/Structs/DownloadReference.cs`

8.8 ModIO.IModIoWebRequest Interface Reference

Inheritance diagram for `ModIO.IModIoWebRequest`:



Public Member Functions

- string **GetResponseHeader** (string name)

Properties

- bool **isDone** [get]
- ulong **downloadedBytes** [get]
- ulong **uploadedBytes** [get]
- float **downloadProgress** [get]
- float **uploadProgress** [get]

The documentation for this interface was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Interfaces/IModIoWebRequest.cs

8.9 ModIO.InstalledMod Struct Reference

Struct used to represent a mod that already exists on the current device. You can view the subscribed users to this mod as well as the directory and modprofile associated to it.

Public Attributes

- List< long > **subscribedUsers**
The usernames of all the known users on this device that are subscribed to this mod
- bool **updatePending**
Whether or not the mod has been marked for an update
- string **directory**
the directory of where this mod is installed
- string **metadata**
The metadata for the version of the mod that is currently installed (Not to be mistaken with the metadata located inside of ModProfile.cs)
- string **version**
the version of this installed mod
- string **changeLog**
the change log for this version of the installed mod
- DateTime **dateAdded**
The date that this version of the mod was submitted to mod.io
- [ModProfile](#) **modProfile**
The profile of this mod, including the summary and name
- bool **enabled**
Whether the mod has been marked as enabled or disabled by the user

8.9.1 Detailed Description

Struct used to represent a mod that already exists on the current device. You can view the subscribed users to this mod as well as the directory and modprofile associated to it.

See also

[ModIOUnity.GetSystemInstalledMods](#), [ModProfile](#)

8.9.2 Member Data Documentation

8.9.2.1 enabled

```
bool ModIO.InstalledMod.enabled
```

Whether the mod has been marked as enabled or disabled by the user

See also

`ModIOUnity.EnableMod`, `ModIOUnity.DisableMod`

The documentation for this struct was generated from the following file:

- `Assets/Plugins/mod.io/Runtime/Structs/InstalledMod.cs`

8.10 ModIO.Implementation.InstalledModExtensions Class Reference

Static Public Member Functions

- static `UserInstalledMod AsInstalledModsUser` (this `InstalledMod` mod, long `userId`)

The documentation for this class was generated from the following file:

- `Assets/Plugins/mod.io/Runtime/Structs/InstalledModExtensions.cs`

8.11 ModIO.Util.ISimpleMessage Interface Reference

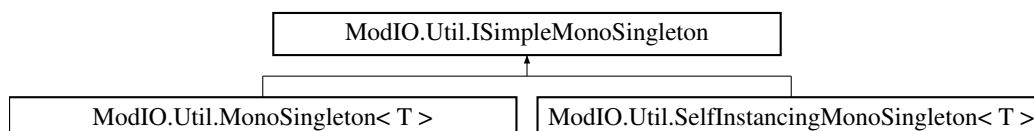
Inherited by `ModIO.Util.MessagePoke`.

The documentation for this interface was generated from the following file:

- `Assets/Plugins/mod.io/Runtime/Utility/SimpleMessageHub.cs`

8.12 ModIO.Util.ISimpleMonoSingleton Interface Reference

Inheritance diagram for `ModIO.Util.ISimpleMonoSingleton`:



Public Member Functions

- void **SetupSingleton** ()

The documentation for this interface was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Utility/SelfInstantiatingMonoSingleton.cs

8.13 ModIO.ModComment Struct Reference

Public Attributes

- long **id**
Unique id of the comment.
- [ModId](#) **modId**
Unique id of the parent mod. This is now deprecated and will be removed in future API versions, please use `resource_id` instead.
- long **resourceId**
Unique id of the resource.
- long **submittedBy**
Unique id of the user who posted the comment.
- long **dateAdded**
Unix timestamp of date comment was posted.
- string [threadPosition](#)
Levels of nesting in a comment thread. You should order by this field, to maintain comment grouping. How it works:
- long **karma**
Karma received for the comment (can be positive or negative).
- [CommentDetails](#) **commentDetails**

8.13.1 Member Data Documentation

8.13.1.1 threadPosition

```
string ModIO.ModComment.threadPosition
```

Levels of nesting in a comment thread. You should order by this field, to maintain comment grouping. How it works:

- The first comment will have the position '01'.
- The second comment will have the position '02'.

If someone responds to the second comment the position will be '02.01'.

- A maximum of 3 levels is supported.

The documentation for this struct was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/ModComment.cs

8.14 ModIO.Implementation.API.Objects.ModDependencies Struct Reference

A struct representing all of the information available for a Mod's Dependencies.

Public Attributes

- [ModId](#) **modId**
- string **modName**
- DateTime **dateAdded**

8.14.1 Detailed Description

A struct representing all of the information available for a Mod's Dependencies.

See also

[ModIOUnity.GetModDependencies](#), [ModIOUnityAsync.GetModDependencies](#), [ModDependenciesObject](#)

The documentation for this struct was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/ModDependencies.cs

8.15 ModIO.Implementation.API.Objects.ModDependenciesObject Struct Reference

A struct representing all of the information available for a [ModDependenciesObject](#).

Public Attributes

- int **mod_id**
- string **mod_name**
- int **date_added**

8.15.1 Detailed Description

A struct representing all of the information available for a [ModDependenciesObject](#).

See also

[ModIOUnity.GetModDependencies](#), [ModIOUnityAsync.GetModDependencies](#)

The documentation for this struct was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/ModDependenciesObject.cs

8.16 ModIO.ModfileDetails Class Reference

Public Attributes

- [ModId?](#) **modId**
[ModId](#) of the mod that you wish to upload the modfile to. (Must be assigned)
- string **directory**
The directory containing all of the files that makeup the mod. The directory and all of its contents will be compressed and uploaded when submitted via [ModIOUnity.UploadModfile](#).
- string **changelog**
the changelog for this file version of the mod.
- string **version**
The version number of this modfile as a string (eg 0.2.11)
- string [metadata](#)
Your own custom metadata that can be uploaded with the modfile.

8.16.1 Member Data Documentation

8.16.1.1 metadata

```
string ModIO.ModfileDetails.metadata
```

Your own custom metadata that can be uploaded with the modfile.

the metadata has a maximum size of 50,000 characters.

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Classes/ModfileDetails.cs

8.17 ModIO.ModId Struct Reference

A struct representing the globally unique identifier for a specific mod profile.

Public Member Functions

- **ModId** (long id)

Static Public Member Functions

- static implicit **operator long** ([ModId](#) id)
- static **operator ModId** (long id)

Static Public Attributes

- static readonly [ModId](#) **Null** = new [ModId](#)(0L)

Properties

- long **id** [get, set]

Private Attributes

- long **_id**

8.17.1 Detailed Description

A struct representing the globally unique identifier for a specific mod profile.

The documentation for this struct was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/ModID.cs

8.18 ModIO.ModIOUnity Class Reference

Main interface for the mod.io Unity plugin. Every method within ModIOUnity.cs that has a callback can also be found in [ModIOUnityAsync](#) with an asynchronous alternative method (if you'd rather not use callbacks)

Static Public Member Functions

- static bool [IsInitialized](#) ()
You can use this to quickly identify whether or not the plugin has been initialized.
- static void [SetLoggingDelegate](#) ([LogMessageDelegate](#) loggingDelegate)
*Assigns the logging delegate the plugin uses to output log messages that otherwise go to UnityEngine.Debug.↔
Log(string)*
- static [Result](#) [InitializeForUser](#) (string userProfileIdentifier, [ServerSettings](#) serverSettings, [BuildSettings](#) buildSettings)
Initializes the Plugin using the provided settings for a specified user. Loads the local state of mods installed on the system as well as relevant mods to the user. Loads the state of mods installed on the system as well as the set of mods the specified user has installed on this device.
- static [Result](#) [InitializeForUser](#) (string userProfileIdentifier)
Initializes the Plugin using the provided settings for a specified user. Loads the local state of mods installed on the system as well as relevant mods to the user. Loads the state of mods installed on the system as well as the set of mods the specified user has installed on this device.
- static void [Shutdown](#) (Action shutdownComplete)
Cancels any running public operations, frees plugin resources, and invokes any pending callbacks with a cancelled result code.
- static void [RequestAuthenticationEmail](#) (string emailAddress, Action< [Result](#) > callback)
Sends an email with a security code to the specified Email Address. The security code is then used to Authenticate the user session using [ModIOUnity.SubmitEmailSecurityCode\(\)](#)
- static void [SubmitEmailSecurityCode](#) (string securityCode, Action< [Result](#) > callback)

- Attempts to Authenticate the current session by submitting a security code received by email from [ModIOUnity.RequestAuthenticationEmail](#)*
- static void [GetTermsOfUse](#) (Action< [ResultAnd](#)< [TermsOfUse](#) > > callback)
This retrieves the terms of use text to be shown to the user to accept/deny before authenticating their account via a third party provider, eg steam or google.
 - static void [AuthenticateUserViaSteam](#) (string steamToken, string emailAddress, [TermsHash](#)? hash, Action< [Result](#) > callback)
Attempts to authenticate a user via the steam API.
 - static void [AuthenticateUserViaEpic](#) (string epicToken, string emailAddress, [TermsHash](#)? hash, Action< [Result](#) > callback)
Attempts to authenticate a user via the epic API.
 - static void [AuthenticateUserViaGOG](#) (string gogToken, string emailAddress, [TermsHash](#)? hash, Action< [Result](#) > callback)
Attempts to authenticate a user via the GOG API.
 - static void [AuthenticateUserViaPlayStation](#) (string authCode, string emailAddress, [TermsHash](#)? hash, PlayStationEnvironment environment, Action< [Result](#) > callback)
Attempts to authenticate a user via the GOG API.
 - static void [AuthenticateUserViaItch](#) (string itchToken, string emailAddress, [TermsHash](#)? hash, Action< [Result](#) > callback)
Attempts to authenticate a user via the Itch.io API.
 - static void [AuthenticateUserViaXbox](#) (string xboxToken, string emailAddress, [TermsHash](#)? hash, Action< [Result](#) > callback)
Attempts to authenticate a user via the Xbox API.
 - static void [AuthenticateUserViaSwitch](#) (string SwitchNsald, string emailAddress, [TermsHash](#)? hash, Action< [Result](#) > callback)
Attempts to authenticate a user via the switch API.
 - static void [AuthenticateUserViaDiscord](#) (string discordToken, string emailAddress, [TermsHash](#)? hash, Action< [Result](#) > callback)
Attempts to authenticate a user via the Discord API.
 - static void [AuthenticateUserViaGoogle](#) (string googleToken, string emailAddress, [TermsHash](#)? hash, Action< [Result](#) > callback)
Attempts to authenticate a user via the Google API.
 - static void [AuthenticateUserViaOculus](#) (OculusDevice oculusDevice, string nonce, long userId, string oculusToken, string emailAddress, [TermsHash](#)? hash, Action< [Result](#) > callback)
Attempts to authenticate a user via the Oculus API.
 - static void [IsAuthenticated](#) (Action< [Result](#) > callback)
Informs you if the current user session is authenticated or not.
 - static [Result](#) [LogOutCurrentUser](#) ()
De-authenticates the current Mod.io user for the current session and clears all user-specific data stored on the current device. Installed mods that do not have other local users subscribed will be uninstalled if [ModIOUnity.EnableModManagement\(\)](#) has been used to enable the mod management system. (If ModManagement is enabled).
 - static void [GetTagCategories](#) (Action< [ResultAnd](#)< [TagCategory](#)[] > > callback)
Gets the existing tags for the current game Id that can be used when searching/filtering mods.
 - static void [GetMods](#) ([SearchFilter](#) filter, Action< [ResultAnd](#)< [ModPage](#) > > callback)
Uses a [SearchFilter](#) to retrieve a specific Mod Page and returns the ModProfiles and total number of mods based on the Search Filter.
 - static void [GetMod](#) (ModId modId, Action< [ResultAnd](#)< [ModProfile](#) > > callback)
Requests a single [ModProfile](#) from the mod.io server by its [ModId](#).
 - static void [GetModComments](#) (ModId modId, [SearchFilter](#) filter, Action< [ResultAnd](#)< [CommentPage](#) > > callback)
Get all comments posted in the mods profile. Successful request will return an array of Comment Objects. We recommended reading the filtering documentation to return only the records you want.
 - static void [GetModDependencies](#) (ModId modId, Action< [ResultAnd](#)< [ModDependencies](#)[] > > callback)

- Retrieves a list of ModDependenciesObjects that represent mods that depend on a mod.*

 - static void [GetCurrentUserRatings](#) (Action< [ResultAnd](#)< [Rating](#)[] > > callback)

Get all mod rating's submitted by the authenticated user. Successful request will return an array of Rating Objects.
- static void [GetCurrentUserRatingFor](#) (ModId modId, Action< [ResultAnd](#)< ModRating > > callback)

Gets the rating that the current user has given for a specified mod. You must have an authenticated session for this to be successful.
- static void [RateMod](#) (ModId modId, ModRating rating, Action< [Result](#) > callback)

Used to submit a rating for a specified mod.
- static void [SubscribeToMod](#) (ModId modId, Action< [Result](#) > callback)

Adds the specified mod to the current user's subscriptions.
- static void [UnsubscribeFromMod](#) (ModId modId, Action< [Result](#) > callback)

Removes the specified mod from the current user's subscriptions.
- static [SubscribedMod](#)[] [GetSubscribedMods](#) (out [Result](#) result)

Retrieves all of the subscribed mods for the current user.
- static void [GetCurrentUser](#) (Action< [ResultAnd](#)< [UserProfile](#) > > callback)

Gets the current user's [UserProfile](#) struct. Containing their mod.io username, user id, language, timezone and download references for their avatar.
- static void [MuteUser](#) (long userId, Action< [Result](#) > callback)

Mutes a user which effectively hides any content from that specified user
- static void [UnmuteUser](#) (long userId, Action< [Result](#) > callback)

Un-mutes a user which effectively reveals previously hidden content from that user
- static void [FetchUpdates](#) (Action< [Result](#) > callback)

This retrieves the user's subscriptions from the mod.io server and synchronises it with our local instance of the user's subscription data. If mod management has been enabled via [ModIOUnity.EnableModManagement\(\)](#) then it may begin to install/uninstall mods.
- static [Result](#) [EnableModManagement](#) ([ModManagementEventDelegate](#) modManagementEventDelegate)

Enables the mod management system. When enabled it will automatically download, install, update and delete mods according to the authenticated user's subscriptions.
- static [Result](#) [DisableModManagement](#) ()

Disables the mod management system and cancels any ongoing jobs for downloading or installing mods.
- static [ProgressHandle](#) [GetCurrentModManagementOperation](#) ()

Returns a [ProgressHandle](#) with information on the current mod management operation.
- static [InstalledMod](#)[] [GetSystemInstalledMods](#) (out [Result](#) result)

Gets an array of mods that are installed on the current device.
- static [UserInstalledMod](#)[] [GetInstalledModsForUser](#) (out [Result](#) result, bool includeDisabledMods=false)

Gets an array of mods that are installed for the current user.
- static [Result](#) [ForceUninstallMod](#) (ModId modId)

This informs the mod management system that this mod should be uninstalled if not subscribed by the current user. (such as a mod installed by a different user not currently active).
- static bool [IsModManagementBusy](#) ()

Checks if the automatic management process is currently awake and performing a mod management operation, such as installing, downloading, uninstalling, updating.
- static bool **EnableMod** (ModId modId)
- static bool **DisableMod** (ModId modId)
- static void [AddDependenciesToMod](#) (ModId modId, ICollection< [ModId](#) > dependencies, Action< [Result](#) > callback)

Adds the specified mods as dependencies to an existing mod.
- static void [RemoveDependenciesFromMod](#) (ModId modId, ICollection< [ModId](#) > dependencies, Action< [Result](#) > callback)

Removes the specified mods as dependencies for another existing mod.
- static [CreationToken](#) [GenerateCreationToken](#) ()

Gets a token that can be used to create a new mod profile on the mod.io server.

- static void [CreateModProfile](#) ([CreationToken](#) token, [ModProfileDetails](#) modProfileDetails, Action< [ResultAnd](#)< [ModId](#) > > callback)
Creates a new mod profile on the mod.io server based on the details provided from the [ModProfileDetails](#) object provided. Note that you must have a logo, name and summary assigned in [ModProfileDetails](#) in order for this to work.
- static void [EditModProfile](#) ([ModProfileDetails](#) modProfile, Action< [Result](#) > callback)
This is used to edit or change data (except images) in an existing mod profile on the mod.io server. If you want to add or edit images, use [UploadModMedia](#).
- static [ProgressHandle](#) [GetCurrentUploadHandle](#) ()
This will return null if no upload operation is currently being performed.
- static void [UploadModfile](#) ([ModfileDetails](#) modfile, Action< [Result](#) > callback)
Used to upload a mod file to a mod profile on the mod.io server. A mod file is the actual archive of a mod. This method can be used to update a mod to a newer version (you can include changelog information in [ModfileDetails](#)).
- static void [UploadModMedia](#) ([ModProfileDetails](#) modProfileDetails, Action< [Result](#) > callback)
This is used to update the logo of a mod or the gallery images. This works very similar to [EditModProfile](#) except it only affects the images.
- static void [ArchiveModProfile](#) ([ModId](#) modId, Action< [Result](#) > callback)
Removes a mod from being visible on the mod.io server.
- static void [GetCurrentUserCreations](#) ([SearchFilter](#) filter, Action< [ResultAnd](#)< [ModPage](#) > > callback)
Get all mods the authenticated user added or is a team member of. Successful request will return an array of Mod Objects. We recommended reading the filtering documentation to return only the records you want.
- static void [AddTags](#) ([ModId](#) modId, string[] tags, Action< [Result](#) > callback)
Adds the provided tags to the specified mod id. In order for this to work the authenticated user must have permission to edit the specified mod. Only existing tags as part of the game Id will be added.
- static void [AddModComment](#) ([ModId](#) modId, [CommentDetails](#) commentDetails, Action< [ResultAnd](#)< [ModComment](#) > > callback)
Adds a comment to a mod profile. Successfully adding a comment returns the Mod Comment object back.
- static void [DeleteModComment](#) ([ModId](#) modId, long commentId, Action< [Result](#) > callback)
Delete a comment from a mod profile. Successful request will return 204 No Content and fire a MOD_COMMENT←_DELETED event.
- static void [UpdateModComment](#) ([ModId](#) modId, string content, long commentId, Action< [ResultAnd](#)< [ModComment](#) > > callback)
Update a comment for the corresponding mod. Successful request will return the updated Comment Object.
- static void [DeleteTags](#) ([ModId](#) modId, string[] tags, Action< [Result](#) > callback)
Deletes the specified tags from the mod. In order for this to work the authenticated user must have permission to edit the specified mod.
- static void [DownloadImage](#) ([DownloadReference](#) downloadReference, Action< [ResultAnd](#)< byte[] > > callback)
Downloads a texture based on the specified download reference.
- static void [Report](#) ([Report](#) report, Action< [Result](#) > callback)
Reports a specified mod to mod.io.

8.18.1 Detailed Description

Main interface for the mod.io Unity plugin. Every method within ModIOUnity.cs that has a callback can also be found in [ModIOUnityAsync](#) with an asynchronous alternative method (if you'd rather not use callbacks)

See also

[ModIOUnityAsync](#)

8.18.2 Member Function Documentation

8.18.2.1 AddDependenciesToMod()

```
static void ModIO.ModIOUnity.AddDependenciesToMod (
    ModId modId,
    ICollection< ModId > dependencies,
    Action< Result > callback ) [static]
```

Adds the specified mods as dependencies to an existing mod.

If the dependencies already exist they will be ignored and the result will return success

Parameters

<i>modId</i>	ModId of the mod you want to add dependencies to
<i>dependencies</i>	The ModIds that you want to add (max 5 at a time)
<i>callback</i>	callback with the result of the request

See also

[Result](#), [ModId](#), [ModIOUnity.RemoveDependenciesFromMod](#), [ModIOUnityAsync.RemoveDependenciesFromMod](#), [ModIOUnityAsync.AddDependenciesToMod](#)

```
void Example()
{
    var dependencies = new List<ModId>()
    {
        (ModId)1001,
        (ModId)1002,
        (ModId)1003
    };
    ModIOUnity.AddDependenciesToMod(mod.id, dependencies, AddDependenciesCallback);
}
void AddDependenciesCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Successfully added dependencies to mod");
    }
    else
    {
        Debug.Log("Failed to add dependencies to mod");
    }
}
```

8.18.2.2 AddModComment()

```
static void ModIO.ModIOUnity.AddModComment (
    ModId modId,
    CommentDetails commentDetails,
    Action< ResultAnd< ModComment > > callback ) [static]
```

Adds a comment to a mod profile. Successfully adding a comment returns the Mod Comment object back.

Keep in mind you can use mentions in the comment content, such as "Hello there, @<john-doe>"

Parameters

<i>modId</i>	Id of the mod to add the comment to
<i>commentDetails</i>	the new comment to be added
<i>callback</i>	callback with the result of the operation

See also

[Result](#), [ResultAnd](#), [ModComment](#), [CommentDetails](#), [GetModComments](#), [DeleteModComment](#), [EditModComment](#), [ModIOUnityAsync.AddModComment](#)

```
ModId modId;
void Example()
{
    CommentDetails comment = new CommentDetails(0, "Hello world!");
    ModIOUnity.AddModComment(modId, comment, AddCommentCallback);
}
void AddCommentCallback(ResultAnd<ModComment> response)
{
    if (response.result.Succeeded())
    {
        Debug.Log("added comment");
    }
    else
    {
        Debug.Log("failed to add comment");
    }
}
```

8.18.2.3 AddTags()

```
static void ModIO.ModIOUnity.AddTags (
    ModId modId,
    string[] tags,
    Action< Result > callback ) [static]
```

Adds the provided tags to the specified mod id. In order for this to work the authenticated user must have permission to edit the specified mod. Only existing tags as part of the game id will be added.

Parameters

<i>modId</i>	Id of the mod to add tags to
<i>tags</i>	array of tags to be added
<i>callback</i>	callback with the result of the operation

See also

[Result](#), [DeleteTags](#), [ModIOUnityAsync.AddTags](#)

```
ModId modId;
string[] tags;
void Example()
{
    ModIOUnity.AddTags(modId, tags, AddTagsCallback);
}
void AddTagsCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("added tags");
    }
    else
    {
        Debug.Log("failed to add tags");
    }
}
```

8.18.2.4 ArchiveModProfile()

```
static void ModIO.ModIOUnity.ArchiveModProfile (
    ModId modId,
    Action< Result > callback ) [static]
```

Removes a mod from being visible on the mod.io server.

If you want to delete a mod permanently you can do so from a web browser.

Parameters

<i>modId</i>	the id of the mod to delete
<i>callback</i>	callback with the result of the operation

See also

[Result](#), [CreateModProfile](#), [EditModProfile](#), [ModIOUnityAsync.ArchiveModProfile](#)

```
ModId modId;
void Example()
{
    ModIOUnity.ArchiveModProfile(modId, ArchiveModCallback);
}
void ArchiveModCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("archived mod profile");
    }
    else
    {
        Debug.Log("failed to archive mod profile");
    }
}
```

8.18.2.5 AuthenticateUserViaDiscord()

```
static void ModIO.ModIOUnity.AuthenticateUserViaDiscord (
    string discordToken,
    string emailAddress,
    TermsHash? hash,
    Action< Result > callback ) [static]
```

Attempts to authenticate a user via the Discord API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>discordToken</i>	the user's discord token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()
<i>callback</i>	Callback to be invoked when the operation completes

See also

[GetTermsOfUse](#), [ModIOUnityAsync.AuthenticateUserViaDiscord](#)

```
// First we get the Terms of Use to display to the user and cache the hash
void GetTermsOfUse_Example()
{
    ModIOUnity.GetTermsOfUse(GetTermsOfUseCallback);
}
void GetTermsOfUseCallback(ResultAnd#60;TermsOfUse#62; response)
{
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}
// Once we have the Terms of Use and hash we can attempt to authenticate
void Authenticate_Example()
{
    ModIOUnity.AuthenticateUserViaDiscord(discordToken, "johndoe@gmail.com", modIOTermsOfUse.hash,
        AuthenticationCallback);
}
void AuthenticationCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}
```

8.18.2.6 AuthenticateUserViaEpic()

```
static void ModIO.ModIOUnity.AuthenticateUserViaEpic (
    string epicToken,
    string emailAddress,
    TermsHash? hash,
    Action< Result > callback ) [static]
```

Attempts to authenticate a user via the epic API.

Parameters

<i>epicToken</i>	the user's epic token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()
<i>callback</i>	Callback to be invoked when the operation completes

See also

[GetTermsOfUse](#), [ModIOUnityAsync.AuthenticateUserViaEpic](#)

8.18.2.7 AuthenticateUserViaGOG()

```
static void ModIO.ModIOUnity.AuthenticateUserViaGOG (
    string gogToken,
    string emailAddress,
    TermsHash? hash,
    Action< Result > callback ) [static]
```

Attempts to authenticate a user via the GOG API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>gogToken</i>	the user's gog token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()
<i>callback</i>	Callback to be invoked when the operation completes

See also

[GetTermsOfUse](#), [ModIOUnityAsync.AuthenticateUserViaGOG](#)

```
// First we get the Terms of Use to display to the user and cache the hash
void GetTermsOfUse_Example()
{
    ModIOUnity.GetTermsOfUse(GetTermsOfUseCallback);
}
void GetTermsOfUseCallback(ResultAndTermsOfUse response)
{
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}
// Once we have the Terms of Use and hash we can attempt to authenticate
void Authenticate_Example()
{
    ModIOUnity.AuthenticateUserViaGOG(gogToken, "johndoe@gmail.com", modIOTermsOfUse.hash,
        AuthenticationCallback);
}
void AuthenticationCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}
```

8.18.2.8 AuthenticateUserViaGoogle()

```
static void ModIO.ModIOUnity.AuthenticateUserViaGoogle (
    string googleToken,
    string emailAddress,
```

```
TermsHash? hash,
Action< Result > callback ) [static]
```

Attempts to authenticate a user via the Google API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>googleToken</i>	the user's google token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()
<i>callback</i>	Callback to be invoked when the operation completes

See also

[GetTermsOfUse](#), [ModIOUnityAsync.AuthenticateUserViaGoogle](#)

```
// First we get the Terms of Use to display to the user and cache the hash
void GetTermsOfUse_Example()
{
    ModIOUnity.GetTermsOfUse(GetTermsOfUseCallback);
}
void GetTermsOfUseCallback(ResultAndTermsOfUse response)
{
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}
// Once we have the Terms of Use and hash we can attempt to authenticate
void Authenticate_Example()
{
    ModIOUnity.AuthenticateUserViaGoogle(googleToken, "johndoe@gmail.com", modIOTermsOfUse.hash,
        AuthenticationCallback);
}
void AuthenticationCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}
```

8.18.2.9 AuthenticateUserViaItch()

```
static void ModIO.ModIOUnity.AuthenticateUserViaItch (
    string itchioToken,
    string emailAddress,
    TermsHash? hash,
    Action< Result > callback ) [static]
```

Attempts to authenticate a user via the Itch.io API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>itchioToken</i>	the user's itch token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()
<i>callback</i>	Callback to be invoked when the operation completes

See also

[GetTermsOfUse](#), [ModIOUnityAsync.AuthenticateUserViaItch](#)

```
// First we get the Terms of Use to display to the user and cache the hash
void GetTermsOfUse_Example()
{
    ModIOUnity.GetTermsOfUse(GetTermsOfUseCallback);
}
void GetTermsOfUseCallback(ResultAndTermsOfUse response)
{
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}
// Once we have the Terms of Use and hash we can attempt to authenticate
void Authenticate_Example()
{
    ModIOUnity.AuthenticateUserViaItch(itchioToken, "johndoe@gmail.com", modIOTermsOfUse.hash,
        AuthenticationCallback);
}
void AuthenticationCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}
```

8.18.2.10 AuthenticateUserViaOculus()

```
static void ModIO.ModIOUnity.AuthenticateUserViaOculus (
    OculusDevice oculusDevice,
    string nonce,
    long userId,
    string oculusToken,
    string emailAddress,
    TermsHash? hash,
    Action< Result > callback ) [static]
```

Attempts to authenticate a user via the Oculus API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>oculusDevice</i>	the device your authenticating on
---------------------	-----------------------------------

Parameters

<i>nonce</i>	the nonce
<i>oculusToken</i>	the user's oculus token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()
<i>callback</i>	Callback to be invoked when the operation completes
<i>userId</i>	

See also

[GetTermsOfUse](#), [ModIOUnityAsync.AuthenticateUserViaOculus](#)

```
// First we get the Terms of Use to display to the user and cache the hash
void GetTermsOfUse_Example()
{
    ModIOUnity.GetTermsOfUse(GetTermsOfUseCallback);
}
void GetTermsOfUseCallback(ResultAndTermsOfUse response)
{
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}
// Once we have the Terms of Use and hash we can attempt to authenticate
void Authenticate_Example()
{
    ModIOUnity.AuthenticateUserViaOculus(oculusDevice.Quest,
                                         nonce,
                                         userId,
                                         oculusToken,
                                         "johndoe@gmail.com",
                                         modIOTermsOfUse.hash, AuthenticationCallback);
}
void AuthenticationCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}
```

8.18.2.11 AuthenticateUserViaPlayStation()

```
static void ModIO.ModIOUnity.AuthenticateUserViaPlayStation (
    string authCode,
    string emailAddress,
    TermsHash? hash,
    PlayStationEnvironment environment,
    Action< Result > callback ) [static]
```

Attempts to authenticate a user via the GOG API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>authCode</i>	the user's auth code
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()
<i>environment</i>	the PSN account environment
<i>callback</i>	Callback to be invoked when the operation completes

See also

[GetTermsOfUse](#), [ModIOUnityAsync.AuthenticateUserViaGOG](#)

```
// First we get the Terms of Use to display to the user and cache the hash
void GetTermsOfUse_Example()
{
    ModIOUnity.GetTermsOfUse(GetTermsOfUseCallback);
}
void GetTermsOfUseCallback(ResultAndTermsOfUse response)
{
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}
// Once we have the Terms of Use and hash we can attempt to authenticate
void Authenticate_Example()
{
    ModIOUnity.AuthenticateUserViaPlaystation(authCode, "johndoe@gmail.com", modIOTermsOfUse.hash,
        PlayStationEnvironment.np, AuthenticationCallback);
}
void AuthenticationCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}
```

8.18.2.12 AuthenticateUserViaSteam()

```
static void ModIO.ModIOUnity.AuthenticateUserViaSteam (
    string steamToken,
    string emailAddress,
    TermsHash? hash,
    Action< Result > callback ) [static]
```

Attempts to authenticate a user via the steam API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>steamToken</i>	the user's steam token
<i>emailAddress</i>	the user's email address (Can be null)
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse() (Can be null if submitted once before)
<i>callback</i>	Callback to be invoked when the operation completes

See also

[GetTermsOfUse](#), [ModIOUnityAsync.AuthenticateUserViaSteam](#)

```
// First we get the Terms of Use to display to the user and cache the hash
void GetTermsOfUse_Example()
{
    ModIOUnity.GetTermsOfUse(GetTermsOfUseCallback);
}
void GetTermsOfUseCallback(ResultAndTermsOfUse response)
{
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}
// Once we have the Terms of Use and hash we can attempt to authenticate
void Authenticate_Example()
{
    ModIOUnity.AuthenticateUserViaSteam(steamToken, "johndoe@gmail.com", modIOTermsOfUse.hash,
        AuthenticationCallback);
}
void AuthenticationCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}
```

8.18.2.13 AuthenticateUserViaSwitch()

```
static void ModIO.ModIOUnity.AuthenticateUserViaSwitch (
    string SwitchNsaId,
    string emailAddress,
    TermsHash? hash,
    Action< Result > callback ) [static]
```

Attempts to authenticate a user via the switch API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>SwitchNsaId</i>	the user's switch NSA id token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()
<i>callback</i>	Callback to be invoked when the operation completes

See also

[GetTermsOfUse](#), [ModIOUnityAsync.AuthenticateUserViaSwitch](#)

```
// First we get the Terms of Use to display to the user and cache the hash
void GetTermsOfUse_Example()
{
    ModIOUnity.GetTermsOfUse(GetTermsOfUseCallback);
```

```

}
void GetTermsOfUseCallback(ResultAndTermsOfUse response)
{
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}
// Once we have the Terms of Use and hash we can attempt to authenticate
void Authenticate_Example()
{
    ModIOUnity.AuthenticateUserViaSwitch(switchToken, "johndoe@gmail.com", modIOTermsOfUse.hash,
        AuthenticationCallback);
}
void AuthenticationCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}
}

```

8.18.2.14 AuthenticateUserViaXbox()

```

static void ModIOUnity.AuthenticateUserViaXbox (
    string xboxToken,
    string emailAddress,
    TermsHash? hash,
    Action< Result > callback ) [static]

```

Attempts to authenticate a user via the Xbox API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>xboxToken</i>	the user's xbl token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()
<i>callback</i>	Callback to be invoked when the operation completes

See also

[GetTermsOfUse](#), [ModIOUnityAsync.AuthenticateUserViaXbox](#)

```

// First we get the Terms of Use to display to the user and cache the hash
void GetTermsOfUse_Example()
{
    ModIOUnity.GetTermsOfUse(GetTermsOfUseCallback);
}
void GetTermsOfUseCallback(ResultAndTermsOfUse response)
{
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
}

```

```

    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}
// Once we have the Terms of Use and hash we can attempt to authenticate
void Authenticate_Example()
{
    ModIOUnity.AuthenticateUserViaXbox(xboxToken, "johndoe@gmail.com", modIOTermsOfUse.hash,
        AuthenticationCallback);
}
void AuthenticationCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}
}

```

8.18.2.15 CreateModProfile()

```

static void ModIO.ModIOUnity.CreateModProfile (
    CreationToken token,
    ModProfileDetails modProfileDetails,
    Action< ResultAnd< ModId > > callback ) [static]

```

Creates a new mod profile on the mod.io server based on the details provided from the [ModProfileDetails](#) object provided. Note that you must have a logo, name and summary assigned in [ModProfileDetails](#) in order for this to work.

Note that this will create a new profile on the server and can be viewed online through a browser.

Parameters

<i>token</i>	the token allowing a new unique profile to be created from ModIOUnity.GenerateCreationToken()
<i>modProfileDetails</i>	the mod profile details to apply to the mod profile being created
<i>callback</i>	a callback with the Result of the operation and the ModId of the newly created mod profile (if successful)

See also

[GenerateCreationToken](#), [CreationToken](#), [ModProfileDetails](#), [Result](#), [ModId](#), [ModIOUnityAsync.CreateModProfile](#)

```

ModId newMod;
Texture2D logo;
CreationToken token;
void Example()
{
    token = ModIOUnity.GenerateCreationToken();
    ModProfileDetails profile = new ModProfileDetails();
    profile.name = "mod name";
    profile.summary = "a brief summary about this mod being submitted"
    profile.logo = logo;
    ModIOUnity.CreateModProfile(token, profile, CreateProfileCallback);
}
void CreateProfileCallback(ResultAnd<ModId> response)
{
    if (response.result.Succeeded())
    {
        newMod = response.value;
    }
}

```



```

        Debug.Log("created new mod profile with id " + response.value.ToString());
    }
    else
    {
        Debug.Log("failed to create new mod profile");
    }
}

```

8.18.2.16 DeleteModComment()

```

static void ModIO.ModIOUnity.DeleteModComment (
    ModId modId,
    long commentId,
    Action< Result > callback ) [static]

```

Delete a comment from a mod profile. Successful request will return 204 No Content and fire a MOD_COMMENT↔_DELETED event.

Parameters

<i>modId</i>	Id of the mod to add the comment to
<i>comment↔Id</i>	The id for the comment to be removed
<i>callback</i>	callback with the result of the operation

See also

[Result](#), [ModComment](#), [CommentDetails](#), [DeleteModComment](#), [EditModComment](#), [ModIOUnityAsync.DeleteModComment](#)

```

private ModId modId;
private long commentId;
void Example()
{
    ModIOUnity.DeleteModComment(modId, commentId, DeleteCommentCallback);
}
void DeleteCommentCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("deleted comment");
    }
    else
    {
        Debug.Log("failed to delete comment");
    }
}

```

8.18.2.17 DeleteTags()

```

static void ModIO.ModIOUnity.DeleteTags (
    ModId modId,
    string[] tags,
    Action< Result > callback ) [static]

```

Deletes the specified tags from the mod. In order for this to work the authenticated user must have permission to edit the specified mod.

Parameters

<i>modId</i>	the id of the mod for deleting tags
<i>tags</i>	array of tags to be deleted
<i>callback</i>	callback with the result of the operation

See also

[Result](#), [AddTags](#), [ModIOUnityAsync.DeleteTags](#)

```
ModId modId;
string[] tags;
void Example()
{
    ModIOUnity.DeleteTags(modId, tags, DeleteTagsCallback);
}
void DeleteTagsCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("deleted tags");
    }
    else
    {
        Debug.Log("failed to delete tags");
    }
}
```

8.18.2.18 DisableModManagement()

```
static Result ModIO.ModIOUnity.DisableModManagement ( ) [static]
```

Disables the mod management system and cancels any ongoing jobs for downloading or installing mods.

```
void Example() { Result result = ModIOUnity.DisableModManagement();

if (result.Succeeded()) { Debug.Log("disabled mod management"); } else {
Debug.Log("failed to disable mod management"); } }
```

8.18.2.19 DownloadImage()

```
static void ModIO.ModIOUnity.DownloadImage (
    DownloadReference downloadReference,
    Action< ResultAnd< byte[]> > callback ) [static]
```

Downloads a texture based on the specified download reference.

You can get download references from UserProfiles and ModProfiles

Parameters

<i>downloadReference</i>	download reference for the texture (eg UserObject.avatar_100x100)
<i>callback</i>	callback with the Result and Texture2D from the download

See also

[Result](#), [DownloadReference](#), [Texture2D](#), [ModIOUnityAsync.DownloadTexture](#)

```
ModProfile mod;
void Example()
{
    ModIOUnity.DownloadTexture(mod.logoImage_320x180, DownloadTextureCallback);
}
void DownloadTextureCallback(ResultAndTexture2D response)
{
    if (response.result.Succeeded())
    {
        Debug.Log("downloaded the mod logo texture");
    }
    else
    {
        Debug.Log("failed to download the mod logo texture");
    }
}
```

8.18.2.20 EditModProfile()

```
static void ModIO.ModIOUnity.EditModProfile (
    ModProfileDetails modProfile,
    Action< Result > callback ) [static]
```

This is used to edit or change data (except images) in an existing mod profile on the mod.io server. If you want to add or edit images, use [UploadModMedia](#).

You need to assign the [ModId](#) of the mod you want to edit inside of the [ModProfileDetails](#) object included in the parameters

Parameters

<i>modProfile</i>	the mod profile details to apply to the mod profile being created
<i>callback</i>	a callback with the Result of the operation and the ModId of the newly created mod profile (if successful)

See also

[ModProfileDetails](#), [Result](#), [ModIOUnityAsync.EditModProfile](#)

```
ModId modId;
void Example()
{
    ModProfileDetails profile = new ModProfileDetails();
    profile.modId = modId;
    profile.summary = "a new brief summary about this mod being edited";
    ModIOUnity.EditModProfile(profile, EditProfileCallback);
}
void EditProfileCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("edited mod profile");
    }
    else
    {
        Debug.Log("failed to edit mod profile");
    }
}
```

8.18.2.21 EnableModManagement()

```
static Result ModIO.ModIOUnity.EnableModManagement (
    ModManagementEventDelegate modManagementEventDelegate ) [static]
```

Enables the mod management system. When enabled it will automatically download, install, update and delete mods according to the authenticated user's subscriptions.

This requires the current session to have an authenticated user, otherwise `Result.IsAuthenticationError()` from the [Result](#) will equal true.

Parameters

<i>modManagementEventDelegate</i>	A delegate that gets called everytime the ModManagement system runs an event (can be null)
-----------------------------------	--

Returns

A [Result](#) for whether or not mod management was enabled

See also

[Result](#), [DisableModManagement](#), [IsAuthenticated](#)

```
void Example()
{
    Result result = ModIOUnity.EnableModManagement (ModManagementDelegate);
}
void ModManagementDelegate (ModManagementEventType eventType, ModId modId)
{
    Debug.Log("a mod management event of type " + eventType.ToString() + " has been invoked");
}
```

8.18.2.22 FetchUpdates()

```
static void ModIO.ModIOUnity.FetchUpdates (
    Action< Result > callback ) [static]
```

This retrieves the user's subscriptions from the mod.io server and synchronises it with our local instance of the user's subscription data. If mod management has been enabled via [ModIOUnity.EnableModManagement\(\)](#) then it may begin to install/uninstall mods.

This requires the current session to have an authenticated user, otherwise `Result.IsAuthenticationError()` from the [Result](#) will equal true.

Parameters

<i>callback</i>	callback with the Result of the operation
-----------------	---

See also

[Result](#), [EnableModManagement\(ModIO.ModManagementEventDelegate\)](#), [IsAuthenticated](#), [RequestAuthenticationEmail](#), [SubmitEmailSecurityCode](#), [AuthenticateUserViaDiscord](#), [AuthenticateUserViaGoogle](#), [AuthenticateUserViaGOG](#),

[AuthenticateUserViaItch](#), [AuthenticateUserViaOculus](#), [AuthenticateUserViaSteam](#), [AuthenticateUserViaSwitch](#), [AuthenticateUserViaXbox](#), [ModIOUnity.Async.FetchUpdates](#)

```
void Example()
{
    ModIOUnity.FetchUpdates(FetchUpdatesCallback);
}
void FetchUpdatesCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("updated user subscriptions");
    }
    else
    {
        Debug.Log("failed to get user subscriptions");
    }
}
```

8.18.2.23 ForceUninstallMod()

```
static Result ModIO.ModIOUnity.ForceUninstallMod (
    ModId modId ) [static]
```

This informs the mod management system that this mod should be uninstalled if not subscribed by the current user. (such as a mod installed by a different user not currently active).

Normally if you wish to uninstall a mod you should unsubscribe and use [ModIOUnity.EnableModManagement\(\)](#) and the process will be handled automatically. However, if you want to uninstall a mod that is subscribed to a different user session this method will mark the mod to be uninstalled to free up disk space. Alternatively you can use [ModIOUnity.RemoveUserData\(\)](#) to remove a user from the local registry. If no other users are subscribed to the same mod it will be uninstalled automatically.

Parameters

<i>modId</i>	The ModId of the mod to uninstall
--------------	---

See also

[Result](#), [SubscribeToMod](#), [UnsubscribeFromMod](#), [EnableModManagement](#), [LogOutCurrentUser](#)

```
ModProfile mod;
void Example()
{
    Result result = ModIOUnity.ForceUninstallMod(mod.id);
    if (result.Succeeded())
    {
        Debug.Log("mod marked for uninstall");
    }
    else
    {
        Debug.Log("failed to mark mod for uninstall");
    }
}
```

8.18.2.24 GenerateCreationToken()

```
static CreationToken ModIO.ModIOUnity.GenerateCreationToken ( ) [static]
```

Gets a token that can be used to create a new mod profile on the mod.io server.

Returns

a [CreationToken](#) used in [ModIOUnity.CreateModProfile\(\)](#)

See also

[CreationToken](#), [ModProfileDetails](#), [Result](#), [ModId](#), [CreateModProfile](#), [EditModProfile](#)

```
void Example()
{
    CreationToken token = ModIOUnity.GenerateCreationToken();
}
```

8.18.2.25 GetCurrentModManagementOperation()

```
static ProgressHandle ModIO.ModIOUnity.GetCurrentModManagementOperation ( ) [static]
```

Returns a [ProgressHandle](#) with information on the current mod management operation.

Returns

Optional [ProgressHandle](#) object containing information regarding the progress of the operation. Null if no operation is running

See also

[ProgressHandle](#), [EnableModManagement](#)

```
void Example()
{
    ProgressHandle handle = ModIOUnity.GetCurrentModManagementOperation();
    if (handle != null)
    {
        Debug.Log("current mod management operation is " + handle.OperationType.ToString());
    }
    else
    {
        Debug.Log("no current mod management operation");
    }
}
```

8.18.2.26 GetCurrentUploadHandle()

```
static ProgressHandle ModIO.ModIOUnity.GetCurrentUploadHandle ( ) [static]
```

This will return null if no upload operation is currently being performed.

Uploads are not handled by the mod management system, these are handled separately.

Returns

A [ProgressHandle](#) informing the upload state and progress. Null if no upload operation is running.

See also

[UploadModfile](#), [ArchiveModProfile](#)

```
void Example()
{
    ProgressHandle handle = ModIOUnity.GetCurrentUploadHandle();
    if (handle != null)
    {
        Debug.Log("Current upload progress is: " + handle.Progress.ToString());
    }
}
```

8.18.2.27 GetCurrentUser()

```
static void ModIO.ModIOUnity.GetCurrentUser (
    Action< ResultAnd< UserProfile > > callback ) [static]
```

Gets the current user's [UserProfile](#) struct. Containing their mod.io username, user id, language, timezone and download references for their avatar.

This requires the current session to have an authenticated user, otherwise `Result.IsAuthenticationError()` from the [Result](#) will equal true.

Parameters

<i>callback</i>	callback with the Result and the UserProfile
-----------------	--

See also

[Result](#), [UserProfile](#), [IsAuthenticated](#), [ModIOUnityAsync.GetCurrentUser](#)

```
void Example()
{
    ModIOUnity.GetCurrentUser(GetUserCallback);
}
void GetUserCallback(ResultAnd<UserProfile> response)
{
    if (response.result.Succeeded())
    {
        Debug.Log("Got user: " + response.value.username);
    }
    else
    {
        Debug.Log("failed to get user");
    }
}
```

8.18.2.28 GetCurrentUserRatingFor()

```
static void ModIO.ModIOUnity.GetCurrentUserRatingFor (
    ModId modId,
    Action< ResultAnd< ModRating > > callback ) [static]
```

Gets the rating that the current user has given for a specified mod. You must have an authenticated session for this to be successful.

Note that the rating can be 'None'

Parameters

<i>modId</i>	the id of the mod to check for a rating
<i>callback</i>	callback with the result and rating of the specified mod

See also

[ModRating](#), [ModId](#), [ResultAnd](#)

```
void Example()
{
```

```

    ModId modId = new ModId(1234);
    ModIOUnity.GetCurrentUserRatingFor(modId, GetRatingCallback);
}
void GetRatingCallback(ResultAnd<ModRating> response)
{
    if (response.result.Succeeded())
    {
        Debug.Log($"retrieved rating: {response.value}");
    }
    else
    {
        Debug.Log("failed to get rating");
    }
}
}

```

8.18.2.29 GetCurrentUserRatings()

```

static void ModIO.ModIOUnity.GetCurrentUserRatings (
    Action< ResultAnd< Rating[]> > callback ) [static]

```

Get all mod rating's submitted by the authenticated user. Successful request will return an array of Rating Objects.

Parameters

<i>callback</i>	callback with the Result and an array of RatingObject
-----------------	---

See also

[ModId](#), [RatingObject](#), [ResultAnd](#)

```

void Example()
{
    ModIOUnity.GetCurrentUserRatings(GetCurrentUserRatingsCallback);
}
void GetCurrentUserRatingsCallback(ResultAnd<Rating[]> response)
{
    if (response.result.Succeeded())
    {
        foreach(var ratingObject in response.value)
        {
            Debug.Log($"retrieved rating '{ratingObject.rating}' for {ratingObject.modId}");
        }
    }
    else
    {
        Debug.Log("failed to get ratings");
    }
}
}

```

8.18.2.30 GetInstalledModsForUser()

```

static UserInstalledMod[] ModIO.ModIOUnity.GetInstalledModsForUser (
    out Result result,
    bool includeDisabledMods = false ) [static]

```

Gets an array of mods that are installed for the current user.

Parameters

<i>result</i>	an out Result to inform whether or not it was able to get installed mods
<i>includeDisabledMods</i>	optional parameter. When true it will include mods that have been marked as disabled via the DisableMod method

See also

[UserInstalledMod](#), [GetSubscribedMods](#), [ModIOUnity.DisableMod](#), [ModIOUnity.EnableMod](#)

Returns

an array of [InstalledModUser](#) for each existing mod installed for the user

```
void Example()
{
    InstalledModUser[] mods = ModIOUnity.GetSystemInstalledModsUser(out Result result);
    if (result.Succeeded())
    {
        Debug.Log("found " + mods.Length.ToString() + " mods installed");
    }
    else
    {
        Debug.Log("failed to get installed mods");
    }
}
```

8.18.2.31 GetMod()

```
static void ModIO.ModIOUnity.GetMod (
    ModId modId,
    Action< ResultAnd< ModProfile > > callback ) [static]
```

Requests a single [ModProfile](#) from the mod.io server by its [ModId](#).

If there is a specific mod that you want to retrieve from the mod.io database you can use this method to get it.

Parameters

<i>modId</i>	the ModId of the ModProfile to get
<i>callback</i>	callback with the Result and ModProfile

See also

[ModId](#), [ModProfile](#), [Result](#), [ModIOUnityAsync.GetMod](#)

```
void Example()
{
    ModId modId = new ModId(1234);
    ModIOUnity.GetMod(modId, GetModCallback);
}
void GetModCallback(ResultAnd<ModProfile> response)
{
    if (response.result.Succeeded())
    {
        Debug.Log("retrieved mod " + response.value.name);
    }
    else
    {
        Debug.Log("failed to get mod");
    }
}
```

8.18.2.32 GetModComments()

```
static void ModIO.ModIOUnity.GetModComments (
    ModId modId,
```

```

        SearchFilter filter,
        Action< ResultAnd< CommentPage > > callback ) [static]

```

Get all comments posted in the mods profile. Successful request will return an array of Comment Objects. We recommended reading the filtering documentation to return only the records you want.

Parameters

<i>filter</i>	The filter to apply when searching through comments (can only apply pagination parameters, Eg. page size and page index)
<i>callback</i>	callback invoked with the Result and CommentPage

See also

[CommentPage](#), [ModComment](#), [SearchFilter](#), [ModId](#), [Result](#), [ResultAnd](#), [ModIOUnityAsync.GetModComments](#)

8.18.2.33 GetModDependencies()

```

static void ModIO.ModIOUnity.GetModDependencies (
    ModId modId,
    Action< ResultAnd< ModDependencies[] > > callback ) [static]

```

Retrieves a list of ModDependenciesObjects that represent mods that depend on a mod.

This function returns only immediate mod dependencies, meaning that if you need the dependencies for the dependent mods, you will have to make multiple calls and watch for circular dependencies.

Parameters

<i>modId</i>	the ModId of the mod to get dependencies
<i>callback</i>	callback with the Result and an array of ModDependenciesObjects

See also

[ModId](#), [Result](#), [ResultAnd](#), [ModDependenciesObject](#), [ModIOUnityAsync.GetModDependencies](#)

```

void Example()
{
    ModId modId = new ModId(1234);
    ModIOUnity.GetModDependencies(modId, GetModCallback);
}
void GetModCallback(ResultAnd<ModDependenciesObject[]> response)
{
    if (response.result.Succeeded())
    {
        ModDependenciesObject[] modDependenciesObjects = response.value;
        Debug.Log("retrieved mods dependencies");
    }
    else
    {
        Debug.Log("failed to get mod dependencies");
    }
}

```

8.18.2.34 GetMods()

```
static void ModIO.ModIOUnity.GetMods (
    SearchFilter filter,
    Action< ResultAnd< ModPage > > callback ) [static]
```

Uses a [SearchFilter](#) to retrieve a specific Mod Page and returns the ModProfiles and total number of mods based on the Search Filter.

A [ModPage](#) contains a group of mods based on the pagination filters in [SearchFilter](#). eg, if you use `SearchFilter.SetPageIndex(0)` and `SearchFilter.SetPageSize(100)` then `ModPage.mods` will contain mods from 1 to 100. But if you set `SearchFilter.SetPageIndex(1)` then it will have mods from 101 to 200, if that many exist. (note that 100 is the maximum page size).

Parameters

<i>filter</i>	The filter to apply when searching through mods (also contains pagination parameters)
<i>callback</i>	callback invoked with the Result and ModPage

See also

[SearchFilter](#), [ModPage](#), [Result](#), [ModIOUnityAsync.GetMods](#)

```
void Example()
{
    SearchFilter filter = new SearchFilter();
    filter.SetPageIndex(0);
    filter.SetPageSize(10);
    ModIOUnity.GetMods(filter, GetModsCallback);
}
void GetModsCallback(Result result, ModPage modPage)
{
    if (result.Succeeded())
    {
        Debug.Log("ModPage has " + modPage.modProfiles.Length + " mods");
    }
    else
    {
        Debug.Log("failed to get mods");
    }
}
```

8.18.2.35 GetSubscribedMods()

```
static SubscribedMod[] ModIO.ModIOUnity.GetSubscribedMods (
    out Result result ) [static]
```

Retrieves all of the subscribed mods for the current user.

Note that these are not installed mods only mods the user has opted as 'subscribed'. Also, ensure you have called [ModIOUnity.FetchUpdates\(\)](#) at least once during this session in order to have an accurate collection of the user's subscriptions.

Parameters

<i>result</i>	an out parameter for whether or not the method succeeded
---------------	--

See also

[Result](#), [SubscribedMod](#), [FetchUpdates](#)

Returns

an array of the user's subscribed mods

```
void Example()
{
    SubscribedMod[] mods = ModIOUnity.GetSubscribedMods(out Result result);
    if (result.Succeeded())
    {
        Debug.Log("use has " + mods.Length + " subscribed mods");
    }
    else
    {
        Debug.Log("failed to get user mods");
    }
}
```

8.18.2.36 GetSystemInstalledMods()

```
static InstalledMod[] ModIO.ModIOUnity.GetSystemInstalledMods (
    out Result result ) [static]
```

Gets an array of mods that are installed on the current device.

Note that these will not be subscribed by the current user. If you wish to get all of the current user's installed mods use [ModIOUnity.GetSubscribedMods\(\)](#) and check the `SubscribedMod.status` equals `SubscribedModStatus.Installed`.

Parameters

<i>result</i>	an out Result to inform whether or not it was able to get installed mods
---------------	--

See also

[InstalledMod](#), [GetSubscribedMods](#)

Returns

an array of [InstalledMod](#) for each existing mod installed on the current device (and not subscribed by the current user)

```
void Example()
{
    InstalledMod[] mods = ModIOUnity.GetSystemInstalledMods(out Result result);
    if (result.Succeeded())
    {
        Debug.Log("found " + mods.Length.ToString() + " mods installed");
    }
    else
    {
        Debug.Log("failed to get installed mods");
    }
}
```

8.18.2.37 GetTagCategories()

```
static void ModIO.ModIOUnity.GetTagCategories (
    Action< ResultAnd< TagCategory[] > > callback ) [static]
```

Gets the existing tags for the current game Id that can be used when searching/filtering mods.

Tags come in category groups, eg "Color" could be the name of the category and the tags themselves could be { "Red", "Blue", "Green" }

Parameters

<i>callback</i>	the callback with the result and tags retrieved
-----------------	---

See also

[SearchFilter](#), [TagCategory](#), [Result](#), [ModIOUnityAsync.GetTagCategories](#)

```
void Example()
{
    ModIOUnity.GetTagCategories(GetTagsCallback);
}
void GetTagsCallback(ResultAnd<TagCategory[]> response)
{
    if (response.result.Succeeded())
    {
        foreach(TagCategory category in response.value)
        {
            foreach(Tag tag in category.tags)
            {
                Debug.Log(tag.name + " tag is in the " + category.name + "category");
            }
        }
    }
    else
    {
        Debug.Log("failed to get game tags");
    }
}
```

8.18.2.38 GetTermsOfUse()

```
static void ModIO.ModIOUnity.GetTermsOfUse (
    Action< ResultAnd< TermsOfUse > > callback ) [static]
```

This retrieves the terms of use text to be shown to the user to accept/deny before authenticating their account via a third party provider, eg steam or google.

If the callback succeeds it will also provide a [TermsOfUse](#) struct that contains a [TermsHash](#) struct which you will need to provide when calling a third party authentication method such as [ModIOUnity.AuthenticateUserViaSteam\(\)](#)

Parameters

<i>callback</i>	Callback to invoke once the operation is complete containing a result and a hash code to use for authentication via third party providers.
-----------------	--

See also

[TermsOfUse](#), [AuthenticateUserViaDiscord](#), [AuthenticateUserViaGoogle](#), [AuthenticateUserViaGOG](#), [AuthenticateUserViaItch](#), [AuthenticateUserViaOculus](#), [AuthenticateUserViaSteam](#), [AuthenticateUserViaSwitch](#), [AuthenticateUserViaXbox](#), [AuthenticateUserViaPlayStation](#), [ModIOUnityAsync.GetTermsOfUse](#)

```
void Example()
{
    ModIOUnity.GetTermsOfUse(GetTermsOfUseCallback);
}
void GetTermsOfUseCallback(Result&#60;TermsOfUse&#62; response)
{
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}
```

8.18.2.39 InitializeForUser() [1/2]

```
static Result ModIO.ModIOUnity.InitializeForUser (
    string userProfileIdentifier ) [static]
```

Initializes the Plugin using the provided settings for a specified user. Loads the local state of mods installed on the system as well as relevant mods to the user. Loads the state of mods installed on the system as well as the set of mods the specified user has installed on this device.

Parameters

<i>userProfileIdentifier</i>	Name of the directory to store the user's data in.
------------------------------	--

See also

[Result](#), [Shutdown](#)

```
void Example()
{
    ModIOUnity.InitializeForUser("ExampleUser", InitializationCallback);
}
void InitializationCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Initialized plugin");
    }
    else
    {
        Debug.Log("Failed to initialize plugin");
    }
}
```

8.18.2.40 InitializeForUser() [2/2]

```
static Result ModIO.ModIOUnity.InitializeForUser (
    string userProfileIdentifier,
    ServerSettings serverSettings,
    BuildSettings buildSettings ) [static]
```

Initializes the Plugin using the provided settings for a specified user. Loads the local state of mods installed on the system as well as relevant mods to the user. Loads the state of mods installed on the system as well as the set of mods the specified user has installed on this device.

Parameters

<i>userProfileIdentifier</i>	Name of the directory to store the local profile's data (unrelated to the authenticated user)
<i>serverSettings</i>	Data used by the plugin to connect with the mod.io service.
<i>buildSettings</i>	Data used by the plugin to interact with the platform.

See also

[FetchUpdates](#), [ServerSettings](#), [BuildSettings](#), [Result](#), [Shutdown](#)

```
void Example()
{
    // Setup a ServerSettings struct
    ServerSettings serverSettings = new ServerSettings();
    serverSettings.serverURL = "https://api.test.mod.io/v1";
    serverSettings.gameId = 1234;
    serverSettings.gameKey = "1234567890abcdefghijklmnopqrstuvwxyz";
    // Setup a BuildSettings struct
    BuildSettings buildSettings = new BuildSettings();
    buildSettings.logLevel = LogLevel.Verbose;
    buildSettings.userPortal = UserPortal.None;
    buildSettings.requestCacheLimitKB = 0; // No limit
    ModIOUnity.InitializeForUser("ExampleUser", serverSettings, buildSettings, InitializationCallback);
}

void InitializationCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Initialized plugin");
    }
    else
    {
        Debug.Log("Failed to initialize plugin");
    }
}
```

8.18.2.41 IsAuthenticated()

```
static void ModIO.ModIOUnity.IsAuthenticated (
    Action< Result > callback ) [static]
```

Informs you if the current user session is authenticated or not.

Parameters

<i>callback</i>	
-----------------	--

See also

[Result](#), [ModIOUnityAsync.IsAuthenticated](#)

```
void Example()
{
    ModIOUnity.IsAuthenticated(IsAuthenticatedCallback);
}

void IsAuthenticatedCallback(Result result)
{
    if (result.Succeeded())
    {
        // ...
    }
}
```

```
        Debug.Log("current session is authenticated");
    }
    else
    {
        Debug.Log("current session is not authenticated");
    }
}
```

8.18.2.42 IsInitialized()

```
static bool ModIO.ModIOUnity.IsInitialized ( ) [static]
```

You can use this to quickly identify whether or not the plugin has been initialized.

Returns

true if the plugin is initialized

```
void Example()
{
    if (ModIOUnity.IsInitialized())
    {
        Debug.Log("The plugin is initialized");
    }
    else
    {
        Debug.Log("The plugin is not initialized");
    }
}
```

8.18.2.43 IsModManagementBusy()

```
static bool ModIO.ModIOUnity.IsModManagementBusy ( ) [static]
```

Checks if the automatic management process is currently awake and performing a mod management operation, such as installing, downloading, uninstalling, updating.

Returns

True if automatic mod management is currently performing an operation.

See also

[EnableModManagement](#), [DisableModManagement](#), [GetCurrentModManagementOperation](#)

```
void Example()
{
    if (ModIOUnity.IsModManagementBusy())
    {
        Debug.Log("mod management is busy");
    }
    else
    {
        Debug.Log("mod management is not busy");
    }
}
```


8.18.2.44 LogoutCurrentUser()

```
static Result ModIO.ModIOUnity.LogoutCurrentUser ( ) [static]
```

De-authenticates the current Mod.io user for the current session and clears all user-specific data stored on the current device. Installed mods that do not have other local users subscribed will be uninstalled if [ModIOUnity.EnableModManagement\(\)](#) has been used to enable the mod management system. (If ModManagement is enabled).

If you dont want to erase a user be sure to use [ModIOUnity.Shutdown\(\)](#) instead. If you re-initialize the plugin after a shutdown the user will still be authenticated.

See also

[EnableModManagement\(ModIO.ModManagementEventDelegate\)](#), [Result](#)

```
//static async void Example()
{
    Result result = await ModIOUnity.LogoutCurrentUser();
    if(result.Succeeded())
    {
        Debug.Log("The current user has been logged and their local data removed");
    }
    else
    {
        Debug.Log("Failed to log out the current user");
    }
}
```

8.18.2.45 MuteUser()

```
static void ModIO.ModIOUnity.MuteUser (
    long userId,
    Action< Result > callback ) [static]
```

Mutes a user which effectively hides any content from that specified user

The userId can be found from the [UserProfile](#). Such as ModProfile.creator.userId

Parameters

<i>userId</i>	The id of the user to be muted
<i>callback</i>	callback with the Result of the request

See also

[UserProfile](#)

8.18.2.46 RateMod()

```
static void ModIO.ModIOUnity.RateMod (
    ModId modId,
```

```
ModRating rating,
Action< Result > callback ) [static]
```

Used to submit a rating for a specified mod.

This can be used to change/overwrite previous ratings of the current user.

Parameters

<i>modId</i>	the m= ModId of the mod being rated
<i>rating</i>	the rating to give the mod. Allowed values include ModRating.Positive , ModRating.Negative , ModRating.None
<i>callback</i>	callback with the result of the request

See also

[ModRating](#), [Result](#), [ModId](#), [ModIOUnityAsync.RateMod](#)

```
ModProfile mod;
void Example()
{
    ModIOUnity.RateMod(mod.id, ModRating.Positive, RateModCallback);
}
void RateModCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Successfully rated mod");
    }
    else
    {
        Debug.Log("Failed to rate mod");
    }
}
```

8.18.2.47 RemoveDependenciesFromMod()

```
static void ModIO.ModIOUnity.RemoveDependenciesFromMod (
    ModId modId,
    ICollection< ModId > dependencies,
    Action< Result > callback ) [static]
```

Removes the specified mods as dependencies for another existing mod.

If the dependencies dont exist they will be ignored and the result will return success

Parameters

<i>modId</i>	ModId of the mod you want to remove dependencies from
<i>dependencies</i>	The ModIds that you want to remove (max 5 at a time)
<i>callback</i>	callback with the result of the request

See also

[Result](#), [ModId](#), [dependencies](#), [ModIOUnity.AddDependenciesToMod](#), [ModIOUnityAsync.RemoveDependenciesFromMod](#), [ModIOUnityAsync.AddDependenciesToMod](#)

```

void Example()
{
    var dependencies = new List<ModId>()
    {
        (ModId)1001,
        (ModId)1002,
        (ModId)1003
    };
    ModIOUnity.RemoveDependenciesFromMod(mod.id, dependencies, RemoveDependenciesCallback);
}
void RemoveDependenciesCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Successfully removed dependencies from mod");
    }
    else
    {
        Debug.Log("Failed to removed dependencies from mod");
    }
}

```

8.18.2.48 Report()

```

static void ModIO.ModIOUnity.Report (
    Report report,
    Action< Result > callback ) [static]

```

Reports a specified mod to mod.io.

Parameters

<i>report</i>	the object containing all of the details of the report you are sending
<i>callback</i>	callback with the Result of the report

See also

[Report](#), [Result](#), [ModIOUnityAsync.Report](#)

```

void Example()
{
    Report report = new Report(new ModId(123),
                               ReportType.Generic,
                               "reporting this mod for a generic reason",
                               "JohnDoe",
                               "johndoe@mod.io");
    ModIOUnity.Report(report, ReportCallback);
}
void ReportCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("successfully sent a report");
    }
    else
    {
        Debug.Log("failed to send a report");
    }
}

```

8.18.2.49 RequestAuthenticationEmail()

```

static void ModIO.ModIOUnity.RequestAuthenticationEmail (
    string emailaddress,
    Action< Result > callback ) [static]

```

Sends an email with a security code to the specified Email Address. The security code is then used to Authenticate the user session using [ModIOUnity.SubmitEmailSecurityCode\(\)](#)

The callback will return a [Result](#) object. If the email is successfully sent `Result.Succeeded()` will equal true. If you haven't Initialized the plugin then `Result.IsInitializationError()` will equal true. If the string provided for the email address is not .NET compliant `Result.IsAuthenticationError()` will equal true.

Parameters

<i>emailaddress</i>	the Email Address to send the security code to, eg "JohnDoe@gmail.com"
<i>callback</i>	Callback to invoke once the operation is complete

See also

[SubmitEmailSecurityCode](#), [Result](#), [ModIOUnityAsync.RequestAuthenticationEmail](#)

```
void Example()
{
    ModIOUnity.RequestAuthenticationEmail("johndoe@gmail.com", RequestAuthenticationCallback);
}
void RequestAuthenticationCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Succeeded to send security code");
    }
    else
    {
        Debug.Log("Failed to send security code to that email address");
    }
}
```

8.18.2.50 SetLoggingDelegate()

```
static void ModIO.ModIOUnity.SetLoggingDelegate (
    LogMessageDelegate loggingDelegate ) [static]
```

Assigns the logging delegate the plugin uses to output log messages that otherwise go to `UnityEngine.Debug.Log(string)`

If you don't wish to see [mod.io] logs appearing in the Unity console you can set your own delegate for handling logs and ignore them or display them elsewhere.

Parameters

<i>loggingDelegate</i>	The delegate for receiving log messages
------------------------	---

See also

[LogMessageDelegate](#), [LogLevel](#)

```
void Example()
{
    // Send logs to MyLoggingDelegate instead of Debug.Log
    ModIOUnity.SetLoggingDelegate(MyLoggingDelegate);
}
public void MyLoggingDelegate(LogLevel logLevel, string logMessage)
{
    // Handle the log entry
    if (logLevel == LogLevel.Error)
```

```
{  
    Debug.Log("We received an error with message: " + logMessage);  
}  
}
```

8.18.2.51 Shutdown()

```
static void ModIO.ModIOUnity.Shutdown (  
    Action shutdownComplete ) [static]
```

Cancels any running public operations, frees plugin resources, and invokes any pending callbacks with a cancelled result code.

Callback results invoked during a shutdown operation can be checked with `Result.IsCancelled()`

See also

[Result](#)

```
void Example()  
{  
    ModIOUnity.Shutdown(ShutdownCallback);  
}  
void ShutdownCallback()  
{  
    Debug.Log("Finished shutting down the ModIO Plugin");  
}
```

8.18.2.52 SubmitEmailSecurityCode()

```
static void ModIO.ModIOUnity.SubmitEmailSecurityCode (  
    string securityCode,  
    Action< Result > callback ) [static]
```

Attempts to Authenticate the current session by submitting a security code received by email from [ModIOUnity.RequestAuthenticationEmail\(\)](#)

It is intended that this function is used after [ModIOUnity.RequestAuthenticationEmail\(\)](#) is performed successfully.

Parameters

<i>securityCode</i>	The security code received from an authentication email
<i>callback</i>	Callback to invoke once the operation is complete

See also

[RequestAuthenticationEmail](#), [Result](#), [ModIOUnityAsync.SubmitEmailSecurityCode](#)

```
void Example(string userSecurityCode)  
{  
    ModIOUnity.SubmitEmailSecurityCode(userSecurityCode, SubmitCodeCallback);  
}  
void SubmitCodeCallback(Result result)  
{  
    if (result.Succeeded())
```

```

    {
        Debug.Log("You have successfully authenticated the user");
    }
    else
    {
        Debug.Log("Failed to authenticate the user");
    }
}

```

8.18.2.53 SubscribeToMod()

```

static void ModIO.ModIOUnity.SubscribeToMod (
    ModId modId,
    Action< Result > callback ) [static]

```

Adds the specified mod to the current user's subscriptions.

If mod management has been enabled via [ModIOUnity.EnableModManagement\(\)](#) then the mod will be downloaded and installed.

Parameters

<i>modId</i>	ModId of the mod you want to subscribe to
<i>callback</i>	callback with the result of the request

See also

[Result](#), [ModId](#), [EnableModManagement\(ModIO.ModManagementEventDelegate\)](#), [GetCurrentModManagementOperation](#), [ModIOUnityAsync.SubscribeToMod](#)

```

ModProfile mod;
void Example()
{
    ModIOUnity.SubscribeToMod(mod.id, SubscribeCallback);
}
void SubscribeCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Successfully subscribed to mod");
    }
    else
    {
        Debug.Log("Failed to subscribe to mod");
    }
}

```

8.18.2.54 UnmuteUser()

```

static void ModIO.ModIOUnity.UnmuteUser (
    long userId,
    Action< Result > callback ) [static]

```

Un-mutes a user which effectively reveals previously hidden content from that user

The `userId` can be found from the [UserProfile](#). Such as `ModProfile.creator.userId`

Parameters

<i>userId</i>	The id of the user to be muted
<i>callback</i>	callback with the Result of the request

See also

[UserProfile](#)**8.18.2.55 UnsubscribeFromMod()**

```
static void ModIO.ModIOUnity.UnsubscribeFromMod (
    ModId modId,
    Action< Result > callback ) [static]
```

Removes the specified mod from the current user's subscriptions.

If mod management has been enabled via [ModIOUnity.EnableModManagement\(\)](#) then the mod will be uninstalled at the next opportunity.

Parameters

<i>modId</i>	ModId of the mod you want to unsubscribe from
<i>callback</i>	callback with the result of the request

See also

[Result](#), [ModId](#), [EnableModManagement\(ModIO.ModManagementEventDelegate\)](#), [GetCurrentModManagementOperation](#), [ModIOUnityAsync.UnsubscribeFromMod](#)

```
ModProfile mod;
void Example()
{
    ModIOUnity.UnsubscribeFromMod(mod.id, UnsubscribeCallback);
}
void UnsubscribeCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("Successfully unsubscribed from mod");
    }
    else
    {
        Debug.Log("Failed to unsubscribe from mod");
    }
}
```

8.18.2.56 UpdateModComment()

```
static void ModIO.ModIOUnity.UpdateModComment (
    ModId modId,
    string content,
```

```
long commentId,  
Action< ResultAnd< ModComment > > callback ) [static]
```

Update a comment for the corresponding mod. Successful request will return the updated Comment Object.

Parameters

<i>modId</i>	Id of the mod the comment is on
<i>content</i>	Updated contents of the comment.
<i>commentId</i>	The id for the comment you wish to edit
<i>callback</i>	callback with the result of the operation

See also

[ResultAnd](#), [ModComment](#), [ModIOUnityAsync.UpdateModComment](#)

```
private string content = "This is a Comment";
long commentId = 12345;
ModId modId = (ModId)1234;
void UpdateMod()
{
    ModIOUnity.UpdateModComment(modId, content, commentId, UpdateCallback);
}
void UpdateCallback(ResultAnd<ModComment> resultAnd)
{
    if (resultAnd.result.Succeeded())
    {
        Debug.Log("Successfully Updated Comment!");
    }
    else
    {
        Debug.Log("Failed to Update Comment!");
    }
}
```

8.18.2.57 UploadModfile()

```
static void ModIO.ModIOUnity.UploadModfile (
    ModfileDetails modfile,
    Action< Result > callback ) [static]
```

Used to upload a mod file to a mod profile on the mod.io server. A mod file is the actual archive of a mod. This method can be used to update a mod to a newer version (you can include changelog information in [ModfileDetails](#)).

If you want to upload images such as a new logo or gallery images, you can use [UploadModMedia](#) instead.

Parameters

<i>modfile</i>	the mod file and details to upload
<i>callback</i>	callback with the Result of the upload when the operation finishes

See also

[Result](#), [ModfileDetails](#), [ArchiveModProfile](#), [GetCurrentUploadHandle](#), [ModIOUnityAsync.UploadModfile](#), [UploadModMedia](#)

```
ModId modId;
void Example()
{
    ModfileDetails modfile = new ModfileDetails();
    modfile.modId = modId;
    modfile.directory = "files/mods/mod_123";
    ModIOUnity.UploadModfile(modfile, UploadModCallback);
}
```

```
void UploadModCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("uploaded mod file");
    }
    else
    {
        Debug.Log("failed to upload mod file");
    }
}
```

8.18.2.58 UploadModMedia()

```
static void ModIO.ModIOUnity.UploadModMedia (
    ModProfileDetails modProfileDetails,
    Action< Result > callback ) [static]
```

This is used to update the logo of a mod or the gallery images. This works very similar to EditModProfile except it only affects the images.

Parameters

<i>modProfileDetails</i>	this holds the reference to the images you wish to upload
<i>callback</i>	a callback with the Result of the operation

See also

[ModProfileDetails](#), [Result](#), [EditModProfile](#), [ModIOUnityAsync.UploadModMedia](#)

```
ModId modId;
Texture2D newTexture;
void Example()
{
    ModProfileDetails profile = new ModProfileDetails();
    profile.modId = modId;
    profile.logo = newTexture;
    ModIOUnity.UploadModMedia(profile, UploadProfileCallback);
}
void UploadProfileCallback(Result result)
{
    if (result.Succeeded())
    {
        Debug.Log("uploaded new mod logo");
    }
    else
    {
        Debug.Log("failed to uploaded mod logo");
    }
}
```

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/ModIOUnity.cs

8.19 ModIO.ModIOUnityAsync Class Reference

Main async interface for the mod.io Unity plugin. Every method within ModIOUnity.cs that has a callback can also be found in [ModIOUnityAsync](#) with an asynchronous alternative method (if you'd rather not use callbacks).

Static Public Member Functions

- static async Task [Shutdown](#) ()

Cancels any running public operations, frees plugin resources, and invokes any pending callbacks with a cancelled result code.

- static async Task< [Result](#) > [RequestAuthenticationEmail](#) (string emailAddress)

Sends an email with a security code to the specified Email Address. The security code is then used to Authenticate the user session using [ModIOUnity.SubmitEmailSecurityCode\(\)](#)

- static async Task< [Result](#) > [SubmitEmailSecurityCode](#) (string securityCode)

Attempts to Authenticate the current session by submitting a security code received by email from [ModIOUnity.RequestAuthenticationEmail](#)

- static async Task< [ResultAnd](#)< [TermsOfUse](#) > > [GetTermsOfUse](#) ()

- static async Task< [Result](#) > [AuthenticateUserViaSteam](#) (string steamToken, string emailAddress, [TermsHash?](#) hash)

Attempts to authenticate a user via the steam API.

- static async Task< [Result](#) > [AuthenticateUserViaEpic](#) (string epicToken, string emailAddress, [TermsHash?](#) hash)

Attempts to authenticate a user via the epic API.

- static async Task< [Result](#) > [AuthenticateUserViaPlayStation](#) (string authCode, string emailAddress, [TermsHash?](#) hash, PlayStationEnvironment environment)

Attempts to authenticate a user via the steam API.

- static async Task< [Result](#) > [AuthenticateUserViaGOG](#) (string gogToken, string emailAddress, [TermsHash?](#) hash)

Attempts to authenticate a user via the GOG API.

- static async Task< [Result](#) > [AuthenticateUserViaItch](#) (string itchioToken, string emailAddress, [TermsHash?](#) hash)

Attempts to authenticate a user via the Itch.io API.

- static async Task< [Result](#) > [AuthenticateUserViaXbox](#) (string xboxToken, string emailAddress, [TermsHash?](#) hash)

Attempts to authenticate a user via the Xbox API.

- static async Task< [Result](#) > [AuthenticateUserViaSwitch](#) (string switchToken, string emailAddress, [TermsHash?](#) hash)

Attempts to authenticate a user via the switch API.

- static async Task< [Result](#) > [AuthenticateUserViaDiscord](#) (string discordToken, string emailAddress, [TermsHash?](#) hash)

Attempts to authenticate a user via the discord API.

- static async Task< [Result](#) > [AuthenticateUserViaGoogle](#) (string googleToken, string emailAddress, [TermsHash?](#) hash)

Attempts to authenticate a user via the google API.

- static async Task< [Result](#) > [AuthenticateUserViaOculus](#) (OculusDevice oculusDevice, string nonce, long userId, string oculusToken, string emailAddress, [TermsHash?](#) hash)

Attempts to authenticate a user via the oculus API.

- static async Task< [Result](#) > [IsAuthenticated](#) ()

Informs you if the current user session is authenticated or not.

- static async Task< [ResultAnd](#)< [TagCategory](#)[] > > [GetTagCategories](#) ()

Gets the existing tags for the current game Id that can be used when searching/filtering mods.

- static async Task< [ResultAnd](#)< [ModPage](#) > > [GetMods](#) ([SearchFilter](#) filter)

Uses a [SearchFilter](#) to retrieve a specific Mod Page and returns the ModProfiles and total number of mods based on the Search Filter.

- static async Task< [ResultAnd](#)< [ModProfile](#) > > [GetMod](#) (ModId modId)

Requests a single [ModProfile](#) from the mod.io server by its [ModId](#).

- static async Task< [ResultAnd](#)< [CommentPage](#) > > [GetModComments](#) (ModId modId, [SearchFilter](#) filter)

Get all comments posted in the mods profile. Successful request will return an array of Comment Objects. We recommended reading the filtering documentation to return only the records you want.

- static async Task< [ResultAnd< ModComment > >](#) [AddModComment](#) (ModId modId, [CommentDetails](#) commentDetails)
Retrieves a list of ModDependenciesObjects that represent mods that depend on a mod.
- static async Task< [Result >](#) [DeleteModComment](#) (ModId modId, long commentId)
Delete a comment from a mod profile. Successful request will return 204 No Content and fire a MOD_COMMENT↔_DELETED event.
- static async Task< [ResultAnd< ModComment > >](#) [UpdateModComment](#) (ModId modId, string content, long commentId)
Update a comment for the corresponding mod. Successful request will return the updated Comment Object.
- static async Task< [ResultAnd< ModDependencies\[\] > >](#) [GetModDependencies](#) (ModId modId)
- static async Task< [ResultAnd< Rating\[\] > >](#) [GetCurrentUserRatings](#) ()
Get all mod rating's submitted by the authenticated user. Successful request will return an array of Rating Objects.
- static async Task< [ResultAnd< ModRating > >](#) [GetCurrentUserRatingFor](#) (ModId modId)
Gets the rating that the current user has given for a specified mod. You must have an authenticated session for this to be successful.
- static async Task< [Result >](#) [RateMod](#) (ModId modId, ModRating rating)
Used to submit a rating for a specified mod.
- static async Task< [Result >](#) [SubscribeToMod](#) (ModId modId)
Adds the specified mod to the current user's subscriptions.
- static async Task< [Result >](#) [UnsubscribeFromMod](#) (ModId modId)
Removes the specified mod from the current user's subscriptions.
- static async Task< [ResultAnd< UserProfile > >](#) [GetCurrentUser](#) ()
Gets the current user's [UserProfile](#) struct. Containing their mod.io username, user id, language, timezone and download references for their avatar.
- static void [MuteUser](#) (long userId)
Mutes a user which effectively hides any content from that specified user
- static void [UnmuteUser](#) (long userId)
Un-mutes a user which effectively reveals previously hidden content from that user
- static async Task< [Result >](#) [FetchUpdates](#) ()
This retrieves the user's subscriptions from the mod.io server and synchronises it with our local instance of the user's subscription data. If mod management has been enabled via [ModIOUnity.EnableModManagement\(\)](#) then it may begin to install/uninstall mods.
- static async Task< [Result >](#) [AddDependenciesToMod](#) (ModId modId, ICollection< [ModId](#) > dependencies)
Adds the specified mods as dependencies to an existing mod.
- static async Task< [Result >](#) [RemoveDependenciesFromMod](#) (ModId modId, ICollection< [ModId](#) > dependencies)
Removes the specified mods as dependencies for another existing mod.
- static async Task< [ResultAnd< ModId > >](#) [CreateModProfile](#) ([CreationToken](#) token, [ModProfileDetails](#) modProfileDetails)
Creates a new mod profile on the mod.io server based on the details provided from the [ModProfileDetails](#) object provided. Note that you must have a logo, name and summary assigned in [ModProfileDetails](#) in order for this to work.
- static async Task< [Result >](#) [EditModProfile](#) ([ModProfileDetails](#) modprofile)
This is used to edit or change data in an existing mod profile on the mod.io server.
- static async Task< [Result >](#) [UploadModfile](#) ([ModfileDetails](#) modfile)
Used to upload a mod file to a mod profile on the mod.io server. A mod file is the actual archive of a mod. This method can be used to update a mod to a newer version (you can include changelog information in [ModfileDetails](#)).
- static async Task< [Result >](#) [UploadModMedia](#) ([ModProfileDetails](#) modProfileDetails)
This is used to update the logo of a mod or the gallery images. This works very similar to [EditModProfile](#) except it only affects the images.
- static async Task< [Result >](#) [ArchiveModProfile](#) (ModId modId)
Removes a mod from being visible on the mod.io server.
- static async Task< [ResultAnd< ModPage > >](#) [GetCurrentUserCreations](#) ([SearchFilter](#) filter)

Get all mods the authenticated user added or is a team member of. Successful request will return an array of Mod Objects. We recommended reading the filtering documentation to return only the records you want.

- static async Task< [Result](#) > [AddTags](#) ([ModId](#) modId, string[] tags)
Adds the provided tags to the specified mod id. In order for this to work the authenticated user must have permission to edit the specified mod. Only existing tags as part of the game Id will be added.
- static async Task< [Result](#) > [DeleteTags](#) ([ModId](#) modId, string[] tags)
Deletes the specified tags from the mod. In order for this to work the authenticated user must have permission to edit the specified mod.
- static async Task< [ResultAnd](#)< byte[] > > [DownloadImage](#) ([DownloadReference](#) downloadReference)
Downloads a texture based on the specified download reference.
- static async Task< [Result](#) > [Report](#) ([Report](#) report)
Reports a specified mod to mod.io.

8.19.1 Detailed Description

Main async interface for the mod.io Unity plugin. Every method within ModIOUnity.cs that has a callback can also be found in [ModIOUnityAsync](#) with an asynchronous alternative method (if you'd rather not use callbacks).

See also

[ModIOUnity](#)

8.19.2 Member Function Documentation

8.19.2.1 AddDependenciesToMod()

```
static async Task< Result > ModIO.ModIOUnityAsync.AddDependenciesToMod (
    ModId modId,
    ICollection< ModId > dependencies ) [static]
```

Adds the specified mods as dependencies to an existing mod.

If the dependencies already exist they will be ignored and the result will return success

Parameters

<i>modId</i>	ModId of the mod you want to add dependencies to
<i>dependencies</i>	The ModIds that you want to add (max 5 at a time)

See also

[Result](#), [ModId](#), [ModIOUnity.AddDependenciesToMod](#), [ModIOUnity.RemoveDependenciesFromMod](#),
[ModIOUnityAsync.RemoveDependenciesFromMod](#)

```
async void Example()
{
    var dependencies = new List<#60;ModId#62;
    {
        (ModId) 1001,
```

```

        (ModId)1002,
        (ModId)1003
    };
    Result result = await ModIOUnityAsync.AddDependenciesToMod(mod.id, dependencies);
    if (result.Succeeded())
    {
        Debug.Log("Successfully added dependencies to mod");
    }
    else
    {
        Debug.Log("Failed to add dependencies to mod");
    }
}

```

8.19.2.2 AddModComment()

```

static async Task< ResultAnd< ModComment > > ModIO.ModIOUnityAsync.AddModComment (
    ModId modId,
    CommentDetails commentDetails ) [static]

```

Retrieves a list of ModDependenciesObjects that represent mods that depend on a mod.

This function returns only immediate mod dependencies, meaning that if you need the dependencies for the dependent mods, you will have to make multiple calls and watch for circular dependencies.

See also

[ModId](#), [Result](#), [ResultAnd](#), [ModDependenciesObject](#), [ModIOUnity.GetModDependencies](#)

```

async void Example()
{
    ModId modId = new ModId(1234);
    var resultAnd = await ModIOUnityAsync.GetModDependencies(modId);
    if (resultAnd.result.Succeeded())
    {
        ModDependenciesObject[] modDependenciesObjects = resultAnd.value;
        Debug.Log("retrieved mods dependencies");
    }
    else
    {
        Debug.Log("failed to get mod dependencies");
    }
}

```

Parameters

<i>modId</i>	
<i>commentDetails</i>	

Returns

8.19.2.3 AddTags()

```

static async Task< Result > ModIO.ModIOUnityAsync.AddTags (
    ModId modId,
    string[] tags ) [static]

```

Adds the provided tags to the specified mod id. In order for this to work the authenticated user must have permission to edit the specified mod. Only existing tags as part of the game Id will be added.

Parameters

<i>mod</i> ↔ <i>Id</i>	Id of the mod to add tags to
<i>tags</i>	array of tags to be added

See also

[Result](#), [DeleteTags](#), [ModIOUnityAsync.AddTags](#)

```
ModId modId;
string[] tags;
async void Example()
{
    Result result = await ModIOUnity.AddTags(modId, tags);
    if (result.Succeeded())
    {
        Debug.Log("added tags");
    }
    else
    {
        Debug.Log("failed to add tags");
    }
}
```

8.19.2.4 ArchiveModProfile()

```
static async Task< Result > ModIO.ModIOUnityAsync.ArchiveModProfile (
    ModId modId ) [static]
```

Removes a mod from being visible on the mod.io server.

If you want to delete a mod permanently you can do so from a web browser.

Parameters

<i>mod</i> ↔ <i>Id</i>	the id of the mod to delete
---------------------------	-----------------------------

See also

[Result](#), [CreateModProfile](#), [EditModProfile](#)

```
ModId modId;
async void Example()
{
    Result result = await ModIOUnityAsync.ArchiveModProfile(modId);
    if (result.Succeeded())
    {
        Debug.Log("archived mod profile");
    }
    else
    {
        Debug.Log("failed to archive mod profile");
    }
}
```


8.19.2.5 AuthenticateUserViaDiscord()

```
static async Task< Result > ModIO.ModIOUnityAsync.AuthenticateUserViaDiscord (
    string discordToken,
    string emailAddress,
    TermsHash? hash ) [static]
```

Attempts to authenticate a user via the discord API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>discordToken</i>	the user's steam token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()

See also

[GetTermsOfUse](#)

```
// First we get the Terms of Use to display to the user and cache the hash
async void GetTermsOfUse_Example()
{
    ResultAndTermsOfUse response = await ModIOUnityAsync.GetTermsOfUse();
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}

// Once we have the Terms of Use and hash we can attempt to authenticate
async void Authenticate_Example()
{
    Result result = await ModIOUnityAsync.AuthenticateUserViaDiscord(discordToken, "johndoe@gmail.com",
        modIOTermsOfUse.hash);
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}
```

8.19.2.6 AuthenticateUserViaEpic()

```
static async Task< Result > ModIO.ModIOUnityAsync.AuthenticateUserViaEpic (
    string epicToken,
    string emailAddress,
    TermsHash? hash ) [static]
```

Attempts to authenticate a user via the epic API.

Parameters

<i>epicToken</i>	the user's epic token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()

See also

[GetTermsOfUse](#), [ModIOUnity.AuthenticateUserViaEpic](#)

```
// First we get the Terms of Use to display to the user and cache the hash
async void GetTermsOfUse_Example()
{
    ResultAndTermsOfUser response = await ModIOUnityAsync.GetTermsOfUse();
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}

// Once we have the Terms of Use and hash we can attempt to authenticate
async void Authenticate_Example()
{
    Result result = await ModIOUnityAsync.AuthenticateUserViaEpic(epicToken, "johndoe@gmail.com",
        modIOTermsOfUse.hash);
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}
```

8.19.2.7 AuthenticateUserViaGOG()

```
static async Task< Result > ModIO.ModIOUnityAsync.AuthenticateUserViaGOG (
    string gogToken,
    string emailAddress,
    TermsHash? hash ) [static]
```

Attempts to authenticate a user via the GOG API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>gogToken</i>	the user's steam token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()

See also

[GetTermsOfUse](#)

```
// First we get the Terms of Use to display to the user and cache the hash
async void GetTermsOfUse_Example()
{
    ResultAnd<TermsOfUser> response = await ModIOUnityAsync.GetTermsOfUse();
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}

// Once we have the Terms of Use and hash we can attempt to authenticate
async void Authenticate_Example()
{
    Result result = await ModIOUnityAsync.AuthenticateUserViaGOG(gogToken, "johndoe@gmail.com",
        modIOTermsOfUse.hash);
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}
```

8.19.2.8 AuthenticateUserViaGoogle()

```
static async Task< Result > ModIO.ModIOUnityAsync.AuthenticateUserViaGoogle (
    string googleToken,
    string emailAddress,
    TermsHash? hash ) [static]
```

Attempts to authenticate a user via the google API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>googleToken</i>	the user's steam token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()

See also

[GetTermsOfUse](#)

```
// First we get the Terms of Use to display to the user and cache the hash
async void GetTermsOfUse_Example()
{
    ResultAnd<TermsOfUser> response = await ModIOUnityAsync.GetTermsOfUse();
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}

// Once we have the Terms of Use and hash we can attempt to authenticate
async void Authenticate_Example()
```

```

{
    Result result = await ModIOUnityAsync.AuthenticateUserViaGoogle(googleToken, "johndoe@gmail.com",
        modIOTermsOfUse.hash);
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}

```

8.19.2.9 AuthenticateUserViaItch()

```

static async Task< Result > ModIO.ModIOUnityAsync.AuthenticateUserViaItch (
    string itchioToken,
    string emailAddress,
    TermsHash? hash ) [static]

```

Attempts to authenticate a user via the Itch.io API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>itchioToken</i>	the user's steam token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()

See also

[GetTermsOfUse](#)

```

// First we get the Terms of Use to display to the user and cache the hash
async void GetTermsOfUse_Example()
{
    Result&#60;TermsOfUser&#62; response = await ModIOUnityAsync.GetTermsOfUse();
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}
// Once we have the Terms of Use and hash we can attempt to authenticate
async void Authenticate_Example()
{
    Result result = await ModIOUnityAsync.AuthenticateUserViaItch(itchioToken, "johndoe@gmail.com",
        modIOTermsOfUse.hash);
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}

```

8.19.2.10 AuthenticateUserViaOculus()

```
static async Task< Result > ModIO.ModIOUnityAsync.AuthenticateUserViaOculus (
    OculusDevice oculusDevice,
    string nonce,
    long userId,
    string oculusToken,
    string emailAddress,
    TermsHash? hash ) [static]
```

Attempts to authenticate a user via the oculus API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>oculusToken</i>	the user's oculus token
<i>oculusDevice</i>	the device you're authenticating on
<i>nonce</i>	the nonce
<i>userId</i>	the user id
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()

See also

[GetTermsOfUse](#)

```
// First we get the Terms of Use to display to the user and cache the hash
async void GetTermsOfUse_Example()
{
    ResultAndTermsOfUse response = await ModIOUnityAsync.GetTermsOfUse();
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}

// Once we have the Terms of Use and hash we can attempt to authenticate
async void Authenticate_Example()
{
    Result result = await ModIOUnityAsync.AuthenticateUserViaOculus(OculusDevice.Quest,
                                                                    nonce,
                                                                    userId,
                                                                    oculusToken,
                                                                    "johndoe@gmail.com",
                                                                    modIOTermsOfUse.hash);

    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}
```

8.19.2.11 AuthenticateUserViaPlayStation()

```
static async Task< Result > ModIO.ModIOUnityAsync.AuthenticateUserViaPlayStation (
    string authCode,
```

```

        string emailAddress,
        TermsHash? hash,
        PlayStationEnvironment environment ) [static]

```

Attempts to authenticate a user via the steam API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>authCode</i>	the user's authcode token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()
<i>environment</i>	the PSN account environment

See also

[GetTermsOfUse](#)

```

// First we get the Terms of Use to display to the user and cache the hash
async void GetTermsOfUse_Example()
{
    ResultAnd<TermsOfUser> response = await ModIOUnityAsync.GetTermsOfUse();
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}
// Once we have the Terms of Use and hash we can attempt to authenticate
async void Authenticate_Example()
{
    Result result = await ModIOUnityAsync.AuthenticateUserViaPlayStation(authCode, "johndoe@gmail.com",
        modIOTermsOfUse.hash, PlayStationEnvironment.np);
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}

```

8.19.2.12 AuthenticateUserViaSteam()

```

static async Task< Result > ModIO.ModIOUnityAsync.AuthenticateUserViaSteam (
    string steamToken,
    string emailAddress,
    TermsHash? hash ) [static]

```

Attempts to authenticate a user via the steam API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>steamToken</i>	the user's steam token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()

See also

[GetTermsOfUse](#)

```
// First we get the Terms of Use to display to the user and cache the hash
async void GetTermsOfUse_Example()
{
    ResultAndTermsOfUser response = await ModIOUnityAsync.GetTermsOfUse();
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}

// Once we have the Terms of Use and hash we can attempt to authenticate
async void Authenticate_Example()
{
    Result result = await ModIOUnityAsync.AuthenticateUserViaSteam(steamToken, "johndoe@gmail.com",
        modIOTermsOfUse.hash);
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}
```

8.19.2.13 AuthenticateUserViaSwitch()

```
static async Task< Result > ModIO.ModIOUnityAsync.AuthenticateUserViaSwitch (
    string switchToken,
    string emailAddress,
    TermsHash? hash ) [static]
```

Attempts to authenticate a user via the switch API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>switchToken</i>	the user's steam token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()

See also

[GetTermsOfUse](#)

```
// First we get the Terms of Use to display to the user and cache the hash
async void GetTermsOfUse_Example()
{
    ResultAndTermsOfUser response = await ModIOUnityAsync.GetTermsOfUse();
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}
```

```

    }
}
// Once we have the Terms of Use and hash we can attempt to authenticate
async void Authenticate_Example()
{
    Result result = await ModIOUnityAsync.AuthenticateUserViaItch(switchToken, "johndoe@gmail.com",
        modIOTermsOfUse.hash);
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}
}

```

8.19.2.14 AuthenticateUserViaXbox()

```

static async Task< Result > ModIO.ModIOUnityAsync.AuthenticateUserViaXbox (
    string xboxToken,
    string emailAddress,
    TermsHash? hash ) [static]

```

Attempts to authenticate a user via the Xbox API.

You will first need to get the terms of use and hash from the [ModIOUnity.GetTermsOfUse\(\)](#) method.

Parameters

<i>xboxToken</i>	the user's steam token
<i>emailAddress</i>	the user's email address
<i>hash</i>	the TermsHash retrieved from ModIOUnity.GetTermsOfUse()

See also

[GetTermsOfUse](#)

```

// First we get the Terms of Use to display to the user and cache the hash
async void GetTermsOfUse_Example()
{
    ResultAndTermsOfUser response = await ModIOUnityAsync.GetTermsOfUse();
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
        // Cache the terms of use (which has the hash for when we attempt to authenticate)
        modIOTermsOfUse = response.value;
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}
// Once we have the Terms of Use and hash we can attempt to authenticate
async void Authenticate_Example()
{
    Result result = await ModIOUnityAsync.AuthenticateUserViaItch(xboxToken, "johndoe@gmail.com",
        modIOTermsOfUse.hash);
    if (result.Succeeded())
    {
        Debug.Log("Successfully authenticated user");
    }
    else
    {
        Debug.Log("Failed to authenticate");
    }
}
}

```


8.19.2.15 CreateModProfile()

```
static async Task< ResultAnd< ModId > > ModIO.ModIOUnityAsync.CreateModProfile (
    CreationToken token,
    ModProfileDetails modProfileDetails ) [static]
```

Creates a new mod profile on the mod.io server based on the details provided from the [ModProfileDetails](#) object provided. Note that you must have a logo, name and summary assigned in [ModProfileDetails](#) in order for this to work.

Note that this will create a new profile on the server and can be viewed online through a browser.

Parameters

<i>token</i>	the token allowing a new unique profile to be created from ModIOUnity.GenerateCreationToken()
<i>modProfileDetails</i>	the mod profile details to apply to the mod profile being created

See also

[ModIOUnity.GenerateCreationToken](#), [CreationToken](#), [ModProfileDetails](#), [Result](#), [ModId](#)

```
ModId newMod;
Texture2D logo;
CreationToken token;
async void Example()
{
    token = ModIOUnity.GenerateCreationToken();
    ModProfileDetails profile = new ModProfileDetails();
    profile.name = "mod name";
    profile.summary = "a brief summary about this mod being submitted"
    profile.logo = logo;
    ResultAnd<ModId> response = await ModIOUnityAsync.CreateModProfile(token, profile);
    if (response.result.Succeeded())
    {
        newMod = response.value;
        Debug.Log("created new mod profile with id " + response.value.ToString());
    }
    else
    {
        Debug.Log("failed to create new mod profile");
    }
}
```

8.19.2.16 DeleteModComment()

```
static async Task< Result > ModIO.ModIOUnityAsync.DeleteModComment (
    ModId modId,
    long commentId ) [static]
```

Delete a comment from a mod profile. Successful request will return 204 No Content and fire a MOD_COMMENT↵_DELETED event.

Parameters

<i>modId</i>	Id of the mod to add the comment to
<i>comment↵Id</i>	The id for the comment to be removed

See also

[Result](#), [ModComment](#), [CommentDetails](#), [DeleteModComment](#), [ModIOUnity.DeleteModComment](#), [EditModComment](#)

```
private ModId modId;
private long commentId;
void Example()
{
    var result = await ModIOUnityAsync.DeleteModComment(modId, commentId);
    if (result.Succeeded())
    {
        Debug.Log("deleted comment");
    }
    else
    {
        Debug.Log("failed to delete comment");
    }
}
```

8.19.2.17 DeleteTags()

```
static async Task< Result > ModIO.ModIOUnityAsync.DeleteTags (
    ModId modId,
    string[] tags ) [static]
```

Deletes the specified tags from the mod. In order for this to work the authenticated user must have permission to edit the specified mod.

Parameters

<i>modId</i>	the id of the mod for deleting tags
<i>tags</i>	array of tags to be deleted

See also

[Result](#), [AddTags](#), [ModIOUnityAsync.DeleteTags](#)

```
ModId modId;
string[] tags;
async void Example()
{
    Result result = await ModIOUnity.DeleteTags(modId, tags);
    if (result.Succeeded())
    {
        Debug.Log("deleted tags");
    }
    else
    {
        Debug.Log("failed to delete tags");
    }
}
```

8.19.2.18 DownloadImage()

```
static async Task< ResultAnd< byte[] > > ModIO.ModIOUnityAsync.DownloadImage (
    DownloadReference downloadReference ) [static]
```

Downloads a texture based on the specified download reference.

You can get download references from UserProfiles and ModProfiles

Parameters

<i>downloadReference</i>	download reference for the texture (eg UserObject.avatar_100x100)
--------------------------	---

See also

[Result](#), [DownloadReference](#), [Texture2D](#)

```
ModProfile mod;
async void Example()
{
    ResultAnd#60;Texture2D#62; response = await ModIOUnityAsync.DownloadTexture(mod.logoImage_320x180);
    if (response.result.Succeeded())
    {
        Debug.Log("downloaded the mod logo texture");
    }
    else
    {
        Debug.Log("failed to download the mod logo texture");
    }
}
```

8.19.2.19 EditModProfile()

```
static async Task< Result > ModIO.ModIOUnityAsync.EditModProfile (
    ModProfileDetails modprofile ) [static]
```

This is used to edit or change data in an existing mod profile on the mod.io server.

You need to assign the [ModId](#) of the mod you want to edit inside of the [ModProfileDetails](#) object included in the parameters

Parameters

<i>modprofile</i>	the mod profile details to apply to the mod profile being created
-------------------	---

See also

[ModProfileDetails](#), [Result](#)

```
ModId modId;
async void Example()
{
    ModProfileDetails profile = new ModProfileDetails();
    profile.modId = modId;
    profile.summary = "a new brief summary about this mod being edited"
    Result result = await ModIOUnityAsync.EditModProfile(profile);
    if (result.Succeeded())
    {
        Debug.Log("edited mod profile");
    }
    else
    {
        Debug.Log("failed to edit mod profile");
    }
}
```

8.19.2.20 FetchUpdates()

```
static async Task< Result > ModIO.ModIOUnityAsync.FetchUpdates ( ) [static]
```

This retrieves the user's subscriptions from the mod.io server and synchronises it with our local instance of the user's subscription data. If mod management has been enabled via [ModIOUnity.EnableModManagement\(\)](#) then it may begin to install/uninstall mods.

This requires the current session to have an authenticated user, otherwise `Result.IsAuthenticationError()` from the [Result](#) will equal true.

See also

[Result](#), [ModIOUnity.EnableModManagement\(ModIO.ModManagementEventDelegate\)](#), [IsAuthenticated](#), [RequestAuthenticationEmail](#), [SubmitEmailSecurityCode](#), [AuthenticateUserViaDiscord](#), [AuthenticateUserViaGoogle](#), [AuthenticateUserViaGOG](#), [AuthenticateUserViaItch](#), [AuthenticateUserViaOculus](#), [AuthenticateUserViaSteam](#), [AuthenticateUserViaSwitch](#), [AuthenticateUserViaXbox](#)

```
async void Example()
{
    Result result = await ModIOUnityAsync.FetchUpdates();
    if (result.Succeeded())
    {
        Debug.Log("updated user subscriptions");
    }
    else
    {
        Debug.Log("failed to get user subscriptions");
    }
}
```

8.19.2.21 GetCurrentUser()

```
static async Task< ResultAnd< UserProfile > > ModIO.ModIOUnityAsync.GetCurrentUser ( ) [static]
```

Gets the current user's [UserProfile](#) struct. Containing their mod.io username, user id, language, timezone and download references for their avatar.

This requires the current session to have an authenticated user, otherwise `Result.IsAuthenticationError()` from the [Result](#) will equal true.

See also

[Result](#), [UserProfile](#), [IsAuthenticated](#)

```
async void Example()
{
    ResultAnd<UserProfile> response = await ModIOUnityAsync.GetCurrentUser();
    if (response.result.Succeeded())
    {
        Debug.Log("Got user: " + response.value.username);
    }
    else
    {
        Debug.Log("failed to get user");
    }
}
```

8.19.2.22 GetCurrentUserRatingFor()

```
static async Task< ResultAnd< ModRating > > ModIO.ModIOUnityAsync.GetCurrentUserRatingFor (
    ModId modId ) [static]
```

Gets the rating that the current user has given for a specified mod. You must have an authenticated session for this to be successful.

Note that the rating can be 'None'

Parameters

<i>modId</i>	the id of the mod to check for a rating
--------------	---

See also

[ModRating](#), [ModId](#), [ResultAnd](#)

```
async void Example()
{
    ModId modId = new ModId(1234);
    ResultAnd<ModRating> response = await ModIOUnityAsync.GetCurrentUserRatingFor(modId);
    if (response.result.Succeeded())
    {
        Debug.Log($"retrieved rating: {response.value}");
    }
    else
    {
        Debug.Log("failed to get rating");
    }
}
```

8.19.2.23 GetCurrentUserRatings()

```
static async Task< ResultAnd< Rating[] > > ModIO.ModIOUnityAsync.GetCurrentUserRatings ( )
[static]
```

Get all mod rating's submitted by the authenticated user. Successful request will return an array of Rating Objects.

See also

[ModId](#), [Rating](#), [ResultAnd](#)

```
async void Example()
{
    ResultAnd<Rating[]> response = await ModIOUnityAsync.GetCurrentUserRatings();
    if (response.result.Succeeded())
    {
        foreach(var ratingObject in response.value)
        {
            Debug.Log($"retrieved rating {ratingObject.rating} for {ratingObject.modId}");
        }
    }
    else
    {
        Debug.Log("failed to get ratings");
    }
}
```

8.19.2.24 GetMod()

```
static async Task< ResultAnd< ModProfile > > ModIO.ModIOUnityAsync.GetMod (
    ModId modId ) [static]
```

Requests a single [ModProfile](#) from the mod.io server by its [ModId](#).

If there is a specific mod that you want to retrieve from the mod.io database you can use this method to get it.

Parameters

<i>modId</i>	the ModId of the ModProfile to get
--------------	--

See also

[ModId](#), [ModProfile](#), [Result](#)

```
async void Example()
{
    ModId modId = new ModId(1234);
    ResultAnd<ModProfile> response = await ModIOUnityAsync.GetMod(modId);
    if (response.result.Succeeded())
    {
        Debug.Log("retrieved mod " + response.value.name);
    }
    else
    {
        Debug.Log("failed to get mod");
    }
}
```

8.19.2.25 GetModComments()

```
static async Task< ResultAnd< CommentPage > > ModIO.ModIOUnityAsync.GetModComments (
    ModId modId,
    SearchFilter filter ) [static]
```

Get all comments posted in the mods profile. Successful request will return an array of Comment Objects. We recommended reading the filtering documentation to return only the records you want.

Parameters

<i>filter</i>	The filter to apply when searching through comments (can only apply pagination parameters, Eg. page size and page index)
---------------	--

See also

[CommentPage](#), [ModComment](#), [SearchFilter](#), [ModId](#), [Result](#), [ResultAnd](#), [ModIOUnity.GetModComments](#)

8.19.2.26 GetModDependencies()

```
static async Task< ResultAnd< ModDependencies[] > > ModIO.ModIOUnityAsync.GetModDependencies (
    ModId modId ) [static]
```

8.19.2.27 GetMods()

```
static async Task< ResultAnd< ModPage > > ModIO.ModIOUnityAsync.GetMods (
    SearchFilter filter ) [static]
```

Uses a [SearchFilter](#) to retrieve a specific Mod Page and returns the ModProfiles and total number of mods based on the Search Filter.

A [ModPage](#) contains a group of mods based on the pagination filters in [SearchFilter](#). eg, if you use [SearchFilter.SetPageIndex\(0\)](#) and [SearchFilter.SetPageSize\(100\)](#) then [ModPage.mods](#) will contain mods from 1 to 100. But if you set [SearchFilter.SetPageIndex\(1\)](#) then it will have mods from 101 to 200, if that many exist. (note that 100 is the maximum page size).

Parameters

<i>filter</i>	The filter to apply when searching through mods (also contains pagination parameters)
---------------	---

See also

[SearchFilter](#), [ModPage](#), [Result](#)

```
async void Example()
{
    SearchFilter filter = new SearchFilter();
    filter.SetPageIndex(0);
    filter.SetPageSize(10);
    ResultAnd<ModPage> response = await ModIOUnityAsync.GetMods(filter);
    if (response.result.Succeeded())
    {
        Debug.Log("ModPage has " + response.value.modProfiles.Length + " mods");
    }
    else
    {
        Debug.Log("failed to get mods");
    }
}
```

8.19.2.28 GetTagCategories()

```
static async Task< ResultAnd< TagCategory[] > > ModIO.ModIOUnityAsync.GetTagCategories ( )
[static]
```

Gets the existing tags for the current game Id that can be used when searching/filtering mods.

Tags come in category groups, eg "Color" could be the name of the category and the tags themselves could be { "Red", "Blue", "Green" }

See also

[SearchFilter](#), [TagCategory](#), [Result](#)

```
async void Example()
{
    ResultAnd<TagCategory[]> response = await ModIOUnityAsync.GetTagCategories();
    if (response.result.Succeeded())
    {
        foreach(TagCategory category in response.value)
        {
            foreach(Tag tag in category.tags)
            {
                Debug.Log(tag.name + " tag is in the " + category.name + "category");
            }
        }
    }
    else
    {
        Debug.Log("failed to get game tags");
    }
}
```

8.19.2.29 GetTermsOfUse()

```
static async Task< ResultAnd< TermsOfUse > > ModIO.ModIOUnityAsync.GetTermsOfUse ( ) [static]
```

This retrieves the terms of use text to be shown to the user to accept/deny before authenticating their account via a third party provider, eg steam or google.

If the operation succeeds it will also provide a [TermsOfUse](#) struct that contains a [TermsHash](#) struct which you will need to provide when calling a third party authentication method such as [ModIOUnity.AuthenticateUserViaSteam\(\)](#)

See also

[TermsOfUse](#), [AuthenticateUserViaDiscord](#), [AuthenticateUserViaGoogle](#), [AuthenticateUserViaGOG](#), [AuthenticateUserViaItch](#), [AuthenticateUserViaOculus](#), [AuthenticateUserViaSteam](#), [AuthenticateUserViaSwitch](#), [AuthenticateUserViaXbox](#), [AuthenticateUserViaPlayStation](#)

```
async void Example()
{
    ResultAnd<TermsOfUse> response = await ModIOUnityAsync.GetTermsOfUse();
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully retrieved the terms of use: " + response.value.termsOfUse);
    }
    else
    {
        Debug.Log("Failed to retrieve the terms of use");
    }
}
```

8.19.2.30 IsAuthenticated()

```
static async Task< Result > ModIO.ModIOUnityAsync.IsAuthenticated ( ) [static]
```

Informs you if the current user session is authenticated or not.

See also

[Result](#)

```
async void Example()
{
    Result result = await ModIOUnityAsync.IsAuthenticated();
    if (result.Succeeded())
    {
        Debug.Log("current session is authenticated");
    }
    else
    {
        Debug.Log("current session is not authenticated");
    }
}
```

8.19.2.31 MuteUser()

```
static void ModIO.ModIOUnityAsync.MuteUser (
    long userId ) [static]
```

Mutes a user which effectively hides any content from that specified user

The userId can be found from the [UserProfile](#). Such as `ModProfile.creator.userId`

Parameters

<i>userId</i>	The id of the user to be muted
---------------	--------------------------------

See also

[UserProfile](#)

8.19.2.32 RateMod()

```
static async Task< Result > ModIO.ModIOUnityAsync.RateMod (
    ModId modId,
    ModRating rating ) [static]
```

Used to submit a rating for a specified mod.

This can be used to change/overwrite previous ratings of the current user.

Parameters

<i>modId</i>	the m= ModId of the mod being rated
<i>rating</i>	the rating to give the mod. Allowed values include ModRating.Positive, ModRating.Negative, ModRating.None

See also

[ModRating](#), [Result](#), [ModId](#)

```
ModProfile mod;
async void Example()
{
    Result result = await ModIOUnityAsync.RateMod(mod.id, ModRating.Positive);
    if (result.Succeeded())
    {
        Debug.Log("Successfully rated mod");
    }
    else
    {
        Debug.Log("Failed to rate mod");
    }
}
```

8.19.2.33 RemoveDependenciesFromMod()

```
static async Task< Result > ModIO.ModIOUnityAsync.RemoveDependenciesFromMod (
    ModId modId,
    ICollection< ModId > dependencies ) [static]
```

Removes the specified mods as dependencies for another existing mod.

If the dependencies dont exist they will be ignored and the result will return success

Parameters

<i>modId</i>	ModId of the mod you want to remove dependencies from
<i>dependencies</i>	The ModIds that you want to remove (max 5 at a time)

See also

[Result](#), [ModId](#), [dependencies](#), [ModIOUnity.AddDependenciesToMod](#), [ModIOUnity.RemoveDependenciesFromMod](#), [ModIOUnityAsync.AddDependenciesToMod](#)

```
void Example()
{
    var dependencies = new List<ModId>()
    {
        (ModId)1001,
        (ModId)1002,
        (ModId)1003
    };
    Result result = await ModIOUnityAsync.RemoveDependenciesFromMod(mod.id, dependencies);
    if (result.Succeeded())
    {
        Debug.Log("Successfully removed dependencies from mod");
    }
    else
    {
        Debug.Log("Failed to removed dependencies from mod");
    }
}
```

8.19.2.34 Report()

```
static async Task< Result > ModIO.ModIOUnityAsync.Report (
    Report report ) [static]
```

Reports a specified mod to mod.io.

Parameters

<i>report</i>	the object containing all of the details of the report you are sending
---------------	--

See also

[Report](#), [Result](#)

```
async void Example()
{
    Report report = new Report(new ModId(123),
        ReportType.Generic,
        "reporting this mod for a generic reason",
        "JohnDoe",
        "johndoe@mod.io");
    Result result = await ModIOUnityAsync.Report(report);
    if (result.Succeeded())
    {
        Debug.Log("successfully sent a report");
    }
    else
    {
        Debug.Log("failed to send a report");
    }
}
```

8.19.2.35 RequestAuthenticationEmail()

```
static async Task< Result > ModIO.ModIOUnityAsync.RequestAuthenticationEmail (
    string emailAddress ) [static]
```

Sends an email with a security code to the specified Email Address. The security code is then used to Authenticate the user session using [ModIOUnity.SubmitEmailSecurityCode\(\)](#)

The operation will return a [Result](#) object. If the email is successfully sent [Result.Succeeded\(\)](#) will equal true. If you haven't Initialized the plugin then [Result.IsInitializationError\(\)](#) will equal true. If the string provided for the emailAddress is not .NET compliant [Result.IsAuthenticationError\(\)](#) will equal true.

Parameters

<i>emailaddress</i>	the Email Address to send the security code to, eg "JohnDoe@gmail.com"
---------------------	--

See also

[SubmitEmailSecurityCode](#), [Result](#)

```
async void Example()
{
    Result result = await ModIOUnityAsync.RequestAuthenticationEmail("johndoe@gmail.com");
    if (result.Succeeded())
    {
        Debug.Log("Succeeded to send security code");
    }
    else
    {
        Debug.Log("Failed to send security code to that email address");
    }
}
```

8.19.2.36 Shutdown()

```
static async Task ModIO.ModIOUnityAsync.Shutdown ( ) [static]
```

Cancels any running public operations, frees plugin resources, and invokes any pending callbacks with a cancelled result code.

pending operations during a shutdown can be checked with [Result.IsCancelled\(\)](#)

See also

[Result](#)

```
async void Example()
{
    await ModIOUnityAsync.Shutdown();
    Debug.Log("Finished shutting down the ModIO Plugin");
}
```

8.19.2.37 SubmitEmailSecurityCode()

```
static async Task< Result > ModIO.ModIOUnityAsync.SubmitEmailSecurityCode (
    string securityCode ) [static]
```

Attempts to Authenticate the current session by submitting a security code received by email from [ModIOUnity.RequestAuthenticationEmail\(\)](#)

It is intended that this function is used after [ModIOUnity.RequestAuthenticationEmail\(\)](#) is performed successfully.

Parameters

<i>securityCode</i>	The security code received from an authentication email
---------------------	---

See also

[RequestAuthenticationEmail](#), [Result](#)

```

async void Example(string userSecurityCode)
{
    Result result = await ModIOUnityAsync.SubmitEmailSecurityCode(userSecurityCode);
    if (result.Succeeded())
    {
        Debug.Log("You have successfully authenticated the user");
    }
    else
    {
        Debug.Log("Failed to authenticate the user");
    }
}

```

8.19.2.38 SubscribeToMod()

```

static async Task< Result > ModIO.ModIOUnityAsync.SubscribeToMod (
    ModId modId ) [static]

```

Adds the specified mod to the current user's subscriptions.

If mod management has been enabled via [ModIOUnity.EnableModManagement\(\)](#) then the mod will be downloaded and installed.

Parameters

<i>modId</i>	ModId of the mod you want to subscribe to
--------------	---

See also

[Result](#), [ModId](#), [ModIOUnity.EnableModManagement\(ModIO.ModManagementEventDelegate\)](#), [ModIOUnity.GetCurrentModMan](#)

```

ModProfile mod;
async void Example()
{
    Result result = await ModIOUnityAsync.SubscribeToMod(mod.id);
    if (result.Succeeded())
    {
        Debug.Log("Successfully subscribed to mod");
    }
    else
    {
        Debug.Log("Failed to subscribe to mod");
    }
}

```

8.19.2.39 UnmuteUser()

```

static void ModIO.ModIOUnityAsync.UnmuteUser (
    long userId ) [static]

```

Un-mutes a user which effectively reveals previously hidden content from that user

The `userId` can be found from the [UserProfile](#). Such as `ModProfile.creator.userId`

Parameters

<i>userId</i>	The id of the user to be muted
---------------	--------------------------------

See also

[UserProfile](#)

8.19.2.40 UnsubscribeFromMod()

```
static async Task< Result > ModIO.ModIOUnityAsync.UnsubscribeFromMod (
    ModId modId ) [static]
```

Removes the specified mod from the current user's subscriptions.

If mod management has been enabled via [ModIOUnity.EnableModManagement\(\)](#) then the mod will be uninstalled at the next opportunity.

Parameters

<i>modId</i>	ModId of the mod you want to unsubscribe from
--------------	---

See also

[Result](#), [ModId](#), [ModIOUnity.EnableModManagement\(ModIO.ModManagementEventDelegate\)](#), [ModIOUnity.GetCurrentModMan](#)

```
ModProfile mod;
async void Example()
{
    Result result = await ModIOUnityAsync.UnsubscribeFromMod(mod.id);
    if (result.Succeeded())
    {
        Debug.Log("Successfully unsubscribed from mod");
    }
    else
    {
        Debug.Log("Failed to unsubscribe from mod");
    }
}
```

8.19.2.41 UpdateModComment()

```
static async Task< ResultAnd< ModComment > > ModIO.ModIOUnityAsync.UpdateModComment (
    ModId modId,
    string content,
    long commentId ) [static]
```

Update a comment for the corresponding mod. Successful request will return the updated Comment Object.

Parameters

<i>modId</i>	Id of the mod the comment is on
<i>content</i>	Updated contents of the comment.
<i>commentId</i>	The id for the comment you wish to edit

See also

[ResultAnd](#), [ModComment](#), [ModIOUnity.UpdateModComment](#)

```
private string content = "This is a Comment";
long commentId = 12345;
ModId modId = (ModId)1234;
async void UpdateMod()
{
    var response = await ModIOUnityAsync.UpdateModComment(modId, content, commentId);
    if (response.result.Succeeded())
    {
        Debug.Log("Successfully Updated Comment!");
    }
    else
    {
        Debug.Log("Failed to Update Comment!");
    }
}
```

8.19.2.42 UploadModfile()

```
static async Task< Result > ModIO.ModIOUnityAsync.UploadModfile (
    ModfileDetails modfile ) [static]
```

Used to upload a mod file to a mod profile on the mod.io server. A mod file is the actual archive of a mod. This method can be used to update a mod to a newer version (you can include changelog information in [ModfileDetails](#)).

Parameters

<i>modfile</i>	the mod file and details to upload
----------------	------------------------------------

See also

[Result](#), [ModfileDetails](#), [ArchiveModProfile](#), [ModIOUnity.GetCurrentUploadHandle](#)

```
ModId modId;
async void Example()
{
    ModfileDetails modfile = new ModfileDetails();
    modfile.modId = modId;
    modfile.directory = "files/mods/mod_123";
    Result result = await ModIOUnityAsync.UploadModfile(modfile);
    if (result.Succeeded())
    {
        Debug.Log("uploaded mod file");
    }
    else
    {
        Debug.Log("failed to upload mod file");
    }
}
```

8.19.2.43 UploadModMedia()

```
static async Task< Result > ModIO.ModIOUnityAsync.UploadModMedia (
    ModProfileDetails modProfileDetails ) [static]
```

This is used to update the logo of a mod or the gallery images. This works very similar to EditModProfile except it only affects the images.

Parameters

<i>modProfileDetails</i>	this holds the reference to the images you wish to upload
--------------------------	---

See also

[ModProfileDetails](#), [Result](#), [EditModProfile](#)

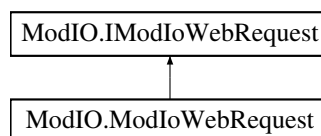
```
ModId modId;
Texture2D newTexture;
async void Example()
{
    ModProfileDetails profile = new ModProfileDetails();
    profile.modId = modId;
    profile.logo = newTexture;
    Result result = await ModIOUnityAsync.UploadModMedia(profile);
    if (result.Succeeded())
    {
        Debug.Log("uploaded new mod logo");
    }
    else
    {
        Debug.Log("failed to uploaded mod logo");
    }
}
```

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/ModIOUnityAsync.cs

8.20 ModIO.ModIoWebRequest Class Reference

Inheritance diagram for ModIO.ModIoWebRequest:



Public Member Functions

- **ModIoWebRequest** (UnityWebRequest unityWebRequest)
- string [GetResponseHeader](#) (string name)

Public Attributes

- UnityWebRequest **unityWebRequest**

Properties

- bool [isDone](#) [get]
- ulong [downloadedBytes](#) [get]
- float [downloadProgress](#) [get]
- float [uploadProgress](#) [get]
- ulong [uploadedBytes](#) [get]

8.20.1 Member Function Documentation

8.20.1.1 GetResponseHeader()

```
string ModIO.ModIoWebRequest.GetResponseHeader (  
    string name )
```

Implements [ModIO.IModIoWebRequest](#).

8.20.2 Property Documentation

8.20.2.1 downloadedBytes

```
ulong ModIO.ModIoWebRequest.downloadedBytes [get]
```

Implements [ModIO.IModIoWebRequest](#).

8.20.2.2 downloadProgress

```
float ModIO.ModIoWebRequest.downloadProgress [get]
```

Implements [ModIO.IModIoWebRequest](#).

8.20.2.3 isDone

```
bool ModIO.ModIoWebRequest.isDone [get]
```

Implements [ModIO.IModIoWebRequest](#).

8.20.2.4 uploadedBytes

```
ulong ModIO.ModIoWebRequest.uploadedBytes [get]
```

Implements [ModIO.IModIoWebRequest](#).

8.20.2.5 uploadProgress

```
float ModIO.ModIoWebRequest.uploadProgress [get]
```

Implements [ModIO.IModIoWebRequest](#).

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Classes/ModIoWebRequest.cs

8.21 ModIO.ModPage Struct Reference

A struct containing the ModProfiles and total number of remaining results that can be acquired with the [SearchFilter](#) used in the GetMods request.

Public Attributes

- [ModProfile\[\]](#) `modProfiles`
The mod profiles retrieved from this pagination request
- long [totalSearchResultsFound](#)
the total results that could be found. eg there may be a total of 1,000 mod profiles but this [ModPage](#) may only contain the first 100, depending on the [SearchFilter](#) pagination settings.

8.21.1 Detailed Description

A struct containing the ModProfiles and total number of remaining results that can be acquired with the [SearchFilter](#) used in the GetMods request.

See also

[ModIOUnity.GetMods](#), [ModIOUnityAsync.GetMods](#)

8.21.2 Member Data Documentation

8.21.2.1 modProfiles

[ModProfile](#) [] `ModIO.ModPage.modProfiles`

The mod profiles retrieved from this pagination request

See also

[ModIOUnity.GetMods](#), [ModIOUnityAsync.GetMods](#)

8.21.2.2 totalSearchResultsFound

`long ModIO.ModPage.totalSearchResultsFound`

the total results that could be found. eg there may be a total of 1,000 mod profiles but this [ModPage](#) may only contain the first 100, depending on the [SearchFilter](#) pagination settings.

See also

[SearchFilter](#), [SearchFilter.SetPageIndex](#), [SearchFilter.SetPageSize](#), [ModIOUnity.GetMods](#), [ModIOUnityAsync.GetMods](#)

The documentation for this struct was generated from the following file:

- `Assets/Plugins/mod.io/Runtime/Structs/ModPage.cs`

8.22 ModIO.ModProfile Struct Reference

A struct representing all of the information available for a [ModProfile](#).

Public Attributes

- [ModId](#) **id**
- `string[]` **tags**
- `ModStatus` **status**
- `bool` **visible**
- `string` **name**
- `string` **summary**
- `string` **description**
- `string` **homePageUrl**
- `ContentWarnings` **contentWarnings**
- `DateTime` **dateAdded**
- `DateTime` **dateUpdated**
- `DateTime` **dateLive**
- [DownloadReference](#)[] **galleryImages_Original**
- [DownloadReference](#)[] **galleryImages_320x180**
- [DownloadReference](#)[] **galleryImages_640x360**
- [DownloadReference](#) **logoImage_320x180**

- [DownloadReference](#) **logImage_640x360**
- [DownloadReference](#) **logImage_1280x720**
- [DownloadReference](#) **logImage_Original**
- [UserProfile](#) **creator**
- [DownloadReference](#) **creatorAvatar_50x50**
- [DownloadReference](#) **creatorAvatar_100x100**
- [DownloadReference](#) **creatorAvatar_Original**
- string **metadata**
The meta data for this mod, not to be confused with the meta data of the specific version
- string **latestVersion**
The most recent version of the mod that exists
- string **latestChangelog**
the change log for the most recent version of this mod
- DateTime **latestDateFileAdded**
the date for when the most recent mod file was uploaded
- KeyValuePair< string, string >[] **metadataKeyValuePairs**
the KVP meta data for this mod profile. Not to be confused with the meta data blob or the meta data for the installed version of the mod
- [ModStats](#) **stats**
- long **archiveFileSize**

8.22.1 Detailed Description

A struct representing all of the information available for a [ModProfile](#).

See also

[ModIOUnity.GetMod](#), [ModIOUnityAsync.GetMod](#)

8.22.2 Member Data Documentation

8.22.2.1 metadata

```
string ModIO.ModProfile.metadata
```

The meta data for this mod, not to be confused with the meta data of the specific version

See also

[InstalledMod](#)

The documentation for this struct was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/ModProfile.cs

8.23 ModIO.ModProfileDetails Class Reference

Use this class to fill out the details of a Mod Profile that you'd like to create or edit. If you're submitting this via CreateModProfile you must assign values to logo, name and summary, otherwise the submission will be rejected (All fields except modId are optional if submitting this via EditModProfile)

Public Attributes

- **ModId?** [modId](#)
Make sure to set this field when submitting a request to Edit a Mod Profile
- **bool?** [visible](#)
Whether this mod will appear as public or hidden.
- **byte[]** [logo](#)
Image file which will represent your mods logo. Must be gif, jpg or png format and cannot exceed 8MB in filesize. Dimensions must be at least 512x288 and we recommend you supply a high resolution image with a 16 / 9 ratio. mod.io will use this image to make three thumbnails for the dimensions 320x180, 640x360 and 1280x720
- **List< byte[] >** [images](#)
Image files that will be included in the mod profile details.
- **string** [name](#)
Name of your mod
- **string** [name_id](#)
Path for the mod on mod.io. For example: <https://gamenname.mod.io/mod-name-id-here>. If no name_id is specified the [name](#) will be used. For example: 'Stellaris Shader Mod' will become 'stellaris-shader-mod'. Cannot exceed 80 characters
- **string** [summary](#)
Summary for your mod, giving a brief overview of what it's about. Cannot exceed 250 characters.
- **string** [description](#)
Detailed description for your mod, which can include details such as 'About', 'Features', 'Install Instructions', 'FAQ', etc. HTML supported and encouraged
- **string** [homepage_url](#)
Official homepage for your mod. Must be a valid URL
- **int?** [maxSubscribers](#)
This will create a cap on the number of subscribers for this mod. Set to 0 to allow for infinite subscribers.
- **ContentWarnings?** [contentWarning](#)
This is a Bitwise enum so you can assign multiple values
- **string** [metadata](#)
Your own custom metadata that can be uploaded with the mod profile. (This is for the entire mod profile, a unique metadata field can be assigned to each modfile as well)
- **string[]** [tags](#)
The tags this mod profile has. Only tags that are supported by the parent game can be applied. (Invalid tags will be ignored)
- **CommunityOptions?** [communityOptions](#) = CommunityOptions.AllowCommenting
Select which interactions players can have with your mod. 0 = None 1 = Ability to comment (default) ? = Add the options you want together, to enable multiple options

8.23.1 Detailed Description

Use this class to fill out the details of a Mod Profile that you'd like to create or edit. If you're submitting this via CreateModProfile you must assign values to logo, name and summary, otherwise the submission will be rejected (All fields except modId are optional if submitting this via EditModProfile)

See also

[ModIOUnity.CreateModProfile](#), [ModIOUnity.EditModProfile](#)

8.23.2 Member Data Documentation

8.23.2.1 communityOptions

CommunityOptions? ModIO.ModProfileDetails.communityOptions = CommunityOptions.AllowCommenting

Select which interactions players can have with your mod. 0 = None 1 = Ability to comment (default) ? = Add the options you want together, to enable multiple options

Can be null

8.23.2.2 contentWarning

ContentWarnings? ModIO.ModProfileDetails.contentWarning

This is a Bitwise enum so you can assign multiple values

See also

ContentWarnings

Can be null

8.23.2.3 description

string ModIO.ModProfileDetails.description

Detailed description for your mod, which can include details such as 'About', 'Features', 'Install Instructions', 'FAQ', etc. HTML supported and encouraged

Can be null

8.23.2.4 homepage_url

string ModIO.ModProfileDetails.homepage_url

Official homepage for your mod. Must be a valid URL

Can be null

8.23.2.5 images

List<byte[]> ModIO.ModProfileDetails.images

Image files that will be included in the mod profile details.

Can be null

8.23.2.6 logo

```
byte [] ModIO.ModProfileDetails.logo
```

Image file which will represent your mods logo. Must be gif, jpg or png format and cannot exceed 8MB in filesize. Dimensions must be at least 512x288 and we recommend you supply a high resolution image with a 16 / 9 ratio. mod.io will use this image to make three thumbnails for the dimensions 320x180, 640x360 and 1280x720

Can be null if using EditModProfile

See also

[ModIOUnity.EditModProfile](#)

8.23.2.7 maxSubscribers

```
int? ModIO.ModProfileDetails.maxSubscribers
```

This will create a cap on the number of subscribers for this mod. Set to 0 to allow for infinite subscribers.

Can be null

8.23.2.8 metadata

```
string ModIO.ModProfileDetails.metadata
```

Your own custom metadata that can be uploaded with the mod profile. (This is for the entire mod profile, a unique metadata field can be assigned to each modfile as well)

See also

[ModfileDetails](#)

the metadata has a maximum size of 50,000 characters.

Can be null

8.23.2.9 modId

```
ModId? ModIO.ModProfileDetails.modId
```

Make sure to set this field when submitting a request to Edit a Mod Profile

Can be null

8.23.2.10 name

```
string ModIO.ModProfileDetails.name
```

Name of your mod

Can be null if using EditModProfile

See also

[ModIOUnity.EditModProfile](#)

8.23.2.11 name_id

```
string ModIO.ModProfileDetails.name_id
```

Path for the mod on mod.io. For example: <https://gamename.mod.io/mod-name-id-here>. If no name_id is specified the [name](#) will be used. For example: 'Stellaris Shader Mod' will become 'stellaris-shader-mod'. Cannot exceed 80 characters

Can be null

8.23.2.12 summary

```
string ModIO.ModProfileDetails.summary
```

Summary for your mod, giving a brief overview of what it's about. Cannot exceed 250 characters.

This field must be assigned when submitting a new Mod Profile

Can be null if using EditModProfile

See also

[ModIOUnity.EditModProfile](#)

8.23.2.13 tags

```
string [] ModIO.ModProfileDetails.tags
```

The tags this mod profile has. Only tags that are supported by the parent game can be applied. (Invalid tags will be ignored)

Can be null

8.23.2.14 visible

```
bool? ModIO.ModProfileDetails.visible
```

Whether this mod will appear as public or hidden.

Can be null

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Classes/ModProfileDetails.cs

8.24 ModIO.ModStats Struct Reference

Detailed stats about a Mod's ratings, downloads, subscribers, popularity etc

Public Attributes

- [ModId](#) **modId**
- long **popularityRankPosition**
- long **popularityRankTotalMods**
- long **downloadsToday**
- long **downloadsTotal**
- long **subscriberTotal**
- long **ratingsTotal**
- long **ratingsPositive**
- long **ratingsNegative**
- long **ratingsPercentagePositive**
- float **ratingsWeightedAggregate**
- string **ratingsDisplayText**

8.24.1 Detailed Description

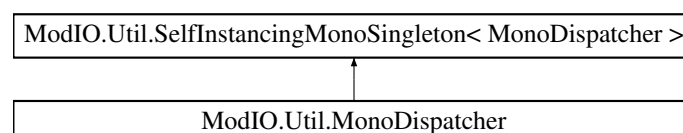
Detailed stats about a Mod's ratings, downloads, subscribers, popularity etc

The documentation for this struct was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/ModStats.cs

8.25 ModIO.Util.MonoDispatcher Class Reference

Inheritance diagram for ModIO.Util.MonoDispatcher:



Public Member Functions

- bool **MainThread** ()
- void **Run** (Action action)
- Task **EnqueueAsync** (Action action)

Protected Member Functions

- override void **Awake** ()

Private Member Functions

- void **Update** ()

Private Attributes

- Thread **mainThread**
- readonly ConcurrentQueue< Action > **actions** = new ConcurrentQueue<Action>()

Additional Inherited Members

8.25.1 Member Function Documentation

8.25.1.1 Awake()

```
override void ModIO.Util.MonoDispatcher.Awake ( ) [protected], [virtual]
```

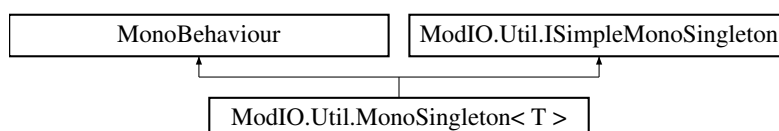
Reimplemented from [ModIO.Util.SelfInstantingMonoSingleton< MonoDispatcher >](#).

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Utility/MonoDispatcher.cs

8.26 ModIO.Util.MonoSingleton< T > Class Template Reference

Inheritance diagram for ModIO.Util.MonoSingleton< T >:



Public Member Functions

- void [SetupSingleton](#) ()

Static Public Member Functions

- static bool **SingletonIsInstantiated** ()

Protected Member Functions

- virtual void **Awake** ()
- virtual void **OnDestroy** ()
- virtual void **OnApplicationQuit** ()

Static Protected Attributes

- static T **_instance**

Properties

- static T? **Instance** [get, private set]

8.26.1 Member Function Documentation

8.26.1.1 SetupSingleton()

```
void ModIO.Util.MonoSingleton< T >.SetupSingleton ( )
```

Implements [ModIO.Util.ISimpleMonoSingleton](#).

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Utility/MonoSingleton.cs

8.27 ModIO.Util.Mutex Class Reference

This serves only as an abstract handle for using lock(mutex) to synchronize IO operations

8.27.1 Detailed Description

This serves only as an abstract handle for using lock(mutex) to synchronize IO operations

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Utility/Mutex.cs

8.28 ModIO.ProgressHandle Class Reference

A [ProgressHandle](#) can only be used to monitor the progress of an operation and cannot be used to cancel or suspend ongoing operations. The `OperationType` enum field specifies what type of operation this handle is for. The `Progress` field can be used to get the percentage (0.0 - 1.0) of the progress. The `Completed` and `Failed` fields can be used to determine if the operation is complete and whether or not it failed.

Properties

- `ModId modId` [get, set]
The [ModId](#) of the mod that this operation pertains to.
- `ModManagementOperationType OperationType` [get, set]
The type of operation being performed, eg. Download, Upload, Install
- `float Progress` [get, set]
The progress of the operation being performed, float range from 0.0f - 1.0f
- `long BytesPerSecond` [get, set]
The average number of bytes being processed per second by the operation (Updated every 10 milliseconds)
- `bool Completed` [get, set]
Is set to True when the operation has finished
- `bool Failed` [get, set]
Is set to True if the operation encounters an error or is cancelled before completion

8.28.1 Detailed Description

A [ProgressHandle](#) can only be used to monitor the progress of an operation and cannot be used to cancel or suspend ongoing operations. The `OperationType` enum field specifies what type of operation this handle is for. The `Progress` field can be used to get the percentage (0.0 - 1.0) of the progress. The `Completed` and `Failed` fields can be used to determine if the operation is complete and whether or not it failed.

8.28.2 Property Documentation

8.28.2.1 BytesPerSecond

```
long ModIO.ProgressHandle.BytesPerSecond [get], [set]
```

The average number of bytes being processed per second by the operation (Updated every 10 milliseconds)

Only applicable to Download and Upload operations

8.28.2.2 Completed

```
bool ModIO.ProgressHandle.Completed [get], [set]
```

Is set to True when the operation has finished

If an operation fails then Completed will still be True, therefore it is recommended to check Failed as well

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Classes/ProgressHandle.cs

8.29 ModIO.Implementation.API.Objects.Rating Struct Reference

A struct representing all of the information available for a [Rating](#).

Public Attributes

- [ModId](#) **modId**
- [ModRating](#) **rating**
- [DateTime](#) **dateAdded**

8.29.1 Detailed Description

A struct representing all of the information available for a [Rating](#).

See also

[ModIOUnity.GetCurrentUserRatings](#), [ModIOUnityAsync.GetCurrentUserRatings](#), [RatingObject](#)

The documentation for this struct was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/Rating.cs

8.30 ModIO.Implementation.API.Objects.RatingObject Struct Reference

A struct representing all of the information available for a [ModDependenciesObject](#).

Public Attributes

- uint **game_id**
- long **mod_id**
- int **rating**
- long **date_added**

8.30.1 Detailed Description

A struct representing all of the information available for a [ModDependenciesObject](#).

See also

[ModIOUnity.GetCurrentUserRatings](#), [ModIOUnityAsync.GetCurrentUserRatings](#)

The documentation for this struct was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/RatingObject.cs

8.31 ModIO.Report Class Reference

Used in conjunction with [ModIOUnity.Report\(\)](#) to send a report to the mod.io server for a specific mod.

Public Member Functions

- [Report](#) ([ModId](#) modId, ReportType type, string summary, string user, string contactEmail)
convenience constructor for making a report. All of the parameters are mandatory to make a successful report.
- bool **CanSend** ()

Public Attributes

- long? **id**
- string **summary**
- ReportType? **type**
- ReportResourceType? **resourceType**
- string **user**
- string **contactEmail**

8.31.1 Detailed Description

Used in conjunction with [ModIOUnity.Report\(\)](#) to send a report to the mod.io server for a specific mod.

8.31.2 Constructor & Destructor Documentation

8.31.2.1 Report()

```
ModIO.Report.Report (
    ModId modId,
    ReportType type,
    string summary,
    string user,
    string contactEmail )
```

convenience constructor for making a report. All of the parameters are mandatory to make a successful report.

Parameters

<i>modId</i>	the id of the mod being reported
<i>type</i>	the type of report
<i>summary</i>	CANNOT BE NULL explanation of the issue being reported
<i>user</i>	CANNOT BE NULL user reporting the issue
<i>contactEmail</i>	CANNOT BE NULL user email address

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Classes/Report.cs

8.32 ModIO.Result Struct Reference

Struct returned from [ModIO](#) callbacks to inform the caller if the operation succeeded.

Public Member Functions

- bool **Succeeded** ()
- bool **IsCancelled** ()
- bool **IsInitializationError** ()
- bool **IsAuthenticationError** ()
- bool **IsInvalidSecurityCode** ()
- bool **IsInvalidEmailAddress** ()
- bool **IsPermissionError** ()
- bool **IsNetworkError** ()
Checks if the result failed due to no internet connection
- bool **IsStorageSpaceInsufficient** ()

Properties

- string **message** [get]
A string message explaining the result error code in more detail (If one exists).
- uint **errorCode** [get]
The error code for the result. 0 = Success

8.32.1 Detailed Description

Struct returned from [ModIO](#) callbacks to inform the caller if the operation succeeded.

8.32.2 Member Function Documentation

8.32.2.1 IsNetworkError()

```
bool ModIO.Result.IsNetworkError ( )
```

Checks if the result failed due to no internet connection

Returns

true if the result failed due to no internet connection

The documentation for this struct was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/Result.cs

8.33 ModIO.ResultAnd< T > Class Template Reference

Convenience wrapper for essentially a Tuple.

Public Attributes

- [Result](#) result
- T value

8.33.1 Detailed Description

Convenience wrapper for essentially a Tuple.

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Classes/ResultAnd.cs

8.34 ModIO.SearchFilter Class Reference

Used to build a filter that is sent with requests for retrieving mods.

Public Member Functions

- void [AddSearchPhrase](#) (string phrase)
Adds a phrase into the filter to be used when filtering mods in a request.
- void [AddTag](#) (string tag)
Adds a tag to be used in filtering mods for a request.
- void [SortBy](#) ([SortModsBy](#) category)
Determines what category mods should be sorted and returned by. eg if the category [SortModsBy.Downloads](#) was used, then the results would be returned by the number of downloads. Depending on the Ascending or Descending setting, it will start or end with mods that have the highest or lowest number of downloads.
- void [SetToAscending](#) (bool isAscending)
Determines the order of the results being returned. eg should results be filtered from highest to lowest, or lowest to highest.
- void [SetPageIndex](#) (int pageIndex)
Sets the zero based index of the page. eg if there are 1,000 results based on the filter settings provided, and the page size is 100. Setting this to 1 will return the mods from 100-200. Whereas setting this to 0 will return the first 100 results.
- void [SetPageSize](#) (int pageSize)
Sets the maximum page size of the request. eg if there are 50 results and the index is set to 0. If the page size is set to 10 you will receive the first 10 results. If the page size is set to 100 you will only receive the total 50 results, because there are no more to be got.
- void [AddUser](#) (long userId)
Adds a specific user to the filter, so that mods that were not created by the user (or other users added to the filter) will not be returned.
- bool [IsSearchFilterValid](#) (out [Result](#) result)
You can use this method to check if a search filter is setup correctly before using it in a [GetMods](#) request.

Private Attributes

- bool **hasPageIndexBeenSet** = false
- bool **hasPageSizeBeenSet** = false

8.34.1 Detailed Description

Used to build a filter that is sent with requests for retrieving mods.

See also

[ModIOUnity.GetMods](#), [ModIOUnityAsync.GetMods](#)

8.34.2 Member Function Documentation

8.34.2.1 AddSearchPhrase()

```
void ModIO.SearchFilter.AddSearchPhrase (
    string phrase )
```

Adds a phrase into the filter to be used when filtering mods in a request.

Parameters

<i>phrase</i>	the string to be added to the filter
---------------	--------------------------------------

8.34.2.2 AddTag()

```
void ModIO.SearchFilter.AddTag (
    string tag )
```

Adds a tag to be used in filtering mods for a request.

Parameters

<i>tag</i>	the tag to be added to the filter
------------	-----------------------------------

See also

[Tag](#), [TagCategory](#)

8.34.2.3 AddUser()

```
void ModIO.SearchFilter.AddUser (
    long userId )
```

Adds a specific user to the filter, so that mods that were not created by the user (or other users added to the filter) will not be returned.

Parameters

<i>userId</i>	Id of the user to add
---------------	-----------------------

See also

[UserProfile](#)

8.34.2.4 IsSearchFilterValid()

```
bool ModIO.SearchFilter.IsSearchFilterValid (
    out Result result )
```

You can use this method to check if a search filter is setup correctly before using it in a GetMods request.

Parameters

<i>result</i>	
---------------	--

Returns

true if the filter is valid

See also

[ModIOUnity.GetMods](#), [ModIOUnityAsync.GetMods](#)

8.34.2.5 SetPageIndex()

```
void ModIO.SearchFilter.SetPageIndex (
    int pageIndex )
```

Sets the zero based index of the page. eg if there are 1,000 results based on the filter settings provided, and the page size is 100. Setting this to 1 will return the mods from 100-200. Whereas setting this to 0 will return the first 100 results.

Parameters

<i>pageIndex</i>	
------------------	--

See also

[SetPageSize](#)

8.34.2.6 SetPageSize()

```
void ModIO.SearchFilter.SetPageSize (
    int pageSize )
```

Sets the maximum page size of the request. eg if there are 50 results and the index is set to 0. If the page size is set to 10 you will receive the first 10 results. If the page size is set to 100 you will only receive the total 50 results, because there are no more to be got.

Parameters

<i>pageSize</i>	
-----------------	--

See also

[SetPageIndex](#)

8.34.2.7 SetToAscending()

```
void ModIO.SearchFilter.SetToAscending (
    bool isAscending )
```

Determines the order of the results being returned. eg should results be filtered from highest to lowest, or lowest to highest.

Parameters

<i>isAscending</i>	
--------------------	--

8.34.2.8 SortBy()

```
void ModIO.SearchFilter.SortBy (
    SortModsBy category )
```

Determines what category mods should be sorted and returned by. eg if the category SortModsBy.Downloads was used, then the results would be returned by the number of downloads. Depending on the Ascending or Descending setting, it will start or end with mods that have the highest or lowest number of downloads.

Parameters

<i>category</i>	the category to sort the request
-----------------	----------------------------------

See also

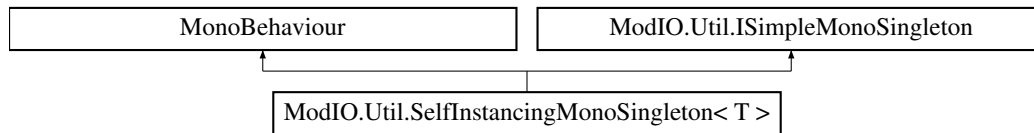
[SetToAscending](#)

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Classes/SearchFilter.cs

8.35 ModIO.Util.SelfInstantiatingMonoSingleton< T > Class Template Reference

Inheritance diagram for ModIO.Util.SelfInstantiatingMonoSingleton< T >:



Public Member Functions

- void [SetupSingleton](#) ()

Static Public Member Functions

- static bool **SingletonIsInstantiated** ()

Protected Member Functions

- virtual void **Awake** ()
- virtual void **OnDestroy** ()
- virtual void **OnApplicationQuit** ()

Static Protected Attributes

- static T **_instance**

Properties

- static T **Instance** [get, private set]

8.35.1 Member Function Documentation

8.35.1.1 SetupSingleton()

void [ModIO.Util.SelfInstantiatingMonoSingleton](#)< T >.SetupSingleton ()

Implements [ModIO.Util.ISimpleMonoSingleton](#).

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Utility/SelfInstantiatingMonoSingleton.cs

8.36 ModIO.ServerSettings Struct Reference

Describes the server settings to use for the [ModIO](#) Plugin. This can be setup directly from the inspector when editing the config settings file, or you can instantiate and use this at runtime with the Initialize method

Public Member Functions

- **ServerSettings** ([ServerSettings](#) serverSettings)

Public Attributes

- string **serverURL**
URL for the mod.io server to connect to.
- uint **gameId**
Game Id as can be found on mod.io Web UI.
- string **gameKey**
mod.io Service API Key used by your game to connect.
- string **languageCode**
Language code for the localizing message responses. See <https://docs.mod.io/#localization> for possible values.
- bool **disableUploads**
Disables uploading mods and modfiles for this build.

8.36.1 Detailed Description

Describes the server settings to use for the [ModIO](#) Plugin. This can be setup directly from the inspector when editing the config settings file, or you can instantiate and use this at runtime with the Initialize method

See also

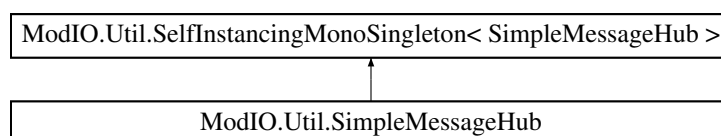
[BuildSettings](#), [ModIOUnity.InitializeForUser](#), [ModIOUnityAsync.InitializeForUser](#)

The documentation for this struct was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/ServerSettings.cs

8.37 ModIO.Util.SimpleMessageHub Class Reference

Inheritance diagram for ModIO.Util.SimpleMessageHub:



Public Member Functions

- [SimpleMessageUnsubscribeToken Subscribe< T >](#) (Action< T > subscription)
- void [Publish< T >](#) (T message)
- void [PublishThreadSafe< T >](#) (T message)
- void [ClearTypeSubscriptions< T >](#) ()

Protected Member Functions

- override void [OnDestroy](#) ()

Private Member Functions

- void [Update](#) ()

Private Attributes

- readonly Dictionary< Type, List< Action< [ISimpleMessage](#) > > > [dictionary](#)
- List< [ISimpleMessage](#) > [threadSafeMessages](#) = new List<[ISimpleMessage](#)>()

Additional Inherited Members

8.37.1 Member Function Documentation

8.37.1.1 OnDestroy()

```
override void ModIO.Util.SimpleMessageHub.OnDestroy ( ) [protected], [virtual]
```

Reimplemented from [ModIO.Util.SelfInstantiatingMonoSingleton< SimpleMessageHub >](#).

8.37.1.2 Publish< T >()

```
void ModIO.Util.SimpleMessageHub.Publish< T > (
    T message )
```

Type Constraints

T: *class*

T: *ISimpleMessage*

8.37.1.3 PublishThreadSafe< T >()

```
void ModIO.Util.SimpleMessageHub.PublishThreadSafe< T > (
    T message )
```

Type Constraints***T* : class*****T* : ISimpleMessage****8.37.1.4 Subscribe< T >()**

```
SimpleMessageUnsubscribeToken ModIO.Util.SimpleMessageHub.Subscribe< T > (
    Action< T > subscription )
```

Type Constraints***T* : class*****T* : ISimpleMessage****8.37.2 Member Data Documentation****8.37.2.1 dictionary**

```
readonly Dictionary<Type, List<Action<ISimpleMessage> > > ModIO.Util.SimpleMessageHub.dictionary [private]
```

Initial value:

```
=
    new Dictionary<Type, List<Action<ISimpleMessage>>>()
```

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Utility/SimpleMessageHub.cs

8.38 ModIO.Util.SimpleMessageUnsubscribeToken Class Reference**Public Member Functions**

- **SimpleMessageUnsubscribeToken** (Action unsub)
- void **Unsubscribe** ()

Private Attributes

- Action **unsubAction**

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Utility/SimpleMessageUnsubscribeToken.cs

8.39 ModIO.Util.SimpleSingleton< T > Class Template Reference

Properties

- static T **Instance** [get, set]

Static Private Attributes

- static T **_instance**

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Utility/SimpleSingleton.cs

8.40 ModIO.SubscribedMod Struct Reference

Represents the [ModProfile](#) of a mod the current user has subscribed to. Contains the status and a directory (if installed) and the associated [ModProfile](#).

Public Attributes

- [SubscribedModStatus](#) **status**
- string **directory**
- [ModProfile](#) **modProfile**
- bool **enabled**

Whether the mod has been marked as enabled or disabled by the user

8.40.1 Detailed Description

Represents the [ModProfile](#) of a mod the current user has subscribed to. Contains the status and a directory (if installed) and the associated [ModProfile](#).

Note this is not necessarily an installed mod. You will need to check the status to see whether or not it is installed.

See also

status, [SubscribedModStatus](#), [ModProfile](#), [ModIOUnity.GetSubscribedMods](#)

8.40.2 Member Data Documentation

8.40.2.1 enabled

```
bool ModIO.SubscribedMod.enabled
```

Whether the mod has been marked as enabled or disabled by the user

See also

`ModIOUnity.EnableMod`, `ModIOUnity.DisableMod`

The documentation for this struct was generated from the following file:

- `Assets/Plugins/mod.io/Runtime/Structs/SubscribedMod.cs`

8.41 ModIO.Tag Struct Reference

Represents a [Tag](#) that can be assigned to a mod.

Public Attributes

- string **name**
- int **totalUses**

8.41.1 Detailed Description

Represents a [Tag](#) that can be assigned to a mod.

See also

[TagCategory](#), [ModIOUnity.GetTagCategories](#), [ModIOUnityAsync.GetTagCategories](#)

The documentation for this struct was generated from the following file:

- `Assets/Plugins/mod.io/Runtime/Structs/Tag.cs`

8.42 ModIO.TagCategory Struct Reference

Represents a particular category of tags.

Public Attributes

- string **name**
- [Tag\[\]](#) **tags**
- bool **multiSelect**
- bool **hidden**
- bool **locked**

8.42.1 Detailed Description

Represents a particular category of tags.

See also

[ModIOUnity.GetTagCategories](#), [ModIOUnityAsync.GetTagCategories](#), [Tag](#)

The documentation for this struct was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/TagCategory.cs

8.43 ModIO.TermsHash Struct Reference

This is the hash that identifies the TOS. Used to validate the TOS requirement when attempting to authenticate a user.

Public Attributes

- string **md5hash**

8.43.1 Detailed Description

This is the hash that identifies the TOS. Used to validate the TOS requirement when attempting to authenticate a user.

See also

[TermsOfUse](#), [ModIOUnity.GetTermsOfUse](#), [ModIOUnityAsync.GetTermsOfUse](#)

The documentation for this struct was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/TermsHash.cs

8.44 ModIO.TermsOfUse Struct Reference

TOS object received from a successful use of [ModIOUnity.GetTermsOfUse](#) This is used when attempting to authenticate via a third party. You must retrieve the TOS and input it along with an authentication request.

Public Attributes

- string **termsOfUse**
- [TermsOfUseLink\[\]](#) **links**
- [TermsHash](#) **hash**

8.44.1 Detailed Description

TOS object received from a successful use of [ModIOUnity.GetTermsOfUse](#) This is used when attempting to authenticate via a third party. You must retrieve the TOS and input it along with an authentication request.

See also

[ModIOUnity.GetTermsOfUse](#), [ModIOUnityAsync.GetTermsOfUse](#), [ModIOUnity.AuthenticateUserViaDiscord](#), [ModIOUnity.AuthenticateUserViaGoogle](#), [ModIOUnity.AuthenticateUserViaGOG](#), [ModIOUnity.AuthenticateUserViaItch](#), [ModIOUnity.AuthenticateUserViaOculus](#), [ModIOUnity.AuthenticateUserViaSteam](#), [ModIOUnity.AuthenticateUserViaSwitch](#), [ModIOUnity.AuthenticateUserViaXbox](#)

The documentation for this struct was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/TermsOfUse.cs

8.45 ModIO.TermsOfUseLink Struct Reference

Represents a url as part of the TOS. The 'required' field can be used to determine whether or not it is a TOS requirement to be displayed to the end user when viewing the TOS text.

Public Attributes

- string **name**
- string **url**
- bool **required**

8.45.1 Detailed Description

Represents a url as part of the TOS. The 'required' field can be used to determine whether or not it is a TOS requirement to be displayed to the end user when viewing the TOS text.

The documentation for this struct was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/TermsOfUseLink.cs

8.46 ModIO.UserInstalledMod Struct Reference

Struct used to represent a mod that already exists on the current device. You can view the subscribed users to this mod as well as the directory and modprofile associated to it.

Public Attributes

- bool **updatePending**
Whether or not the mod has been marked for an update
- string **directory**
the directory of where this mod is installed
- string **metadata**
The metadata for the version of the mod that is currently installed (Not to be mistaken with the metadata located inside of ModProfile.cs)
- string **version**
the version of this installed mod
- string **changeLog**
the change log for this version of the installed mod
- DateTime **dateAdded**
The date that this version of the mod was submitted to mod.io
- [ModProfile](#) **modProfile**
The profile of this mod, including the summary and name
- bool [enabled](#)
Whether the mod has been marked as enabled or disabled by the user

8.46.1 Detailed Description

Struct used to represent a mod that already exists on the current device. You can view the subscribed users to this mod as well as the directory and modprofile associated to it.

8.46.2 Member Data Documentation

8.46.2.1 [enabled](#)

```
bool ModIO.UserInstalledMod.enabled
```

Whether the mod has been marked as enabled or disabled by the user

See also

[ModIOUnity.EnableMod](#), [ModIOUnity.DisableMod](#)

The documentation for this struct was generated from the following file:

- `Assets/Plugins/mod.io/Runtime/Structs/UserInstalledMod.cs`

8.47 ModIO.UserProfile Struct Reference

Represents a particular mod.io user with their username, DownloadReferences for getting their avatar, as well as their language and timezone.

Public Attributes

- string **username**
The display name of the user's mod.io account
- long **userId**
This is the unique Id of the user.
- string **portal_username**
The display name of the user's account they authenticated with. Eg if they authenticated with Steam it would be their Steam username.
- [DownloadReference](#) **avatar_original**
- [DownloadReference](#) **avatar_50x50**
- [DownloadReference](#) **avatar_100x100**
- string **timezone**
- string **language**

8.47.1 Detailed Description

Represents a particular mod.io user with their username, DownloadReferences for getting their avatar, as well as their language and timezone.

The documentation for this struct was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Structs/UserProfile.cs

8.48 ModIO.Util.Utility Class Reference

Static Public Member Functions

- static string [GenerateHumanReadableNumber](#) (long number)
changes an int64 number into something more human readable such as "12.6K"
- static string **GenerateHumanReadableTimeStringFromSeconds** (int seconds)
- static string **GenerateHumanReadableStringForBytes** (long bytes)
- static string **GetModStatusAsString** ([ProgressHandle](#) handle)
- static string **GetModStatusAsString** ([SubscribedMod](#) mod)
- static string [EncodeEncryptedSteamAppTicket](#) (byte[] ticketData, uint ticketSize)
You can use this to convert your byte[] steam app ticket into a trimmed base64 encoded string to be used for the steam authentication.
- static List< T > [FindEverythingInScene](#)< T > ()
Finds everything in a loaded scene. Slow.

8.48.1 Member Function Documentation

8.48.1.1 EncodeEncryptedSteamAppTicket()

```
static string ModIO.Util.Utility.EncodeEncryptedSteamAppTicket (
    byte[] ticketData,
    uint ticketSize ) [static]
```

You can use this to convert your byte[] steam app ticket into a trimmed base64 encoded string to be used for the steam authentication.

Parameters

<i>ticketData</i>	the byte[] steam app ticket data
<i>ticketSize</i>	the desired length of the ticket to be trimmed to

See also

SetupSteamAuthenticationOption

Returns

base 64 encoded string from the provided steam app ticket

8.48.1.2 FindEverythingInScene< T >()

```
static List< T > ModIO.Util.Utility.FindEverythingInScene< T > ( ) [static]
```

Finds everything in a loaded scene. Slow.

Type Constraints

T* : *Component

8.48.1.3 GenerateHumanReadableNumber()

```
static string ModIO.Util.Utility.GenerateHumanReadableNumber (
    long number ) [static]
```

changes an int64 number into something more human readable such as "12.6K"

Parameters

<i>number</i>	the long to convert to readable string
---------------	--

Returns

The documentation for this class was generated from the following file:

- Assets/Plugins/mod.io/Runtime/Utility/Utility.cs

Index

- AddDependenciesToMod
 - ModIO.ModIOUnity, [42](#)
 - ModIO.ModIOUnityAsync, [83](#)
- AddModComment
 - ModIO.ModIOUnity, [43](#)
 - ModIO.ModIOUnityAsync, [84](#)
- AddSearchPhrase
 - ModIO.SearchFilter, [126](#)
- AddTag
 - ModIO.SearchFilter, [127](#)
- AddTags
 - ModIO.ModIOUnity, [44](#)
 - ModIO.ModIOUnityAsync, [84](#)
- AddUser
 - ModIO.SearchFilter, [127](#)
- ArchiveModProfile
 - ModIO.ModIOUnity, [44](#)
 - ModIO.ModIOUnityAsync, [86](#)
- AuthenticateUserViaDiscord
 - ModIO.ModIOUnity, [45](#)
 - ModIO.ModIOUnityAsync, [86](#)
- AuthenticateUserViaEpic
 - ModIO.ModIOUnity, [46](#)
 - ModIO.ModIOUnityAsync, [87](#)
- AuthenticateUserViaGOG
 - ModIO.ModIOUnity, [46](#)
 - ModIO.ModIOUnityAsync, [88](#)
- AuthenticateUserViaGoogle
 - ModIO.ModIOUnity, [47](#)
 - ModIO.ModIOUnityAsync, [89](#)
- AuthenticateUserViaItch
 - ModIO.ModIOUnity, [48](#)
 - ModIO.ModIOUnityAsync, [90](#)
- AuthenticateUserViaOculus
 - ModIO.ModIOUnity, [49](#)
 - ModIO.ModIOUnityAsync, [90](#)
- AuthenticateUserViaPlayStation
 - ModIO.ModIOUnity, [50](#)
 - ModIO.ModIOUnityAsync, [91](#)
- AuthenticateUserViaSteam
 - ModIO.ModIOUnity, [51](#)
 - ModIO.ModIOUnityAsync, [92](#)
- AuthenticateUserViaSwitch
 - ModIO.ModIOUnity, [52](#)
 - ModIO.ModIOUnityAsync, [93](#)
- AuthenticateUserViaXbox
 - ModIO.ModIOUnity, [53](#)
 - ModIO.ModIOUnityAsync, [94](#)
- Awake
 - ModIO.Util.MonoDispatcher, [119](#)
- BytesPerSecond
 - ModIO.ProgressHandle, [121](#)
- CommentDetails, [30](#)
- CommentObjects
 - ModIO.CommentPage, [31](#)
- communityOptions
 - ModIO.ModProfileDetails, [115](#)
- Completed
 - ModIO.ProgressHandle, [121](#)
- contentWarning
 - ModIO.ModProfileDetails, [115](#)
- CreateModProfile
 - ModIO.ModIOUnity, [54](#)
 - ModIO.ModIOUnityAsync, [94](#)
- DeleteModComment
 - ModIO.ModIOUnity, [55](#)
 - ModIO.ModIOUnityAsync, [95](#)
- DeleteTags
 - ModIO.ModIOUnity, [55](#)
 - ModIO.ModIOUnityAsync, [96](#)
- description
 - ModIO.ModProfileDetails, [115](#)
- dictionary
 - ModIO.Util.SimpleMessageHub, [133](#)
- DisableModManagement
 - ModIO.ModIOUnity, [56](#)
- downloadedBytes
 - ModIO.ModIoWebRequest, [110](#)
- DownloadImage
 - ModIO.ModIOUnity, [56](#)
 - ModIO.ModIOUnityAsync, [96](#)
- downloadProgress
 - ModIO.ModIoWebRequest, [110](#)
- EditModProfile
 - ModIO.ModIOUnity, [57](#)
 - ModIO.ModIOUnityAsync, [97](#)
- enabled
 - ModIO.InstalledMod, [35](#)
 - ModIO.SubscribedMod, [135](#)
 - ModIO.UserInstalledMod, [138](#)
- EnableModManagement
 - ModIO.ModIOUnity, [57](#)
- EncodeEncryptedSteamAppTicket
 - ModIO.Util.Utility, [139](#)
- FetchUpdates

- ModIO.ModIOUnity, 58
- ModIO.ModIOUnityAsync, 97
- FindEverythingInScene< T >
 - ModIO.Util.Utility, 140
- ForceUninstallMod
 - ModIO.ModIOUnity, 59
- GenerateCreationToken
 - ModIO.ModIOUnity, 59
- GenerateHumanReadableNumber
 - ModIO.Util.Utility, 140
- GetCurrentModManagementOperation
 - ModIO.ModIOUnity, 60
- GetCurrentUploadHandle
 - ModIO.ModIOUnity, 60
- GetCurrentUser
 - ModIO.ModIOUnity, 60
 - ModIO.ModIOUnityAsync, 98
- GetCurrentUserRatingFor
 - ModIO.ModIOUnity, 61
 - ModIO.ModIOUnityAsync, 98
- GetCurrentUserRatings
 - ModIO.ModIOUnity, 62
 - ModIO.ModIOUnityAsync, 99
- GetInstalledModsForUser
 - ModIO.ModIOUnity, 62
- GetMod
 - ModIO.ModIOUnity, 63
 - ModIO.ModIOUnityAsync, 99
- GetModComments
 - ModIO.ModIOUnity, 63
 - ModIO.ModIOUnityAsync, 100
- GetModDependencies
 - ModIO.ModIOUnity, 64
 - ModIO.ModIOUnityAsync, 100
- GetMods
 - ModIO.ModIOUnity, 64
 - ModIO.ModIOUnityAsync, 100
- GetResponseHeader
 - ModIO.ModIoWebRequest, 110
- GetSubscribedMods
 - ModIO.ModIOUnity, 65
- GetSystemInstalledMods
 - ModIO.ModIOUnity, 66
- GetTagCategories
 - ModIO.ModIOUnity, 66
 - ModIO.ModIOUnityAsync, 101
- GetTermsOfUse
 - ModIO.ModIOUnity, 67
 - ModIO.ModIOUnityAsync, 101
- homepage_url
 - ModIO.ModProfileDetails, 115
- images
 - ModIO.ModProfileDetails, 115
- InitializeForUser
 - ModIO.ModIOUnity, 68
- IsAuthenticated
 - ModIO.ModIOUnity, 69
 - ModIO.ModIOUnityAsync, 102
- isDone
 - ModIO.ModIoWebRequest, 110
- IsInitialized
 - ModIO.ModIOUnity, 70
- IsModManagementBusy
 - ModIO.ModIOUnity, 70
- IsNetworkError
 - ModIO.Result, 124
- IsSearchFilterValid
 - ModIO.SearchFilter, 127
- IsValid
 - ModIO.DownloadReference, 33
- logo
 - ModIO.ModProfileDetails, 115
- LogOutCurrentUser
 - ModIO.ModIOUnity, 70
- maxSubscribers
 - ModIO.ModProfileDetails, 116
- metadata
 - ModIO.ModfileDetails, 38
 - ModIO.ModProfile, 113
 - ModIO.ModProfileDetails, 116
- modId
 - ModIO.ModProfileDetails, 116
- ModIO, 23
 - SortModsBy, 25
 - SubscribedModStatus, 25
 - UserPortal, 25
- ModIO.AuthenticationServiceProviderExtensions, 29
- ModIO.BuildSettings, 29
- ModIO.CommentPage, 30
 - CommentObjects, 31
 - totalSearchResultsFound, 31
- ModIO.CreationToken, 32
- ModIO.DownloadReference, 32
 - IsValid, 33
- ModIO.IModIoWebRequest, 33
- ModIO.Implementation, 26
- ModIO.Implementation.API, 26
- ModIO.Implementation.API.Objects, 26
- ModIO.Implementation.API.Objects.ModDependencies, 37
- ModIO.Implementation.API.Objects.ModDependenciesObject, 37
- ModIO.Implementation.API.Objects.Rating, 122
- ModIO.Implementation.API.Objects.RatingObject, 122
- ModIO.Implementation.InstalledModExtensions, 35
- ModIO.InstalledMod, 34
 - enabled, 35
- ModIO.ModComment, 36
 - threadPosition, 36
- ModIO.ModfileDetails, 38
 - metadata, 38
- ModIO.ModId, 38
- ModIO.ModIOUnity, 39

- AddDependenciesToMod, [42](#)
- AddModComment, [43](#)
- AddTags, [44](#)
- ArchiveModProfile, [44](#)
- AuthenticateUserViaDiscord, [45](#)
- AuthenticateUserViaEpic, [46](#)
- AuthenticateUserViaGOG, [46](#)
- AuthenticateUserViaGoogle, [47](#)
- AuthenticateUserViaItch, [48](#)
- AuthenticateUserViaOculus, [49](#)
- AuthenticateUserViaPlayStation, [50](#)
- AuthenticateUserViaSteam, [51](#)
- AuthenticateUserViaSwitch, [52](#)
- AuthenticateUserViaXbox, [53](#)
- CreateModProfile, [54](#)
- DeleteModComment, [55](#)
- DeleteTags, [55](#)
- DisableModManagement, [56](#)
- DownloadImage, [56](#)
- EditModProfile, [57](#)
- EnableModManagement, [57](#)
- FetchUpdates, [58](#)
- ForceUninstallMod, [59](#)
- GenerateCreationToken, [59](#)
- GetCurrentModManagementOperation, [60](#)
- GetCurrentUploadHandle, [60](#)
- GetCurrentUser, [60](#)
- GetCurrentUserRatingFor, [61](#)
- GetCurrentUserRatings, [62](#)
- GetInstalledModsForUser, [62](#)
- GetMod, [63](#)
- GetModComments, [63](#)
- GetModDependencies, [64](#)
- GetMods, [64](#)
- GetSubscribedMods, [65](#)
- GetSystemInstalledMods, [66](#)
- GetTagCategories, [66](#)
- GetTermsOfUse, [67](#)
- InitializeForUser, [68](#)
- IsAuthenticated, [69](#)
- IsInitialized, [70](#)
- IsModManagementBusy, [70](#)
- LogOutCurrentUser, [70](#)
- MuteUser, [71](#)
- RateMod, [71](#)
- RemoveDependenciesFromMod, [72](#)
- Report, [73](#)
- RequestAuthenticationEmail, [73](#)
- SetLoggingDelegate, [74](#)
- Shutdown, [75](#)
- SubmitEmailSecurityCode, [75](#)
- SubscribeToMod, [76](#)
- UnmuteUser, [76](#)
- UnsubscribeFromMod, [77](#)
- UpdateModComment, [77](#)
- UploadModfile, [79](#)
- UploadModMedia, [80](#)
- ModIO.ModIOUnityAsync, [80](#)
- AddDependenciesToMod, [83](#)
- AddModComment, [84](#)
- AddTags, [84](#)
- ArchiveModProfile, [86](#)
- AuthenticateUserViaDiscord, [86](#)
- AuthenticateUserViaEpic, [87](#)
- AuthenticateUserViaGOG, [88](#)
- AuthenticateUserViaGoogle, [89](#)
- AuthenticateUserViaItch, [90](#)
- AuthenticateUserViaOculus, [90](#)
- AuthenticateUserViaPlayStation, [91](#)
- AuthenticateUserViaSteam, [92](#)
- AuthenticateUserViaSwitch, [93](#)
- AuthenticateUserViaXbox, [94](#)
- CreateModProfile, [94](#)
- DeleteModComment, [95](#)
- DeleteTags, [96](#)
- DownloadImage, [96](#)
- EditModProfile, [97](#)
- FetchUpdates, [97](#)
- GetCurrentUser, [98](#)
- GetCurrentUserRatingFor, [98](#)
- GetCurrentUserRatings, [99](#)
- GetMod, [99](#)
- GetModComments, [100](#)
- GetModDependencies, [100](#)
- GetMods, [100](#)
- GetTagCategories, [101](#)
- GetTermsOfUse, [101](#)
- IsAuthenticated, [102](#)
- MuteUser, [102](#)
- RateMod, [103](#)
- RemoveDependenciesFromMod, [103](#)
- Report, [104](#)
- RequestAuthenticationEmail, [104](#)
- Shutdown, [105](#)
- SubmitEmailSecurityCode, [105](#)
- SubscribeToMod, [106](#)
- UnmuteUser, [106](#)
- UnsubscribeFromMod, [107](#)
- UpdateModComment, [107](#)
- UploadModfile, [108](#)
- UploadModMedia, [108](#)
- ModIO.ModIOWebRequest, [109](#)
 - downloadedBytes, [110](#)
 - downloadProgress, [110](#)
 - GetResponseHeader, [110](#)
 - isDone, [110](#)
 - uploadedBytes, [110](#)
 - uploadProgress, [111](#)
- ModIO.ModPage, [111](#)
 - modProfiles, [111](#)
 - totalSearchResultsFound, [112](#)
- ModIO.ModProfile, [112](#)
 - metadata, [113](#)
- ModIO.ModProfileDetails, [114](#)
 - communityOptions, [115](#)
 - contentWarning, [115](#)

- description, 115
- homepage_url, 115
- images, 115
- logo, 115
- maxSubscribers, 116
- metadata, 116
- modId, 116
- name, 116
- name_id, 117
- summary, 117
- tags, 117
- visible, 117
- ModIO.ModStats, 118
- ModIO.ProgressHandle, 121
 - BytesPerSecond, 121
 - Completed, 121
- ModIO.Report, 123
 - Report, 123
- ModIO.Result, 124
 - IsNetworkError, 124
- ModIO.ResultAnd< T >, 125
- ModIO.SearchFilter, 125
 - AddSearchPhrase, 126
 - AddTag, 127
 - AddUser, 127
 - IsSearchFilterValid, 127
 - SetPageIndex, 128
 - SetPageSize, 128
 - SetToAscending, 129
 - SortBy, 129
- ModIO.ServerSettings, 131
- ModIO.SubscribedMod, 134
 - enabled, 135
- ModIO.Tag, 135
- ModIO.TagCategory, 135
- ModIO.TermsHash, 136
- ModIO.TermsOfUse, 136
- ModIO.TermsOfUseLink, 137
- ModIO.UserInstalledMod, 137
 - enabled, 138
- ModIO.UserProfile, 138
- ModIO.Util, 26
- ModIO.Util.ISimpleMessage, 35
- ModIO.Util.ISimpleMonoSingleton, 35
- ModIO.Util.MonoDispatcher, 118
 - Awake, 119
- ModIO.Util.MonoSingleton< T >, 119
 - SetupSingleton, 120
- ModIO.Util.Mutex, 120
- ModIO.Util.SelfInstantiatingMonoSingleton< T >, 129
 - SetupSingleton, 130
- ModIO.Util.SimpleMessageHub, 131
 - dictionary, 133
 - OnDestroy, 132
 - Publish< T >, 132
 - PublishThreadSafe< T >, 132
 - Subscribe< T >, 133
- ModIO.Util.SimpleMessageUnsubscribeToken, 133
- ModIO.Util.SimpleSingleton< T >, 134
- ModIO.Util.Utility, 139
 - EncodeEncryptedSteamAppTicket, 139
 - FindEverythingInScene< T >, 140
 - GenerateHumanReadableNumber, 140
- modProfiles
 - ModIO.ModPage, 111
- MuteUser
 - ModIO.ModIOUnity, 71
 - ModIO.ModIOUnityAsync, 102
- name
 - ModIO.ModProfileDetails, 116
- name_id
 - ModIO.ModProfileDetails, 117
- OnDestroy
 - ModIO.Util.SimpleMessageHub, 132
- Plugins, 27
- Plugins.mod, 27
- Plugins.mod.io, 27
- Plugins.mod.io.Runtime, 27
- Plugins.mod.io.Runtime.Utility, 27
- Plugins.mod.io.Runtime.Utility.CoroutineRunner, 31
- Publish< T >
 - ModIO.Util.SimpleMessageHub, 132
- PublishThreadSafe< T >
 - ModIO.Util.SimpleMessageHub, 132
- RateMod
 - ModIO.ModIOUnity, 71
 - ModIO.ModIOUnityAsync, 103
- RemoveDependenciesFromMod
 - ModIO.ModIOUnity, 72
 - ModIO.ModIOUnityAsync, 103
- Report
 - ModIO.ModIOUnity, 73
 - ModIO.ModIOUnityAsync, 104
 - ModIO.Report, 123
- RequestAuthenticationEmail
 - ModIO.ModIOUnity, 73
 - ModIO.ModIOUnityAsync, 104
- SetLoggingDelegate
 - ModIO.ModIOUnity, 74
- SetPageIndex
 - ModIO.SearchFilter, 128
- SetPageSize
 - ModIO.SearchFilter, 128
- SetToAscending
 - ModIO.SearchFilter, 129
- SetupSingleton
 - ModIO.Util.MonoSingleton< T >, 120
 - ModIO.Util.SelfInstantiatingMonoSingleton< T >, 130
- Shutdown
 - ModIO.ModIOUnity, 75
 - ModIO.ModIOUnityAsync, 105

- SortBy
 - ModIO.SearchFilter, [129](#)
- SortModsBy
 - ModIO, [25](#)
- SubmitEmailSecurityCode
 - ModIO.ModIOUnity, [75](#)
 - ModIO.ModIOUnityAsync, [105](#)
- Subscribe< T >
 - ModIO.Util.SimpleMessageHub, [133](#)
- SubscribedModStatus
 - ModIO, [25](#)
- SubscribeToMod
 - ModIO.ModIOUnity, [76](#)
 - ModIO.ModIOUnityAsync, [106](#)
- summary
 - ModIO.ModProfileDetails, [117](#)
- tags
 - ModIO.ModProfileDetails, [117](#)
- threadPosition
 - ModIO.ModComment, [36](#)
- totalSearchResultsFound
 - ModIO.CommentPage, [31](#)
 - ModIO.ModPage, [112](#)
- UnmuteUser
 - ModIO.ModIOUnity, [76](#)
 - ModIO.ModIOUnityAsync, [106](#)
- UnsubscribeFromMod
 - ModIO.ModIOUnity, [77](#)
 - ModIO.ModIOUnityAsync, [107](#)
- UpdateModComment
 - ModIO.ModIOUnity, [77](#)
 - ModIO.ModIOUnityAsync, [107](#)
- uploadedBytes
 - ModIO.ModIoWebRequest, [110](#)
- UploadModfile
 - ModIO.ModIOUnity, [79](#)
 - ModIO.ModIOUnityAsync, [108](#)
- UploadModMedia
 - ModIO.ModIOUnity, [80](#)
 - ModIO.ModIOUnityAsync, [108](#)
- uploadProgress
 - ModIO.ModIoWebRequest, [111](#)
- UserPortal
 - ModIO, [25](#)
- visible
 - ModIO.ModProfileDetails, [117](#)