**Практическая работа №7**

Пример на языке C#: отправка приложений, надстроек и тестируемых возможностей

В этой практической работе представлены примеры кода на C# по использованию [API отправки в Microsoft Store](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/create-and-manage-submissions-using-windows-store-services) для решения этих задач.

* [Создание отправки приложения](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/csharp-code-examples-for-the-windows-store-submission-api#create-app-submission)
* [Создание отправки надстройки](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/csharp-code-examples-for-the-windows-store-submission-api#create-add-on-submission)
* [Обновление отправки надстройки](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/csharp-code-examples-for-the-windows-store-submission-api#update-add-on-submission)
* [Создание отправки тестового пакета](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/csharp-code-examples-for-the-windows-store-submission-api#create-flight-submission)

Вы можете ознакомиться с каждым примером, чтобы подробнее узнать о демонстрируемой в нем задаче, либо вы можете собрать все примеры кода в этой практической работе в консольное приложение. Для сборки примеров создайте консольное приложение C# с именем **DeveloperApiCSharpSample** в Visual Studio, скопируйте каждый пример в отдельный файл с кодом в проекте и соберите проект.

**Предварительные требования**

В этих примерах используются следующие библиотеки:

* Microsoft.WindowsAzure.Storage.dll. Эта библиотека доступна в [Пакете SDK Azure для .NET](https://azure.microsoft.com/downloads/). Ее также можно получить путем установки [пакета NuGet WindowsAzure.Storage](https://www.nuget.org/packages/WindowsAzure.Storage).
* Пакет NuGet [Newtonsoft.Json](https://www.newtonsoft.com/json) от Newtonsoft.

**Основная программа**

В следующем примере реализуется программа командной строки, вызывающая другие методы из примеров в этой пр для демонстрации различных вариантов использования API отправки в Microsoft Store. Адаптация программы для собственного использования.

* Назначьте свойства ApplicationId, InAppProductId и FlightId идентификатору приложения, надстройке и тестовому пакету, которыми вы хотите управлять.
* Назначьте свойства ClientId и ClientSecret идентификатору клиента и ключу своего приложения и замените строку ClientId в URL-адресе TokenEndpoint идентификатором владельца для своего приложения.

namespace DeveloperApiCSharpSample

{

class Program

{

static void Main(string[] args)

{

var config = new ClientConfiguration()

{

ApplicationId = "...",

InAppProductId = "...",

FlightId = "...",

ClientId = "...",

ClientSecret = "...",

ServiceUrl = "https://manage.devcenter.microsoft.com",

TokenEndpoint = "https://login.microsoftonline.com/<tenantid>/oauth2/token",

Scope = "https://manage.devcenter.microsoft.com",

};

new FlightSubmissionUpdateSample(config).RunFlightSubmissionUpdateSample();

new InAppProductSubmissionUpdateSample(config).RunInAppProductSubmissionUpdateSample();

new InAppProductSubmissionCreateSample(config).RunInAppProductSubmissionCreateSample();

new AppSubmissionUpdateSample(config).RunAppSubmissionUpdateSample();

}

}

}

**Вспомогательный класс ClientConfiguration**

Образец приложения использует вспомогательный класс ClientConfiguration для передачи данных Azure Active Directory и данных приложения остальным методам из примеров, в которых используется API отправки в Microsoft Store.

namespace DeveloperApiCSharpSample

{

/// <summary>

/// Configuration class

/// </summary>

public class ClientConfiguration

{

/// <summary>

/// Client Id of your AAD app.

/// Example" ba3c223b-03ab-4a44-aa32-38aa10c27e32

/// </summary>

public string ClientId { get; set; }

/// <summary>

/// Client secret of your AAD app

/// </summary>

public string ClientSecret { get; set; }

/// <summary>

/// Service root endpoint.

/// Example: https://manage.devcenter.microsoft.com

/// </summary>

public string ServiceUrl { get; set; }

/// <summary>

/// Token endpoint to which the request is to be made. Specific to your AAD app

/// Example: https://login.windows.net/d454d300-128e-2d81-334a-27d9b2baf002/oauth2/token

/// </summary>

public string TokenEndpoint { get; set; }

/// <summary>

/// Resource scope. If not provided (set to null), default one is used for the production API

/// endpoint ("https://manage.devcenter.microsoft.com")

/// </summary>

public string Scope { get; set; }

/// <summary>

/// Application ID.

/// Example: 9WZANCRD4AMD

/// </summary>

public string ApplicationId { get; set; }

/// <summary>

/// In-app-product ID;

/// Example: 9WZBMAAD4VVV

/// </summary>

public string InAppProductId { get; set; }

/// <summary>

/// Flight Id

/// Example: 62211033-c2fa-3934-9b03-d72a6b2a171d

/// </summary>

public string FlightId { get; set; }

}

}

**Создание отправки приложения**

В следующем примере реализован класс, который использует несколько методов из API отправки в Microsoft Store для обновления отправки приложения. RunAppSubmissionUpdateSampleМетод в классе создает новую передачу в виде клона последней опубликованной отправки, а затем обновляет и фиксирует клонированную отправку в центр партнеров. В частности, метод RunAppSubmissionUpdateSample выполняет следующие задачи:

1. Сначала метод [получает данные для указанного приложения](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/get-an-app).
2. Затем он [удаляет ожидающую отправку для приложения](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/delete-an-app-submission), если она существует.
3. После этого [выполняется создание новой отправки для приложения](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/create-an-app-submission) (новая отправка — это копия последней опубликованной отправки).
4. Код изменяет некоторые сведения о новой отправке и отправляет новый пакет отправки в хранилище BLOB-объектов Azure.
5. Затем он [обновляет](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/update-an-app-submission) и [фиксирует](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/commit-an-app-submission) новую отправку в Центре партнеров.
6. Наконец, он периодически [проверяет состояние новой отправки](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/get-status-for-an-app-submission), пока она не будет успешно зафиксирована.

namespace DeveloperApiCSharpSample

{

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Net.Http;

using System.Threading.Tasks;

using Newtonsoft.Json.Linq;

/// <summary>

/// This sample update does a full submission update, updating listings info, images, and packages

/// </summary>

public class AppSubmissionUpdateSample

{

private ClientConfiguration ClientConfig;

/// <summary>

/// Constructor

/// </summary>

/// <param name="c">An instance of ClientConfiguration that contains all parameters populated</param>

public AppSubmissionUpdateSample(ClientConfiguration c)

{

this.ClientConfig = c;

}

public void RunAppSubmissionUpdateSample()

{

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// SETTINGS

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

var appId = this.ClientConfig.ApplicationId;

var clientId = this.ClientConfig.ClientId;

var clientSecret = this.ClientConfig.ClientSecret;

var serviceEndpoint = this.ClientConfig.ServiceUrl;

var tokenEndpoint = this.ClientConfig.TokenEndpoint;

// Get authorization token.

Console.WriteLine("Getting authorization token ");

var accessToken = IngestionClient.GetClientCredentialAccessToken(

tokenEndpoint,

clientId,

clientSecret).Result;

Console.WriteLine("Getting application ");

var client = new IngestionClient(accessToken, serviceEndpoint);

dynamic app = client.Invoke<dynamic>(

HttpMethod.Get,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.GetApplicationUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

appId),

requestContent: null).Result;

Console.WriteLine(app.ToString());

// Let's get the last published submission, and print its contents, just for information.

if (app.lastPublishedApplicationSubmission == null)

{

// It is not possible to create the very first submission through the API.

throw new InvalidOperationException(

"You need at least one published submission to create new submissions through API.");

}

// Let's see if there is a pending submission. Warning! If it was created through the API,

// it will be deleted so that we could create a new one in its stead.

if (app.pendingApplicationSubmission != null)

{

var submissionId = app.pendingApplicationSubmission.id.Value as string;

// Try deleting it. If it was NOT created via the API, then you need to manually

// delete it from the dashboard. This is done as a safety measure to make sure that a

// user and an automated system don't make conflicting edits.

Console.WriteLine("Deleting the pending submission");

client.Invoke<dynamic>(

HttpMethod.Delete,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.GetSubmissionUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

appId,

submissionId),

requestContent: null).Wait();

}

// Create a new submission, which will be an exact copy of the last published submission.

Console.WriteLine("Creating a new submission");

dynamic clonedSubmission = client.Invoke<dynamic>(

HttpMethod.Post,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.CreateSubmissionUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

appId),

requestContent: null).Result;

// Update some property on the root submission object.

clonedSubmission.notesForCertification = "This is a test update, updating listing info, images, and packages";

// Now, assume we have an en-us listing. Let's try to change its description.

clonedSubmission.listings["en-us"].baseListing.description = "This is my new en-Us description!";

// Update images.

// Assuming we have at least 1 image, let's delete one image.

clonedSubmission.listings["en-us"].baseListing.images[0].fileStatus = "PendingDelete";

var images = new List<dynamic>();

images.Add(clonedSubmission.listings["en-us"].baseListing.images[0]);

images.Add(

new

{

fileStatus = "PendingUpload",

fileName = "rectangles.png",

imageType = "Screenshot",

description = "This is a new image uploaded through the API!",

});

clonedSubmission.listings["en-us"].baseListing.images = JToken.FromObject(images.ToArray());

// Update packages.

// Let's say we want to delete the existing package.

clonedSubmission.applicationPackages[0].fileStatus = "PendingDelete";

// Now, let's add a new package.

var packages = new List<dynamic>();

packages.Add(clonedSubmission.applicationPackages[0]);

packages.Add(

new

{

fileStatus = "PendingUpload",

fileName = "package.appx",

minimumDirectXVersion = "None",

minimumSystemRam = "None"

});

clonedSubmission.applicationPackages = JToken.FromObject(packages.ToArray());

var clonedSubmissionId = clonedSubmission.id.Value as string;

// Uploaded the zip archive with all new files to the SAS url returned with the submission.

var fileUploadUrl = clonedSubmission.fileUploadUrl.Value as string;

Console.WriteLine("FileUploadUrl: " + fileUploadUrl);

Console.WriteLine("Uploading file");

IngestionClient.UploadFileToBlob(@"..\..\files.zip", fileUploadUrl).Wait();

// Update the submission.

Console.WriteLine("Updating the submission");

client.Invoke<dynamic>(

HttpMethod.Put,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.UpdateUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

appId,

clonedSubmissionId),

requestContent: clonedSubmission).Wait();

// Tell the system that we are done updating the submission.

// Update the submission.

Console.WriteLine("Committing the submission");

client.Invoke<dynamic>(

HttpMethod.Post,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.CommitSubmissionUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

appId,

clonedSubmissionId),

requestContent: null).Wait();

// Let's periodically check the status until it changes from "CommitsStarted" to either

// successful status or a failure.

Console.WriteLine("Waiting for the submission commit processing to complete. This may take a couple of minutes.");

string submissionStatus = null;

do

{

Task.Delay(TimeSpan.FromSeconds(5)).Wait();

dynamic statusResource = client.Invoke<dynamic>(

HttpMethod.Get,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.ApplicationSubmissionStatusUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

appId,

clonedSubmissionId),

requestContent: null).Result;

submissionStatus = statusResource.status.Value as string;

Console.WriteLine("Current status: " + submissionStatus);

}

while ("CommitStarted".Equals(submissionStatus));

if ("CommitFailed".Equals(submissionStatus))

{

Console.WriteLine("Submission has failed. Please checkt the Errors collection of the submissionResource response.");

return;

}

else

{

Console.WriteLine("Submission commit success! Here are some data:");

dynamic submission = client.Invoke<dynamic>(

HttpMethod.Get,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.GetSubmissionUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

appId,

clonedSubmissionId),

requestContent: null).Result;

Console.WriteLine("Packages: " + submission.applicationPackages);

Console.WriteLine("en-US description: " + submission.listings["en-us"].baseListing.description);

Console.WriteLine("Images: " + submission.listings["en-us"].baseListing.images);

}

}

}

}

**Создание отправки надстройки**

В следующем примере реализован класс, который использует несколько методов из API отправки в Microsoft Store для создания новой отправки надстройки. Метод RunInAppProductSubmissionCreateSample в классе выполняет следующие задачи:

1. Сначала метод [создает новую надстройку](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/create-an-add-on).
2. Затем он [создает новую отправку для надстройки](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/create-an-add-on-submission).
3. Код передает ZIP-архив, содержащий значки для отправки, в хранилище BLOB-объектов Azure.
4. Затем она [фиксирует новую отправку в центр партнеров](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/commit-an-add-on-submission).
5. Наконец, он периодически [проверяет состояние новой отправки](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/get-status-for-an-add-on-submission), пока она не будет успешно зафиксирована.

namespace DeveloperApiCSharpSample

{

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Net.Http;

using System.Threading.Tasks;

/// <summary>

/// Sample code for how to create add-ons, and how to create and update add-on submissions.

/// </summary>

public class InAppProductSubmissionCreateSample

{

private ClientConfiguration ClientConfig;

/// <summary>

/// Constructor

/// </summary>

/// <param name="c">An instance of ClientConfiguration that contains all parameters populated</param>

public InAppProductSubmissionCreateSample(ClientConfiguration c)

{

this.ClientConfig = c;

}

public void RunInAppProductSubmissionCreateSample()

{

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// SETTINGS

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

var appId = this.ClientConfig.ApplicationId;

var clientId = this.ClientConfig.ClientId;

var clientSecret = this.ClientConfig.ClientSecret;

var serviceEndpoint = this.ClientConfig.ServiceUrl;

var tokenEndpoint = this.ClientConfig.TokenEndpoint;

// Get authorization token

Console.WriteLine("Getting authorization token ");

var accessToken = IngestionClient.GetClientCredentialAccessToken(

tokenEndpoint,

clientId,

clientSecret).Result;

Console.WriteLine("Creating a new add-on");

dynamic newIap = new

{

applicationIds = new List<string>() { appId },

productType = "Durable",

productId = "Sample-" + Guid.NewGuid().ToString(),

};

var client = new IngestionClient(accessToken, serviceEndpoint);

dynamic iapCreated = client.Invoke<dynamic>(

HttpMethod.Post,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.CreateInAppUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant),

requestContent: newIap).Result;

Console.WriteLine(iapCreated.ToString());

var iapId = iapCreated.id.Value as string;

// Create a new submission, which will be an exact copy of the last published submission

Console.WriteLine("Creating a new submission");

dynamic newSubmission = new

{

contentType = "BookDownload",

keywords = new List<string> { "book", "download" },

lifeTime = "ThreeDays",

targetPublishMode = "Immediate",

visibility = "Public",

pricing = new

{

priceId = "Free",

},

listings = new Dictionary<string, dynamic>()

{

{

"en-us",

new

{

description = "Sample IAP description",

title = "Sample IAP title",

icon = new

{

FileName = "icon300x300.png",

FileStatus = "PendingUpload",

},

}

}

}

};

// Because it's a new add-on, we are going to create a new submission instead of

// modifying the last published one. If you had a published add-on, you could

// pass "null" as request body to clone the latest published submission and then

// perform a PUT call. Alternatively, you can always post the new submission entirely

// even if you already have a published submission but you'll have to upload the image each time.

dynamic createdSubmission = client.Invoke<dynamic>(

HttpMethod.Post,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.InAppSubmissionUrl,

IngestionClient.Version,

IngestionClient.Tenant,

iapId),

requestContent: newSubmission).Result;

Console.WriteLine(createdSubmission);

var submissionId = createdSubmission.id.Value as string;

// Upload the zip archive with all new files to the SAS URL returned with the submission.

var fileUploadUrl = createdSubmission.fileUploadUrl.Value as string;

Console.WriteLine("FileUploadUrl: " + fileUploadUrl);

Console.WriteLine("Uploading file");

IngestionClient.UploadFileToBlob(@"..\..\files.zip", fileUploadUrl).Wait();

// Tell the system that we are done updating the submission.

// Update the submission

Console.WriteLine("Committing the submission");

client.Invoke<dynamic>(

HttpMethod.Post,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.InAppProductCommitSubmissionUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

iapId,

submissionId),

requestContent: null).Wait();

// Periodically check the status until it changes from "CommitsStarted" to either

// successful status or a failure.

Console.WriteLine("Waiting for the submission commit processing to complete. This may take a couple of minutes.");

string submissionStatus = null;

do

{

Task.Delay(TimeSpan.FromSeconds(5)).Wait();

dynamic statusResource = client.Invoke<dynamic>(

HttpMethod.Get,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.InAppSubmissionStatusUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

iapId,

submissionId),

requestContent: null).Result;

submissionStatus = statusResource.status.Value as string;

Console.WriteLine("Current status: " + submissionStatus);

}

while ("CommitStarted".Equals(submissionStatus));

if ("CommitFailed".Equals(submissionStatus))

{

Console.WriteLine("Submission has failed. Please check the Errors collection of the submissionResource response.");

return;

}

else

{

Console.WriteLine("Submission commit success!");

}

}

}

}

**Обновление отправки надстройки**

В следующем примере реализован класс, который использует несколько методов в API отправки в Microsoft Store для обновления имеющейся отправки надстройки. RunInAppProductSubmissionUpdateSampleМетод в классе создает новую передачу в виде клона последней опубликованной отправки, а затем обновляет и фиксирует клонированную отправку в центр партнеров. В частности, метод RunInAppProductSubmissionUpdateSample выполняет следующие задачи:

1. Сначала метод [получает данные для указанной надстройки](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/get-an-add-on).
2. Затем он [удаляет ожидающую отправку для надстройки](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/delete-an-add-on-submission), если она существует.
3. После этого [выполняется создание новой отправки для надстройки](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/create-an-add-on-submission) (новая отправка — это копия последней опубликованной отправки).
4. Затем он [обновляет](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/update-an-add-on-submission) и [фиксирует](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/commit-an-add-on-submission) новую отправку в Центре партнеров.
5. Наконец, он периодически [проверяет состояние новой отправки](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/get-status-for-an-add-on-submission), пока она не будет успешно зафиксирована.

namespace DeveloperApiCSharpSample

{

using System;

using System.Globalization;

using System.Net.Http;

using System.Threading.Tasks;

/// <summary>

/// Sample code for how to update add-on submissions

/// </summary>

public class InAppProductSubmissionUpdateSample

{

private ClientConfiguration ClientConfig;

/// <summary>

/// Constructor

/// </summary>

/// <param name="c">An instance of ClientConfiguration that contains all parameters populated</param>

public InAppProductSubmissionUpdateSample(ClientConfiguration c)

{

this.ClientConfig = c;

}

public void RunInAppProductSubmissionUpdateSample()

{

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// SETTINGS

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

var iapId = this.ClientConfig.InAppProductId;

var clientId = this.ClientConfig.ClientId;

var clientSecret = this.ClientConfig.ClientSecret;

var serviceEndpoint = this.ClientConfig.ServiceUrl;

var tokenEndpoint = this.ClientConfig.TokenEndpoint;

// Get authorization token

Console.WriteLine("Getting authorization token ");

var accessToken = IngestionClient.GetClientCredentialAccessToken(

tokenEndpoint,

clientId,

clientSecret).Result;

Console.WriteLine("Getting the add-on");

var client = new IngestionClient(accessToken, serviceEndpoint);

dynamic iap = client.Invoke<dynamic>(

HttpMethod.Get,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.GetInAppUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

iapId),

requestContent: null).Result;

Console.WriteLine(iap.ToString());

// Let's see if there is a pending submission. Warning! If it was created through the API,

// it will be deleted so that we could create a new one in its stead.

if (iap.pendingInAppProductSubmission != null)

{

var submissionId = iap.pendingInAppProductSubmission.id.Value as string;

// Let's try deleting it. If it was NOT created via the API, then you need to manually

// delete it from the dashboard. This is a safety measure to make sure that a human user and

// an automated system don't make conflicting edits.

Console.WriteLine("Deleting the pending submission");

client.Invoke<dynamic>(

HttpMethod.Delete,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.InAppSubmissionUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

iapId,

submissionId),

requestContent: null).Wait();

}

// Create a new submission, which will be an exact copy of the last published submission.

Console.WriteLine("Creating a new submission");

dynamic clonedSubmission = client.Invoke<dynamic>(

HttpMethod.Post,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.InAppSubmissionUrl,

IngestionClient.Version,

IngestionClient.Tenant,

iapId),

requestContent: null).Result;

var clonedSubmissionId = clonedSubmission.id.Value as string;

Console.WriteLine(clonedSubmission.ToString());

// Update the add-on price and keep the rest unchanged.

clonedSubmission.pricing.priceId = "Tier2"; // $0.99

// Because we are not uploading any new images, we don't need to upload the zip file.

// Update the submission.

Console.WriteLine("Updating the submission");

client.Invoke<dynamic>(

HttpMethod.Put,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.InAppSubmissionUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

iapId,

clonedSubmissionId),

requestContent: clonedSubmission).Wait();

// Tell the system that we are done updating the submission.

Console.WriteLine("Committing the submission");

client.Invoke<dynamic>(

HttpMethod.Post,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.InAppProductCommitSubmissionUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

iapId,

clonedSubmissionId),

requestContent: null).Wait();

// Periodically check the status until it changes from "CommitsStarted" to either

// successful status or a failure.

Console.WriteLine("Waiting for the submission commit processing to complete. This may take a couple of minutes.");

string submissionStatus = null;

do

{

Task.Delay(TimeSpan.FromSeconds(5)).Wait();

dynamic statusResource = client.Invoke<dynamic>(

HttpMethod.Get,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.InAppSubmissionStatusUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

iapId,

clonedSubmissionId),

requestContent: null).Result;

submissionStatus = statusResource.status.Value as string;

Console.WriteLine("Current status: " + submissionStatus);

}

while ("CommitStarted".Equals(submissionStatus));

if ("CommitFailed".Equals(submissionStatus))

{

Console.WriteLine("Submission has failed. Please check the Errors collection of the submissionResource response.");

return;

}

else

{

Console.WriteLine("Submission commit success! Here is the new price:");

dynamic sub = client.Invoke<dynamic>(

HttpMethod.Get,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.InAppSubmissionUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

iapId,

clonedSubmissionId),

requestContent: null).Result;

Console.WriteLine(sub.pricing.priceId.Value as string);

}

}

}

}

**Создание отправки тестового пакета**

В следующем примере реализован класс, который использует несколько методов в API отправки в Microsoft Store для обновления отправки тестового пакета. RunFlightSubmissionUpdateSampleМетод в классе создает новую передачу в виде клона последней опубликованной отправки, а затем обновляет и фиксирует клонированную отправку в центр партнеров. В частности, метод RunFlightSubmissionUpdateSample выполняет следующие задачи:

1. Сначала метод [получает данные для указанного тестового пакета](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/get-a-flight).
2. Затем он [удаляет ожидающую отправку для тестового пакета](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/delete-a-flight-submission), если она существует.
3. После этого [выполняется создание новой отправки для тестового пакета](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/create-a-flight-submission) (новая отправка — это копия последней опубликованной отправки).
4. Код передает новый пакет для отправки в хранилище BLOB-объектов Azure.
5. Затем он [обновляет](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/update-a-flight-submission) и [фиксирует](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/commit-a-flight-submission) новую отправку в Центре партнеров.
6. Наконец, он периодически [проверяет состояние новой отправки](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/get-status-for-a-flight-submission), пока она не будет успешно зафиксирована.

namespace DeveloperApiCSharpSample

{

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Globalization;

using System.Net.Http;

using System.Threading.Tasks;

using Newtonsoft.Json.Linq;

/// <summary>

/// Demonstrates how to update a flight submission with a new package

/// </summary>

public class FlightSubmissionUpdateSample

{

private ClientConfiguration ClientConfig { get; set; }

/// <summary>

/// Constructor

/// </summary>

/// <param name="c">An instance of ClientConfiguration that contains all parameters populated</param>

[DebuggerStepThrough]

public FlightSubmissionUpdateSample(ClientConfiguration c)

{

this.ClientConfig = c;

}

public void RunFlightSubmissionUpdateSample()

{

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// SETTINGS

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

var appId = this.ClientConfig.ApplicationId;

var flightId = this.ClientConfig.FlightId;

var clientId = this.ClientConfig.ClientId;

var clientSecret = this.ClientConfig.ClientSecret;

var serviceEndpoint = this.ClientConfig.ServiceUrl;

var tokenEndpoint = this.ClientConfig.TokenEndpoint;

var scope = this.ClientConfig.Scope;

// Get authorization token

Console.WriteLine("Getting authorization token ");

var accessToken = IngestionClient.GetClientCredentialAccessToken(

tokenEndpoint,

clientId,

clientSecret,

scope).Result;

Console.WriteLine("Getting flight");

var client = new IngestionClient(accessToken, serviceEndpoint);

dynamic flight = client.Invoke<dynamic>(

HttpMethod.Get,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.GetFlightUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

appId,

flightId),

requestContent: null).Result;

Console.WriteLine(flight.ToString());

if (flight.pendingFlightSubmission != null)

{

var submissionId = flight.pendingFlightSubmission.id.Value as string;

// Let's try deleting it. If it was NOT creationg via the API, then you need to

// manually delete it from the dashboard. This is a safety measure to make sure that a

// human user and an automated system don't make conflicting edits.

Console.WriteLine("Deleting the pending submission");

client.Invoke<dynamic>(

HttpMethod.Delete,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.GetFlightSubmissionUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

appId,

flightId,

submissionId),

requestContent: null).Wait();

}

// Create a new submission, which will be an exact copy of the last published submission.

Console.WriteLine("Creating a new submission");

dynamic flightSubmission = client.Invoke<dynamic>(

HttpMethod.Post,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.CreateFlightSubmissionUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

appId,

flightId),

requestContent: null).Result;

// Update packages.

// Let's say we want to delete the existing package:

flightSubmission.flightPackages[0].fileStatus = "PendingDelete";

// Let's add a new package.

var packages = new List<dynamic>();

packages.Add(flightSubmission.flightPackages[0]);

packages.Add(

new

{

fileStatus = "PendingUpload",

fileName = "package.appx",

});

flightSubmission.flightPackages = JToken.FromObject(packages.ToArray());

var flightSubmissionId = flightSubmission.id.Value as string;

// Upload the zip archive with all new files to the SAS URL returned with the submission.

var fileUploadUrl = flightSubmission.fileUploadUrl.Value as string;

Console.WriteLine("FileUploadUrl: " + fileUploadUrl);

Console.WriteLine("Uploading file");

IngestionClient.UploadFileToBlob(@"..\..\files.zip", fileUploadUrl).Wait();

// Update the submission.

Console.WriteLine("Updating the submission");

client.Invoke<dynamic>(

HttpMethod.Put,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.GetFlightSubmissionUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

appId,

flightId,

flightSubmissionId),

requestContent: flightSubmission).Wait();

// Tell the system that we are done updating the submission.

Console.WriteLine("Committing the submission");

client.Invoke<dynamic>(

HttpMethod.Post,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.CommitFlightSubmissionUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

appId,

flightId,

flightSubmissionId),

requestContent: null).Wait();

// Periodically check the status until it changes from "CommitsStarted" to either

// successful status or a failure.

Console.WriteLine("Waiting for the submission commit processing to complete. This may take a couple of minutes.");

string submissionStatus = null;

do

{

Task.Delay(TimeSpan.FromSeconds(5)).Wait();

dynamic statusResource = client.Invoke<dynamic>(

HttpMethod.Get,

relativeUrl: string.Format(

CultureInfo.InvariantCulture,

IngestionClient.FlightSubmissionStatusUrlTemplate,

IngestionClient.Version,

IngestionClient.Tenant,

appId,

flightId,

flightSubmissionId),

requestContent: null).Result;

submissionStatus = statusResource.status.Value as string;

Console.WriteLine("Current status: " + submissionStatus);

}

while ("CommitStarted".Equals(submissionStatus));

if ("CommitFailed".Equals(submissionStatus))

{

Console.WriteLine("Submission has failed. Please check the Errors collection of the submissionResource response.");

return;

}

else

{

Console.WriteLine("Submission commit success!");

}

}

}

}

**Вспомогательный класс IngestionClient**

Класс IngestionClient предоставляет вспомогательные методы, используемые другими методами в образце приложения для выполнения следующих задач:

* [Получение маркера доступа Azure AD](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/create-and-manage-submissions-using-windows-store-services#obtain-an-azure-ad-access-token), который можно использовать для вызова методов в API отправки в Microsoft Store. После получения маркера доступа у вас будет 60 минут, чтобы использовать его в вызовах к API отправки Microsoft Store до окончания срока действия маркера. После истечения срока действия маркера можно сформировать новый маркер.
* Upload ZIP-архив, содержащий новые ресурсы для приложения или надстройки, отправкой в служба хранилища больших двоичных объектов Azure. дополнительные сведения о передаче ZIP-архива в хранилище Blob-объектов Azure служба хранилища для отправки приложений и надстроек см. в соответствующих инструкциях в разделе [создание отправки приложения](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/manage-app-submissions#create-an-app-submission) и [создание отправки надстройки](https://learn.microsoft.com/ru-ru/windows/uwp/monetize/manage-add-on-submissions#create-an-add-on-submission).
* Обработайте запросы HTTP для API отправки в Microsoft Store.

namespace DeveloperApiCSharpSample

{

using System;

using System.Collections.Generic;

using System.IO;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Text;

using System.Threading.Tasks;

using Newtonsoft.Json;

using Microsoft.WindowsAzure.Storage.Blob;

/// <summary>

/// This class is a proxy that abstracts the functionality of the API service

/// </summary>

public class IngestionClient : IDisposable

{

public static readonly string Version = "1.0";

public static readonly string Tenant = "my";

public static readonly string GetSubmissionUrlTemplate = "/v{0}/{1}/applications/{2}/submissions/{3}";

public static readonly string CommitSubmissionUrlTemplate = "/v{0}/{1}/applications/{2}/submissions/{3}/commit";

public static readonly string UpdateUrlTemplate = "/v{0}/{1}/applications/{2}/submissions/{3}/";

public static readonly string ApplicationUrl = "/v{0}/{1}/applications";

public static readonly string ApplicationUrlWithContinuation = "/v{0}/{1}/{2}";

public static readonly string GetApplicationUrlTemplate = "/v{0}/{1}/applications/{2}";

public static readonly string GetApplicationIapsWithContinuationUrlTemplate = "/v{0}/{1}/{2}";

public static readonly string CreateSubmissionUrlTemplate = "/v{0}/{1}/applications/{2}/submissions";

public static readonly string GetApplicationIapsUrlTemplate = "/v{0}/{1}/applications/{2}/listinappproducts";

public static readonly string CreateInAppUrlTemplate = "/v{0}/{1}/inappproducts";

public static readonly string GetInAppUrlTemplate = "/v{0}/{1}/inappproducts/{2}";

public static readonly string InAppSubmissionUrlTemplate = "/v{0}/{1}/inappproducts/{2}/submissions/{3}";

public static readonly string InAppSubmissionUrl = "/v{0}/{1}/inappproducts/{2}/submissions";

public static readonly string InAppProductCommitSubmissionUrlTemplate = "/v{0}/{1}/inappproducts/{2}/submissions/{3}/commit";

public static readonly string GetApplicationFlightsUrlTemplate = "/v{0}/{1}/applications/{2}/listflights";

public static readonly string GetApplicationFlightsWithContinuationUrlTemplate = "/v{0}/{1}/{2}";

public static readonly string CreateNewFlightUrlTemplate = "/v{0}/{1}/applications/{2}/flights";

public static readonly string GetFlightUrlTemplate = "/v{0}/{1}/applications/{2}/flights/{3}";

public static readonly string CreateFlightSubmissionUrlTemplate = "/v{0}/{1}/applications/{2}/flights/{3}/submissions";

public static readonly string GetFlightSubmissionUrlTemplate = "/v{0}/{1}/applications/{2}/flights/{3}/submissions/{4}";

public static readonly string CommitFlightSubmissionUrlTemplate = "/v{0}/{1}/applications/{2}/flights/{3}/submissions/{4}/commit";

public static readonly string FlightSubmissionStatusUrlTemplate = "/v{0}/{1}/applications/{2}/flights/{3}/submissions/{4}/status";

public static readonly string ApplicationSubmissionStatusUrlTemplate = "/v{0}/{1}/applications/{2}/submissions/{3}/status";

public static readonly string InAppSubmissionStatusUrlTemplate = "/v{0}/{1}/inappproducts/{2}/submissions/{3}/status";

private HttpClient httpClient;

private readonly string accessToken;

/// <summary>

/// Initializes a new instance of the <see cref="IngestionClient" /> class.

/// </summary>

/// <param name="accessToken">

/// The acces token. This is JWT a token obtained from AAD allowing the caller to invoke the API

/// on behalf of a user

/// </param>

/// <param name="serviceUrl">The service URL.</param>

public IngestionClient(string accessToken, string serviceUrl)

{

if (string.IsNullOrEmpty(accessToken))

{

throw new ArgumentNullException("accessToken");

}

if (string.IsNullOrEmpty(serviceUrl))

{

throw new ArgumentNullException("serviceUrl");

}

this.accessToken = accessToken;

this.httpClient = new HttpClient

{

BaseAddress = new Uri(serviceUrl)

};

this.DefaultHeaders = new Dictionary<string, string>();

}

/// <summary>

/// Gets the default headers.

/// </summary>

public Dictionary<string, string> DefaultHeaders { get; private set; }

/// <summary>

/// Performs application-defined tasks associated with freeing, releasing, or resetting

/// unmanaged resources.

/// </summary>

public void Dispose()

{

if (this.httpClient != null)

{

this.httpClient.Dispose();

this.httpClient = null;

GC.SuppressFinalize(this);

}

}

/// <summary>

/// Gets the authorization token for the provided client id, client secret, and the scope.

/// This token is usually valid for 1 hour, so if your submission takes longer than that to complete,

/// make sure to get a new one periodically.

/// </summary>

/// <param name="tokenEndpoint">Token endpoint to which the request is to be made. Specific to your

/// AAD app. Example: https://login.windows.net/d454d300-128e-2d81-334a-27d9b2baf002/oauth2/token </param>

/// <param name="clientId">Client Id of your AAD app. Example" ba3c223b-03ab-4a44-aa32-38aa10c27e32</param>

/// <param name="clientSecret">Client secret of your AAD app</param>

/// <param name="scope">Scope. If not provided, default one is used for the production API endpoint.</param>

/// <returns>Autorization token. Prepend it with "Bearer: " and pass it in the request header as the

/// value for "Authorization: " header.</returns>

public static async Task<string> GetClientCredentialAccessToken(

string tokenEndpoint,

string clientId,

string clientSecret,

string scope = null)

{

if (scope == null)

{

scope = "https://manage.devcenter.microsoft.com";

}

dynamic result;

using (HttpClient client = new HttpClient())

{

string tokenUrl = tokenEndpoint;

using (

HttpRequestMessage request = new HttpRequestMessage(

HttpMethod.Post,

tokenUrl))

{

string strContent =

string.Format(

"grant\_type=client\_credentials&client\_id={0}&client\_secret={1}&resource={2}",

clientId,

clientSecret,

scope);

request.Content = new StringContent(strContent, Encoding.UTF8,

"application/x-www-form-urlencoded");

using (HttpResponseMessage response = await client.SendAsync(request))

{

string responseContent = await response.Content.ReadAsStringAsync();

result = JsonConvert.DeserializeObject(responseContent);

}

}

}

return result.access\_token;

}

/// <summary>

/// Uploads a file to blob using a SAS url

/// </summary>

/// <param name="fileName">Path to your zip file</param>

/// <param name="sasUrl">The SAS url which was returned to you when you cloned the submission

/// in FileUploadUrl</param>

/// <returns>A task which will complete when the file finishes uploading</returns>

public static async Task UploadFileToBlob(string fileName, string sasUrl)

{

using (Stream stream = new FileStream(fileName, FileMode.Open))

{

var blockBob = new CloudBlockBlob(new Uri(sasUrl));

await blockBob.UploadFromStreamAsync(stream);

}

}

/// <summary>

/// Invokes the specified HTTP method.

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="httpMethod">The HTTP method.</param>

/// <param name="relativeUrl">The relative URL.</param>

/// <param name="requestContent">Content of the request.</param>

/// <returns>instance of the type T</returns>

/// <exception cref="ServiceException"></exception>

public async Task<T> Invoke<T>(HttpMethod httpMethod,

string relativeUrl,

object requestContent)

{

using (var request = new HttpRequestMessage(httpMethod, relativeUrl))

{

this.SetRequest(request, requestContent);

using (HttpResponseMessage response = await this.httpClient.SendAsync(request))

{

T result;

if (this.TryHandleResponse(response, out result))

{

return result;

}

if (response.IsSuccessStatusCode)

{

var resource = JsonConvert.DeserializeObject<T>(await response.Content.ReadAsStringAsync());

return resource;

}

throw new Exception(response.Content.ReadAsStringAsync().Result);

}

}

}

/// <summary>

/// Sets the request.

/// </summary>

/// <param name="request">The request.</param>

/// <param name="requestContent">Content of the request.</param>

protected virtual void SetRequest(HttpRequestMessage request, object requestContent)

{

request.Headers.Add(Constants.RequestHeaders.CorrelationIdHeader, Guid.NewGuid().ToString());

request.Headers.Add(Constants.RequestHeaders.MSRequestIdHeader, Guid.NewGuid().ToString());

request.Headers.Authorization = new AuthenticationHeaderValue("Bearer", this.accessToken);

foreach (var header in this.DefaultHeaders)

{

request.Headers.Add(header.Key, header.Value);

}

if (requestContent != null)

{

request.Content = new StringContent(JsonConvert.SerializeObject(requestContent),

Encoding.UTF8,

Constants.HttpMimeTypes.JsonContentType);

}

}

/// <summary>

/// Tries the handle response.

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="response">The response.</param>

/// <param name="result">The result.</param>

/// <returns>true if the response was handled</returns>

protected virtual bool TryHandleResponse<T>(HttpResponseMessage response, out T result)

{

result = default(T);

return false;

}

private static class Constants

{

public static class RequestHeaders

{

/// <summary>

/// Corresponds to TraceCorrelationId in SLL. This is a GUID that is newly generated

/// by FD for every request coming from the client.

/// </summary>

public const string CorrelationIdHeader = "MS-CorrelationId";

/// <summary>

/// Corresponds to RequestCorrelationId in SLL. This is a GUID that is newly generated

/// by FD for every request that it makes to the downstream services.

/// </summary>

public const string MSRequestIdHeader = "MS-RequestId";

}

public static class HttpMimeTypes

{

/// <summary>

/// The json content type

/// </summary>

public const string JsonContentType = "application/json";

}

}

}

}