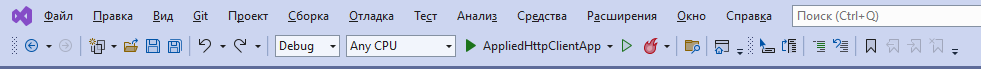
Прикладной Http клиент. Инструкция по использованию.

# Сборка решения

Для сборки библиотеки и подключения в прикладном решении Directum RX, необходимо скачать библиотеку с [репозитория](https://tfsozrrx.directum.ru/DefaultCollection/DirRXCustom/_git/AppliedHttpClient) и выполнить сборку решения.

Решение, скачанное с репозитория необходимо открыть в Visual Studio, после выполнить сборку. Более подробно прочитать про сборку можно в [справке](https://learn.microsoft.com/ru-ru/visualstudio/get-started/csharp/run-program?view=vs-2022) Visual Studio.

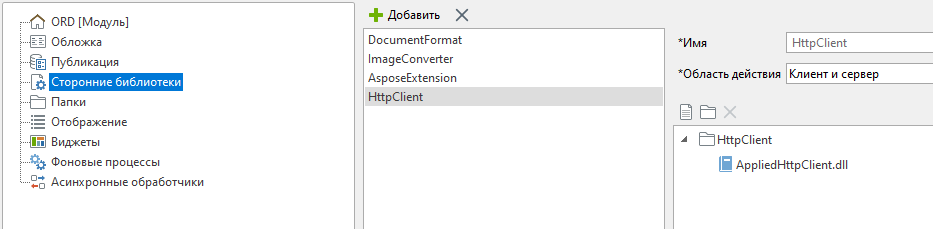
Для сборки решения необходимо выбрать зеленую стрелку «Запуск» и имя проекта на панели инструментов Visual Studio. В данном примере выбран проект AppliedHttpClientApp.



# Подключение библиотеки

Добавление сторонней библиотеки в Directum RX описано в [справке](https://help.npo-comp.ru/DirectumRX/4.4/ru-RU/index.html?sungerodev_moduleeditor_assembliestab.htm). При добавление сторонней библиотеки необходимо добавить файл с расширением dll. Файлы с расширением pdb, в случае если такие имеются, добавлять не стоит, т.к они нужны для отладки решения в Visual Studio.

### Пример подключенной библиотеки



# Описание методов библиотеки

## Методы HTTP запроса

В библиотеке предусмотрено 4 метода HTTP запроса, а именно Get, Post, Put, Delete. Под каждый из этих методов выделены одноименные методы.

## Методы добавления аутентификации

В библиотеке добавлены 3 методы аутентификации, а именно Basic, Bearer, Ntlm. Для добавления к запросу необходимого метода аутентификации вызовите одноименный метод.

* Basic - вместе с запросом отправляется логин и пароль в виде строки зашифрованной в base64;
* Bearer - вместе с запросом отправляется токен доступа;
* Ntlm - вместе с запросом отправляется логин и пароль.

## Методы добавления тела запроса

При необходимости отправить с запросом какие-либо данные определены три метода:

* BodyJson – добавляет к телу запроса данные в виде json строки;
* BodyXml - добавляет к телу запроса данные в виде xml строки;
* Upload – позволяет выполнить загрузку файла на удаленный сервер.

## Обработка результата запроса

Для обработки результата запроса определено 2 метода:

* OnSuccess – выполняется в случае успешного выполнения запроса;

Данный метод в себе содержит структуру ответа из следующих свойств:

* + StatusCode – статус код запроса;
  + Headers – заголовки ответа;
  + Response – ответ на запрос в виде потока.
* OnFail - выполняется в случае ошибки при выполнении запроса.

## Отправка запроса

Для отправки запроса определен метод Send.

# Примеры использования в прикладном коде

Перед использованием библиотеки в прикладном коде, необходимо подключить библиотеку с помощью оператора using.

Отправка метода Get и получение ответа в виде json строки

В примере используется fake api, в котором получаем с сервера данные о продукте в виде json строки.

**// Добавленная сторонняя библиотека**

**using** AppliedHttpClient;

**namespace** DirRX.Http.Client  
{  
  **public** class ModuleFunctions  
  {  
    /// <**summary**>  
    /// Получение продукта.  
    /// <**/summary**>  
    **public** virtual void **GetProduct**()  
    {  
      var resultProduct = string.Empty;  
        
      AppliedHttpClient.AppliedHttpClient.**Get**("https://www.onlinetool.in/api/products/2/")  
       // Получение ответа при успешном запросе  
      .**OnSuccess**(result =>  
       {  
         **using** (StreamReader sr = **new** **StreamReader**(result.Response))  
         {  
           resultProduct = sr.**ReadToEnd**();  
         }  
       })  
        // Получение ответа при ошибке во время выполнения запроса  
      .**OnFail**(result =>  
      {  
        resultProduct = result.Response;  
      })  
      .**Send**();  
    }

}

### Применение basic аутентификации

**// Добавленная сторонняя библиотека**

**using** AppliedHttpClient;

**namespace** DirRX.Http.Client  
{  
  **public** class ModuleFunctions  
  {  
    /// <**summary**>  
    /// Выполнить аутентификацию.  
    /// <**/summary**>  
    **public** virtual void **Authenticate**()  
    {  
      var resultProduct = string.Empty;  
             AppliedHttpClient.AppliedHtttpClient.**Get**("https://localhost:57345/api/Values/basic")  
        // Добавление basic аутентификации  
        .**Basic**("login", "password")  
       // Получение ответа при успешном запросе  
      .**OnSuccess**(result =>  
       {  
         **using** (StreamReader sr = **new** **StreamReader**(result.Response))  
         {  
           resultProduct = sr.**ReadToEnd**();  
         }  
       })  
        // Получение ответа при ошибке во время выполнения запроса  
      .**OnFail**(result =>  
      {  
        resultProduct = result.Response;  
      })  
      .**Send**();  
    }

}

### Загрузка файла на удаленный сервер

**// Добавленная сторонняя библиотека**

**using** AppliedHttpClient;

**namespace** DirRX.Http.Client  
{  
  **public** class ModuleFunctions  
  {  
    /// <**summary**>  
    /// Загрузить файл на удаленный сервер.  
    /// <**/summary**>  
    **public** virtual void **UploadFile**()  
    {  
      var resultProduct = string.Empty;  
                   AppliedHttpClient.AppliedHtttpClient.**Post**("http://localhost:5000/weatherforecast/upload")  
        // Загрузка файлов на удаленный сервер  
        .**Upload**("D:\TestDoc.docx", "file")  
       // Получение ответа при успешном запросе  
      .**OnSuccess**(result =>  
       {  
         **using** (StreamReader sr = **new** **StreamReader**(result.Response))  
         {  
           resultProduct = sr.**ReadToEnd**();  
         }  
       })  
        // Получение ответа при ошибке во время выполнения запроса  
      .**OnFail**(result =>  
      {  
        resultProduct = result.Response;  
      })  
      .**Send**();  
    }

}