Разработка сторонних клиентских приложений

TotalApi SDK

# Типы клиентских приложений

Клиентские приложения можно условно разделить на два типа:

1. **Приложения без бекенда** – толстые клиенты без удалённой серверной части для работы которых не требуется сетевое соединение. Примерами таких приложений являются почти все десктопные, а также многие мобильные приложения.
2. **Приложения с бекендом** – тонкие (в большинстве случаев) клиенты, имеющие удалённую серверную часть (обычно сервисы для доступа к данным). Примером таких приложений являются почти все известные веб-приложения, многие мобильные приложения, а также некоторые десктоп приложения, требующие интернет соединения для своей работы для обработки данных.

С точки зрения **TotalApi** нет особой разницы между этими типами приложения, однако есть разница в используемом виде аутентификации для каждого из типов приложения.

# Состав клиентского SDK .NET

Для использования **TotalApi SDK** в клиентском приложении необходимо наличие следующих библиотек:

* **TotalApi.Utils.Common**
* **TotalApi.Utils.IoC**
* **TotalApi.Utils.Wcf**
* **TotalApi.Core**
* **TotalApi.Billing.Client**
* **TotalApi.Telematics.Api**
* **TotalApi.Telematics.Client**

Последние две библиотеки необходимы, если вы собираетесь использовать телематическую часть подсистемы **TotalApi**.

При написании *клиентского* приложения не включайте в проект и не помещайте в папку SDK другие библиотеки из полного состава. В противном случае возможна некорректная работа приложения.

# Запуск и подключение

Для инициализации SDK и подключения к сервисам **TotalApi** необходимо указать адрес подключения к серверу биллинга **TotalApi**.

Это можно сделать через конфигурационный файл TotalApi.Server.Host.json:

{  
 "serverHost": "https://svc.totalapi.io:4444"  
 }

После чего при старте приложения следует вызвать метод инициализации клиентского SDK:

TotalApiBootstrapper.AppModulesSearchPattern = "myApp.\*.dll";  
 TotalApiBootstrapper.Create(".");

Параметром вызова является каталог, в котором находятся файлы SDK. (Или список каталогов, разделённых точкой с запятой).

SDK для своей работы использует [MEF](https://msdn.microsoft.com/library/dd460648%28v=vs.110%29.aspx) и данный вызов производит компоновку композиции. Хорошая новость: теперь вы можете использовать [MEF](https://msdn.microsoft.com/library/dd460648%28v=vs.110%29.aspx) в своём приложении для своих классов без дополнительного конфигурирования, если названия сборок, содержащих экспортируемые классы, соответствуют маске, установленной в свойстве AppModulesSearchPattern. Если данный параметр не установлен – никакие другие сборки не будут автоматически подключены. (Сборка из которой была произведена инициализация [MEF](https://msdn.microsoft.com/library/dd460648%28v=vs.110%29.aspx) будет подключена к композиции в любом случае).

Не устанавливайте маску \*.dll, т.к. в этом случае будет сделана попытка добавить в композицию все сборки, находящиеся в указанных папках, (в том числе и системные), что может привести к очень долгому старту программы.

Не меняйте названия файлов SDK, т.к. они включаются в композицию по маске TotalApi.\*.dll

# Авторизация

При использовании **TotalApi SDK** весь механизм авторизации осуществляется автоматически клиентским модулем биллинга (TotalApi.Billing.Client.dll). Разработчик должен только указать значения авторизационных ключей ([ApiKey](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#ApiKey) или [AppKey](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#AppKey) или [AppKey](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#AppKey), [UserLogin](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#UserLogin) и [UserPassword](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#UserPassword)). Вид авторизации определяется автоматически в зависимости от того, что было указано.

Указать значения авторизационных ключей можно

* либо через конфигурационный файл TotalApi.Server.Host.json (только [ApiKey](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#ApiKey) или [AppKey](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#AppKey)):

{  
 "apiKey": "myApiKey"  
 }

* либо явно

TotalApiAuth.AppKey = "myAppKey";  
 TotalApiAuth.UserLogin = "user1";  
 TotalApiAuth.UserPassword = "pass";

Для **приложений с бекендом** следует использовать [ApiKey-аутентификацию](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#ApiKeyАвторизация) для доступа к сервисам **TotalApi**. Причём вся работа с сервисами **TotalApi** должна обязательно осуществляться на бекенде с целью сокрытия секретного [ApiKey](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#ApiKey) от пользователей.

Схема использования [ApiKey-аутентификацию](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#ApiKeyАвторизация) для доступа к сервисам **TotalApi**:

TotalAPI

Клиентское

приложение

Бекенд

приложения

ApiKey

Так как кодовая база **приложений без бекенда** полностью доступна для любого пользователя, нельзя использовать [ApiKey-аутентификацию](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#ApiKeyАвторизация) для доступа к сервисам **TotalApi**. В противном случае секретный [ApiKey](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#ApiKey) может быть легко скомпрометирован, т.к. он должен так или иначе присутствовать в кодовой базе приложения.

Поэтому для **приложений без бекенда** следует использовать [AppKey-аутентификацию](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#AppKeyАвторизация) и [AppUser-аутентификацию](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#AppUserАвторизация).

Рассмотрим типичный пример многопользовательского приложение для трекинга зарегистрированных устройств слежения без бекенда на примере десктопного приложения.

1. Произвольный пользователь скачивает приложение и устанавливает его на своём компьютере.
2. При запуске приложения его просят ввести логин/пароль, либо зарегистрироваться в системе.
3. Пользователь регистрируется в системе и в сервисы **TotalApi** уходит запрос на добавление нового пользователя. Для того, чтобы этот запрос был успешным приложение должно использовать [AppKey-аутентификацию](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#AppKeyАвторизация) и [AppKey](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#AppKey) сохранённый в коде программы. Это единственный метод в **TotalApi**, который можно сделать, используя [AppKey-аутентификацию](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#AppKeyАвторизация).
4. После регистрации пользователь вводит свой логин и пароль, которые он указал при регистрации и приложение в дальнейшем использует их вместе с [AppKey](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#AppKey) для осуществления [AppUser-аутентификаци](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#AppUserАвторизация)и, при выполнении остальных операций (добавление/удаление устройств слежения, получение координат от них и т.д.).
5. Другой пользователь, даже зная [AppKey](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#AppKey), не сможет получить данные первого пользователя, не зная его логина и пароля.

Схема использования [AppKey-аутентификации](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#AppKeyАвторизация) и [AppUser-аутентификаци](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#AppUserАвторизация)и для доступа к сервисам **TotalApi**:

Register

TotalAPI

Клиентское

приложение

AppKey

Login + Password

Схема использования [AppKey-аутентификации](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#AppKeyАвторизация) и [AppUser-аутентификации](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#AppUserАвторизация) для доступа к сервисам **TotalApi**:

TotalAPI

Клиентское

приложение

AppKey + UserLogin + UserPassword

Login + Password

Подробно виды аутентификации описаны в [этом документе](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx).

# Примеры взаимодействия с TotalAPI, используя SDK

## Базовые API

Для доступа к набору базовых API программного комплекса **TotalApi**, используется статический класс CoreApi. Ниже приведено их краткое описание с примерами использования:

## Репозиторий

Доступ к [распределённому репозиторию](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/RepositoryAPI.docx) программного комплекса **TotalApi**, осуществляется посредством интерфейса CoreApi.Repository:

// The sample of the using Repository API  
 // Getting entities count (without any condition)  
 var startCount = CoreApi.Repository.Count<Device>();  
 // Creating a new entity  
 var **device** = new Device  
 {  
 Name = "My Device",   
 ModelCode = 100,  
 PhoneNumber = "+555-1111"  
 };  
 **device** = CoreApi.Repository.Save(**device**, true);  
  
 // Getting the Id of created entity  
 var deviceId = **device**.Id;  
  
 // Modifying the entity  
 **device**.Imei = "12345678901234";  
 CoreApi.Repository.Save(**device**, false);  
   
 // Searching an entity by Id - it should be found  
 **device** = CoreApi.Repository.Find<Device>(deviceId);  
 Debug.Assert(**device**.Id == deviceId, "Should be equal");  
   
 // Searching an entity with a query (scalar result)  
 **device** = CoreApi.Repository  
 .ExecuteScalar<Device>(q => q.FirstOrDefault(d => d.Imei == "12345678901234"));  
 Debug.Assert(**device**.Id == deviceId, "Should be equal");  
   
 // Searching an entity with a query (multi result)  
 **device** = CoreApi.Repository  
 .ExecuteQuery<Device>(q => q.Where(d => d.Imei == "12345678901234"))  
 .First();  
 Debug.Assert(**device**.Id == deviceId, "Should be equal");  
   
 // Getting entities count (without any condition)  
 var **currentCount** = CoreApi.Repository.Count<Device>();  
 Debug.Assert(**currentCount** == startCount + 1, "Should be equal");  
   
 // Deleting an entity  
 CoreApi.Repository.Delete<Device>(deviceId);  
   
 // Searching an entity by Id - it should not be found  
 **device** = CoreApi.Repository.Find<Device>(deviceId);  
 Debug.Assert(**device** == null, "Should be null");  
   
 // Getting entities count (without any condition)  
 **currentCount** = CoreApi.Repository.Count<Device>();  
 Debug.Assert(**currentCount** == startCount, "Should be equal");

Обратите внимание, что при вызове методов репозитория необходимо указывать тип бизнес-объекта, с которым мы работаем, за исключением тех случаев, когда этот тип заранее известен компилятору. Но не нужно указывать адрес (URL) подсистемы, которая обслуживает бизнес-объекты заданного типа (в нашем случае это телематическая подсистема **TotalApi**).

Также обратите внимание на то, каким образом передаются условия отбора при вызове методов ExecuteScalar(), ExecuteQuery() и Count() . При передаче условий отбора таким образом фильтрация данных будет происходить на серверной стороне, что может существенно ускорить работу запроса. Однако в таком случае можно использовать только стандартные функции и операторы, а также реальные поля бизнес-объектов. Другими словами, не любое условие можно указать в качестве параметра. Если же всё-таки это необходимо – воспользуйтесь отбором результата на клиентской стороне после выполнения запроса без этого условия.

**device** = CoreApi.Repository.ExecuteQuery<Device>().First(d => **this**.IsMatched(d));

## События

Работа с [распределёнными событиями](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/EventsAPI.docx) программного комплекса **TotalApi**, осуществляется посредством интерфейса CoreApi.EventManager:

class Subscriber : IEvent, IEvent<OnPing>, IEvent<OnDeviceStatusChanged>  
{  
 public Subscriber()  
 {  
 // Подписка на получения событий.

CoreApi.EventManager.Subscribe(this);

// Подписка на получения событий OnDeviceStatusChanged от устройств "1", "2", "3".

TelematicsApi.Events.SubscribeOnDeviceEvents(new [] {"1", "2", "3"}, true);

}  
  
 public void HandleEvent(object e)  
 {  
 Console.WriteLine("1. IEvent: " + e.GetType().FullName));  
 }  
  
 public void HandleEvent(OnPing e)  
 {  
 Console.WriteLine("2. OnPing: " + e.Content));  
 }  
  
 public void HandleEvent(OnDeviceStatusChanged e)  
 {  
 Console.WriteLine("3. OnDeviceStatusChanged: {0}".Fmt(e.DeviceStatus.Id)));  
 }  
}

...  
 CoreApi.EventManager.Publish(new OnPing("Test"));  
 ...

**Важное замечание:** Подписчик события может поймать это событие даже если он было послано из другой подсистемы программного комплекса. В данном примере подписчик принимает события об изменении статуса устройств слежения которое генерируется в другом приложении и даже другом компьютере.

Для получения некоторых событий (таких как OnDeviceStatusChanged) необходимо вызывать дополнительный метод подписки (в данном случае TelematicsApi.Events.SubscribeOnDeviceEvents) для того, чтобы клиентскому приложению приходили только события от указанных устройств слежения. Это необходимо для уменьшения нагрузки на клиентскую и серверную части программного комплекса, связанной с передачей ненужных данных. Например, при наличии порядка 100К активных устройств слежения, которые передают координаты каждую секунду – получим 100К уведомлений в секунду, которые будут отправляться всем подписавшимся клиентам. С другой стороны, скорее всего на каждом из клиентов нет необходимости следить в реальном времени за всеми 100К устройствами, скорее всего каждый из них следит за своим ограниченным небольшим (по сравнению со 100К) набором. В этом случае правильно спроектированное клиентское приложение должно подписаться только на получение данных для своего набора устройств слежения, что, очевидно, существенно снизит нагрузку.

Также и отправка события посредством CoreApi.EventManager.Publish() может отправить события ко всем подписчикам программного комплекса. Даже другому экземпляру этого приложения.

Помимо стандартных событий, можно использовать свои собственные. Для этого достаточно создать собственный класс события, пронаследовав его от TotalApiEventObject. Такие события будут передаваться между приложениями одного программного комплекса **TotalApi** имеющих одинаковую аутентификацию. (Например, оба приложения используют [ApiKey-аутентификацию](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#ApiKeyАвторизация) с одинаковым [ApiKey](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#ApiKey) или [AppUser-аутентификацию](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/AuthenticateAPI.docx#AppUserАвторизация) с одинаковым набором аутентификационных данных.

## Протоколирование

Для протоколирования программного комплекса **totalApi**, используйте интерфейс CoreApi.Logger:

CoreApi.Logger.Log("Found config file '{0}'. Loading...", fileName);  
 CoreApi.Logger.Error(e);  
 CoreApi.Logger.Warning("GC has disposed orphan subscriber.");

По умолчанию протоколирование ведётся в консоль и файл, согласно параметрам протоколирования, указанным в конфигурационном файле. Однако можно легко добавить свои обработчики, просто экспортировав один или несколько классов, реализующих интерфейс ILogger.

## Телематический API

Для доступа к набору телематических API модуля **TotalApi** программного комплекса **TotalApi**, используется статический класс TelematicsApi. Подробные примеры его использования в клиентском приложении приведены [в документации телематического API](https://d.docs.live.net/2977baf408e5856c/TotalApi/API%20docs/TelematicsAPI.docx).