Быстрый старт с TelegaPi для Delphi

# Установка.

Скачиваем: <https://bitbucket.org/RareGods/cloudapi/downloads/>

Распакуйте архив.

В среде Delphi добавьте 2 пути (Options/Delphi options/Library path):

* <путь установки TelegaPi>\Install\Win32\Debug
* <путь установки TelegaPi>\DCU\Win32-Debug

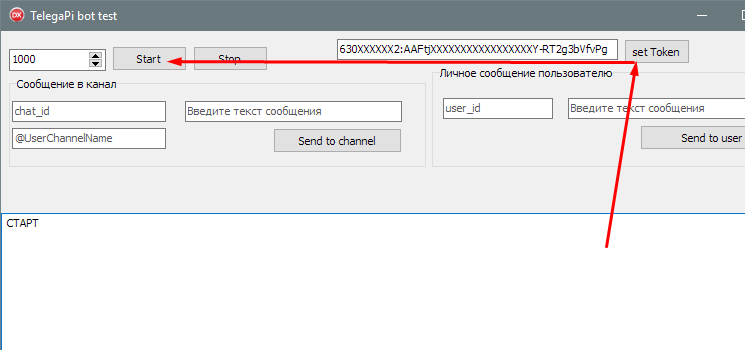
В Browsing path можно добавить ссылки на папки к исходникам:

* <путь установки TelegaPi>\Source\CoreAPI
* <путь установки TelegaPi>\Source\Telegram.org

Откройте файл *<путь установки TelegaPi>All API.groupproj*, сделайте Build и затем Install. Закройте.

Откройте файл *<путь установки TelegaPi>InstallAndDemo.groupproj*, сделайте Build и затем Install. Закройте.

Начнём с простого UI приложения. Создайте пустое VCL приложение, киньте на форму две компоненты: TtgReceiverUI и TTelegramBot. В tgReceiverUI1.Bot выберите TelegramBot1. У компоненты TelegramBot1 заполните свойство token или пропишите его в edToken.Text, киньте на форму TMemo. Киньте на форму несколько TEdit и 4 кнопки и TSpinEdit. Визуально это выглядит так:



После запуска приложения нажмите setToken, затем Start. Всё, ваше приложение раз в секунду обращается к Телеграм-серверу и получает от него «Updates», т.е. те данные, которые шлют пользователи через свои Телеграм-клиенты Телеграм-боту.

Найдите своего бота в Телеграм-клиенте и обратитесь к нему.

Важно. Пользователь сам должен обратиться к боту первым или присоединиться к нему, к вашему боту. При этом сразу отправляется команда /Start, на которую ваше приложение должно уметь отреагировать каким-нибудь приветственным сообщением. Когда пользователь напишет боту или присоединится, в TMemo увидите сырое (RAW) JSON-сообщение. При этом edUserId (справа) в данном примере заполниться и там будет TelegramID пользователя-отправителя. Сразу же можно написать текст и нажать btnUserSend. Пользователь, получит ваш текст.

У сервиса Телеграм есть несколько разных типов сообщений: текст, картинка, виртуальная клавиатура, звук, видео и т.д. К сожалению, сообщений в виде каруселей нет, как это сделано в Вайбер или Фейсбук. Но сообщения делятся ещё на 2 типа: для пользователя и для (публичного) канала. Забегая вперёд, напишу, что для канала chat\_id – это отрицательное целочисленное значение, а chat\_id для пользователя – это положительное целочисленное значение.

Далее вам нужно зарегистрировать/создать свой аккаунт в Телеграм, если его ещё нет и создать бота в Телеграм-мессенджере. Вам потребуется смартфон и установленное приложение Telegram.

Чтобы отправлять сообщения в публичный чат (канал), нужно с помощью Телеграм-приложения привязать вашего бота к вашему каналу. Кстати, согласно ограничений Телеграм, в публичном канале нельзя получить ID пользователя-отправителя сообщения, т.к. Телеграм-сервер отдаёт боту только ID канала.

Как зарегистрировать своего бота: <https://tlgrm.ru/docs/bots#3-how-do-i-create-a-bot>

# Описание компонент.

## TTelegramBot

Это основная компонента. Она используется для всех типов приложений: UI, console, service в отличие от компонент-ресиверов (TtgReceiverXxxxxx), где используются разные типы ресиверов.

*UI приложение* - происходит регистрация компоненты в IDE. Вызов событий синхронизируется.

*Service приложение* - происходит регистрация компоненты в IDE. Вызов событий НЕ синхронизируется.

*Консольное приложение* - НЕ происходит регистрация компоненты в IDE. Вызов событий НЕ синхронизируется.

Под синхронизацией подразумевается синхронизация компоненты-ресивера с основным потоком приложения, как это выглядит в исходниках:

procedure TtgReceiverUI.EventParser(AUpdates: System.TArray<TelegAPI.Types.ItgUpdate>);

begin

TThread.Synchronize(nil,

procedure

begin

inherited EventParser(AUpdates);

end);

end;

### Свойства TTelegramBot

**Token** – для ключа, который вы получили от главного Телеграм бота [@BotFather](https://telegram.me/botfather)

**Domain** – здесь всегда API URL <https://api.telegram.org/bot>

После регистрации бота, вы получите токен вида «630007585:BBFtjqPkJFbY77XACgr98KY-RT2g3bXxXxx». Никому не передавайте и не показывайте токен. Присвойте токе полю *TelegramBot1.Token* в коде или в дизайне.

### Методы TTelegramBot

**GetMe** – возвращает ItgUser, где можно узнать данные своего бота: id, username, LanguageCode и т.д.

**SendPhoto** – а так же SendAnimation, SendContact и т.д. отправить медийное сообщение.

**SetWebhook** – установить Webhook. Telegram Webhook – это технология, позволяющая отслеживать в чате события в реальном времени и отправлять информацию о нем на указанный адрес. Для Webhook вам потребуется SSL-сертификат, а для getUpdates – не потребуется. Telegram поддерживает всего 4 различных порта при работе с самоподписанными сертификатами (443, 80, 88 или 8443). Теоретически, это означает, что на одном ПК может быть запущено не больше 4х ботов на вебхуках. <https://core.telegram.org/bots/webhooks>

Что нужно для рабочего коннекта Webhook: адрес (URL) страницы с программным кодом; поддержка IPv4; возможность сервера обрабатывать HTTPS трафик; наличие SSL сертификата. Прежде чем использовать Webhooks убедитесь, что вам доступен этот функционал. Создание бота для Telegram на Webhooks предпочтительнее, чем использования метода getUpdates. Во втором случае приходится закладывать в код постоянное обращение этой команды к Телеграм-серверу. Это делается в циклическом режиме. Webhook держит постоянное соединение, но запросы отправляются только когда произошло событие на Телеграм-сервер, например, пользователь ввел сообщение и сервер пересылает вашему приложению эту команду.

### События TTelegramBot

**OnSendData** – происходит, когда компонента отправляет что-то на сервер. Можно посмотреть RAW URL, который передаётся серверу в параметре AURL. В параметр AData (в виде MultipartFormData) может передаваться дамп данных, если бот отправляет музыку, видео или клавиатуру. Как это выглядит в исходниках:

function TApiRequest.FormDataToString(AFormData: TMultipartFormData): string;

var

LStrList: TStringList;

LPos: Int64;

begin

LStrList := TStringList.Create;

try

LPos := AFormData.Stream.Position;

AFormData.Stream.Position := 0;

LStrList.LoadFromStream(AFormData.Stream);

Result := LStrList.Text;

AFormData.Stream.Position := LPos;

finally

LStrList.Free;

end;

end;

**OnReceiveRawData** – происходит, когда компонента получает что-то от сервера. Можно посмотреть RAW данные в виде JSON.

Пример RAW AData: *{"ok":true,"result":[{"update\_id":519749xxx,*

*"channel\_post":{"message\_id":68,"chat":{"id":-100120381xxxx,"title":"Заголовок канала","username":"имя\_пользователя\_канала","type":"channel"},"date":1540381986,"text":"текст"}}]}*

**OnError** – срабатывает, когда от сервера приходит сообщение об ошибке, например, «(400) chat not found». Из параметра Exception: ECloudApiException можно узнать: код, описание, url запроса, ответ и дату со временем. В методе Exception.**ToString** можно получить строку из даты, времени, кода и сообщения.

## TtgReceiverUI

Компонента-ресивер для получения данных от сервера Телеграм.

### Свойства TtgReceiverUI

**AllowedUpdates** – какие виды сообщение запрашивать у сервера.

**Bot** – основная компонента типа TTelegramBot.

**MessageOffset** – заполняется автоматически. С помощью этого свойства указываем серверу, какое последнее обновление (UpdateID) нам пришло. Изменяя это значение, можно запрашивать у Телеграм-сервера старые сообщения.

**PollingInterval** – микросекунды, интервал - через сколько времени запрашивать данные от Телеграм -сервера, по умолчанию 1000 мкс, т.е. 1 секунда. При изменении интервала не обязательно перезапускать ресивер, т.е. не обязательно выполнять tgReceiverUI1.Start и затем tgReceiverUI1.Stop.

### Методы TtgReceiverUI

**Start/Stop** – запуск и остановка ресивера.

**IsActive** – Проверка, активен ли ресивер.

### События TtgReceiverUI

**OnCallbackQuery** –

**OnChannelPost** – когда в публичный канал пользователь отправляет сообщение, срабатывает это событие. Параметр AMessage: ITgMessage – это JSON-объект. RAW данные можно получить так:

Memo1.Lines.Add('OnChannelPost RAW:' + (AMessage as TBaseJson).AsJson);

Как описано выше, в публичном канале нельзя получить ID пользователя-отправителя, но можно получить ID публичного канала.

**OnMessage** – срабатывает, когда боту приходит личное (приватное) сообщение от пользователя. Здесь можно получить данные о пользователе в AMessage.From. Структура From (ItgUser = interface) : ID, IsBot, Username и т.д. Декодированный текст сообщения - AMessage.Text. У интерфейса AMessage много других методов, например, Date, Audio, Document, Caption, Contact.

Когда получаете объект AMessage.Contact, то всегда проверяйте, что он не равен NIL. Т.к. Contact будет заполнен только, если пользователь выберет в Телеграм-клиенте меню «Отправить мои контакты», тогда боту придет объект, содержащий телефон, имя и др. информацию.

## TtgUserLink

**TtgUserLink** – это запись (Record) с полями ID, UserName и т.д. TtgUserLink - класс, в котором хранится указатель на конкретного адресата (пользователь, чат, канал). В нем хранится как правило ID пользователя/чата. Либо же имя канала, в который будет отправляться сообщение. В основном, этот класс заполняет программист, в котором указывает куда будет доставляться сообщение. Рекомендуется использовать, когда отправляется кому-то/куда-то сообщение, т.е. в качестве адресата (получателя).

## ItgUser

**ItgUser** – это запись (структура) с полями UserName, UserID, FirstName, LastName и т.д. Это информация от Телеграм-сервера о том пользователе, который написал в приватный чат, т.е. личное сообщение боту. У компоненты-ресивера см. событие OnMessage или OnChannelPost, где параметр AMessage.**From** будет содержать некоторую информацию о пользователе и/или чате/канале.