Checkbox Developer Manual

Getting started to communicate with the Checkbox for a developer is the scope for this document. How a connection is setup, how to use and configure the Checkbox in order to get started.

Checkbox Connection

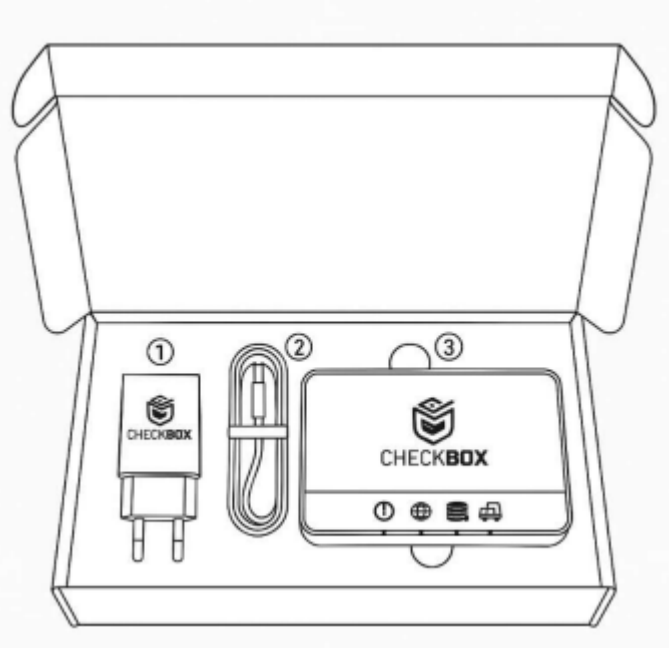
This Chapter will explain how a checkbox needs to be connected physically, and how you can get a status out of it to check if everything works well.

Configuring the Communication

Physical Connections to the Checkbox

Power

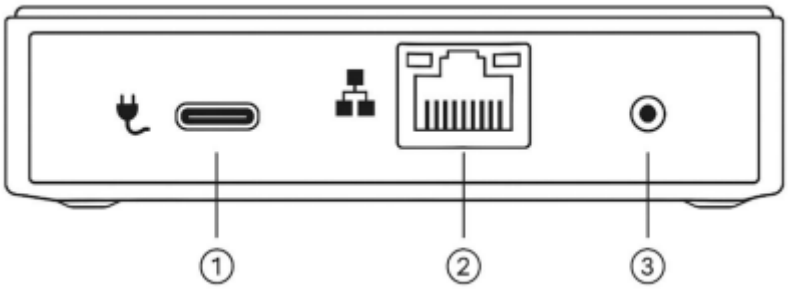
The Checkbox needs to be powered, powering is done by an USB-C cable. In the box there is an adapter (1) and a power cable (2) available to use.



Network Connection

In order to be able to cmmunicate with the Checkbox, there is a network connection needed. To achieve this there are 2 options.

1. Ethernet Connection with a standard network cable (RJ45)
2. A Wi-Fi connection



(1) is the USB power connector
(2) is the Ethernet connector
(3) is the Multifunctional Button

Network Settings Configuration

The first time you want to connect to the Checkbox, you will have to use a network cable. By default the DHCP is enabled.
once there is a connection available with internet connectivity, you have multiple ways to setup your connection

1. The Checkbox Tools Backend

You can login to the Tools backend, as a developer you should have an account in order to set all the settings to the Checkbox, and view its status, whereabouts, historical data, etc.

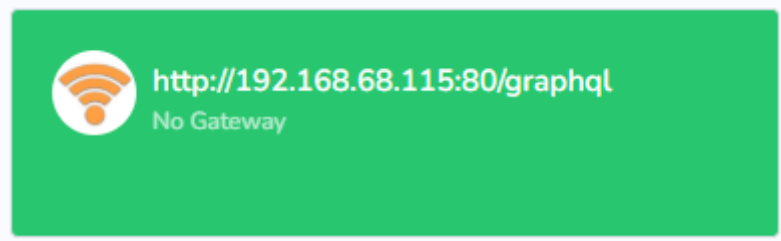
2. The Admin Console GraphQl endpoint

The checkbox is equipped with a GraphQl endpoint where you can talk to, that endpoint is explained further down this document. One of the things you can achieve there is changing the network configuration, setting the Wifi credentials, and so on.

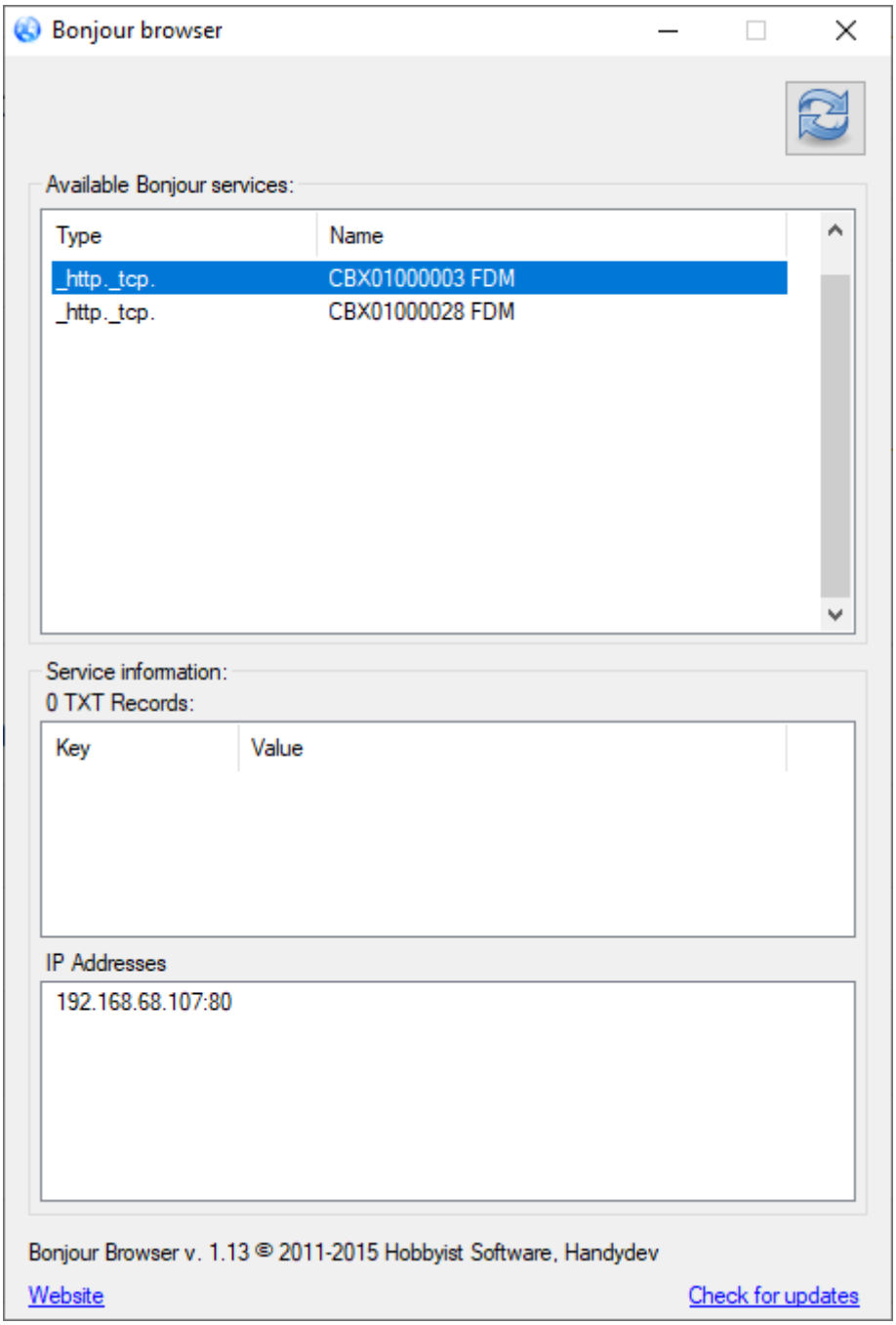
Finding the network address

Once a checkbox receives an address, Whether it is configured manually or it is retrieved by means of DHCP there are a few ways to get to know the actual IP-address of the Checkbox, This is needed in order to communicate with the Checkbox

1. The Network information on the devices detail page



2. Bonjour Service Scanning



A Checkbox is advertising on the bonjour service with its ID in the Name

Using the xxx.local dns method

If you are on the same LAN as the Checkbox, there is another method to find the IP or even access the device at runtime.

Instead of using the ip address, you can also use the hostname of the device. The hostname is comprised out of the Identifier of the Checkbox followed by .local. You can see pinging to this hostname below

```
c:\ping CBX01000003.local
```

```
Pinging CBX01000003.local [192.168.68.107] with 32 bytes of data:
Reply from 192.168.68.107: bytes=32 time=1ms TTL=64
Reply from 192.168.68.107: bytes=32 time<1ms TTL=64
Reply from 192.168.68.107: bytes=32 time<1ms TTL=64
Reply from 192.168.68.107: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.68.107:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms
```

Optional Checkbox Bridge

Bridge Settings

There is an optional possibility for a Checkbox to have a Bridge functionality activated. If this option is activated, once the Checkbox is connected to the internet, you can send all communication to the Bridge instead of the Checkbox directly.
The big benefit of this functionality is that you don't need a lot of configuration to be done on the Pos side, you could just know the Identifier of the Checkbox and add an extra header in the Http requests. The address of the bridge is always the same, whether you want to get to Chebox A, or Checkbox B, you always send the request to the same endpoint with a different header.

The only thing that is important is if you use the Bridge you need 2 additional request headers

> checkbox-client-key: <YOUR CLIENT KEY>
> checkbox-id: <CHECKBOX IDENTIFIER>

the Client key can be found in your developer account in **Tools->Client Keys**

If your account has rights to use the Bridge, you will be able to see the Api Keys for the Client over here.

Bridge Url for the POS endpoint

```
https://api.checkbox.be/bridge/v1/pos/graphql
```

Bridge Url for the AC endpoint

```
https://api.checkbox.be/bridge/v1/ac/graphql
```

Bridge not allowed

If the Checkbox doesn't have the option enabled, you will see this error

```
{
    "errors": [
        {
```

```
            "message": "Checkbox is not allowed to use API",
            "extensions": {
                "code": "NOT_ALLLOWED"
            }
        }
    ]
}
```

POS GraphQl endpoint

The Checkbox is equipped with a GraphQl interface, this is the most important interface to be able to communicate with the Checkbox and sign the events coming from a POS system

Connection Setup and Status

Transport Protocol

The transport protocol used to communicate with the Checkbox is **HTTP**

The choice of not using https i since it would over complicate things for a local network.

There is a way to setup a reverse proxy, where you can include https security if you need to go over the internet for the connection.

GraphQl Endpoint

http://<CHECKBOX ADDRESS>/graphql

CHECKBOX ADDRESS can be one of the following

- Local IP address
- Local hostname **identifier.local**
- Bridge hostname

Authorization

The first thing you need to know is that this GraphQl endpoint is secured with a bearer token. In order for the box to react on the request, you will need to configure and send the Bearer token inside the http header

> Authorization: Bearer <token>

The actual token can be found in several places

1. The **label** that is on the physical Checkbox
2. In the **Checkbox Tools**

Get the status from the POS endpoint

Once you know how to setup the connection for your Checkbox, you can try a simple GraphQl query in order to test if everything is working fine.

```
query Status {
    status(language: NL) {
        initialized
        device {
            fdmId
            fdmSwVersion
            fdmDateTime
            bufferCapacityUsed
        }
    }
}
```

If you receive an answer from the Checkbox it means that everything is ok.

This is an example of a valid response

```
{
    "data": {
        "status": {
            "initialized": true,
            "device": {
                "fdmId": "CBX01000003",
                "fdmSwVersion": "1.0.0",
                "fdmDateTime": "2025-05-31T15:57:34.672Z",
                "bufferCapacityUsed": 0
            }
        }
    }
}
```

POS GraphQl endpoint

Introspection

A GraphQl interface can be queried for what it is able to do, what type of objects it needs, what type it will return and so on, this is called the introspection of the endpoint.

```
{"query":"
    query IntrospectionQuery {
      __schema {

        queryType { name }
        mutationType { name }
        subscriptionType { name }
        types {
          ...FullType
        }
        directives {
          name
          description

          locations
          args {
            ...InputValue
          }
        }
      }
    }

    fragment FullType on __Type {
      kind
      name
      description

      fields(includeDeprecated: true) {
        name
        description
        args {
          ...InputValue
```

```
    }
    type {
      ...TypeRef
    }
    isDeprecated
    deprecationReason
  }
  inputFields {
    ...InputValue
  }
  interfaces {
    ...TypeRef
  }
  enumValues(includeDeprecated: true) {
    name
    description
    isDeprecated
    deprecationReason
  }
  possibleTypes {
    ...TypeRef
  }
}

fragment InputValue on __InputValue {
  name
  description
  type { ...TypeRef }
  defaultValue


}

fragment TypeRef on __Type {
  kind
  name
  ofType {
    kind
    name
    ofType {
```

```
          kind
          name
          ofType {
            kind
            name
            ofType {
              kind
              name
              ofType {
                kind
                name
                ofType {
                  kind
                  name
                  ofType {
                    kind
                    name
                    ofType {
                      kind
                      name
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  ","variables":{}}
```

With this query you are able to fetch all the information needed to continue the development

POS GraphQl endpoint

Pos Mutations

The supported mutations by this endpoint are specified by SPF/FOD, so it is possible to look inside their document to find the details, or you can use the introspection query in order to see what is supported.

If you want to get started you can follow this example to create your first Work In mutation

Work In Mutation

AC GraphQl Endpoint

The Checkbox is equipped with a second GraphQl interface, this is the extra interface to be able to communicate with the Checkbox in order to change settings, read FOD settings, fetch errors, and so on.

Connection Setup

Transport Protocol

The transport protocol used to communicate with the Checkbox is **HTTP.**

GraphQl Endpoint

http://<CHECKBOX ADDRESS>/ac/graphql

CHECKBOX ADDRESS can be one of the following

- Local IP address
- Local hostname **identifier.local**
- Bridge hostname

Authorization

The AC does not need a special token for authorization.

Introspection

A GraphQl interface can be queried for what it is able to do, what type of objects it needs, what type it will return and so on, this is called the introspection of the endpoint.

```
{"query":"
    query IntrospectionQuery {
      __schema {

        queryType { name }
        mutationType { name }
        subscriptionType { name }
        types {
          ...FullType
        }
        directives {
          name
          description

          locations
          args {
            ...InputValue
          }
        }
      }
    }

    fragment FullType on __Type {
      kind
      name
      description

      fields(includeDeprecated: true) {
        name
        description
        args {
          ...InputValue
```

```
    }
    type {
      ...TypeRef
    }
    isDeprecated
    deprecationReason
  }
  inputFields {
    ...InputValue
  }
  interfaces {
    ...TypeRef
  }
  enumValues(includeDeprecated: true) {
    name
    description
    isDeprecated
    deprecationReason
  }
  possibleTypes {
    ...TypeRef
  }
}

fragment InputValue on __InputValue {
  name
  description
  type { ...TypeRef }
  defaultValue


}

fragment TypeRef on __Type {
  kind
  name
  ofType {
    kind
    name
    ofType {
```

```
          kind
          name
          ofType {
            kind
            name
            ofType {
              kind
              name
              ofType {
                kind
                name
                ofType {
                  kind
                  name
                  ofType {
                    kind
                    name
                    ofType {
                      kind
                      name
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  ","variables":{}}
```

With this query you are able to fetch all the information needed to continue the development

Queries

Admin Infos

This query will fetch the most important data and internal settings from the Checkbox set by the Checkbox or set by Fod.

Failed Requests

The failed requests will return an array with requests to the FOD servers that were unable to be delivered and what the rootcause was of the failure.

Fod Buffer Messages

The Checkbox can buffer some messages to send to FOD, while they haven't been sent, they are in the internal buffer, this is the way to read them out.

Fod Buffer Sent Messages

The Checkbox has a buffer where the sent messages are stored, with this query you are able to read that buffer

Mutations

Set Network Config

With this mutation you are able to set the network settings to the Checkbox Device. As well as the Wifi credentials.

Set Ws Init

This mutation allows the Ws Init Url to be set differently, this should only be done when instructed by FOD. Since this is the starting point for their services.

Set Ws Ntp 3

The checkbox comes with preset Ntp servers in order to retrieve the time. There is a third server that can be set in order if the other 2 are unavailable for some reason. That NTP server url can be set with this mutation.

Sync Ntp

This mutation is manually fetching the NTP service in order to sync the time at this moment.

Call Ws Query

This mutation is manually fetching the Ws Query of the FOD server in order to fetch the settings, execute the wanted queries, and so on.

Do Reboot

This mutation will make the Checkbox do a reboot.

Emulator

If you don't have a physical Checkbox available, there is also an emulator available.

Connection Setup

Transport Protocol

The transport protocol used to communicate with the Checkbox is **HTTPS**

GraphQl Endpoint

POS Endpoint

https://api.checkbox.be/checkboxemulator/graphql

AC Endpoint

https://api.checkbox.be/checkboxemulator/ac/graphql

Authorization

The emulator is secured with your API key, you can find this in the Tools backend, if you have Client keys available, this key should be used in the header field **checkbox-client-key.**
In order to select the proper emulator you want to use, you need to provide a custom header with the identifier of the emulator in a header field **checkbox-id**

```
checkbox-client-key: <CLIENT-KEY>
checkbox-id: <EMULATOR IDENTIFIER>
```

Once you have these things in place, you can manipulate the emulator on the Tools platform.

SDK

There is an SDK available. This SDK can be used on 2 levels, the first level is the fact you don't need to have any GraphQl knowledge, it is making an abstraction of the GraphQl interface, and make the mutations and queries available in code to execute.

SDK

.Net Core Sdk

The .Net Core Sdk is available on Nuget,

It is a cross platform SDK, .Net Core is running on different hardware and different OS'es, it does support Windows MacOS, Linux, Android, and iOS.

If you want to get started with the SDK, you can see a tutorial on this location

Getting Started with the SDK

Helping Tools

There are several tools that can help with the development to test requests

Helping Tools

Postman

Postman is a tool that can be used to send requests, configure headers, add authentication, visualize responses and so on.