Getting Started With the IGEL OS App SDK

Building Your Own IGEL OS App

Version 2024-07-19

Table of Contents

1 Change Log for This Document

1.1 Changes Since Version 2023-10-10

- Added chapter "Virtual Channels" Multimedia Plugins and Virtual Channels (see page 35)
- Changed specification for app name in app.json, see Editing Basic Metadata (see page 14)
- Changed format specification for the description, see Editing Basic Metadata (see page 14)
- Made clear which version of IGEL OS must be used for testing; see Installing and Testing Your App (see page 42)
- Updated underlying software versions to SDK 0.9.18 and IGEL OS 12 Developer Edition 12.2.2

1.2 Changes Since Version 2023-08-17

- Made clear that the "SDK" version of IGEL OS must be used, and which version is required; see Prerequisites (see page 10).
- Added info about `public_version`, see Editing Basic Metadata (see page 14).
- Added info about icon formats, see Editing Basic Metadata (see page 14).
- The requirement to specify a minimal base system version was removed because this is done automatically in the build process; see Editing Basic Metadata (see page 14).
- Various edits in the Microsoft Teams PWA example; see Example: Building the Microsoft Teams Progressive Web App (PWA) as an IGEL OS App (see page 72).
- "Signing Your App" has been removed, and the signing command has been integrated into Building Your App (see page 39).
- The chapter "Using the Keys and Configuration Provided in the IGEL OSC for Signing Your Apps" has been removed as it mainly consists of instructions on how to extract the keys from the OSC ZIP file. This is no longer necessary because the keys are integrated into igelpkg with version 0.9.10 or higher.
- The small section "Next Steps" which only mentions the submission of the app to IGEL, has been removed from "Building Your App". When the process has been consolidated, the relevant information will be added.
- The chapter Installing and Testing Your App (see page 42) has been added.
- Added chapter Other Metadata (see page 23). In this issue of "Getting Started With the IGEL OS App SDK", the chapter contains the metadata that is added to the outputted metadata during the build. A complete `app.json` reference is planned for the future.
- The chapter Versioning (see page 12) has been changed for clarity and completeness; the public version number is no longer described as optional.
- Additional information about versioning has been added to the examples:
  - Editing Basic Metadata (see page 14)
  - Example: Building SuperTuxKart as IGEL OS App (see page 52)
  - Example: Building the Microsoft Teams Progressive Web App (PWA) as an IGEL OS App (see page 72)
- The flags `compressed` and `prefer_btrfs` added to Handling Files and Partitions (see page 28)
- The `summary` field which provides the display name for the app has been added where applicable.

- Various smaller improvements and corrections have been made.

1.3 Changes Since Version 2023-10-10

- Change of name: Instead of "SDK" version of IGEL OS, the name "IGEL OS Developer Edition" is used.

2 Introduction

This document provides you with the basic knowledge that is required to create an IGEL OS App that can be submitted to IGEL for review.

> **Blueprint for IGEL Apps**
>
> If you have difficulties developing your app or do not have sufficient Linux knowledge, you can request an app blueprint from IGEL. You can use the app blueprint for testing the basic functionalities. A graphical configuration interface will not be included. Also, metadata, like a logo/icon, or a description, might not be included.
> To request a blueprint, you must provide our development team with all data that is relevant to your app. The contact address is xyz@igel.com.

The process comprises the following steps:

1. Install the IGEL OS App SDK on a Linux system.
2. Build and test your app.
3. Make sure that your app meets the requirements of the review by IGEL.
4. Submit your app for review.

3 Prerequisites

- Machine with Ubuntu Linux 18.04 or higher, or another Debian-based Linux distribution
- The IGEL OS APP SDK, version 0.9.18 or higher
- A device with IGEL OS 12.2.2 Developer Edition or higher (the name of the OSC ISO file contains "SDK").
  If 3D graphics are required, a hardware device might be a good choice; For details, see https://wiki.test.toolchain.igel.kreuzwerker.net/hardware/en/devices-supported-by-igel-os-12-81496425.html. To test if an app can be installed and started, a virtual machine is sufficient.
- A local terminal is configured on the device. For instructions, see Terminals (see page 10)

4 Installing igelpkg

1. Copy the files
   `igel-build-tools-static<version>_amd64.deb`
   and `igelpkg_<version>_amd64.deb` to your development machine.

2. Go to the directory that contains the files and enter the following command:

   ```
   sudo apt install ./igel-build-tools-static<version>_amd64.deb ./igelpkg_<versio
   n>_amd64.deb
   ```

   Example:

   ```
   sudo apt install ./igel-build-tools-static_1.3.7_amd64.deb ./igelpkg_0.9.12_amd
   64.deb
   ```

To show the general help, enter

```
igelpkg --help
```

To get help for a specific module, enter the module's name and then `--help`, for instance:

```
igelpkg build --help
```

5 Versioning

For IGEL OS apps, there are two versioning conventions, a technical version number and a public version number.

5.1 Technical Version Number

The technical version number is defined in `app.json` in the field `version`. It is used for ordering the versions in the UMS and in the App Portal, and for calculating dependencies. It must comply with the Semantic Versioning (SemVer) specifications, see https://semver.org/.

The technical version number consists of two components. The first component is the version number of the app that is re-packaged as an IGEL OS app, according to the vendor's versioning scheme. The second component is the versioning of the IGEL OS app itself.

5.1.1 Prerelease Versions (Release Candidates)

For prerelease versions (release candidates), the following must be added at the end of the vendor-specific version number: `+<BUILD NUMBER>.1.rc.<RELEASE CANDIDATE NUMBER>`. Initially, for a new build, the build number is 0. Example: `23.5.1+0.1.rc.1` is the release candidate for `23.5.1+1`

5.1.2 Released Versions

For released versions, the following must be added at the end of the vendor-specific version number: `+<BUILD NUMBER + 1>`. Example: `23.5.1+1`

When you submit an app to IGEL, always send it as a prerelease version.

5.2 Public Version Number

The public version number is defined in `app.json` in the field `public_version`. It is used in the App Portal, in the UMS, and in the **About** window on the endpoint device.

The public version number consists of two components. The first component is the version number of the app that is re-packaged as an IGEL OS app, according to the vendor's versioning scheme. The second component is the versioning of the IGEL OS app itself; it starts with the keyword `BUILD`.

5.2.1 Prerelease Versions (Release Candidates)

For prerelease versions, the following must be added at the end of the vendor-specific version number: `BUILD <BUILD NUMBER + 1>.0 RC <NUMBER>`. The build number is described under Technical Version Number (see page 12). Example: `23.5.1 BUILD 1.0 RC 1` corresponds to the technical version number `23.5.1+0.1.rc.1`

5.2.2 Released Versions

For release versions, the following must be added at the end of the vendor-specific version number: `BUILD <BUILD NUMBER + 1>.0`. Example: `23.5.1 BUILD 1.0`

See also Versioning Scheme for IGEL OS 12.2 or Higher (see page 12).

6 Creating the Initial Data Structure

The complete initial data structure is created with `igelpkg new -n [app name] -V [version number]`.

6.1 Hello World Example

To have a first look at the data structure, we will create a "Hello World" example. Please note that `igelpkg new` requires a version number. This version number will be included in the directory name of our data structure. If you use Git as the version control system for development, this directory name is arbitrary and can be changed at any time.

```
igelpkg new -n hello_world -V 1.0.0
```

The data structure in `hello_world-1.0.0/` should look as follows:

```
app.json
igel/
+-- debian.json
+-- dirs.json
+-- ignore.json
+-- install.json
+-- thirdparty.json
+-- variable.json
+-- buildCommands.sh
+-- install.sh
data/
+-- config/
| +-- config.param
| +-- ui.json
| +-- translation.json
+-- <icons>
+-- descriptions/
| +-- en
+-- changelogs/
| +-- en
+-- eula/
input/
+-- all/
```

7 Editing Basic Metadata

The metadata for your IGEL OS App is located in `app.json` and in the `data/` directory. This includes icons that will be displayed in the IGEL App Portal and in the IGEL UMS 12. In the following, the mandatory resp. most important directories and metadata files are explained.

7.1 Required Information in app.json

The data described below must be present in `app.json`.

7.1.1 Name

The technical name of the app is defined by the field `name`. It is populated automatically when the `igelpkg new` command is issued. Example: `hello_world`

The name must fulfill the following criteria:

- Alphanumerical characters in upper or lower case are allowed
- Underscores are allowed, but not before the first character
- The name must be between 3 and 42 characters long

The display name, which is shown in the IGEL App Portal, in the UMS, and on the endpoint device, is defined by the field `summary/en`.

7.1.2 Version

The technical version of the app is defined in the field `version`. The field `version` is populated automatically when the `command igelpkg new` is issued.

The display version of the app is defined in the field `public_version`.

When the package is sent to IGEL for review, the version numbers must comply with the versioning scheme for IGEL OS apps. For details, see Versioning (see page 12).

7.1.3 Author

This is the creator of the package. The author is defined in the field `author`. The field must be filled manually.

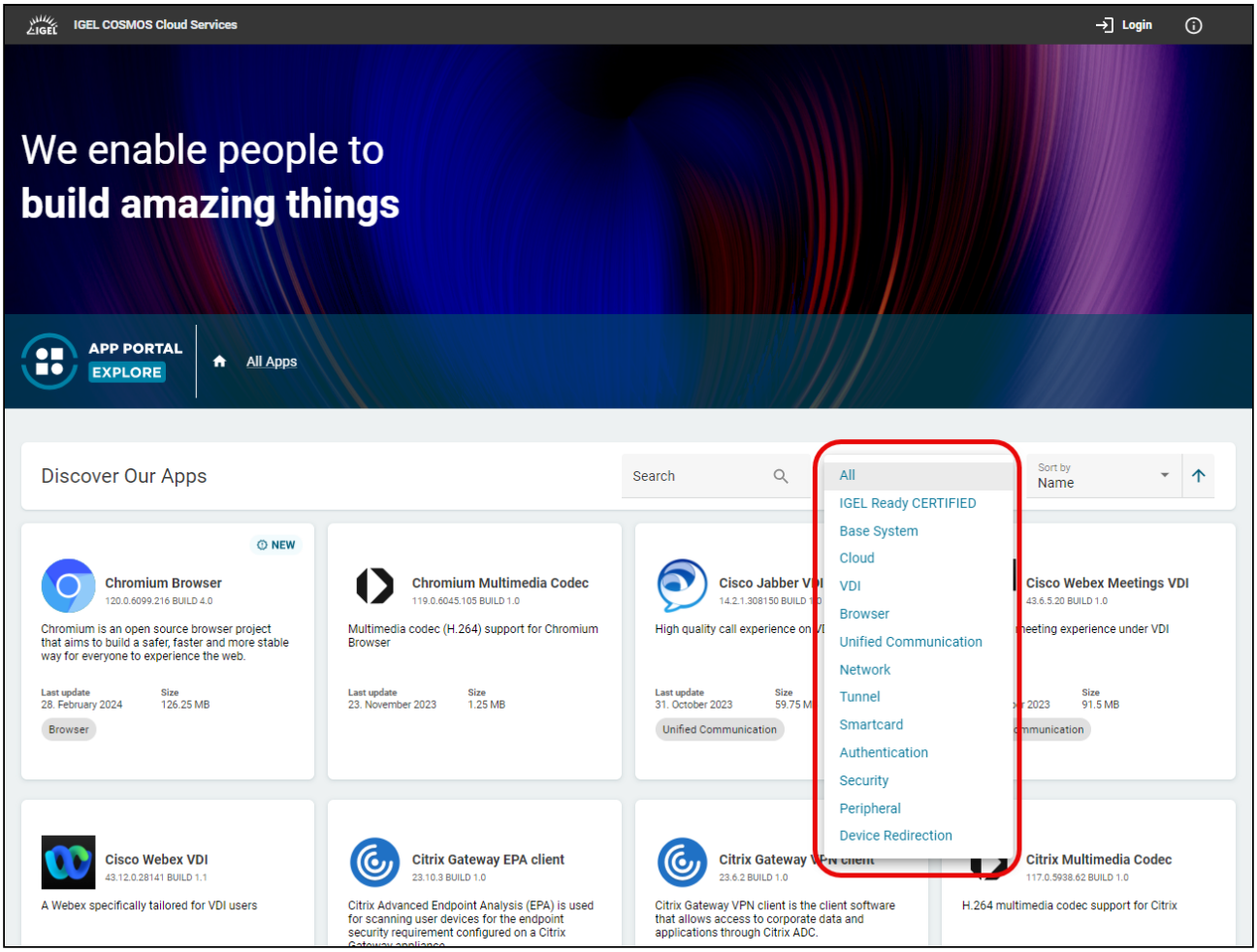
7.1.4 Vendor

This is the vendor of the application that has been re-packaged for IGEL OS by the `author`. The vendor is defined in the field `vendor`. The field must be filled manually.

7.1.5 Categories

The categories are used in the App Portal and the Universal Management Suite (UMS) to make finding the right app for a specific use case easier. Hence, one or more meaningful categories should be provided using the field `categories`. The content is of the type list.

▶ To get a current list of available categories, go to https://app.igel.com/ and open the **Categories** menu.



7.1.6 Minimal Hello World Example

Empty fields can be removed safely.

```
{
  "name": "hello_world",
  "version": "1.0.0+0.1.rc.1",
  "public_version": "1.0.0 BUILD 1.0 RC 1",
  "release": "",
  "summary": {
    "en": "hello_world"
  },
  "icons": {
    "app": "hello_world_portal.svg",
    "monochrome": "hello_world_monochrome.svg"
  },
  "categories": [
    "misc"
  ],
  "author": "IGEL",
  "vendor": "Acme",
  "provides": [
    {
      "name": "",
      "version": ""
    }
  ],
  "conflicts": [
    {
      "name": "",
      "version": ""
    }
  ],
  "requires": [
    {
      "name": "",
      "version": ""
    }
  ],
  "suggests": [
    {
      "name": "",
      "version": ""
    }
  ]
}
```

7.2 Icons

The icons for display in the IGEL App Portal and in the IGEL UMS 12 are stored in the `data` directory, in our example, `hello_world-1.0.0/data/`

Two icons can be defined, although currently only one of them is supported:

- `"app"`: Icon of any color that will be displayed in the App Portal and in the UMS
- `"monochrome"`: Icon that will be used in future versions when theming is implemented

The icon is defined in `app.json` as follows (the file names are arbitrary):

```
"icons": {
  "app": "<app_name>_portal.svg",
  "monochrome": "<app_name>_monochrome.svg"
},
```

Example:

```
"icons": {
  "app": "hello_world_portal.svg",
  "monochrome": "hello_world_monochrome.svg"
},
```

7.2.1 Icon Format - General

The `app` icon and the `monochrome` icon must have the following properties:

- The icons must be in SVG format.
- To make sure that the icons can be displayed by all browsers, only those SVG features that are marked as "SVG Basic" should be used. Also, filters should not be used.
- Animations should be avoided.
- Isolated (transparent background)
- Square shape
- The `<svg>` element must contain the `viewBox` attribute
- The `viewBox` must be aligned to the image border; there must be no gaps

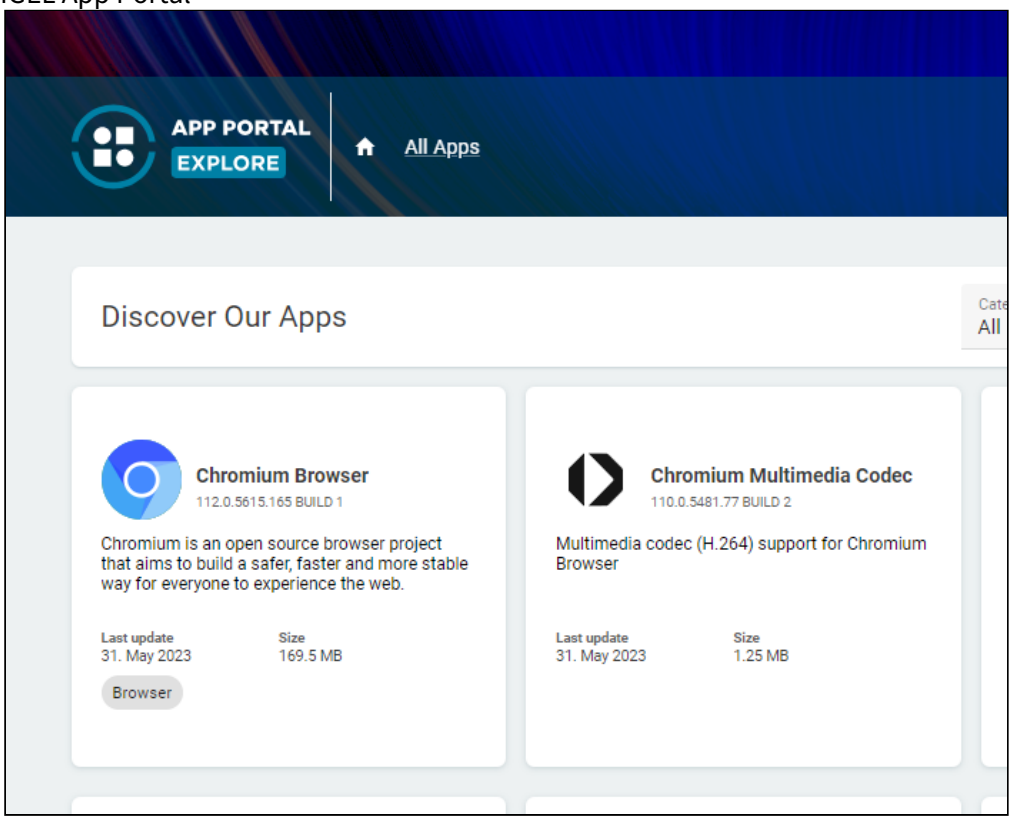
7.2.2 Icon Format - Monochrome Icon

- The icon is monochrome
- Only shapes are used, no strokes
- Only "fill color" is used, no "stroke color"

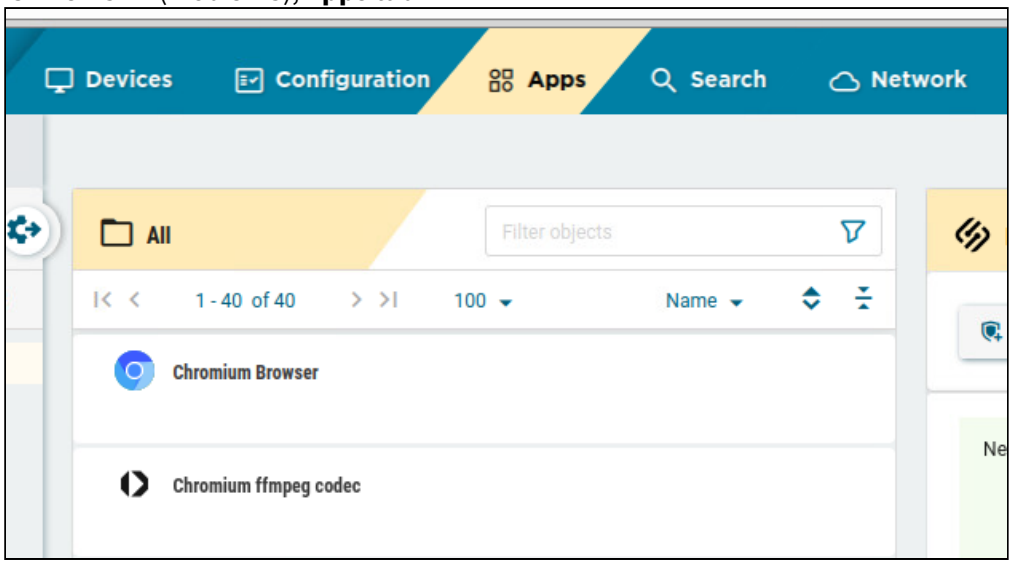
7.2.3 Icon Display

The icon is displayed at the following locations:

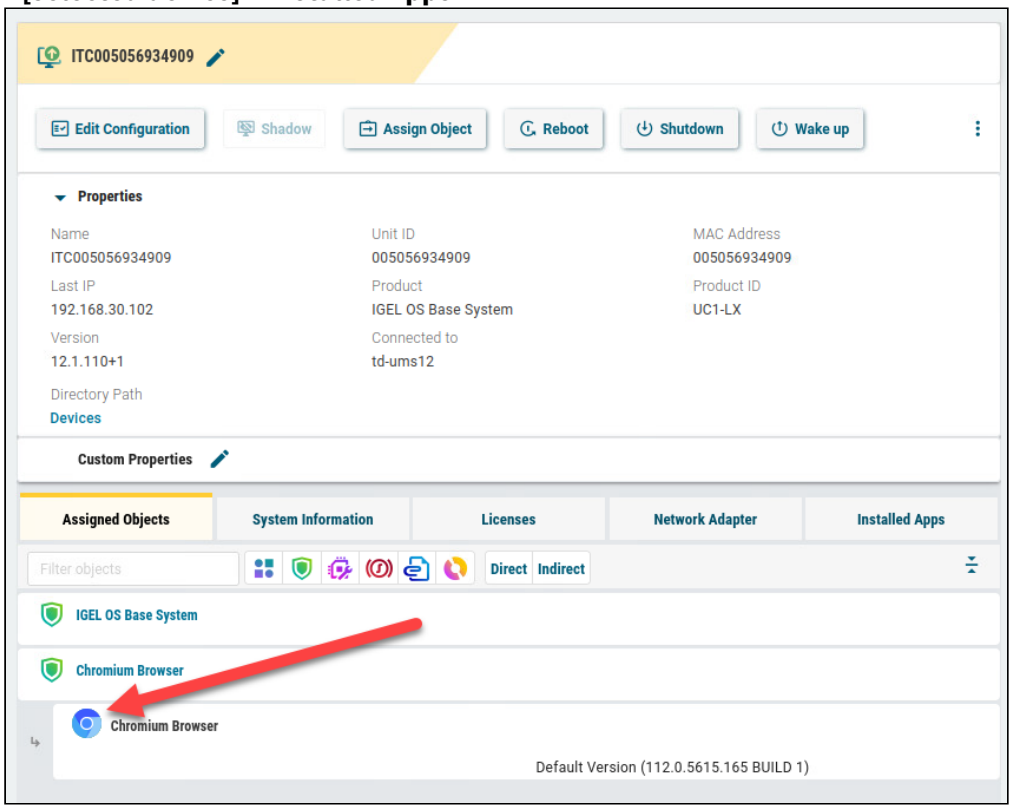
- IGEL App Portal

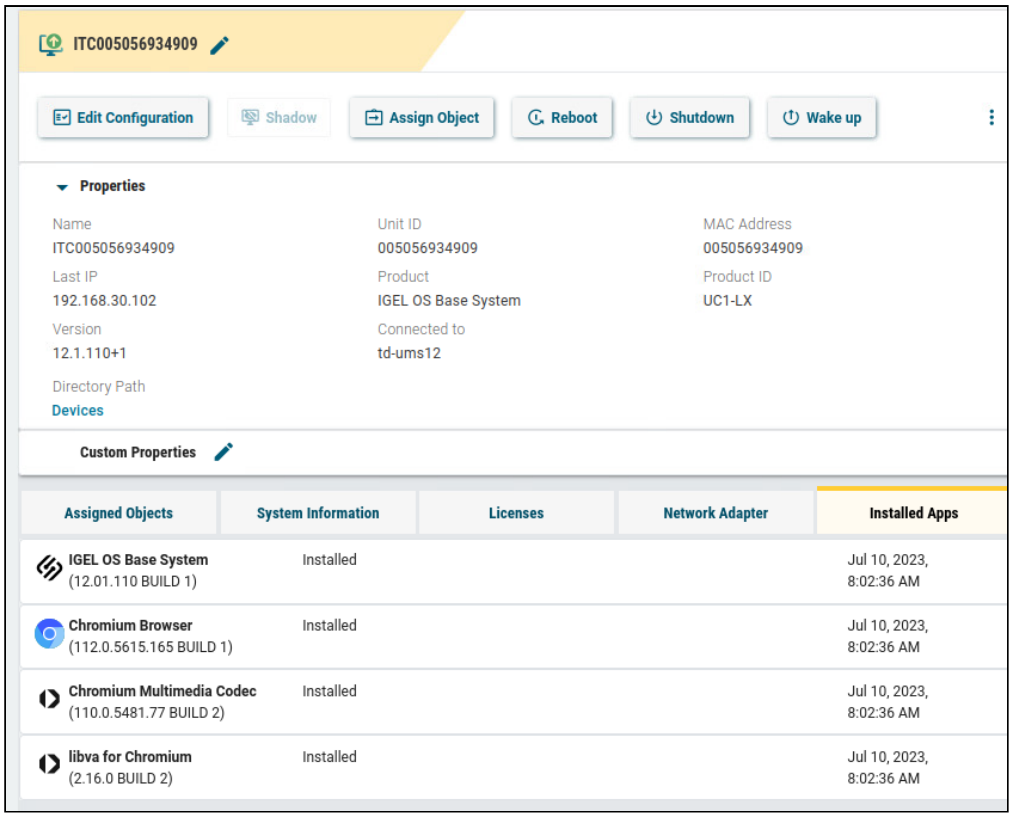


- IGEL UMS 12 (Web UMS), **Apps** tab



- UMS 12 (Web UMS), under **Devices > [selected device] > Assigned Objects**, and under **Devices > [selected device] > Installed Apps**





7.3 Description to Be Displayed in the IGEL App Portal

The file `data/descriptions/en` contains a long description and a short description of the app in the English language. Both descriptions will be displayed in the IGEL App Portal when the app is publicly available.

The description consists of a short description and a long description. The format is further specified as follows:

1. Short description (plain text)
2. Empty line
3. Long description (plain text or Markdown)

Example:

```
The Citrix Workspace app allows users to access virtual desktops and hosted
applications delivered by XenDesktop and XenApp.

# Citrix Workspace app
Citrix Workspace app gives users instant access to all their SaaS and web
applications, their files and mobile, and their virtual apps and desktops from an
easy-to-use, all-in-one interface. The Citrix Workspace app provides users with a
single point of access to all workspace services. Users get seamless and secure
access to all the applications they need to stay productive, including features such
as embedded browsing and single sign-on.
```

7.3.1 Localization of the Description

If your app is localized, adding description files in the appropriate languages is recommended. For instance, the filename for the German description would be `de`.

7.4 EULA

When your app requires an EULA, store a file named with the appropriate language code in `data/eula`. The EULA is displayed in the IGEL App Portal. Examples: `data/eula/en` for the English EULA file, `data/eula/de` for the German EULA file, and so on.

7.5 Changelog (Release Notes)

It is recommended to add a changelog file to your app package to inform your users about the changes between subsequent versions. It will be displayed in the GEL App Portal when the app is publicly available.

The file path is `data/changelogs/en` for the English text.

During the build format, the changelog is rendered as HTML. The authoring format is specified in the following.

7.5.1 Basic Structure

Changelogs for IGEL OS Apps consist of the following elements:

- Main statements that may contain sub-statements and sub-sub-statement
- Setup paths and registry keys of the GUI elements that control app parameters; these are formatted as standardized tables

7.5.2 Basic Writing Rules

- Always end sentences with a full stop.
- Every main statement is a list entry. To create a list entry, use `*`

7.5.3 Nested Lists

Nested lists are formatted as follows:

```
* Main statement
   - sub-statement
      -- sub-sub-statement
```

Please note that the indentation is added for readability; it will be ignored during the build process.

7.5.4 Formatting

- To mark a piece of text as bold, encase it with *
  Example:

```
Normal text *bold text* normal text
```

- To create a code field, encase the text with `
  Example:

```
Enter `reboot` to restart your device.
```

- To create a link to a URL, use the following syntax:

```
[> DISPLAY_TEXT](URL)
```

  Example:

```
[> Devices Supported by IGEL OS 12](https://
wiki.test.toolchain.igel.kreuzwerker.net/os12-supported-hardware)
```

7.5.5 Tables

7.5.6 Localization of the Changelog

If your app is localized, adding changelog files in the appropriate languages is recommended. For instance, the filename for the German description would be de.

8 Other Metadata

This chapter describes the metadata that can be present in `app.json`.

> The data presented here is not complete yet; it will be completed in a future version of this document.

8.1 Keys Added to the Output app.json During Build/Deployment

The following table shows which keys are added to the `app.json` during the build process. Those keys will be available in the `app.json` that is packaged in the output `.ipkg` file. The source `app.json` is not modified by the build process.

Name	Type	Value/Meaning
`architectures`	Array of strings	Included architectures
`build_date`	Integer	Build date as UNIX timestamp
`meta_data_version`	Integer	Version of the metadata
`used_debian_packages`	Array of dictionaries	Debian packages used to build this app
`config_schema_version`	Integer	If not set, the default value of 1 will be used.
`author`	String	If not set, the default value from the global configuration will be used which is "IGEL Technology GmbH" inside IGEL
`apptype`	Integer	The type the app: 0 = firmware (base system), 1 = IGEL app 2 = Third party app
`description`	Dictionary	The long description within each file in the directory `data/descriptions/` will be an entry in this dictionary.
`short_description`	Dictionary	The short description within each file in the directory `data/descriptions/` will be an entry in this dictionary.
`licenses`	Array of strings	List of included licenses
`partitions`	Dictionary	List of partitions per architecture

Name	Type	Value/Meaning
`public_version`	String	An alternative version number of the app used for display in the IGEL App Portal and the UMS Web App. If not set, the version field will be copied over. Semantic versioning is not necessary for this version number.

9 Integrating Debian Packages, Tarballs, and Other Archive Types

To integrate archives into your IGEL OS App, you define the sources in `igel/thirdparty.json`.

With `igel/install.json`, you can set file permissions, select the files to be integrated, and specify whether binary files should be stripped of comments and other non-essential content.

9.1 Defining the Sources in igel/thirdparty.json

9.1.1 Required Fields

- `url`: URL to the archive. The following schemas are supported:
  - `file`
  - `http`
  - `https`
  - `ftp`
- `licenses<n>.name`: Name of the license. If possible, use the name according to https://spdx.org/.

9.1.2 Optional Fields

- `dest`: The destination folder for storing the extracted data, relative to the directory `igelpkg.tmp/`. If you have several 3rd party input archives, it makes sense to use subfolders. If two 3rd party archives both have files that share the same name, using subfolders is required; otherwise, an overwrite error would occur. If `dest` is not specified, the files are stored directly under `igelpkg.tmp/`.
- `extract`: If false then files are not extracted but just copied directly into the app. The default is `true`.
- `licenses[n].text`: If `licenses[n].name` is not available at https://spdx.org/, `licenses[n].text` or `licenses[n].file` must be provided (see next item).
- `licenses[n].file`: A file containing the license text may be specified instead of licenses[n].text.

9.1.3 Example

```
[
   {
      "url": "file:///mnt/igelfiles/software/hello_world.tar.gz",
      "dest": "",
      "extract": true,
      "licenses": [
         {
            "name": "LGPL-3.0+",
            "text": "GNU Lesser General Public License v3.0 or later"
         }
      ]
   }
]
```

9.2 Refining Your Selection in igel/install.json

9.2.1 Required Fields

- `source`: Defines which files from the archive are to be integrated into the target package. You can use regular expressions; everything that is matched will be integrated. Example snippet: `"source": ".*"` will integrate everything in the archive.

9.2.2 Optional Fields

- `destination`: The file destination. If wildcards are used, the destination is a folder. If a single file is used, `destination` is the new filename. All parent folders will be created. If nothing is specified here, the filename and folders are the same as in the source.
- `owner`: Owner of the file as `uid:gid`. The default is `0:0`, which means that the user is `root`, and the group is `root`. If you need a special user, please ask your IGEL contact for a uid.
- `permissions`: Permissions to be set for files. By default, the permissions remain unchanged.
- `excludes`: A list of regular expressions of filenames to be excluded. These excluded files are ignored so no need to define them also in ignore.json.

  The files excluded here are ignored, so there is no need to define them also in `ignore.json`.

- `strip`: If set to true, symbols and other data will be stripped from binary files to reduce the file size. The default is `true`.

9.2.3 Example

```
[
   {
      "source": ".*",
      "excludes": [
      ""
      ],
     "destination": "/usr/local",
     "owner": "0:0",
     "permissions": "644",
     "strip": true
   }
]
```

9.3 Archive Types That Are Supported Out-Of-The-Box

The following archive types are supported by `igelpkg` without any configuration changes:

- tar.*
- tgz
- gz
- zip
- deb
- bz2

10 Handling Files and Partitions

10.1 Defining which Directories Will Be Created in the Output Path

With `igel/dirs.json`, you can define which directories will be created in the output path. This may also include read/write partitions that can be used for user data.

10.1.1 Required Fields

- `path`: Path of the directory to be created

10.1.2 Optional Fields

- `owner`: Owner of the file as `uid:gid`. The default is 0:0, that is, the user `root` and the group `root`.
- `permissions`: Permissions to be set for files.
- `persistent`: If true, the folder will be linked to the app's read/write partition, and the data will be persistent. The app needs to have a read/write partition.

10.1.3 Example

In the following example, the directories `opt/Citrix/ICAClient/pkginf` and `etc/igel/configuration.d/app/cwa` are created as non-persistent, with `root` as the owner and the `root` group is the owning group, whereas the directory `/userhome/.ICAClient_persistent` is created as persistent with `user` as the owner and the `users` group as the owning group. Please note that for persistent data, you must define a read/write partition; see Defining a Read/Write Partition for Persistent Data (see page 28).

```
[
   {
       "path": "opt/Citrix/ICAClient/pkginf"
   },
   {
       "path": "etc/igel/configuration.d/app/cwa"
   },
   {
       "path": "/userhome/.ICAClient_persistent",
       "persistent": true,
       "owner": "777:100"
   }
]
```

10.2 Defining a Read/Write Partition for Persistent Data

If you want your app to store user data persistently, you must define a read/write partition in your `app.json`.

10.2.1 Partition Size

The partition size can be defined with the key `size`. The following values are possible:

- `small`
- `medium`
- `large`
- Exact size in KiB

When `small`, `medium`, or `large` are defined, the size is calculated depending on the device's storage size.

10.2.2 Filesystem/Compression

You can define the filesystem for the read/write partition. This is done with the following `flags`:

- `compressed`: If this flag is present, the size of the partition and the presence of the flag `prefer_btrfs` determine which filesystem will be used for the partition. If it is not present, EXT4 will be used as the filesystem.
- `prefer_btrfs`: If both this flag and the flag `compressed` are present, and the partition size is 128 MB or greater, BTRFS will be used as the filesystem. If this flag is not present or the partition size is smaller than 128 MB, and the `compressed` flag is present, NTFS will be used as the filesystem.

Filesystem	Size	Code Snippet
BTRFS	>= 128 MB	`"rw_partition": { "size": "medium", "flags": [ "compressed", "prefer_btrfs" ] }`
NTFS	< 128 MB	`"rw_partition": { "size": "medium", "flags": [ "compressed", "prefer_btrfs" ] }`
NTFS	>= 128 MB	`"rw_partition": { "size": "medium", "flags": [ "compressed" ] }`

Filesystem	Size	Code Snippet
EXT4	Any size	`"rw_partition": { "size": "medium" }`

10.2.3 Example

The following snippets are derived from the MS-Teams example; see Example: Building the Microsoft Teams Progressive Web App (PWA) as an IGEL OS App (see page 72).

1. Add the following code to your `app.json`:

```
"rw_partition": {
   "size": "large",
   "flags": [
      "compressed",
      "prefer_btrfs"
    ]
}
```

2. To define the storage path, set the directory's owner to `user`, and make the storage persistent, edit `igel/dirs.json` as follows:

```
[
  {
    "path": "/userhome/.config/ms-teams",
    "owner": "777:100",
    "permissions": "",
    "persistent": true
  }
]
```

11 Partition Layout on the Device

When the app has been installed on the device, the app partitions are mounted at `/services/<app_name>`, e.g. `/services/hello_world`

The directory and file structure of the resulting igelfs partition is mostly defined by the content of `/igel/install.json`. All files and directories mentioned there are available on the igelfs partition according to their destination path. Some files and directories are created by `igelpkg`. These will be explained below.

11.1 /etc/setup.def.d/

The content of `data/config/` is processed and the files are written to the igelfs partition at `/etc/setup.def.d/`. If all input files are present, the following files are written to the output:

- `<app_name>/.setup.def`
- `<app_name>/.setup.def.ui.json`
- `<app_name>/.setup.def.translation`

11.2 .licenses/

This directory contains one file per license that is included in the app. An app can contain parts of multiple source projects and therefore could include several licenses. The name of the file is the SPDX license identifier, if there is one. Otherwise, the name was explicitly specified by the app creator
and the content is the license text.

11.3 .scripts/

This directory contains the scripts and data that are needed to install the apps.

11.4 .scripts/install.sh

This script is executed during the installation of the app. The script was created by igelpkg based on `/igel/install.sh` and some automatically detected information.

11.5 ./scripts/cache.json

This file contains all paths that will be linked or copied to the `/cache` partition. Those paths will be linked from `/cache` to `/` at every boot, which makes them available in the root filesystem. The file is generated by igelpkg.

11.6 ./scripts/post_mount.sh

If any directories are marked as persistent in `igel/dirs.json`, this script is created by igelpkg. When the app is installed, the script creates the required directories on the read/write partition.

11.7 .icons/

This directory contains all icons referenced in `app.json`. These icons are displayed in the UMS (Web App) when the app is registered by the device.

11.8 .config/

This directory contains the configuration data for the app.

11.9 .config/rwpartition.json

This file is included when the app needs its own read/write partition for storing session data. See also Handling Files and Partitions (see page 28).

11.10 .eula/

This directory contains EULAs (End User License Agreements) in English (filename: `en`) or other languages, if required. The UMS defines if the EULA needs to be accepted locally by the user or can be accepted company-wide via the UMS.

12 Handling Users

You can add users and groups to be used by your app. This is done by creating the directory `/etc/igel-userdb/` and adding user and group definitions in JSON format according to the following specifications:

- https://systemd.io/USER_RECORD/
- https://systemd.io/GROUP_RECORD/
- https://www.freedesktop.org/software/systemd/man/nss-systemd.html

> ⚠ If you are planning to add users or groups, make sure to contact IGEL beforehand. This is necessary to avoid conflicts with other apps that might otherwise use the same user IDs (`uid`) or group IDs (`gid`) as your app.

The following information from the user and group definition files is pushed to the traditional files `/etc/passwd`, `/etc/shadow`, and `/etc/group`:

- `userName`
- `uid`
- `privileged.hashedPassword[0]`
- `gid`
- `realName`
- `homeDirectory`
- `shell`
- `memberOf`
- `groupName`
- `members`

Any other entries are ignored.

12.1 Simple Example

To create a user and a group with a minimum of data, you create two JSON files under `input/all/etc/igel-userdb/`

1. In your app directory, add the directory structure `input/all/etc/igel-userdb/` (e.g. `hello_world-1.0.0/input/all/etc/igel-userdb/`)
2. Create a user definition file named `myuser.user`, for instance, with the following structure:

```
{
    "userName" : "myuser",
    "uid" : 10001,
    "gid" : 10001
}
```

3. Create a user definition file named `mygroup.group`, for instance, with the following structure:

```
{
    "groupName" : "mygroup"
    "gid" : 10001
}
```

13 Multimedia Plugins and Virtual Channels

This chapter is relevant if the app you want to create is intended to integrate a multimedia plugin into a VDI app by adding a virtual channel. An example of a multimedia plugin app Is the **Zoom Media Plugins for VDI** (https://app.igel.com/#/api/zoomvdi) that integrates with various VDI apps, e.g. the **Citrix Workspace App** (https://app.igel.com/#/api/cwa).

To integrate a multimedia plugin into a VDI app, we create a config file in JSON that contains information for the VDI app. This file must be placed under `<PLUGIN_APP_DIRECTORY>/input/all/etc/igel/configuration.d/app/<VDI_APP_NAME>/<CONFIG_FILE_NAME>.json`, where `VDI_APP_NAME` is the VDI app to which the plugin is to be added. `VDI_APP_NAME` can be one of the following:

- `cpc` = IGEL Windows 365
- `avd` = IGEL Azure Virtual Desktop
- `rdp` = IGEL Remote Desktop
- `horizon` = VMware Horizon Client
- `cwa` = Citrix Workspace App

As for the config file, a good file name would be `<PLUGIN_APP_NAME>_config.json`.

If for instance, the plugin app is **Zoom Media Plugins for VDI** and the VDI app is **IGEL Azure Virtual Desktop**, a valid file path might look as follows: `zoomvdi/input/all/etc/igel/configuration.d/app/avd/zoomvdi_config.json`

There are two JSON schemes for adding virtual channels: a simple virtual channel scheme for AVD, RDP, Win365, and Horizon, and a special scheme for Citrix sessions.

13.1 Simple Virtual Channel Scheme

If no change in `module.ini` is required, the scheme looks like this:

```
{
  "enabled": true,
  "type": "[svc|dvc]",
  "plugin_path": "[link target]",
  "plugin_name": "[file name of the link source]",
  "plugin_source": "[directory path of the link source]"
}
```

Explanations:

- `"enabled"`: If set to `true`, the virtual channel is enabled. If set to `false`, the virtual channel is disabled without removing the file.
- `"type"`: This field is only relevant for the DVI/Cloud apps IGEL Windows 365, IGEL Azure Virtual Desktop, and IGEL Remote Desktop. It defines whether a static or a dynamic virtual channel is created. The plugins by IGEL, as well as freerdp, for instance, use dynamic virtual channels. For further information, see the following articles by Microsoft: Static Virtual Channels[1] and Dynamic

1 https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-rdpbcgr/343e4888-4c48-4054-b0e3-4e0762d1993c

Virtual Channels[2].
  - `"svc"`: Static virtual channel
  - `"dvc"`: Dynamic virtual channel (default)
- `"plugin_path"`: The path to which the library is linked, i.e. the link target
- `"plugin_name"`: The name of the library that is to be linked
- `"plugin_source"`: The path where the library is stored

13.2 Virtual Channel Scheme for Citrix Sessions

The scheme for adding a virtual channel to a Citrix session looks like this:

```
{
    "enabled": true,
    "module": {
        "virtualdriver": {
            "[section / virtual channel name]": {
                "DriverName": "[file path to library]"
            }
        }
    }
}
```

Explanations:

- `"enabled"`: If set to `true`, the virtual channel is enabled. If set to `false`, the virtual channel is disabled without removing the file.
- `"module"`: Stands for the Citrix configuration file module.ini
- `"virtualdriver"`: Tells the configuration program that special routines must be executed for the parameter
- `"[section / virtual channel name]"`: Name of the virtual driver and new section created in `module.ini`
- `"[file path to library]"`: Full path to the shared library, including the file name. The library is automatically linked into the ICAClient directory and entered as `DriverName` in the newly created section in `module.ini`.

13.3 Examples

13.3.1 Zoom VDI Plugin for IGEL Azure Virtual Desktop (AVD)

File name: `zoomvdi/input/all/etc/igel/configuration.d/app/avd/zoomvdi_config.json`

```
{
```

2 https://learn.microsoft.com/en-us/windows/win32/termserv/dynamic-virtual-channels

```
  "enabled": true,
  "type": "svc",
  "plugin_path": "/etc/avd/",
  "plugin_name": "libZoomMediaWVD.so",
  "plugin_source": "/usr/lib/x86_64-linux-gnu/freerdp/"
}
```

13.3.2 Zoom VDI Plugin for VMware Horizon

File name: `zoomvdi/input/all/etc/igel/configuration.d/app/horizon/zoomvdi_config.json`

```
{
  "enabled" : true,
  "plugin_path": "/etc/vmware/",
  "plugin_name": "libZoomMediaVmware.so",
  "plugin_source": "/usr/lib/vmware/view/vdpService/"
}
```

13.3.3 Example: Virtual Channel for Adding the Zoom VDI Plugin to Citrix

File name: `zoomvdi/input/all/etc/igel/configuration.d/app/cwa/zoomvdi_config.json`

```
{
    "enabled": true,
    "module": {
        "virtualdriver": {
            "ZoomMedia": {
                "DriverName": "/services/zoomvdi/opt/Citrix/ICAClient/ZoomMedia.so"
            }
        }
    }
}
```

14 Creating a Service That Is Controlled By systemd

If you need to start a service at boot time, `systemd` can be utilized. For this purpose, a unit file needs to be deployed to the default `systemd` paths. The service can be enabled via an install script of the app. All this is described below.

14.1 Unit Configuration File

For detailed information on unit configuration, see https://www.freedesktop.org/software/systemd/man/systemd.unit.html and https://www.freedesktop.org/software/systemd/man/latest/systemd.service.html

1. Create the file according to the following example.

```
[Unit]
Description=<app_name> configuration

[Service]
Type=oneshot
RemainAfterExit=Yes
ExecStart=/services/<app_name>/<path_to_binary>
ExecStop=/services/<app_name>/<path_to_binary>
```

2. Save the file to your app's data structure under one of the following file paths, depending on the user under which the service is to run:
   - root: `input/all/etc/systemd/system/<name>.service`
   - user: `input/all/etc/systemd/user/<name>.service`

14.2 Install Script

▶ In your app's data structure, edit `igel/install.sh` depending on the user under which the service is to run.

```
#!/bin/bash
# For root services:
enable_system_service <name>.service
# For user services:
enable_system_user_service <name>.service
```

15 Building Your App

15.1 Checking the Metadata

Before you build your app, make sure that your metadata is complete and correct.

As a minimum requirement for a valid app, the fields `author` and `vendor` must be filled out.

15.1.1 Hello World Example

1. Enter your app's directory and open `app.js` with your favorite editor.

```
cd hello_world-1.0.0
vi app.json
```

2. Fill in the necessary fields.

```
{
  "author": "Ike Igel",
  "categories": [
    "Unified Communication"
  ],
  "conflicts": [
    {
      "name": "",
      "version": ""
    }
  ],
  "icons": {
    "app": "app.svg",
    "monochrome": "monochrome.svg"
  },
  "name": "hello_world",
  "provides": [
    {
      "name": "",
      "version": ""
    }
  ],
  "public_version": "1.0.0 BUILD 1.0 RC 1",
  "release": "",
  "requires": [
    {
      "name": "",
      "version": ""
    }
  ],
  "suggests": [
    {
      "name": "",
      "version": ""
    }
  ],
  "summary": {
    "en": "Hello World"
  },
  "vendor": "IGEL Technology",
  "version": "1.0.0+0.1.rc.1"
}
```

3. Save `app.json`

15.2 Building and Signing the App

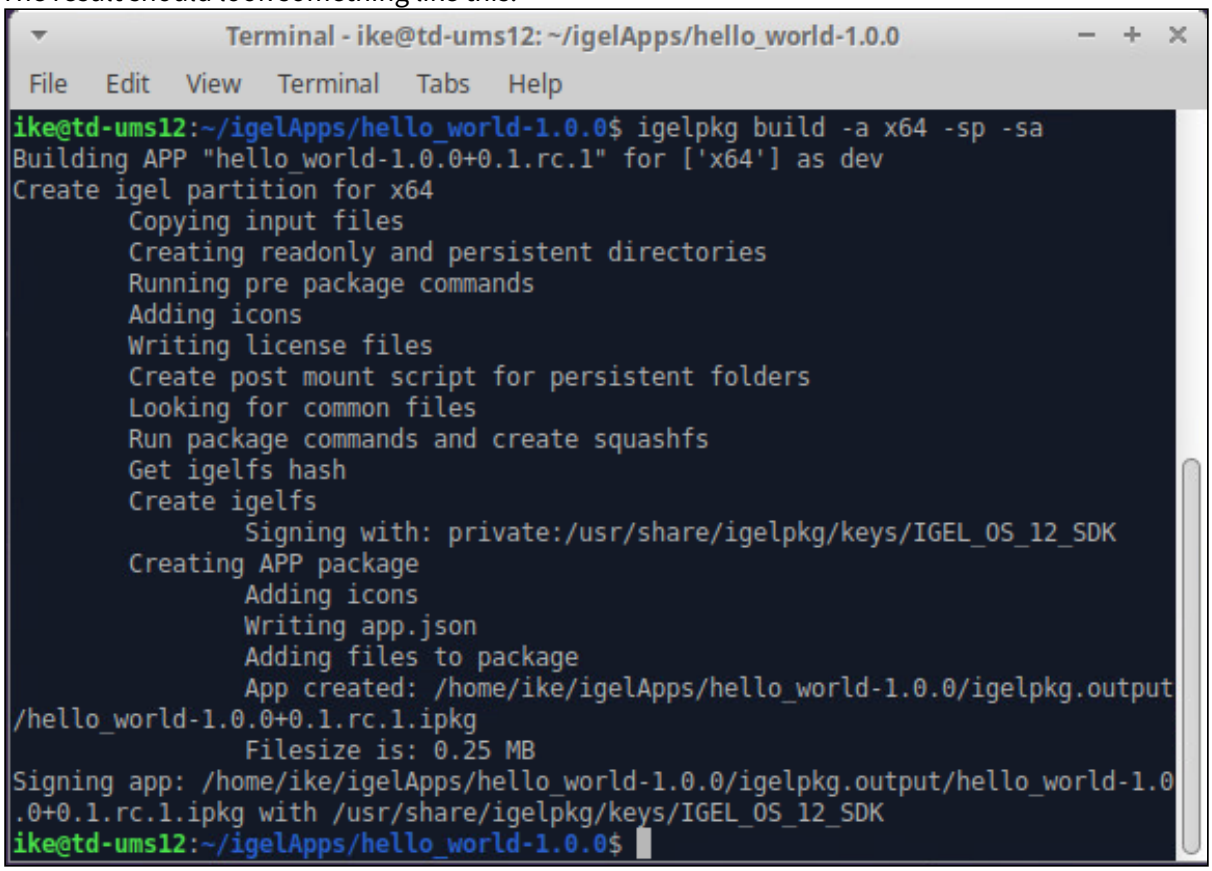
We will build and sign our app in one go by adding the options `-sp` (sign publisher) and `-sa` (sign author).

15.2.1 Hello World Example

- In your app's directory, enter:

```
igelpkg build -a x64 -sp -sa
```

The result should look something like this:



Debugging

For debugging purposes, you can add the `-k` option and the `-l` option :
`igelpkg build -kl -a x64`
With the `-k` option, the temporary files will not be deleted during the build process. The temporary files can be found in the following directories:

- `igelpkg.out/`
- `igelpkg.tmp/`

With the `-l` option, the log file `igelpkg.log` is written to the app's data structure, e.g. `hello_world-1.0.0/igelpkg.log`

16 Installing and Testing Your App

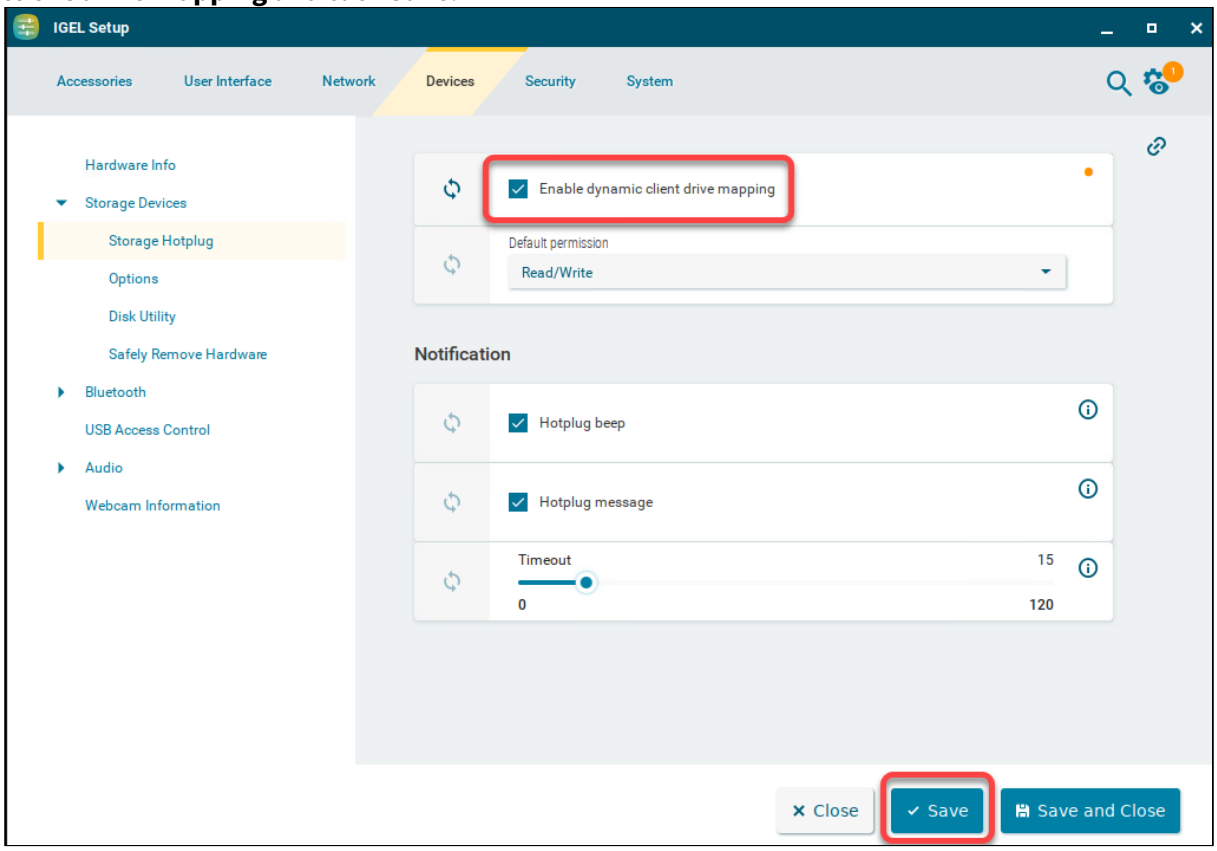
Before you can send your app to IGEL, you must test it on an endpoint device. It is recommended to install the app from a local USB drive or a network drive (NFS or SMB/Windows).

16.1 Making Sure You Have the Right Version of IGEL OS on Your Device

▶ If you have not done so already, install IGEL OS 12.01.120 Developer Edition or higher on your test device. You can recognize this version by the "SDK" in the file name.

16.2 Deploying Your App from a USB Flash Drive

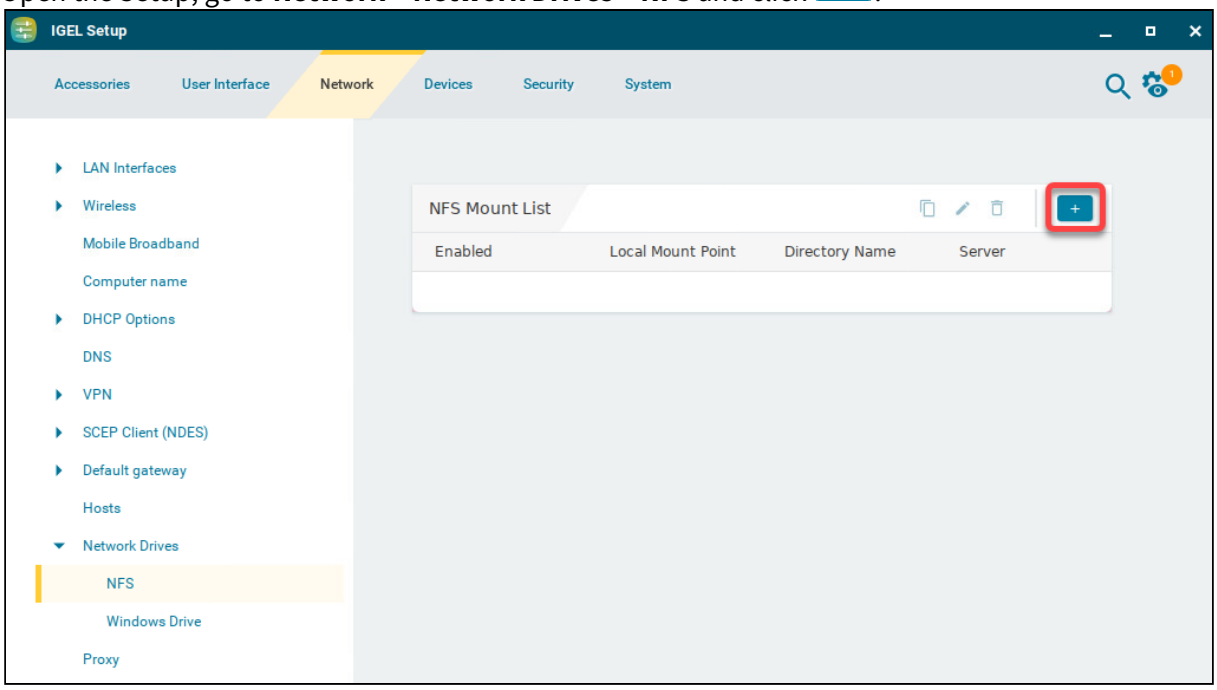
1. Open the Setup, go to **Devices > Storage Devices > Storage Hotplug**, activate **Enable dynamic client drive mapping** and click **Save**.



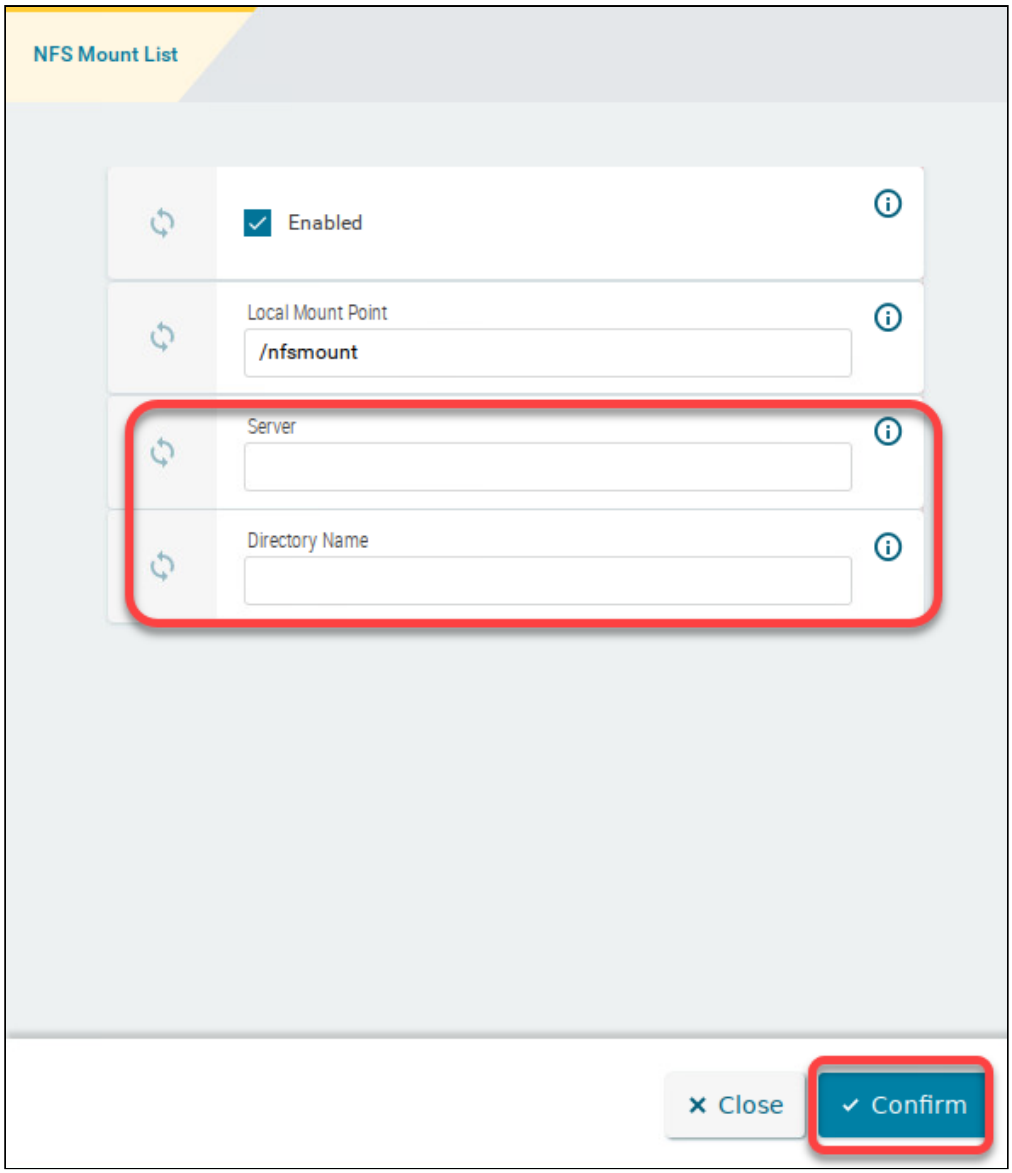
2. Plug the USB flash drive into the device.
   The USB flash drive is mounted under `/userhome/media`

16.3 Deploying Your App from an NFS Drive

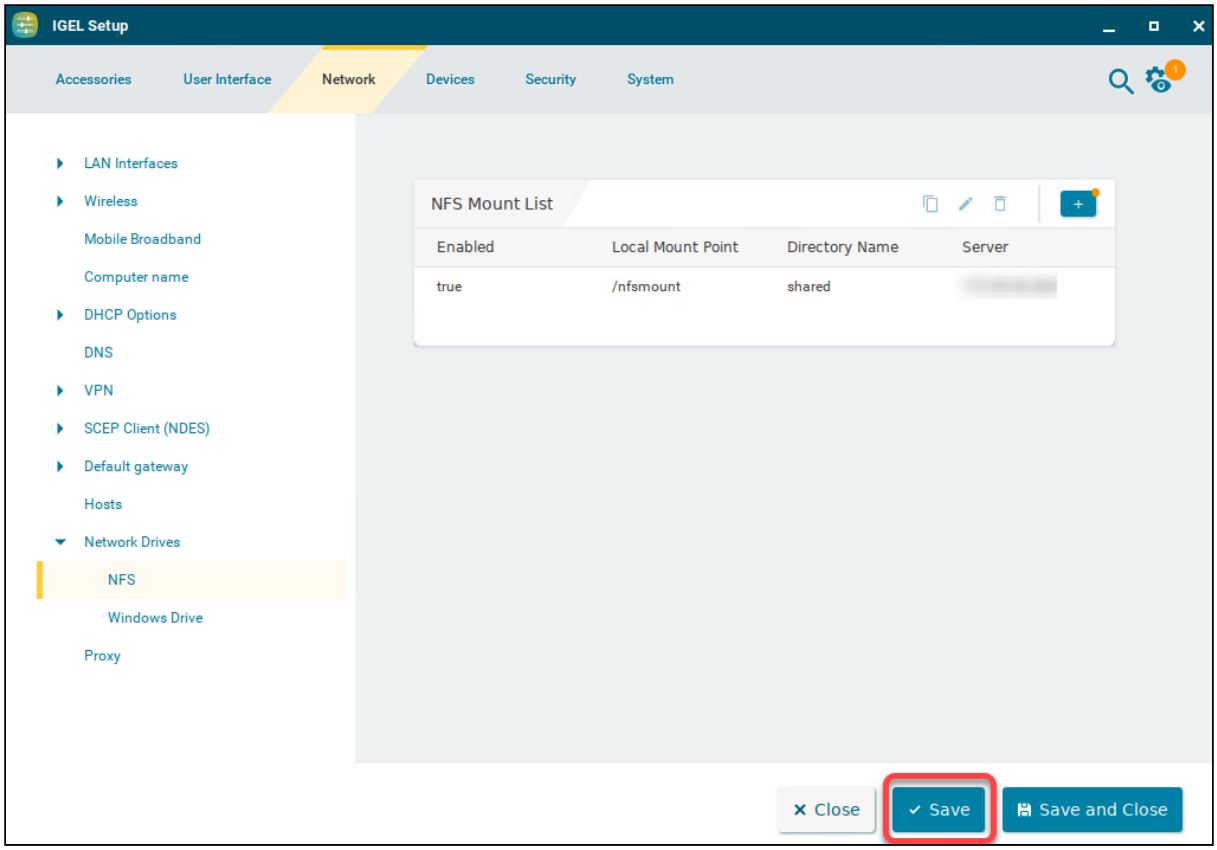
1. Open the Setup, go to **Network > Network Drives > NFS** and click +.



2. Enter the address of the **Server** and the **Directory Name** of the exported directory, and click **Confirm**.


NFS Mount List
Enabled
Local Mount Point
/nfsmount
Server
Directory Name
Close
Confirm

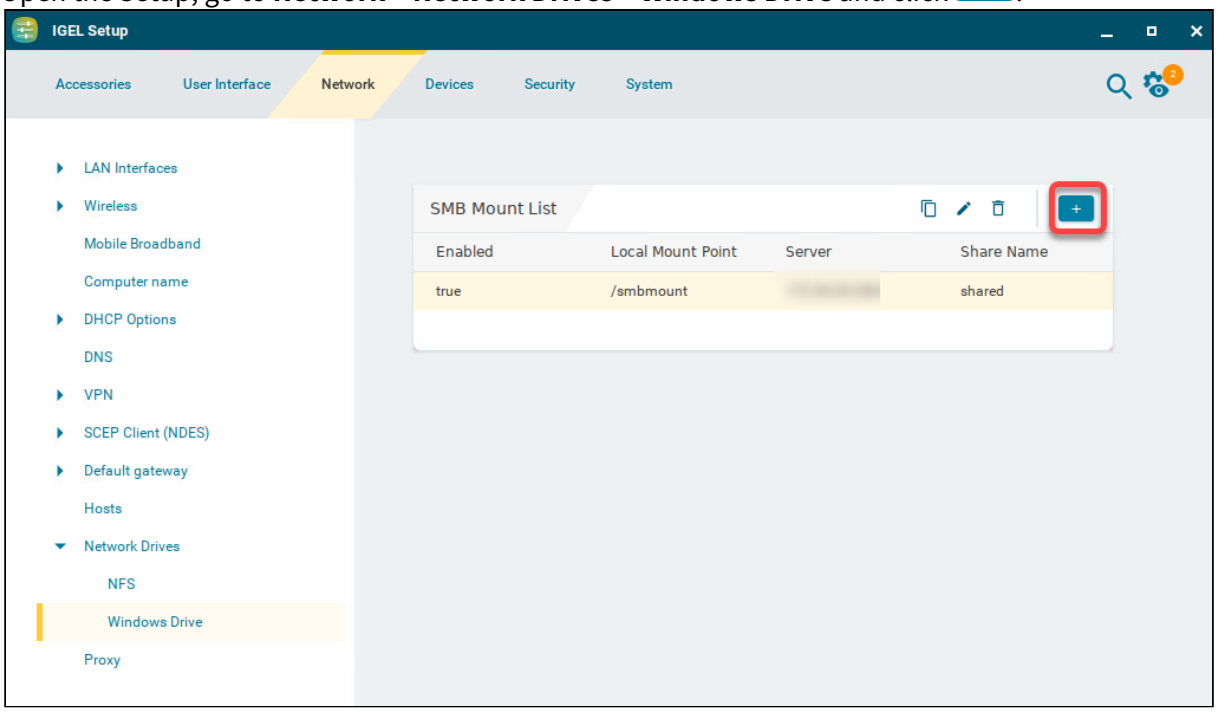
3. Click **Save**.



After a confirmation dialog, the network drive is mounted under `/nfsmount`

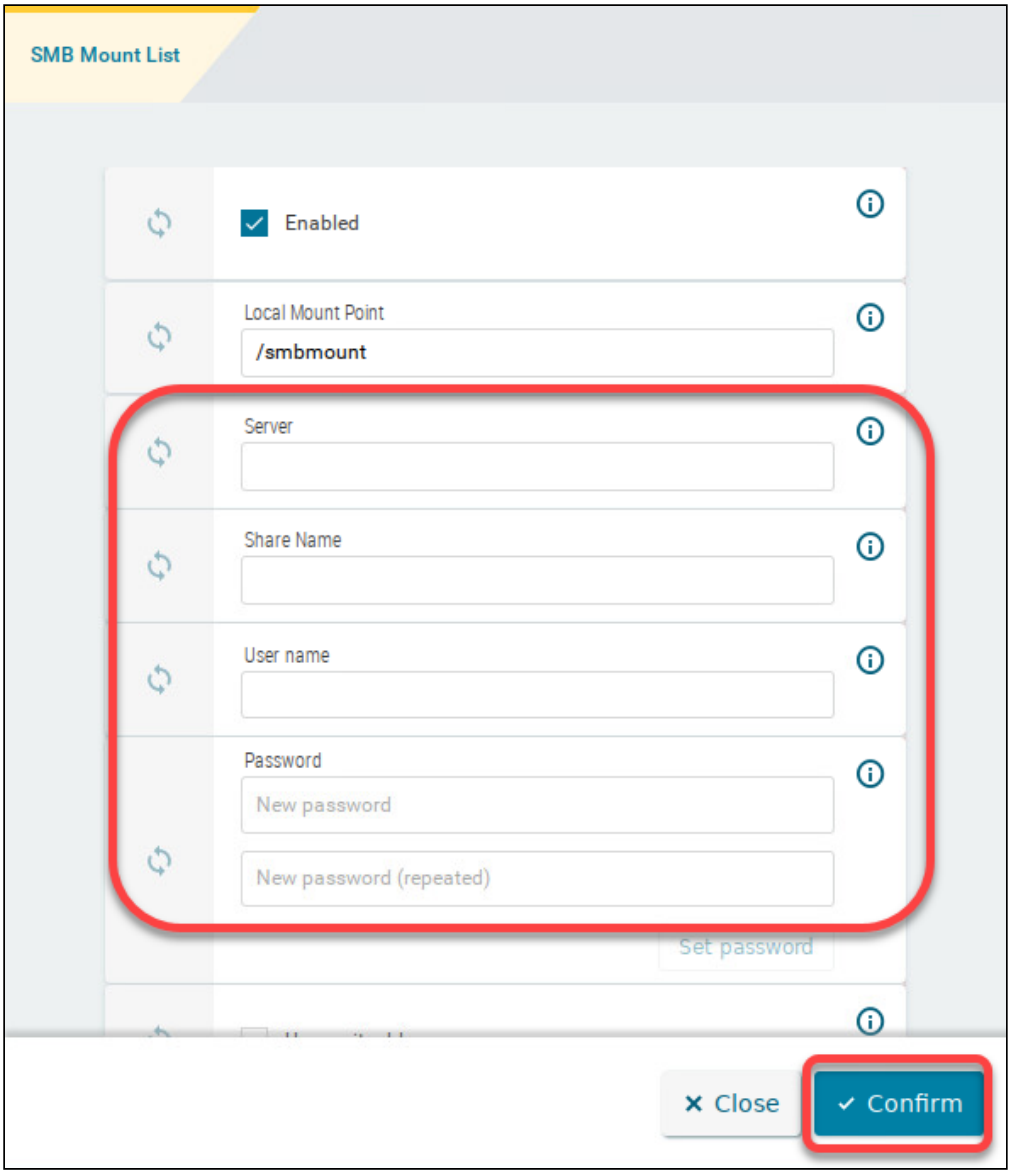
16.4 Deploying Your App from a Windows Drive

1. Open the Setup, go to **Network > Network Drives > Windows Drive** and click  .

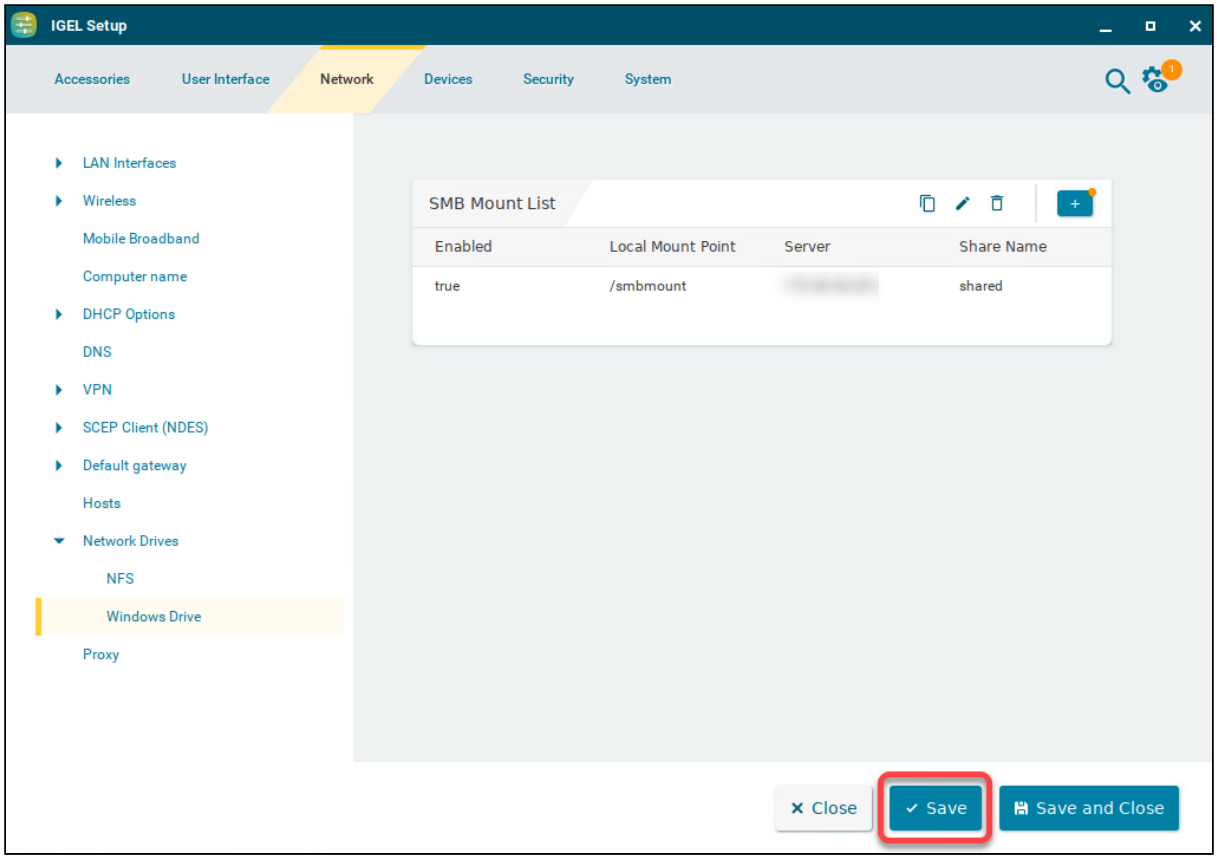


2. Enter the following data and afterward click **Confirm**:
   - **Server**: Address of the SMB server
   - **Share name**: Name of the shared directory
   - **Username**: Name of the user with access to the share

- **Password**: Password of the user with access to the share



3. Click **Save**.



After a confirmation dialog, the network drive is mounted under `/smbmount`

16.5 Installing Your App

▶ Open a terminal on your device, log in as root, and enter the following command:

```
igelpkgctl install -f <PATH TO YOUR IPKG FILE>/<PACKAGE NAME>.ipkg
```

After the reboot, the app can be tested.

17 Review Criteria

17.1 General Criteria

- The name of your app (AppName) must be unique.
- If a read/write partition is part of your app, the mount path conventions must be respected, see Handling Files and Partitions (see page 28).
- Do not include dependencies that already exist in the base system. If you cannot avoid this, you must add the ldconfig path to your dependencies

17.2 Packaging and Installation

- The installation process must be done in a single install script, i.e. the install script MUST NOT call external code, e.g. side-load an installer.
- All app content MUST be delivered in the app package and be installed from the data shipped in the `.ipkg` file, i.e. no binaries, icons, and other content is side-loaded from remote sources.
- The install script MUST NOT perform file changes on base system files, that includes in particular:
  - Changing the ownership of files
  - Changing read/write permissions
  - Setting special bits, such as `setUID`
- The install script MUST access files in /etc as if it was running with user permissions, i.e. files that are read-only for root (such as /etc/shadow) MUST NOT be accessed.
- The install script MUST exit with an integer greater than zero but less than 255 in case of error(s).
- The install script MUST NOT enable or disable services except those that are part of the app. It may only deliver services files according to the IGEL app specification which are then included in the boot order by IGEL mechanisms.
- The install script SHOULD NOT symlink files to read-only locations, as these changes will be gone at the next reboot.
- An app MUST only ship packages directly needed for its operation.
- All delivered packages MUST either provide full license information or a written declaration on licensing and the conformity of the license delivery MUST be given. For details, see Integrating Debian Packages, Tarballs, and Other Archive Types (see page 25).
- The app MUST be free of viruses, malware, and other harmful software to the best knowledge of the app submitter, and such a declaration MUST be given to IGEL in written form.

17.3 Signature

- On submission, the app MUST be signed correctly with the IGEL app SDK key.

17.4 Certificates

- The app MUST ship certificates that are not already in the system's certificate store. It is recommended to store these certificates under `/services/rw/<appName>/certs`

- The app MUST NOT copy its own certificates to the global system certificate store (`/etc/ssl/certs`).

17.5 File System / Partitions

- An app MUST NOT mount or unmount partitions. Creation and delivery of a read-write partition for app data is provided via the rwpartition mechanism.
- SetUID and programs with capabilities SHOULD be avoided. The app MUST not modify binaries of the base system or of other apps to add special permissions.
- An app SHOULD write all of its created data to its corresponding read/write partition.
- An app MUST NOT write directly to `/wfs` or `/etc`
- Volatile data, valid for only the current boot, SHOULD be written to temporary locations. It is recommended to use a separate folder for this app, i.e. `/tmp/<appName>/`
- An app MUST not modify the file permissions and ownership of its own read-only partition, of the read-only partition of any other apps, or of the read-only partition of the base system file during runtime.
- An app SHOULD NOT modify directly any read/write data of other apps or of the base system without asking.

17.6 Ports

- The app MUST only open ports relevant to its direct purpose. Each open port must have clear, visible relation to its provided function or reasoning MUST be given in written form.
- The app provider MUST disclose which ports the app opens to detect collisions with other services.

17.7 Sensitive Data

- The app MUST NOT store or send user-sensitive data collected by it (e.g. passwords) in clear text.
- The app SHOULD make use of one of the system's secret handling facilities to store sensitive data:
  - `libsecret`,
  - `org.freedesktop.secrets`
  - The Python module `secretstorage`. For more information, see https://secretstorage.readthedocs.io/en/latest/

17.8 AppArmor

- If AppArmor is used, the following applies: Apparmor profiles MUST follow the naming convention given in the SDK documentation. An apparmor profile MUST only address and/or limit components of the app.

17.9 Logging

- Logging by the app MUST NOT include debug information unless explicitly enabled via setting.

17.10 Setup (UI for App Configuration)

- The app settings MUST begin with `app.<appname>` with `app.` being added by the SDK. Any other settings definition given in the app's setup.def will not be accepted.

18 Example: Building SuperTuxKart as IGEL OS App

In this little tutorial, we will build the Linux game SuperTuxKart as an IGEL OS app. We will get the Debian packages from the Ubuntu repository.

We will walk you through a typical app development process which includes testing and debugging.

You can find the SuperTuxKart App for IGEL OS on GitHub under https://github.com/IGELTechnologyGmbH/supertuxkart

18.1 Prerequisites

- IGEL OS12 App SDK and basic knowledge of how to use it
- IGEL OS 12 Base System 12.2.0 or higher

18.2 Creating the Package Structure

Use the following command to create the directory structure for your package and define, for instance, 1.1.0 as the version number:

```
igelpkg new -n supertuxkart -V 1.1.0
```

18.3 Adding the Debian Packages

1. Go to https://packages.ubuntu.com/focal/supertuxkart and find the name of the Supertux package(s).
   As you should have found out, the package names are `supertuxkart` and `supertuxkart-data`.

2. Open `supertuxkart-1.1.0/igel/debian.json` and add the two packages:

```
[
  {
    "package": "supertuxkart"
  },
  {
    "package": "supertuxkart-data"
  },
]
```

> The `debian.json` snippet that has been created initially contains fields for adding licenses; see also Integrating Debian Packages, Tarballs, and Other Archive Types (see page 25). Normally, you can skip these fields because the licenses are retrieved automatically from the Debian package. However, if this should fail, you must add the licenses manually using the corresponding fields. Later in this tutorial, we will see an example of this.

18.4 Providing the Mandatory Metadata

To have a minimal set of metadata, we must provide the author and the vendor of the package. Also, we must provide the technical and the public version number according to the versioning rules for IGEL apps; in this example, we define the first release candidate of build 1 (for details, see Versioning (see page 12)).

▶ Open `supertuxkart-1.1.0/app.json` and edit tn the `author` the `vendor`, the `version`, the `public_version` , and the `summary`.

```
{
  "author": "IGEL",
  "categories": [
    "misc"
  ],
  "conflicts": [
    {
      "name": "",
      "version": ""
    }
  ],
  "icons": {
    "app": "supertuxkart_portal.svg",
    "monochrome": "supertuxkart_monochrome.svg"
  },
  "name": "supertuxkart",
  "provides": [
    {
      "name": "",
      "version": ""
    }
  ],
  "public_version": "1.0.0 BUILD 1.0 RC 1",
  "release": "",
  "requires": [
    {
      "name": "",
      "version": ""
    }
  ],
  "suggests": [
    {
      "name": "",
```

```
      "version": ""
    }
  ],
  "summary": {
    "en": "SuperTuxKart"
  },
  "vendor": "Acme",
  "version": "1.0.0+0.1.rc.1"
}
```

18.5 Building the App, First Try

▶ Go to `supertuxkart-1.1.0/` and enter the following command to build the app:

```
igelpkg build -r focal
```

18.6 Checking for Missing Dependencies

We will now run a dependency check to see if any libraries are missing. For this purpose, we need to have the IGEL OS Base System app in place. The dependency-checking tool `igelpkg check` will then determine which dependencies are not already satisfied by the base system.

1. Go to https://app.igel.com and download the latest version of the IGEL OS Base System.
2. Put the IGEL OS Base System package (`base_system-<version>.ipkg`) to a convenient location near your `supertuxkart-1.1.0/` directory.
3. From `supertuxkart-1.1.0/`, use `igelpkg check` to check the dependencies:

   ```
   igelpkg check -l igelpkg.output/supertuxkart-1.1.0.ipkg <PATH TO YOUR BASE
   SYSTEM PACKAGE>/base_system-<version>.ipkg
   ```

   With the option -l, the logfile igelpkg.log will be created. We note that some libraries are missing:
   - `libGLEW.so.2.1`
   - `libsquish.so.0`
   - `libopenal.so.1`
   - `libraqm.so.0`

18.7 Adding the Missing Dependencies

Apply the following procedure for all missing libraries:

1. Search for the missing library at https://packages.ubuntu.com/; select **focal** as the **Distribution**. To search for the missing `libmcpp.so.0`, for instance, the search string would be https://packages.ubuntu.com/search?searchon=contents&keywords=libmcpp.so.0&mode=exactfilename&suite=focal&arch=any

2. Add the package to `debian.json` as you did with `supertuxkart` and `supertuxkart-data`.

```
[
  {
    "package": "supertuxkart"
  },
  {
    "package": "supertuxkart-data"
  },
  {
    "package": "libmcpp0"
  },
]
```

If the process of iterating through the dependencies seems too tedious, we have prepared a more complete `debian.json` as a shortcut:

```
[
  {
    "package": "supertuxkart"
  },
  {
    "package": "supertuxkart-data"
  },
  {
    "package": "libmcpp0"
  },
  {
    "package": "libglew2.1"
  },
  {
    "package": "libsquish0"
  },
  {
    "package": "libopenal1"
  },
  {
    "package": "libraqm0"
  }
]
```

18.8 Building the App, Second Try

▶ In the `supertuxkart-1.1.0/` directory, enter the build command again:

```
igelpkg build -r focal
```

This time, `igelpkg` complains that it could not retrieve any license information for the Debian package `libmcpp0`. This is because `libmcpp0` has a non-standard license; with most Debian packages, `igelpkg` can retrieve the license automatically from the repository. We will fix the missing license in the next step.

18.9 Fixing the Non-standard License for libmcpp0

To fix the license issue, we reference a copyright file from the `libmcpp0` package. Our `debian.json` now looks like this:

```
[
  {
    "package": "supertuxkart"
  },
  {
    "package": "supertuxkart-data"
  },
  {
    "package": "libmcpp0",
    "licenses": [
      {
          "name": "mcpp-2.7",
          "file": "%tmp%/usr/share/doc/libmcpp0/copyright"
      }
    ]
  },
  {
    "package": "libglew2.1"
  },
  {
    "package": "libsquish0"
  },
  {
    "package": "libopenal1"
  },
  {
    "package": "libraqm0"
  }
]
```

18.10 Another Build

▶ In the `supertuxkart-1.1.0/` directory, enter the build command again:

```
igelpkg build -r focal
```

18.11 Checking the Dependencies Again

▶ To check if we still have unresolved dependencies, run the checker again.

```
igelpkg check -l igelpkg.output/supertuxkart-1.1.0.ipkg <PATH TO YOUR BASE SYSTEM
PACKAGE>/base_system-<version>.ipkg
```

The dependency check has found another issue; the following object is missing:

- `libsndio.so.7.0`

18.12 Adding the Missing Dependency

▶ Add `libsndio7.0` to your `debian.json` so that it looks like this:

```
[
  {
    "package": "supertuxkart"
  },
  {
    "package": "supertuxkart-data"
  },
  {
    "package": "libmcpp0",
    "licenses": [
      {
          "name": "mcpp-2.7",
          "file": "%tmp%/usr/share/doc/libmcpp0/copyright"
      }
    ]
  },
  {
    "package": "libglew2.1"
  },
  {
    "package": "libsquish0"
  },
```

```
  {
    "package": "libopenal1"
  },
  {
    "package": "libraqm0"
  },
  {
    "package": "libsndio7.0"
  }
]
```

18.13 Building the App Again and Signing It

Now that all dependencies should be referenced, we can build it again and sign it.

▶ To build and sign the app in one go, enter

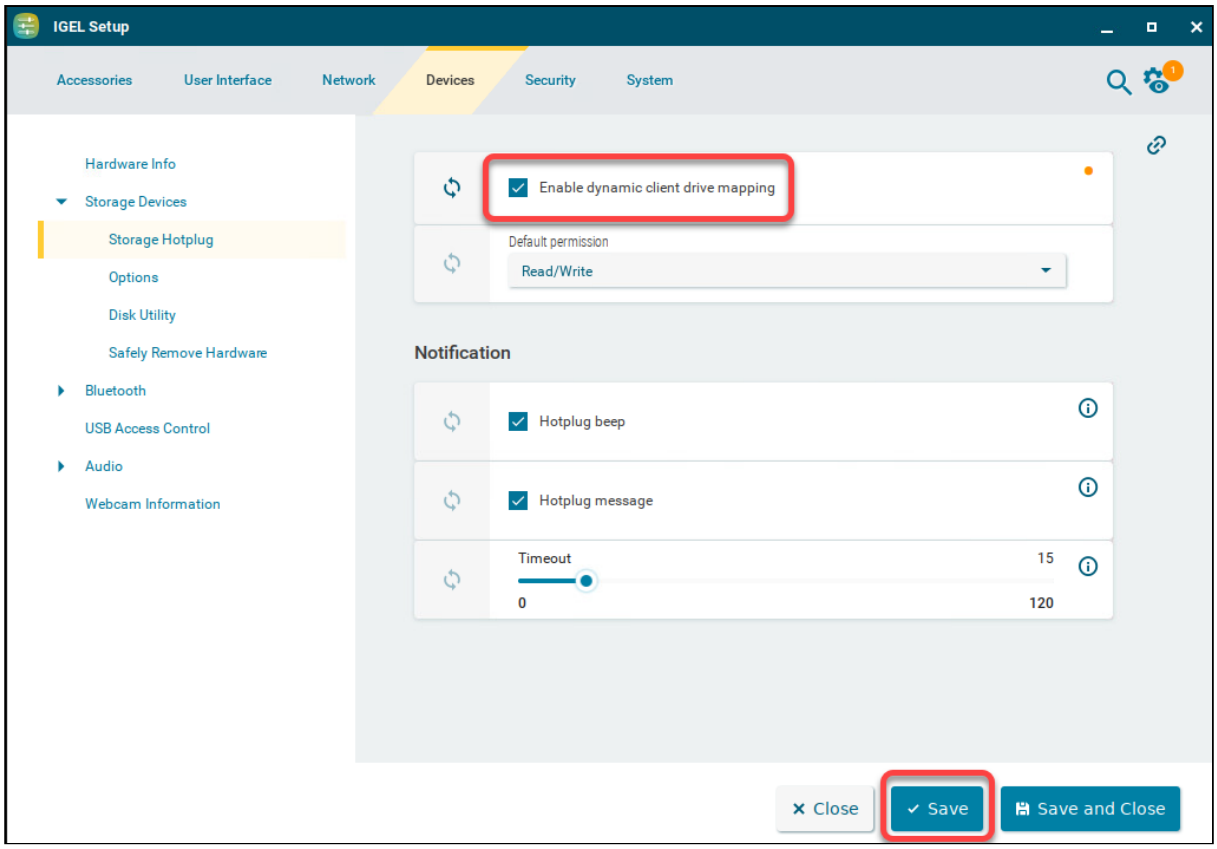
```
igelpkg build -r focal -sp
```

18.14 Installing the App on an IGEL OS Device

It is recommended to install the app from a local USB drive or a network drive (NFS or SMB/Windows).

18.14.1 USB Flash Drive

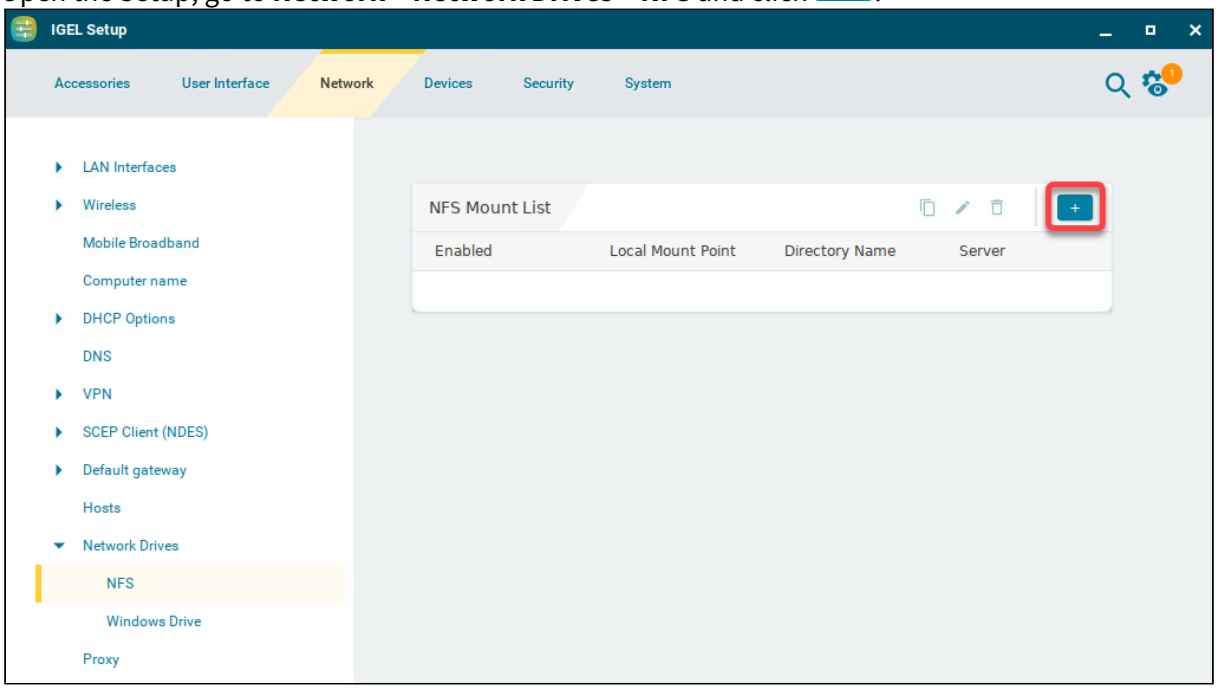
1. Open the Setup, go to **Devices > Storage Devices > Storage Hotplug**, activate **Enable dynamic client drive mapping** and click **Save**.



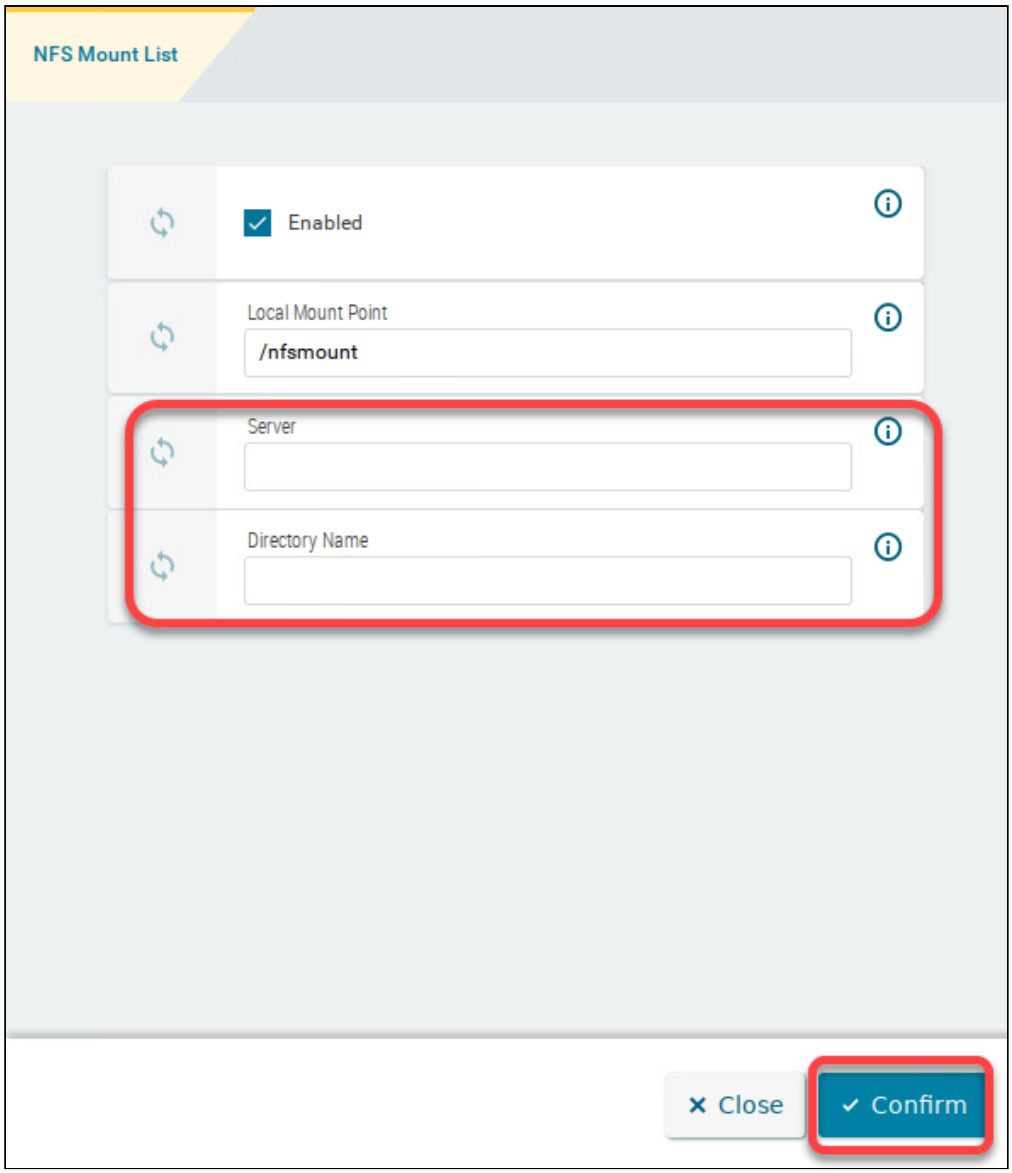
2. Plug the USB flash drive into the device.
   The USB flash drive is mounted under `/userhome/media`

18.14.2 NFS Drive

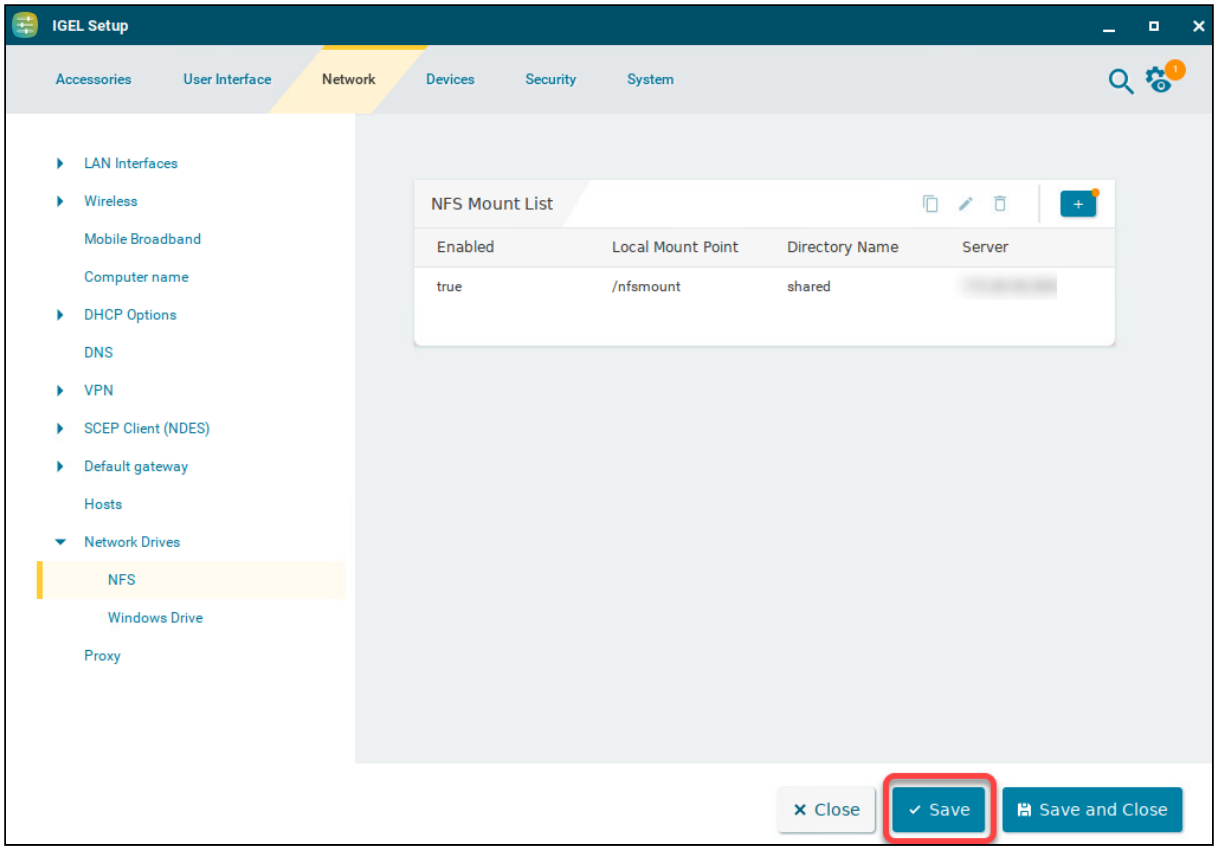
1. Open the Setup, go to **Network > Network Drives > NFS** and click  .



2. Enter the address of the **Server** and the **Directory Name** of the exported directory, and click **Confirm**.


NFS Mount List
Enabled
Local Mount Point
/nfsmount
Server
Directory Name
Close
Confirm

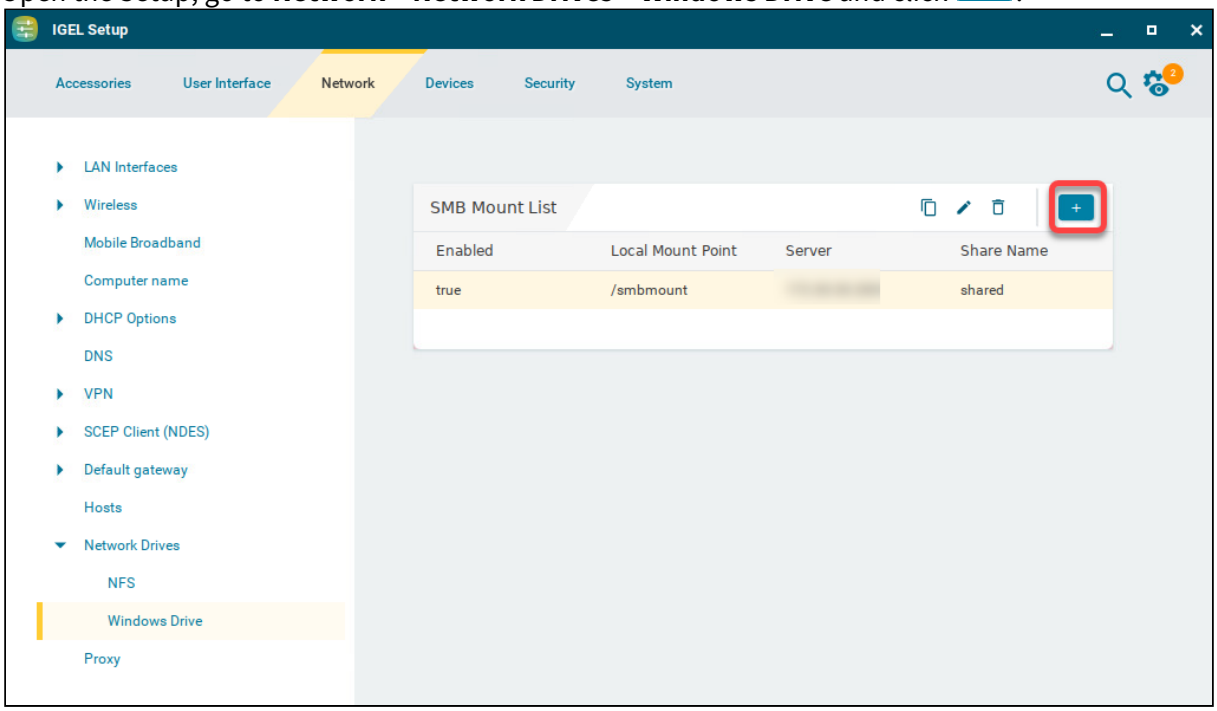
3. Click **Save**.



After a confirmation dialog, the network drive is mounted under `/nfsmount`

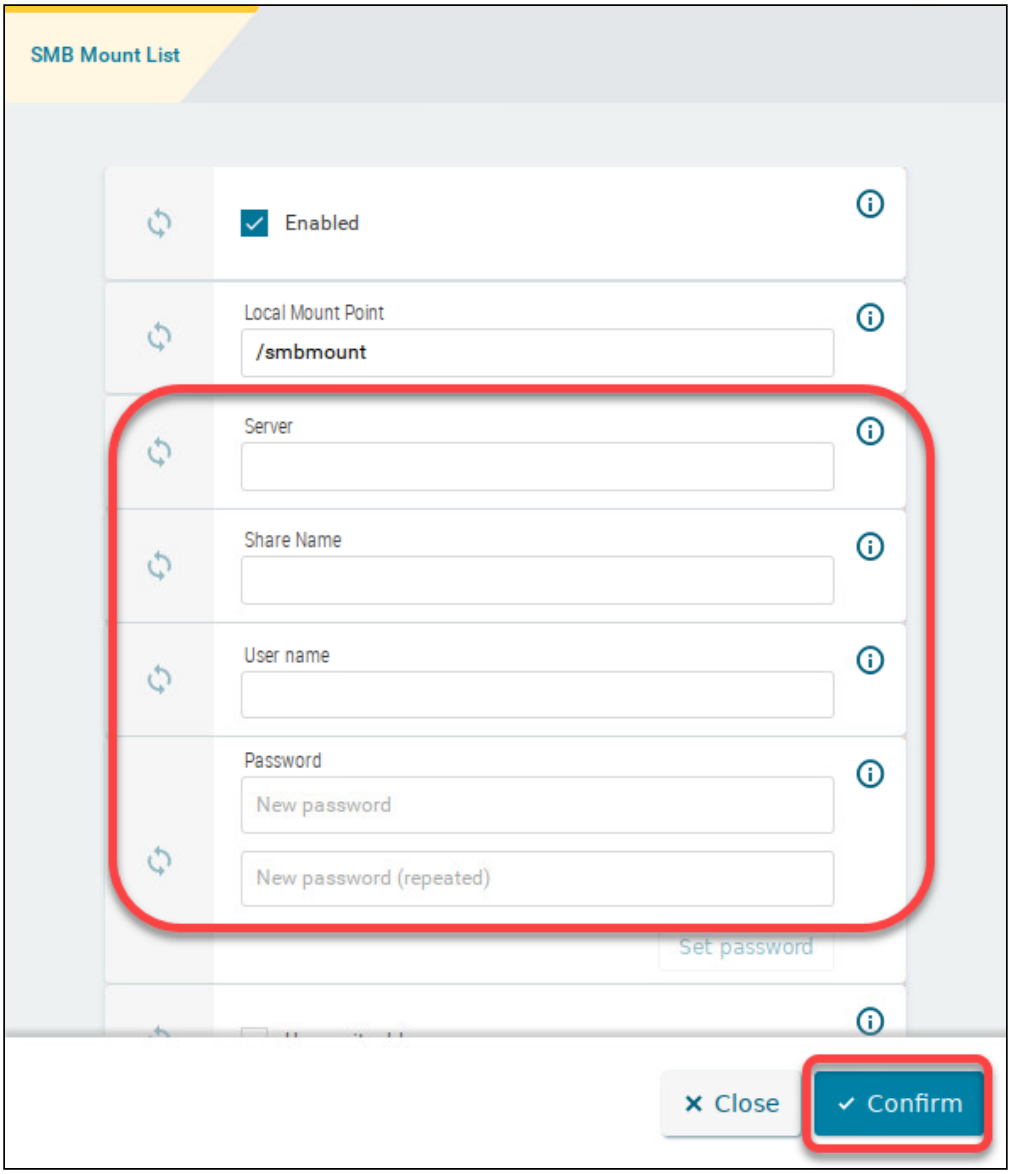
18.14.3 Windows Drive

1. Open the Setup, go to **Network > Network Drives > Windows Drive** and click  .

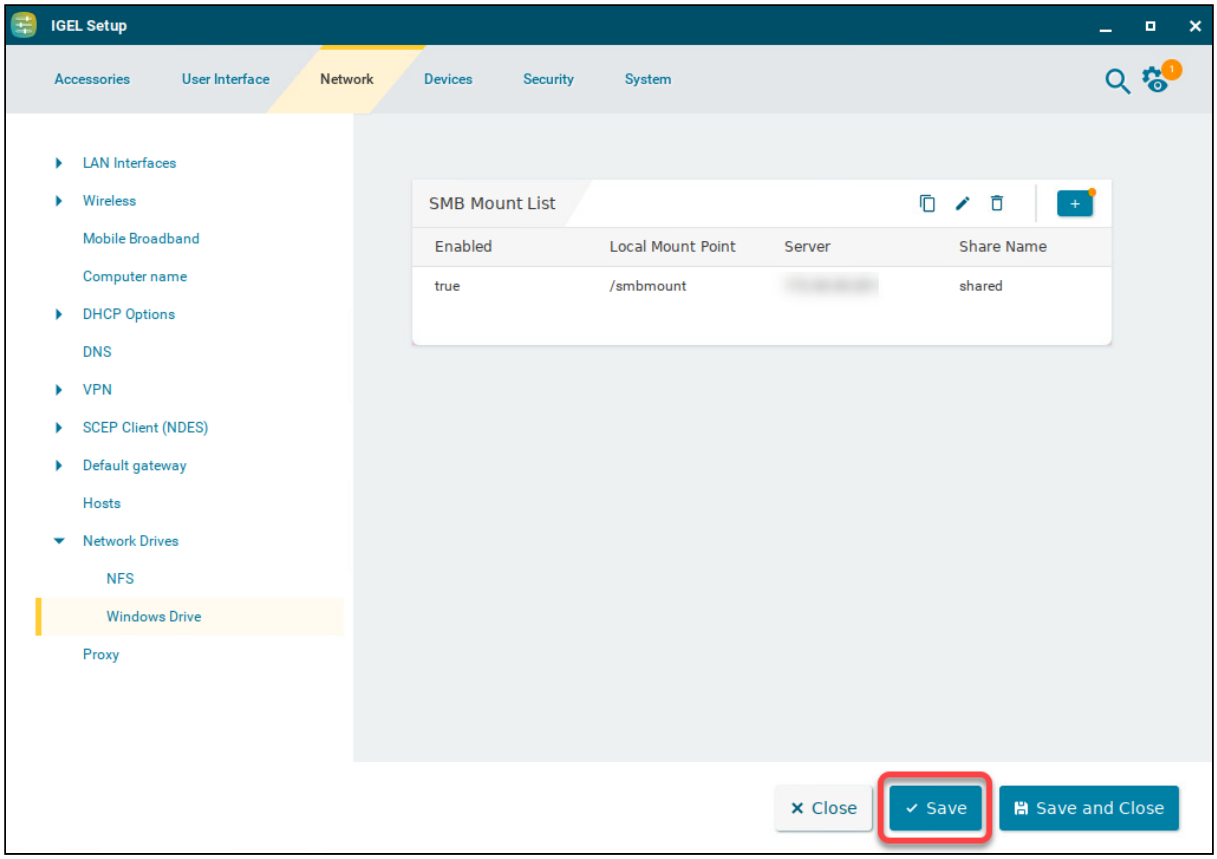


2. Enter the following data and afterward click **Confirm**:
   - **Server**: Address of the SMB server
   - **Share name**: Name of the shared directory
   - **Username**: Name of the user with access to the share

- **Password**: Password of the user with access to the share



3. Click **Save**.



After a confirmation dialog, the network drive is mounted under `/smbmount`

18.14.4 App Installation

▶ Open a terminal on your device, log in as root, and enter the following command:

```
igelpkgctl install -f <PATH TO YOUR IPKG FILE>/supertuxkart-1.1.0.ipkg
```

18.15 Starting the App, First Try

▶ Open a terminal on your device, log in as `user`, and enter the following command :

```
/services/supertuxkart/usr/games/supertuxkart
```

The program issues error messages because of missing fonts.

```
[info    ] ShaderFilesManager: Compiling shader: /usr/share/games/supertuxkart/data/shaders
/uniformcolortexturedquad.frag
[info    ] ShaderFilesManager: Compiling shader: /usr/share/games/supertuxkart/data/shaders
/texturedquad.frag
[info    ] ShaderFilesManager: Compiling shader: /usr/share/games/supertuxkart/data/shaders
/coloredquad.vert
[info    ] ShaderFilesManager: Compiling shader: /usr/share/games/supertuxkart/data/shaders
/coloredquad.frag
[info    ] ShaderFilesManager: Compiling shader: /usr/share/games/supertuxkart/data/shaders
/colortexturedquad.vert
[info    ] ShaderFilesManager: Compiling shader: /usr/share/games/supertuxkart/data/shaders
/colortexturedquad.frag
[info    ] irr_driver: GLSL supported.
[fatal   ] [FileManager]: Can not find file 'Cantarell-Regular.otf' in '/usr/share/games/su
pertuxkart/data/ttf/'
user@IT                :~$
```

18.16 Adding the Missing Fonts

After a few cycles that include searching in https://packages.ubuntu.com, we can complete our `debian.json`:

```
[
  {
    "package": "supertuxkart"
  },
  {
    "package": "supertuxkart-data"
  },
  {
    "package": "libmcpp0",
    "licenses": [
      {
          "name": "mcpp-2.7",
          "file": "%tmp%/usr/share/doc/libmcpp0/copyright"
      }
    ]
  },
  {
    "package": "libglew2.1"
  },
  {
    "package": "libsquish0"
  },
  {
    "package": "libopenal1"
  },
  {
    "package": "libraqm0"
  },
  {
```

```
    "package": "libsndio7.0"
  },
    {
    "package": "fonts-cantarell"
  },
  {
    "package": "fonts-noto-core"
  },
  {
    "package": "fonts-noto-ui-core"
  },
  {
    "package": "fonts-noto-color-emoji"
  }
]
```

18.17 Building the App Once Again

Now that all fonts should be included, we can build it once again, which includes signing, of course.

- To build and sign the app, enter

```
igelpkg build -r focal -sp
```

18.18 Installing the App Once Again

- Open a terminal on your device, log in as root, and enter the following command to reinstall the app:

```
igelpkgctl install -rf <PATH TO YOUR IPKG FILE>/supertuxkart-1.1.0.ipkg
```

18.19 Starting the App, Second Try

- Open a terminal on your device, log in as user, and enter the following command :

```
/services/supertuxkart/usr/games/supertuxkart
```

The game should now start.

18.20 Creating a Starting Method for the Session

To enable the user to start our app, we need to define a session instance and a starting script,

The session instance is defined using a special syntax; for the starting script, we use Python.

18.20.1 Defining a Session

In this example, we create a configuration file named `sessions.param` to define a session for SuperTuxKart.

▶ In the `supertuxkart-1.1.0/data/config/` directory, create a file named `sessions.param` with the following content::

```
<sessions>
    <supertuxkart%>
        <name>
            value=<SuperTuxKart>
        </name>
        <icon>
            value=<supertuxkart>
        </icon>
        <run_only_once>
            value=<true>
        </run_only_once>
        extends_base=<sessions.base%>
    </supertuxkart%>
    <supertuxkart0>
    </supertuxkart0>
</sessions>
```

18.20.2 Creating a Start Script

In this example, we create a Python script that starts the SuperTuxKart binary.

The start script is referenced by the element `<supertuxkart0></supertuxkart0>` in the `sessions.param` we created above. This complies with the naming conventions for sessions in IGEL OS.

Considering that most IDEs use the file ending for applying the correct syntax highlighting, we name the start script file `supertuxkart.py` and then add a symbolic link named `supertuxkart0`

1. In the `supertuxkart-1.1.0/input/all/` directory, create a directory named `config/` and a subdirectory named `sessions/`
2. In the newly created directory `supertuxkart-1.1.0/input/all/config/sessions/`, create the script `supertuxkart.py` as follows:

```
#!/usr/bin/env python3

import subprocess
```

```
command=['/services/supertuxkart/usr/games/supertuxkart']
subprocess.run(command)
```

3. Make the script executable:

```
chmod +x input/all/config/sessions/supertuxkart.py
```

4. In the directory `input/all/config/sessions/`, create a symbolic link named `supertuxkart0`

```
ln -s supertuxkart.py supertuxkart0
```

18.21 Building, Installing, and Testing the Refined App

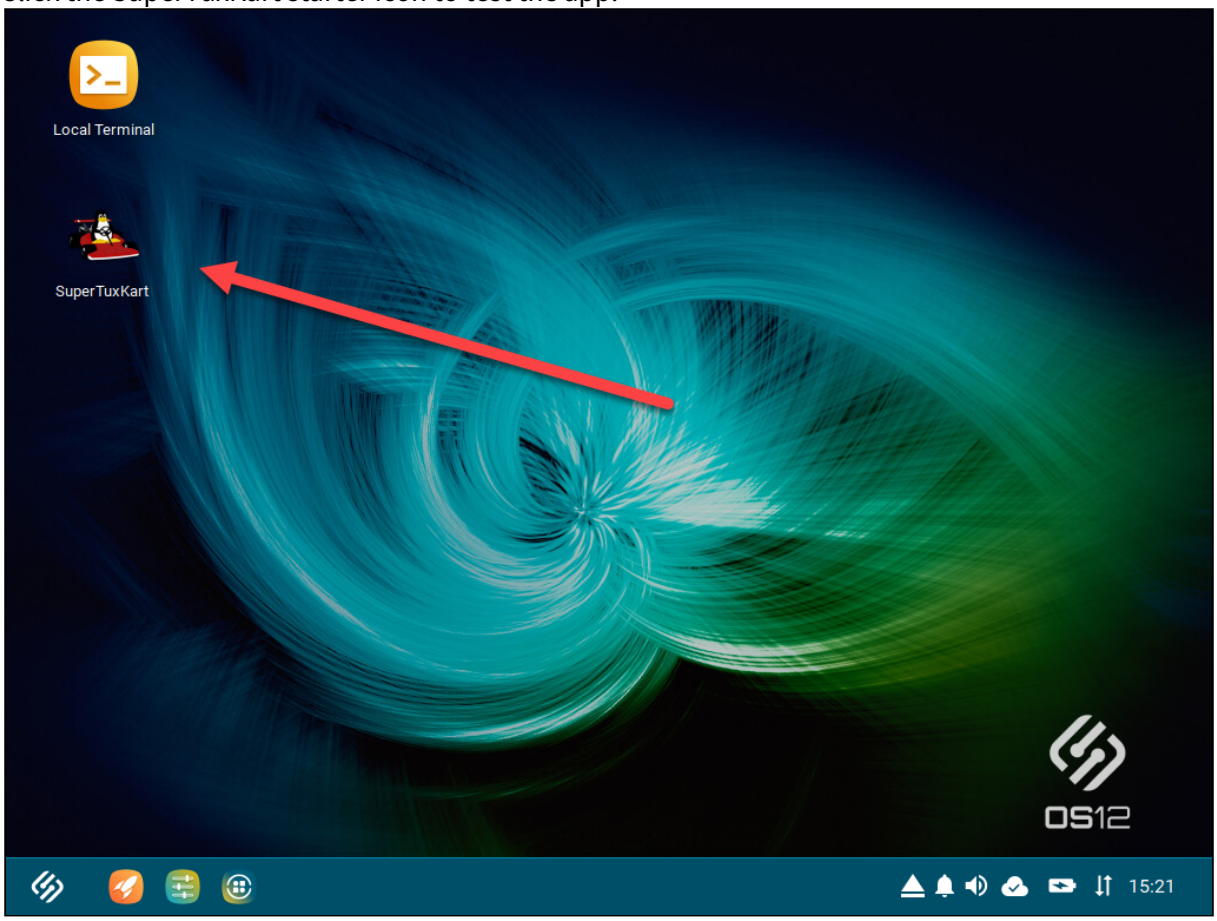
1. Build and sign the app.

```
igelpkg build -r focal -sp
```

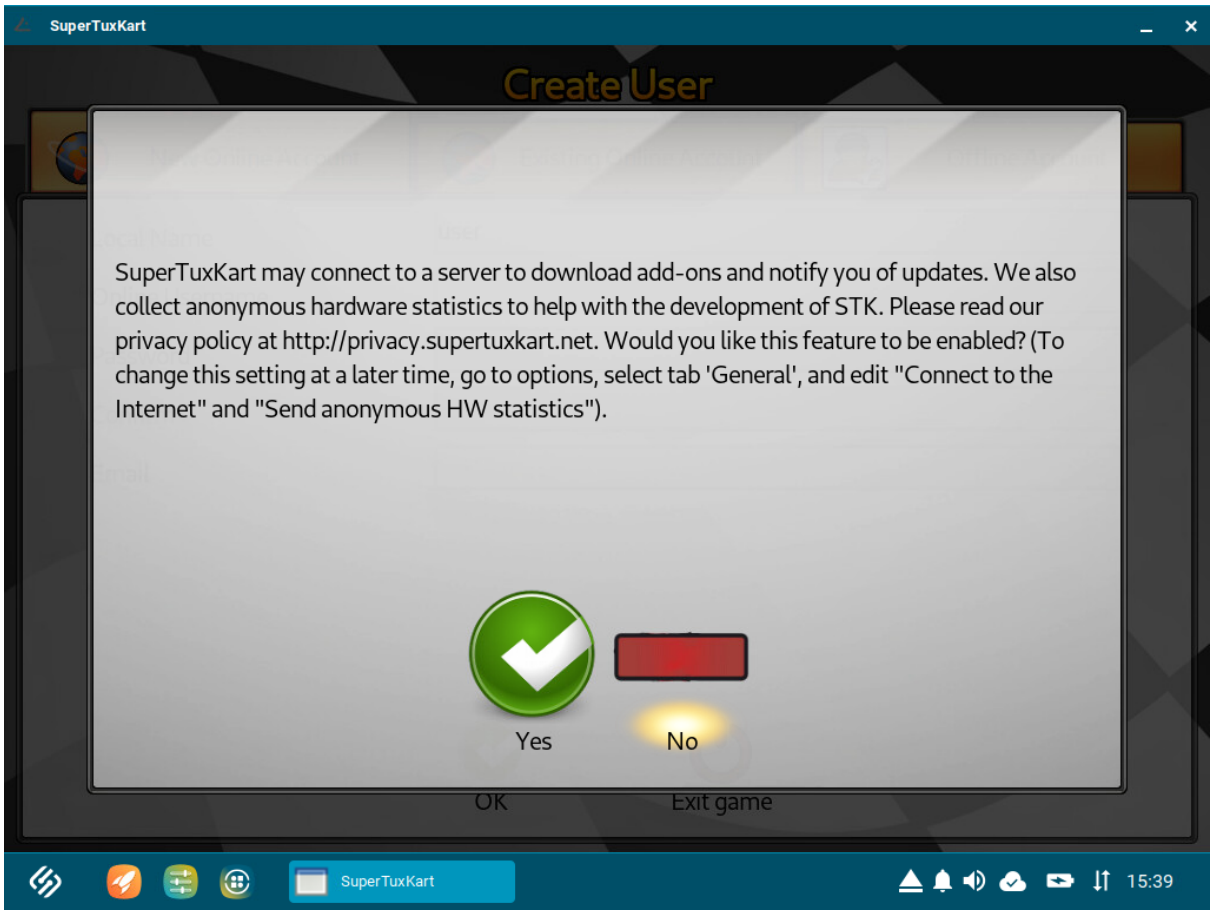
2. Reinstall the app.

```
igelpkgctl install -rf <PATH TO YOUR IPKG FILE>/supertuxkart-1.1.0.ipkg
```

3. Click the SuperTuxKart starter icon to test the app.



If everything went well, SuperTuxKart starts up.



18.22 Adding an App Icon for Display in the App Portal and the UMS

For display in the App Portal and the Universal Management Suite (UMS), an icon with the filename defined in `app.json` must be in the `data/` directory. We will use the icon that came with the Debian package and convert it to SVG.

In the following example, we use ImageMagick for the conversion.

▶ Convert the icon and store it in the `data/` directory:

```
convert igelpkg.output/data_x64/usr/share/icons/hicolor/128x128/apps/supertuxkart.png
data/supertuxkart_portal.svg
```

19 Example: Building the Microsoft Teams Progressive Web App (PWA) as an IGEL OS App

This tutorial gives you an example of how to build a productive Progressive Web App (PWA) as an IGEL OS app.

The Microsoft Teams PWA runs inside the Chromium browser. Our app will start Chromium with parameters that specify the PWA to run, a specific profile, and a specific set of policies.

19.1 Prerequisites

- IGEL OS12 App SDK and basic knowledge of how to use it
- IGEL OS 12 Base System 12.2.0 or higher
- Chromium Browser version 115 or higher

19.2 Creating the Package Structure

Use the following command to create the directory structure for your package and define a version number, for instance, 1.0.0:

```
igelpkg new -n ms_teams_pwa -V 1.0.0
```

19.3 Providing the Basic Metadata

To have a basic set of metadata, we provide the author and the vendor of the package, a summary in English, and at least one category.

▶ Open `ms_teams_pwa-1.0.0/app.json` and edit the following fields:

- `author`
- `vendor`
- `version`
- `icons/app`
- `icons/monochrome`
- `public_version`
- `summary/en`
- `categories`
- `is_pwa`

```
{
  "author": "IGEL Technology GmbH",
  "categories": [
    "Unified Communications",
    "Browser-Based Application"
  ],
```

```
  "conflicts": [
    {
      "name": "",
      "version": ""
    }
  ],
  "icons": {
    "app": "ms_teams_pwa_portal.svg",
    "monochrome": "ms_teams_pwa_monochrome.svg"
  },
  "name": "ms_teams_pwa",
  "provides": [
    {
      "name": "",
      "version": ""
    }
  ],
  "public_version": "1.0.0 BUILD 1.0 RC 1",
  "release": "",
  "requires": [
    {
      "name": "",
      "version": ""
    }
  ],
  "suggests": [
    {
      "name": "",
      "version": ""
    }
  ],
  "summary": {
    "en": "Microsoft Teams Progressive Web App"
  },
  "vendor": "Microsoft Corporation",
  "version": "1.0.0+0.1.rc.1",
  "is_pwa": true
}
```

19.4 Defining Dependencies

To function, our PWA needs the Chromium browser, of course. We define Chromium as a dependency so that it is installed automatically with our app.

Also, a specific version of the base system is required to be able to define a path for separate Chromium policies and to ensure that the `is_pwa` flag to be supported.

- In your `app.json`, fill in the fields under `requires` as follows:

```
"requires": [
   {
        "name": "chromium",
        "version": ">=115.0.5790"
    },
    {
      "name": "base_system",
      "version": ">=12.2.0-0"
    }
],
```

19.5 Defining a Read/Write Partition

A read/write partition is required to store user data locally on the device. The data will be stored persistently.

The following sizes can be defined:

- `small`
- `medium`
- `large`

In addition, the following flags can be added:

- `compressed`:
- `prefer_btrfs`

To have enough space, we will use `large`. Additionally, we add the flag `compressed` to save storage space and choose `prefer_btrfs` as the compression method.

1. Add the following code to your `app.json`:

   ```
   "rw_partition": {
      "size": "large",
      "flags": [
         "compressed",
         "prefer_btrfs"
       ]
   }
   ```

2. To define the storage path, set the directory's owner to `user`, and make the storage persistent, edit `ms_teams_pwa-1.0.0/igel/dirs.json` as follows:

   ```
   [
     {
       "path": "/userhome/.config/ms-teams",
       "owner": "777:100",
       "permissions": "",
   ```

```
    "persistent": true
  }
]
```

19.6 Complete app.json

With all the edits we have made, our app.json should look like this:

```
{
  "author": "IGEL Technology GmbH",
  "categories": [
    "Unified Communications",
    "Browser-Based Application"
  ],
  "conflicts": [
    {
      "name": "",
      "version": ""
    }
  ],
  "icons": {
    "app": "ms_teams_pwa_portal.svg",
    "monochrome": "ms_teams_pwa_monochrome.svg"
  },
  "name": "ms_teams_pwa",
  "provides": [
    {
      "name": "",
      "version": ""
    }
  ],
  "public_version": "1.0.0 BUILD 1.0 RC 1",
  "release": "",
  "requires": [
    {
        "name": "chromium",
        "version": ">=115.0.5790"
    },
    {
      "name": "base_system",
      "version": ">=12.2.0-0"
    }
  ],
  "suggests": [
    {
      "name": "",
      "version": ""
    }
```

```
  ],
  "summary": {
    "en": "Microsoft Teams Progressive Web App"
  },
  "vendor": "Microsoft Corporation",
  "version": "1.0.0+0.1.rc.1",
  "is_pwa": true,
  "rw_partition": {
     "size": "large",
     "flags": [
         "compressed",
         "prefer_btrfs"
     ]
  }
}
```

19.7 Providing an Icon

In this example, we will use one icon file for display in the UMS and in the App Portal and as a starter icon.

For display in the UMS and in the App Portal, the file is referenced in the `app.json`. For use as a starter icon, the file is referenced in the `config.param` that is described further below.

1. Acquire or create the appropriate icon in a color version and a monochrome version. SVG is the image format. For our example, you can download the icon file here: ms_teams_pwa.svg[3]

2. Put the file into `ms_teams_pwa-1.0.0/data/` and rename it to `ms_teams_pwa_portal.svg`

3. Create the directory structure `ms_teams_pwa-1.0.0/input/all/usr/share/icons/hicolor/scalable/apps/` and copy the icon file into it.

19.8 Defining a Session

For our purposes, it will be sufficient to create a minimal session definition and a session launcher, that is, a starter script. The starter script must be created in `ms_teams_pwa-1.0.0/input/all/config/sessions`, and its name must match the session defined in the `config.param` that will be created in the following.

In this example, we create a simple configuration with a single session instance. Please contact the SDK support team if your app requires multiple session instances.

1. To define the session, edit `ms_teams_pwa-1.0.0/data/config/config.param` as follows:

3 https://igel-jira.atlassian.net/wiki/download/attachments/71106623/ms_teams_pwa.svg?api=v2&cacheVersion=1&modificationDate=1716293244181&version=1

```
<sessions>
    <ms_teams_pwa%>
        <name_localized>
            displayname[en_us]=<MS Teams PWA>
        </name_localized>
        <icon>
            value=<ms_teams_pwa>
        </icon>
        <type>
            displayname[en_us]=<Browser>
        </type>
        extends_base=<sessions.base%>
        node_action=<wm_postsetup>
    </ms_teams_pwa%>
    <ms_teams_pwa0>
    </ms_teams_pwa0>
</sessions>
<config_schema_version>
    value=<1>
    type=<integer>
    runtime=<false>
</config_schema_version>
```

2. Create the directory structure `ms_teams_pwa-1.0.0/input/all/config/sessions/` - This is where we will create the starter script.

3. To create the starter script, go to `ms_teams_pwa-1.0.0/input/all/config/sessions/` and create the file `ms_teams_pwa0` (as defined in the `config.param`) with the following content:

```
#!/bin/bash
services/chromium/usr/lib/chromium-browser/chromium-pwa-ms_teams_pwa --
app=https://teams.microsoft.com/ --user-data-dir=<path_to_pwa_profile> --
policy-suffix=teams
```

The command line switches have the following meanings:
`--app`: URL of the Web App, in our case, MS Teams
`--user-data-dir`: Chromium will use the defined path as its profile location; this is the read/write partition we have defined for this app. Also, Chromium will start its own instance of the Chromium browser.
`--policy-suffix`: This will use `/etc/chromium-<policy-suffix>/policies` instead of the default path (does only work in conjunction with `--user-data-dir`)
Example:

```
#!/bin/bash
```

```
/services/chromium/usr/lib/chromium-browser/chromium-browser --app=https://
teams.microsoft.com/ --user-data-dir=/services_rw/ms_teams_pwa/
userhome/.config/ms-teams --policy-suffix=teams
```

4. Make the starter script executable:

```
chmod +x ms_teams_pwa0
```

19.9 Configuring the IGEL Setup

1. Edit `ms_teams_pwa-1.0.0/data/config/ui.json` to define the structure of the Setup page:

```
{
  "root": {
    "id": "root",
    "childrenIds": ["base"]
  },
  "base": {
    "id": "base",
    "parentId": "root",
    "nlsResourceId": "title",
    "title": "MS Teams PWA",
    "childrenIds": ["settings", "session"]
  },
  "session": {
    "id": "session",
    "parentId": "base",
    "nlsResourceId": "session.label",
    "title": "MS Teams PWA Session",
    "instancePage": true,
    "fixInstanceId": 0,
    "elements": {
      "0": {
        "type": "sessionPage",
        "paramId": "app.ms_teams_pwa.sessions.ms_teams_pwa",
        "instancePageId": "sessions.genericInstancePage"
      }
    }
  }
}
```

2. Edit `ms_teams_pwa-1.0.0/data/config/translation.json` to define the label for the session and the title of the Setup page. In this example, we use English and German.

```
{
    "session.label": {
        "de": "MS Teams PWA Sitzung",
        "en": "MS Teams PWA Session"
    },
    "title": {
        "de": "MS Teams PWA",
        "en": "MS Teams PWA"
    }
}
```

19.10 Adding a Description

When your app is publicly available, a description is needed for display in the App Portal. For more information, see Editing Basic Metadata (see page 14).

▶ Edit the file `data/descriptions/en` so that it contains a long description and a short description of the app in English. An empty line separates the long and the short descriptions.

19.11 Building and Signing the App

▶ Change to `ms_teams_pwa-1.0.0/` and enter the following command:

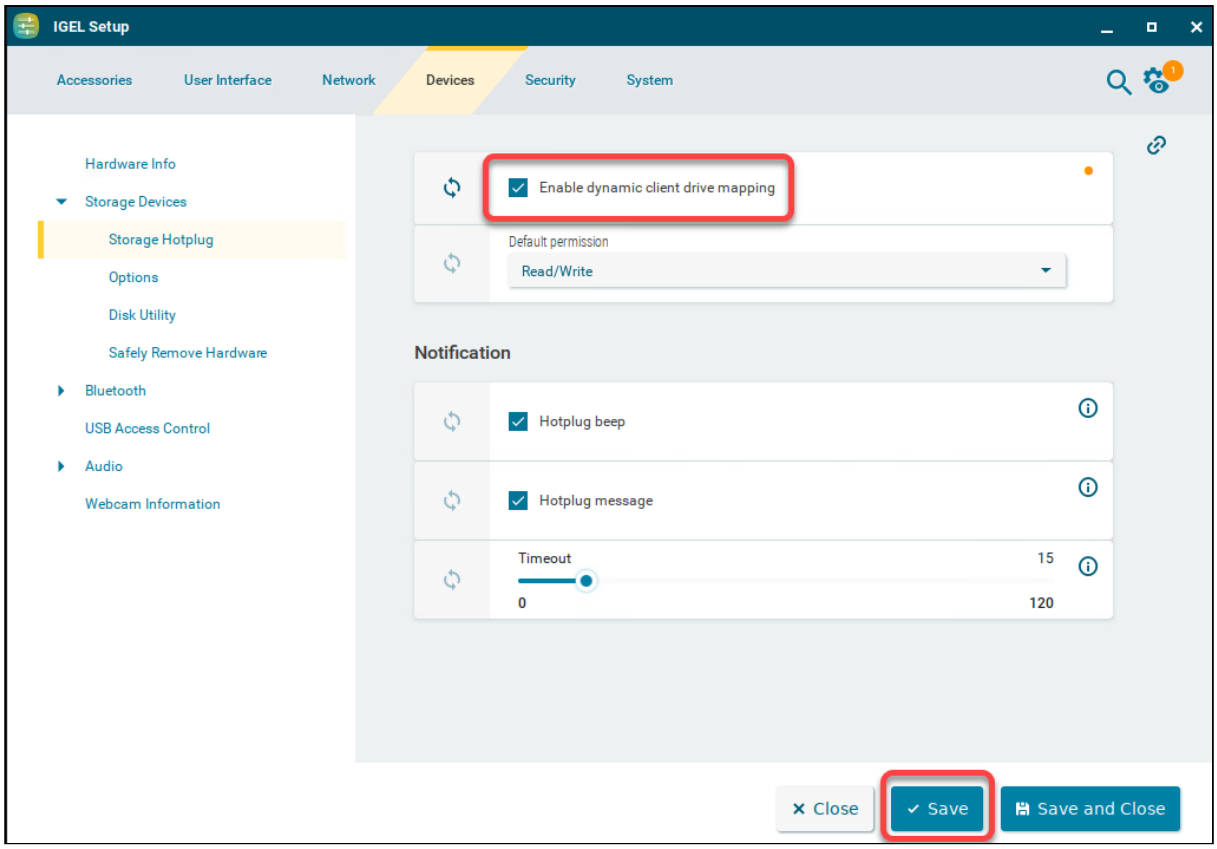
```
igelpkg build -r focal -sp
```

19.12 Installing the App on an IGEL OS Device

Installing the app from a local USB drive or a network drive (NFS or SMB/Windows) is recommended.

19.12.1 USB Flash Drive

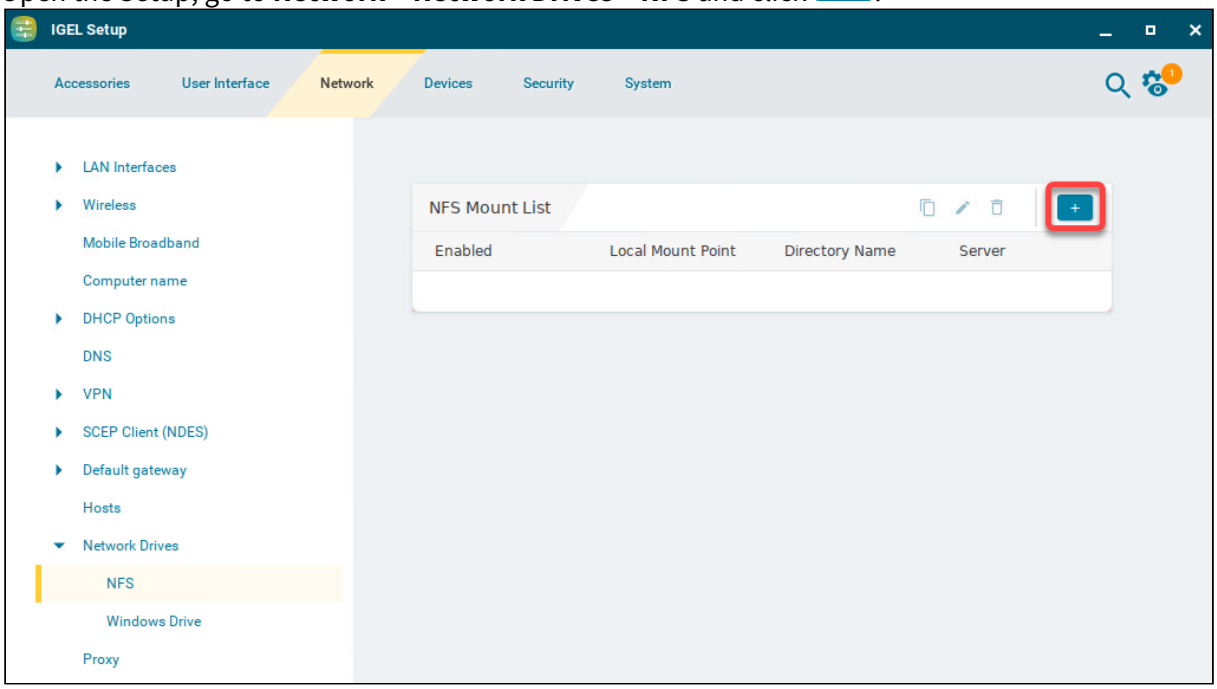
1. Open the Setup, go to **Devices > Storage Devices > Storage Hotplug**, activate **Enable dynamic client drive mapping** and click **Save**.



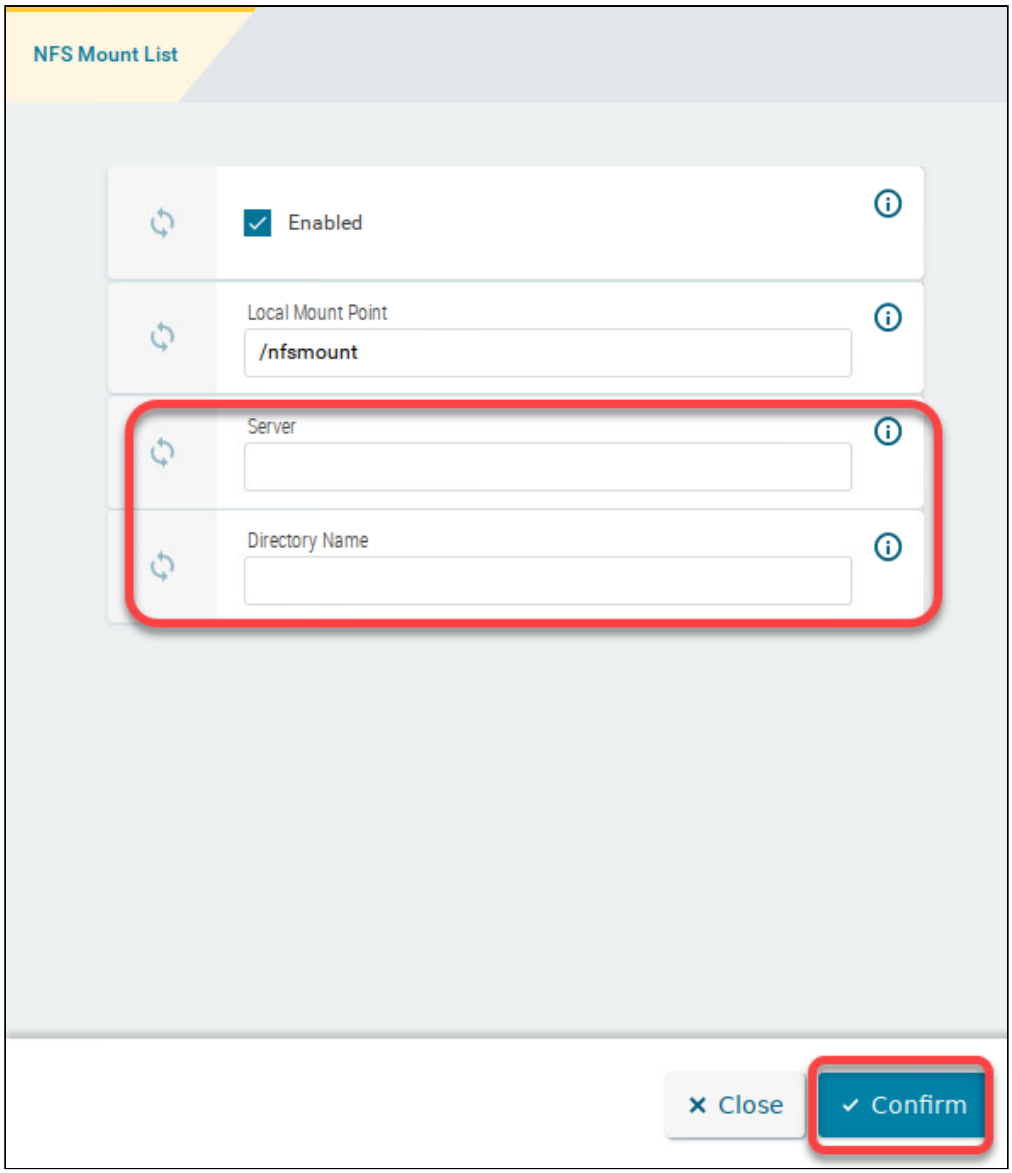
2. Plug the USB flash drive into the device.
   The USB flash drive is mounted under `/userhome/media`

19.12.2 NFS Drive

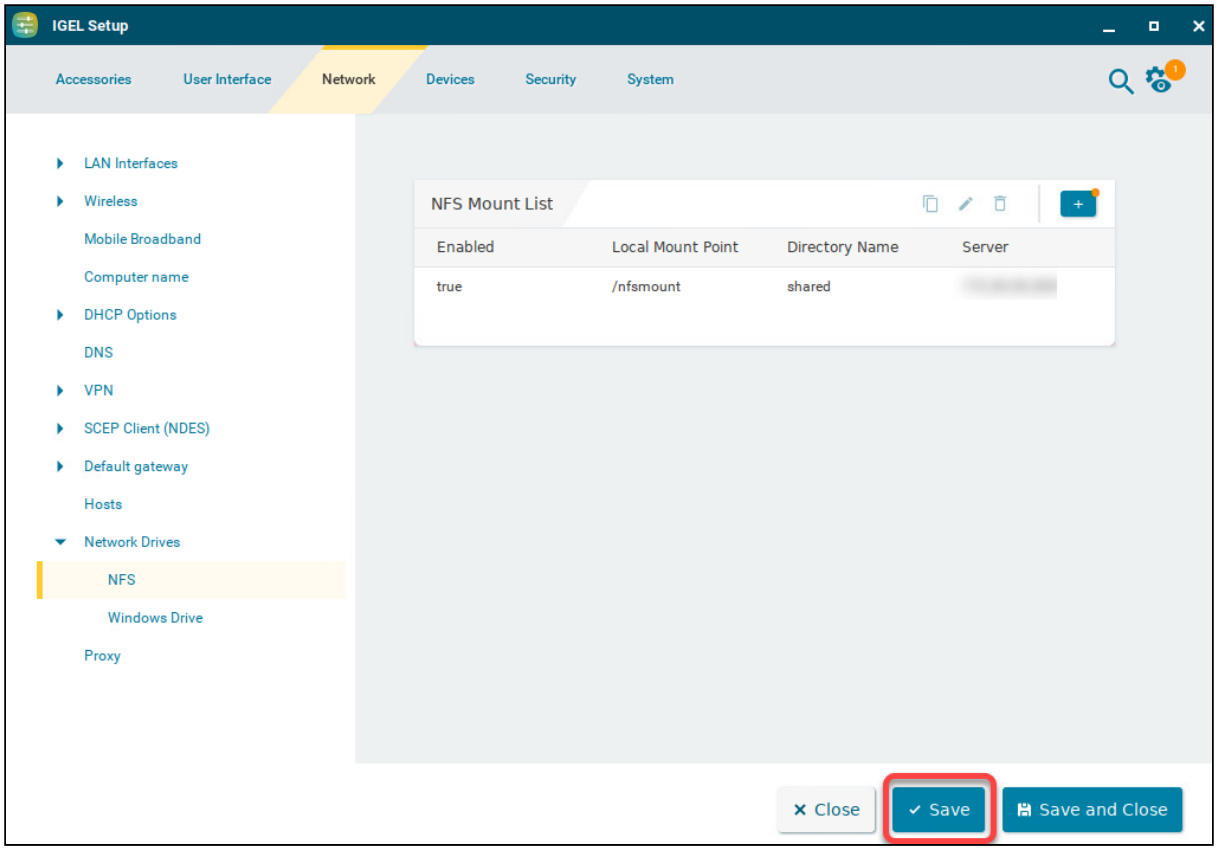
1. Open the Setup, go to **Network > Network Drives > NFS** and click +.



2. Enter the address of the **Server** and the **Directory Name** of the exported directory, and click **Confirm**.


NFS Mount List
Enabled
Local Mount Point
/nfsmount
Server
Directory Name
Close
Confirm

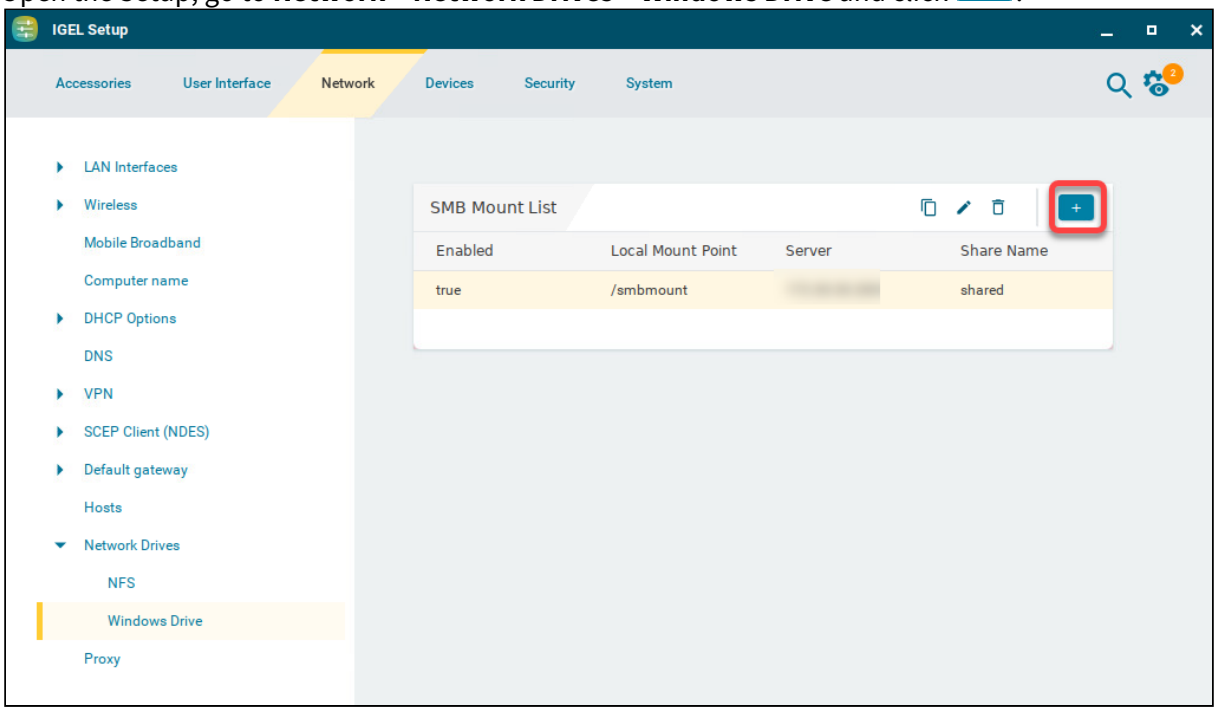
3. Click **Save**.



After a confirmation dialog, the network drive is mounted under `/nfsmount`

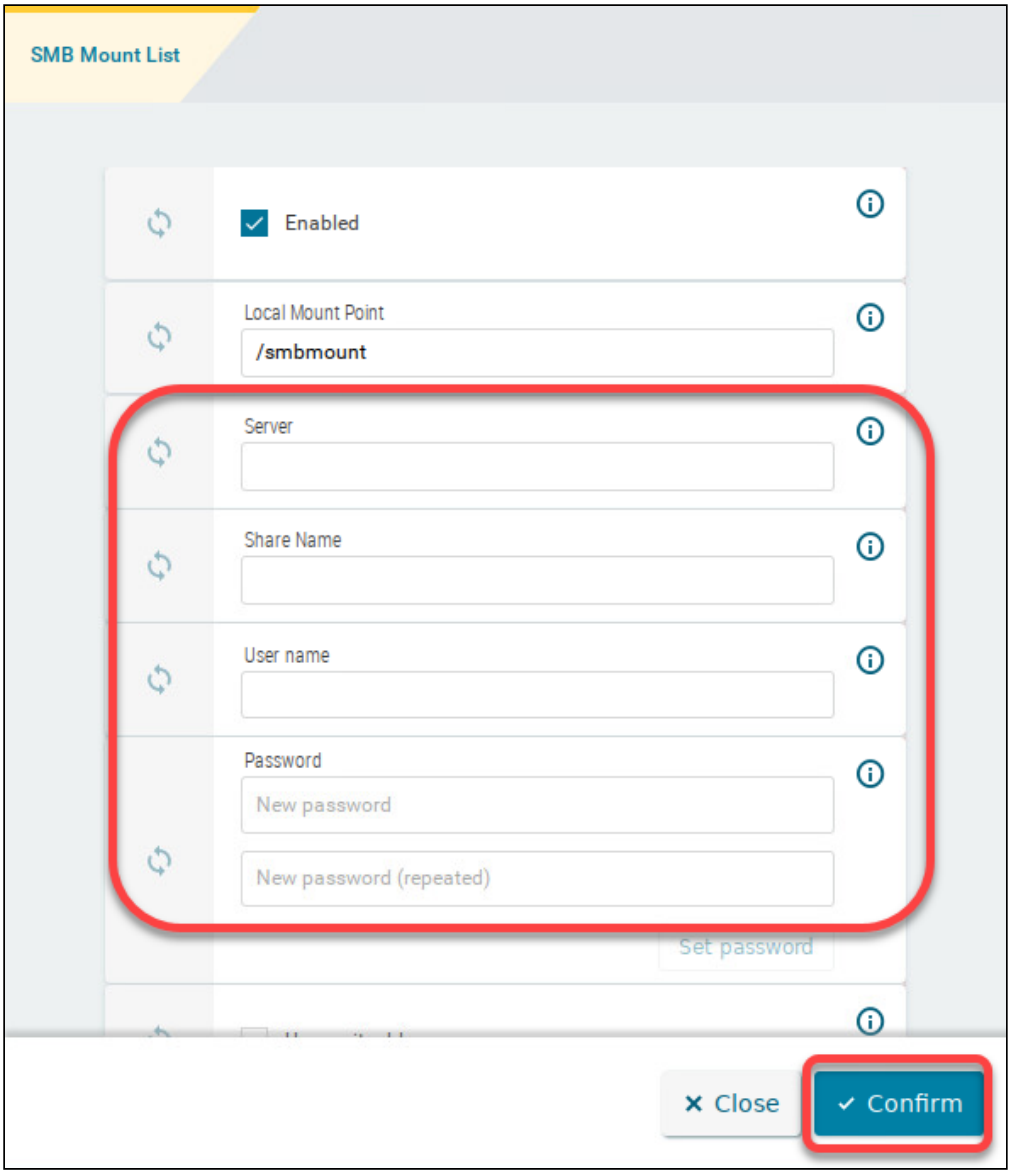
19.12.3 Windows Drive

1. Open the Setup, go to **Network > Network Drives > Windows Drive** and click  .

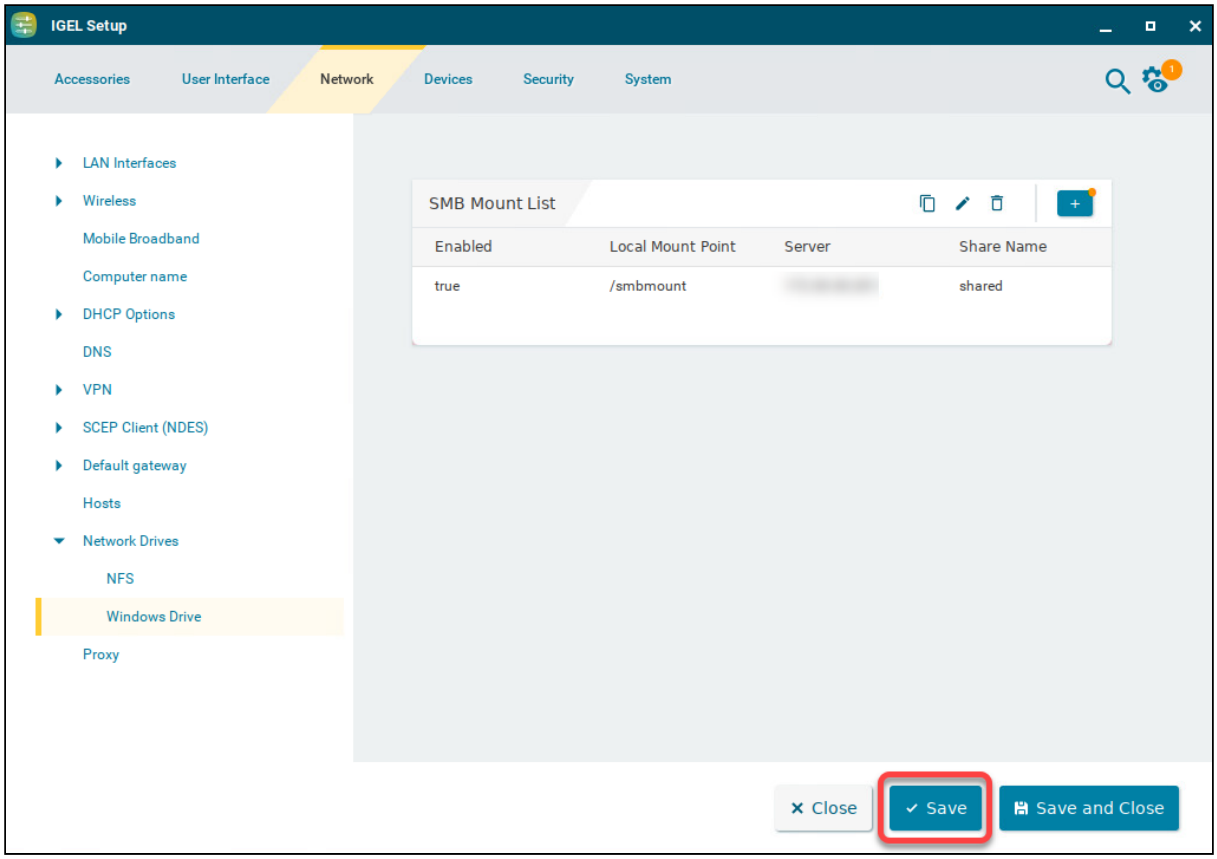


2. Enter the following data and afterward click **Confirm**:
   - **Server**: Address of the SMB server
   - **Share name**: Name of the shared directory
   - **Username**: Name of the user with access to the share

- **Password**: Password of the user with access to the share



3. Click **Save**.



After a confirmation dialog, the network drive is mounted under `/smbmount`

19.12.4 App Installation

1. Make sure your device has IGEL OS 12 Base System 12.2.0 or higher.
2. Open a terminal on your device, log in as root, and enter the following command:

```
igelpkgctl install -f <PATH TO YOUR IPKG FILE>/ms_teams_pwa-1.0.0.ipkg
```

After the reboot, the app can be tested.

19.13 Testing the App

▶ Click the MS Teams PWA starter icon to test the app.


Lokales Terminal
Chromium browser
MS Teams PWA
SuperTuxKart
OS12
14:47