Getting Started With the IGEL OS App SDK

Building Your Own IGEL OS App

- DRAFT -

Exported on 08/17/2023

Table of Contents

This document provides you with the basic knowledge that is required to create an IGEL OS App with `igelpkg`. and publish it on the IGEL App Portal.

Basically, the procedure is as follows:

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.

1 Prerequisites

- Machine with Ubuntu Linux 18.04 or higher, or another Debian-based Linux distribution
- The IGEL OS APP SDK, version 0.9.8 or higher
- Device with IGEL OS 12.01.120. To test if the app can be installed and started, a virtual machine is sufficient. If 3D graphics are required, a hardware device might be a good choice. To test if an app can be installed and started, a virtual machine is sufficient.
- A local terminal is configured on the device

2 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_[version]_amd64.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
```

3 Versioning

The version number must be compliant with the Semantic Versioning (SemVer) specifications, see https://semver.org/.

In addition to SemVer, we use the following schema for IGEL OS apps that are submitted for review/publication:

- For prerelease versions, `-rc[NUMBER]` is added at the end of the version number. Example: `23.1.100-rc.1`
- For release versions, `+[NUMBER]` is added at the end of the version number. Example: `23.1.100+1`

Would it make sense to use this numbering scheme for all examples, i.e. hello_world-1.0.0.100-rc-1 and so on?

4 Creating the Initial Data Structure

You can create the complete initial data structure with `igelpkg new -n [app name] -V [version number]`. For important details on versioning, see Versioning(see page 10).

4.1 Hello World Example

```
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/
```

5 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.

5.1 Required Information

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

5.1.1 Name

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

5.1.2 Version

The version of the app is defined in the field `version`. The SemVer (Semantic Versioning) format is used. The field is populated automatically when the `igelpkg new` command is issued. Example: `1.0.0`

5.1.3 Author

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

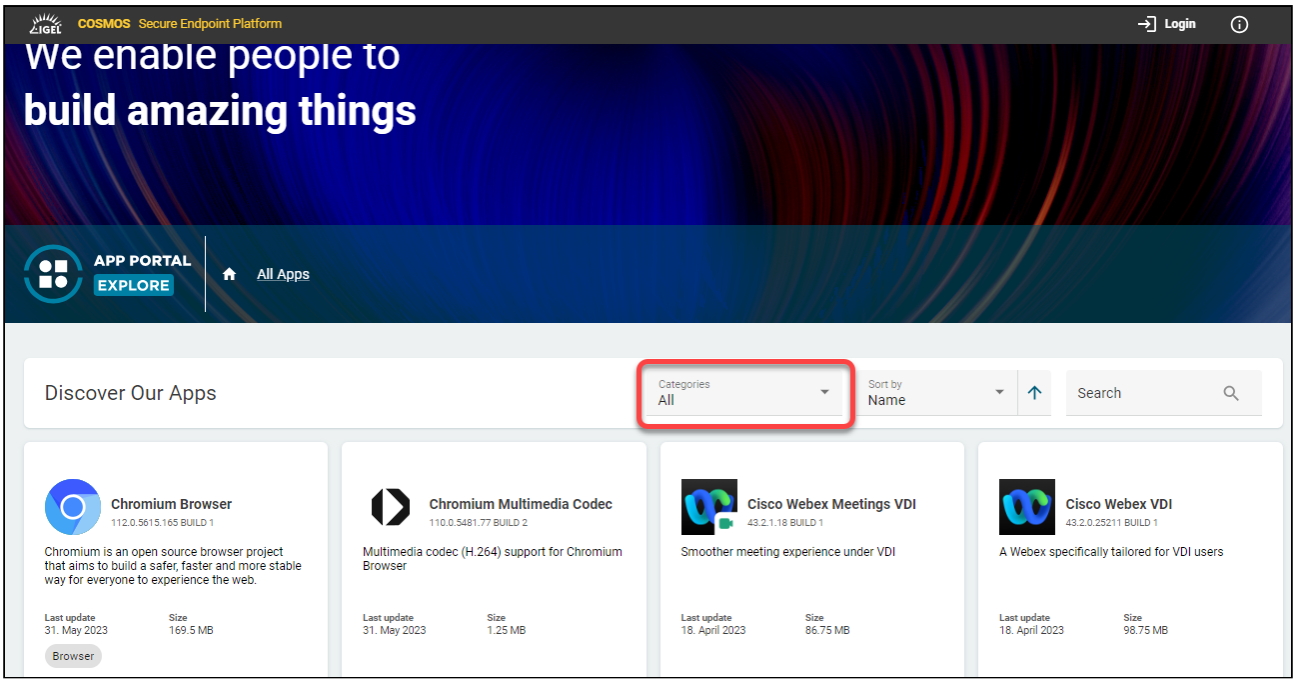
5.1.4 Vendor

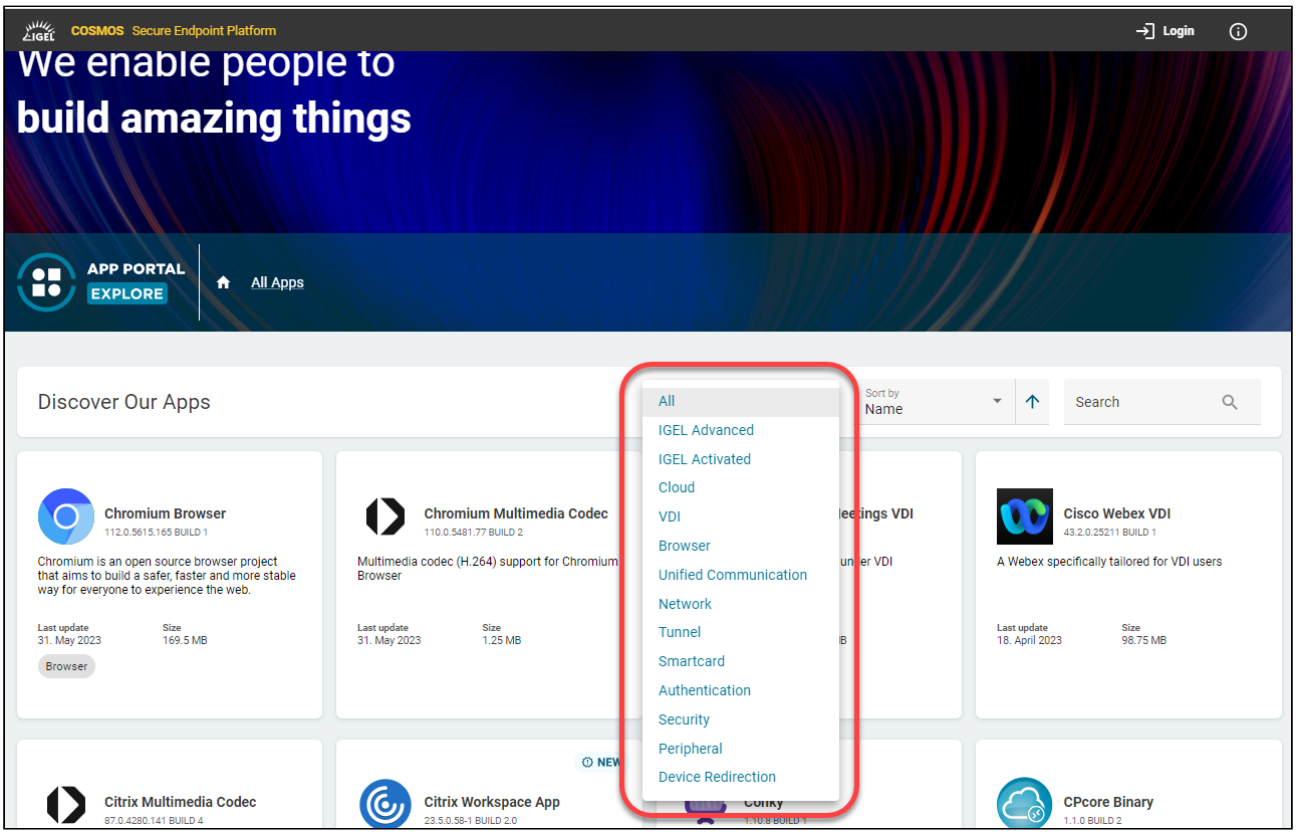
This is the vendor of the application that has been re-packaged for IGEL OS by the `author`. The vendor is defined in the field `vendor`. The field must be filled manually.

5.1.5 Categories

The categories are used in the App Portal and in the Universal Management Suite (UMS) to make it easier to find the right app for a specific use case. 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 the categories currently available, go to https://app.igel.com/ and open the **Categories** menu.


COSMOS Secure Endpoint Platform
Login
We enable people to
build amazing things
APP PORTAL
EXPLORE
All Apps
Discover Our Apps
Categories
All
Sort by
Name
Search
Chromium Browser
112.0.5615.165 BUILD 1
Chromium is an open source browser project that aims to build a safer, faster and more stable way for everyone to experience the web.
Last update
31. May 2023
Size
169.5 MB
Browser
Chromium Multimedia Codec
110.0.5481.77 BUILD 2
Multimedia codec (H.264) support for Chromium Browser
Last update
31. May 2023
Size
1.25 MB
Cisco Webex Meetings VDI
43.2.1.18 BUILD 1
Smoother meeting experience under VDI
Last update
18. April 2023
Size
86.75 MB
Cisco Webex VDI
43.2.0.25211 BUILD 1
A Webex specifically tailored for VDI users
Last update
18. April 2023
Size
98.75 MB


COSMOS Secure Endpoint Platform
Login
We enable people to
build amazing things
APP PORTAL
EXPLORE
All Apps
Discover Our Apps
Sort by
Name
Search
All
IGEL Advanced
IGEL Activated
Cloud
VDI
Browser
Unified Communication
Network
Tunnel
Smartcard
Authentication
Security
Peripheral
Device Redirection
Chromium Browser
112.0.5615.165 BUILD 1
Chromium is an open source browser project that aims to build a safer, faster and more stable way for everyone to experience the web.
Last update
31. May 2023
Size
169.5 MB
Browser
Chromium Multimedia Codec
110.0.5481.77 BUILD 2
Multimedia codec (H.264) support for Chromium Browser
Last update
31. May 2023
Size
1.25 MB
Cisco Webex VDI
43.2.0.25211 BUILD 1
A Webex specifically tailored for VDI users
Last update
18. April 2023
Size
98.75 MB
Citrix Multimedia Codec
87.0.4280.141 BUILD 4
Citrix Workspace App
23.5.0.58-1 BUILD 2.0
CPcore Binary
1.1.0 BUILD 2

5.1.6 Minimal Hello World Example

Empty fields can be removed safely.

```
{
  "name": "hello_world",
  "version": "1.0.0",
  "public_version": "",
  "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": ""
    }
  ]
}
```

5.2 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/` Which image formats/sizes are supported?

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

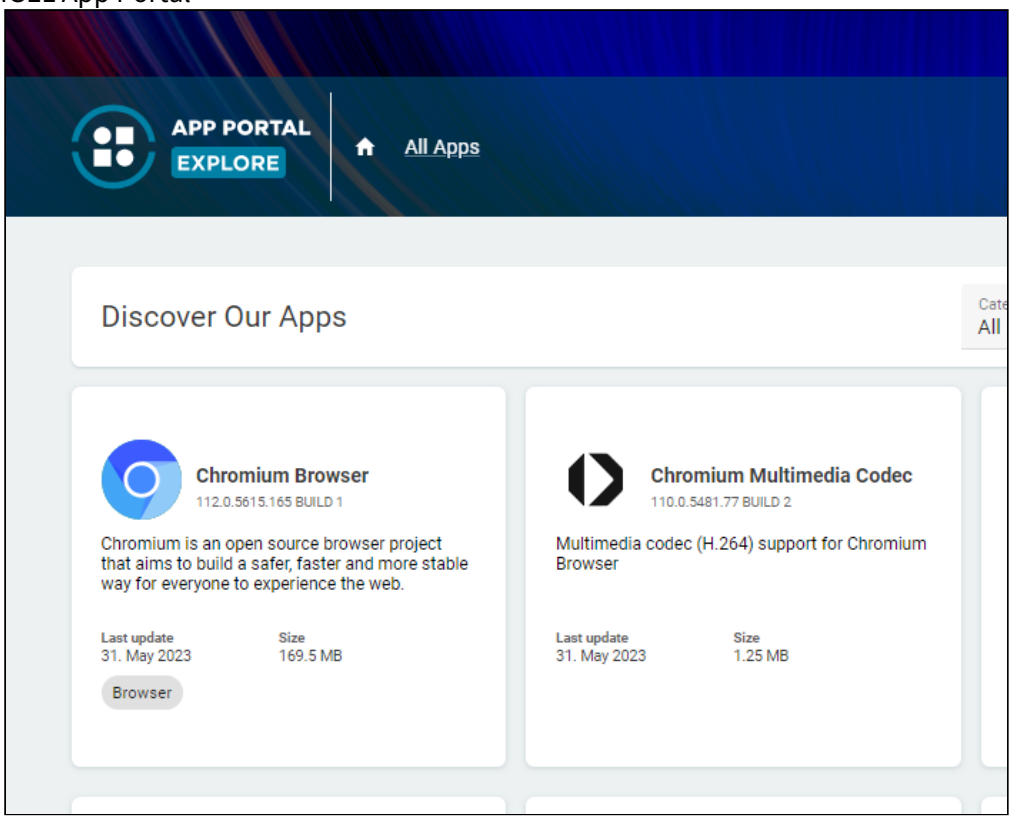
- `"app"`: Icon of any color that will be displayed in the App Portal and in the UMS
- `"monochrome"`: Icon that will be used in future versions when theming is implemented

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

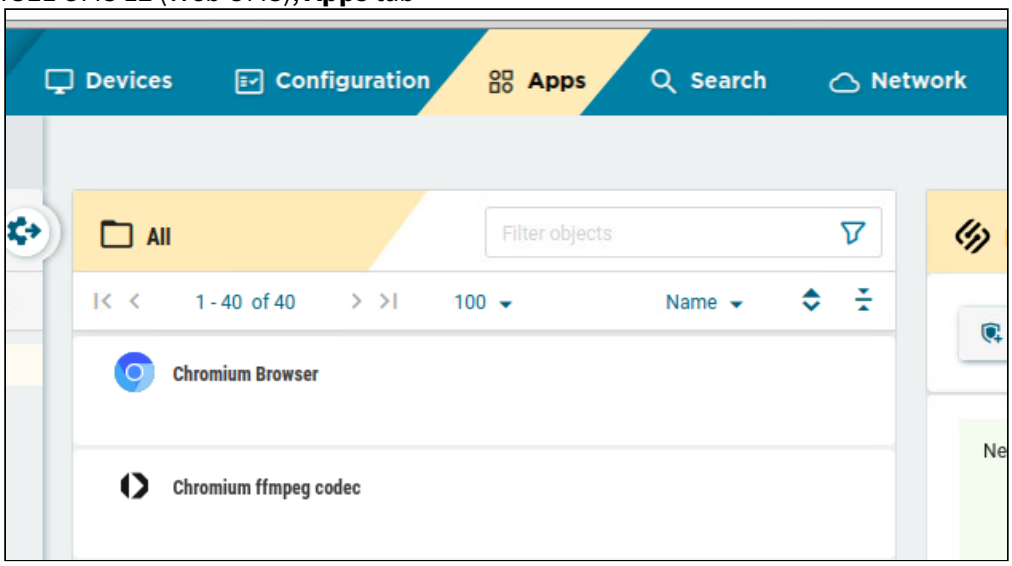
```
"icons": {
  "app": "app.svg",
  "monochrome": "monochrome.svg"
},
```

The icon is displayed at the following locations:

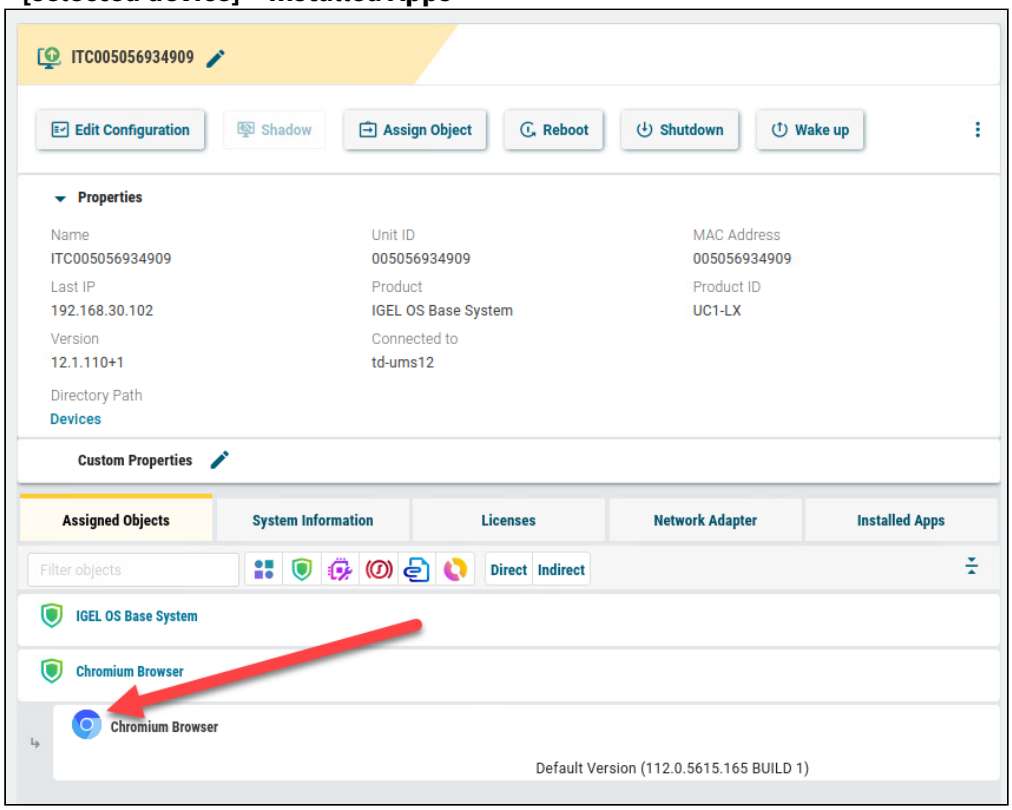
- IGEL App Portal

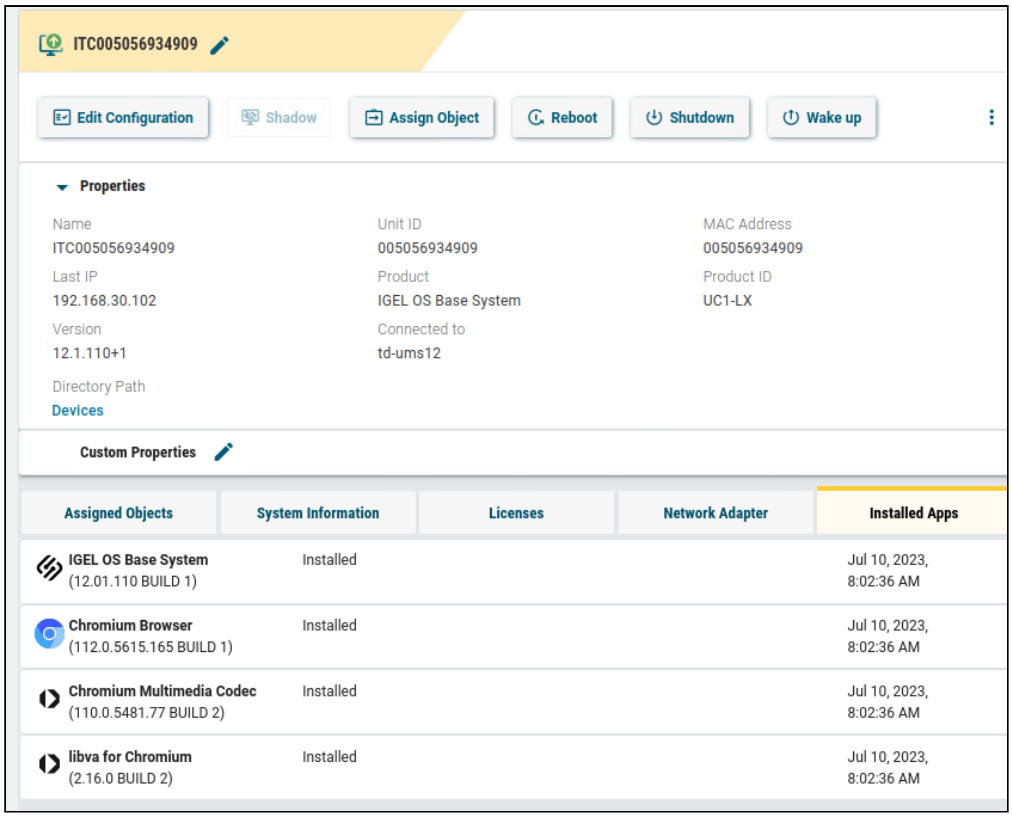


- IGEL UMS 12 (Web UMS), **Apps** tab



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





5.3 Descriptions 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 English. Both descriptions will be displayed in the IGEL App Portal.

Please note the format: Short description, followed by an empty line, followed by the long description. See also the dummy text in `data/description/en`.

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

5.4 EULA

When your app requires a 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.

6 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.

6.1 Defining the Sources in igel/thirdparty.json

6.1.1 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/.

6.1.2 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.

6.1.3 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"
         }
      ]
   }
]
```

6.2 Refining Your Selection in igel/install.json

6.2.1 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.

6.2.2 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`.

6.2.3 Example

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

6.3 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

7 Handling Files and Partitions

7.1 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.

7.1.1 Required Fields

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

7.1.2 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. What are the default permission, i.e. which permissions are valid when nothing is specified here?
- `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.

7.1.3 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 22).

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

7.2 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`.

The following sizes can be defined:

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

7.2.1 Example

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

```
"rw_partition": {
      "size": "large"
  }
```

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
  }
]
```

8 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.

8.1 /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`

8.2 .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.

8.3 .scripts/

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

8.4 .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.

8.5 ./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.

8.6 ./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.

8.7 .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.

8.8 .config/

This directory contains the configuration data for the app.

8.9 .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 21).

8.10 .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. TODO How to do this?

9 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.

9.1 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
}
```

10 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, `systemd` service files need to be deployed to the default `systemd` paths. The service can be enabled via the installation script of the app. This is described by the examples below.

10.1 Example 1: System Service That Will Run as Root

The following example shows how to define a system service that will run as the user `root` on an IGEL OS device.

10.1.1 Unit Configuration File

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

1. Create a file according to the following minimal 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 `input/all/etc/systemd/system/<name>.service`

10.1.2 Install Script

- In your app's data structure, edit `igel/install.sh` according to the following example:

```
#!/bin/bash
enable_system_service <name>.service
```

10.2 Example 2: System Service That Will Run as User

The following example shows a system service that will run as the user `user`. The example is derived from the authentication manager for the Citrix Workspace App (CWA).

10.2.1 Unit Configuration File

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

The unit configuration file

```
[Unit]
Description=Citrix Workspace App - AuthManagerDaemon

[Service]
Type=simple
ExecStart=/services/cwa/opt/Citrix/ICAClient/AuthManagerDaemon
```

In the app's data structure, this unit configuration file might be saved under `input/all/etc/systemd/user/cwa-authmanager.service`

10.2.2 Install Script

In `igel/install.sh`, a corresponding is added to run the system service as `user`:

```
#!/bin/bash
enable_system_user_service cwa-authmanager.service
```

11 Building Your App

11.1 Checking the Metadata

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

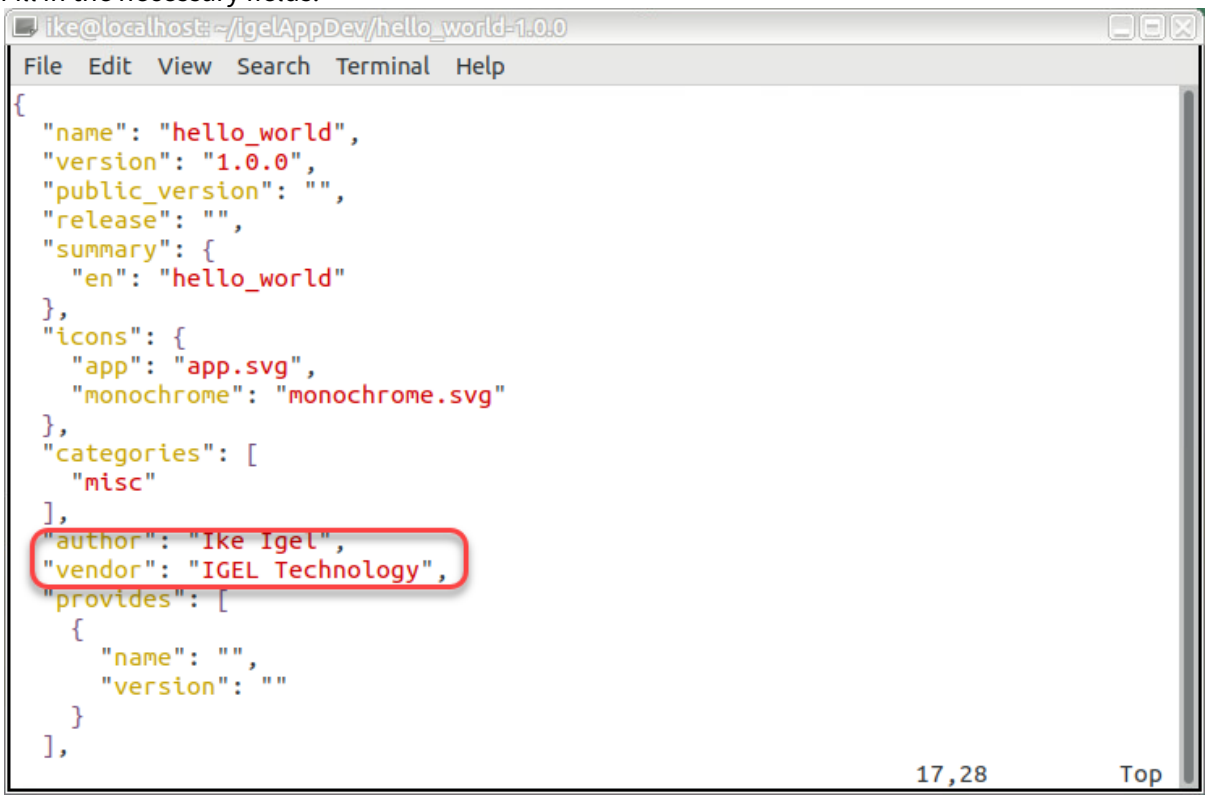
As a minimum requirement for a valid app, the fields `author` and `vendor` must be filled out.

11.1.1 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.



3. Save `app.json`

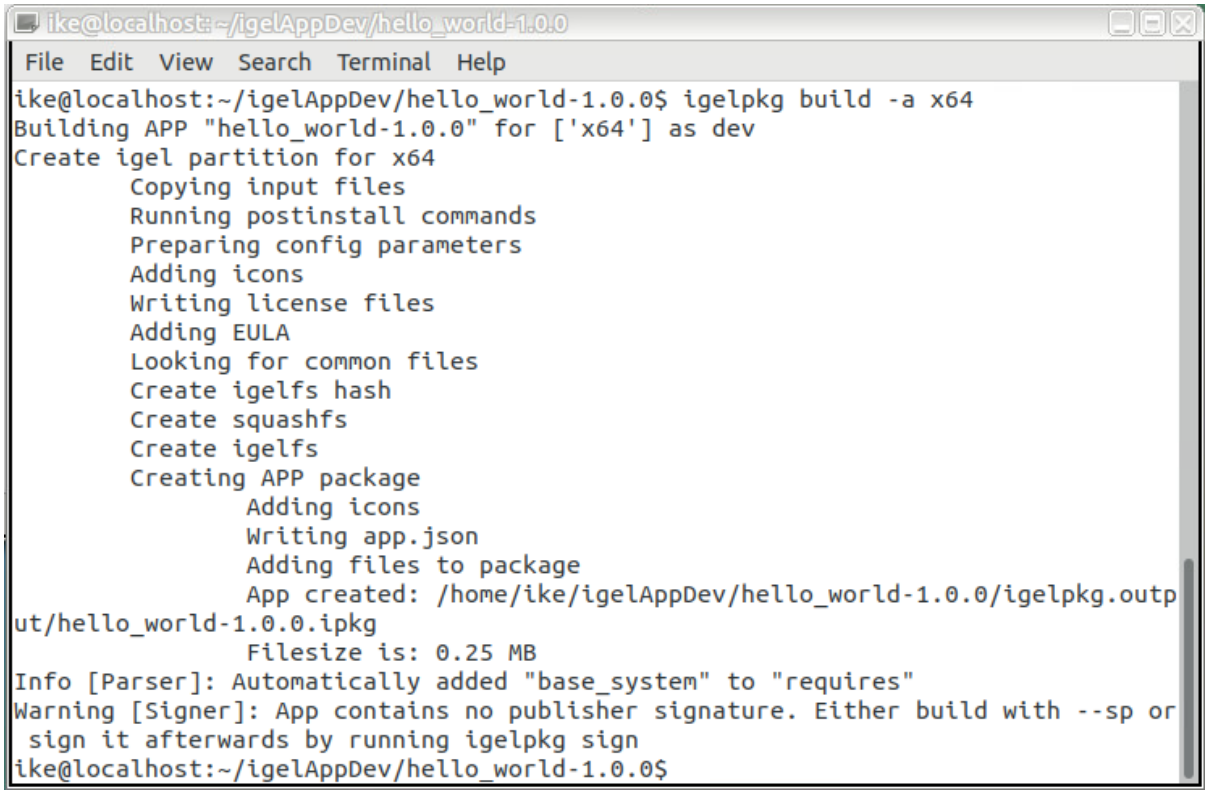
11.2 Building the App

11.2.1 Hello World Example

- In your app's directory, enter:

```
igelpkg build -a x64
```

The result should look 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`

11.3 Next Steps

When you have successfully created your app package, you can sign it and submit it to IGEL for evaluation. When the evaluation has been successful, your app is distributed via the IGEL OS App Portal.

12 Using the Keys and Configuration Provided in the IGEL OSC for Signing Your Apps

Your OSC zip file (SDK variant) contains the ISO file to boot your endpoint device with IGEL OS 12, along with the keys and a configuration file for signing your IGEL OS apps.

1. Put the OSC zip file on your development workstation and unzip it.

   ```
   unzip osc_sdk_12.01.120.1.zip
   ```

   The output should look like this:

   ```
   ike@td-ums12:~/Downloads$ unzip osc_sdk_12.01.120.1.zip
   Archive:  osc_sdk_12.01.120.1.zip
      creating: preparestick/
     inflating: preparestick/osc_sdk12.01.120.1.iso
    extracting: preparestick/licenses_osc_sdk.zip
     inflating: preparestick/readme12.01.120.1.txt
     inflating: preparestick/DiscUtils.dll
     inflating: preparestick/dd.exe
     inflating: preparestick/help.txt
     inflating: preparestick/license.txt
     inflating: preparestick/preparestick.exe
     inflating: preparestick/preparestick.exe.config
     inflating: preparestick/readme.txt
     inflating: IGEL_OS_12_SDK
     inflating: IGEL_OS_12_SDK.pub
     inflating: sign.json
   ike@td-ums12:~/Downloads$
   ```

   The files in the directory `preparestick/` are used for creating a bootable USB stick; for further information, see https://kb.igel.com/igelos-11.07/en/igel-os-creator-manual-57335174.html.
   The following files will be used for signing the app:
   `IGEL_OS_12_SDK`: Private key
   `IGEL_OS_12_SDK.pub`: Public key
   `sign.json`: Configuration file that tells `igelpkg` where the keys are to be found.

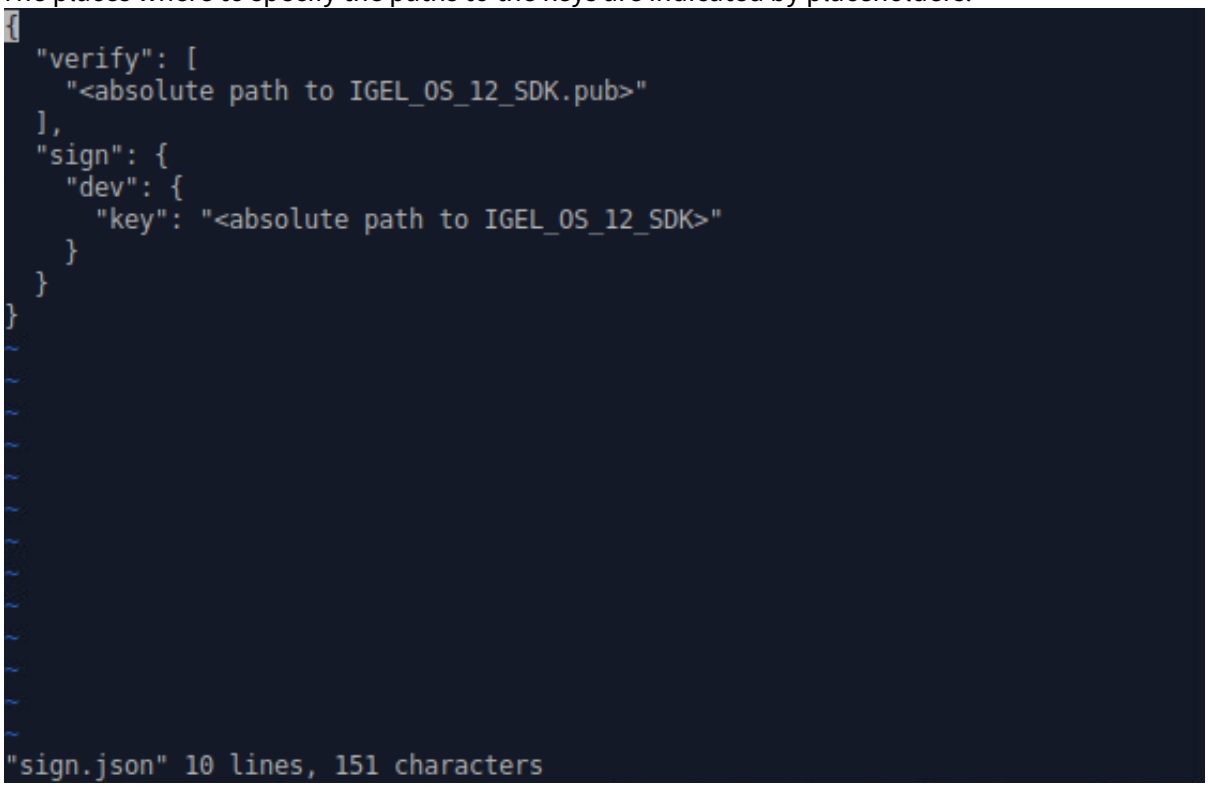
2. Create a directory for the keys, e.g. under `/usr/share/igelpkg/`, and copy the key files into it.

   ```
   sudo mkdir /usr/share/igelpkg/keys
   sudo cp IGEL_OS_12_SDK /usr/share/igelpkg/keys
   sudo cp IGEL_OS_12_SDK.pub /usr/share/igelpkg/keys
   ```

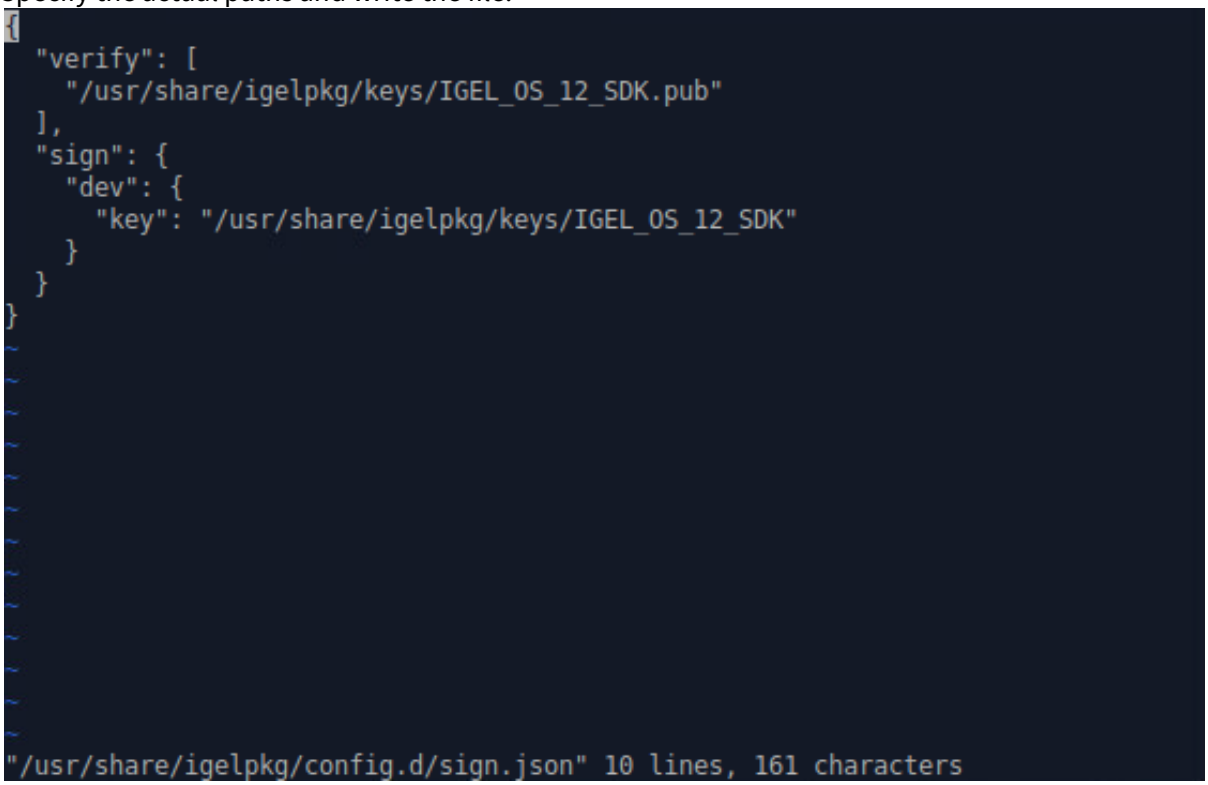
3. Open `sign.json` in a text editor of your choice.

   ```
   sudo vi sign.json
   ```

The places where to specify the paths to the keys are indicated by placeholders.



4. Specify the actual paths and write the file.



5. Copy `sign.json` to the appropriate directory.

```
sudo cp sign.json /usr/share/igelpkg/config.d
```

You can now sign your app with your own public/private key pair; for details on signing your app, see Signing Your App(see page 35).

13 Signing Your App

The igelpkg tool must be able to sign the app with a public/private key pair. For this purpose, the SDK package you obtained from IGEL contains a certificate you can use to sign your app.

Alternatively, you can use your own keys.

To sign an app you have already built:

1. Change into the app's directory, if you are not already there.

```
cd hello_world-1.0/
```

2. Sign the app.

```
igelpkg sign -a -p igelpkg.output/hello_world-1.0.0.ipkg
```

14 Review Criteria

14.1 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 21).
- 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

14.2 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 18).
- 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.

14.3 Signature

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

14.4 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`).

14.5 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.

14.6 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.

14.7 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/

14.8 AppArmor

- 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.

14.9 Logging

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

14.10 Metadata

- The metadata of the app MUST provide a minimal base system version in the

14.11 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.

15 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

15.1 Creating the Package Structure

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

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

15.2 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:

   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 18). 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.

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

15.3 Providing the Mandatory Metadata

To have a minimal set of metadata, we must provide the author and the vendor of the package.

- Open `supertuxkart-1.1.0/app.json` and fill in the `author` and the `vendor` field.

```
{
  "author": "IGEL",
  "categories": [
    "misc"
  ],
  "conflicts": [
    {
      "name": "",
      "version": ""
    }
  ],
  "icons": {
    "app": "app.svg",
    "monochrome": "monochrome.svg"
  },
  "name": "supertuxkart",
  "provides": [
    {
      "name": "",
      "version": ""
    }
  ],
  "public_version": "",
  "release": "",
  "requires": [
    {
      "name": "",
      "version": ""
    }
  ],
  "suggests": [
    {
      "name": "",
      "version": ""
    }
  ],
  "summary": {
    "en": "supertuxkart"
  },
  "vendor": "Acme",
  "version": "1.1.0"
}
```

15.4 Building the App, First Try

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

```
igelpkg build -r focal
```

15.5 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 IGEL OS Base System version.
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`

15.6 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"
  }
]
```

15.7 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.

15.8 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"
  }
]
```

15.9 Another Build

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

```
igelpkg build -r focal
```

15.10 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`

15.11 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"
```

```
    }
]
```

15.12 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

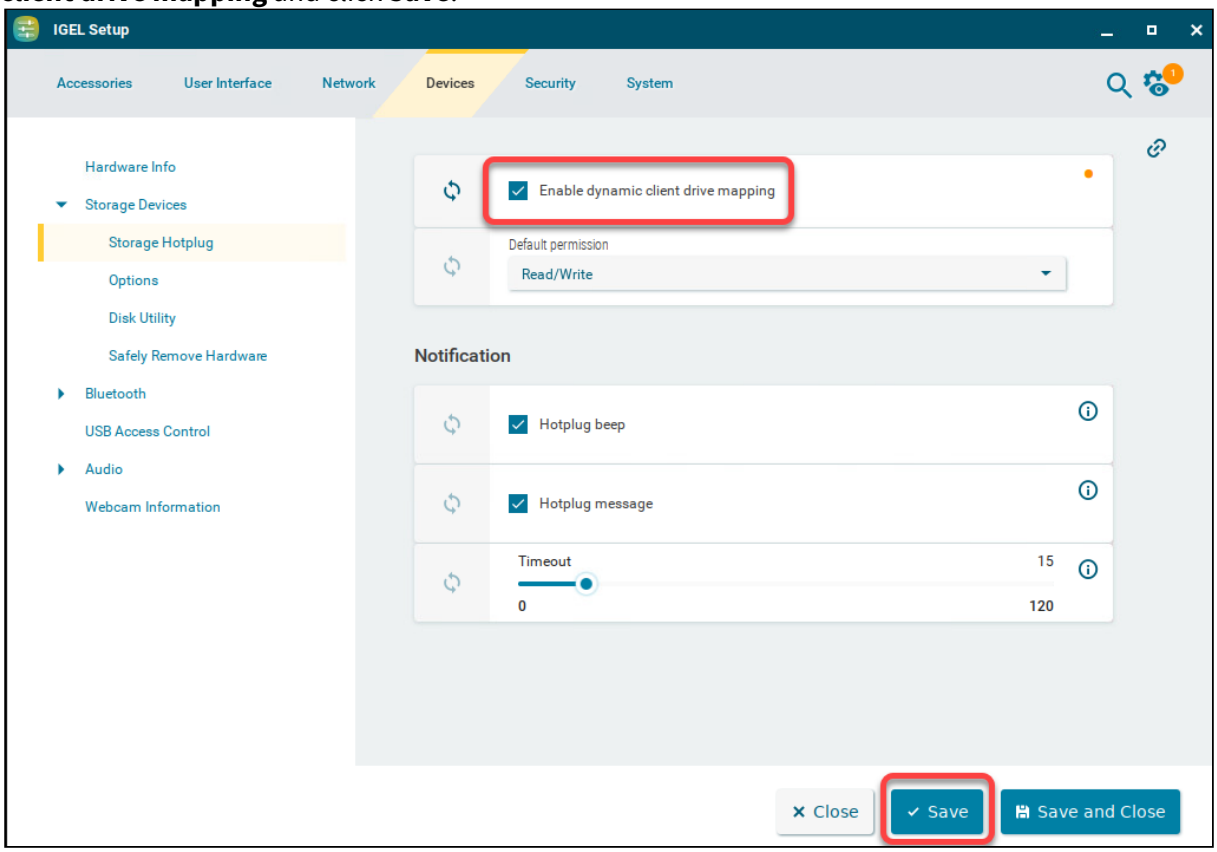
```
igelpkg build -r focal -sp
```

15.13 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).

15.13.1 USB Flash Drive

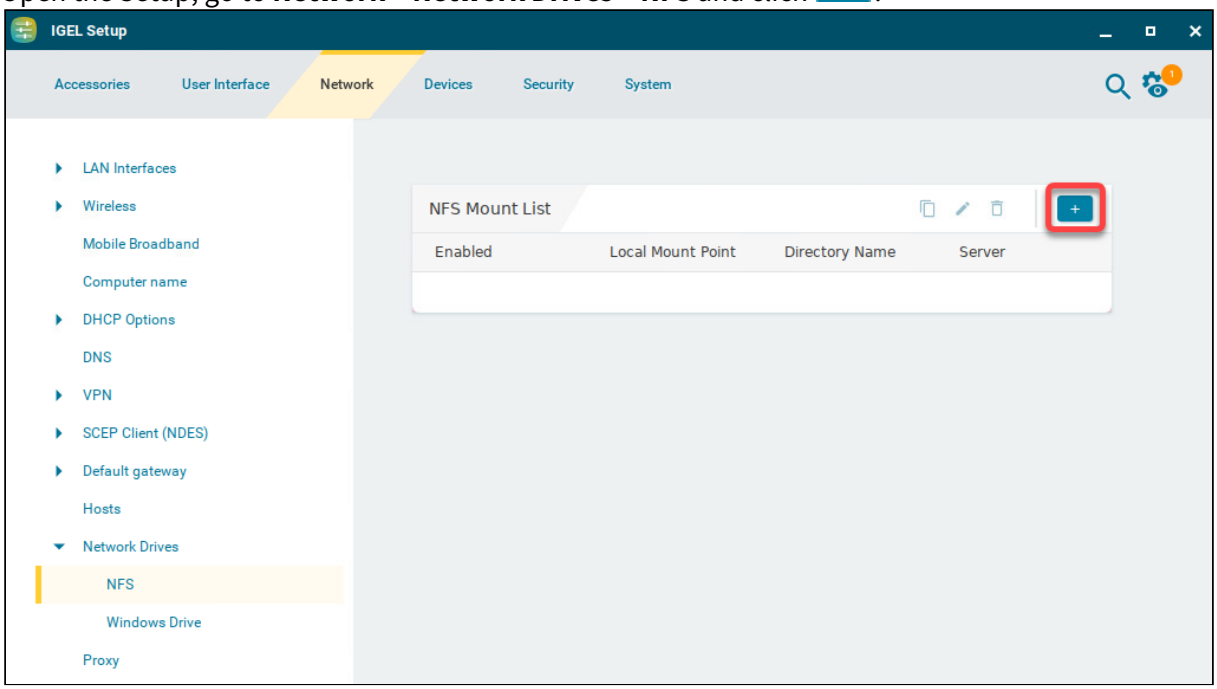
1. Open the Setup, go to **Devices > Storage Devices > Storage Hotplug**, activate **Enable dynamic client drive mapping** and click **Save**.



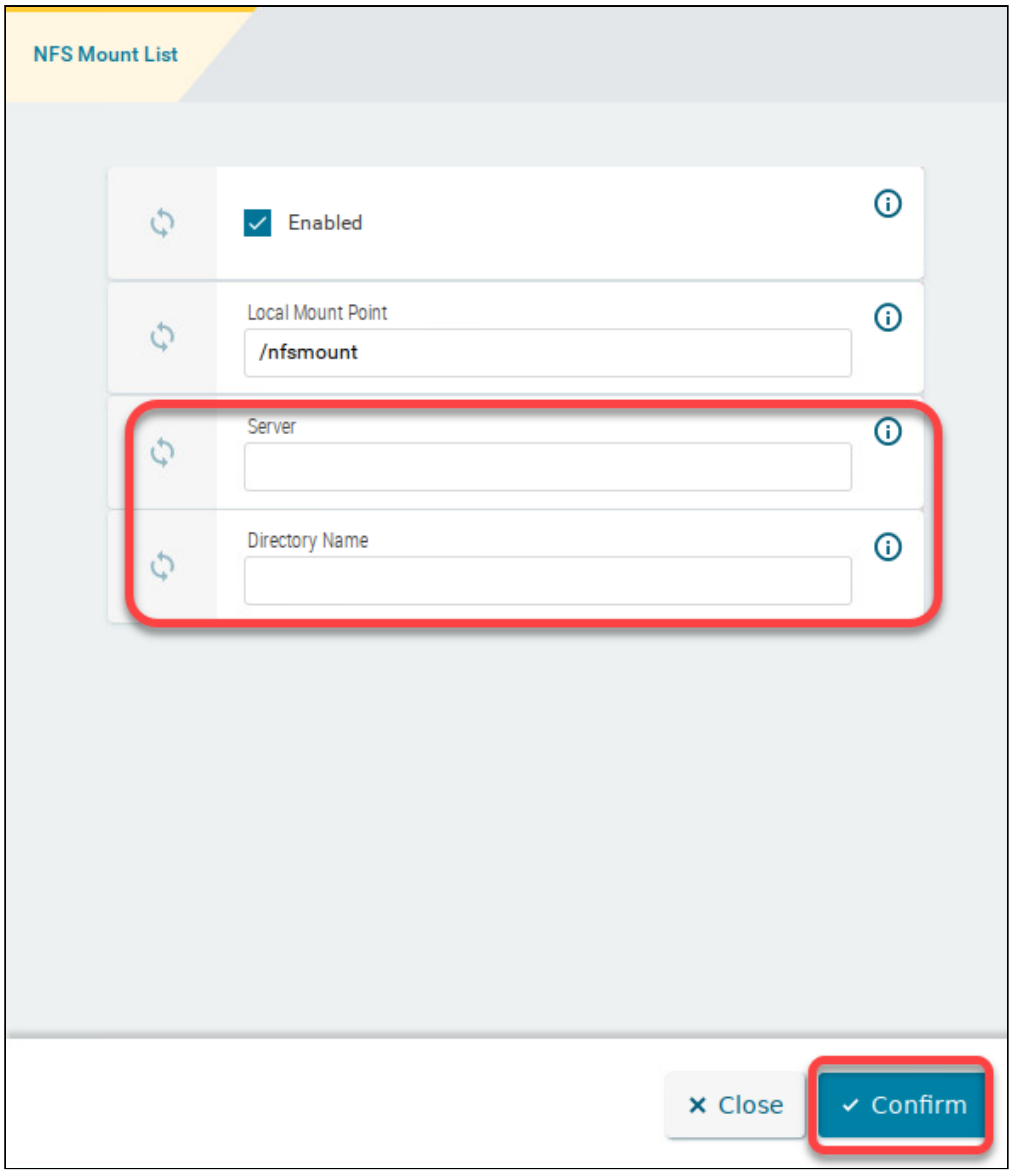
2. Plug the USB flash drive into the device.
   The USB flash drive is mounted under `/userhome/media`

15.13.2 NFS Drive

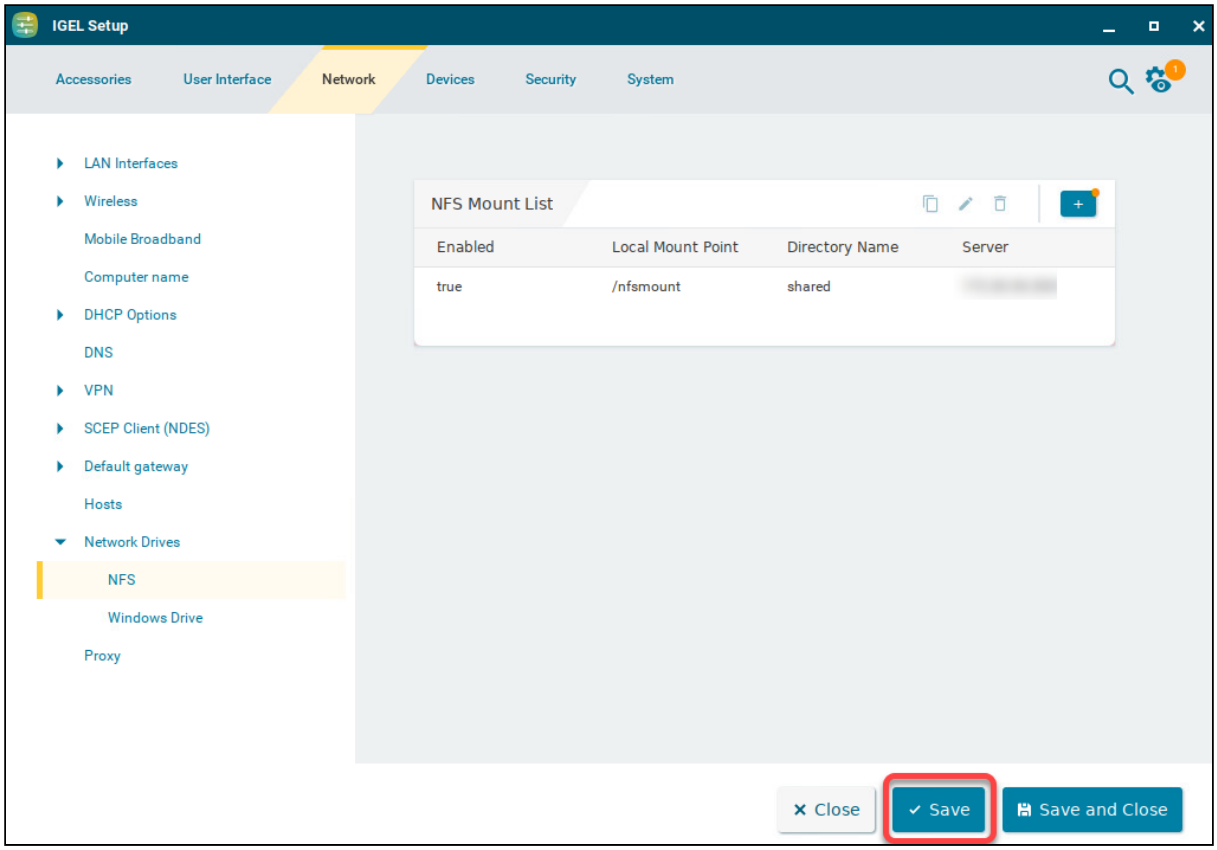
1. Open the Setup, go to **Network > Network Drives > NFS** and click  .



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

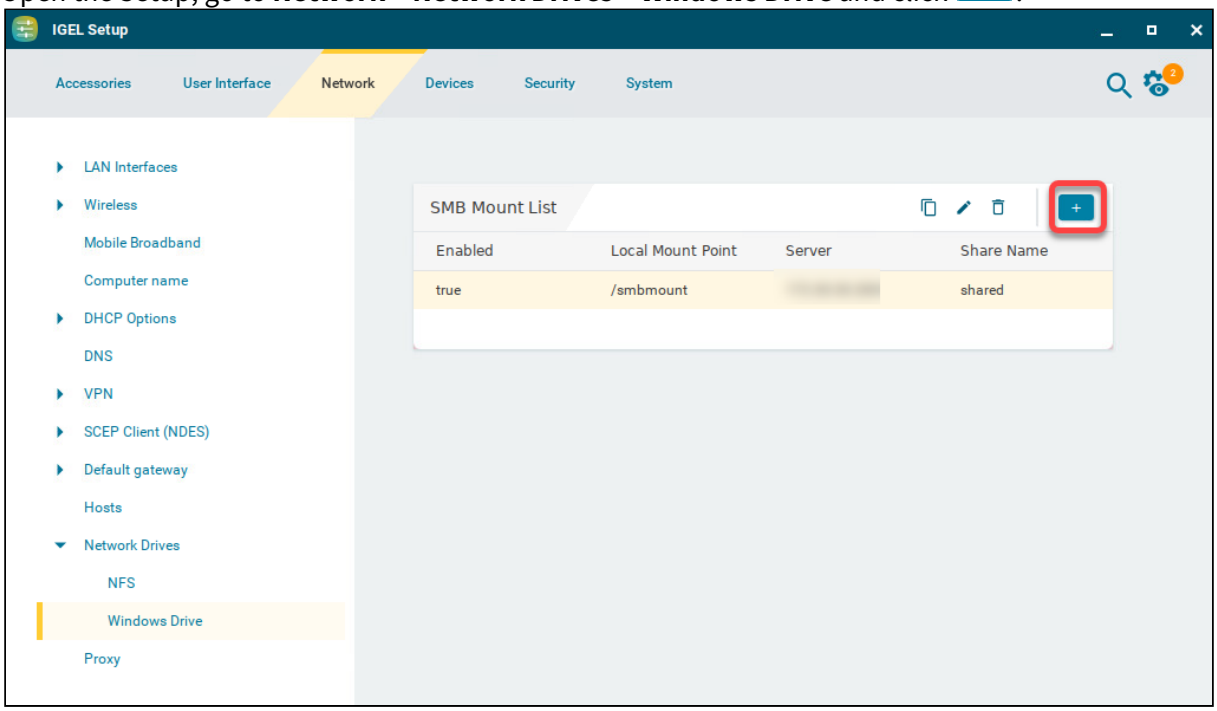
3. Click **Save**.



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

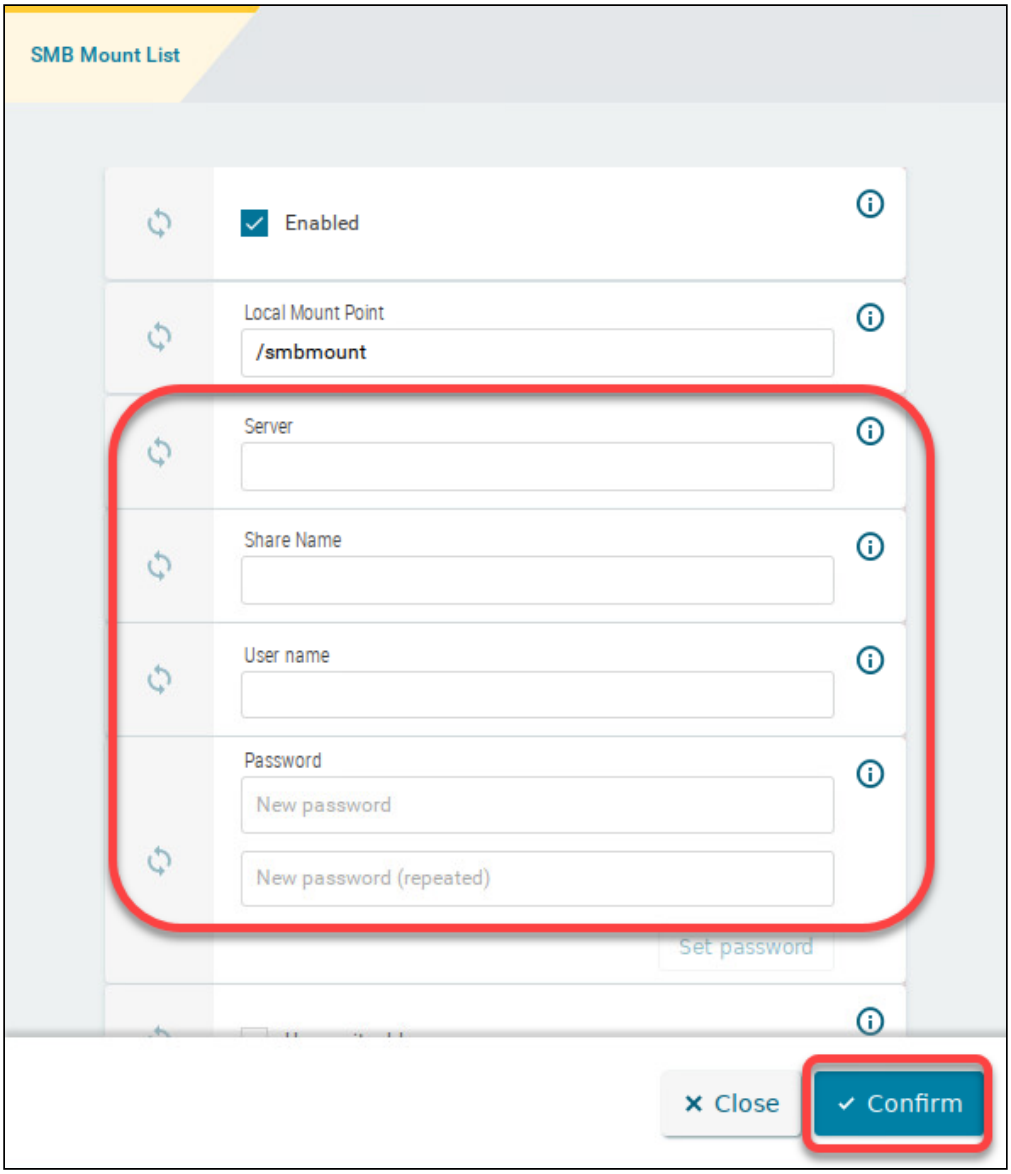
15.13.3 Windows Drive

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

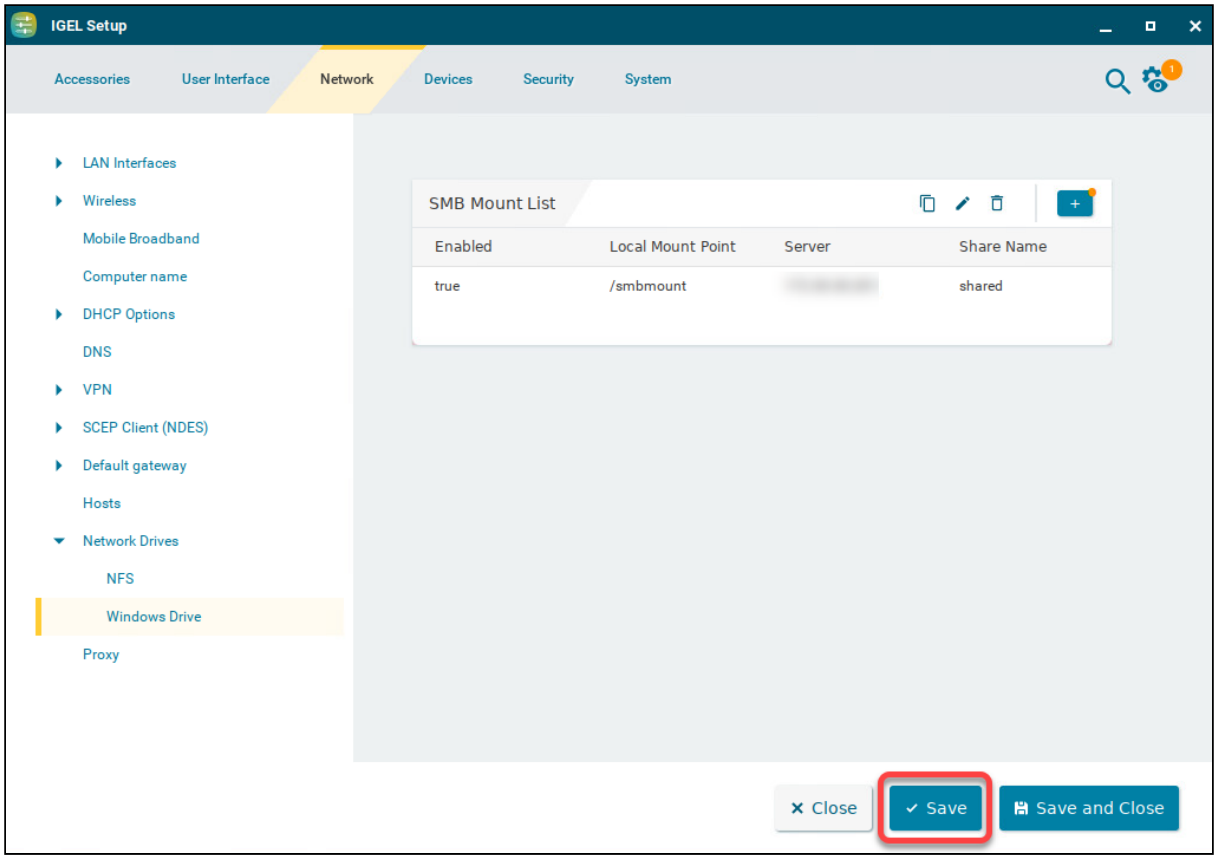


2. Enter the following data and afterward click **Confirm**:
   - **Server**: Address of the SMB server
   - **Share name**: Name of the shared directory
   - **Username**: Name of the user with access to the share

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



3. Click **Save**.



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

15.13.4 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
```

15.14 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                :~$
```

15.15 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"
  }
]
```

15.16 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
```

15.17 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
```

15.18 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.

15.19 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.

15.19.1 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>
```

15.19.2 Creating a Start Script

CAVEAT! The procedure described in this section will not work with the current version of igelpkg. See https://jira.igel.com/browse/LX8-6276

UPDATE: There is a workaround, see DOC-3519[1], comments. The issue will be fixed in 0.9.8

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:

1 https://jira.igel.com/browse/DOC-3519

```
#!/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
```

15.20 Building, Installing, and Testing the Refined App

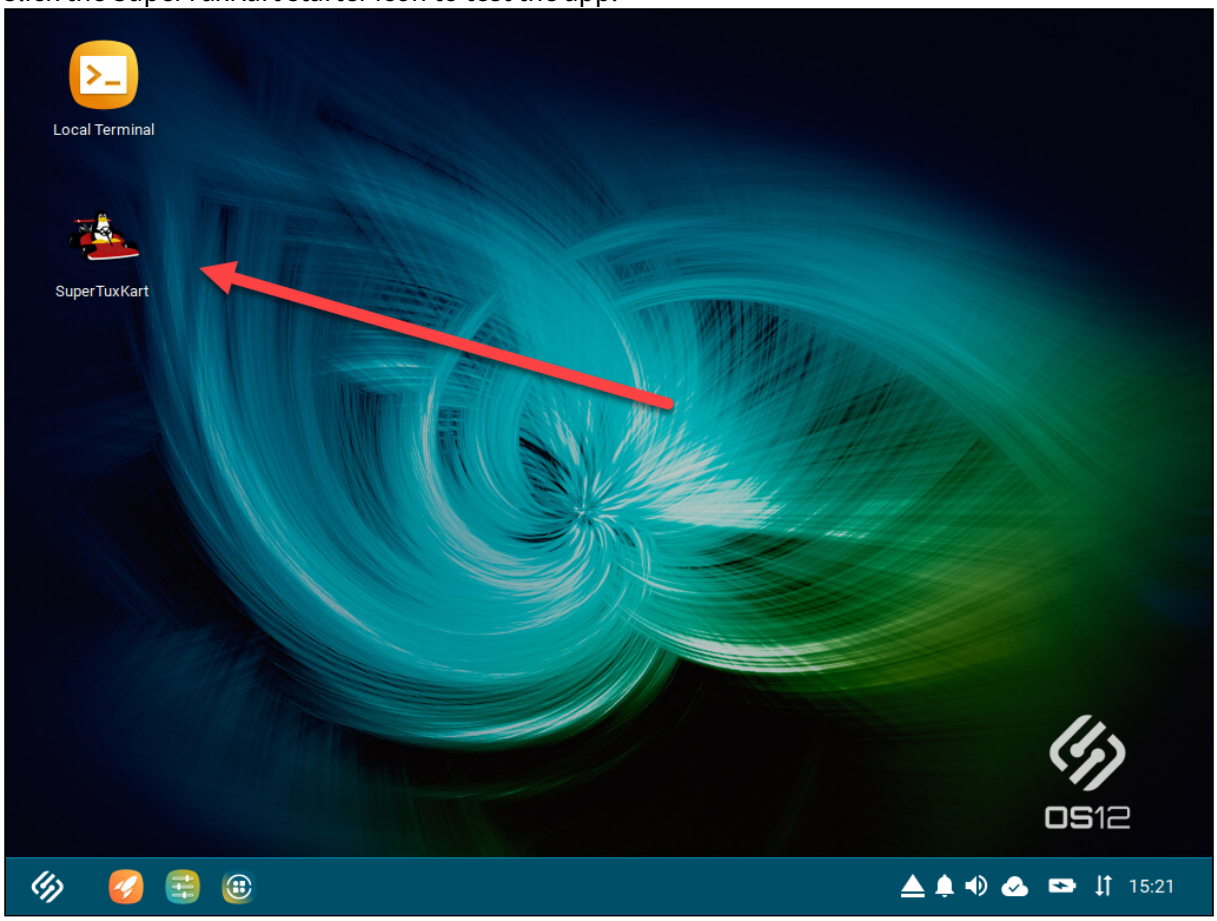
1. Build and sign the app.

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

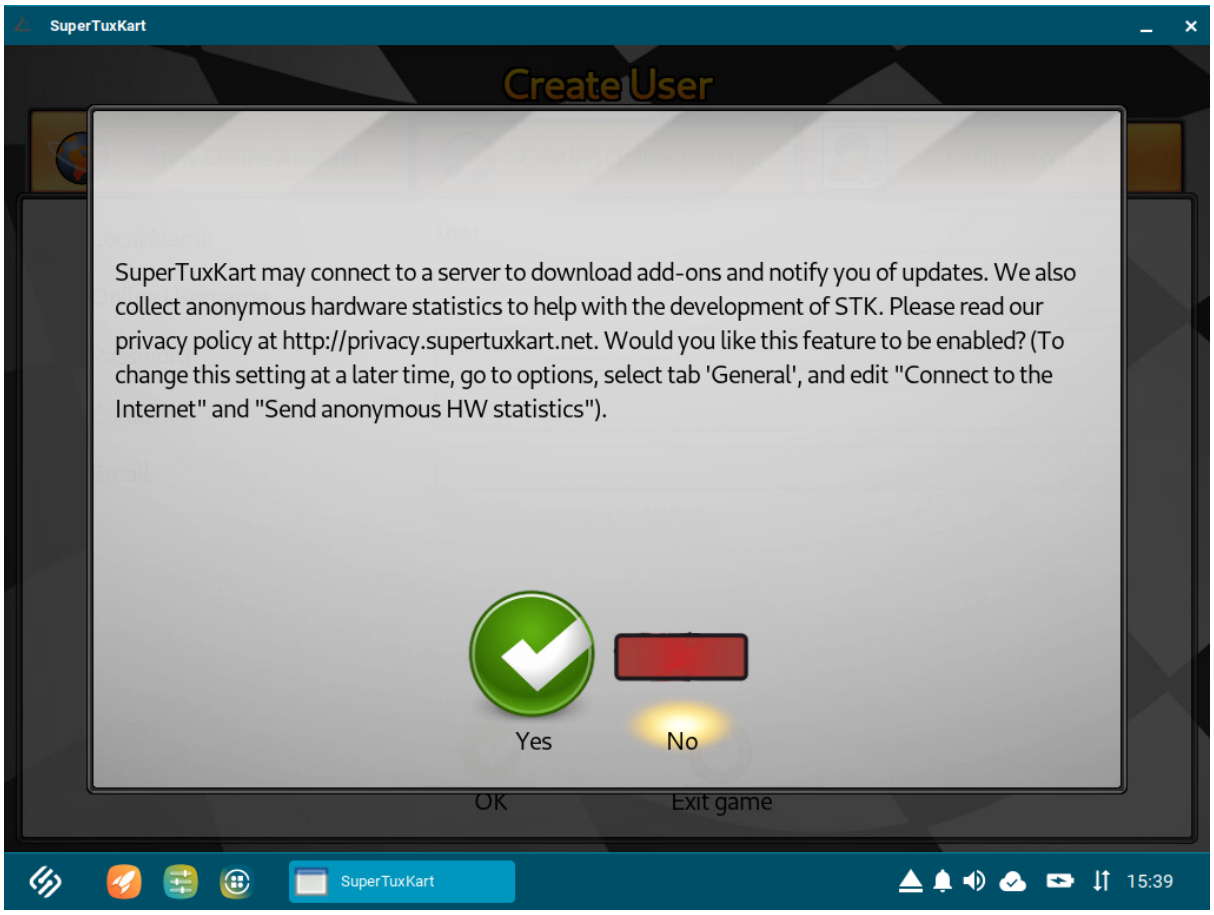
2. Reinstall the app.

```
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.



15.21 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 must be present in the `data/` directory.

- Copy the icon that came with the Debian package to the `data/` directory:

```
cp igelpkg.output/data_x64/usr/share/icons/hicolor/128x128/apps/supertuxkart.png
data/app.png
```

16 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.

16.1 Creating the Package Structure

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

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

16.2 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 fill in the following fields:

- `author`
- `vendor`
- `summary > en`
- `categories`

```
{
  "author": "IGEL Technology GmbH",
  "categories": [
    "Unified Communication"
  ],
  "conflicts": [
    {
      "name": "",
      "version": ""
    }
  ],
  "icons": {
    "app": "app.svg",
    "monochrome": "monochrome.svg"
  },
  "name": "ms_teams_pwa",
  "provides": [
    {
      "name": "",
```

```
      "version": ""
    }
  ],
  "public_version": "",
  "release": "",
  "requires": [
    {
      "name": "",
      "version": ""
    }
  ],
  "suggests": [
    {
      "name": "",
      "version": ""
    }
  ],
  "summary": {
    "en": "Microsoft Teams Progressive Web App"
  },
  "vendor": "Microsoft Corporation",
  "version": "1.0.0"
}
```

16.3 Defining Dependencies

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

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

```
"requires": [
    {
      "name": "chromium",
      "version": ">=114.0.5735"
    }
```

16.4 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`

To have enough space, we will use `large`.

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

```
"rw_partition": {
      "size": "large"
  }
```

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:

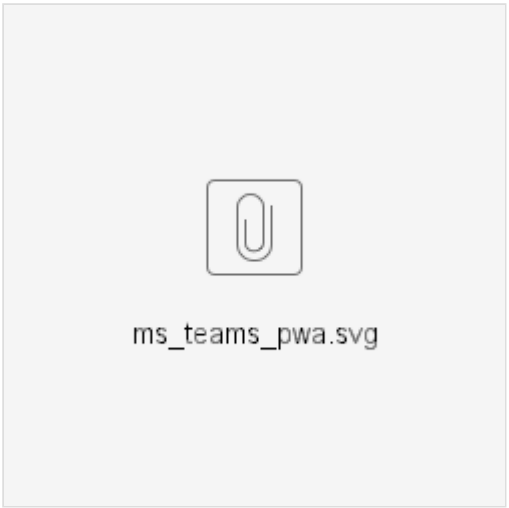
```
[
  {
    "path": "/userhome/.config/ms-teams",
    "owner": "777:100",
    "permissions": "",
    "persistent": true
  }
]
```

16.5 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. You can use SVG or PNG as the image format. For our example, you can download the icon file here:



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

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

16.6 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. Can we provide contact details?

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

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

The command line switches have the following meanings:
`--user-data-dir`: Chromium will use the defined path as its profile location. It will also 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`)
Should we add something about Chromium policies as described in LX8-6351?

4. Make the starter script executable:

```
chmod +x ms_teams_pwa0
```

16.7 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"
    }
}
```

16.8 Building and Signing the App

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

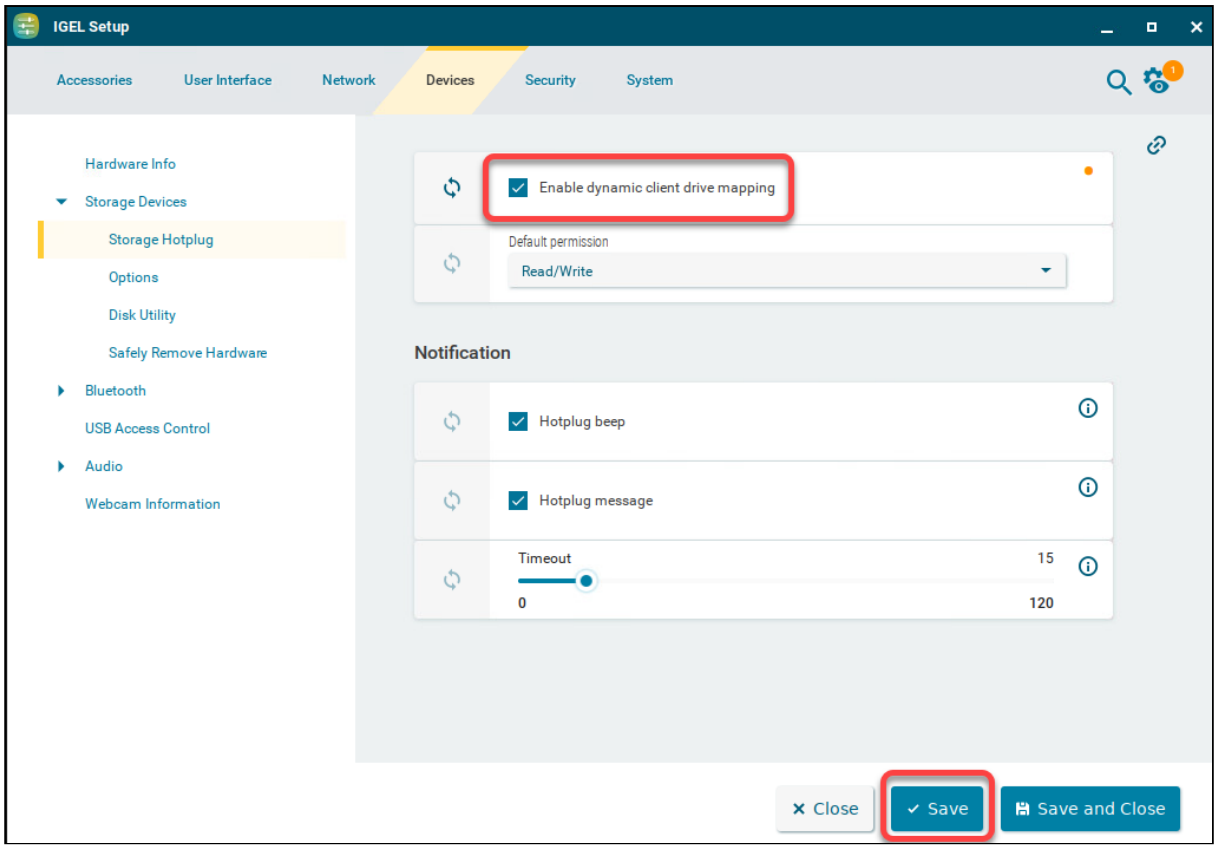
```
igelpkg build -r focal -sp
```

16.9 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.

16.9.1 USB Flash Drive

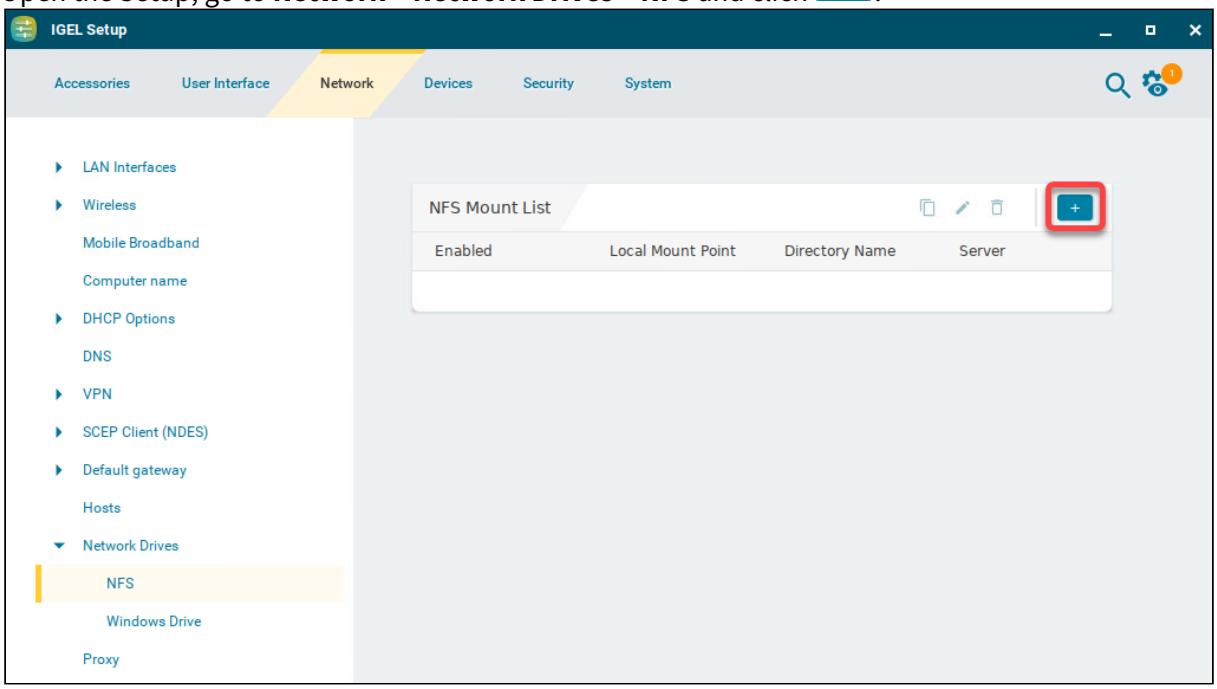
1. Open the Setup, go to **Devices > Storage Devices > Storage Hotplug**, activate **Enable dynamic client drive mapping** and click **Save**.



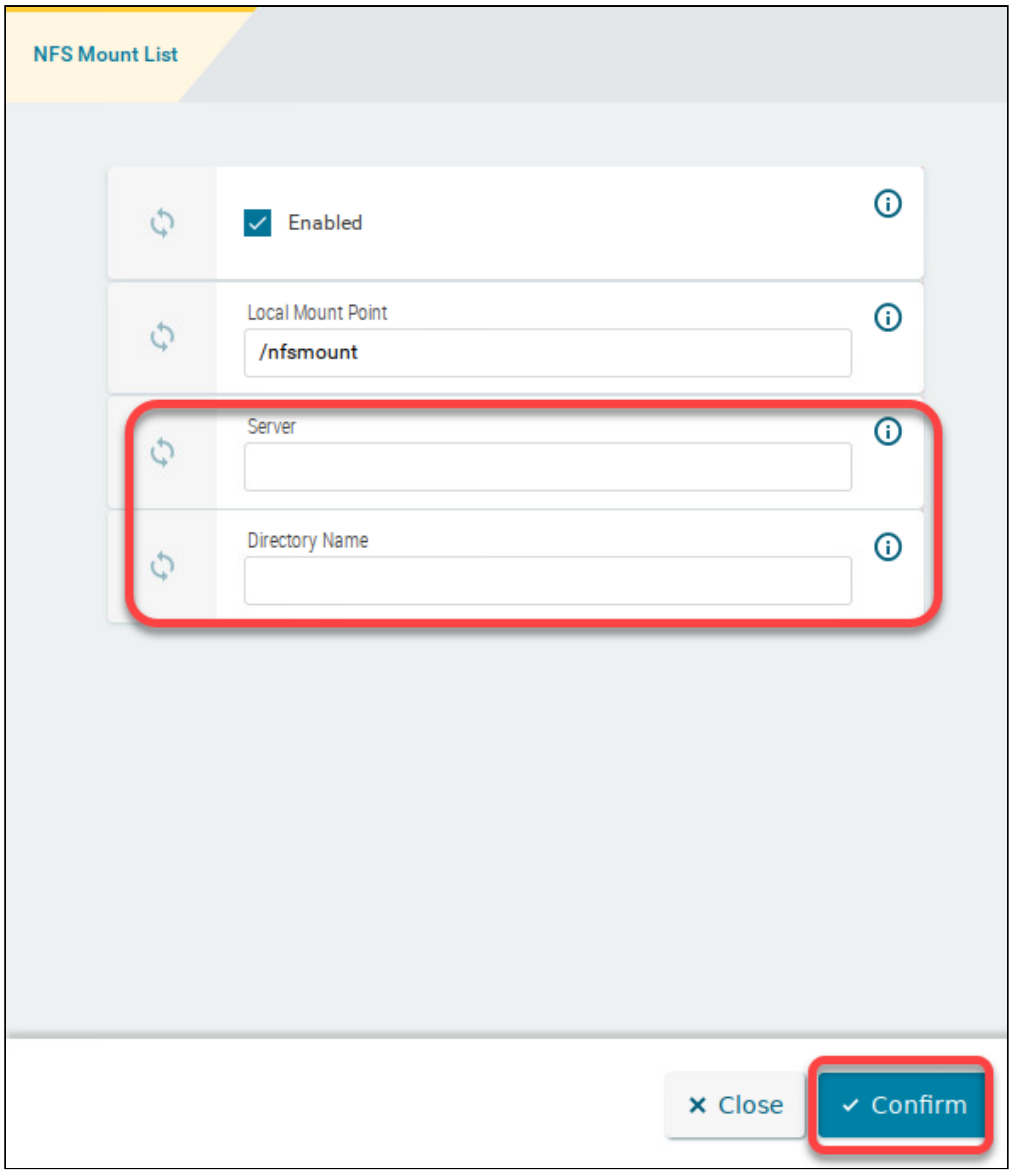
2. Plug the USB flash drive into the device.
   The USB flash drive is mounted under `/userhome/media`

16.9.2 NFS Drive

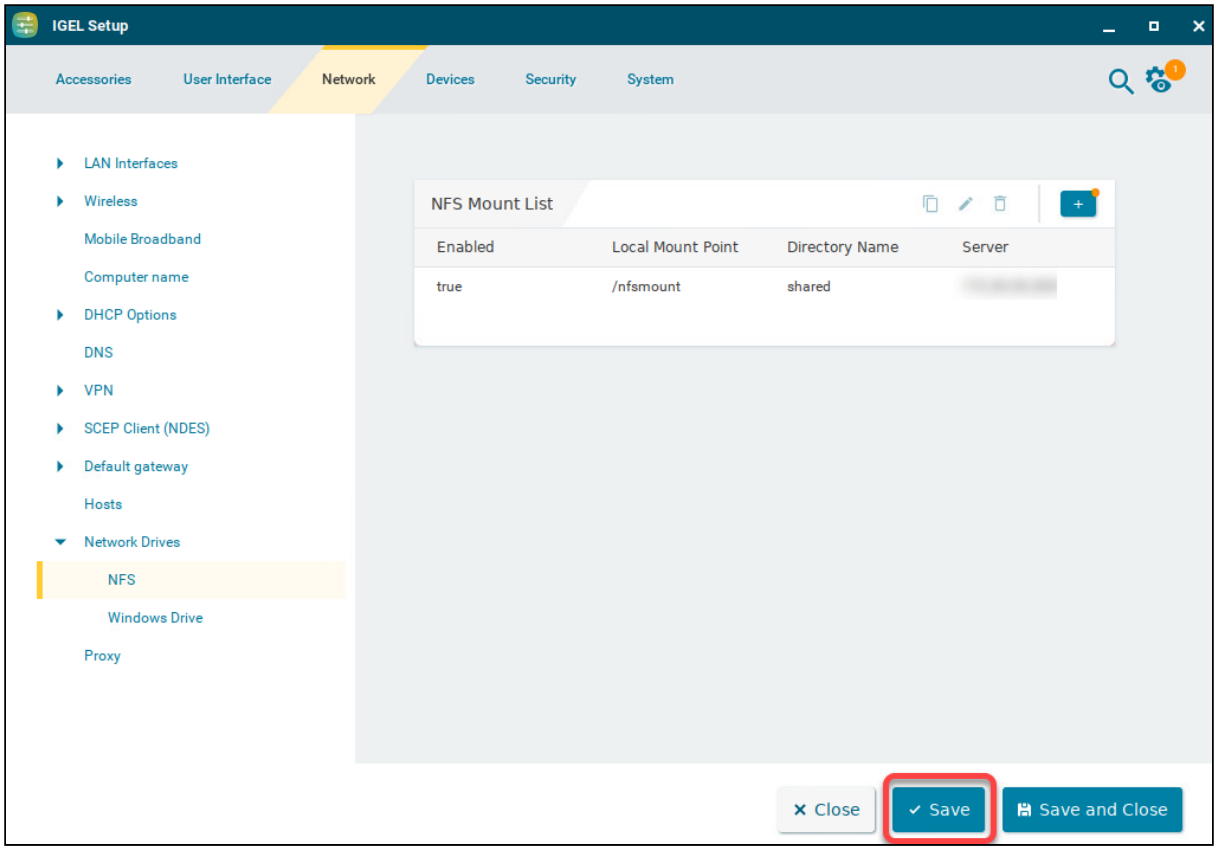
1. Open the Setup, go to **Network > Network Drives > NFS** and click +.



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

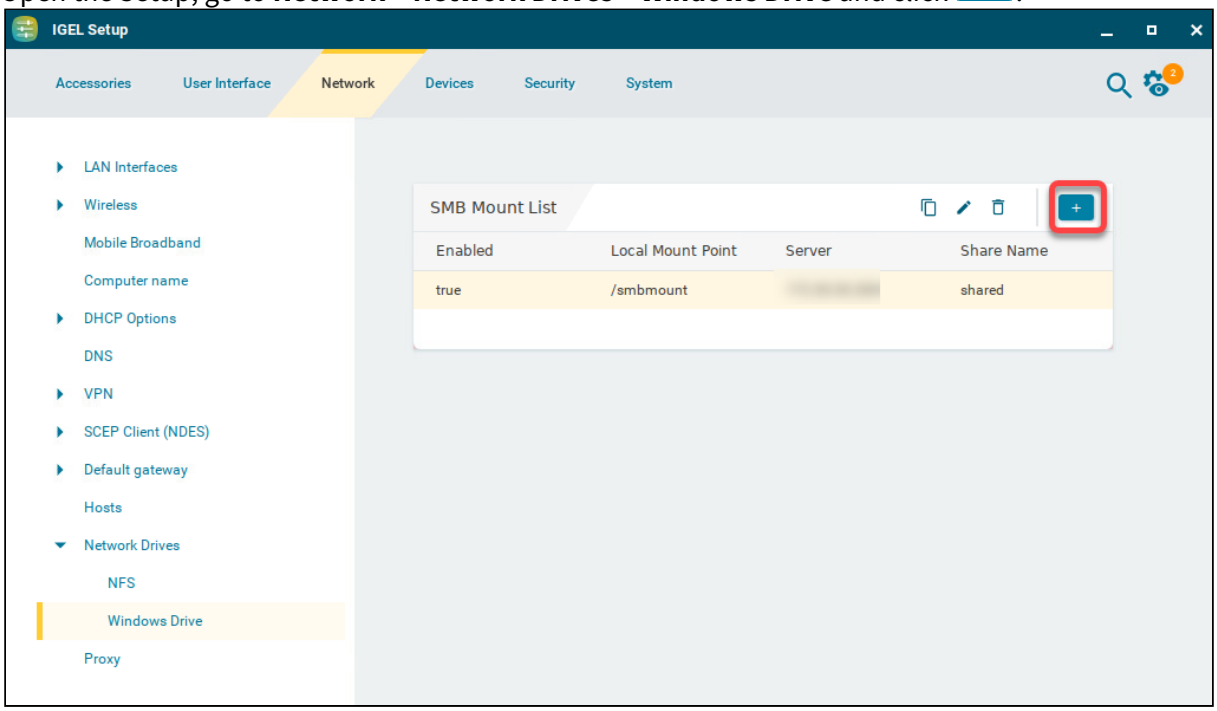
3. Click **Save**.



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

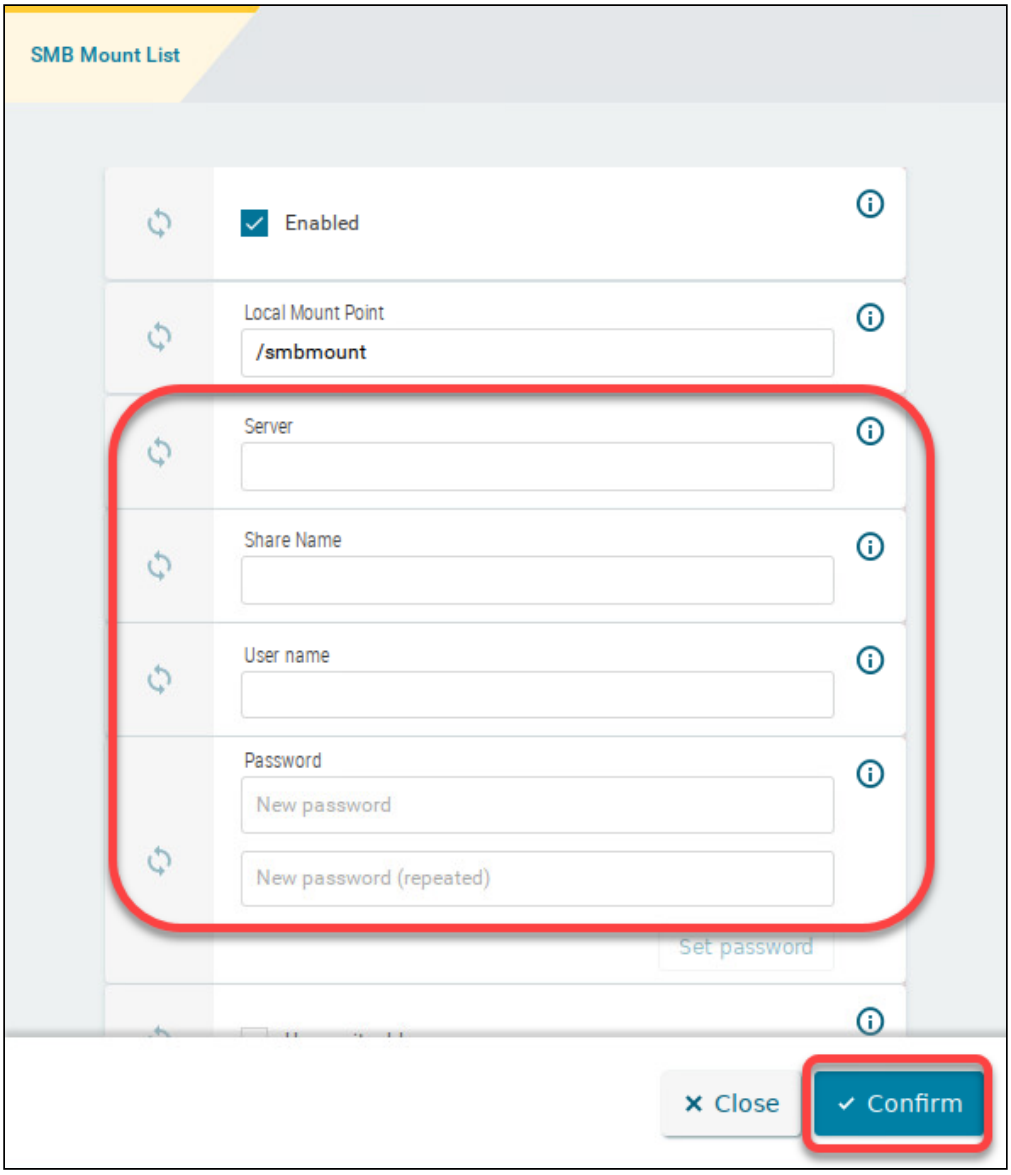
16.9.3 Windows Drive

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

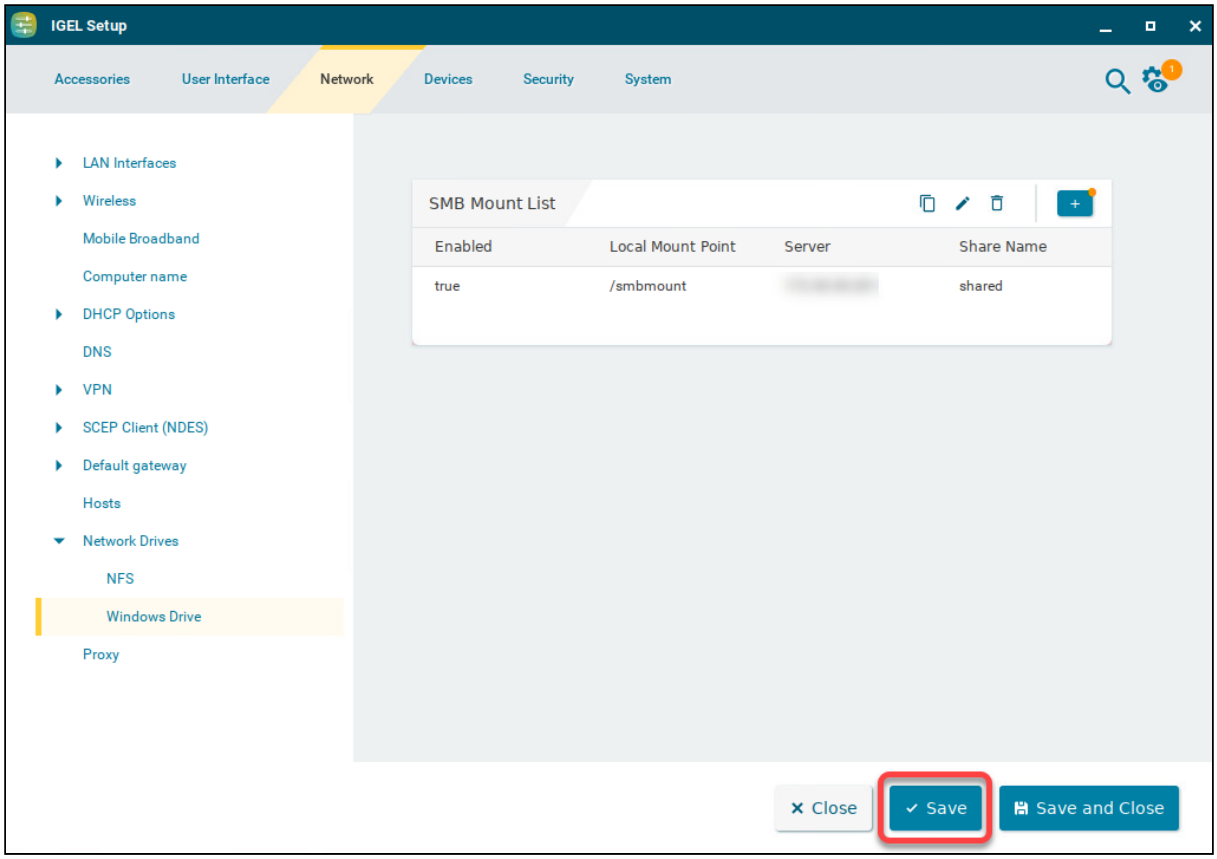


2. Enter the following data and afterward click **Confirm**:
   - **Server**: Address of the SMB server
   - **Share name**: Name of the shared directory
   - **Username**: Name of the user with access to the share

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



3. Click **Save**.



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

16.9.4 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>/ms_teams_pwa-1.0.0.ipkg
```

After a reboot, the app can be tested.

16.10 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