LocalCosmos App Kit Documentation

Thomas Uher

Aug 24, 2022

Contents:

CHAPTER 1

Creating Frontends

Local Cosmos offers you the possibility to create your own Frontend.

1.1 Structure of a Frontend

Your frontend needs to have the following directory structure:

```
└── www
    ├── localcosmos
    ├── online_content
    │   └── templates
    ├── fact_sheets
    │   └── templates
    ├── locales
    │   └── LANGUAGE_CODE
    │       └── plain.js
    ├── webapp
    ├── android
    ├── ios
    └── settings.json
```

1.1.1 www folder

The `www` folder will later represent your Apache Cordova www folder, containing all files of your (web)app.

1.1.2 localcosmos folder

The `localcosmos` folder is the folder where the Local Cosmos App Kit will put all generated contents, like identification keys, in. Do not put anything into this folder by yourself, unless you are developing using test content. During the build phase of your App on Local Cosmos, this folder will be deleted, recreated and auto-populated.

1.1.3 online_content folder

If the user creates Online Content in the app Kit, he has to select a template for his content. You supply those templates in the `online_content` folder of your frontend. Read *How to write Online Content and Fact Sheet templates* for more information on how to write templates.

1.1.4 fact_sheets folder

If the user creates Online Content in the app Kit, he has to select a template for his content. You supply those templates in the `fact_sheets` folder of your frontend. Read *How to write Online Content and Fact Sheet templates* for more information on how to write templates.

1.1.5 locales folder

Contains translations of your apps texts. For each language, a folder using its LANGUAGE_CODE is required. Example:

```
locales
├── de
│   └── plain.js
└── en
    └── plain.js
```

You only have to supply translations for button texts and such. The translations for the user-generated content like identification keys are made within the App Kit, and are placed in the locale directories automatically during build.

1.1.6 webapp folder

Holds specific files only for the webapp version of your app.

1.1.7 android folder

Holds specific files ony for the android version of your app.

1.1.8 ios folder

Holds specific files only for the ios version of your app.

1.1.9 settings.json

The settings file of your app. The settings.json file has two functions:

1. it defines which content the user can upload for the Frontend component on the Local Cosmos App Kit
2. During build, these settings are merged with settings provided by the Local Cosmos App Kit and then made available to your app at www/settings.json

Example settings.json file

```
{
    "frontend" : "FRONTEND_NAME",
    "PRIMARY_LANGUAGE" : "en",
    "version" : "0.1",
    "online_content" : {
        "verbose_template_names" : {
            "page/free_page.html" : {
                "de" : "Freie Seite",
                "en" : "Free Page"
            },
            "feature/news.html" : {
                "de" : "News Eintrag",
                "en" : "News entry"
            }
        },
        "max_flag_levels" : 2,
        "flags" : {
            "main_navigation" : {
                "template_name" : "flag/main_navigation.html",
                "name" : "Main Navigation"
            }
        }
    },
    "user_content" : {
        "images" : {
            "app_background" : {
                "restrictions" : {
                    "file_type" : ["jpg","jpeg","png"],
                    "ratio" : "10:16"
                },
                "help_text" : "Ratio: 10:16. Will be displayed using the app or␣
↪webapp on a smarthpone in portrait mode.",
                "required" : false
            },
            "app_launcher_icon" : {
                "restrictions" : {
                    "file_type" : ["svg"],
                    "ratio" : "1:1"
                },
                "help_text" : "Has to be an .SVG vector graphic AND the ratio must be␣
↪1:1. Will be displayed on your phones home screen to launch your app",
                "required" : true
            },
            "app_pc_background" : {
                "restrictions" : {
                    "file_type" : ["jpg","jpeg","png"],
                    "ratio" : "2:1"
                },
                "help_text" : "Will be displayed in the webapp using a pc",
                "required" : false
            },
            "logo" : {
                "restrictions" : {
                    "file_type" : ["svg"]
                },
                "help_text" : "Will be displayed on the apps main page",
                "required" : true
```

(continues on next page)

(continued from previous page)

```
            }
        },
        "texts" : {
            "about_app" : {
                "required" : false,
                "help_text" : "Describe your app"
            },
            "app_sources" : {
                "required" : false,
                "help_text" : "Sources you used to build this app."
            }
        }
    },
    "android" : {
        "launcher_icon" : {
            "type" : "user_content",
            "image_identifier" : "app_launcher_icon"
        },
        "splashscreen" : {
            "type" : "user_content",
            "image_identifier" : "app_splashscreen"
        }
    },
    "ios" : {
        "launcher_icon" : {
            "type" : "user_content",
            "image_identifier" : "app_launcher_icon"
        },
        "splashscreen" : {
            "type" : "user_content",
            "image_identifier" : "app_splashscreen"
        }
    }
}
```

The build process of Local Cosmos adds some settings to this file, like `PRIMARY_LANGUAGE`, according to the settings the app creator made in the app kit.

settings.online_content

settings.user_content

You can make your frontend configurable, so the app creator can choose a background, logo etc.

setting	description
user_content.images	Define the images which the app creator can upload using the App Kit. For example “background_image”
user_content.texts	Define the texts the app creator can create using the App Kit. For example “about_app”. Do not add legal notice as this is automatically added by the App Kit as a requirement.

settings.android

To successfully build an android app, several assets are required.

setting	description
android.launcher_icon	The icon shown on the phone screen
android.launcher_icon.type	Currently, only “user_content” is supported. This indicates, that the app creator has to upload the image using the Local Cosmos App Kit
android.launcher_icon.image_identifier	A unique identifier for this image.
android.splashscreen	The image shown during app start
android.splashscreen.type	Currently, only “user_content” is supported. This indicates, that the app creator has to upload the image using the Local Cosmos App Kit
android.splashscreen.image_identifier	A unique identifier for this image.

settings.ios

see settings.android

1.2 Building a frontend with vue.js

1.2.1 Create a vue.js project

First we will create a folder which contains both the vue.js app and the apache cordova app.

```
mkdir lcfrontend
cd lcfrontend
```

We will first create the vue.js app. Later, we will build it, and then move the build output to apache cordova.

```
npm init vue@latest
```

`npm init` will ask you for several options. In this tutorial, we named the frontend: `myfrontend-vue`, and this will be the vue.js project name. You can choose whatever options you wish, but make sure to **add the vue.js router**. The command will create a new folder with the name of your project, in this case `myfrontend-vue`.

Next, run the following commands.

```
cd myfrontend-vue
npm install
npm run dev
```

Take a look at the default vue3 app in your browser to make sure everything has worked so far.

1.2.2 Turn the vue.js app into a Cordova app

Create the Cordova project

First, install apache Cordova inside your `lcfrontend` folder, not inside your `myfrontend-vue` folder.

```
cd lcfrontend
npm install cordova
```

or install cordova globally using

```
npm install -g cordova
```

In this tutorial, we assume Cordova is installed locally in `lcfrontend`. If you choose to install it globally, you can use the `cordova` command directly and drop the `node_modules/cordova/bin/` prefix for all commands.

Now, we will create a blank Cordova app. We use the name of the frontend, `myfrontend`, for the cordova app:

```
cordova create myfrontend org.localcosmos.myfrontend MyFrontend
```

Cd into myfrontend, add the browser platform as well as the device plugin and run the cordova app.

```
cd myfrontend
../node_modules/cordova/bin/cordova platform add browser
../node_modules/cordova/bin/cordova plugin add cordova-plugin-device
../node_modules/cordova/bin/cordova build browser
../node_modules/cordova/bin/cordova run browser
```

Check your browser to make sure your blank Cordova app is working. You will see a green blinking **device is ready** area next to the Cordova logo. This means, your app successfully listened to Cordovas `deviceready` event. We will have to make our vue.js app listen to this event.

Copy cordova js over to your vue.js app

To turn your vue.js app into a Cordova app, copy the following to your vue.js app.

```
cd lcfrontend
cp myfrontend/platforms/browser/www/cordova.js myfrontend-vue/public
cp myfrontend/platforms/browser/www/cordova_plugins.js myfrontend-vue/public
```

(continues on next page)

(continued from previous page)

```
cp -r myfrontend/platforms/browser/www/plugins myfrontend-vue/public
cp -r myfrontend/platforms/browser/www/cordova-js-src js myfrontend-vue/public
```

You now have copied the files cordova creates during build over to your vue.js app. These files will not be part of the frontend you upload to the Local Cosmos App kit later, but are required for development.

An alternative approach is to configure Vite to build directly into cordovas www folder and then use Cordovas commands to preview your app.

Update index.html

Update the file `myfrontend-vue/index.html` by adding `cordova.js`.

myfrontend-vue/index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <link rel="icon" href="/favicon.ico" />

    <meta http-equiv="Content-Security-Policy" content="default-src 'self' data:␣
↪https://ssl.gstatic.com 'unsafe-eval'; style-src 'self' 'unsafe-inline'; media-src␣
↪*; img-src 'self' data: content:;">
    <meta name="format-detection" content="telephone=no">
    <meta name="msapplication-tap-highlight" content="no">
    <meta name="viewport" content="initial-scale=1, width=device-width, viewport-
↪fit=cover">

    <title>Vite App</title>
</head>
<body>
    <div id="app"></div>
    <script type="module" src="/src/main.js"></script>
    <script src="cordova.js"></script>
</body>
</html>
```

Listen to deviceready

We want our vue.js app to load after cordovas `deviceready` event has been emitted. This is a requirement for cordova apps to work properly. Replace the content of `myfrontend-vue/src/main.js` with the following content.

myfrontend-vue/src/main.js:

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'

import './assets/main.css'

function onDeviceReady(event){

    console.log('Running cordova-' + cordova.platformId + '@' + cordova.version);
```

(continues on next page)

(continued from previous page)

```
    const app = createApp(App)

    app.use(router)

    app.mount('#app')
}

document.addEventListener('deviceready', onDeviceReady, false);
```

To test if everythin worked, start your vite development server with

```
npm run dev
```

The vue.js default app should load, and you should find the following in the console output:

```
Running cordova-browser@6.0.0
```

1.2.3 Comply with the Local Cosmos frontend structure

Up to now, we simply used the default vue.js app. We will have to alter the project structure to make it work with Local Cosmos. Remember that Local Cosmos is not restricted to vue.js. You can write frontends for Local Cosmos with any framework you want, or without using a framework at all.

First, we will create all necessary folders:

```
cd lcfrontend/myfrontend-vue
cd public
mkdir localcosmos
mkdir online_content
mkdir fact_sheets
mkdir webapp
mkdir android
mkdir ios
```

Next, create the file `settings.json` in `myfrontend-vue/public` and put the following into it.

```
{
    "frontend" : "MyFrontend",
    "version" : "0.1",
    "PRIMARY_LANGUAGE" : "en"
}
```

Replace `MyFrontend` with the name of your frontend.

1.2.4 Copy development data

Without data or content, you obviously cannot create a frontend. Download the localcosmos development data and put it into `public/localcosmos`.

1.2.5 Localization

Local Cosmos frontends require localization, even if your frontend supports only one language. The reason is the **Glossary** feature. Every single piece of text can contain a reference to a term in the glossary.

Let's take a look at locale entry with a reference to a glossary entry:

```
{
    "brush-like rigid bushes, single filaments": "brush-like <span class=\"glossary_
↪link tap\" action=\"glossary\" data-term=\"cmlnaWQ=\">rigid </span> bushes, single␣
↪filaments",
}
```

In this example, the word `rigid` occurs in the glossary of the app and thus is wrapped in a `<span>` with the class `glossary-link`.

As a frontend developer, you might want to display plain text without html that refers to the glossary. For example the text on a button should not contain a glossary link. Therefore, two files are provided for each localization:

```
plain.json
glossarized.json
```

`plain.json` contains only plain text **without** any references to the glossary. `glossarized.json` contains the same localization **with** links to glossary entries.

As a result of this structure, a localization library which supports namespacing is required. In this tutorial, i18next will be used. For implementation, *i18next-vue <https://i18next.github.io/i18next-vue/>* will be used.

Localization with i18next

First, download the locale development data and put it into `public/locales`.

Next, install i18next

```
cd myfrontend-vue
npm install i18next-vue
npm install i18next-http-backend
```

As localizations are present as files, the http backend is required to load those files using an xhr request. This makes the loading of the localization asynchronous. As a result, we have to adjust *main.js* as follows:

myfrontend-vue/src/main.js:

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'

import i18next from 'i18next';
import I18NextVue from 'i18next-vue';
import HttpApi from 'i18next-http-backend';


function onDeviceReady(event) {

    console.log('Running cordova-' + cordova.platformId + '@' + cordova.version);

    const app = createApp(App)

```

(continues on next page)

(continued from previous page)

```
    fetch('settings.json').then(r => r.json()).then(settingsData => {

        const lcSettings = settingsData;

        app.provide('lcSettings', lcSettings);

        app.use(router)

        i18next.use(HttpApi).init({
            language: navigator.language,
            fallbackLng: lcSettings.PRIMARY_LANGUAGE,
            debug: true,
            ns: ['plain', 'glossarized'],
            defaultNS: 'glossarized',
            fallbackNS: 'plain', // if app has no glossary, glossarized is not present
            backend: {
                // load from i18next-gitbook repo
                loadPath: '/locales/{{lng}}/{{ns}}.json',
                crossDomain: true
            }
        }).then(() => {

            app.use(I18NextVue, { i18next });

            app.mount('#app')

        });

    });

}

document.addEventListener('deviceready', onDeviceReady, false);
```

First, we fetch and parse the `settings.json` file because it contains required information about languages. We are then making sure that i18next has finished loading before the app is created.

We also made the settings available in the global scope.

Fetching glossarized and plain locale entries in your app

By default, the glossarized localization will be fetched.

```
<template>

    Glossarized (default): <span v-html="$t('WORD')"></span><br>
    Plain text: {{ $t('plain:WORD') }}

</template>
```

As the glossarized localization might contain a `<span>` element, you always have to fetch it using the `v-html` directive of vue.js.

1.2.6 Configure vite to build for Cordova

First, we move cordovas default www folder to not lose it when vite builds.

```
cd lcfrontend/myfrontend
mv www www_default
```

Next, we configure Vite to build directly into cordovas www folder.

Open `vite.config.js` and add the following to your config:

```
build : {
    outDir : '../myfrontend/www'
}
```

Run your vue.js app on an Android device

First, run the build command

```
npm run build
```

This will build your vue.js app directly into the cordova www folder, as specified in `vite.config.js`. If you take a look at the www folder inside of `myfrontend`, you will notice that the cordova specific files and folders we copied over to vue.js earlier are present. These files will be added by cordova automatically, so we have to remove them.

```
cd lcfrontend/myfrontend/www
rm cordova.js
rm cordova_plugins.js
rm -r plugins
rm -r cordova-js-src
```

Connect your android development device to your PC and check if adb has recognized it. The android Sdk has to be installed.

```
adb devices
```

Your device should be listed.

Finally, add and run the android platform

```
cd lcfrontend/myfrontend
../node_modules/cordova/bin/cordova platform add android
../node_modules/cordova/bin/cordova platform run android
```

If everything worked, you will see your Local Cosmos App built with vue.js on your android device.

1.3 How to write Online Content and Fact Sheet templates

1.4 APIs

CHAPTER 2

Import from Excel

2.1 Basics

To create importable data structures, you need the following skills:

- creating folders
- basic Excel, working with multiple spreadsheets, cell contents and cell backgrounds
- basic understanding of tree structures
- creating .xls or .xlsx files
- creating .zip files
- creating .jpg or .png images with specific dimensions

2.2 Glossary

If a glossary term occurs in the app texts, it is automatically recognized and a link to its definition is added.

2.2.1 1. The Excel File

You have to create one Excel file. The Excel file has to be named accoring to the name of your glossary.

Name of glossary on localcosmos.org	Name of Excel file
Glossary	`Glossary.xlsx`

2.2.2 2. Excel Content

Before you continue to read, it is recommended to download the example Excel file. `download example Excel file`.

Important: all content you put into your Excel files has to be in the primary language of your Local Cosmos App

The Columns

The glossary Excel file has 3 columns: `Term`, `Synonyms` and `Definition`.

	A	B	C
1	Term	Synonyms	Definition
2	bark		Bark is the outermost layers of stems and roots of woody plants.
3	Ash dieback	Hymenoscyphus fraxineus	Hymenoscyphus fraxineus is an Ascomycete fungus that causes ash dieback, a chronic fungal disease of ash trees in Europe.
4	Wood-decay fungus	Soft rot | White rot	A wood-decay fungus is any species of fungus that digests moist wood, causing it to rot.

Column A: Term

The term is the text that will be automatically recognized if it occurs in a text of your app. A link to its definition is set, so the user can look it up. This term detection happens during `build`, and therefore can only be seen after you built your app.

Column B: Synonyms

One or more synonyms of the term. Separated by the character "|". Synonyms are also automatically detected in texts, just like terms.

Column C: Definition

The definition of the term.

2.2.3 3. Uploading data

An uploadable `Glossary.xlsx` has to be compressed as a .zip file. The content of this .zip file is only the `Glossary.xlsx` file.

2.3 Nature Guides

Nature Guides are identification keys or species lists.

2.3.1 1. The Excel Files

You have to create two Excel files:

1. excel file containing the Nature Guide, named exaclty like you named your Nature Guide on localcosmos.org
2. excel file containing image licences, named *Image Licences.xlsx*

Naming the Excel file

The import will only succeed if you name your Excel files correctly. The name of the Excel file containing the Nature Guide has to match the name of the Nature Guide you created on localcosmos.org.

Name of Nature Guide on localcosmos.org	Name of Excel file
Identify trees	`Identify trees.xls` or `Identify trees.xlsx`

The Excel file containing the image licences has to be named `Image_Licences.xlsx`.

2.3.2 2. Nature Guide Excel

Before you continue to read, it is recommended to download the example Excel file. `download example Excel file`.

Important: all content you put into your Excel files has to be in the primary language of your Local Cosmos App

The Tree sheet

The Nature Guide Excel requires at exactly one sheet named `Tree`. This sheet defines the identification tree. A tree with only one level (= no Parent Nodes) would result in a simple species list. Each line of this sheet equals one entry in the identification tree - except the first one which defines the columns.

Example: Sheet "Tree" of "Identify Trees.xlsx":

	A	B	C	D	E
1	Node Name	Parent Node	Taxonomic Source	Scientific Name	Decision Rule
2	Deciduous Trees				with leaves, bare in winter
3	Conifers				with needles
4	Spruce	Conifers	taxonomy.sources.col	Picea abies	
5	Larch	Conifers	taxonomy.sources.col	Larix decidua	
6	Sycamore maple	Deciduous Trees	taxonomy.sources.col	Acer pseudoplatanus	
7	Silver birch	Deciduous Trees	taxonomy.sources.col	Betula pendula	
8	Oak	Deciduous Trees	taxonomy.sources.col	Quercus robur	

Columns of the Tree sheet

Column A: Node Name

A Node is a leveled entry in the identification tree and can have any name. It can be something like `Conifers`, `Whales` or even `Stones`. It does not have to be a biological taxon.

Column B: Parent Node

The parent node of this node. If empty, this node will be displayed on the first identification step. If a Parent Node is specified, this node will appear after the user selected the specified Parent Node.

Column C: Taxonomic Source

The columns `Taxonomic Source` and `Scientific Name` only should be filled if the node is an identification result.

Currently, only two taxonomic sources are available, **Catalogue Of Life** and **Custom taxonomy**. Therefore, this cell has to be filled with `taxonomy.sources.col` (recommended) or `taxonomy.sources.custom`, if you supply a custom taxonomy. Leave empty if your node is not a biological taxon or if you do not want to use taxonomic features.

Column D: Scientific Name

The columns `Taxonomic Source` and `Scientific Name` only should be filled if the node is an **identification result**. Fill in a scientific name **without author** into this cell. Example: `Larix decidua`.

Column E: Decision Rule

A `Decision Rule` is a rule when to decide to choose this taxon or information how to identify it. Decision Rules are optional and make sense if you do not want to use a trait based itentification for this level of the identification tree.

Matrix Sheets

A matrix sheet is used to define an identification matrix for a specific level of the identifcation tree. The level the matrix is used for is specified by a parent node. The name of the matrix sheet inside your Excel document has to be set accordingly.

Name of Parent Node	Name of Matrix Sheet
Deciduous Trees	Matrix_Deciduous Trees

In the example Excel file, a matrix sheet for all deciduous trees is used, and thus is named after the Parent Node `Deciduous Trees`.

Nodes are entered in Column A, Matrix Filters are entered from Column B onwards.

Example: Sheet "Matrix_Deciduous Trees" of "Identify Trees.xlsx":

	A	B	C	D	E	F
1	*name of filter* ->	Leaf structure	Shape of the leaf	Color of the bark	Taxon-omy	Length of the leaf
2	*trait type* ->	DescriptiveTex-tAndImages	DescriptiveTex-tAndImages	Color	Taxon	Range
3	*unit* ->					cm
4	*step* ->					1
5	Sycamore Maple	side by side	lobed	grey		10-18
6	Silver birch	alternating	heart shaped	white		3.5-7
7	Oak	alternating	lobed	brown | grey		8-15

Within the matrix sheet, the first 4 rows are used to define the matrix filters (=traits).

- row 1: Name of the matrix filter (trait)
- row 2: Type of the matrix filter. Available matrix filter types are: `DescriptiveTextAndImages`, `Color`, `Range`, `Number`, `Taxon`
- row 3 (optional): unit, for example `cm`
- row 4: step of the Range. Only applies if row 2 (type) is `Range`. Defines the step of the rendered slider.

Row 5 onwards are used to assign values to nodes. If you want to assign more than one value to a node, use the OR seprator `|`. For example `brown | grey`, which stands for `brown OR grey`.

You can create one Matrix Sheet for each Parent Node, but no Matrix Sheet is required.

Matrix Filter Types

DescriptiveTextAndImages

A text with an image. Suitable for traits like "Shape of the leaf".

Color

Colors consist of a name and a color code. Both are defined in the `Colors Sheet`. In the Matrix Sheet you only reference colors by name, as defined in the `Colors Sheet`.

Range

A range of numbers, for example from 10cm to 50cm. You can define the step of the range in row 4. If the step is `1`, the range slider, which the app user uses to select a value, would consist of the numbers 10, 11, 12, ... 48, 49, 50.

Number

Numbers that are no ranges, for example the numbers 2,4,5,8.

Taxon

Taxonomic filters are defined in the `Taxonomic Filters Sheet`. You can only add a taxonomic filter, but you cannot assign values in the Matrix Sheet as you can with the other matrix filters. Taxonomic Filters work automatically using the taxonomic backend of your App.

Colors Sheet

The Colors Sheet is used to define colors. Column A sets the name of the color. Column B sets the actual color by using a cell background.

Taxonomic Filters Sheet

This sheet has to be named `Taxonomic Filters`, and your Excel file may only have one `Taxonomic Filters` sheet.

Example: Sheet "Taxonomic Filters" of "Identify Trees.xlsx":

	A	B	C	D
1	Scientific names	Taxonomic sources	Matrices	Matrix Filter Name
2	Fagaceae	taxonomy.sources.col	Matrix_Deciduous Trees	Taxonomy
3	Sapindales	taxonomy.sources.col	Matrix_Deciduous Trees	Taxonomy
4	Oleaceae	taxonomy.sources.col	Matrix_Deciduous Trees	Taxonomy

Column A (Scientific Names): Scientific name of the taxon which will act as a filter.

Column B (Taxonomic sources): See "Columns of the tree sheet".

Column C (Matrices): The Matrix Sheet this taxonomic filter is used by.

Column D (Matrix Filter Name): The Name of the Matrix filter, has to match the name of the matrix filter in the references Tree Sheet.

2.3.3 3. Images

You upload your Nature Guide as a `.zip` file. Within this `.zip` file, you can supply images for the following assets:

- Nodes
- Matrix Filters of the type `DescriptiveTextAndImages`

All images have to reside in a folder called `images`. All images for Nodes have to reside in `images/Tree`. All images for matrix filters have to reside in the folder `images/Matrix_<parent_node>/<matrix_filter_name>/`, and the name of the image has to match the value.

Example: `images/Matrix_Deciduous Trees/Shape of the leaf/heart shaped.jpg`

For the example Excel file, you would have a folder structure similar to this:

```
nature_guide
├── Identify Trees.xlsx
├── images
│   ├── Tree
│   │   ├── Conifers.jpg
│   │   ├── Deciduous Trees.jpg
│   │   ├── Oak.jpg
│   │
│   ├── Matrix_Deciduous Trees
│   │   ├── Shape of the leaf
│   │   │   ├── heart shaped.jpg
```

Each Tree Image has to be exactly 600px x 600px in dimensions. Each Matrix Filter Image has to be exactly 400px x 400px in dimensions.

2.3.4 4. Image Licences Excel

You have to supply an image licence alongside its creator for all your images. The image licences are provided by the file `Image Licences.xlsx`. `download example Excel file`.

You have to supply at least the columns `Image` (column A), `Licence` (column B) and `Creator` (column C). `Creator link` (column D) is optional.

The `Image` column expects paths to the image, relative to your `image` folder, where the images reside.

Examples: `Tree/Conifers.jpg` or `Matrix_Deciduous Trees/Shape of the leaf/heart_shaped.jpg`.

Only short licence names are allowed for the `Licence` Column. Available Licences are:

Full Licence Name	Short name
Public Domain Dedication	CC0
Creative Commons Attribution	CC BY
Creative Commons Attribution-ShareAlike	CC BY-SA
Creative Commons Attribution-NoDerivs	CC BY-ND
Creative Commons Attribution-NonCommercial	CC BY-NC
Creative Commons Attribution-NonCommercial-ShareAlike	CC BY-NC-SA
Creative Commons Attribution-NonCommercial-NoDerivs	CC BY-NC-ND
Public Domain Mark	PDM

Example: Sheet "Image Licences" of "Identify Trees.xslx":

	A	B	C	D
1	Image	Licence	Creator	Creator link
2	Tree/Deciduous Trees.jpg	CC0	Pablo Picasso	
3	Tree/Conifers.jpg	CC BY	Vincent van Gogh	https://localcosmos.org
4	Tree/Silver birch.jpg	CC BY-SA	Claude Monet	
4	Tree/Matrix_Deciduous Trees/Leaf structure/lobed.jpg	CC0	Claude Monet	

2.3.5 5. Uploading data

All uploadable Nature Guides consist of the folder `images`, the file `<name_of_nature_guide>.xlsx`, and the file `Image Licences.xlsx`. You have to create a `.zip` file containing these 3 items. After you have created your zip file, you can upload it in the localcosmos.org app kit.

`download example zip file.`

2.4 Taxon Profiles

CHAPTER 3

Indices and tables

- genindex
- modindex
- search