SPX-GC | SmartPX Graphics Controller

Manage and control graphics for CasparCG and streaming applications.

> Readme updated Jan 08 2021. See RELEASE_NOTES.md for latest changes (v.1.0.9).

SPX-GC is professional graphics controller for live television productions and web streaming. Browser based GUI can control HTML graphics templates on CasparCG server(s) and/or live stream applications such as OBS, vMix or Wirecast.

*SPX-GC is pronounced **G.C.** [dʒiːsiː] or just "geesee"* 😉

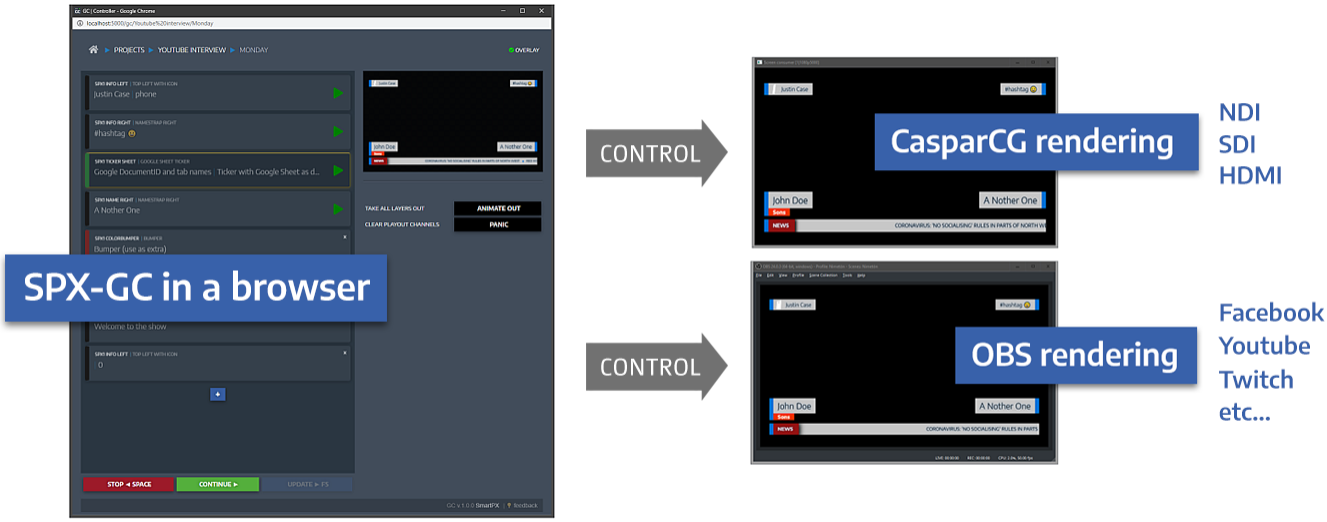


Table of Contents

SPX-GC can be used to playout lower thirds, bumpers, logos and other on-screen graphics in live web streams or live TV broadcasts. Content for the graphic templates are entered into *elements* which are stored on *rundowns* within *projects*.

Software is based on a NodeJS server and can be run on Windows, Mac or Linux computers, on-premise or using cloud instances for remote work scenarios.

Graphic templates are typical HTML templates used with CasparCG and other HTML compatible renderers. Integrating existing templates with SPX-GC is done by adding *a template definition* (javascript-snippet) to them.

Originally SPX-GC was developed by SmartPX for YLE, a public broadcaster in Finland. Thanks **Markus Nygård** for the challenge! 🤘

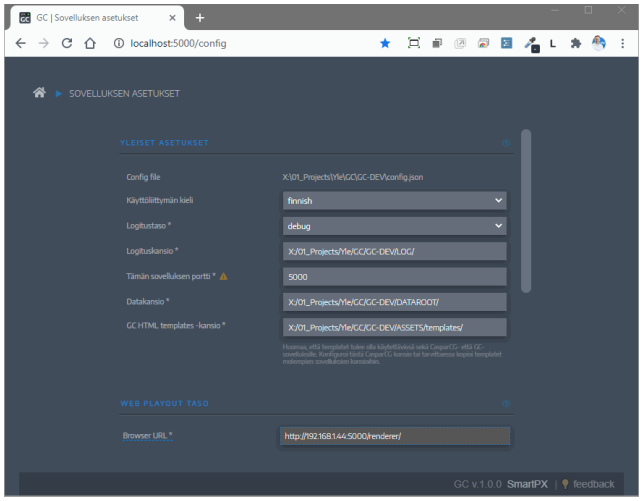
> If you need custom HTML templates or functionality get in touch tuomo@smartpx.fi.

Live demo

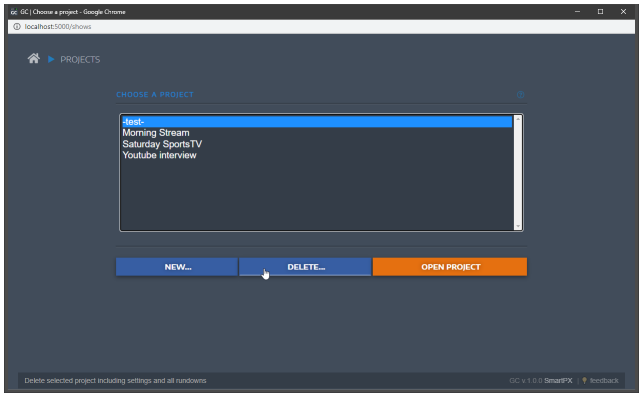
SPX-GC running in the cloud: **http://35.228.47.121:5000**

Please be aware there is just *one instance* running for demo purposes, so expect clashes and overall quirky experience if multiple users are logged in at once. Changes made in demo are reset automatically few times a day. (Also pay attention to the version number, it may not be the latest version.)

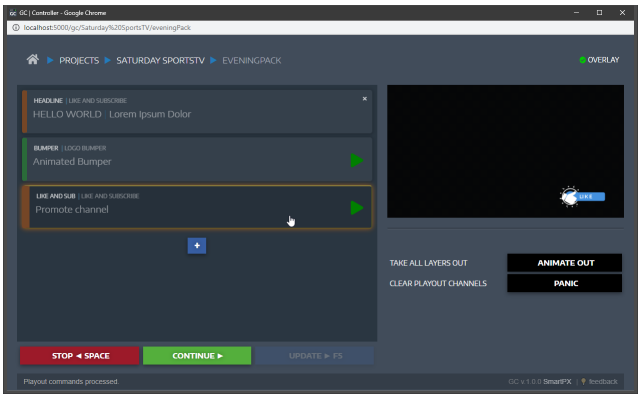
Screenshots



SPX-GC UI is browser based and can be operated with a mouse or keyboard. Additonal *extra controls* can be added as *plugins* to execute specific tasks or to trigger events in external devices.



Content is managed in *projects*. Each project can have unlimited amount of *rundowns* and *graphics templates*. Projects and their rundowns and settings are stored in *dataroot -folder*.



Main Controller: rundown with few items and a local preview. Items can be edited and controlled also with keyboard shortcuts. Fullscreen viewing mode recommended. Buttons below preview are customizeable.

An introduction video on Youtube. There are more images in the screenshots -folder.

Installation

SPX-GC can be installed using a **ready-to-go binary package** which includes all required software components. Developers can alternatively get the full source code and run SPX-GC with `npm scripts`, see section install source code.

> Source is updated more frequently than binary packages. See package.json file for current version.

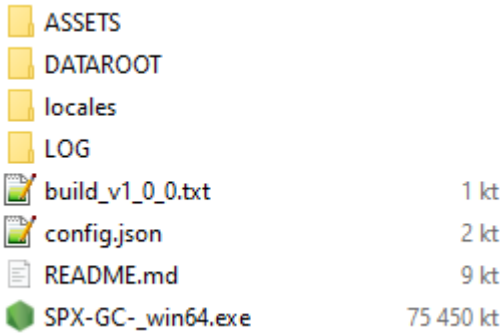
Available pre-built packages:

Operating system	Zip file	Build date	Notes
Windows	SPX-GC_1_0_9_win64.zip	Dec 30 2020	The app is cross platform and is mostly developed and tested on Windows. Approx 56% users are on Windows.
MacOS	SPX-GC_1_0_9_macos64.zip	Dec 30 2020	6% of current users are on Mac.
Linux	SPX-GC_1_0_9_linux64.zip	Dec 30 2020	Tested with some flavours of Debian and Ubuntu but user's input is appreciated here, see feedback. 38% of users are on Linux

> Please get in touch if you have problems downloading or installing these files.

Option 1: **Install a pre-built package**

- Download a zip-file for your system using one of the links above.
- Create a new folder for the app (for example on Windows `C:/Program Files/SPXGC/`, or on Linux `/SPXGC`).
- Extract the zip-file to that folder.
- Locate the executable (for example `SPX-GC_win64.exe` on Windows) and double click it to start the SPX-GC server. A console window should open (and remain open) and show some startup information. When running application the first time it will create a file structure shown in the below screenshot. Note: unzipping and running SPX-GC does *not* usually require admin priviledges.
- On Linux or Mac add execute permission to the file (`chmod a+x SPX-GC_linux64 [or ...macos64]`) and launch it in a console (`./SPX-GC_linux64 [or ...macos64]`).
- See next steps in section first launch.



Option 2: **Install from source code**

Developers can get the source code from the repository with git and run the application using NodeJS and npm.

- Create an empty folder on your system and fetch the source code using a `git clone` command:

```
git clone https://github.com/TuomoKu/SPX-GC.git
```

- After downloading the source, install required additional dependencies (node_modules) with

```
npm install
```

- See `package.json` for available scripts, but in **development** the typical start script would be `npm run dev` which will use *nodemon* to restart the server when changes are made to source files.

```
npm run dev
```

pm2 process manager

- Installation of `pm2` process manager (https://pm2.keymetrics.io/) can help in advanced production scenarios.

- To run the server in **production mode** use `npm start` which will run the server in the background with `pm2` process manager which will automatically restart the server if a crash occurs. Deeper usage and configuration options of *pm2* is outside the scope of this readme-file.

```
npm start
```

Run multiple instances

- To run several instances of SPX-GC (on different ports) with `pm2` prepare a `ecosystem.config.js` -file to same folder as config.json with details of each instance, such as:

```
// Example "ecosystem.config.js" file for pm2 to run multiple instances of SPX-GC.

module.exports = {
  apps : [
    {
    'name': 'GC1',
    'script': 'server.js',
    'args': 'config.json'
    },
    {
      'name': 'GC2',
      'script': 'server.js',
      'args': 'config5001.json'
      }
  ]
};
```

Then launch multiple instances with pm2:

```
pm2 start ecosystem.config.js
```

Stop all running instances

```
pm2 kill
```

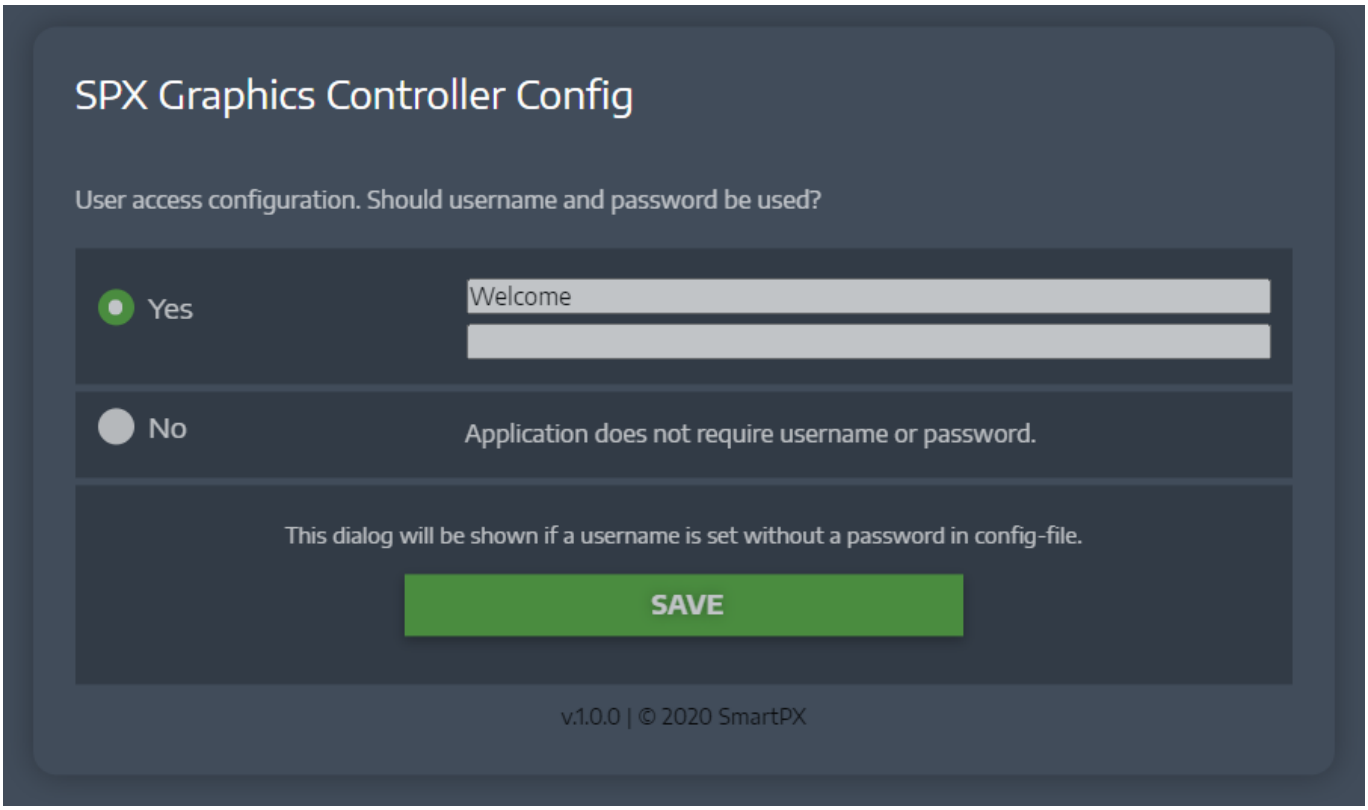
First launch

- Open web browser (such as Chrome) and load SPX-GC gui from url shown in the console at the start-up:

```
--------------------------------
  SPX-GC url:
  http://192.148.1.22:5000
--------------------------------
```

> Port 5000 is the default value in config and can be changed.

If installation and server start-up worked, you should see a Config screen in your browser asking a preference regarding user access.



There are two alternatives:

- `YES`: Username and password are reguired to access the application.
- `NO`: Application will not require a login.
- This config screen is shown
    - at first startup, or
    - when `config.json` is missing, or
    - when `config.json` **has** username but password is left **empty**

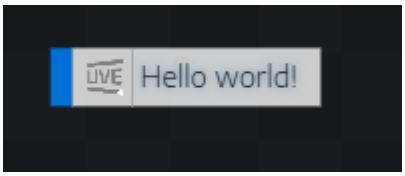
Depending on the selection made, you will either be asked to login or you land to the Welcome page and you are free to explore the application. If password is given it will be stored in the config-file in unreadable, encrypted format.

> By default the dataroot has one "Hello world" -project with "My First Rundown" in it for demonstration purposes.

Start making configuration changes or creating projects and adding templates and adding those to rundowns for playout.

You can also follow these steps to get yourself familiarized with the application:

1. Open SPX-GC in browser, typically at http://localhost:5000
2. Choose 'no login' policy by selecting **No** option and click **Save**
3. Go to **Projects**
4. Add a new project, for instance `My First Project`. (Project's settings opens.)
5. Click **[+]** button to add the first template to the project
6. Browse to `smartpx > Template_Pack_1` -folder and choose `SPX1_INFO_LEFT.html` -template
7. Go back to **Projects**
8. Double click `My First Project` to open it
9. Add a new rundown to this project, for instance `My First Rundown`. (The new empty rundown opens.)
10. Click **[+]** button to add an item to the rundown
11. Pick **SPX1_INFO_LEFT** -template
12. Double click rundown item to edit it, enter "Hello world!" and click **Save** to close the editor
13. Play the item with **SPACEBAR** or by clicking on **PLAY** button at the bottom of rundown list.



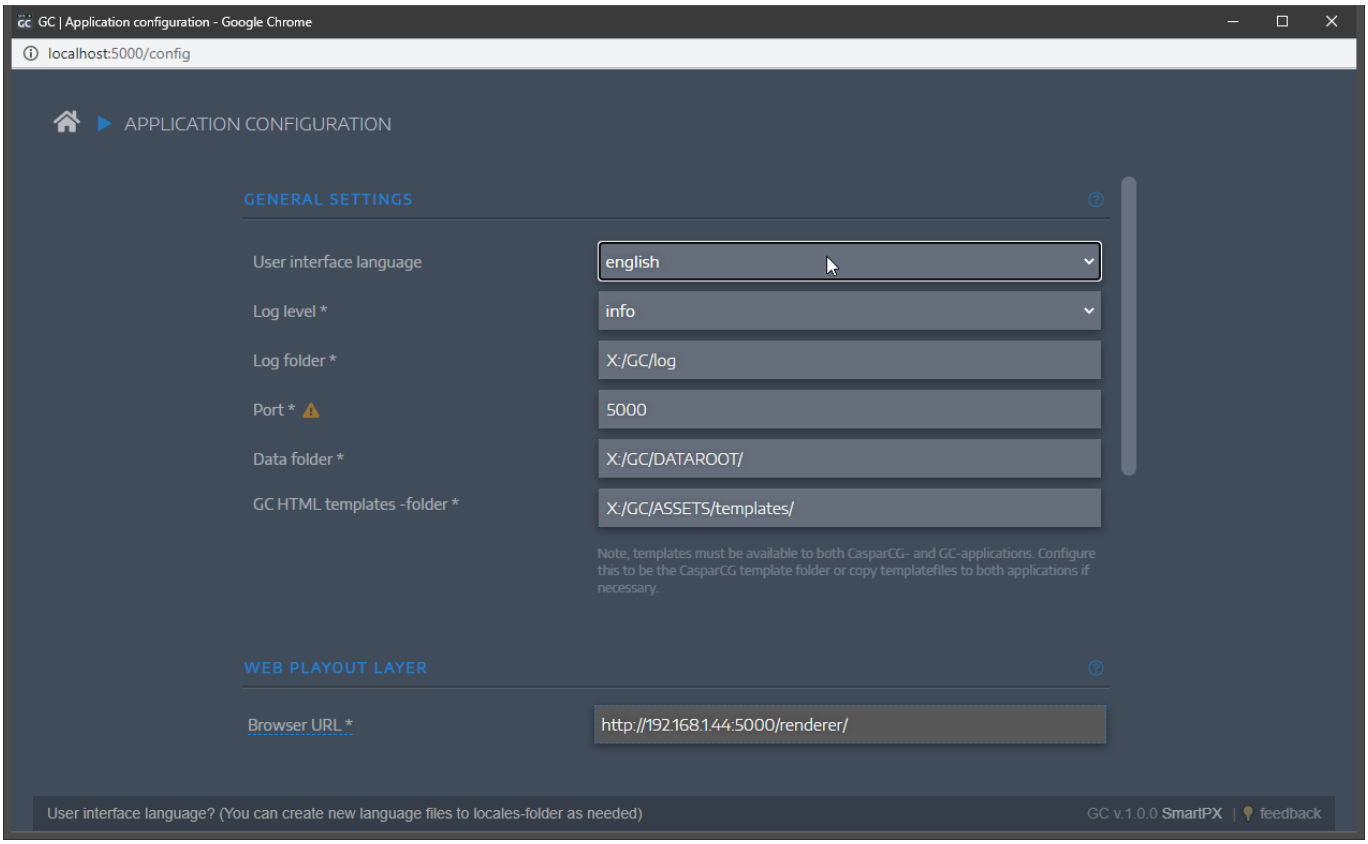
Congratulations! Now go back to your project's settings and add more templates to it...

> When a new version becomes available it will be shown on the Welcome page of the application.

App configuration options

> Application **DOES NOT** come with `config.json` and it will be generated at server start up.

SPX-GC uses a `JSON file` to store configuration settings, such as folder paths, playout server settings or user interface language options. Most of the settings can be changed from the configuration page.



Some rarely used settings are left out from configuration page and can be changed by manually modifying the *config file* in a text editor.

The default configuration file name is `config.json` but it is possible to run the server with a specific configuration file. For instance you might have two instances running on the same system, using shared project files and templates but on different server ports and using different renderers. (See also pm2 process manager)

To run the server with another config, provide the config file as the first command line argument, for example:

```
SPX-GC_win64.exe myOtherConfig.json
```

An example `config.json` of the SPX-GC server

```
{
  "general": {
    "username": "welcome",
    "password": "",
    "hostname": "My main machine",
    "langfile": "english.json",
    "loglevel": "info",
    "logfolder": "X:/GC-DEV/LOG/",
    "port": "5000",
    "dataroot": "X:/DATAROOT/",
    "templatefolder": "X:/GC-DEV/ASSETS/templates/",
    "templatesource": "spxgc-ip-address"
  },
```

```
    },
    "casparcg": {
      "servers": [
        {
          "name": "OVERLAY",
          "host": "localhost",
          "port": "5250"
        },
        {
          "name": "VIDEOWALL",
          "host": "128.120.110.1",
          "port": "5250"
        }
      ]
    },
    "globalExtras": {
      "customscript": "/ExtraFunctions/demoFunctions.js",
      "CustomControls": [
        {
          "ftype": "button",
          "bgclass": "bg_black",
          "text": "ANIMATE OUT",
          "fcall": "stopAll()",
          "description": "Take all layers out"
        },
        {
          "ftype": "button",
          "bgclass": "bg_red",
          "text": "PANIC",
          "fcall": "clearAllChannels()",
          "description": "Clear playout channels"
        }
      ]
    }
  }
```

> **Please note**: the server will fail to start if config is not valid JSON. You can use JSONLint to validate JSON data.

Config parameters

`general.username / password` If *username* is present but the *password* is left blank, the app will ask for login policy, just as with first launch. When both are entered the *password* is saved here (encrypted) and a logic is required to start a session.

`general.templatefolder` contains the HTML templates and their resource files (css, js, images, etc). This root folder is used by SPX-GC's template browser and 'Explore templates folder' menu command (Win only). For playout folder see *templatesource* parameter below.

`general.templatesource` (Added in v 1.0.9) For CasparCG playout the templates can be loaded from the *filesystem* or via *http-connection* provided by SPX-GC. Supported values are:

- `spxgc-ip-address` to automatically use SPX-GC's IP address and http -protocol for playing out templates from SPX-GC's template folder. This is the default behaviour.
- `casparcg-template-path` to playout templates from target CasparCG server's file system template-path. (See *caspar.config* file) Note, in this workflow the templates *must be in two places*: in SPX-GC ASSETS/templates -folder *and* CasparCG's templates folder. And if a changes are done to either location, those changes should also be done to the other. `rsync` or other mirroring technique should be considered...
- `http://<ip-address>` manually entered address can be used when the automatically generated IP address is not usable. For instance Docker containers or VM hosted instances may expose internal IP address which can not be accessed from outside.

> Please note *templatesource* only affects CasparCG playout and not web playout. Also file:// protocol is more restrictive in using external data sources and it can yield javascript errors, such as CORS.

`general.langfile` is a file reference in `locales`-folder for a JSON file containing UI strings in that language. Folder is scanned at server start and files are shown in the configuration as language options.

If you want to add your own language you have to options: You can copy an existing file to another name and modify it's contents or better yet: make a copy of a Google Sheet language document of locale strings and use that to create the locale file. You can also **contribute** to the project by submitting your language back to the project. See the Google Sheet for instructions.

`general.loglevel` default value is `info`. Other possible values are `error` (least), `warn`, `verbose` and `debug` (most log data). All log messages are stored into log files in logfolder. The active file is named `access.log`. Log files can be useful in troubleshooting.

`globalExtras{}` are additional user interface controls, or *plugins*, shown below preview window in all project as opposed to projectExtras which are project specific. Each item has an UI component (a button) and associated function call available in the specified `javascript file`. When a new `config.json` is created it has some demo extra controls to introduce related consepts and possibilities.

Projects and rundowns

All content in SPX-GC is stored as files in `dataroot` folder which is specified in the configuration.

- **Projects** are *subfolders* in the dataroot-folder
- **Rundowns** are *files* in project subfolders.

Projects can be added and removed on the *Projects* page and rundowns can be added and removed inside project on the *Rundows* page. Most changes are saved automatically. If the UI becomes unresponsive it is usually fixed by refreshing the current page (Ctrl+R).

```
File structure of dataroot:

▸ LOG
▸ ASSETS
▾ DATAROOT
```

```
├─▸ Project A
├─▸ Project B
└─▾ Project C
    ├──  profile.json
    └─▾ data
        ├─ Rundown 1.json
        └─ Rundown 2.json
```

> Typically users don't need to do any manual file management using computer's filesystem.

Project specific settings, such as assigned templates and project extras are stored into `profile.json` within each project folder.

Templates can be added to a project on the project settings page. When a template (a .html file) is browsed and selected, the system will scan the file and search for a template definition which will tell SPX-GC what kind of input fields should be generated for that template and how the template is planned to be played out. Template defaults are stored to project's `profile.json` (as "copy") and if HTML template's definition related details are changed afterwards the template must be imported to the project again. The system does not re-scan added templates.

> If selected template does NOT have template definition it will cause an `error:templateDefinitionMissing` -message. See section html templates.

showExtras are additional user interface controls, or *plugins*, shown below preview window in current project as opposed to globalExtras which are shown in every project. Each item has an UI component (a button) and associated function call available in the specified `javascript file`.

An example projects settings `<PROJECT>/profile.json`:

```
{
  "templates": [
    {
      "description": "Hashtag one-liner",
      "playserver": "OVERLAY",
      "playchannel": "1",
      "playlayer": "7",
      "webplayout": "7",
      "out": "4000",
      "uicolor": "7",
      "onair": "false",
      "dataformat": "xml",
      "relpath": "myTemplates/ProjectA/hashtag.html",
      "DataFields": [
        {
          "field": "f0",
          "ftype": "textfield",
          "title": "Social media hashtag",
          "value": "#welldone"
        }
      ],
    }
```

```
    ],
    "showExtras": {
      "customscript": "/ExtraFunctions/demoFunctions.js",
      "CustomControls": [
          {
            "description": "Play simple bumper",
            "ftype": "button",
            "bgclass": "bg_orange",
            "text": "Bumper FX",
            "fcall": "PlayBumper",
          },
          {
            "description": "Corner logo on/off",
            "ftype": "togglebutton",
            "bgclass": "bg_green",
            "text0": "Logo ON",
            "text1": "Logo OFF",
            "fcall": "logoToggle(this)"
          },
          {
            "description": "Sound FX",
            "ftype": "selectbutton",
            "bgclass": "bg_blue",
            "text": "Play",
            "fcall": "playSelectedAudio",
            "value": "yes.wav",
            "items": [
                {
                    "text": "No!",
                    "value": "no.wav"
                },
                {
                    "text": "Yesss!",
                    "value": "yes.wav"
                }
            ]
          },
        ]
    }
}
```

> The above project has just one template (`hashtag.html`) assigned with three extra controls of different types.

Custom control's ftype can be

- **button**: a simple push button (with `text` as caption)
- **togglebutton**: button with separate on / off states
- **selectbutton**: a select list with an execute selection button
- **ftypes**
  - `hidden` value is used, title shown
  - `textfield` a typical input field

- `dropdown` options provided as an array
  - `"items":[ {"text": "Hundred", "value": 100}, {"text": "Dozen", "value": 12} ]`
  - `value` is one of the item array values
- `caption` text of "value" is shown in UI. Useful with static graphics.

Templates

SPX-GC uses HTML templates for visuals.

Templates can have any features supported by the renderers, such as Canvas objects, WebGL animations, CSS transforms and animations, animation libraries, such as GSAP, ThreeJS, Anime, Lottie/Bodymovin and templates can utilize ajax calls for data visualizations and other advanced uses.

SPX-GC comes with a starter template package for reference. See folder ASSETS/templates/smartpx/Template_Pack_1

Video: Use existing HTML templates.

```
Recommended folder structure for templates

▸ LOG
▸ DATAROOT
▾ ASSETS
   ├─▸ video
   ├─▸ media
   └─▾ templates
       ├─▸ smartpx
       ├─▸ yle
       └─▾ myCompany
           ├─▸ ProjectA
           └─▾ ProjectB
               ├─▸ css
               ├─▸ js
               ├─ Template1.html
               └─ Template2.html
```

> The templates must be within `ASSETS/templates` folder structure. It is preferred to have a single subfolder for all *your* templates (myCompany in the example above) and futher subfolders for different *template packs* or *visual styles* within it (ProjectA, ProjectB in the example).

SPX-GC user interface and web playout always loads templates from `ASSETS/templates` folder, but CasparCG playout can be configured to playout *copied* templates from template-path folder configured in CasparCG Server caspar.config -file.

SPXGCTemplateDefinition -object in templates

> **IMPORTANT:** Each HTML template must have an `JSON data object` present in the HTML-files source code, within the HEAD section. Video: use existing HTML templates covers also this topic.

TemplateDefinition configures how a template is supposed to work within SPX-GC; what kinds of controls are shown to the operator and how the graphic should playout, on which server and layer for instance. These values are template's `defaults` and can be changed in the Project Settings view after the template is added to the project.

> See details about supported values below the snippet.

```
<!-- An example template definition object for SPX-GC. -->
<!-- Place it as the last item within the HEAD section -->

<script>
    window.SPXGCTemplateDefinition = {
        "description": "Top left with icon",
        "playserver": "OVERLAY",
        "playchannel": "1",
        "playlayer": "7",
        "webplayout": "7",
        "steps" : "1",
        "out": "manual",
        "uicolor": "2",
        "dataformat": "json",
        "DataFields": [
            {
                "ftype" : "instruction",
                "value" : "A example demo template definition. Learn what it does
and make use of it's capabilities."
            },
            {
                "field" : "f0",
                "ftype" : "textfield",
                "title" : "Info text",
                "value" : ""
            },
            {
                "field": "f1",
                "ftype": "dropdown",
                "title": "Select logo scaling",
                "value": "0.3",
                "items": [
                    {
                        "text": "Tiny logo",
                        "value": "0.3"
                    },
                    {
                        "text": "Huge logo",
                        "value": "1.2"
                    }
                ]
            },
```

```
            {
                "field" : "f2",
                "ftype" : "textarea",
                "title" : "Multiline field",
                "value" : "First line\nSecond line\n\nFourth one"
            },
            {
                "ftype" : "divider"
            },
            {
                "field": "f3",
                "ftype": "filelist",
                "title": "Choose background image",
                "assetfolder" : "/media/images/bg/" ,
                "extension" : "png",
                "value": "/media/images/bg/checker.png",
            },
            {
                "field": "f4",
                "ftype": "number",
                "title": "Rotation degrees",
                "value": "45",
            }
        ]
    };
</script>
```

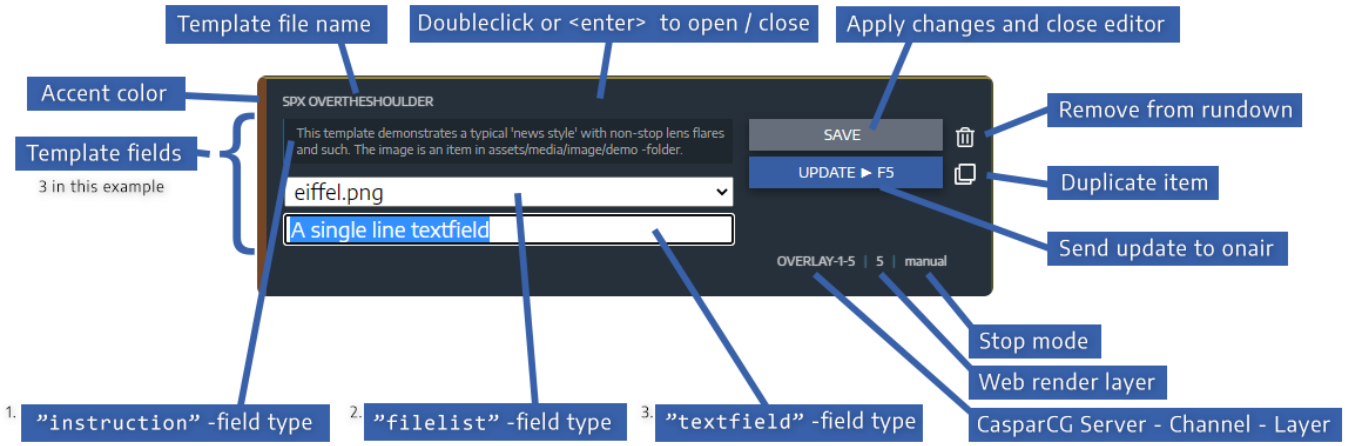
- **playserver**: one of the available CasparCG server names in config or "-" for none
- **playchannel**: CasparCG playout channel
- **playlayer**: CasparCG playout layer
- **webplayout**: a number between 1..20, or "-" for none
- **out**: how layer should be taken out:
  - `manual` default way: press STOP to animate out
  - `none` play only. Suitable for wipes / bumpers
  - `[numeric]` milliseconds until STOP is executed
- **steps**: how many phases in animation? For normal in-out templates this is 1. For templates with 2 or more steps the *Continue* button gets enabled.
- **dataformat**: how template is expecting data
  - `xml` the default
  - `json` used in some special templates
- **ftypes**
  - *ftypes* (for field types) define template's GUI controls in SPX-GC controller
  - the values of first two fileds are used as content preview in the rundown, so the order of fields should be considered for the ease of use
  - The developer of the HTML template can consider how to utilize these values, for instance a `dropdown` control can be used to pick the name of the show host, or it can drive other values via javascript in the templates. See /ASSETS/templates/smartpx -folder for some inspiration.

Field type	Description	Example

Field type	Description	Example
`hidden`	A variable which is not editable by the user. *Value* is used by the template and, *title* shown as static text on UI.	`Red color (#f00)`
`caption`	The *value* is shown in UI. Caption can be used to display texts to operators of the template.	`This template does not have editable values`
`textfield`	A typical single line text input field.	`Firstname Lastname`
`dropdown`	A dropdown selector list. Options is an *items* array, each consisting of *text* (which is visible) and the *value* (which the template will use). The default selection is defined as `value` and it must be one of the values in the *items* array. See an example definition above.	`"items":[ {"text": "Hundred", "value": 100}, {"text": "Dozen", "value": 12} ]`
`textarea`	A multiline text control which accepts *return* key for new lines. (Added in 1.0.2)	`First line \n Second line`
`filelist`	A dropdown selector list for files of of given type *extension* in an *assetfolder* within ASSETS -folderstructure of SPX-GC. This is useful for picking images or other media files in templates. (Added in 1.0.3)	`sport_logo.png, news_logo.png`
`divider`	A utility ftype to add a visual divider to a template. Can be used to create visual seqments for ease of use. (Added in 1.0.3)	
`instruction`	*Value* can be used as a longer help text on the template but does not have any other functionality. (Added in 1.0.6)	`Max 100 characters to the field below.`
`number`	*Value* is exposed as a number field in the template UI. (Added in 1.0.7)	`45`

> **Note** additional user interface controls may be added in future releases.

Anatomy of an example rundown item



Using SPX-GC with OBS / vMix / Wirecast...

SPX-GC's animated graphics and overlays can be integrated used in streaming and videoconferencing with any video- or streaming application which has a support for "Browser" or "HTML Sources". SPX-GC provides a URL address which is entered to the streaming software as a layer / input / source. In OBS use `Browser source`, in vMIX it's called `Web Browser input` and in XSplit it's a `Webpage source`...

```
http://localhost:5000/renderer
```

If you have several inputs (for instance for multiple presenters) you can limit which layers get's rendered to different screens with the `layers` parameter in Renderer url, for instance:

```
http://localhost:5000/renderer/?layers=[2,4,20]
```

Control SPX-GC with external devices such as Elgato Stream Deck...

SPX-GC (v.1.0.8+) rundowns can be loaded and controlled with external devices with http-get commands. See available commands here:

```
http://localhost:5000/api/v1
```

Issues and Feedback

Github issue tracker should be used for bug reports. For other feedback such as feature requests or other comments (for now at least) please use Google Forms feedback form at https://forms.gle/T26xMFyNZt9E9S6d8 or directly by email to tuomo@smartpx.fi.

> All constructive feedback is highly appreciated!

Gotcha's & Known Issues (things to be aware of)

- If UI becomes wonky reload the view (F5 / Ctrl+R).
- There is spagetti code whenever worked tired. Try to accept it...
- Undocumented features do exist. (templateEvents, TTS, pm2, cfg:hostname/usercommapass/greeting...)
- This list shouldn't be. At least not here.

Roadmap

New releases will try address found issues and bugs in older versions and they will also introduce new features and functionality. See table for some planned features and use feedback to submit suggestions.

A marketplace for SPX-GC compatible HTML -templates and plugins will open ~~by the end of 2020~~ soon.

> When a new version becomes available it will be promoted on the Welcome page of the application (if access to internet). Several versions can be installed (into different folders) and if there are no backwards compatibility issues between versions they can be configured to use the same dataroot and template -folder.

Release	Planned features (subject to change)	Timeframe
1.1	~~Mac install folder issue (#3) fix~~. Help page update, internal ~~logic change to fix playlist item issue (#1)~~, ~~http protocol for CasparCG templates~~, simple rundown view for mobile / tablet browsers, automatically running rundowns, item grouping, ~~textarea control~~, ~~item / file duplication~~. Project and ~~rundown~~ rename.	Q1/2021
X.X	Under consideration: mediafile picker, video playback control templates, additional preview modes (while editing, simulation, rtsp stream), MIDI controller integration, global extras editor in appconfig, ~~public API for controls~~, **HTML template marketplace**. Video tutorials. Knowledgebase. Forum. Slack support channel. Free lunches.	TBD

Strikethrough items are already done.

MIT License