Contents

User Guide

This guide explains how this resource is organized, how it is built, and how to maintain this resource.

Resources Organization

The source code repository[1]'s main branch is organized as follows:

path	description
`.github`	github templates and configuration for github actions
`code`	code examples
`docs`	additional helpful documents
`img`	all images
`labs`	lab exercises
`lectures`	lecture notes
`slides`	slides

[1] https://github.com/csci-1301/csci-1301.github.io

path	description
`templates`	templates and meta data files used for building this resource
`vid`	video files
`index.md`	website index page
`404.md`	website 404 page

Additional configuration files are at the root of the source code repository.

Locating course resources

How to obtain the latest version of this resource:

1. visit the accompanying website csci-1301.github.io[2]. This website includes the latest version of the course textbook in all supported formats, links to labs, and all other available student resources.
2. AU-hosted mirror of the website is located at https://spots.augusta.edu/caubert/teaching/csci-1301
3. latest version of built resource is available as a .zip file under releases[3] on Github.

A mirrored version of this website is hosted on spots[4] and updated regularly. Additional copies of this resource can be made available through box or D2L. Because manual effort is required to share the resource through these channels, these may be slightly behind the latest version.

How to obtain much earlier versions of this resource:

1. Complete release history is accessible under releases[5] on Github
2. Earlier versions of this resource will be periodically archived on Galileo.

Editing Resources

If you are new to this project, first read through Contributing Guidelines[6] to learn how you can contribute to the improvement of this resource, and if applicable, how to join a contributing team.

[2] https://csci-1301.github.io
[3] https://github.com/csci-1301/csci-1301.github.io/releases
[4] https://spots.augusta.edu/caubert/teaching/csci-1301/
[5] https://github.com/csci-1301/csci-1301.github.io/releases
[6] /contributing

Best practices for all forms of content

Inclusivity Follow the IT Inclusive Language Guide[7] from the University of Washington:

> use gender-neutral terms; avoid ableist language; focus on people not disabilities or circumstances; avoid generalizations about people, regions, cultures and countries; and avoid slang, idioms, metaphors and other words with layers of meaning and a negative history.

Structure for accessibility

- All resources are titled
  - title each markdown document: either in head meta or in markdown syntax, depending on resource type
  - use subtitles when appropriate
  - title all images with a descriptive title and add an alt-tag
  - title all code blocks in labs and lecture notes

- All resources are labelled when applicable
  - at minimum list prerequisites and security-related aspects
  - see Content Labelling for more details

Resources to assess accessibility:

- Affordable Learning Georgia's guide[8]
- Specific Review Standards from the QM Higher Education Rubric[9]
- UWG Accessibility Services's guide[10]
- Penn State's recommendations for alternative text and complex images.[11]

- WebAim Color Contrast Checker[12]
- WebAIM (Web Accessibility In Mind)[13]

Markdown

[7] https://itconnect.uw.edu/guides-by-topic/identity-diversity-inclusion/inclusive-language-guide/

[8] https://alg.manifoldapp.org/projects/oer-accessibility-series-and-rubric

[9] https://www.qualitymatters.org/sites/default/files/PDFs/StandardsfromtheQMHigherEducationRubric.pdf

[10] https://docs.google.com/document/d/16Ri1XgaXiGx28ooO-zRvYPraV3Aq3F5ZNJYbVDGVnEA/edit?ts=57b4c82d#

[11] https://accessibility.psu.edu/images/

[12] https://webaim.org/resources/contrastchecker/

[13] https://webaim.org/

- text documents are written in readme files using standard markdown syntax
- we will use a convention of always naming such files `readme.md` (lowercase)

Images

- Explain the image in written form.
- Title each image, this will create a URL for the image and enables linking to it.
- Always include a descriptive alt tag for accessibility.
- Do not rely on everyone seeing colors the same way[14].
- Prefer scalable vector images.
- Store images in the repository in `img` directory
  - When referring to images in markdown, use path from root, see example below
  - the image may appear broken locally, but pandoc will resolve the path at build time

Syntax example. The quoted text is the alt tag and in parentheses is path to file

```
!["image of visual studio IDE"](./img/vs_ide.jpg){ width=80% }
```

The `{ width=80% }` attribute is optional.

Source code

- source code programs belong *primarily* in `code` directory
  - the code included in this directory should be a complete program
  - the program should compile and terminate
  - source code that is faulty, partial, or does not terminate can be included in markdown as inline code block
  - we can automatically check these code snippets for syntactical correctness if these guidelines are followed
- code snippet can be included in markdown documents using pandoc-include filter:
- Title each source code block included in markdown, this will create a URL for the code block and enables linking to it.
- code blocks are by default annotated as `csharp`
  - syntax highlighting is applied automatically at build time based on the code block language

[14] https://www.wikiwand.com/en/Color_blindness

  - to use a language other than C#, specify the language locally in the specific code block:

- only include code in text form such that it can be copy-pasted for reuse
- make sure to include blank lines before and after code blocks, since the absence of these can cause the code block to display incorrectly.

Creating new lectures

All lecture notes are under `lectures` directory. This directory also contains an index indicating the related labs and prerequisites for each lecture.

To create a new lecture, e.g. `lecture xyz`:

1. Create a directory called `NNN_lecture_xyz` under `lectures` directory
    - Follow the existing pattern for naming convention which is lowercase and separation by underscores.
    - The numbers `NNN` tell pandoc how to order book content. Use leading zero and increments of 10.
    - Choose this number based on where in the book the new lecture should appear.
2. under the new directory, create a file `readme.md` (lowercase). Write lecture notes in this file using markdown.
    - We use filename `readme.md` because the build script looks for files matching this pattern.

Following these steps will automatically include the new lecture in the book.

If the lecture does not appear, here are the steps for troubleshooting the issue:

1. Check that after committing changes, the automated build has completed successfully
2. The newly created lecture is immediately under `lectures/` directory
3. The `readme.md` exists
4. In `gh-pages` branch, ensure the book is generated
5. Hard refresh the browser page if viewing the resources website

Do not include meta section in individual lecture files because these lectures will be concatenated by pandoc into a single larger document.

Any meta data in individual files would appear somewhere in the middle of the larger document, and as such will not be treated as front matter.

Known issues: When concatenating files pandoc may or may not include empty spaces between individual files. This may cause the subsequent lecture title to not appear in the generated book. For this reason, each lecture file should end with a newline.

Creating new labs

All lab resources are located under `labs` directory. At build time these labs are compiled into instructions in various document formats with an optional, accompanying source code solution.

1. Choose a short and unique name that describes the lab (say, `StringMethods`)
    - follow the existing convention for naming
    - do not number labs or make assumptions about numbering because another instructor may not follow the exact same lab order
2. Create a `labs/StringMethods.md` file
    - write lab instructions in this file. You should include meta data, at minimum a title
    - make the lab standalone to support alternative ordering (avoid assumptions about what was done "last time")
    - do not make assumptions about student using specific OS, include instructions for all supported options (Windows, MacOS, Linux)
    - do not make assumptions about student using Visual Studio, refer to IDE instead
3. (optional) if you want to include starter code with the lab,
    - go to `code/projets`,
    - create a subdirectory with the name of the solution you would like to use,
    - create a subdirectory with the name of the project you would like to use,
    - create a file called `Program.cs` in `code/projects/<solution>/<project>/Program.cs`
    - if you want to add additional classes, add them in `code/projects/<solution>/<project`
      files.

    Do **not** add solution (`sln`) or project (`csproj`) files: they will be created automatically using the project and solution's name you specified, if multiple classes are present they will all be linked,

and the resulting archive will be hosted in the lab's folder as `code/projects/<solution>.zip`.

Note / known issue: when including multiple solutions, the basename should be different, for example: `SomeLab` and `Solution_SomeLab` (instead of `SomeLabSolution`); to ensure solutions are packaged separately from one another.

4. (*obsolete?*) Create an entry for the new lab in the table at `labs/readme.md`. List all prerequisite labs and related lectures.

Using this established build system generates labs that are cross-platform (Windows, MacOS, Linux) and work on different IDEs. Do not attempt to create labs locally as that approach does not have the same cross-platform guarantee.

Content Labelling

(*obsolete?*)

Course resources are labelled with emoji shortcodes or text labels.

Each resource should, at minimum, list its prerequisites and security-related content.

Labelling with shortcodes

Use emoji shortcodes to label following course resources

Description | Shortcode | Icon |
: | | |
Security related aspects will be labelled as "security" | `:shield:` | 🛡 |
Optional parts will be labelled as "optional" | `:question:` | ❓ |
Elements to be incorporated in the future as "soon" | `:soon:` | 🔜 |

Labelling using text labels

1. Each resource will be labelled with prerequisites.

   This is a list of zero or more values. For zero prerequisites write `None`. These requirements are expressed in the associated index of lectures/labs/problems (cf. lectures[15]).

2. Lecture notes and slides will be labelled by related labs, and vice versa

[15] https://github.com/csci-1301/csci-1301.github.io/tree/main/lectures

These requirements are expressed in the associated index of lectures and labs (cf. lectures[16]).

Styling and Templating

Templating files are under `templates` directory.

Templates directory specifies layout files and stylesheets used in the website. These layouts are applied by pandoc when resources are built.

For maintainability reasons it is preferable to apply templates during build time. This strategy makes it easy to edit templates later and apply those changes across all resources. Avoid applying templating to individual resource files whenever possible.

Currently templates directory contains the following:

- `default-code-class.lua` - pandoc filter for annotating code blocks, configured to default to C#, which then allows applying syntax highlighting to all code block.
- `templates/labs` - templates used for generating lab resources and associated pages
- `templates/web` - templates for website and HTML format resources.

Updating docx template

First, output the default template file:

```
{bash} pandoc -o custom-reference.docx --print-default-data-file reference.docx
```

Then, open `reference.docx`, and, following loosely this tutorial[17], do:

- Click pretty much anywhere, and right-click on the highlighted style (displayed if you are under "Home", you may need to scroll down the styles),
- Change the font for everything but the source code,
- Click on the "Block code", then right-click on the highlighted style, and select the font for the source code,
- The font for "Verbatim Char" was also changed, but I am not sure if this has an impact,
- Make sure the fonts are embedded[18],
- Save and close the document.

[16] https://github.com/csci-1301/csci-1301.github.io/tree/main/lectures

[17] https://support.microsoft.com/en-us/office/customize-or-create-new-styles-d38d6e47-f6fc-48eb-a607-1eb120dec563?ui=en-us&rs=en-us&ad=us

[18] https://support.microsoft.com/en-us/office/benefits-of-embedding-custom-fonts-cb3982aa-ea76-4323-b008-86670f222dbc

This was inspired by this post[19] but does not seem to work properly :-/

Updating odt template

First, output the default template file:

```
{bash} pandoc -o custom-reference.odt --print-default-data-file reference.odt
```

Then, open `reference.odt`, and, following loosely this tutorial[20], do:

- View -> Styles
- Right-click on "Preformatted Text", click on "Modify…", and then select the desired font family for source code.
- In the dialog or sidebar which opens make sure the button in the top panel marked with ¶ is highlighted (it is very subtle).
- In the menu at the bottom of the dialog/sidebar choose Applied Styles. Only "Default Paragraph Style" and "Footer" should appear.
- Right-click on "Default Paragraph Style", click on "Modify…", and then select the desired font family for the rest of the text.
- Then, highlight the A next to ¶.
- Right-click on "Source_Text", click on "Modify…", and then select the desired font family for source code.
- File -> Properties -> Font tab, click on "Embed fonts in the document".
- Save and close the document.

Repository Maintenance

This repository uses following tools and technologies:

- git - version control
- Github - to make source code available on the web
- markdown, LaTeX - for writing the resources
- pandoc - for converting documents to various output formats
- make - for specifying how to build this resource
- github actions - to automatically build the resource
- github pages - to serve the accompanying website
- additional packages for specific tasks: texlive, Pygments, pandoc filters, lua filter[21], etc.
- fonts-symbola - to produce the emoji and other symbols in the pdf document.

[19] https://stackoverflow.com/a/70513063

[20] https://github.com/jgm/pandoc/wiki/Defining-custom-DOCX-styles-in-LibreOffice-(and-Word)#libreoffice

[21] https://github.com/jgm/pandoc/issues/2104

- utteranc.es[22] - for feedback through website
- csharpier[23] - to tidy the C## source code

Build outputs

The resource material is organized into specific directories (cf. resource organization). These resources are then compiled into templated documents in various formats using pandoc[24]. Different directories undergo different build steps as defined in the project Makefile[25] and generate various outputs. For example, lecture notes are compiled into a textbook and labs are packaged into individual labs. The makefile explains the exact steps applied to each type of resource.

Github actions

This resource is built automatically every time changes are committed to the main branch of the repository. This is configured to run on Github actions[26]. There are currently two configured workflows[27]: one to build the resource and to deploy it, and a second one to check that any opened pull requests can be built successfully.

The build configuration uses texlive to keep the dependency installation time low. Similarly, the choice of Python packages is preferable for pandoc filters, because they are usually straightforward and fast to install. We want to avoid choosing packages that significantly increase build time.

Currently Github actions offers unlimited free build minutes for public repositories (and 2000 min/mo. for *private* repositories, should we ever need them), which hopefully continues in perpetuity (if it does not there are other alternative services). Going with one specific CI service over another is simply a matter of preference.

Following a successful build, the build script will automatically deploy the generated resources to an accompanying website hosted on github pages[28]. In the repository a special branch `gh-pages` represents the contents of the deployed website. It also allows maintainers to observe the generated build outputs.

[22] https://utteranc.es/
[23] https://github.com/belav/csharpier
[24] https://pandoc.org/MANUAL.html
[25] https://github.com/csci-1301/csci-1301.github.io/blob/main/Makefile
[26] https://github.com/features/actions
[27] https://github.com/csci-1301/csci-1301.github.io/actions
[28] https://pages.github.com/

Creating releases

Currently a github action is setup to do the following: whenever a new commit is made to the main branch, the action will build the resource and add the generated books as a pre-release under releases and tag them as "latest". If a subsequent commit occurs it will overwrite the previous latest files and become the new latest version. This cycle continues until maintainers are ready to make a versioned release (or "package").

Making a versioned release is done as follows:

1. Go to repository releases[29]
2. Choose latest, which contains the files of the latest build
3. Edit this release, giving it a semantic name and a version, such as v1.0.0. Name and version can be the same. (cf. semantic versioning[30])
4. Enter release notes to explain what changed since last release
5. Uncheck "This is a pre-release"
6. Check "Set as the latest release"
7. Update release

Following these steps will generate a new, versioned release. The versioned releases will be manually uploaded to and archived on galileo.

Once this is done, remember to create the next pre-release:

1. Go to the repository releases[31].
2. Click on "Draft a new release".
3. Pick the tag "Latest".
4. Click on "Generate release notes"
5. Check "This is a pre-release"
6. Click on "Publish release"

Building locally

It is generally not necessary to build this resource locally unless the intent is to preview templating changes or to make changes to build scripts. For the purposes of editing content, it is sufficient to make edits to markdown files and commit those changes.

Installing dependencies To find the current list of dependencies needed to build this resource, refer to the build script install section[32],

[29] https://github.com/csci-1301/csci-1301.github.io/releases

[30] https://semver.org/

[31] https://github.com/csci-1301/csci-1301.github.io/releases

[32] https://github.com/csci-1301/csci-1301.github.io/blob/main/.github/workflows/build.yaml#L33-L40

which lists all required packages needed to build the resource. The exact installation steps vary depending on your local operating system.

In general the following dependencies are needed:

- pandoc[33]
- texlive[34]
- make
- python 3.+
- packages and filters: Pygments[35], pandoc-include[36], texlive-xetex[37], texlive-latex-extra, lmodern, librsvg2-bin[38]
- symbola font

For this later, note that starting with version 11[39], the licence is too restrictive for non-personal use. As a consequence, users are asked to make sure they do not use a version greater than v.10.24, which is "free for any use" and archived on-line[40] (curious users can also refer to the related webpage[41]). Note that installing this dependency using a unix-like package manager will result in installing a version of the font that is free to use in any context[42].

You can make sure you are currently using the latest version of panflute by running

```
pip install -U panflute
```

This is needed if running a recent version of pandoc (as of pandoc 3.1.6.1 at least).

Running the build After installing all dependencies, from the repository root, run:

```
make
```

To see a list of other alternative build options run

```
make help
```

[33] https://pandoc.org/installing.html
[34] https://www.tug.org/texlive/
[35] https://pygments.org/download/
[36] https://github.com/DCsunset/pandoc-include#installation
[37] https://tug.org/xetex/
[38] https://askubuntu.com/a/31446
[39] http://web.archive.org/web/20181228102842/http://users.teilar.gr/%7Eg1951d/Symbola.pdf
[40] http://web.archive.org/web/20180307012615/http://users.teilar.gr/~g1951d/Symbola.zip
[41] http://web.archive.org/web/20180307012615/http://users.teilar.gr/~g1951d/
[42] https://metadata.ftp-master.debian.org/changelogs//main/t/ttf-ancient-fonts/ttf-ancient-fonts_2.60-1.1_copyright

Maintaining repository feedback

Resource users can submit feedback about the resource through various means, one of which is leaving comments on the website. This feature is enabled by utteranc.es[43].

To manage user feedback over time, a semester-specific repository is created for issues only. This must be a public repository and located under the same organization as the resources repository. utteranc.es widget is configured to point to this repository. After a semester is over, this feedback repository will be archived, and a new one created for the next semester. This will simultaneously archive all older issues and reset the feedback across website pages.

Migrating feedback repository The steps for migrating feedback target repository are as follows:

1. Create a new **public** repository under `csci-1301` github organization. Follow the established naming convention, and leave all the options except for visibility (which needs to be set to public) by default.
2. Go to repository Issues (make sure issues is enabled in repository settings)
3. Create a new label whose *label name* is `comment` (to match widget configuration[44])
4. Go to `Organization Settings > Installed GitHub Apps`[45]
5. Choose "utterances" > "configure"
6. Under "Repository access" > "Only select repositories"
   - select the repository created in step 1.
   - remove the previous semester feedback repository
7. Save
8. In `csci-1301.github.io` repository open `/templates/web/template.html`
9. Update utteranc.es widget code to point to the new feedback repository created in step 1.

```
<script data-external="1"
        src="https://utteranc.es/client.js"
        repo="csci-1301/{REPOSITORY_NAME}"
```

[43] https://utteranc.es/

[44] https://github.com/csci-1301/csci-1301.github.io/blob/main/templates/web/template.html#L87-L94

[45] https://github.com/organizations/csci-1301/settings/installations

```
        label="comment" …>
</script>
```

10. Commit change to template.html
11. Make sure the feedback works after migration. If it does not, retrace your steps.
12. Archive the earlier feedback repository in its settings.

Maintaining Instructors / G/UCA rights

Every semester,

- The members of the "UCAs" team[46] should be updated,
- A "uca-resources–YYYY" repository should be created, by forking the template[47] (private repository),
- The new repository should be added to the list of repositories of the team[48] (as maintainer),
- The old repository should be deleted from that same list, and then archived.
- GRAs should be added / removed from the instructors list[49], and previous instructors should be removed from that same list,
- GRAs should be added to the "UCAs" teams[50] and given "maintainer" rights (*inside that team*, and not for the whole organization).

[46] https://github.com/orgs/csci-1301/teams/ucas
[47] https://github.com/csci-1301/uca-resources-template
[48] https://github.com/orgs/csci-1301/teams/ucas/repositories
[49] https://github.com/orgs/csci-1301/teams/instructors
[50] https://github.com/orgs/csci-1301/teams/ucas/members