ConTEXt basics for users: Conditional processing

Aditya Mahajan

Abstract

Very often, you want to generate multiple versions of the same document: one version for printing and one for viewing on the screen, one version for students and one version for the instructor, and so on. You can do this in a simple but naive way: create different files set up for the different versions and `\input` the common material, or create some new conditional flags using `\newif` and set them appropriately for conditional processing. Or you could use *modes* — the ConTEXt way of doing conditional processing.

1 Introduction

A mode is similar to a conditional flag, but with a few advantages: new modes need not be explicitly defined (no need for something like `\newif`), multiple modes can be simultaneously enabled or disabled, and the status of multiple modes can be checked easily. Moreover, modes can be set from a command line switch. As a result, multiple versions of a document can be generated without changing the source file.

The name or identifier of a mode can be any combination of letters, digits, or spaces. Names starting with `*` are reserved for system modes.

In this article I explain how to activate a mode and how to check if a mode is active or not.

2 Setting modes

ConTEXt has three commands for setting modes:

- `\enablemode [...]`
- `\disablemode[...]`
- `\preventmode[...]`

The names are self-descriptive. `\enablemode` activates a mode, `\disablemode` deactivates a mode, and `\preventmode` permanently deactivates a mode. All three commands take a list of modes as an argument. For example, you can activate modes named `screen` and `solution` with

```
\enablemode[screen,solution]
```

Modes can also be activated by a command line switch `--modes` to `texexec` or `context`. For example, another way to activate the `screen` and `solution` modes, to run ConTEXt using one of:

```
texexec --mode=screen,solution ...
context --mode=screen,solution ...
```

3 Conditional processing based on modes

You may want to process or ignore a chunk of code if a particular mode is enabled or disabled. Such a chunk of code is specified using `\startmode` and `\startnotmode` environments. Their usage is best explained by an example.

Suppose you want to change the paper size of a document depending on whether it is for print or screen. This can be done in multiple ways. You could set the default paper size for print and change it in screen mode:

```
\setuppapersize[letter][letter]
\startmode[screen]
  \setuppapersize[S6][S6]
\stopmode
```

(S6 is one of the screen-optimized paper sizes in ConTEXt; the paper size has a 4:3 aspect ratio and a width equal to the width of A4 paper.)

Alternatively, you could set a default paper size for the screen and change it if screen mode is not enabled:

```
\setuppapersize[S6][S6]
\startnotmode[screen]
  \setuppapersize[letter][letter]
\stopnotmode
```

`\startmode` and `\startnotmode` can check for multiple modes, by giving a list of modes as their arguments. `\startmode` processes its contents (everything until the next `\stopmode`, thus `\startmode` cannot be nested) if any of the modes are enabled, otherwise (i.e., when all the modes are disabled) `\startmode` ignores its contents. The opposite is `\startnotmode`: it processes its contents (everything until the next `\stopnotmode`) if any of the modes are disabled, otherwise — when all the modes are enabled — the contents are ignored.

`\startmode` and `\startnotmode` are "*or*" environments. They process their contents if any of the modes satisfy the required condition. Their "*and*" counterparts are also available: `\startallmodes` and `\startnotallmodes` process their contents only if all the given modes satisfy the required condition. For example, suppose you want to enable interaction (e.g., hyperlinks) only when both `screen` and `solution` modes are enabled. Then you can use:

```
\startallmodes[screen,solution]
  \setupinteraction[state=start]
\stopallmodes
```

To summarize, the four start-stop environments for checking modes are:

```
\startmode[mode1, mode2, ...]
  % Processed if any of the modes is enabled
\stopmode
```

```
\startnotmode[mode1, mode2, ...]
  % Processed if any of the modes is disabled
\stopnotmode

\startallmodes[mode1, mode2, ...]
  % Processed if all the modes are enabled
\stopallmodes

\startnotallmodes[mode1, mode2, ...]
  % Processed if all the modes are disabled
\stopnotallmodes
```

These environments have `\doif...` alternatives that are useful for short setups. Also, they can be nested.

```
\doifmode        {modes} {content}
\doifnotmode     {modes} {content}
\doifallmodes    {modes} {content}
\doifnotallmodes {modes} {content}
```

The logic for determining when the content is processed is exactly the same as for the `start-stop` commands.

These `\doif` commands each have a variant to process alternative code if the conditions are not satisfied (like the `\else` branch of `\if`).

```
\doifmodeelse       {modes} {content} {alt}
\doifnotmodeelse    {modes} {content} {alt}
\doifallmodeselse   {modes} {content} {alt}
\doifnotallmodeselse{modes} {content} {alt}
```

4 System modes

Besides allowing user-definable modes, ConTEXt provides some system modes. These modes start with a `*` character. Here I will explain only the more commonly used system modes; see the ConTEXt modes manual (`http://pragma-ade.com/general/manuals/mmodes.pdf`) for a complete list.

Perhaps the most useful system modes are `*mkii` and `*mkiv` which determine whether MkII or MkIV is being used. These modes are handy when you want different setups for MkII and MkIV.

Other modes are useful for very specific situations. Some of these are described below.

A document must be run multiple times to get the cross referencing, table of contents, etc. right. However, sometimes you need to do some external processing (e.g., graphic conversion) that only needs to be done once. In such cases, the `*first` mode is handy — it is active only on the first run of the document.

You can use the project-product-component structure for managing large projects like a book series. See the ConTEXt wiki article (`http://wiki.contextgarden.net/Project_structure`) for details of this approach. A product or its components may be compiled separately, and you may want to do something different when a product is compiled or when a component is compiled. To do so, you need to check for modes `*project`, `*product`, `*component`, and `*environment`; these modes are set when the corresponding structure file is processed. For example, the `*product` mode is set whenever a product file is read; more specifically, when `\startproduct` is encountered. Similarly, a mode `*text` is enabled when `\starttext` is encountered, and likewise for the others.

A large document is typically broken down into different section blocks: frontmatter, bodymatter, appendices, and backmatter. Internally, these section blocks are referred to as `frontpart`, `bodypart`, `appendix`, and `backpart`. Each section block sets a system mode with the same name. So, if you want macros that work differently in different section blocks, you can check for modes `*frontpart`, `*bodypart`, and so on.

ConTEXt provides support for multiple languages. Languages are recognized by their IETF language tags, like `en-us` for US English, `en-gb` for British English, `nl` for Dutch, `de` for German, etc. A document has a main language, set with the command `\mainlanguage[...]`, that is used for translated labels like *chapter* and *figure*. You can also switch the current language using `\language[...]` to change the hyphenation rules. Whenever a language is chosen, its identifier is set as a mode. The mode for the main language starts with two `*`. For example, when the main language is US English and the current language is Dutch, the modes `**en-us` and `*nl` are set (notice the extra `*` in `**en-us`).

Other system modes: `*figure` is set when a graphic is found, `*interaction` is set when interaction is enabled, `*grid` is set when grid typesetting is enabled, and `*pdf` and `*dvi` are set when the output is PDF or DVI. Others are too esoteric to describe here. If you are interested, see the modes manual mentioned earlier.

In summary, modes provide generalized conditional processing. A rich set of built-in modes is available.

◇ Aditya Mahajan
`adityam (at) ieee dot org`