Document number:	P2193R1	
Audience:	SG20	
Date:	2020-07-02	
Reply to:	JC van Winkel	(jcvw@google.com)
	Bjarne Stroustrup	(bs@ms.com)
	Florian Sattler	(sattlerf@cs.uni-saarland.de)

How to structure a teaching topic

Introduction

p1725 "Modular Topic Design" describes a modular topic-based model for creating Guidelines for C++ curriculum development. In the document you are reading now, we will present a brief HOWTO assisting people wanting to write teaching guidelines for a topic. We also have a skeleton file here that gives you the layout of a topic.

We also have two (not 100% finished) sample topics:
- Copy Semantics
- User defined Literals

Goal

We want to give guidelines to people who will write a C++ curriculum. These guidelines do **not** dictate how you should teach C++, or what angle you should approach teaching from (compare for example a C++ course for embedded programmers with a course for people with a UI background). Instead, the guidelines are intended to help organize topics, to document dependencies among topics, and offer a common structure of descriptions. Remember "form is liberating", so use this suggestion as an initial approach to each topic.

Many topics are best introduced in stages:

- **Foundational** descriptions give the learner the idea that a facility exists, what benefits it offers, and the basic ways of using it.
- **Main** descriptions go into greater detail. It shows mainstream uses and techniques. For abstraction and organizational mechanisms it also demonstrates how to build them. Note that a main description typically cannot be written concisely without relying on the foundational sections for a variety of facilities. This is a major reason to separate

foundational and main sections of the presentation of a facility. The main section should give the learner a basic (but not detailed) understanding of how a facility might be implemented so that the learner can have a first-order understanding of any costs involved. Avoid language lawyering in the main section.

- **Advanced** descriptions. For most topics there is an expert level of knowledge that most programmers rarely need and techniques that require detailed understanding of language rules or library implementation. Delving into advanced topics early would be distracting and reinforce C++'s unfortunate reputation of being "expert only." The advanced features often are "expert only" but the foundational and main descriptions should not be. Finally, we explicitly mention subjects that are considered too advanced for inclusion in the guidelines.

To help the teacher and the learners navigate the guidelines, these sections should typically be cross-linked to each other as well as to the appropriate sections that they rely on or rely on them..

Never explain a feature or a principle without an example and never give an example without explaining what feature or principle it is an example of.

For example, consider the "User defined literals" topic:

- The foundational level describes built-in suffixes and how we can add and use our own suffixes to offer literals of user-defined types. Examples, such as time literals, std::string literals, and imaginary parts of complex numbers can be used. Without going into details, a simple definition of a literal operator can be shown (e.g., imaginary).
- The main section goes into some detail about how to implement UDLs (e.g., showing how to implement the std::string suffix), how to use constexpr, the use of namespaces, and the dangers of overusing UDL. A list of standard-library suffixes is provided.
- The advanced section shows more advanced techniques (e.g., a formatting keyword `"{}, {}!"_fmt("Hello", "World")`, suffixes for the SI unit, or a detailed examination of the <chrono> unit system. Initially the advanced section can be simply a list of references to information about "interesting and/or important uses beyond the scope of this presentation."

Sections

Topic guidelines should follow the skeleton (we are carefully avoiding the word *template* here...) This way, all guidelines will look similar and once we have a decent collection, people will have an easier time finding exactly what they need for a particular topic, despite that the guidelines are written by many different people. Importantly, the common structure will help writers of new topics.

Below, we will describe the outline that topic guidelines should follow. Note that every guideline will contain (in cursive text) a description of that section as it is mentioned in the skeleton. This reinforces the proper use of the skeleton. As a running example, we will use the User defined Literals topic description, typeset in blue and a different font like this

Overview

Provides a short natural language abstract of the module's contents. Specifies the different levels of teaching.

The Overview should give a brief description of the topic. It also defines the objectives of the three levels that every topic should have.

For UDLs, the following table is listed:

Level	**Objectives**
Foundational	using and understanding UDLs
Main	implementing your own UDLs
Advanced	Advanced use (`"{}, {}!"_fmt("Hello", "World")`)

We deliberately separate out the foundational and advanced levels. A course would not teach everything about a topic in one go, but first discuss the foundational level subjects and then revisit the topic later on to add the advanced subjects. In the "Further studies" section, the guidelines will not discuss the subjects, but may refer to external material. The "Further studies" section will list subjects that are deemed too advanced for general consumption.

Motivation

Why is this important? Why do we want to learn/teach this topic?

This section explains the importance of teaching this topic. Potentially some examples are given.
The UDL topic has:

- Allows clearer expression of intent in C++.
- `std::string`: `"Hello, world!"s`
- `std::chrono`: `3h + 10m + 5s`

Topic introduction

Very brief introduction to the topic.

This section gives a brief introduction to the topic (note that the reader should understand the topic - we are not teaching about the topic, we are giving guidelines to how to **teach** the topic)

The example for the UDL topic is:

- Explain the existence of user defined literals. Example: `12m + 17s` is terse, expressive and type safe.

Foundational: <item>

This section is the core of the topic guideline. It has several subsections: Background/Required Knowledge, Student outcomes, Caveats, and Points to cover. These subsections are repeated in the "Main" section. Note that the "Foundational knowledge" should contain only absolutely basic subjects. The idea is to revisit topics later after more other foundational (and perhaps advanced) topics have been discussed.

For the UDL example, the section heading is: "Foundational knowledge: Using UDLs"

Let's go through these subsections and give examples from the UDL chapter.

Background/Required Knowledge

This is a list of things a student should already be familiar with/master. This will most probably be a bullet list, starting with "A student…". Using the UDL as an exampleagain:

A student:

- knows how to form numeric literals, e.g., `1.5f` means a float of value `1.5`.
- is familiar with the major C++ types:
  - `bool` (Boolean type)
  - `int` (Integer type)
  - `double` (Floating-point type)
  - `std::string` (Text type)
  - `std::vector` (Collection type)
- knows that namespaces exist, and namespace `std`.
- knows what using-declarations and using-directives are.

The author of a curriculum can assume the student already knows about these topics, so they do not have to be explained in this topic. If possible, the mentioned requirements should refer to other topics that describe them and contain a link to the respective topic.

Student outcomes

A list of things "a student should be able to" after the curriculum. The next word should be an action word and testable in an exam. Max 5 items.

The student outcomes should start with an action word, something that a student should actually be able to do. So don't use: "As student should be able to ~~understand how to use a vector and visit all elements~~" but rather something like "write a function that uses a vector and visits each element".

Looking at the UDL example:

A student should be able to:

- use `using namespace std::string_literals` [1].
- recognise UDLs in code that they are reading.
- figure out which UDL definitions for a used type exist.
- identify parts of the standard library that make use of UDLs.
- prevent the dangers of temporaries created with `"blah"s` as well as with `std::string{"blah"}`.
- effectively select the right set of namespaces in using-directives from the sub-namespaces `std::literals`.

[1]: explain that it's okay to use a using-directive to "activate" UDLs.

Caveats

This section mentions subtle points to understand, like anything resulting in implementation-defined, unspecified, or undefined behavior.

This section should explain what areas to pay extra attention to for this topic (at the foundational level!) when creating a curriculum. From the UDL topic:

- A student gets confused by the similarities and differences between built-in suffixes and UDLs and between UDLs from different namespaces.
- A student "activates" two suffixes with the same signature from different namespaces.

Points to cover

This section lists important details for each point.

This section will contain a list of subjects to discuss for this topic in a curriculum, again, at the foundational level.

Main <item>

In this section, we will see the same subsections as in the foundational section, but then aimed at a more advanced level, when having mastered the foundational level and perhaps other foundational and advanced topics. For the UDL case, the heading is:
"Advanced: implementing UDLs"

The "Background/Required Knowledge" subsection will most probably contain quite some references to the foundational section, but can also refer to other foundational and advanced topics.

Advanced:

These are important topics that are not expected to be covered but provide guidance where one can continue to investigate this topic in more depth.

This section mentions (only: mentions, not: discusses) subjects in this topic that are only used in a very advanced application of the facilities in this topic. If applicable, references for further study can be provided.

Finally

When writing a topic, remember you are not teaching that topic to the reader - you are giving guidelines of what to mention and what to avoid to the author of a C++ curriculum.