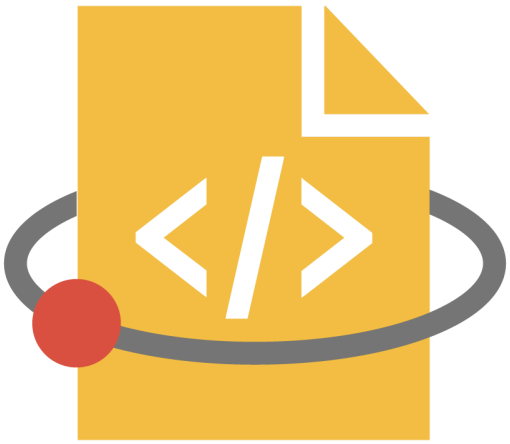

Technical Writer Progress report
Writer - Rachitt Shah

Summary -
This report is a summary of the objectives identified by the writer upon communication with the HPX team members. Broadly, this document has 2 core competencies -

1. Change the API documentation.
2. Look for other platforms for documentation, as a start.

Changing the API documentation

As per the needs of the project, the APIs have been identified as a structure needing change. As per the call with Nikunj on Monday. We had the following agreements -

Parameters:

`args`	The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction objects. The function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last) should expose a signature equivalent to:
	`* <ignored> pred(I const& a, ...);`
	`*`
	The signature does not need to have const&. It will receive the current value of the iteration variable and one argument for each of the induction or reduction objects passed to the algorithms, representing their current values.

`first`	Refers to the beginning of the sequence of elements the algorithm will be applied to.
`policy`	The execution policy to use for the scheduling of the iterations.
`size`	Refers to the number of items the algorithm will be applied to.

Template Parameters:

`Args`	A parameter pack, it's last element is a function object to be invoked for each iteration, the others have to be either conforming to the induction or reduction concept.
`ExPolicy`	The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
`I`	The type of the iteration variable. This could be an (input) iterator type or an integral type.
`Size`	The type of a non-negative integral value specifying the number of items to iterate over.

Returns: The *for_loop_n* algorithm returns a *hpx::future<void>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *void* otherwise.

Function template for_loop_n

hpx::parallel::v2::for_loop_n

Synopsis

```
// In header: <hpx/parallel/algorithms/for_loop.hpp>

template<typename ExPolicy, typename I, typename Size, typename... Args>
  unspecified for_loop_n(ExPolicy && policy, I first, Size size,
                         Args &&... args);
```

Description

The for_loop_n implements loop functionality over a range specified by integral or iterator bounds. For the iterator case, these algorithms resemble for_each from the Parallelism TS, but leave to the programmer when and if to dereference the iterator.

Requires: *I* shall be an integral type or meet the requirements of an input iterator type. The *args* parameter pack shall have at least one element, comprising objects returned by invocations of *reduction* and/or *induction* function templates followed by exactly one element invocable element-access function, *f*. *f* shall meet the requirements of MoveConstructible.

Effects: Applies *f* to each element in the input sequence, with additional arguments corresponding to the reductions and inductions in the *args* parameter pack. The length of the input sequence is last - first.

The first element in the input sequence is specified by *first*. Each subsequent element is generated by incrementing the previous element.

> **Note**
> As described in the C++ standard, arithmetic on non-random-access iterators is performed using advance and distance.
>
> The order of the elements of the input sequence is important for determining ordinal position of an application of *f*, even though the applications themselves may be unordered.

Along with an element from the input sequence, for each member of the *args* parameter pack excluding *f*, an additional argument is passed to each application of *f* as follows:

If the pack member is an object returned by a call to a reduction function listed in section, then the additional argument is a reference to a view of that reduction object. If the pack member is an object returned by a call to induction, then the additional argument is the induction value for that induction object corresponding to the position of the application of *f* in the input sequence.

Complexity: Applies *f* exactly once for each element of the input sequence.

Remarks: If *f* returns a result, the result is ignored.

As per the old documentation, the API docs are well structured. The content for both the documentation is similar, however, in the newer documentation, the docs for parameters, APIs, description and synopsis is scattered all across the page in a less readable format.

Solution - changing our Sphinx theme is a good way to make it more readable, or edit our preexisting themes to create changes as per needed.

Proposed themes -

1. https://sphinx-themes.org/sample-sites/sphinx-theme-pd/
2. https://sphinx-themes.org/sample-sites/sphinx-pdj-theme/

Assumed changes would need to be made to the docs folder. However, I wasn't able to find a config.py file which usually has the docs for templates.

Other possible platforms to use for documentation -

As our core documentation doesn't need changes, I believe it's good to have the following sections on docusaurous - https://codesandbox.io/s/docusaurus , https://docusaurus.io/ , Example - https://docsearch.algolia.com/

Pros - skilled needed are really less, and can create beautiful modern websites with only the knowledge of basic HTML.
Cons - our API documentations is not covered.