Documentation Improvements

Chapel Team, Cray Inc.

Chapel version 1.14

October 6, 2016

Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

Outline

- Primers & Hellos
- chpldoc “use” information
- Chapel Language Man Page
- Users Guide Improvements
- Other Documentation Improvements

Primers & Hellos

Primers & Hellos: Background

- **New users often pointed to “primers” and “hello worlds”**
  - Appearance is important
    - Likely the first Chapel code new users will see
  - Actual ‘documentation’ contained within the source comments

- **Hello Worlds**
  - Accessible via chapel.cray.com or repository
    - Cumbersome to maintain – updates require updating web / repo version
    - Majority of source file is made of comment blocks

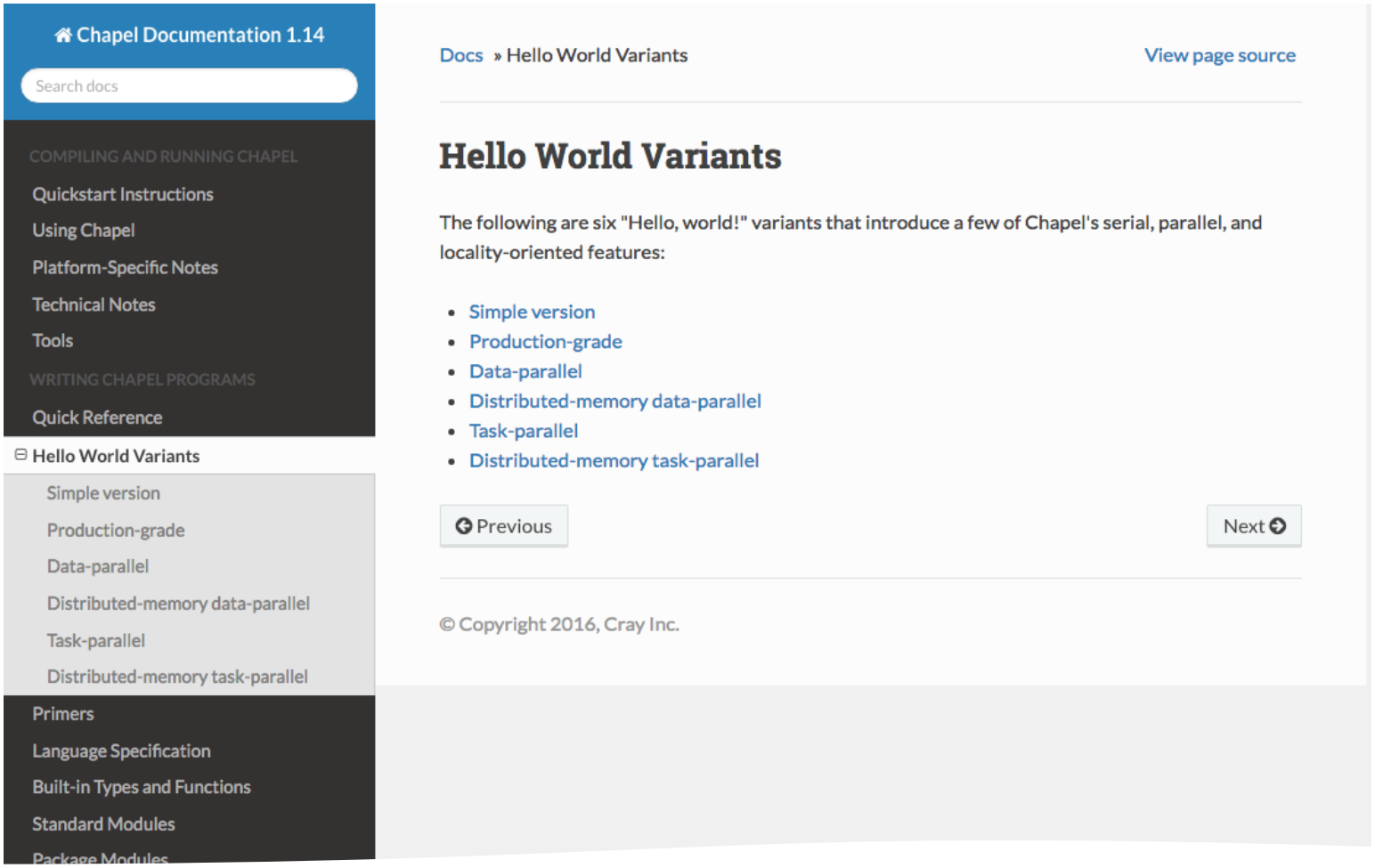
- **Primers**
  - Accessible via repository
    - Typically link users to the Github URL
    - No organized index or URL from the website

Primers & Hellos: This Effort

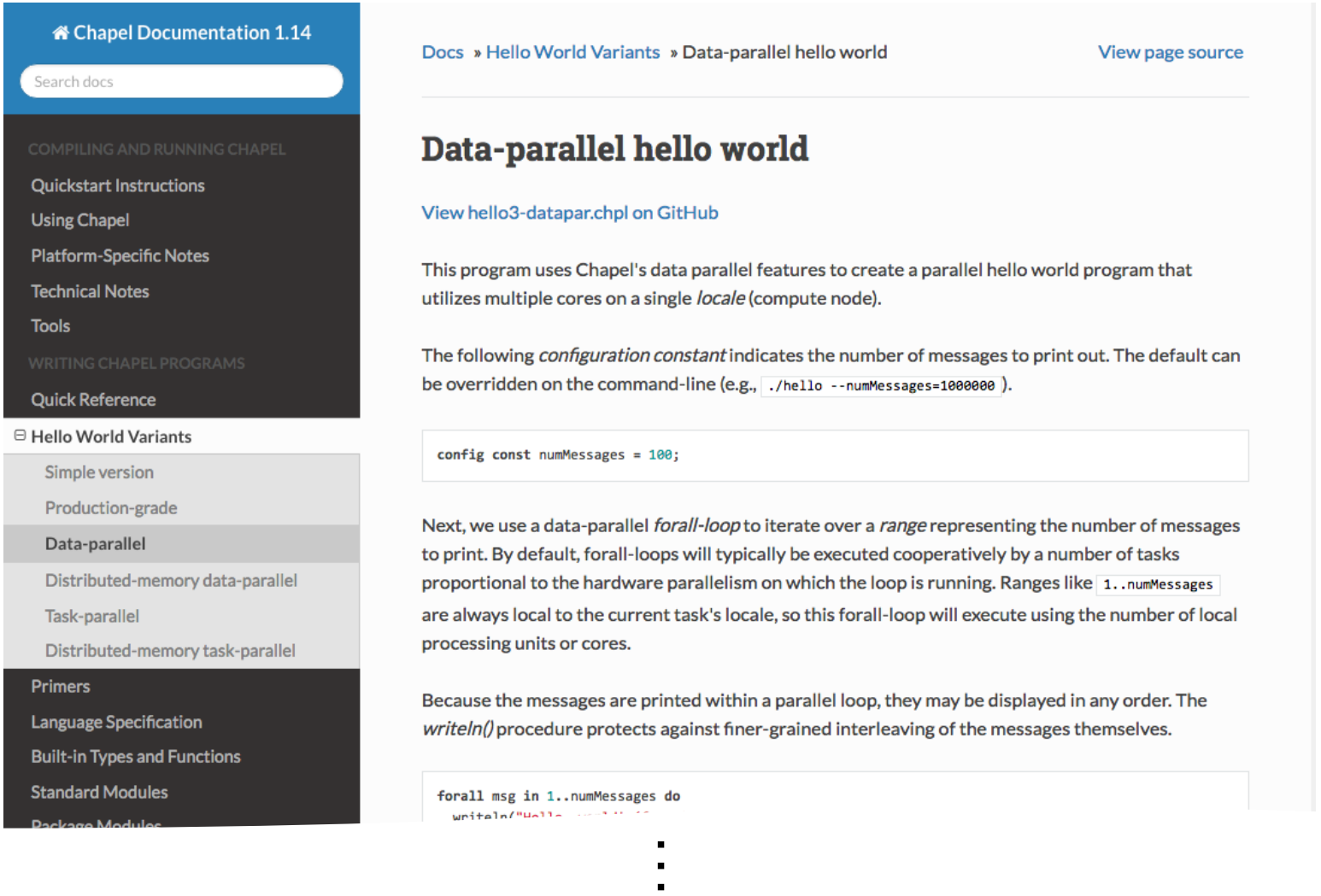
- **chpl2rst.py script converts Chapel -> reStructured Text**
  - Intended for rendering a source file into a tutorial-style rst file
  - Different than chpldoc in several aspects
    - Comment blocks and unindented line comments rendered as plain text
    - Code and indented line comments rendered as code blocks
    - Title and reference label auto-generated, for cross-referencing
    - GitHub URL to actual source code inserted at top of file
  - reStructured Text is rendered into HTML via Sphinx

- **Primers and Hellos are now included with online docs**
  - Primers and Hellos were edited to provide clean renders
  - Online docs include an organized index
  - Primers and Hellos are included as top-level links in sidebar

Primers & Hellos: Hello Worlds Index



Primers & Hellos: Data-parallel hello world



Chapel Documentation 1.14

Search docs

COMPILING AND RUNNING CHAPEL
- Quickstart Instructions
- Using Chapel
- Platform-Specific Notes
- Technical Notes
- Tools

WRITING CHAPEL PROGRAMS
- Quick Reference
- Hello World Variants
  - Simple version
  - Production-grade
  - Data-parallel
  - Distributed-memory data-parallel
  - Task-parallel
  - Distributed-memory task-parallel
- Primers
- Language Specification
- Built-in Types and Functions
- Standard Modules
- Package Modules

Docs » Hello World Variants » Data-parallel hello world — View page source

Data-parallel hello world

View hello3-datapar.chpl on GitHub

This program uses Chapel's data parallel features to create a parallel hello world program that utilizes multiple cores on a single *locale* (compute node).

The following *configuration constant* indicates the number of messages to print out. The default can be overridden on the command-line (e.g., `./hello --numMessages=1000000`).

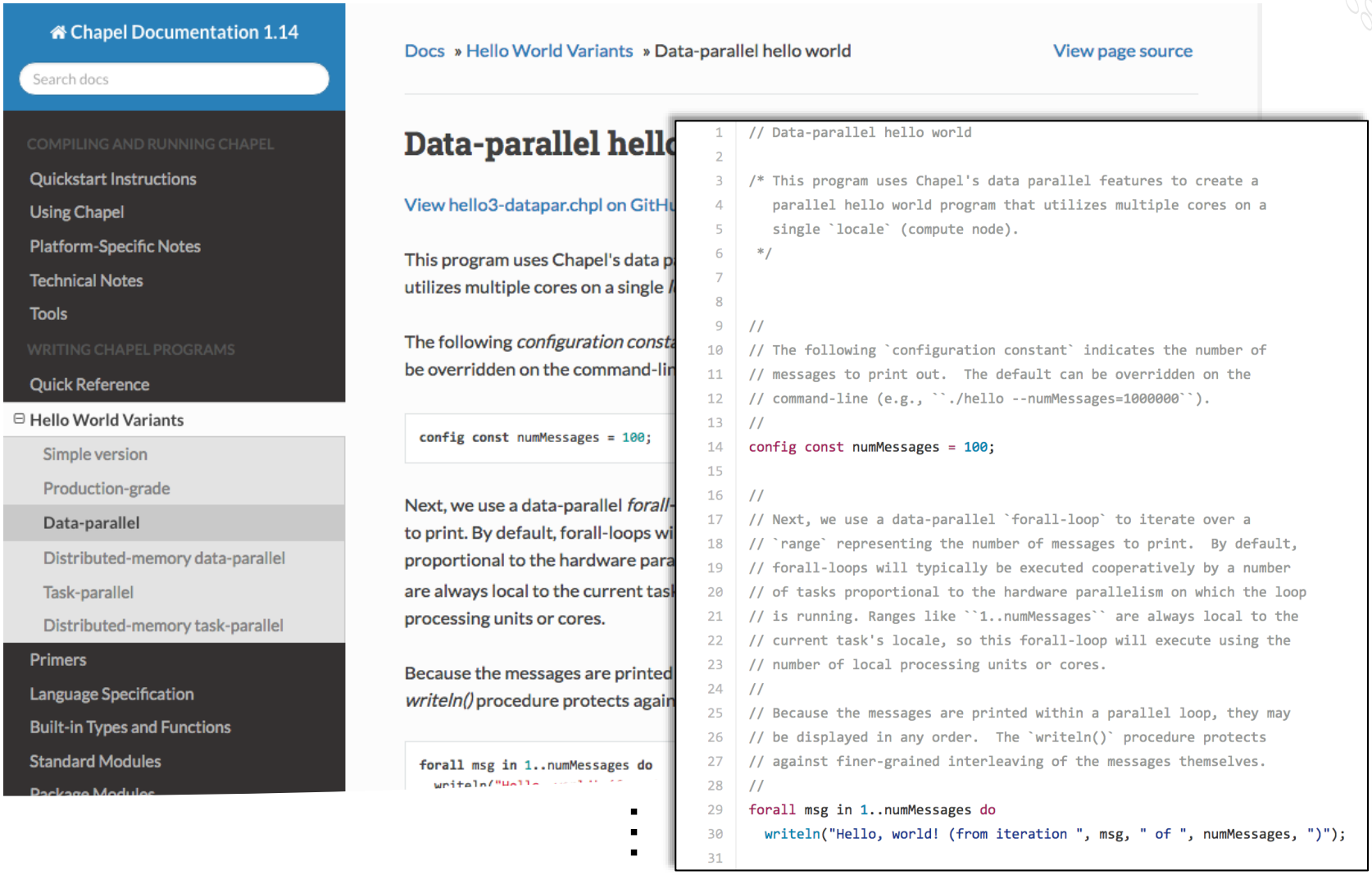
```
config const numMessages = 100;
```

Next, we use a data-parallel *forall-loop* to iterate over a *range* representing the number of messages to print. By default, forall-loops will typically be executed cooperatively by a number of tasks proportional to the hardware parallelism on which the loop is running. Ranges like `1..numMessages` are always local to the current task's locale, so this forall-loop will execute using the number of local processing units or cores.

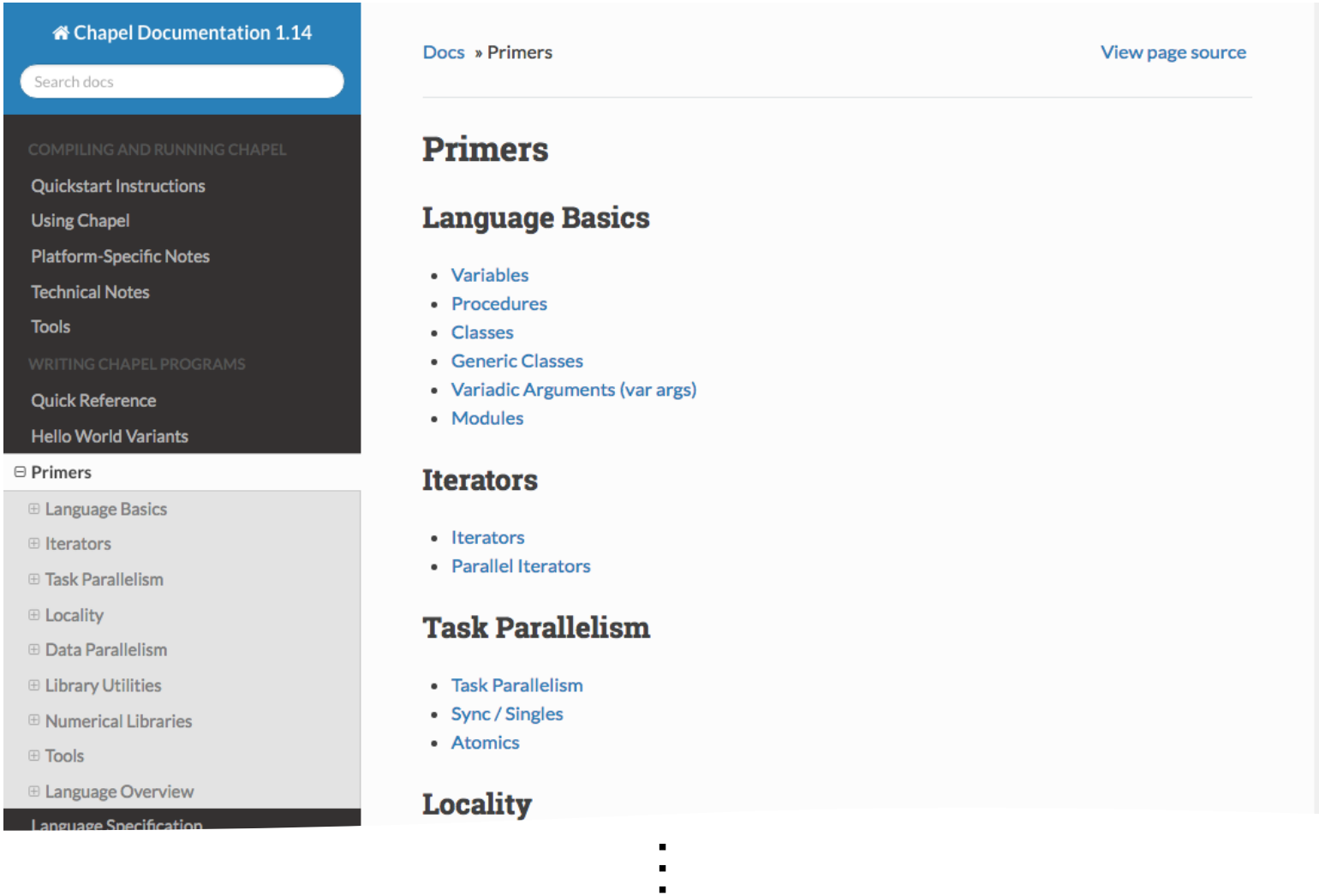
Because the messages are printed within a parallel loop, they may be displayed in any order. The *writeln()* procedure protects against finer-grained interleaving of the messages themselves.

```
forall msg in 1..numMessages do
```

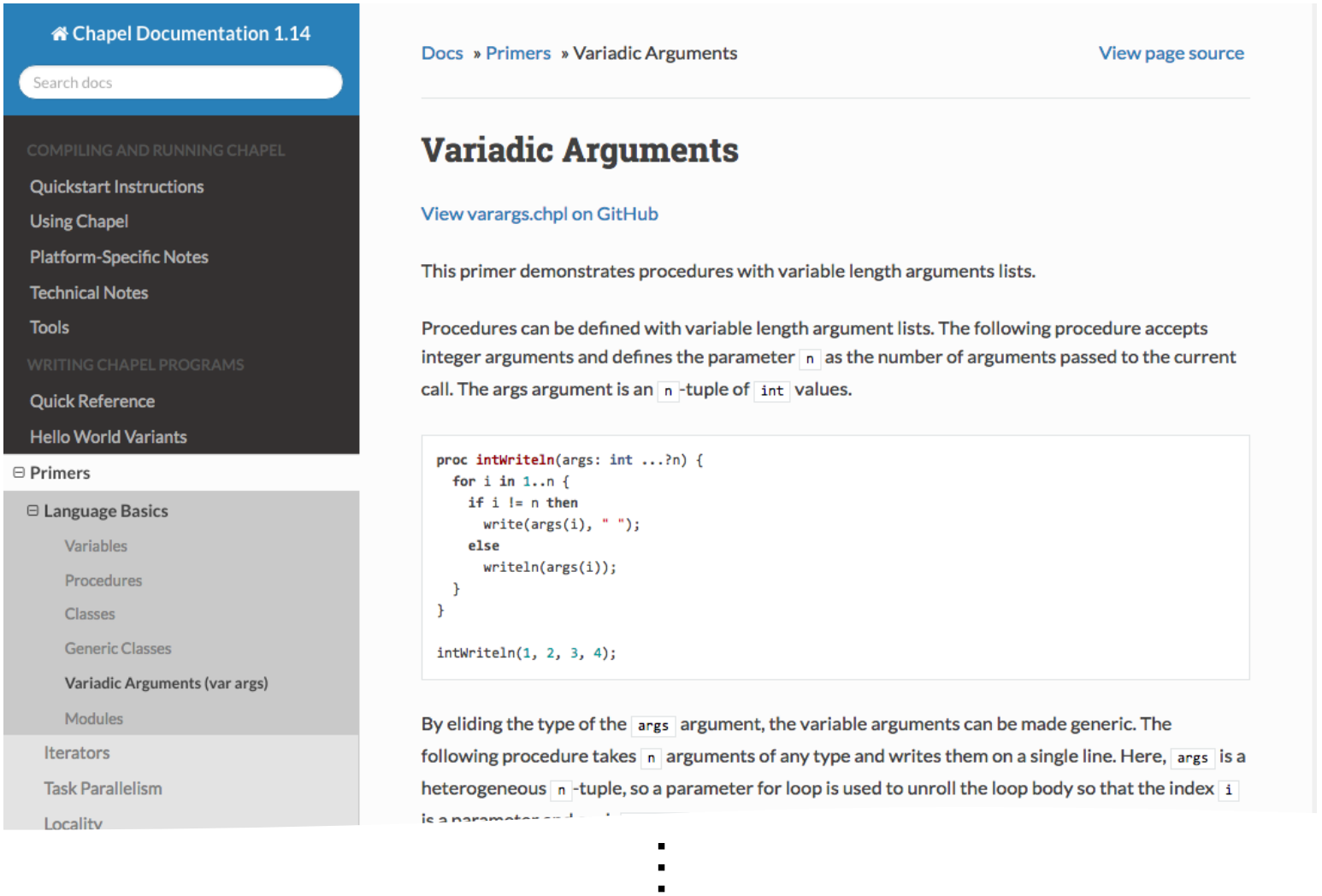
Primers & Hellos: Data-parallel hello world



Primers & Hellos: Primers Index



Primers & Hellos: Primers - Variadic Arguments



Chapel Documentation 1.14

Search docs

COMPILING AND RUNNING CHAPEL
- Quickstart Instructions
- Using Chapel
- Platform-Specific Notes
- Technical Notes
- Tools

WRITING CHAPEL PROGRAMS
- Quick Reference
- Hello World Variants

⊟ Primers

⊟ Language Basics
- Variables
- Procedures
- Classes
- Generic Classes
- Variadic Arguments (var args)
- Modules

Iterators

Task Parallelism

Locality

Docs » Primers » Variadic Arguments — View page source

Variadic Arguments

View varargs.chpl on GitHub

This primer demonstrates procedures with variable length arguments lists.

Procedures can be defined with variable length argument lists. The following procedure accepts integer arguments and defines the parameter `n` as the number of arguments passed to the current call. The args argument is an `n`-tuple of `int` values.

```
proc intWriteln(args: int ...?n) {
  for i in 1..n {
    if i != n then
      write(args(i), " ");
    else
      writeln(args(i));
  }
}

intWriteln(1, 2, 3, 4);
```

By eliding the type of the `args` argument, the variable arguments can be made generic. The following procedure takes `n` arguments of any type and writes them on a single line. Here, `args` is a heterogeneous `n`-tuple, so a parameter for loop is used to unroll the loop body so that the index `i` is a parameter and

⋮

Primers & Hellos: Status and Next Steps

Status:

- Intro Chapel code is more accessible and prettier
- Hello Worlds are more maintainable

Next Steps:

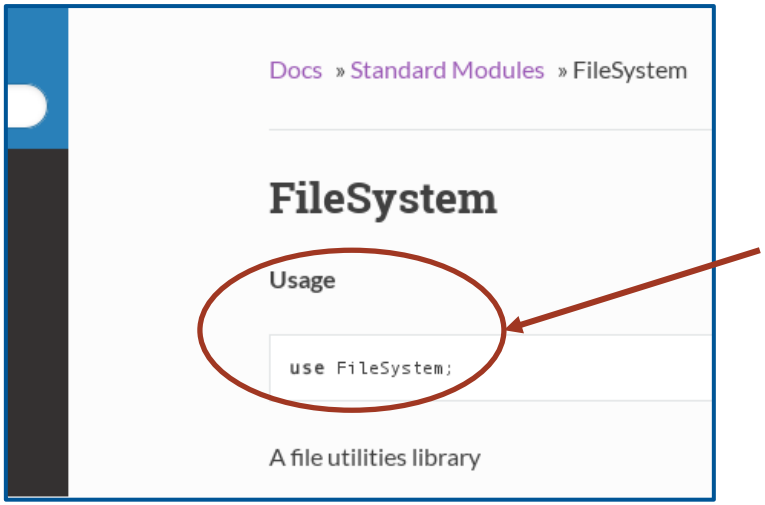
- Continue to improve primers breadth and depth
- Modify reference labels to reflect source filenames
  - Improves readability of cross-references in source code
- Minor feature additions to chpl2rst.py script
  - Add a way to render sequences: /* and */
  - Add a way to maintain indentation across code blocks

chpldoc “use” information

chpldoc “use” information

Background: No information on how to access module in docs

This Effort: Now generate a sample use statement for module



Impact: Users can copy+paste the use directly into their code

Chapel Language Man Page

Chapel Language Man Page

Background: Documentation built by sphinx into html

- Sphinx supports many other output types, but we only supported html
- Building to a man page resulted in errors

This Effort: Officially support building the docs as a man page

Impact: Users & Developers can search docs from CLI

- This man page contains all documentation that comes in html docs
- Accessed via the language man3 page:

```
man chapel
```

Next Steps: Consider other outputs to support

- e.g., individual man pages, pdf, JSON

Users Guide Improvements

Users Guide Improvements

Background:

- Started creating online users guide with version 1.13
  - using Sphinx-based rst → html approach
  - writing lightweight, example-driven articles per topic

This Effort:

- expanded users guide by another 8 articles:

Base Language:

- basic types
- literal values for basic types
- casts
- for loops
- zippered iteration

Task Parallelism:

- cobegins
- coforalls

Data Parallelism:

- forall loops

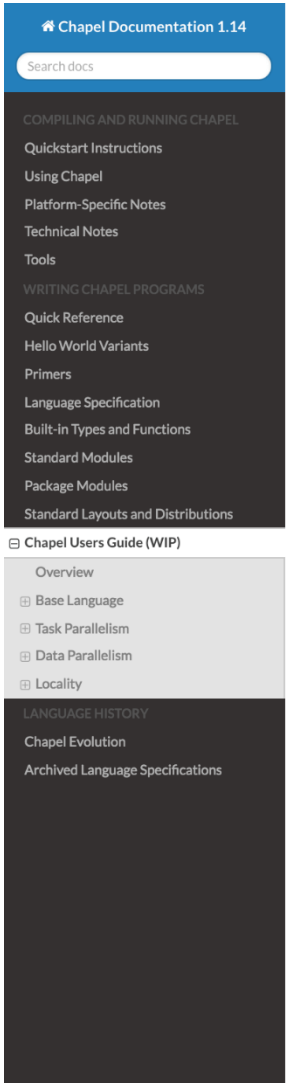
Users Guide Improvements

Impact:

- users guide starting to look non-trivial:

Next Steps:

- keep writing!
- consider using chpl2rst.py for these
  - current approach is fragile w.r.t. test changes



Base Language

This is the core of Chapel and what remains when all features in support of parallelism and locality are removed.

- Hello world: simple console output
- Variable Declarations
- Basic Types: booleans, numbers, and strings
- Literal Values for Basic Types
- Casts: explicit type conversions
- **for**-loops: structured serial iteration
- Zippered Iteration

(more to come...)

Task Parallelism

These are Chapel's lower-level features for creating parallel explicitly and synchronizing between them.

- Task Parallelism Overview
- **begin** Statements: unstructured tasking
- **cobegin** Statements: creating groups of tasks
- **coforall**-loops: loop-based tasking

(more to come...)

Data Parallelism

These are Chapel's higher-level features for creating parallelism more abstractly using a rich set of data structures.

- **forall**-loops: data-parallel loops

(more to come...)

Locality

These are Chapel's features for describing how data and tasks should be mapped to the target architecture for the purposes of performance and scalability.

- *Locales*: representing architectural locality
- Compiling and Executing Multi-Locale Programs
- The **locale** Type and Variables
- **on**-clauses: controlling locality/affinity

(more to come...)

Other Documentation Improvements

Other Documentation Improvements

- **Doc page content updates:**
  - Multilocale instructions
  - Quickstart instructions
  - UDP GASNet conduit notes
  - HDFS module (contributed by Deepak Majeti)
- **New primer: Modules**
- ***Archived Language Specifications* page created**
- ***chplvis* file format documented**
- **A multitude of spelling mistakes in source corrected**
- **Various general formatting improvements to online docs**
- **Several spec improvements**

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

CRAY

THE SUPERCOMPUTER COMPANY