# Documentation Improvements

Chapel Team, Cray Inc.

Chapel version 1.13

April 7, 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

- chpldoc Improvements
- Online Documentation Improvements
- Key Web Updates
- Chapel Users Guide
- Other Documentation Improvements

# chpldoc Improvements

# chpldoc: Traversing ‘use’ Statements

## Background:

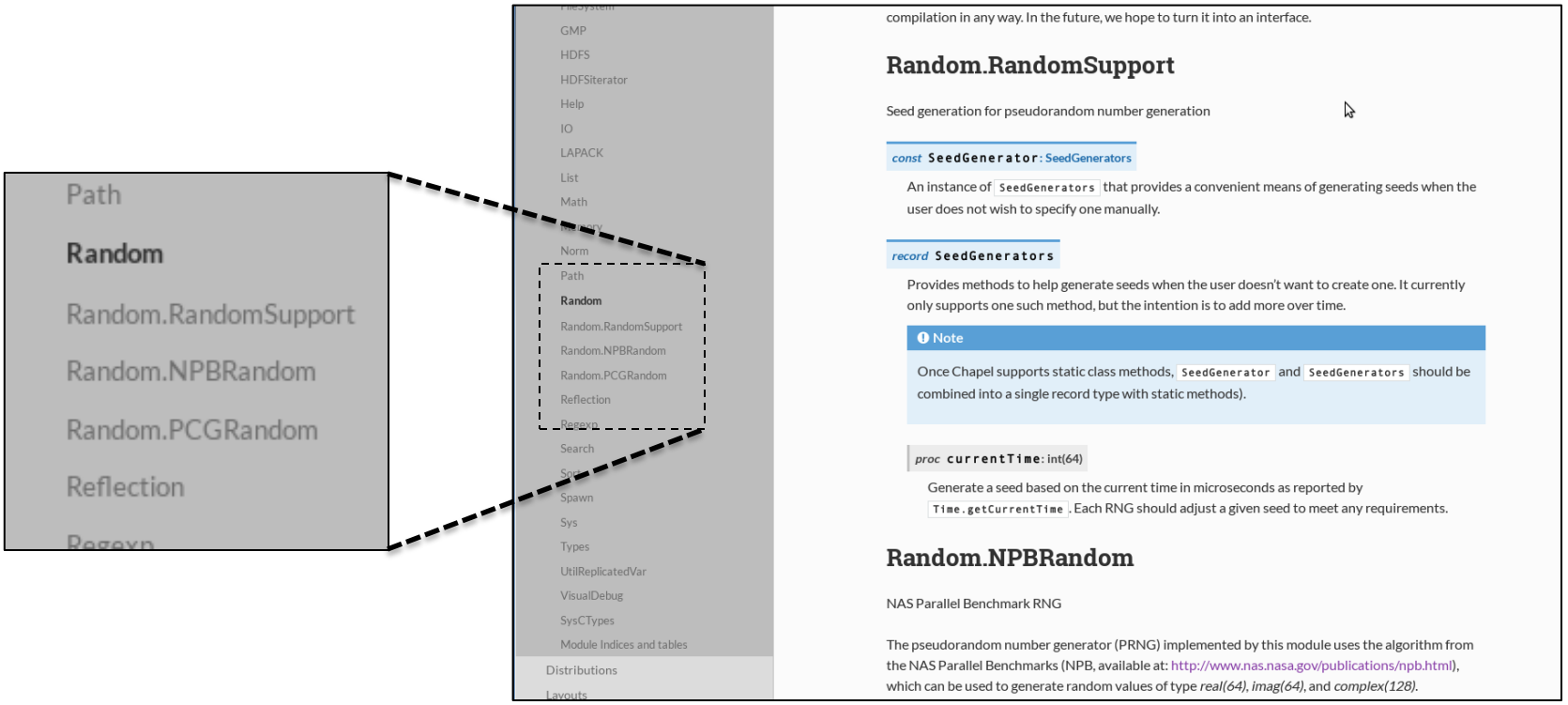
- Previously, chpldoc generated documentation for all ‘use’d modules
  - Arguably unintuitive as an end-user interface
  - Required all dependent modules to be available to generate docs
  - Led to errors when generating docs without a full Chapel installation

## This Effort:

- changed chpldoc to only document files listed on its command line
  - Improves documentation time for a single file slightly
    - (chpldoc is quite fast in either case)
  - New flag reverts to previous behavior: --process-used-modules

# chpldoc: Handling Submodules

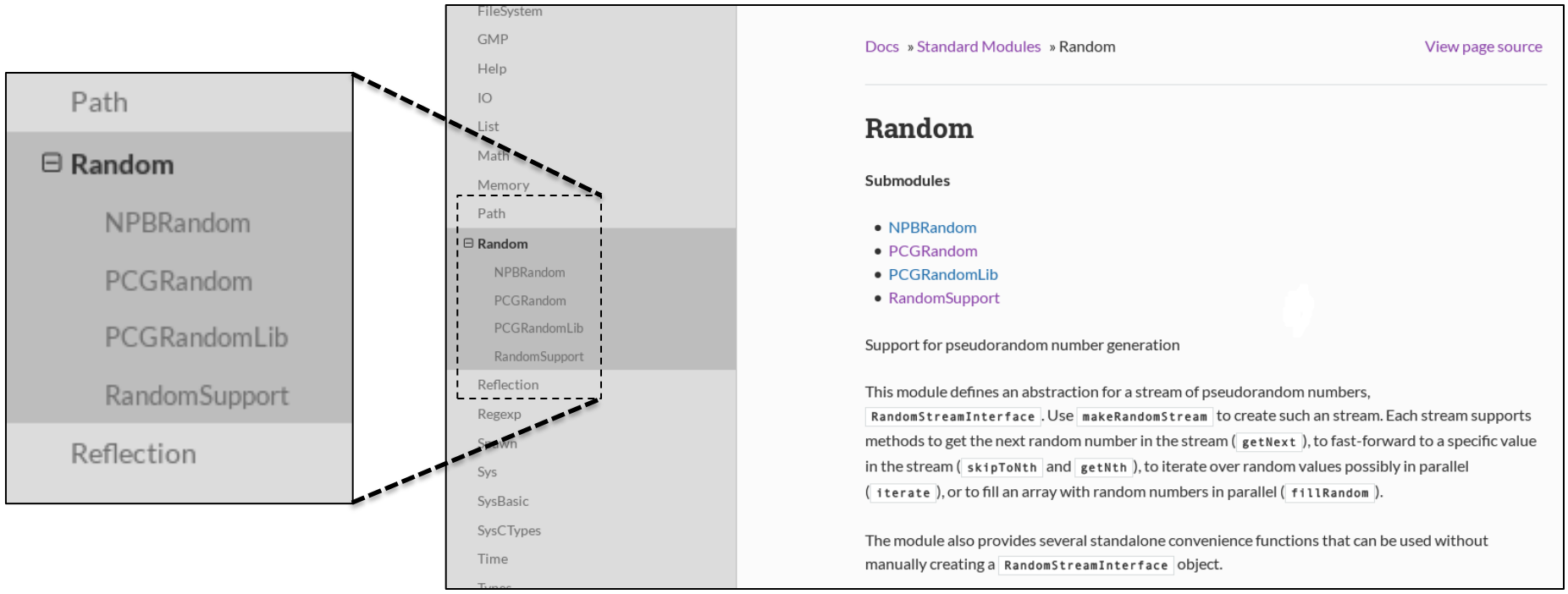
- **Background:** chpldoc generated 1 page per source file
  - So, sub-modules appeared on the same page as their parent module
    - Made documentation dense and difficult to read
    - rst-related bug mis-associated symbols following a sub-module with it



# chpldoc: Handling Submodules

## This Effort:

- Switched to creating 1 page per module
- Sidebar menu now reflects sub-module hierarchy
- Parent modules link to sub-modules at the top of their page
- Avoids incorrect sub-module association bug



# chpldoc: Building Online Docs

## Background: chpldoc built html by default

- Building online docs required repetitive steps
- Difficult to integrate non-chpldoc documentation with chpldoc output
  - e.g., HTML files created from .rst documentation files in the release

## This Effort: New chpldoc flag: --[no-]html

- Only builds intermediate (reStructureText) files
- Enables integrating chpldoc and non-chpldoc documentation

## Impact:

- Online documentation builds ~2x faster
- Users can integrate their chpldoc docs with external files

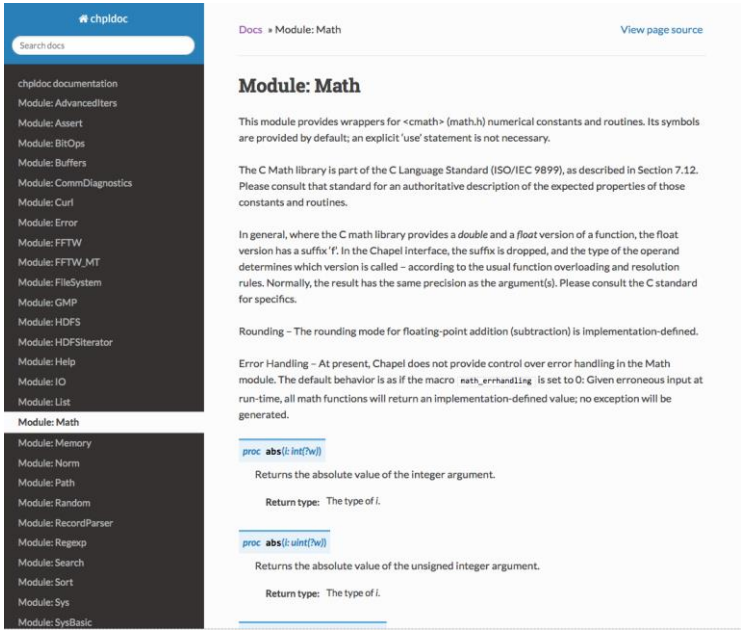
# chpldoc: Next Steps

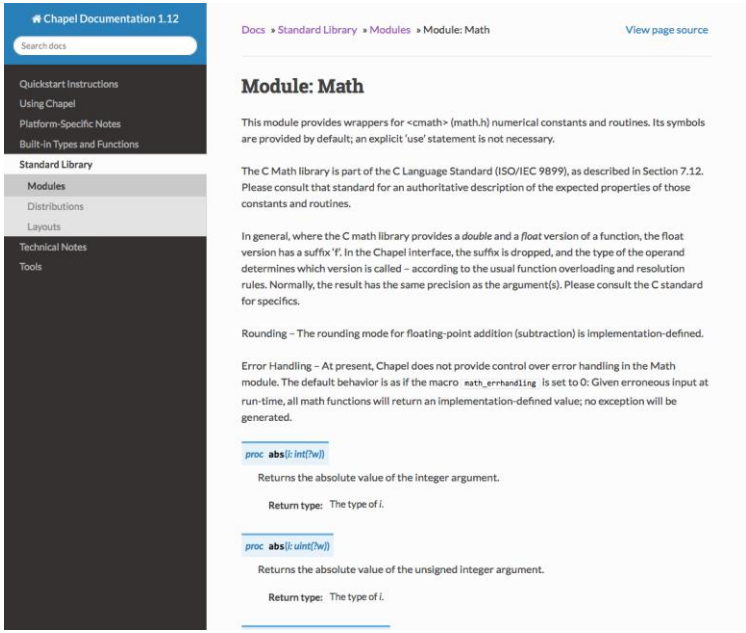
- **Output 'usage' information for each module**
  - (didn't make it into 1.13, but on master now)
- **Change testing of chpldoc to validate .rst output**
  - currently uses distinct text-oriented mode which could then be retired
- **Add class/record view, including inheritance**
  - Plus, class and record index
- **Support testing Chapel code used in chpldoc comments**
  - Similar to Python doctests
- **Link module documentation to source code**
- **Misc bug fixes**
  - Fix accidental output of code within nested block statements
  - Clean up HTML appearance of type and param methods

# Online Documentation Improvements

# Online Documentation: Background

- **v1.12 documentation improved significantly over v1.11**



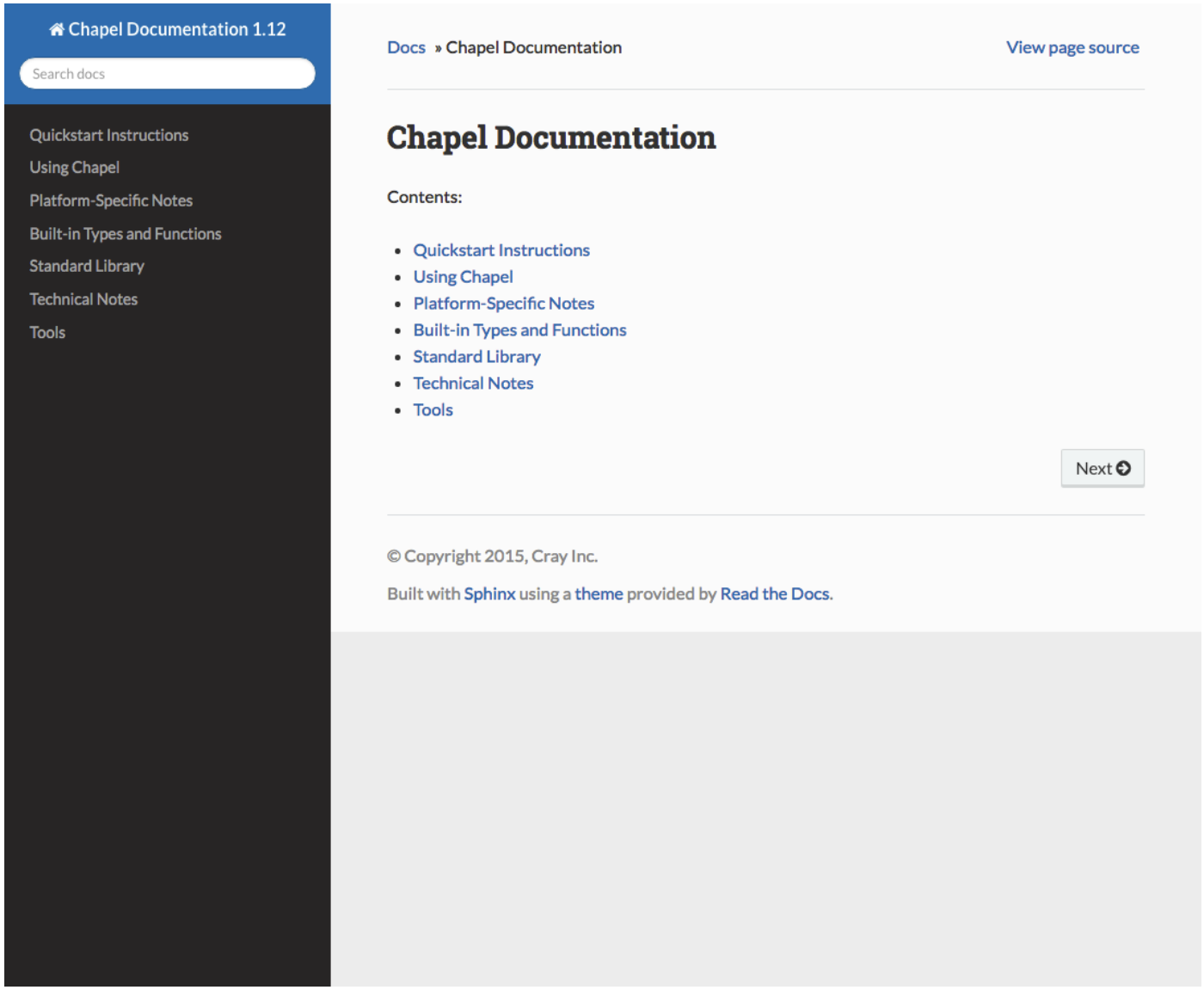




- **But there was still plenty of room for improvement:**
  - Some pages included undocumented items
  - Organization seemed poor at times
  - Some docs remained on chapel.cray.com rather than in docs/

# Online Documentation: This Effort

- **New content:**
  - 'chpl' & 'chpldoc' man pages
  - Moved documents from website to chapel.cray.com/docs hierarchy:
    - Quick Reference, Language Specification, and Language Evolution
  - Improved documentation coverage
  - Started a Chapel Users Guide
- **Improved organization**
  - Standard modules broken into more logical containers
    - Standard and Package modules
    - Standard Layouts and Distributions
  - Numerous other minor changes
- **Improved look-and-feel**
  - Upgraded Sphinx & Read the Docs versions
  - Added broken link detection for documentation via 'make check'
    - Using this, fixed all broken links (over 100)

# Index: 1.12



Chapel Documentation 1.12

Search docs

Quickstart Instructions
Using Chapel
Platform-Specific Notes
Built-in Types and Functions
Standard Library
Technical Notes
Tools

Docs » Chapel Documentation View page source

## Chapel Documentation

Contents:

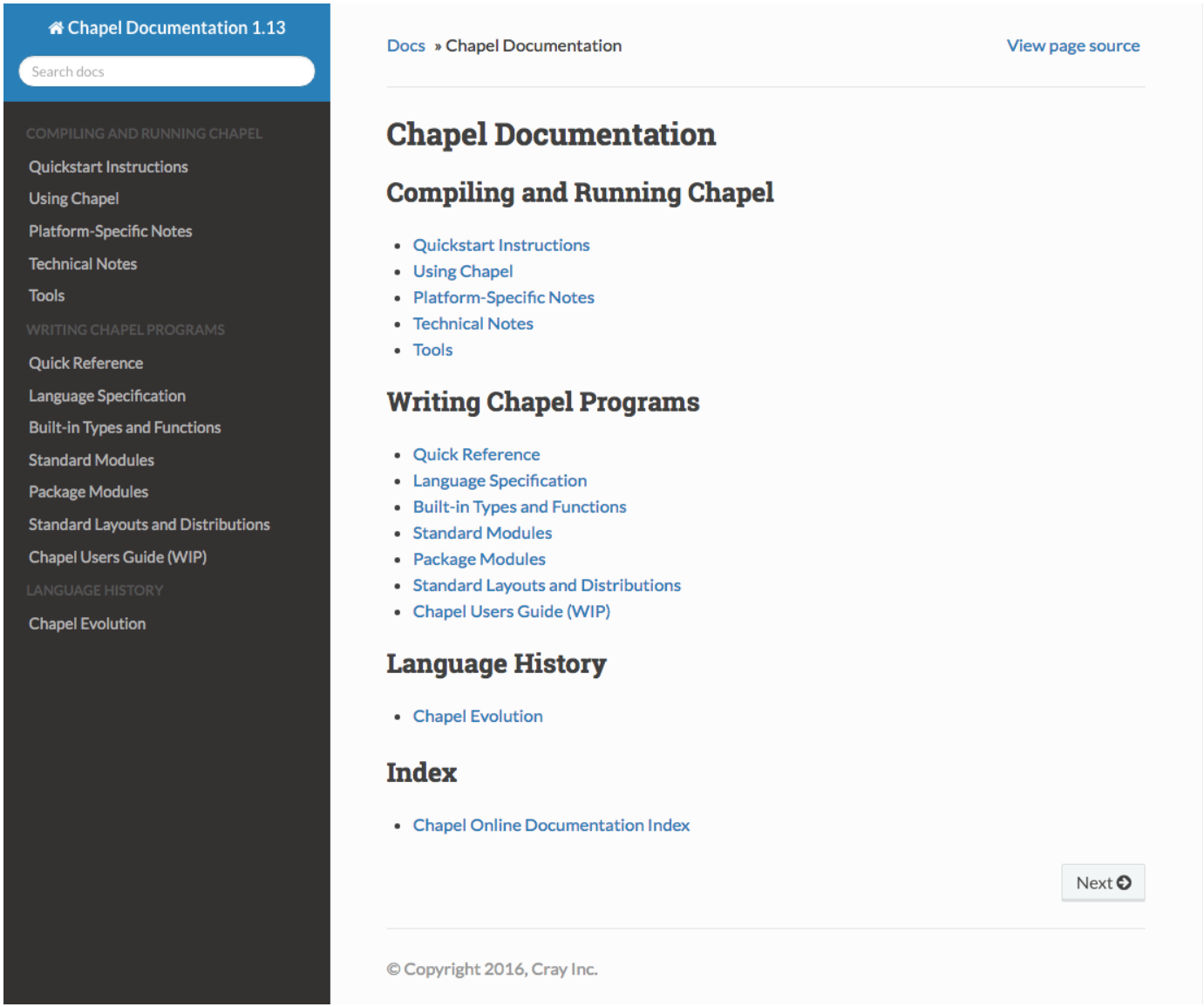
- Quickstart Instructions
- Using Chapel
- Platform-Specific Notes
- Built-in Types and Functions
- Standard Library
- Technical Notes
- Tools

Next



Built with Sphinx using a theme provided by Read the Docs.

# Index: 1.13



Chapel Documentation 1.13

Search docs

COMPILING AND RUNNING CHAPEL

Quickstart Instructions
Using Chapel
Platform-Specific Notes
Technical Notes
Tools

WRITING CHAPEL PROGRAMS

Quick Reference
Language Specification
Built-in Types and Functions
Standard Modules
Package Modules
Standard Layouts and Distributions
Chapel Users Guide (WIP)

LANGUAGE HISTORY

Chapel Evolution

Docs » Chapel Documentation View page source

## Chapel Documentation

### Compiling and Running Chapel

- Quickstart Instructions
- Using Chapel
- Platform-Specific Notes
- Technical Notes
- Tools

### Writing Chapel Programs

- Quick Reference
- Language Specification
- Built-in Types and Functions
- Standard Modules
- Package Modules
- Standard Layouts and Distributions
- Chapel Users Guide (WIP)

### Language History

- Chapel Evolution

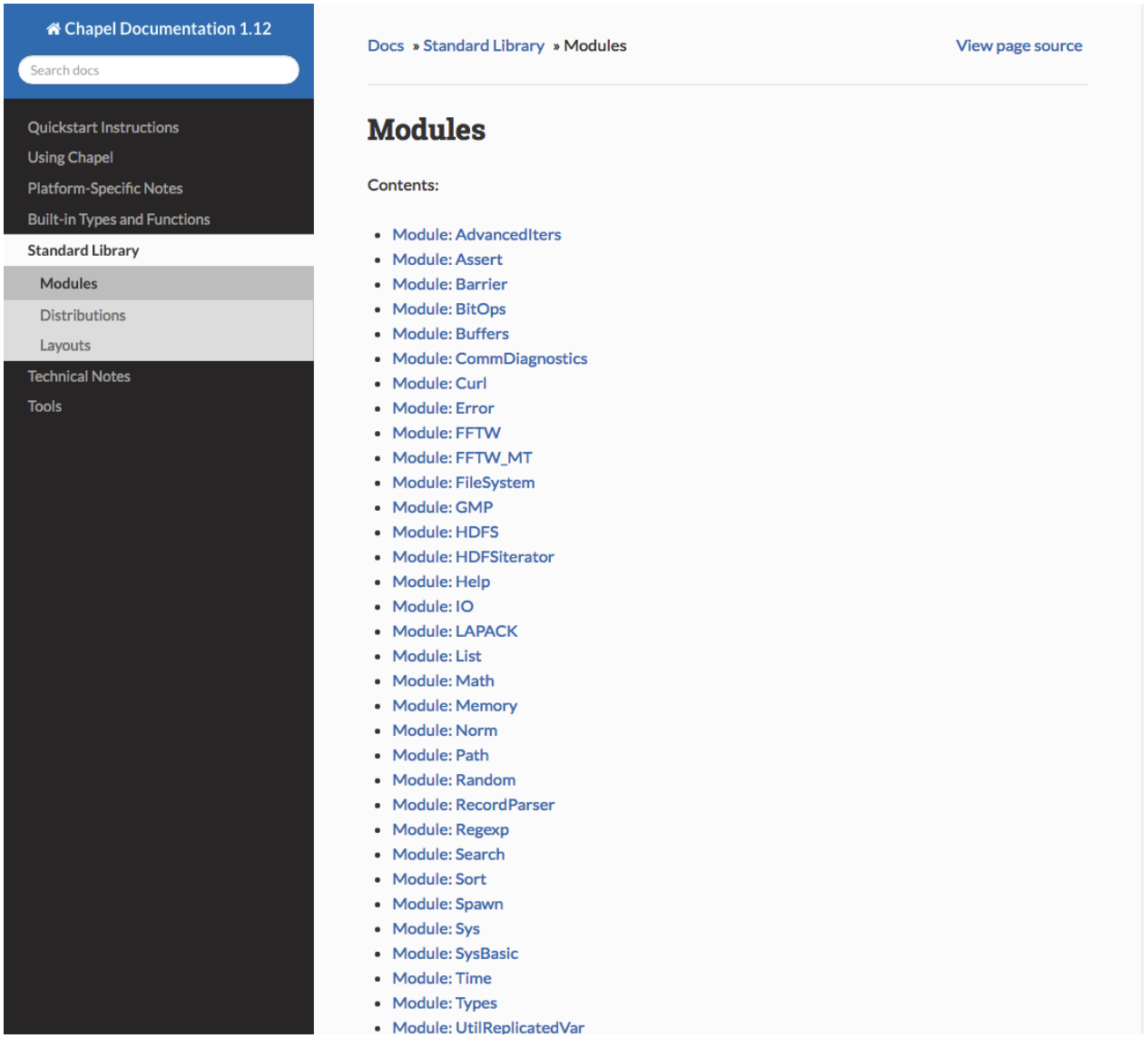
### Index

- Chapel Online Documentation Index

Next

© Copyright 2016, Cray Inc.

# Standard Modules: 1.12



Chapel Documentation 1.12

Search docs

- Quickstart Instructions
- Using Chapel
- Platform-Specific Notes
- Built-in Types and Functions
- Standard Library
  - Modules
  - Distributions
  - Layouts
- Technical Notes
- Tools

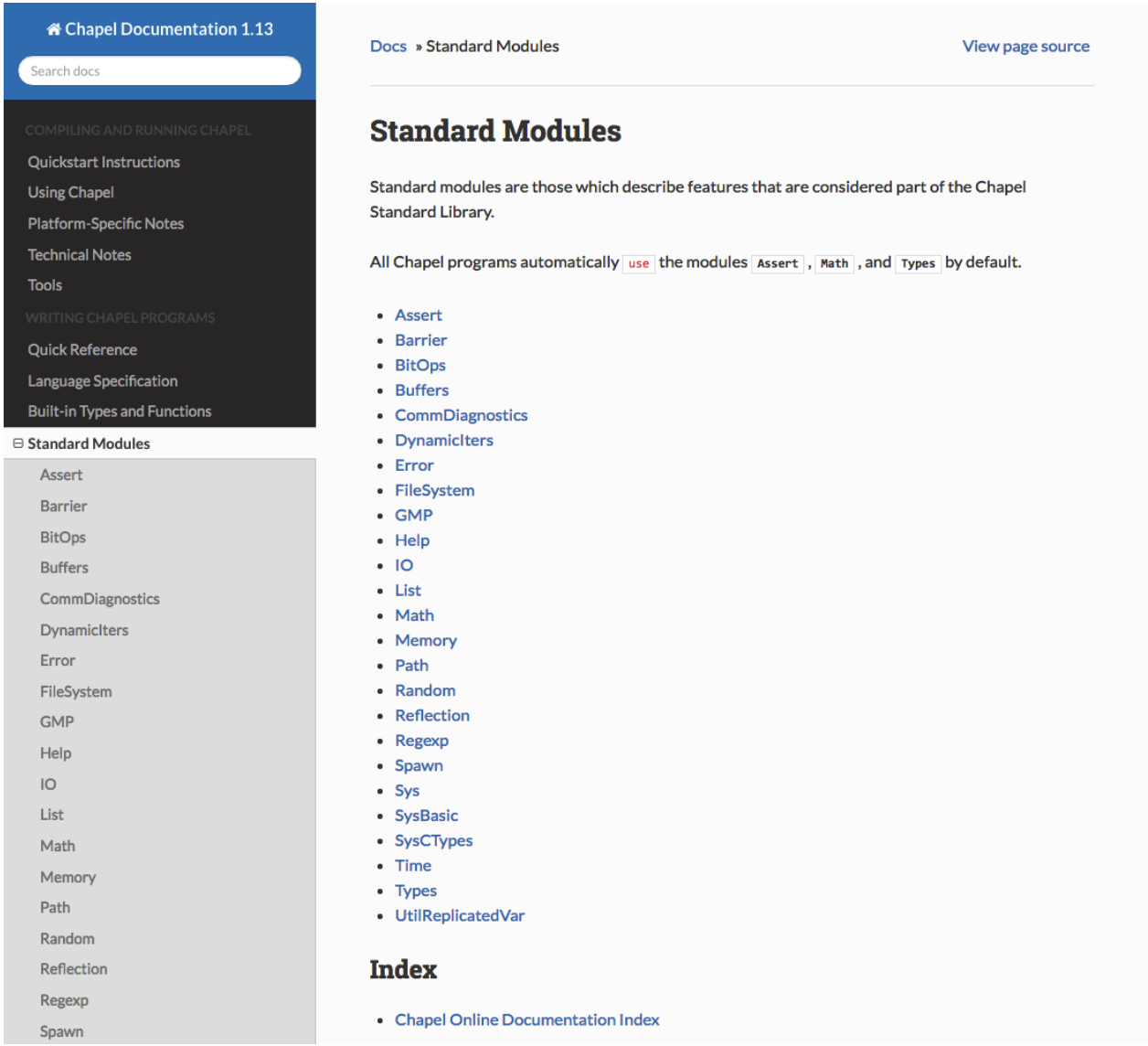
Docs » Standard Library » Modules — View page source

## Modules

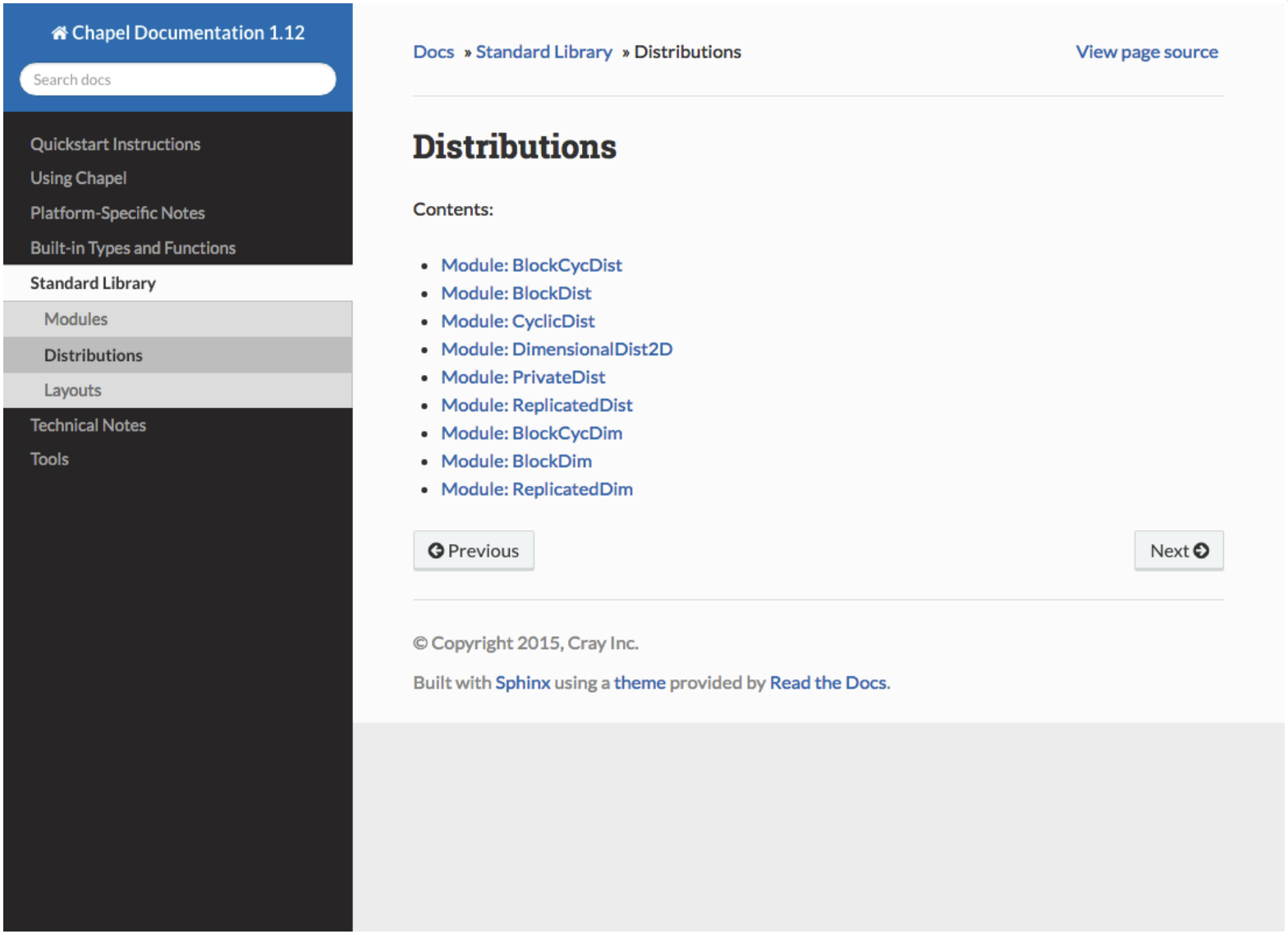
Contents:

- Module: AdvancedIters
- Module: Assert
- Module: Barrier
- Module: BitOps
- Module: Buffers
- Module: CommDiagnostics
- Module: Curl
- Module: Error
- Module: FFTW
- Module: FFTW_MT
- Module: FileSystem
- Module: GMP
- Module: HDFS
- Module: HDFSiterator
- Module: Help
- Module: IO
- Module: LAPACK
- Module: List
- Module: Math
- Module: Memory
- Module: Norm
- Module: Path
- Module: Random
- Module: RecordParser
- Module: Regexp
- Module: Search
- Module: Sort
- Module: Spawn
- Module: Sys
- Module: SysBasic
- Module: Time
- Module: Types
- Module: UtilReplicatedVar

# Standard Modules: 1.13



# Layouts & Distributions: 1.12



Chapel Documentation 1.12

Search docs

Quickstart Instructions
Using Chapel
Platform-Specific Notes
Built-in Types and Functions
Standard Library
Modules
Distributions
Layouts
Technical Notes
Tools

Docs » Standard Library » Distributions View page source

## Distributions

Contents:

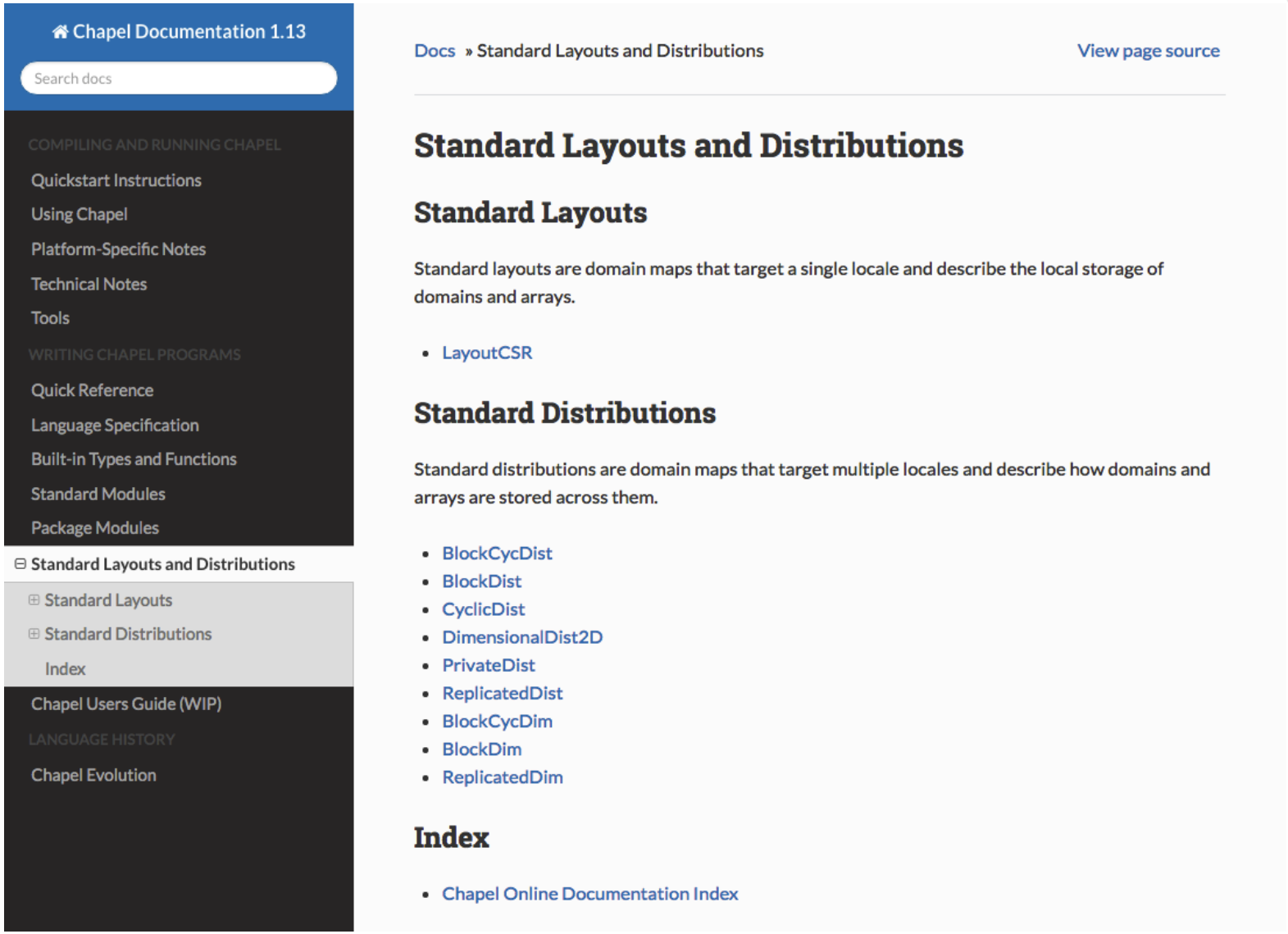
- Module: BlockCycDist
- Module: BlockDist
- Module: CyclicDist
- Module: DimensionalDist2D
- Module: PrivateDist
- Module: ReplicatedDist
- Module: BlockCycDim
- Module: BlockDim
- Module: ReplicatedDim

Previous Next

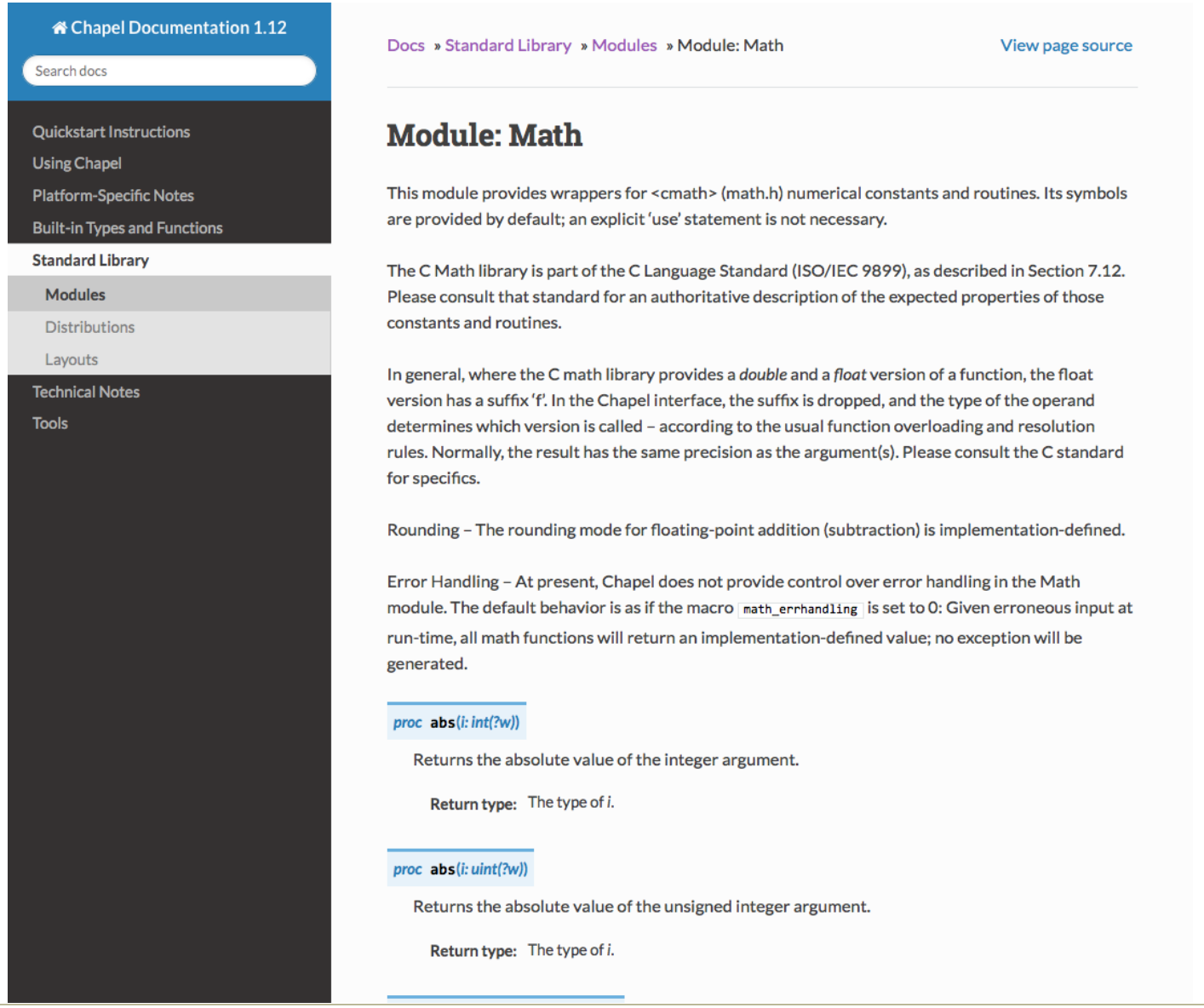
© Copyright 2015, Cray Inc.

Built with Sphinx using a theme provided by Read the Docs.

# Layouts & Distributions: 1.13



# Example Module: 1.12



Chapel Documentation 1.12

Search docs

- Quickstart Instructions
- Using Chapel
- Platform-Specific Notes
- Built-in Types and Functions
- Standard Library
  - Modules
  - Distributions
  - Layouts
- Technical Notes
- Tools

Docs » Standard Library » Modules » Module: Math View page source

## Module: Math

This module provides wrappers for <cmath> (math.h) numerical constants and routines. Its symbols are provided by default; an explicit 'use' statement is not necessary.

The C Math library is part of the C Language Standard (ISO/IEC 9899), as described in Section 7.12. Please consult that standard for an authoritative description of the expected properties of those constants and routines.

In general, where the C math library provides a *double* and a *float* version of a function, the float version has a suffix 'f'. In the Chapel interface, the suffix is dropped, and the type of the operand determines which version is called – according to the usual function overloading and resolution rules. Normally, the result has the same precision as the argument(s). Please consult the C standard for specifics.

Rounding – The rounding mode for floating-point addition (subtraction) is implementation-defined.

Error Handling – At present, Chapel does not provide control over error handling in the Math module. The default behavior is as if the macro `math_errhandling` is set to 0: Given erroneous input at run-time, all math functions will return an implementation-defined value; no exception will be generated.

*proc* **abs**(*i: int(?w)*)

Returns the absolute value of the integer argument.

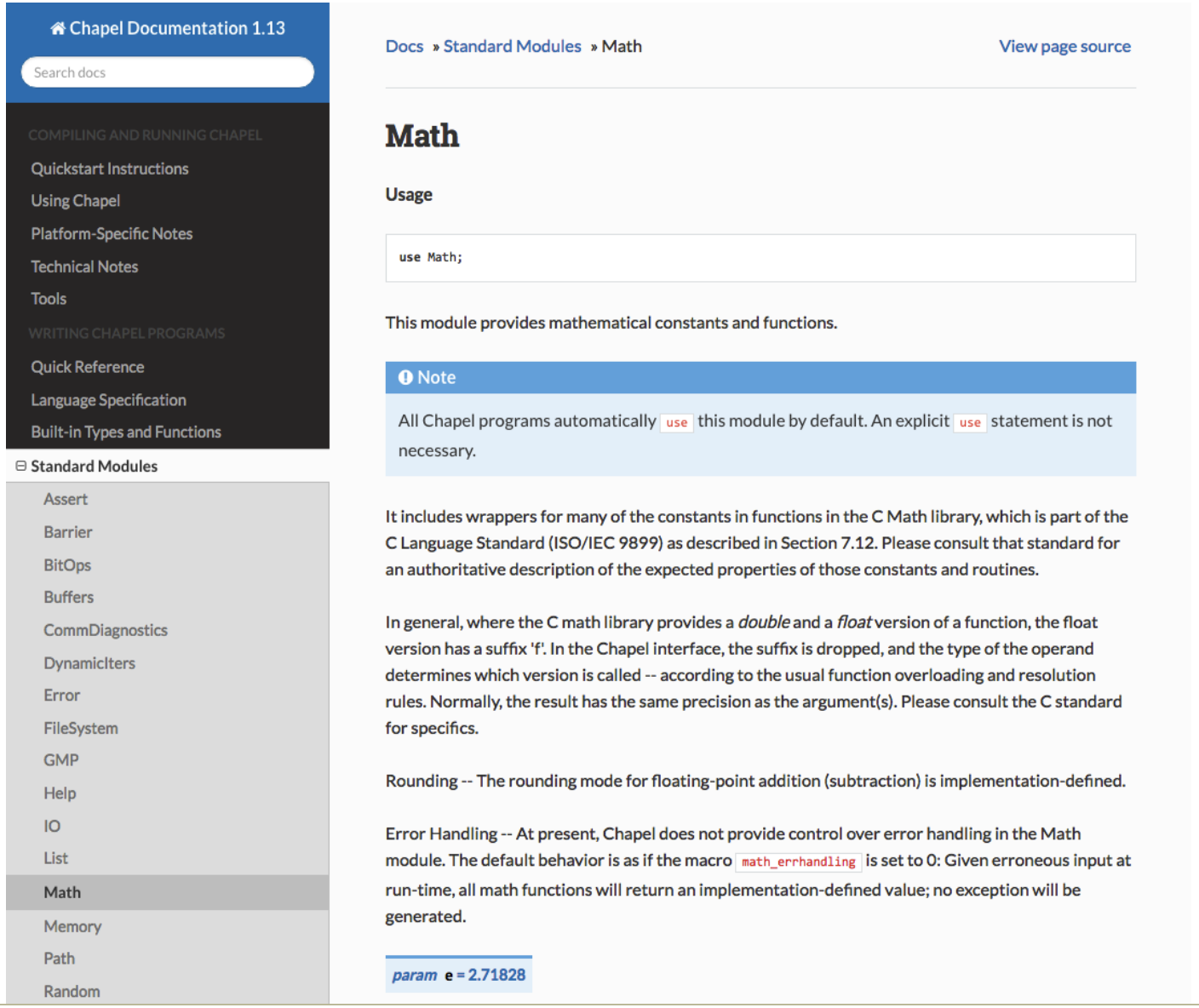
**Return type:** The type of *i*.

*proc* **abs**(*i: uint(?w)*)

Returns the absolute value of the unsigned integer argument.

**Return type:** The type of *i*.

# Example Module: 1.13



Chapel Documentation 1.13

Search docs

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

WRITING CHAPEL PROGRAMS
- Quick Reference
- Language Specification
- Built-in Types and Functions

Standard Modules
- Assert
- Barrier
- BitOps
- Buffers
- CommDiagnostics
- DynamicIters
- Error
- FileSystem
- GMP
- Help
- IO
- List
- Math
- Memory
- Path
- Random

Docs » Standard Modules » Math — View page source

## Math

### Usage

```
use Math;
```

This module provides mathematical constants and functions.

**Note**

All Chapel programs automatically `use` this module by default. An explicit `use` statement is not necessary.

It includes wrappers for many of the constants in functions in the C Math library, which is part of the C Language Standard (ISO/IEC 9899) as described in Section 7.12. Please consult that standard for an authoritative description of the expected properties of those constants and routines.

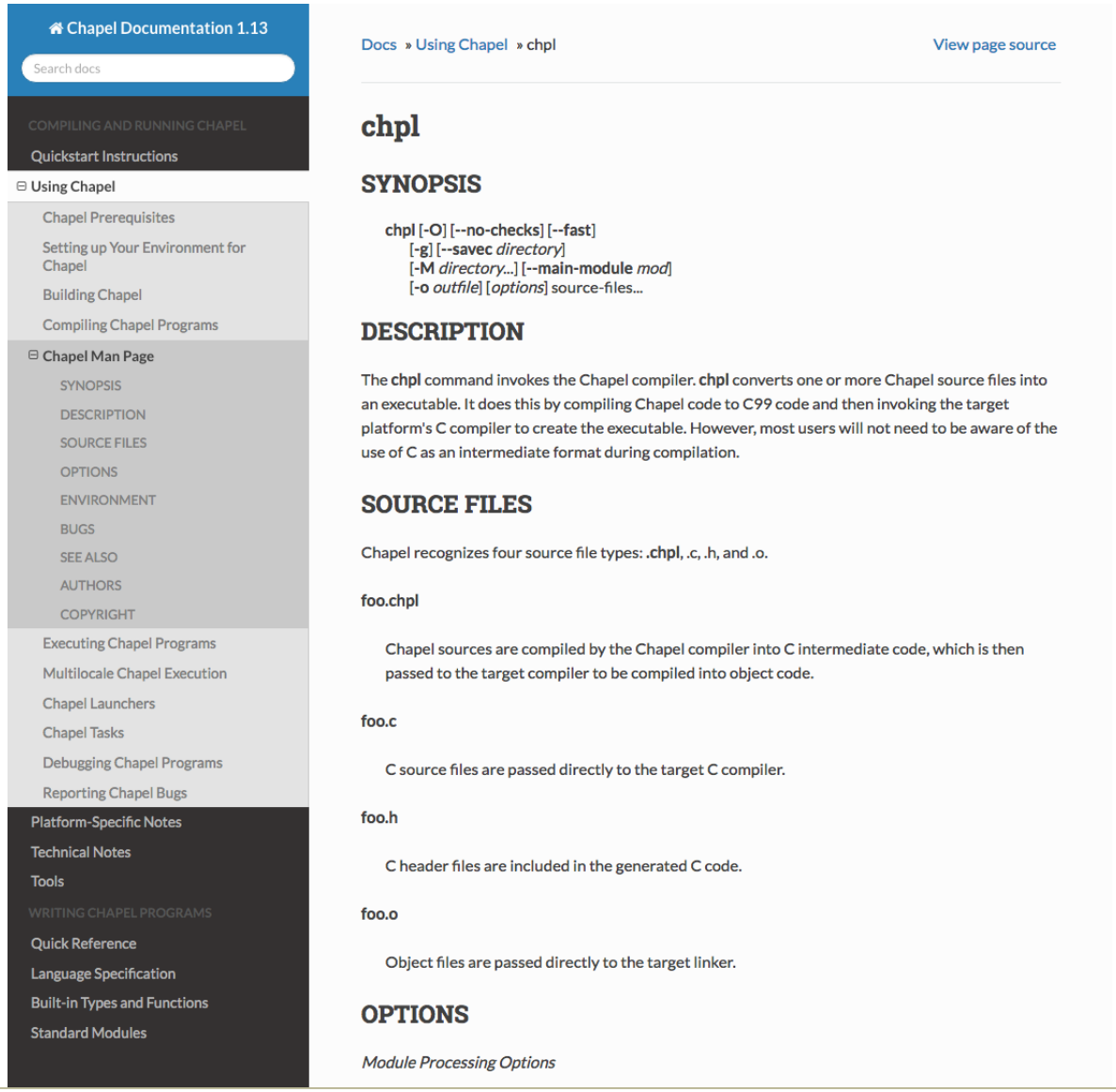
In general, where the C math library provides a *double* and a *float* version of a function, the float version has a suffix 'f'. In the Chapel interface, the suffix is dropped, and the type of the operand determines which version is called -- according to the usual function overloading and resolution rules. Normally, the result has the same precision as the argument(s). Please consult the C standard for specifics.

Rounding -- The rounding mode for floating-point addition (subtraction) is implementation-defined.

Error Handling -- At present, Chapel does not provide control over error handling in the Math module. The default behavior is as if the macro `math_errhandling` is set to 0: Given erroneous input at run-time, all math functions will return an implementation-defined value; no exception will be generated.

*param* **e = 2.71828**

# Online man pages: New in 1.13



Chapel Documentation 1.13

Search docs

COMPILING AND RUNNING CHAPEL
- Quickstart Instructions
- Using Chapel
  - Chapel Prerequisites
  - Setting up Your Environment for Chapel
  - Building Chapel
  - Compiling Chapel Programs
  - Chapel Man Page
    - SYNOPSIS
    - DESCRIPTION
    - SOURCE FILES
    - OPTIONS
    - ENVIRONMENT
    - BUGS
    - SEE ALSO
    - AUTHORS
    - COPYRIGHT
  - Executing Chapel Programs
  - Multilocale Chapel Execution
  - Chapel Launchers
  - Chapel Tasks
  - Debugging Chapel Programs
  - Reporting Chapel Bugs
- Platform-Specific Notes
- Technical Notes
- Tools

WRITING CHAPEL PROGRAMS
- Quick Reference
- Language Specification
- Built-in Types and Functions
- Standard Modules

Docs » Using Chapel » chpl — View page source

## chpl

### SYNOPSIS

**chpl** [**-O**] [**--no-checks**] [**--fast**]
[**-g**] [**--savec** *directory*]
[**-M** *directory...*] [**--main-module** *mod*]
[**-o** *outfile*] [*options*] source-files...

### DESCRIPTION

The **chpl** command invokes the Chapel compiler. **chpl** converts one or more Chapel source files into an executable. It does this by compiling Chapel code to C99 code and then invoking the target platform's C compiler to create the executable. However, most users will not need to be aware of the use of C as an intermediate format during compilation.

### SOURCE FILES

Chapel recognizes four source file types: **.chpl**, .c, .h, and .o.

**foo.chpl**

Chapel sources are compiled by the Chapel compiler into C intermediate code, which is then passed to the target compiler to be compiled into object code.

**foo.c**

C source files are passed directly to the target C compiler.

**foo.h**

C header files are included in the generated C code.

**foo.o**

Object files are passed directly to the target linker.

### OPTIONS

*Module Processing Options*

# Online Documentation: Next Steps

- **Further improvements to look-and-feel / user interface**
  - Fork ‘read-the-docs’ and fine-tune sphinx template to our needs
    - e.g. drop-down menu for versions
- **Add developer section and/or version**
  - Move developer documentation from repository to online docs
  - Document internal methods not intended for users
- **Increase content coverage**
  - Continue adding online documentation for “built-in” features
  - Improve quality of documentation for existing modules

# Key Web Updates

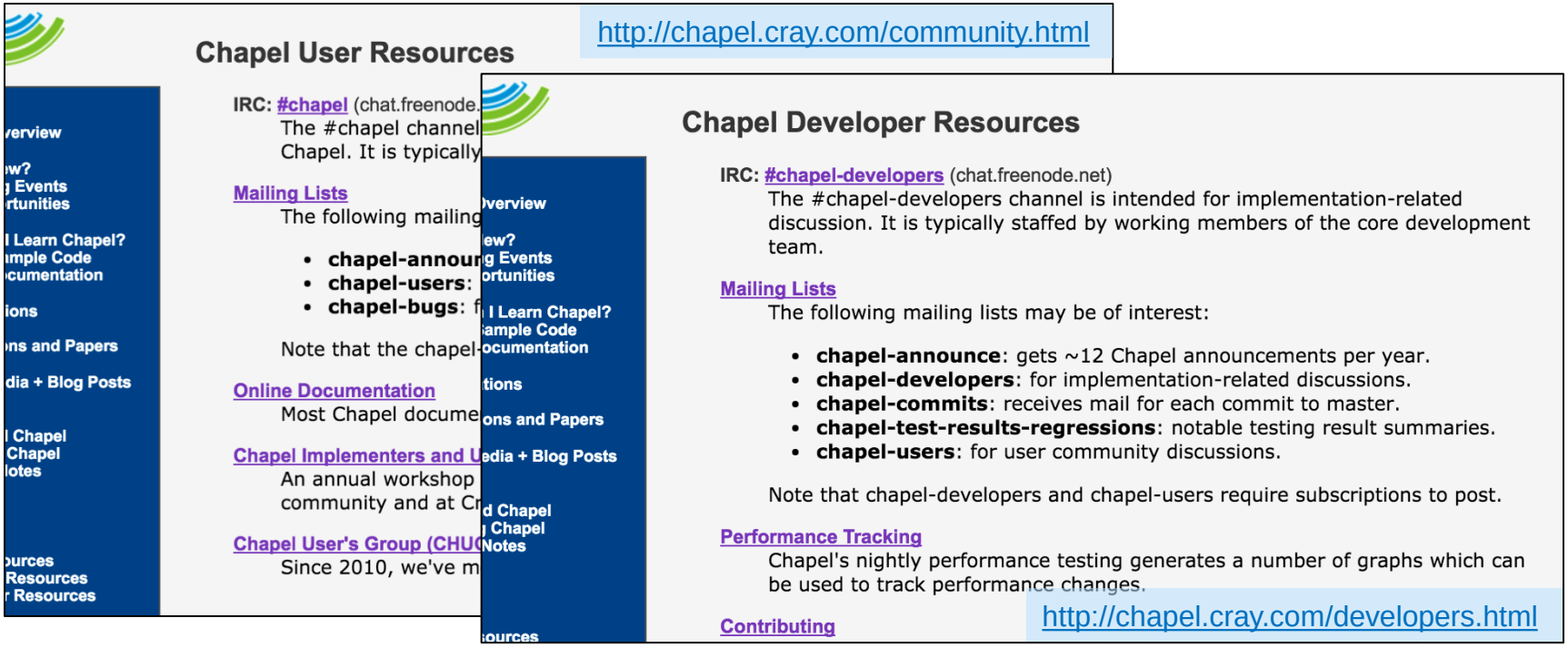
# Web Updates: User / Developer Resources

## Background:

- These pages have existed for awhile, but were wordy and stale

## This Effort:

- Refresh content and make it more concise and approachable



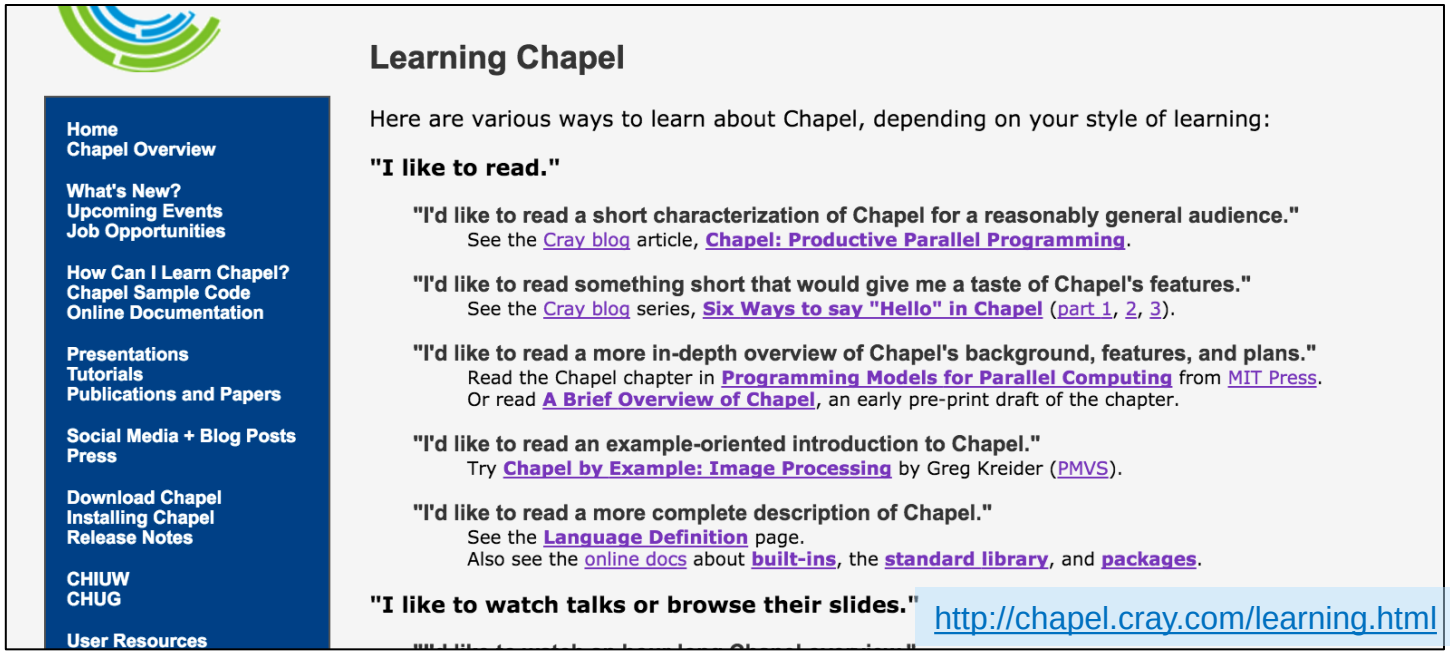
# Web Updates: How Can I Learn Chapel?

## Background:

- This has been increasingly FAQ'd, esp. with blog posts, GSoC, etc.

## This Effort:

- Gathered list of good resources, sorted by learning style
  - e.g., "I like to read" vs. "I'd like to watch talks" vs. "I like to read code"



Home
Chapel Overview

What's New?
Upcoming Events
Job Opportunities

How Can I Learn Chapel?
Chapel Sample Code
Online Documentation

Presentations
Tutorials
Publications and Papers

Social Media + Blog Posts
Press

Download Chapel
Installing Chapel
Release Notes

CHIUW
CHUG

User Resources

**Learning Chapel**

Here are various ways to learn about Chapel, depending on your style of learning:

**"I like to read."**

**"I'd like to read a short characterization of Chapel for a reasonably general audience."**
See the Cray blog article, **Chapel: Productive Parallel Programming**.

**"I'd like to read something short that would give me a taste of Chapel's features."**
See the Cray blog series, **Six Ways to say "Hello" in Chapel** (part 1, 2, 3).

**"I'd like to read a more in-depth overview of Chapel's background, features, and plans."**
Read the Chapel chapter in **Programming Models for Parallel Computing** from MIT Press.
Or read **A Brief Overview of Chapel**, an early pre-print draft of the chapter.

**"I'd like to read an example-oriented introduction to Chapel."**
Try **Chapel by Example: Image Processing** by Greg Kreider (PMVS).

**"I'd like to read a more complete description of Chapel."**
See the **Language Definition** page.
Also see the online docs about **built-ins**, the **standard library**, and **packages**.

**"I like to watch talks or browse their slides."**

http://chapel.cray.com/learning.html

# Chapel Users Guide

# Users Guide: Background and This Effort

## Background:

- Have intended to write a users guide for some time now…
  - goal: more readable, less formal intro to Chapel than the language spec

…yet, it's such a major undertaking that it's been hard to get started

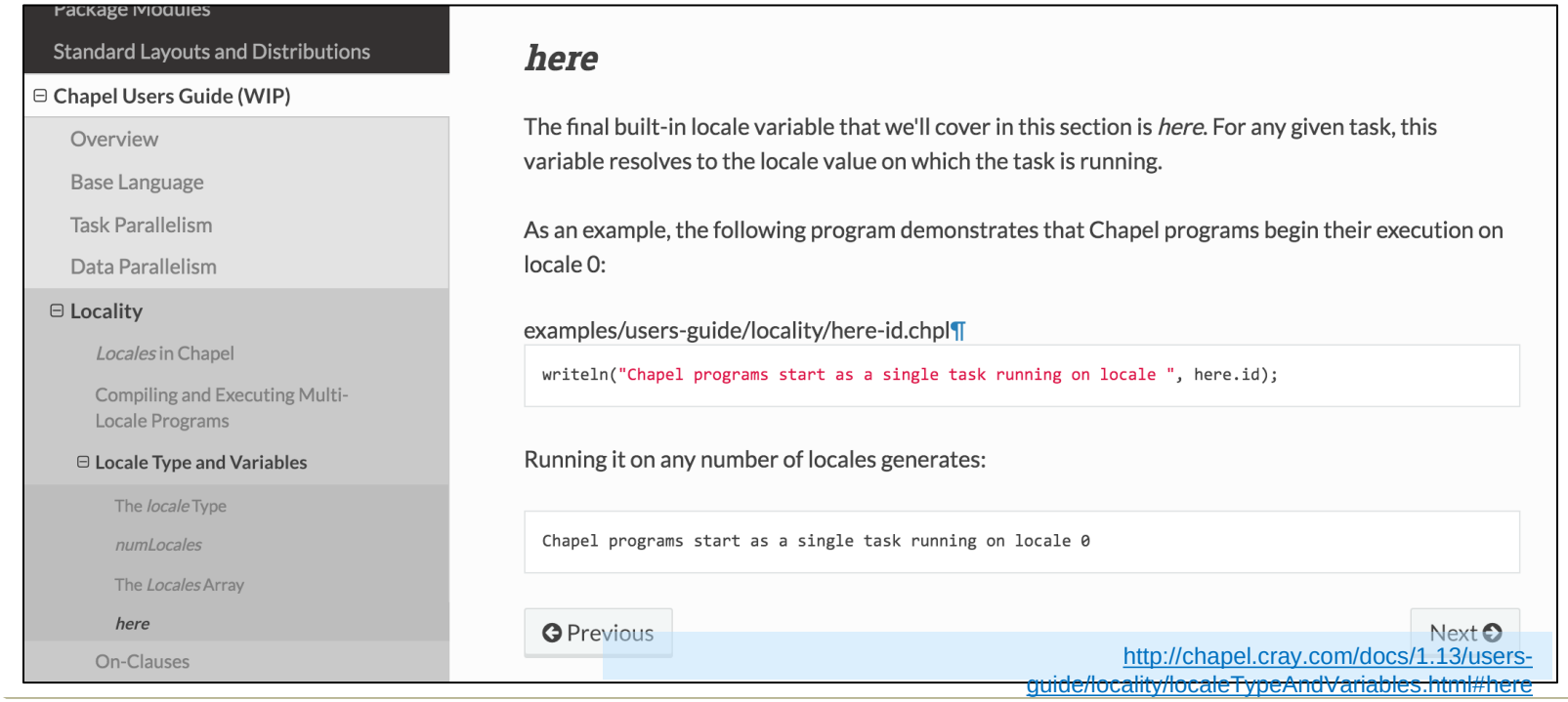
## This Effort:

- Motivated by user comments at CHIUW 2015
  - enthusiasm for first round of online documentation in v1.11
  - desire for similar online documentation covering Chapel language itself
- Last year's chpldoc / Sphinx effort made it easier to get started
  - Sphinx trivially supports...
    - ease of writing via .rst files
    - online deployment, searching
    - ability to trivially include testable code examples
  - Also supports writing "a page per topic" rather than "a whole new book"
  - To that end, started writing the users guide this release cycle

# Users Guide: Status

## Status:

- Taking a breadth-first approach
  - goal: cover key features earlier, yet without ignoring the base language
- Wrote 8 entries so far, available in online docs
  - response to existing docs has been positive

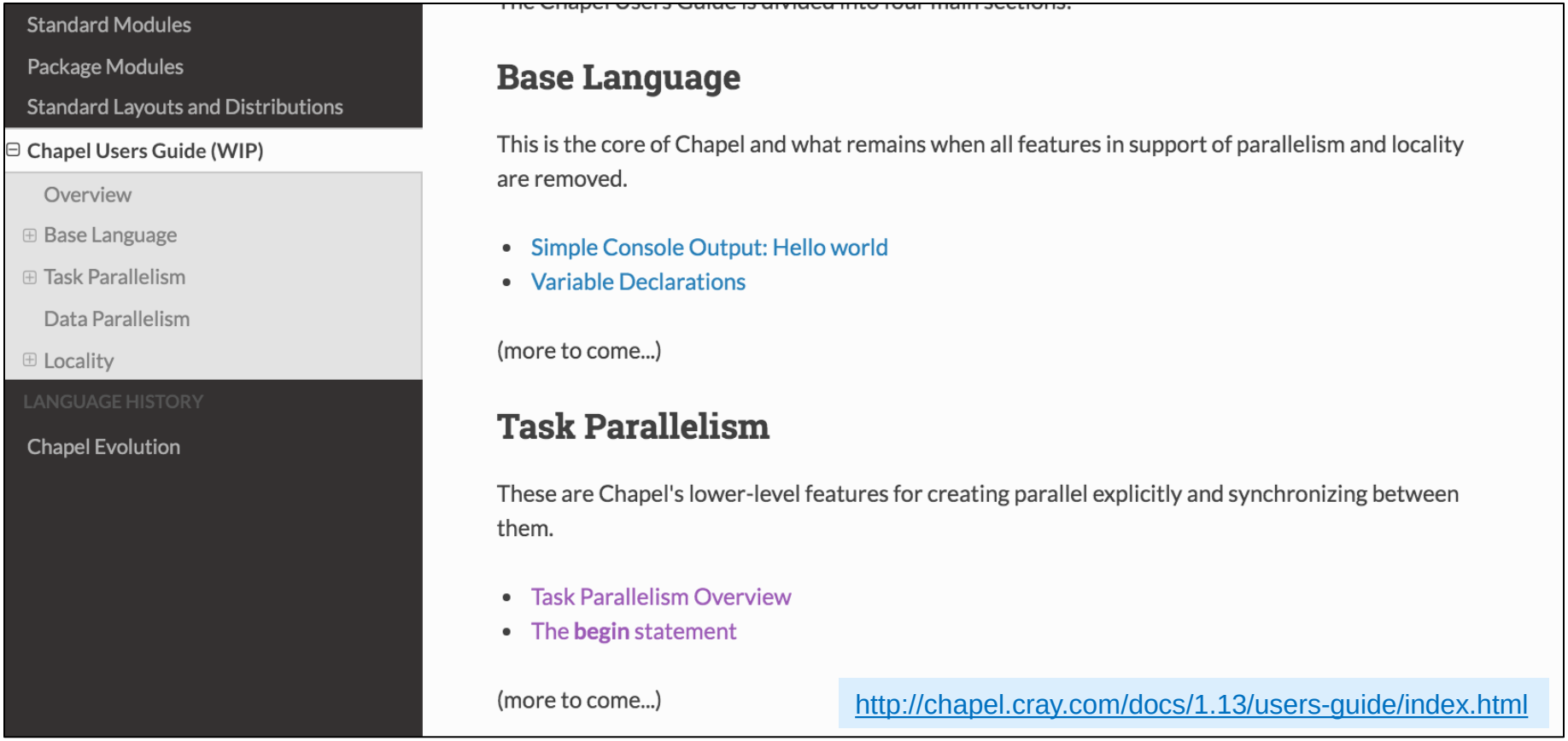



http://chapel.cray.com/docs/1.13/users-guide/locality/localeTypeAndVariables.html#here

# Users Guide: Next Steps

## Next Steps:

- Continue writing, strive to write a few entries per week



# Other Documentation Improvements

# Other Documentation Improvements

- **Split "quick start" information out of top-level README**
  - README.rst
  - QUICKSTART.rst
- **Converted more README-style files to markdown:**
  - ACKNOWLEDGEMENTS.md
  - CHANGES.md
  - CONTRIBUTORS.md
  - PERFORMANCE.md
- **Example test updates**
  - Added ISx and LCALS, improved MiniMD (see benchmarks slides)
  - Fixed typos and improved explanations in primer examples
  - Extended the linkedList example to support additional methods (contributed by Akash Thorat)
  - Updated example tests to reflect language and library changes
  - Retired README.features

# 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