

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



COMPUTE

|

STORE

|

ANALYZE

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

Path

Random

- Random.RandomSupport
- Random.NPBRandom
- Random.PCGRandom
- Reflection
- Spawn

- GMP
- HDFS
- HDFSSIterator
- Help
- IO
- LAPACK
- List
- Math
- Norm
- Path
- Random
- Random.RandomSupport
- Random.NPBRandom
- Random.PCGRandom
- Reflection
- Spawn
- Search
- Sys
- Types
- UtilReplicatedVar
- VisualDebug
- SysCTypes
- Module Indices and tables
- Distributions
- Layouts

compilation in any way. In the future, we hope to turn it into an interface.

Random.RandomSupport

Seed generation for pseudorandom number generation

const SeedGenerator: SeedGenerators

An instance of `SeedGenerators` that provides a convenient means of generating seeds when the user does not wish to specify one manually.

record SeedGenerators

Provides methods to help generate seeds when the user doesn't want to create one. It currently only supports one such method, but the intention is to add more over time.

Note

Once Chapel supports static class methods, `SeedGenerator` and `SeedGenerators` should be combined into a single record type with static methods).

proc currentTime: int(64)

Generate a seed based on the current time in microseconds as reported by `Time.getCurrentTime`. Each RNG should adjust a given seed to meet any requirements.

Random.NPBRandom

NAS Parallel Benchmark RNG

The pseudorandom number generator (PRNG) implemented by this module uses the algorithm from the NAS Parallel Benchmarks (NPB, available at: <http://www.nas.nasa.gov/publications/npb.html>), which can be used to generate random values of type `real(64)`, `imag(64)`, and `complex(128)`.

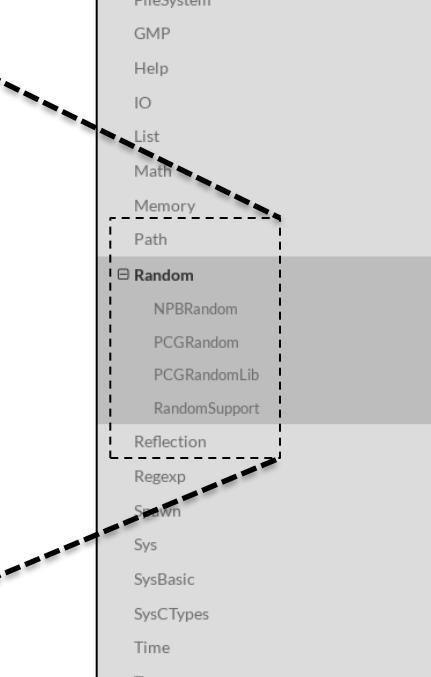
chpldoc: Handling Submodules

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

Path

- ⊖ Random
 - NPBRandom
 - PCGRandom
 - PCGRandomLib
 - RandomSupport
- Reflection



Docs » Standard Modules » Random View page source

Random

Submodules

- [NPBRandom](#)
- [PCGRandom](#)
- [PCGRandomLib](#)
- [RandomSupport](#)

Support for pseudorandom number generation

This module defines an abstraction for a stream of pseudorandom numbers, `RandomStreamInterface`. Use `makeRandomStream` to create such a stream. Each stream supports methods to get the next random number in the stream (`getNext`), to fast-forward to a specific value in the stream (`skipToNth` and `getNth`), to iterate over random values possibly in parallel (`iterate`), or to fill an array with random numbers in parallel (`fillRandom`).

The module also provides several standalone convenience functions that can be used without manually creating a `RandomStreamInterface` object.

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 (reStructuredText) 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



COMPUTE

|

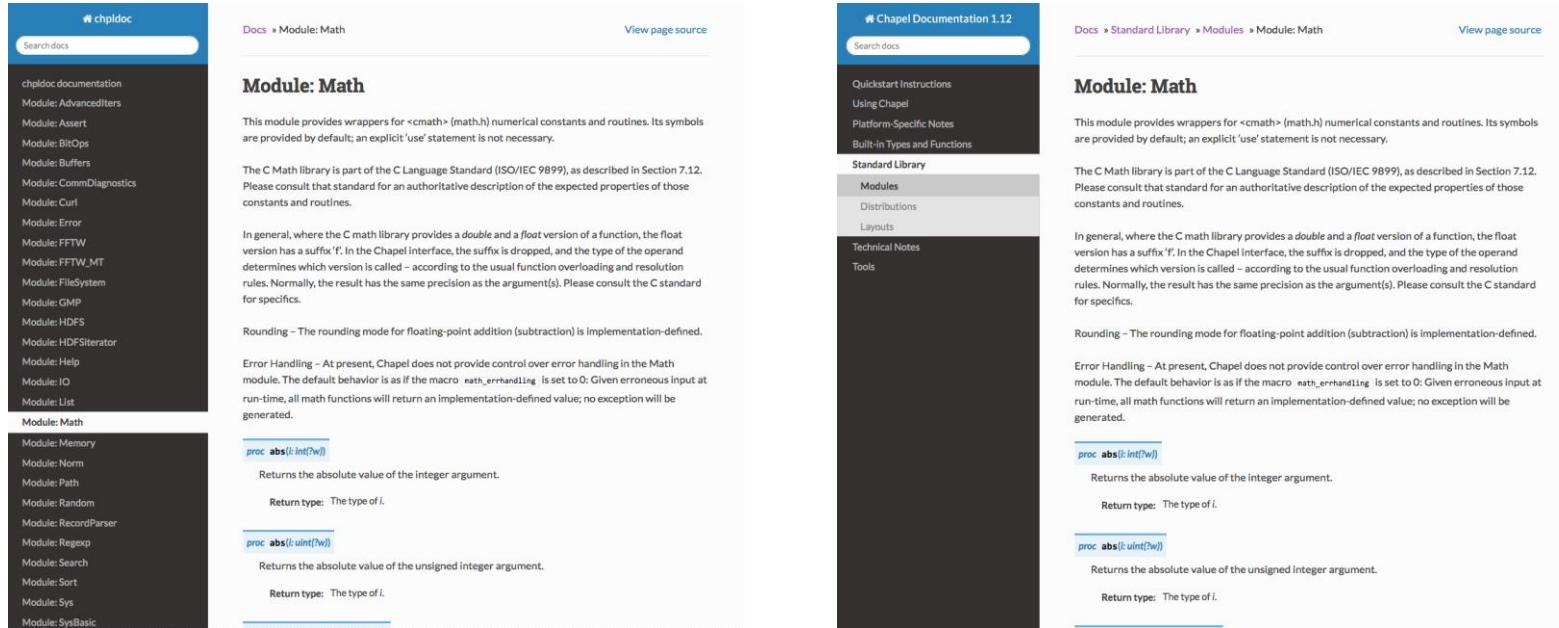
STORE

|

ANALYZE

Online Documentation: Background

- v1.12 documentation improved significantly over v1.11



The image shows two side-by-side screenshots of the Chapel documentation website. The left screenshot is for v1.11 and the right is for v1.12. Both pages are for the 'Module: Math' documentation.

v1.11 Documentation (Left):

- Header:** Shows 'Docs > Module: Math' and a 'View page source' link.
- Left Sidebar:** A dark sidebar with a search bar at the top, followed by a list of modules including 'chpldoc', 'Module: Math', 'Module: Memory', etc.
- Content Area:**
 - Module: Math** section: Describes the module as providing wrappers for <cmath> (math.h) numerical constants and routines. It notes that symbols are provided by default and that an explicit 'use' statement is not necessary.
 - Implementation Notes:** Includes sections on 'Rounding' and 'Error Handling'.
 - Code Examples:** Shows two examples of the `abs` function: one for integers and one for unsigned integers.

v1.12 Documentation (Right):

- Header:** Shows 'Docs > Standard Library > Modules > Module: Math' and a 'View page source' link.
- Left Sidebar:** A dark sidebar with a search bar at the top, followed by a navigation menu with items like 'Quickstart Instructions', 'Using Chapel', 'Platform-Specific Notes', 'Built-in Types and Functions', 'Standard Library', 'Modules' (which is highlighted), 'Distributions', 'Layouts', 'Technical Notes', and 'Tools'.
- Content Area:**
 - Module: Math** section: Describes the module as providing wrappers for <cmath> (math.h) numerical constants and routines. It notes that symbols are provided by default and that an explicit 'use' statement is not necessary.
 - Implementation Notes:** Includes sections on 'Rounding' and 'Error Handling'.
 - Code Examples:** Shows two examples of the `abs` function: one for integers and one for unsigned integers.

- 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/chapel-lang.org](http://docs.chapel-lang.org)



COMPUTE

STORE

ANALYZE

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

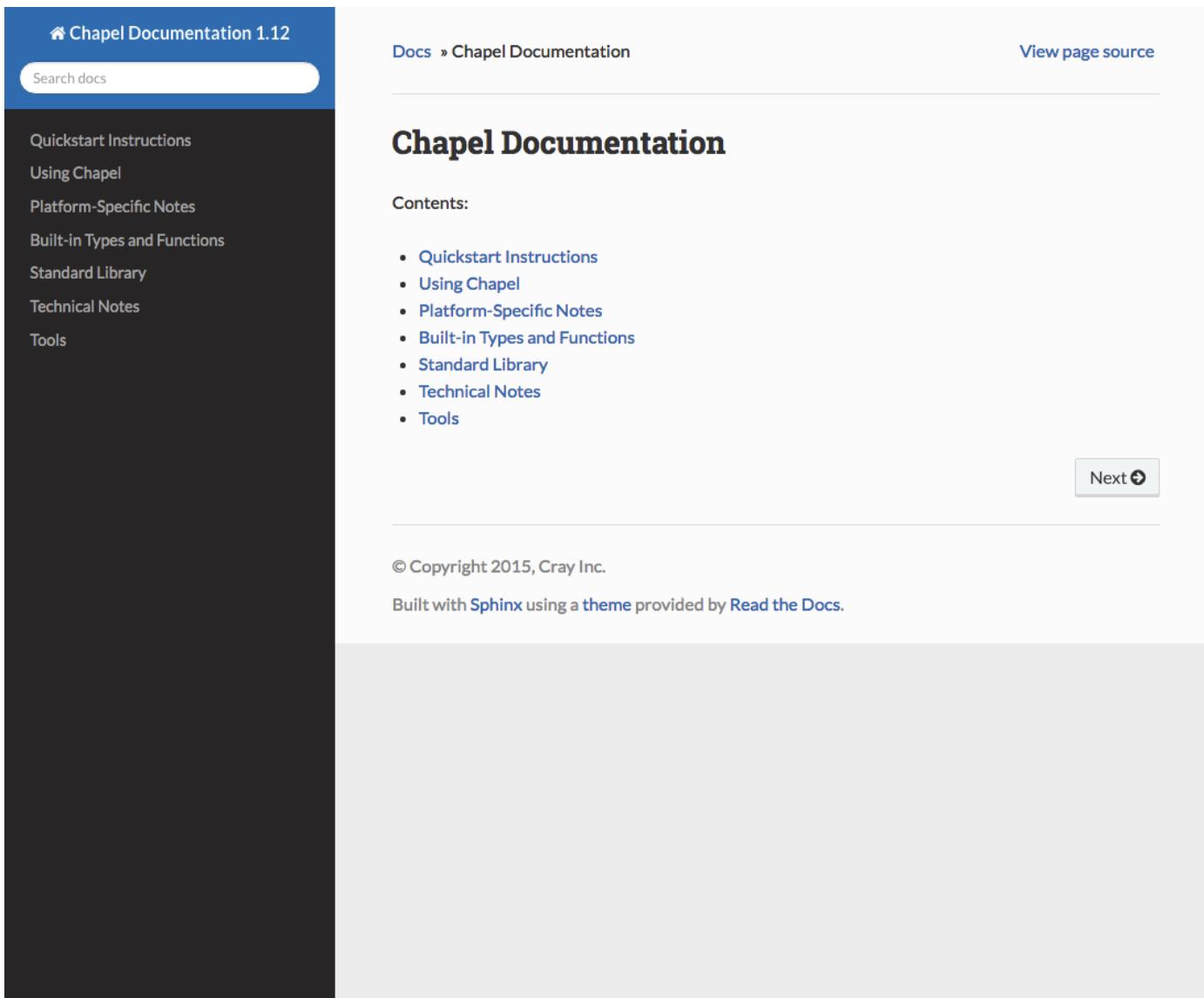
- **Improved organization**

- 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

A screenshot of the Chapel Documentation 1.12 website. The left sidebar has a dark blue header with the text "Chapel Documentation 1.12" and a search bar labeled "Search docs". Below the header is a list of navigation links: "Quickstart Instructions", "Using Chapel", "Platform-Specific Notes", "Built-in Types and Functions", "Standard Library", "Technical Notes", and "Tools". The main content area has a light gray header with the text "Docs » Chapel Documentation" and a "View page source" link. Below the header is the title "Chapel Documentation" in a large, bold, dark font. Underneath the title is the heading "Contents:" followed by a bulleted list of the same navigation links as the sidebar. At the bottom of the page is a footer with the text "© Copyright 2015, Cray Inc." and "Built with Sphinx using a theme provided by Read the Docs." A "Next" button is located in the bottom right corner of the main content area.

COMPUTE

| STORE

| ANALYZE

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](#)

© Copyright 2016, Cray Inc.

[Next](#)



COMPUTE

STORE

ANALYZE

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: AdvancedIter](#)
- [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: HDFSSIterator](#)
- [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](#)



COMPUTE

STORE

ANALYZE

Standard Modules: 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
- DynamicIterators
- Error
- FileSystem
- GMP
- Help
- IO
- List
- Math
- Memory
- Path
- Random
- Reflection
- Regexp
- Spawn
- Sys
- SysBasic
- SysCTypes
- Time
- Types
- UtilReplicatedVar

Docs » Standard Modules

[View page source](#)

Standard Modules

Standard modules are those which describe features that are considered part of the Chapel Standard Library.

All Chapel programs automatically [use](#) the modules `Assert`, `Math`, and `Types` by default.

- [Assert](#)
- [Barrier](#)
- [BitOps](#)
- [Buffers](#)
- [CommDiagnostics](#)
- [DynamicIterators](#)
- [Error](#)
- [FileSystem](#)
- [GMP](#)
- [Help](#)
- [IO](#)
- [List](#)
- [Math](#)
- [Memory](#)
- [Path](#)
- [Random](#)
- [Reflection](#)
- [Regexp](#)
- [Spawn](#)
- [Sys](#)
- [SysBasic](#)
- [SysCTypes](#)
- [Time](#)
- [Types](#)
- [UtilReplicatedVar](#)

Index

- [Chapel Online Documentation Index](#)

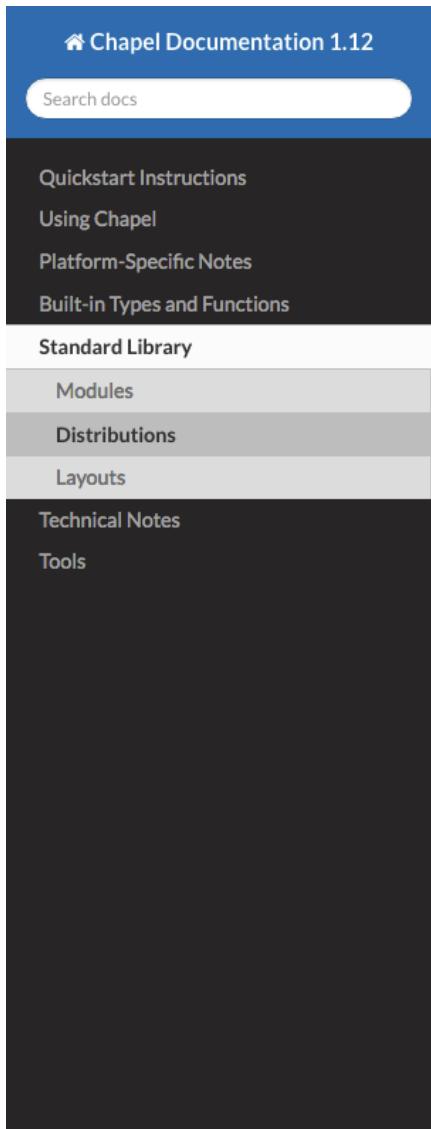


COMPUTE

STORE

ANALYZE

Layouts & Distributions: 1.12

A vertical sidebar for the Chapel Documentation. At the top is a blue header bar with the text "Chapel Documentation 1.12" and a search bar labeled "Search docs". Below this is a black sidebar area containing links: "Quickstart Instructions", "Using Chapel", "Platform-Specific Notes", "Built-in Types and Functions", "Standard Library" (which is expanded), "Modules", "Distributions" (which is selected and highlighted in grey), and "Layouts". Under "Technical Notes" are links for "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](#).



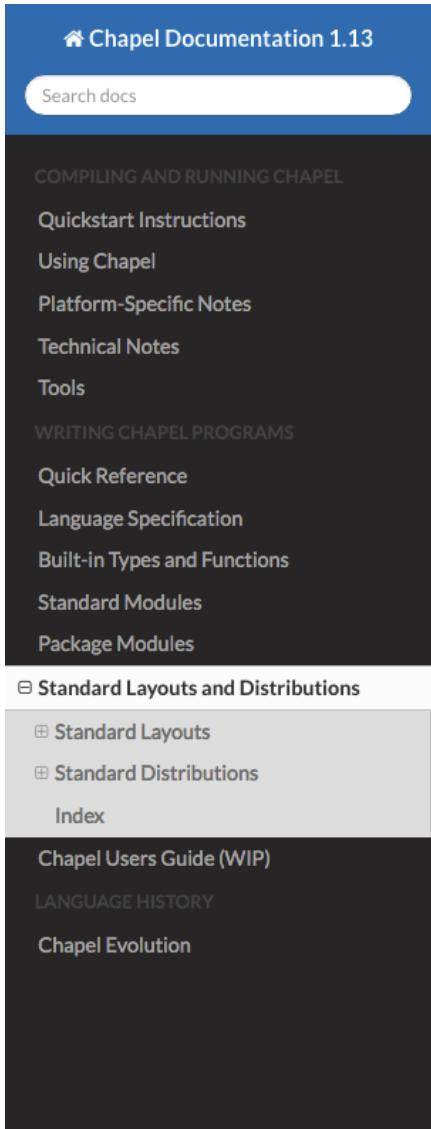
COMPUTE

STORE

ANALYZE

Copyright 2016 Cray Inc.

Layouts & Distributions: 1.13

A screenshot of the Chapel Documentation 1.13 website. The sidebar on the left contains the following navigation items:

- 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
 - ⊖ Standard Layouts
 - ⊖ Standard Distributions
 - Index
- Chapel Users Guide (WIP)
- LANGUAGE HISTORY
- Chapel Evolution

[Docs](#) › Standard Layouts and Distributions

[View page source](#)

Standard Layouts and Distributions

Standard Layouts

Standard layouts are domain maps that target a single locale and describe the local storage of domains and arrays.

- [LayoutCSR](#)

Standard Distributions

Standard distributions are domain maps that target multiple locales and describe how domains and arrays are stored across them.

- [BlockCycDist](#)
- [BlockDist](#)
- [CyclicDist](#)
- [DimensionalDist2D](#)
- [PrivateDist](#)
- [ReplicatedDist](#)
- [BlockCycDim](#)
- [BlockDim](#)
- [ReplicatedDim](#)

Index

- [Chapel Online Documentation Index](#)



COMPUTE

STORE

ANALYZE

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
- DynamicIterators
- 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] [-saver 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



COMPUTE

STORE

ANALYZE

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



COMPUTE

| STORE

| ANALYZE

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

 **Chapel User Resources**

<http://chapel.cray.com/community.html>

Overview
New?
Events
Opportunities

I Learn Chapel?
Sample Code
Documentation

Publications and Papers

Media + Blog Posts

I Chapel
Chapel Notes

Resources
Developer Resources

IRC: [#chapel](#) (chat.freenode.net)
The #chapel channel is intended for general discussion about Chapel. It is typically staffed by working members of the core development team.

Mailing Lists
The following mailing lists may be of interest:

- [chapel-announce](#): for announcements.
- [chapel-users](#): for user community discussions.
- [chapel-bugs](#): for reporting bugs.

Note that the chapel-announce mailing list receives mail for each commit to master.

Online Documentation
Most Chapel documentation is available online at [chapel-lang.org](#).

Chapel Implementers and Users
An annual workshop for Chapel implementers and users is held at Cray's headquarters in Livermore, CA.

Chapel User's Group (CHUG)
Since 2010, we've maintained a user group for Chapel users.

 **Chapel Developer Resources**

<http://chapel.cray.com/developers.html>

Overview
New?
Events
Opportunities

I Learn Chapel?
Sample Code
Documentation

Publications and Papers

Media + Blog Posts

I Chapel
Chapel Notes

Resources
Developer Resources

IRC: [#chapel-developers](#) (chat.freenode.net)
The #chapel-developers channel is intended for implementation-related discussion. It is typically staffed by working members of the core development team.

Mailing Lists
The following mailing lists may be of interest:

- [chapel-announce](#): gets ~12 Chapel announcements per year.
- [chapel-developers](#): for implementation-related discussions.
- [chapel-commits](#): receives mail for each commit to master.
- [chapel-test-results-regressions](#): notable testing result summaries.
- [chapel-users](#): for user community discussions.

Note that chapel-developers and chapel-users require subscriptions to post.

Performance Tracking
Chapel's nightly performance testing generates a number of graphs which can be used to track performance changes.

Contributing



COMPUTE

STORE

ANALYZE

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"



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



COMPUTE

| STORE

| ANALYZE

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

Package Modules

Standard Layouts and Distributions

Chapel Users Guide (WIP)

- Overview
- Base Language
- Task Parallelism
- Data Parallelism

Locality

- Locales in Chapel
- Compiling and Executing Multi-Locale Programs

Locale Type and Variables

- The *localeType*
- numLocales*
- The *LocalesArray*
- here*

On-Clauses

here

The final built-in locale variable that we'll cover in this section is *here*. For any given task, this variable resolves to the locale value on which the task is running.

As an example, the following program demonstrates that Chapel programs begin their execution on locale 0:

```
examples/users-guide/locality/here-id.chpl
```

```
writeln("Chapel programs start as a single task running on locale ", here.id);
```

Running it on any number of locales generates:

```
Chapel programs start as a single task running on locale 0
```

[Previous](#) [Next](#)

<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

THE CHAPEL USERS GUIDE IS DIVIDED INTO FOUR MAIN SECTIONS.

- Standard Modules
- Package Modules
- Standard Layouts and Distributions

⊖ Chapel Users Guide (WIP)

- Overview
- ⊕ Base Language
- ⊕ Task Parallelism
- Data Parallelism
- ⊕ Locality

LANGUAGE HISTORY

- Chapel Evolution

Base Language

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

- Simple Console Output: Hello world
- Variable Declarations

(more to come...)

Task Parallelism

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

- Task Parallelism Overview
- The `begin` statement

(more to come...)

<http://chapel.cray.com/docs/1.13/users-guide/index.html>

Other Documentation Improvements



COMPUTE

|

STORE

|

ANALYZE

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