Documentation Improvements

Chapel Team, Cray Inc.
Chapel version 1.11
April 2nd, 2015
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

● **Background**

● **Standard Module Documentation**

● **Website Updates**

● **Other Documentation Improvements**

● **Documentation Priorities and Next Steps**
Chapel Docs: Background

- Existing Chapel documentation:
  - Chapel Spec
  - Quick Reference Guide
  - READMEs in doc/ directory
  - Online tutorials
  - Primers

1.4. Hello World!

Let's begin with writing one of the simplest programs in any named hello.chpl:

```chapel
writeln("Hello world!");
```

And that's it. To compile it from a terminal in the same directory:

```
chpl -o hello hello.chpl
```

This invokes the Chapel compiler `chpl`, which translates the program because the program has been translated all the way into

```
./hello
```

If the result is not "Hello world!" then you should be concerned...
Chapel Docs: More Background

- Module documentation spread out
- Not easily searchable
- As a result, not updated often
Chapel Docs: This Effort

- Module documentation on the web
- Improved introduction to Chapel on website
  - “Hello, World!” examples on website
  - Code sample front-and-center on main page
Standard Module Documentation
Chapel Docs: Module Docs on the Web

URL: http://chapel.cray.com/docs/latest/
Chapel Docs: Module Documentation

- One page per module
- Module description at top
- Based on code comments
  - Updated chpdoc tool
- Supports rich formatting:
  - Emphasis
  - Bold
  - Links
  - Section titles
  - etc

Module: FileSystem

A file utilities library

The FileSystem module focuses on file and directory properties and operations. It does not cover every interaction involving a file—for instance, path-specific operations live in the Path module, while routines for opening, writing to, or reading from a file live in the io module. Rather, it covers cases where the user would prefer a file or directory to be handled wholesale and/or with minimal interaction. For example, this module contains File/Directory Manipulations and functions for determining the File/Directory Properties. Also included are operations relating to the current process’s file system state, which are performed on a specified locale (Locale State Functionality). The module also contains iterators for traversing the file system (File System Traversal Iterators).

File/Directory Manipulations

- copy
- copyFile
- copyTree
- mkdir
- remove
- symlink
- chmod
- chown
- copyMode
- rename

File/Directory Properties

- getGID
- getMode
- getUTD
- exists
- isDir
- isFile
- isLink
- isMount
- sameFile

Locale State Functionality

- locale.chdir
- locale.cwd
- locale.umask

File System Traversal Iterators

- glob
- listdir
- walkdirs
- findfiles

Constant and Function Definitions

- const S_IRUSR: int

S_IRUSR and the following constants are values of the form S_[R | W | X][USR | GRP | OTH], S_IRWX[U | G | O], S_ISUID, S_ISGID, or S_ISVTX, where R corresponds to readable, W corresponds to writable, X corresponds to executable, UGR and O correspond to user, GRP and O
Chapel Docs: Cross References (links)

- Supports inter-documentation links
- Reader can quickly find types, procedures, etc.

As in standard FFTW usage, the flow is to:

1. Create plan(s) using the `plan_dft*` routines.
2. Execute the plan(s) one or more times using `execute`.
3. Destroy the plan(s) using `destroy_plan`.
4. Call `cleanup`.

```chapel
proc execute(const plan: fftw_plan)

Execute an FFTW plan.

Arguments:  plan : fftw_plan – The plan to execute, as computed by a `plan_dft*()` routine.
```
Chapel Docs: Procedures and Iterators

- **Procedure docs include:**
  - Signature and procedure documentation
  - Arguments, return type and description

- **Supports rich formatting**

```chapel
proc copyTree(src: string, dest: string, copySymbolically: bool = false)
```

Will recursively copy the tree which lives under `src` into `dest`, including all contents, permissions, and metadata. `dest` must not previously exist, this function assumes it can create it and any missing parent directories. If `copySymbolically` is `true`, symlinks will be copied as symlinks, otherwise their contents and metadata will be copied instead.

Will halt with an error message if one is detected.

**Arguments:**
- `src : string` – The root of the source tree to be copied.
- `dest : string` – The root of the destination directory under which the contents of `src` are to be copied (must not exist prior to this function call).
- `copySymbolically : bool` – This argument is used to indicate how to handle symlinks in
Chapel Docs: Procedures and Iterators

- Arguments, return/yield have separate section
- Types can link to class or record docs

```
proc exists(name: string): bool

Determines if the file or directory indicated by

Will halt with an error message if one was dete

Arguments: name : string – The file or directory

Returns: true if the provided argument corre otherwise. Also returns false for bri

Return type: bool
```

```
iter glob(pattern = "*")

Yields filenames that match a given glob pattern (zippered or non-).

Arguments: pattern : string – The glob pattern to

Yields: The matching filenames as strings
```
Chapel Docs: Classes and Records

- Class/record description
- Member docs
- Method and iterators
- Cross-reference all items

record list

A singly linked list.

Note

destroy must be called to reclaim any memory used by the list.

type eltType

The type of the data stored in every node.

var length: int

The number of nodes in the list.

iter these()

Iterate over the list, yielding each element.

Yield type: eltType

proc append(e: eltType)

Append e to the list.
Chapel Docs: Enums, Types, Configs

● Enums
  
  ```chapel
enum MemUnits { Bytes, KB, MB, GB }
  
  The amount of memory returned by `locale.physicalMemory` can be
  bytes or as chunks of $2^{10}$, $2^{20}$, or $2^{30}$ bytes.
  ```

● Types
  
  ● Supports extern types
  
  ● And standalone types

● Configs
  
  ● Supports var/const too
  
  ● Globals also supported
  
  ```chapel
class config param noFFTWsizeChecks = false
  
  Controls execution-time array size checks in the FFTW checks).
  ```
Chapel Docs: Chapel Module Index

- Module Index lists all mods
- Brief description of each mod
- Links directly to modules
Chapel Docs: Search

- Built-in search function
- Google, et al. also work
Chapel Docs: Module Documentation

- All standard modules are documented
- Added and edited a lot of module documentation!
  - ~47k words
  - ~11k lines of reStructuredText
  - Every developer on the Cray team contributed docs
  - Took ~6 weeks to complete all the documentation
Website Updates
Chapel Docs: Website Updates

- “Hello, World!” examples on website
- New code snippet on front page

The Chapel Parallel Programming Language

What's Chapel?

If you're new to Chapel, you may want to:

- read a short introductory blog article or an even shorter overview statement
- read a more in-depth overview paper
- browse the slides from a recent Chapel overview talk
- download the release
- browse six versions of "Hello, world!" in Chapel and learn how to trivially write distributed-memory computations like this:

```chapel
use CyclicDist;  // use the Cyclic distribution library
config const n = 100000;  // override default using ./a.out --n=<val>
forall i in [1..n] dmapped Cyclic(startIdx=i) do
   writeln("Hello from iteration ", i, " of ", n, " running on node ", here.id);
```
Chapel Docs: Website Updates

- **Hello world examples on the website:**
  - [http://chapel.cray.com/hellos.html](http://chapel.cray.com/hellos.html)
  - Concise, yet thorough intro to language

"Hello, world!" Variants in Chapel

Here are six versions of "Hello, world!" from the Chapel release:

1. Simple version
2. "Production Grade" version
3. Data-Parallel version
4. Distributed-Memory Data-Parallel version
5. Task-Parallel version
6. Distributed-Memory Task-Parallel version

For more advanced computations, browse the examples directory from the Chapel release.
Chapel Docs: This Effort

- **Hello world examples on the website:**
  - [http://chapel.cray.com/hellos.html](http://chapel.cray.com/hellos.html)
  - Concise, yet thorough intro to language

---

**Simple Hello World**

This is the simplest "Hello, world!" in Chapel:

```chapel
1 writeln("Hello, world!");
```

*hello.chpl* hosted with 💖 by GitHub

**Next:** "Production-Grade" Hello World
Chapel Docs: This Effort

- Hello world examples on the website:
  - [http://chapel.cray.com/hellos.html](http://chapel.cray.com/hellos.html)
  - Concise, yet thorough intro to language

```
"Production-Grade" Hello World

This version uses a module, main(), and an execution-time configurable message to demonstrate a more structured coding style:

```c
1 /* This program is conceptually very similar to hello.chpl, but it
2 * uses a more structured programming style, explicitly defining a
3 * module, a configuration constant, and a main() procedure.
4 */
5
6 //
7 // define a module named 'Hello'. If a source file defines no
8 // modules, the filename minus its .chpl extension serves as the
9 // module name for the code it contains. Thus, 'hello' would be
10 // the automatic module name for hello.chpl.
11 //
12 module Hello {
```
Chapel Docs: This Effort

- Hello world examples on the website:
  - [http://chapel.cray.com/hellos.html](http://chapel.cray.com/hellos.html)

Task-Parallel Hello World

This version uses Chapel's **coforall-loop** to create a distinct task per iteration which prints its own message:

```chapel
/* This test uses Chapel's task parallel features to create a parallel hello world program that utilizes multiple cores on a single locale (node) */

// Each iteration prints out a message that is unique according to the value of tid. Due to the task parallelism, the messages may come out in any order. However, the writeln() procedure will prevent against finer-grained interleaving of the messages themselves.

coforall tid in 0..#numTasks do
  writeln("Hello, world! (from task " + tid + " of " + numTasks + ")");
```
Chapel Docs: Status

- **Module documentation available online**
  - Generated from source code and comments

- **Hello world programs seen by many web users**
  - Previously requested by critics
  - hellos.html has second-highest pageviews
Other Documentation Improvements
Other Documentation Improvements

- Minor improvements to Quick Reference document
- Documented class/record destructors in spec
  - Most frequently noted undocumented feature…
- Other spec improvements:
  - removed an outdated [] vs. () distinction from the spec
  - clarified that integer literals may be ‘uint’s if sufficiently large
  - additional updates and improvements
- Added a note for ‘zsh’ users to the top-level README
- Improved documentation for slurm* launchers
- Noted long-standing feature to squash reference counting
  - (at the cost of leaking all arrays…)
  - a stopgap, see $CHPL_HOME/PERFORMANCE for details
Documentation Priorities and Next Steps
Documentation Priorities and Next Steps

- Continue to update docs with module changes
- chpldoc Chapel features that have library-like interfaces
  - e.g., arrays-as-vectors, methods on ranges and domains, etc.
- Add primers to web as a broader Chapel tutorial
- Make remaining doc/ READMEs web pages
- Revise “Hello world” comments with web reader in mind
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, URIKA, and YARCDATA. 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.

Copyright 2015 Cray Inc.