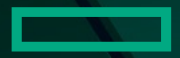


Hewlett Packard
Enterprise

CHAPEL 1.27.0/1.28.0 RELEASE NOTES: LIBRARY IMPROVEMENTS

Chapel Team

June 30, 2022 / September 15, 2022

OUTLINE

- New 'Communication' Module
- 'min' and 'max' Improvements
- Literal and Newline IO Methods
- Reduced I/O Buffer Memory
- New ‘OS’ / ‘OS.POSIX’ Modules
- 2.0 Library Stabilization
- Other Library Improvements

NEW ‘COMMUNICATION’ MODULE

NEW 'COMMUNICATION' MODULE

Background and This Effort

Background:

- Users have requested the ability to move data using low-level get/put calls
  - to avoid potential overheads from Chapel array assignment
  - necessary when working with C pointers
  - 'CopyAggregation' module and some optimized codes have used non-user-facing compiler primitives

This Effort:

- Chapel 1.28 introduces a new standard module: 'Communication'
  - currently only two functions: 'get' and 'put':

```
proc get(dest: c_void_ptr, src: c_void_ptr, srcLocID: int, numBytes: integral)
proc put(dest: c_void_ptr, src: c_void_ptr, destLocID: int, numBytes: integral)
```

NEW 'COMMUNICATION' MODULE

Status and Next Steps

Status:

- Uses of primitives have been replaced with calls to functions in 'Communication'
  - e.g., the 'CopyAggregation' module now uses 'Communication' rather than put/get primitives

Next Steps:

- Expand the interface
  - wide reference manipulation: creating one from a C pointer, getting a C pointer from an existing one
  - non-blocking communication
  - collective communication

- Module structure design
  - where do collectives go?
  - where do existing barrier implementations go?
  - should 'CommDiagnostics' (not a 2.0 module) be a submodule in 'Communication'?

‘MIN’ AND ‘MAX’ IMPROVEMENTS

‘MIN’ AND ‘MAX’ IMPROVEMENTS

Background: Historically, ‘min’ and ‘max’ have had surprising behavior when mixing signed and unsigned

```
var myInt: int, myUint: uint, myInt32: int(32), myUint32: uint(32);

min(myInt, myUint);       // produced a real(64)
max(myInt, myUint);       // produced a real(64)

min(myInt32, myUint32); // produced an int(64)
max(myInt32, myUint32); // produced an int(64)
```

- At the same time, we have supported comparisons (e.g. ‘<’) between signed and unsigned integers

This Effort: Adjust ‘min’ and ‘max’ overloads to support a mix of signed and unsigned integers

Impact: Now, the behavior is less confusing

```
min(myInt, myUint);       // now produces an int(64)
max(myInt, myUint);       // now produces a uint(64)

min(myInt32, myUint32); // now produces an int(32)
max(myInt32, myUint32); // now produces a uint(32)
```

LITERAL AND NEWLINE IO METHODS

LITERAL AND NEWLINE IO METHODS

Background

- Consider a simple textual representation of a list:

```
[1, 2, 3, 4]
```

- A reasonable, but incorrect, approach to reading this list might try to use the string literal "[":

```
var openBracket = "[";
myReader.read(openBracket);    // openBracket set to "[1," and channel points to whitespace
```

  - Channels support read/write methods whose behavior depends on an argument's type, not its value
  - Problem: Reading a string variable **by design** ignores the string contents & reads until whitespace

- Writing a list to a formatted channel, e.g. one configured for JSON, also has challenges for string literals
  - Writing a string to a channel configured for JSON wraps the string in quotes, **by design**:

```
jsonWriter.write("[");    // prints square bracket with quotes: "["
```

  - However, sometimes you may not want that, like when printing out the list's brackets in this example
    - Historical workaround: use the 'ioLiteral' and 'ioNewline' types
    - Channels know that the 'ioLiteral' should bypass any formatting

```
jsonWriter.write(new ioLiteral("["));    // correctly prints: [
```

LITERAL AND NEWLINE IO METHODS

This Effort

- We intend to deprecate ‘ioLiteral’ and ‘ioNewline’ [#19487]
  - It’s potentially confusing to need to use different types
  - In the rest of IO, one generally uses a different method to achieve a different behavior
    - How to apply this philosophy and support the same use cases?
- Introduced ‘readLiteral’, ‘matchLiteral’, and ‘writeLiteral’ (plus ‘*Newline’ versions)
  - These accept either ‘string’ or ‘bytes’ arguments and can ignore leading whitespace, e.g.:

```
proc channel.readLiteral(literal: string, ignoreWhitespace=true): void throws
```

  - These ignore the channel’s formatting (and the Encoder/Decoder when available)
- ‘Read’ and ‘Match’ versions differ in how they handle the case when the literal is not found
  - ‘Read’ will throw
    - Useful when expecting to find the literal, so that errors can be caught or propagated upward
  - ‘Match’ will return ‘false’
    - Sometimes it’s useful to perform a speculative read to see if there’s more data, especially in a loop

LITERAL AND NEWLINE IO METHODS

Impact and Next Steps

Impact: New, unstable methods are available for users to try instead of ‘ioLiteral’ or ‘ioNewline’

- Marked as unstable pending some minor design decisions (see below)
- See documentation for more details
- ‘Read’ and ‘Match’ versions complement each other for elegant code:

```
// reading text like “[1, 2, 3, 4]” into a ‘list(int)’, expects at least one element
r.readLiteral("[");  // throws an error if ‘[’ is not found
do data.append(r.read(int)); while r.matchLiteral(",");  // breaks loop once commas cannot be found
r.readLiteral("]");
```

Next Steps:

- Replace existing uses of ‘ioLiteral’ and ‘ioNewline’ in our internal and standard modules
- Answer design questions in order to stabilize the interface:
  - How to handle leading whitespace in the given ‘string’ or ‘bytes’ argument?
  - Should ‘readNewline’ and ‘matchNewline’ have an optional ‘ignoreWhitespace’ argument?

REDUCED I/O CHANNEL BUFFER MEMORY USAGE

REDUCED I/O CHANNEL BUFFER MEMORY USAGE

Background:

- I/O channels buffer data in memory
- Application reads/writes are buffered in their entirety
- A single read can only be about 1/3 the size of physical memory, a write about 1/2

This Effort:

- Break large reads/writes of unstructured data into smaller fixed-size operations on the underlying file
- Only buffer a portion of the read/write in the channel

Impact:

- Large reads/writes can be almost the size of physical memory
- Significantly reduces memory requirements

Next Steps:

- Investigate reducing buffering of structured data (e.g., array of integers)

NEW ‘OS’ AND ‘OS.POSIX’ MODULES

NEW ‘OS’ AND ‘OS.POSIX’ MODULES

Background:

- Chapel has supported several standard modules providing access to system-level capabilities
- Organization and roles have been poorly defined, but have seen improvements in recent releases
  - See the past few editions of release notes for examples

This Effort:

- Introduced new ‘OS’ and ‘OS.POSIX’ modules
  - **OS:** contains portable features and interfaces using standard Chapel naming conventions
  - **OS.POSIX:** sub-module containing POSIX-specific features using POSIX names whenever possible
    - Implemented a large swath of POSIX features for it
    - Could imagine future sibling modules for other, non-POSIX operating systems if/when desired (e.g., ‘OS.Windows’)

Status:

- Chapel’s features that wrap and support system errors are now contained within this ‘OS’ module
- Most POSIX/POSIX-like features from other system-oriented modules have been deprecated and/or [re]moved
- (see subsequent slides for further details)

2.0 LIBRARY STABILIZATION

2.0 LIBRARY STABILIZATION
Background

- Our primary focus is standard library stabilization
  - *Stabilization:* Going forward, all changes will be backwards-compatible
    - Users should be able to depend on anything not marked @unstable to continue working through all 2.##.# releases.

Our review process:

- On even weeks, we reviewed a new library, scrutinizing
  - naming: the module, public types, public procedures, …
  - placement: is this the right place for these symbols?
  - behaviors / definitions of all public symbols

- On odd weeks we had followed up on a previously reviewed library

- Also created a sub-team to review the IO module
  - IO sub-team members meet regularly and call full-team meetings when part of the interface is ready for discussion

2.0 LIBRARY STABILIZATION

This Effort

- In 1.26 we had:
  - Reviewed 30 standard libraries (out of 38 total)
  - Stabilized 2 standard libraries

- During the 1.27/1.28 release cycles we:
  - Reviewed 8 more standard libraries
  - Implemented many changes based on reviews
  - Finished a first-round review of every module slated for 2.0

- We also added the '@unstable' attribute to mark symbols we don't intend to stabilize for 2.0
  - See the language deck for more details

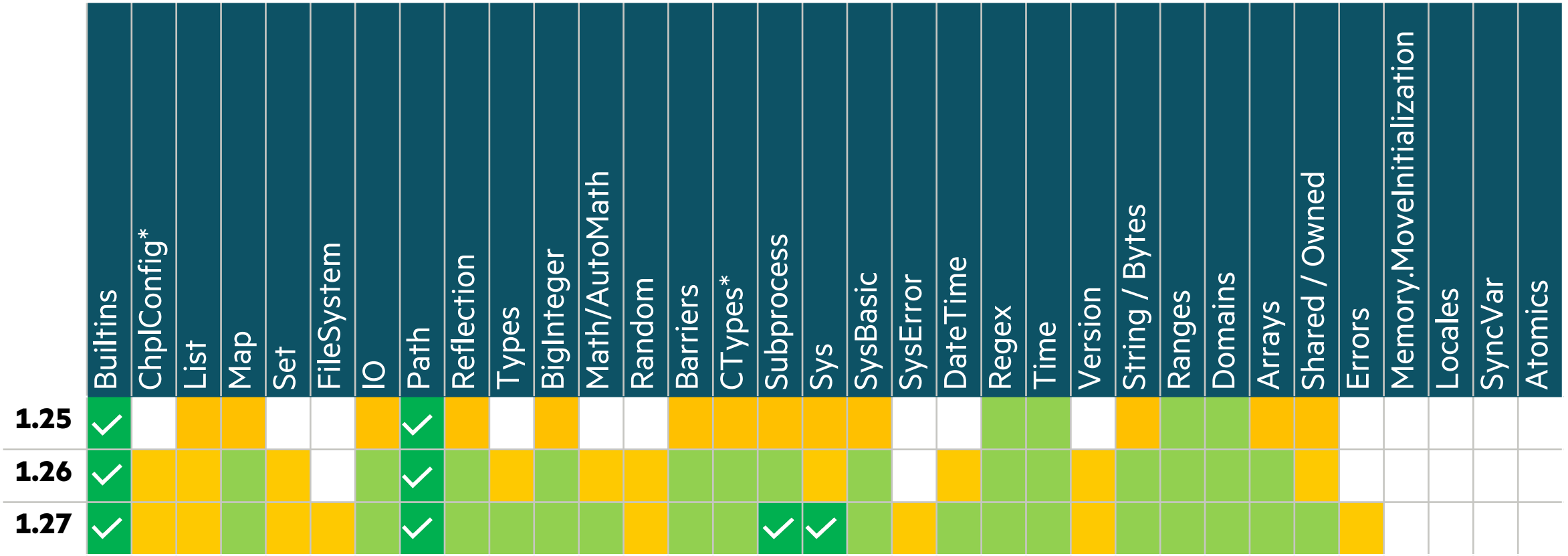
STANDARD LIBRARY STABILIZATION

Status: In Numbers

- 38 modules reviewed
- 5 modules stabilized:
  - Path, Builtins, Subprocess, SysError, Sys
- 7 modules that are close to being stabilized:
  - CTypes, Regex, Time, DateTime, Version, Locales, Types
- 7 modules that we've decided not to stabilize before Chapel 2.0:
  - CommDiagnostics, Memory[.Diagnostics], BitOps, GMP, DynamicIters, VectorizingIterator, Help
- 1 module that we're unlikely to stabilize unless we have time:
  - Heap

2.0 LIBRARY STABILIZATION

Status: Visualized

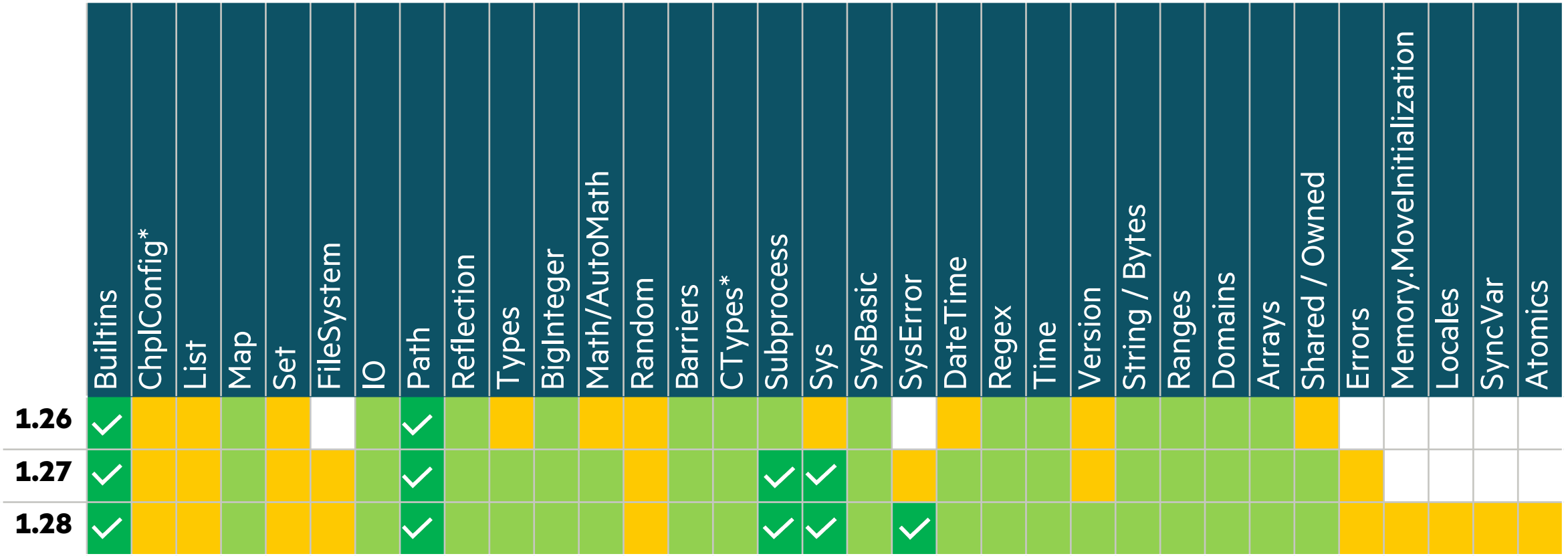


Stable **Progress** **Review Started**

* - ChapelEnv was renamed to ChplConfig, and CPtr / SysCTypes were combined and renamed to CTypes

2.0 LIBRARY STABILIZATION

Status: Visualized



Stable **Progress** **Review Started**

* - ChapelEnv was renamed to ChplConfig, and CPtr / SysCTypes were combined and renamed to CTypes

2.0 LIBRARY STABILIZATION

- Sys
- SysError
- SysBasic
- IO
- Math / AutoMath
- BigInteger
- DateTime / Time
- Types

SYS MODULE

Background:

- The 'Sys' module contained symbols and procedures used in low-level programming
  - Provided thin interfaces over POSIX and other Unix libraries and system calls
  - Acted as a sort of catch-all for systems-level interfaces that didn't have a more logical home

Actions Taken:

- Deprecated the 'Sys' module in favor of organizing content under more specific modules [#19904]
  - POSIX functionality was moved to a new 'OS.POSIX' sub-module
    e.g., 'Sys.sys_fd_set' was deprecated in favor of 'OS.POSIX.FD_SET' in 1.27 and removed in 1.28
  - Most socket functionality was moved to the 'Socket' package module
    e.g., 'Sys.SO_ERROR' was deprecated in favor of 'Socket.SO_ERROR' in 1.28
  - All other symbols were deprecated along with the module itself
    e.g., various network and IP constants unused in other modules or tests

SYSERROR MODULE

Background:

- A module defining common system-level errors

Actions Taken / Decisions Made:

- Deprecated the entire module, moving its contents to the ‘OS’ module
- Renamed functions from ‘SysError’
  - ‘SystemError.fromSyserr’ is now ‘.createSystemError’
- Renamed and moved an error type from ‘SysBasic’ to ‘OS’
  - ‘SysBasic.syserr’ is now ‘OS.errorCode’
- Renamed class names with acronyms to match the preferred style, e.g.,
  - ‘BlockingIOError’ is now ‘BlockingIoError’
  - ‘IOError’ is now ‘IoError’
  - ‘EOFError’ is now ‘EofError’
- The former contents of ‘SysError’ are now considered stable

SYSBASIC MODULE

Background:

- Along with 'SysCTypes' and 'CPtr', this was a grab-bag of C type aliases and error codes
- Had already moved some contents to new 'CTypes' or 'OS.POSIX' modules
  - Needed to decide what to do with the remainder

Actions Taken / Decisions Made:

- 'syserr' has been renamed to 'errorCode' and moved to the 'OS' module
- 'err_t' has been replaced with 'c_int' (to match C interfaces) and has been deprecated [#20123]
- Error codes that were part of 'OS.POSIX' in 1.27 were deprecated and then removed in 1.28

- Decided to deprecate or hide the remainder of the module as implementation details:
  - 'fd_t' will be deprecated [#20128]
  - Error codes we added (e.g., 'EEOF') will be implementation details [#20129, #20130]
  - Linux-specific and POSIX STREAM extension error codes will be deprecated [#20131, #20132]
- This means the whole 'SysBasic' module will be deprecated

IO MODULE

Background and Actions Taken

Background:

- The IO module handles reading and writing to files, as well as formatted IO
  - 'write( )', 'writeln( )' and 'writef( )' are provided by default, all other IO functions are defined in the IO module
- Implements 'file' and 'channel' types
- This module is very large, ~7300 lines
- It also has many known API design issues

Actions Taken:

- IO subteam completed review of the IO module and made proposals for Chapel 2.0
- Presented most of the proposals to the entire Chapel team for feedback and approval
- Continued implementing approved proposals (see next slide)

IO MODULE

Status

Completed:

- Deprecated the ‘iohints’ type in favor of the new ‘ioHintSet’ type [#20141]
- Deprecated the ‘<~>’ operator on channels [#19501]
- Introduced methods for reading and writing literal text and newlines [#19487]
- Marked ‘iostyle’ type as unstable rather than deprecated
- Deprecated ‘start’ and ‘end’ arguments in favor of ‘region’ range [#20133]

Pending:

- Rename I/O ‘channel’ type to ‘fileReader’ and ‘fileWriter’ [#18112]
- Add an extensible Encoder/Decoder mechanism [#18499]
  - Deprecate ‘j’ and ‘h’ format string specifiers in favor of Encoders/Decoders
- Continue redesign and deprecation of various channel methods

IO MODULE

Open Discussions and Next Steps

Open Discussions:

- What should be done with the ‘iokind’ field on channels? [#19314]
- Clean up ‘read’ functionality [#19498]
- Replace ‘readstring’ and ‘readbytes’, mimic Python’s behavior [#19496]
- Should ‘assertEOF’ be replaced with ‘atEOF’? [#19316]
- What should be done with the various file-creating functions? [e.g., openfd: #20143]

Next Steps:

- Reach decisions on the open discussion items above
- Implement the Encoder/Decoder design
- Rename ‘channel’ to ‘fileReader’ and ‘fileWriter’
- Resolve ‘readline’ vs ‘readln’ vs ‘read*line’ functionality [#19495]

MATH / AUTOMATH MODULES

Background:

- 'Math' module provided mathematical constants and functions, e.g., 'e', 'sqrt( )', 'gcd( )'
  - Names were usually based on C's interface, which was influenced by ISO standards
- Was included in all programs by default

Actions Taken / Decisions Made:

- Split into two modules, 'AutoMath' and 'Math' [#18989]
  - 'AutoMath' continues to be included in all programs by default, 'Math' now requires a 'use' or 'import' statement
  - Some symbols will cease to be included by default as they are discussed [#18990]
- Will mostly stick with C/ISO standard conventions

Open Discussions:

- Continue reducing symbols included by default [#18990]
- Rounding support is incomplete, should it be extended for 2.0? [#19024]
- Should we use 'gamma( )' or 'tgamma( )' for the gamma function name? [#19022]

BIGINTEGER MODULE

Background:

- Provides 'bigint' type for storing and manipulating arbitrary precision integers

Actions Taken:

- Deprecated 'fits_*_p( )' methods and replaced with a new 'fitsInto(type t: integral)' method [#17702]

```
var bSmall = if b.fitsInto(int(16)) then b:int(16) else 0;
```

- Unified behaviors of 'bigint.mod( )' and 'bigint.%' with their 'int/uint' counterparts [#17713]
- Modified 'invert' to throw on illegal arguments rather than leaving 'this' undefined [#17708]

Open Discussions:

- Should methods be rewritten to store result in a third argument rather than updating receiver? [#17699]

```
bigint.add(c, a, b);   // semantically equivalent to 'c = a + b;' currently: 'c.add(a, b)'
```

- Revisiting the name of the 'round' enum—conflicts with 'Math.round( )'
- There are 13 other small library stabilization issues remaining that are likely uncontentious
  - See the full list of issues
  - And 9 non-breaking changes in progress

DATETIME / TIME MODULES

Background:

- Provides types and procedures for reasoning about and manipulating dates and times

Actions Taken / Decisions Made:

- Combined the ‘DateTime’ and ‘Time’ modules into a single ‘Time’ module
- Adjusted names of several methods to use camelCase
- Deprecated ‘datetime.today( )’ in favor of its synonym ‘datetime.now( )’
- Marked time zones as unstable – we expect interface changes as concrete time zones are added
- Deprecated subtracting a ‘date’ from a ‘datetime’ – it’s ambiguous what the time part should be in the ‘date’

Open Discussions:

- Discussion about capitalization of symbols, (e.g., ‘DayOfWeek.Monday’ vs. ‘DayOfWeek.monday’) [#18846]
- Some questions about C interoperability wrappers [#18833]
  - Should we keep them as-is? Move them? Remove them? Add a ‘c_’ prefix?
- Rename ‘Timer’ to ‘stopwatch’ and make any other stabilizing changes to its interface [#16393]

TYPES MODULE

Background:

- This module contains functions to query and modify types

Actions Taken:

- Ensured we have 'isXType', 'isXValue', and 'isX' functions for each type 'X'
  - We previously had these for some types and not others; this effort makes things more consistent
- Deprecated 'isFloatType'/ 'isFloatValue' / 'isFloat'
  - These procs returned true for 'real' and 'imag' types/values but not 'complex'
  - Saw instances in our own and user code where 'isFloat' was used when user actually meant 'isReal'
  - Someday we may want to add a generic 'floatingPoint' type (similar to ‘integral’) but aren't settled on the name

Open Discussions:

- Remove type/subtype comparison operators (in favor of named functions) [#19363]
  - Instead of 'derivedClass < parentClass' do 'isSubtype(derivedClass, parentClass)'
  - Removing these causes 100+ failures in one of our user codes
  - However, we have received feedback that users found the operators confusing
  - For the time being we have marked these unstable while we gather more feedback

OTHER LIBRARY IMPROVEMENTS

OTHER LIBRARY IMPROVEMENTS

For a more complete list of library changes and improvements in the 1.27.0 and 1.28.0 releases, refer to the following sections in the CHANGES.md file:

- 'Namespace Changes'
- 'Changes / Feature Improvements in Libraries'
- 'Name Changes in Libraries'
- 'Deprecated / Unstable / Removed Library Features'
- 'Standard Library Modules'
- 'Memory Improvements'
- 'Documentation'
- 'Bug Fixes'

THANK YOU

https://chapel-lang.org
@ChapelLanguage