SST user manual

Version: 0.40.1

Date: 2024-12-12

Stealth Software Technologies, Inc.

Table of Contents

1. Build system

1.1. Overview

The SST build system is built on top of the GNU Autotools. Before you can start using the build system, you need to identify which *package format* of SST you're working with. SST is generally available in three slightly different package formats:

1. The *source repository*. This repository is only available internally within Stealth. The other package formats are generally derived from this one.

2. A *distribution repository*. The primary public distribution repository for SST is available at https://github.com/stealthsoftwareinc/sst. You may also have access to other distribution repositories that Stealth has created for private distribution purposes.

3. A *distribution archive*.

To identify which package format you're working with, use the following algorithm:

1. If you started with an archive file like `sst-<version>.tar.gz` and it extracts to a directory that includes a file named `Makefile.in`, then you're working with a distribution archive.

   IMPORTANT If the extracted directory does not include a file named `Makefile.in`, then you may have mistakenly downloaded a *source archive*. This is what you will typically get if you use a download button when viewing the source repository in a repository manager like GitHub or GitLab. SST does not support being built from a source archive.

   Note that this only applies to download buttons for the source repository, not for a distribution repository. Using a download button on a distribution repository will generally yield a valid distribution archive.

2. Otherwise, if the version control repository includes a file named `Makefile.in` in the root directory, then you're working with a distribution repository.

3. Otherwise, you're working with the source repository.

Once you know which package format you're working with, using the build system consists of three main steps:

1. *Initializing* the build system.

2. *Configuring* the build system.

3. *Running* the build system.

If you're working with the source repository, you need to perform steps 1, 2, and 3. Otherwise, you only need to perform steps 2 and 3. Step 1 is only designed to work on modern Linux systems with a variety of tools installed, but steps 2 and 3 are designed to work on a variety of Unix-like systems with minimal tools installed. The idea is that developers work with the source repository and produce distribution archives and distribution repositories for general use.

1.2. Initializing the build system

Initializing the build system is performed with the `./autogen` command.

1.3. Configuring the build system

Configuring the build system is performed with the `./configure` command.

1.3.1. Build groups

Sometimes you might only be interested in building certain parts of SST, not all of it. For example, if you're using SST in a C project, you might only be interested in building the C library. To provide control over this, both the `./configure` script and the `install/*` scripts support the `--with-build-groups` option, whose syntax is as follows:

```
--with-build-groups=[[+|-]<group>[,[+|-]<group>]...]
```

Each `<group>` in the comma-separated list specifies some part of the build system that should be enabled or disabled. The group will be enabled if the **‘‘’’ prefix is given, or disabled if the ’’‘‘-’’’’ prefix is given. Omitting the prefix is the same as giving the ’’‘‘** prefix. All groups begin as disabled, and the entries of the list will be applied in order. If the `--with-build-groups` option is omitted, the behavior is as if it had been given as `--with-build-groups=default`, i.e., the default set of build groups will be enabled. Enabling or disabling certain build groups may implicitly enable or disable other build groups depending on how they are related.

The build groups are as follows:

`default`

: The default set of build groups: `bash`, `c`, `cpp`, and `java`.

`bash`

: The Bash library.

`c`

`c-autotools`

: The C library as built and installed by Autotools. When disabled, also disables the `cpp-autotools` build group.

`cpp`

`cpp-autotools`

: The C++ library as built and installed by Autotools. When enabled, also enables the `c-autotools` build group.

`java`

`java-gatbps`

: The Java library as built and installed by GATBPS. When disabled, also disables the `java-maven` build group.

`java-maven`

: The Java library as built and installed by Maven.

1.4. Running the build system

Running the build system is performed with the `make` command.

1.5. Makefile target reference

`manual`

: Forwards to the `manual-xz` target.

`manual-xz`

: Builds `doc/sst-0.40.1-manual.tar.xz`.

  Note that this file is subject to `DISTCUT`. In other words, it is included prebuilt in all distribution repositories and distribution archives, and its outgoing edges in the make graph are cut so that it will always be considered up to date. To leave the edges uncut, use `./configure DISTCUT=` or `make DISTCUT=`.

`src/rpm/centos-7/out`

: Builds the CentOS 7 RPM package. The `src/rpm/centos-7/out` directory will be produced, containing several `.rpm` files.

  Example 1.

```
$ make src/rpm/centos-7/out && echo ok
▮
ok

$ tree src/rpm/centos-7/out
src/rpm/centos-7/out
|-- sst-sst-0.16.1-0.1.gc3e38cc91.el7.src.rpm
|-- sst-sst-0.16.1-0.1.gc3e38cc91.el7.x86_64.rpm
‘-- sst-sst-debuginfo-0.16.1-0.1.gc3e38cc91.el7.x86_64.rpm
```

`src/rpm/rhel-7/out`

: Builds the RHEL 7 RPM package. The `src/rpm/rhel-7/out` directory will be produced, containing several `.rpm` files.

Example 2.

```
$ make src/rpm/rhel-7/out && echo ok
▯
ok

$ tree src/rpm/rhel-7/out
src/rpm/rhel-7/out
|-- sst-sst-0.16.1-0.1.gc3e38cc91.el7_9.src.rpm
|-- sst-sst-0.16.1-0.1.gc3e38cc91.el7_9.x86_64.rpm
‘-- sst-sst-debuginfo-0.16.1-0.1.gc3e38cc91.el7_9.x86_64.rpm
```

1.6. Building with Maven

When the `java-maven` build group is enabled, Maven can be used after running `./configure`. However, you should run `make maven-prep` before every `mvn` command.

2. Bash library

2.1. Including the library

1.
```
#! /bin/sh -
set -e || exit $?
. sst.bash
```

2.
```
#! /bin/sh -
set -e || exit $?
if test -f sst.bash; then
  . ./sst.bash
else
  . sst.bash
fi
```

3.
```
#! /bin/sh -
case $0 in /*)
  x=$0
;; *)
  x=./$0
esac
r='\(.*/\)'
x=‘expr "x$x" : "x$r"‘. || exit $?
set -e || exit $?
. "$x/sst/sst.bash"
```

For repository scripts:

```
#-------------------------------------------------------------------------
# Include the SST Bash library
#-------------------------------------------------------------------------
```

```
#
# Include the first sst.bash file found by searching up the directory
# tree starting from the location of this script file. If no such file
# is found, fall back to including plain sst.bash with PATH resolution.
#
# This section is written in portable shell to ensure it works properly
# in any shell.
#

case ${SST_SEARCH_UP_X-} in '')
  case $0 in /*)
    SST_SEARCH_UP_X=$0
  ;; *)
    SST_SEARCH_UP_X=./$0
  esac
  SST_SEARCH_UP_R='\(.*/\)'
  SST_SEARCH_UP_X=`
    expr "x${SST_SEARCH_UP_X?}" : "x${SST_SEARCH_UP_R?}"
  `. || exit $?
  unset SST_SEARCH_UP_R
  SST_SEARCH_UP_X=`
    cd "${SST_SEARCH_UP_X?}" || exit $?
    while :; do
      if test -f sst.bash; then
        case ${PWD?} in *[!/\]*)
          printf '%s\n' "${PWD?}"/ || exit $?
        ;; *)
          printf '%s\n' "${PWD?}" || exit $?
        esac
        exit
      fi
      case ${PWD?} in *[!/\]*)
        cd ..
      ;; *)
        exit
      esac
    done
  ` || exit $?
  export SST_SEARCH_UP_X
esac
set -e || exit $?
. "${SST_SEARCH_UP_X?}"sst.bash
unset SST_SEARCH_UP_X
```

2.2. Bootstrapping

The SST Bash library prelude runs the following bootstrap clause before running any Bash-specific code that may not be portable to other shells:

```
case ${SST_BASH_BOOTSTRAP+x}y$# in
  y0) SST_BASH_BOOTSTRAP= exec bash - "$0" ;;
  y*) SST_BASH_BOOTSTRAP= exec bash - "$0" "$@" ;;
```

```
esac
unset SST_BASH_BOOTSTRAP
```

This clause re-executes the script with the `PATH`-resolved `bash`. There are two reasons for doing this. First, if the script uses a `/bin/bash` shebang on a system where the `PATH`-resolved `bash` is newer than `/bin/bash` (e.g., on macOS), then the script will be re-executed with the newer version. Second, since this code is portable enough to run in any shell, a script can actually use a `/bin/sh` shebang, which is the most portable shell shebang, and still get bootstrapped into the `PATH`-resolved `bash`. The `unset` command prevents other scripts from being tricked into believing they've already bootstrapped.

2.3. Reserved identifiers

The SST Bash library reserves all identifiers beginning with `sst_` or `SST_` in both the variable namespace and the function namespace.

2.4. Postmortem job containers

As part of its EXIT trap, the prelude includes opt-in code that can detect whether the script is running in a CI job container, and if so, commit and push the container to a registry for later inspection. Such a container is called a *postmortem job container*.

The code behaves as follows:

1. If the script is not running in a CI job, stop.
2. If the script is being called by another script that uses the SST Bash library, stop.

   This means that, in a call stack that includes multiple scripts that use the SST Bash library, only the outermost such script will possibly push the postmortem job container.
3. If the `SST_PUSH_POSTMORTEM_JOB_CONTAINER` environment variable is unset or empty, stop.

   You must set the `SST_PUSH_POSTMORTEM_JOB_CONTAINER` environment variable to a nonempty value to opt in. If you leave it unset or empty, postmortem job containers will never be pushed.
4. If the `SST_NO_PUSH_POSTMORTEM_JOB_CONTAINER` environment variable is nonempty, stop.

   You can set the `SST_NO_PUSH_POSTMORTEM_JOB_CONTAINER` environment variable to a nonempty value to opt out regardless of any opting in.
5. If the script is running in a GitLab CI job:

   a. If the job is protected and the `SST_PUSH_PROTECTED_POSTMORTEM_JOB_CONTAINER` is unset or empty, stop.

      You must set the `SST_PUSH_PROTECTED_POSTMORTEM_JOB_CONTAINER` environment variable to a nonempty value to opt in for protected jobs. If you leave it unset or empty, postmortem job containers will never be pushed for protected jobs.

2.5. General development guidelines

This section provides guidelines for using the SST Bash library and writing Bash code in general.

The minimum version of Bash supported by the SST Bash library is Bash 4.1. However, development work is typically done on a system with a newer version of Bash. This makes it easy to accidentally introduce subtle dependencies on the newer version, which you may want to avoid depending on the constraints of your project. This section provides a set of guidelines that you can follow to reduce problems when aiming to support Bash 4.1.

2.5.1. Do not make global variables `readonly`

When a global variable is made `readonly`, it cannot be shadowed by a `local` variable in a function.

Example 3.

Script: `a.bash`

```
set -e || exit $?
. src/bash/sst.bash

w=hello
readonly w
echo $w

function g {
  local w
  w=world
  readonly w
  echo $w
}; readonly -f g

g
```

Possible output

```
hello
a.bash: line 9: local: w: readonly variable
a.bash: error: command exited with status 1: local w
    at g(a.bash:12)
    at main(a.bash:15)
```

This can generally be avoided by preferring to put code into functions and using `local` variables.

Example 4.

Script: `a.bash`

```
set -e || exit $?
. src/bash/sst.bash
```

```
function f {
  local w
  w=hello
  readonly w
  echo $w
}; readonly -f f

function g {
  local w
  w=world
  readonly w
  f
  echo $w
}; readonly -f g

g
```

Output

```
hello
world
```

2.5.2. Do not make `IFS` `readonly`

As an important special case of Section 2.5.1, do not make `IFS` `readonly`.

2.5.3. Global variable declarations

Bash <4.2 does not support `declare -g` in functions. Ignoring shadowing problems, global variable declarations can always be safely omitted except when a function needs to use a global associative array that may not have been declared yet. In this case, instead of writing `declare -g -A foo` in the function, write `declare -A foo` in the global scope just before the function.

For stylistic reasons, you may want to put commented-out `# Bash >=4.2: declare -g` declarations in a function for all global variables used by the function regardless of whether the declarations would technically be necessary. These declarations are helpful to the reader, and you can enable them later if you eventually migrate to Bash >=4.2.

WARNING If you're working on the SST Bash library itself, do not put any global variable declarations at the top of any function file. Although that area looks like it's the global scope, it actually isn't, as the library's automatic function loading mechanism loads function files in function scope. Instead, add a commented-out `# Bash >=4.2: declare -g` declaration to the function, and, if you're declaring a global associative array, also add a declaration to the "Global associative array declarations" section in the prelude.

2.5.4. Always use `x="$@"` to collect the positional parameters separated by spaces

Some versions of Bash sometimes have incorrect behavior for `x=$*` and `x=$@`, while `x="$@"` always works:

```
$ cat a.bash
set -e -u -o pipefail
for v in 4.{1,2,3,4} 5.{0,1}; do
  docker run -i -t --rm bash:$v bash --version | awk 'NR == 1'
  for rhs in '$*' '$@' '"$*"' '"$@"'; do
    printf '  %7s ' "x=$rhs:"
    docker run -i -t --rm bash:$v bash -c '
      set " "
      unset IFS
      x='"$rhs"'
      printf "[%s]\\n" "$x"
    '
  done
done

$ bash a.bash
GNU bash, version 4.1.17(2)-release (x86_64-pc-linux-musl)
    x=$*: [ ]
    x=$@: []
  x="$*": [ ]
  x="$@": [ ]
GNU bash, version 4.2.53(2)-release (x86_64-pc-linux-musl)
    x=$*: [ ]
    x=$@: []
  x="$*": [ ]
  x="$@": [ ]
GNU bash, version 4.3.48(1)-release (x86_64-pc-linux-musl)
    x=$*: []
    x=$@: [ ]
  x="$*": [ ]
  x="$@": [ ]
GNU bash, version 4.4.23(1)-release (x86_64-pc-linux-musl)
    x=$*: []
    x=$@: [ ]
  x="$*": [ ]
  x="$@": [ ]
GNU bash, version 5.0.18(1)-release (x86_64-pc-linux-musl)
    x=$*: [ ]
    x=$@: [ ]
  x="$*": [ ]
  x="$@": [ ]
GNU bash, version 5.1.8(1)-release (x86_64-pc-linux-musl)
    x=$*: [ ]
    x=$@: [ ]
  x="$*": [ ]
  x="$@": [ ]
```

Note that `x="$*"` also always works, but `x="$@"` is still preferred when space separation is desired, as it is not affected by `IFS`.

2.6. Internal development guidelines

This section provides guidelines for working on the SST Bash library itself.

2.7. Core functions

2.7.1. The `sst_nl` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_nl
```

The `sst_nl` function copies the content of standard input to standard output, appending a newline character to the end if not already present.

2.7.2. The `sst_csf` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_csf [<&var>]...
```

If no arguments are given, the `sst_csf` function copies the content of standard input to standard output, appending an "`x`" character and a newline character. Otherwise, the function removes the last character from each `<&var>` if it exists as an "`x`" character, and removes the next last character if it exists as a newline character.

The purpose of the `sst_csf` function is to allow command substitutions to be used that only remove a single trailing newline character instead of as many trailing newline characters as possible. To be more specific, instead of writing `foo=$(bar)`, you can write `foo=$(bar | sst_csf); sst_csf foo`.

The "csf" abbreviation in the function name is short for "**c**ommand **s**ubstitution **f**ix".

When input is said to be *read in* `sst_csf` *form*, it means that the input is taken to be all characters up to but not including a possible single trailing newline character.

When output is said to be *written in* `sst_csf` *form*, it means that the output is followed by a single trailing newline character.

2.7.3. The `sst_ihd` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_ihd [<n>]
```

The `sst_ihd` function copies the content of standard input to standard output, removing `<n>` space characters from the beginning of each line. Lines that do not begin with `<n>` space characters are left unchanged.

If all lines are empty, all lines are left unchanged. Otherwise, if `<n>` is omitted, it is taken to be the length k of the longest possible sequence of space characters at the beginning of the first nonempty line, which may be zero. Otherwise, if `<n>` is negative, it is adjusted to $\max(k + \texttt{<n>}, 0)$.

If the input is empty, the output will be empty. Otherwise, a trailing newline character will be appended to the last line of output if not already present.

The “ihd” abbreviation in the function name is short for “**i**ndented **h**ere-**d**ocument”.

2.7.4. The `sst_ihs` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_ihs [<n>]
```

The `sst_ihs` function is equivalent to the `sst_ihd` function except the first and last lines of input are ignored.

The “ihs” abbreviation in the function name is short for “**i**ndented **h**ere-**s**tring”.

2.7.5. The `sst_join` function

Bash

```
sst_join [<arg>...]
```

The `sst_join` function writes the space-separated concatenation of the arguments to standard output in `[bl_sst_csf]` form. If no arguments are given, the concatenation is the empty string.

2.7.6. The `sst_quote` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_quote [<argument>...]
```

The `sst_quote` function converts the input into a portably quoted shell word.

The algorithm is as follows:

1. Replace every ' character with '\''.
2. Prepend a ' character to the entire result.
3. Append a ' character to the entire result.

The input is the space-separated concatenation of the arguments. If no arguments are given, the input is read from standard input in CSF form instead. The output is always written to standard output in CSF form.

This function never requires the use of `sst_csf`, as the output never ends with a newline character (before it is written to standard output in CSF form). In other words, a plain command substitution always correctly collects the output.

2.7.7. The `sst_quote_list` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_quote_list [<argument>...]
```

The `sst_quote_list` function converts each `<argument>` to a portably quoted shell word and outputs the resulting list of words, separated by space characters. If no `<argument>`s are given, the output is empty.

Each conversion is performed using the same algorithm as the `sst_quote` function.

The output is written to standard output in CSF form.

This function never requires the use of `sst_csf`, as the output never ends with a newline character (before it is written to standard output in CSF form). In other words, a plain command substitution always correctly collects the output.

2.7.8. The `sst_smart_quote` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_smart_quote [<arg>]...
```

The `sst_smart_quote` function is equivalent to the sst_quote function except that if the result matches the regular expression `^'[+./0-9:=A-Z_a-z-]+'$`, the quotes are removed.

This function never requires the use of `sst_csf`, as the output never ends with a newline character (before it is written to standard output in CSF form). In other words, a plain command substitution always correctly collects the output.

2.7.9. The `sst_regex_escape` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_regex_escape [<arg>]...
```

The `sst_regex_escape` function escapes the input for use in a Bash regular expression.

The input is the space-separated concatenation of the arguments. If no arguments are given, the input is read from standard input in CSF form instead. The output is always written to standard output in CSF form.

This function never requires the use of `sst_csf`, as the output never ends with a newline character (before it is written to standard output in CSF form). In other words, a plain command substitution always correctly collects the output.

2.7.10. The `sst_regex_escape_list` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_regex_escape_list [<arg>]...
```

The `sst_regex_escape_list` function outputs a Bash regular expression that matches any of the arguments. If no arguments are given, the regular expression will never match. The regular expression is always parenthesized.

The output is written to standard output in CSF form.

This function never requires the use of `sst_csf`, as the output never ends with a newline character (before it is written to standard output in CSF form). In other words, a plain command substitution always correctly collects the output.

2.7.11. The `sst_barf` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_barf [<arg>]…
```

The `sst_barf` function prints an error message and a stack trace to standard error and terminates the script.

The error message consists of the string `"$0: Error: "`, followed by the space-separated concatenation of the arguments, followed by a newline character.

If the `SST_BARF_STATUS` variable is set to an integer between 0 and 255 inclusive, the exit status will be `$SST_BARF_STATUS`. Otherwise, if `sst_barf` was called implicitly by a failed command, the exit status will be that of the failed command. Otherwise, the exit status will be 1.

2.7.12. The `sst_warn` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_warn [<arg>]…
```

The `sst_warn` function prints a warning message to standard error.

The warning message consists of the string `"$0: warning: "`, followed by the space-separated concatenation of the arguments, followed by a newline character.

Example 5.

Script

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_warn "example message"
```

Possible output

```
./example: warning: example message
```

2.7.13. The `sst_info` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_info [<arg>]…
```

The `sst_info` function prints an informative message to standard output.

The informative message consists of the string `"$0: "`, followed by the space-separated concatenation of the arguments, followed by a newline character.

Example 6.

Script

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_info "example message"
```

Possible output

```
./example: example message
```

2.7.14. The `sst_unimplemented` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_unimplemented
```

2.7.15. The `sst_expect_not_subshell` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash
```

```
sst_expect_not_subshell [<function>]
```

The `sst_expect_not_subshell` function verifies that the current code is not running in a subshell[1]. If the verification fails, `sst_barf` will be called with an appropriate error message.

The error message will state that `<function>` must not be called in a subshell. If `<function>` is omitted, it is inferred as the name of the calling function. If `<function>` is a single "`-`" character, the error message will instead generically state that the current code must not be run in a subshell.

Example 7.

```
#! /bin/sh -
set -e || exit $?
. sst.bash

function foo {
  sst_expect_not_subshell "$@"
}

foo       # ok
(foo)     # error stating that foo must not be called in a subshell
(foo bar) # error stating that bar must not be called in a subshell
(foo -)   # error stating that no subshell was expected
```

2.7.16. The `sst_push_var` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_push_var <&var> [<value>]
```

The `sst_push_var` function saves the current value of `<&var>` onto a stack and sets `<&var>` to `<value>`. The most recent saved value of `<&var>` can be restored by calling `sst_pop_var`.

If `<&var>` is currently unset, its unset-ness will be saved onto the stack, and the corresponding call to `sst_pop_var` will unset `<&var>`.

If `<value>` is omitted, `<&var>` will be unset instead of being set to `<value>`.

2.7.17. The `sst_pop_var` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash
```

```
sst_pop_var <&var>
```

2.7.18. The `sst_set_exit` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_set_exit [<status>]
```

The `sst_set_exit` function takes no action and returns with exit status `<status>`. If `<status>` is omitted, it is taken to be the exit status of the last command executed, in which case the call is effectively a noop.

2.7.19. The `sst_expect_exit_status` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_expect_exit_status [<argument>]...
```

The `sst_expect_exit_status` function verifies that each `<argument>` is an exit status.

A string is an exit status if all of the following are true:

1. The string matches the regular expression `/^(0|[1-9][0-9]{0,2})$/`.
2. The integer value represented by the string is between 0 and 255.

2.8. Array functions

2.8.1. The `sst_array_cmp` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_array_cmp <array1> <array2>
```

The `sst_array_cmp` function compares two arrays. The arrays are compared in lexical order first on their keys, and then on their values. The result is `-1`, `0`, or `1`. The result is printed to standard

output in [bl_sst_csf] form.

WARNING This function currently does not work properly when either of the arrays is an associative array. This is planned to be fixed in the future.

2.8.2. The `sst_array_to_string` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_array_to_string <array>
```

The `sst_array_to_string` function converts an array to a string. The output is written to standard output in [bl_sst_csf] form.

2.8.3. The `sst_array_from_zterm` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_array_from_zterm <array> [<command> [<arg>]…]
```

The `sst_array_from_zterm` function parses a sequence of zero-terminated elements into `<array>`.

If `<command> [<arg>]…` is given, it is run with standard input as its standard input and the elements are parsed from its standard output. Otherwise, the elements are parsed from standard input.

If the element sequence is empty, the resulting array will be empty. Otherwise, the zero terminator of the last element is optional if the last element is nonempty.

WARNING Using `sst_array_from_zterm` in a pipeline is usually a mistake, as any shell is permitted to run any command of a pipeline in a subshell.[2]

For example, in `printf 'a\0' | sst_array_from_zterm xs`, if the `sst_array_from_zterm` command runs in a subshell, it will populate the `xs` array in the subshell and then exit the subshell, leaving the `xs` array in the current shell untouched. `sst_array_from_zterm xs printf 'a\0'`, on the other hand, will always work as intended.

Example 8.

```
sst_array_from_zterm xs printf ''       # xs=()
sst_array_from_zterm xs printf '\0'     # xs=('')
sst_array_from_zterm xs printf 'a'      # xs=('a')
```

```
sst_array_from_zterm xs printf 'a\0'    # xs=('a')
sst_array_from_zterm xs printf '\0\0'   # xs=('' '')
sst_array_from_zterm xs printf 'a\0\0'  # xs=('a' '')
sst_array_from_zterm xs printf 'a\0b'   # xs=('a' 'b')
sst_array_from_zterm xs printf 'a\0b\0' # xs=('a' 'b')
```

2.9. Path functions

2.9.1. The `sst_abs_dir` function

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_abs_dir [<path>]
```

The `sst_abs_dir` function outputs an absolute path to `<path>` that includes a trailing `/` character. `<path>` is not validated against the file system and need not exist.

If `<path>` is empty, `sst_barf` will be called.

The output is guaranteed to begin and end with a `/` character.

If `<path>` is not given, the input is read from standard input in CSF form. The output is always written to standard output in CSF form.

This function never requires the use of `sst_csf`, as the output never ends with a newline character (before it is written to standard output in CSF form). In other words, a plain command substitution always correctly collects the output.

This function is unaffected by `errexit` suspension. Errors will always cause the current shell to exit with a nonzero exit status.

This function is equivalent to calling `sst_add_slash` followed by `sst_abs_prefix` followed by `sst_squish_slashes`, except it may have improved performance.

2.9.2. The `sst_abs_file` function

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_abs_file [<path>]
```

The `sst_abs_file` function outputs an absolute path to `<path>`. `<path>` is not validated against the file system and need not exist.

`sst_barf` will be called if any of the following is true:

1. `<path>` is empty.
2. `<path>` is `..`
3. `<path>` is `...`
4. `<path>` ends with `/`.
5. `<path>` ends with `/..`
6. `<path>` ends with `/...`

The output is guaranteed to begin with a `/` character.

If `<path>` is not given, the input is read from standard input in CSF form. The output is always written to standard output in CSF form.

This function is unaffected by `errexit` suspension. Errors will always cause the current shell to exit with a nonzero exit status.

This function is equivalent to calling `sst_abs_dir` followed by `sst_trim_slashes`, except it may have improved performance and inputs that cannot possibly be file paths are rejected.

2.9.3. The `sst_abs_prefix` function

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_abs_prefix [<path>]
```

The `sst_abs_prefix` function constructs an absolute path to `<path>`, removes any characters after the last `/` character, and outputs the result. `<path>` is not validated against the file system and need not exist. If `<path>` ends with a `/` character, the behavior is as if it ended with `/.` instead.

If `<path>` is empty, `sst_barf` will be called.

The output is guaranteed to begin and end with a `/` character.

The output is guaranteed to be free of `.` and `..` components if any of the following is true:

1. `<path>` is free of `.` and `..` components.
2. `<path>` ends with `/.` but is otherwise free of `.` and `..` components.
3. `<path>` is a `.` character.

If `<path>` is not given, the input is read from standard input in CSF form. The output is always written to standard output in CSF form.

This function never requires the use of `sst_csf`, as the output never ends with a newline character (before it is written to standard output in CSF form). In other words, a plain command substitution always correctly collects the output.

This function is unaffected by `errexit` suspension. Errors will always cause the current shell to exit with a nonzero exit status.

2.9.4. The `sst_add_slash` function

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_add_slash [<path>]
```

The `sst_add_slash` function appends a `/` character to `<path>` if `<path>` ends with a non-`/` character.

If `<path>` is empty, `sst_barf` will be called.

If `<path>` is not given, the input is read from standard input in CSF form. The output is always written to standard output in CSF form.

This function never requires the use of `sst_csf`, as the output never ends with a newline character (before it is written to standard output in CSF form). In other words, a plain command substitution always correctly collects the output.

This function is unaffected by `errexit` suspension. Errors will always cause the current shell to exit with a nonzero exit status.

2.9.5. The `sst_add_slash_abs_prefix` function

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_add_slash_abs_prefix [<path>]
```

The `sst_add_slash_abs_prefix` function is equivalent to calling `sst_add_slash` followed by `sst_abs_prefix`, except it may have improved performance.

If `<path>` is not given, the input is read from standard input in CSF form. The output is always written to standard output in CSF form.

This function never requires the use of `sst_csf`, as the output never ends with a newline character (before it is written to standard output in CSF form). In other words, a plain command substitution always correctly collects the output.

This function is unaffected by `errexit` suspension. Errors will always cause the current shell to exit with a nonzero exit status.

2.9.6. The `sst_add_slash_dot_slash` function

```
#! /bin/sh -
```

```
set -e || exit $?
. sst.bash

sst_add_slash_dot_slash [<path>]
```

The `sst_add_slash_dot_slash` function is equivalent to calling `sst_add_slash` followed by `sst_dot_slash`, except it may have improved performance.

If `<path>` is not given, the input is read from standard input in CSF form. The output is always written to standard output in CSF form.

This function never requires the use of `sst_csf`, as the output never ends with a newline character (before it is written to standard output in CSF form). In other words, a plain command substitution always correctly collects the output.

This function is unaffected by `errexit` suspension. Errors will always cause the current shell to exit with a nonzero exit status.

2.9.7. The `sst_dot_slash` function

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_dot_slash [<path>]
```

The `sst_dot_slash` function prepends `./` to `<path>` if `<path>` does not begin with `/` or `./`.

If `<path>` is empty, `sst_barf` will be called.

If `<path>` is not given, the input is read from standard input in CSF form. The output is always written to standard output in CSF form.

This function is primarily used to adjust an arbitrary path to be safe to use as a program argument. Without adjustment, an unusual path like `--foo` may be mistakenly interpreted as an option instead of an operand. For example, `rm --foo` will fail, but `rm ./--foo` will work.

This function is unaffected by `errexit` suspension. Errors will always cause the current shell to exit with a nonzero exit status.

Example 9. Adjusting an arbitrary path

```
path=$(sst_dot_slash "$path" | [bl_sst_csf])
[bl_sst_csf] path
```

Example 10. Output examples

```
sst_dot_slash /                     # /\n
sst_dot_slash /foo                  # /foo\n
sst_dot_slash ''                    # \n
sst_dot_slash foo                   # ./foo\n
sst_dot_slash -foo                  # ./-foo\n
```

```
sst_dot_slash --foo                  # ./--foo\n
printf 'foo\n:' | sst_dot_slash      # ./foo\n:\n
printf 'foo\n:\n' | sst_dot_slash    # ./foo\n:\n
printf 'foo\n:\n\n' | sst_dot_slash  # ./foo\n:\n\n
```

2.9.8. The `sst_expect_prefix` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_expect_prefix [<path>]
```

The `sst_expect_prefix` function is equivalent to the `sst_get_prefix` function except it also verifies that the output is nonempty.

2.9.9. The `sst_get_prefix` function

Bash

```
sst_get_prefix [<path>]
```

The `sst_get_prefix` function gets the longest initial substring of `<path>` that ends in a `/` character. If `<path>` does not contain a `/` character, the result is the empty string.

If `<path>` is not given, the input is read from standard input in CSF form. The output is always written to standard output in CSF form.

This function never requires the use of `sst_csf`, as the output never ends with a newline character (before it is written to standard output in CSF form). In other words, a plain command substitution always correctly collects the output.

Example 11. Output examples

```
sst_get_prefix foo                   # \n
sst_get_prefix foo/                  # foo/\n
sst_get_prefix foo/bar               # foo/\n
sst_get_prefix foo/bar/              # foo/bar/\n
sst_get_prefix foo/bar/baz           # foo/bar/\n
```

2.9.10. The `sst_safe_dir` function

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_safe_dir [<path>]
```

The `sst_safe_dir` function outputs a path to `<path>` that includes a trailing `/` character. `<path>` is not validated against the file system and need not exist.

If `<path>` is empty, `sst_barf` will be called.

The output is guaranteed to begin with a `/` or `.` character and end with a `/` character.

If `<path>` is not given, the input is read from standard input in CSF form. The output is always written to standard output in CSF form.

This function never requires the use of `sst_csf`, as the output never ends with a newline character (before it is written to standard output in CSF form). In other words, a plain command substitution always correctly collects the output.

This function is unaffected by `errexit` suspension. Errors will always cause the current shell to exit with a nonzero exit status.

This function is equivalent to calling `sst_add_slash` followed by `sst_dot_slash` followed by `sst_squish_slashes`, except it may have improved performance.

2.9.11. The `sst_safe_file` function

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_safe_file [<path>]
```

The `sst_safe_file` function outputs an adjusted path to `<path>`. `<path>` is not validated against the file system and need not exist.

`sst_barf` will be called if any of the following is true:

1. `<path>` is empty.
2. `<path>` is `..`
3. `<path>` is `...`
4. `<path>` ends with `/`.
5. `<path>` ends with `/..`
6. `<path>` ends with `/...`

The output is guaranteed to begin with a `/` or `.` character.

If `<path>` is not given, the input is read from standard input in CSF form. The output is always written to standard output in CSF form.

This function is unaffected by `errexit` suspension. Errors will always cause the current shell to exit with a nonzero exit status.

This function is equivalent to calling `sst_safe_dir` followed by `sst_trim_slashes`, except it may have improved performance and inputs that cannot possibly be file paths are rejected.

2.9.12. The `sst_squish_slashes` function

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_squish_slashes [<path>]
```

The `sst_squish_slashes` function replaces each maximal sequence of `/` characters in `<path>` with a single `/` character, subject to the following exceptions:

1. If `<path>` is exactly `//`, the output will be exactly `//`.
2. If `<path>` begins with `//` followed by a non-`/` character, the initial `//` will be preserved.

If `<path>` is not given, the input is read from standard input in CSF form. The output is always written to standard output in CSF form.

This function is unaffected by `errexit` suspension. Errors will always cause the current shell to exit with a nonzero exit status.

2.9.13. The `sst_trim_slashes` function

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_trim_slashes [<path>]
```

The `sst_trim_slashes` function removes all trailing `/` characters from `<path>`, subject to the following exceptions:

1. If `<path>` is `/`, the output will be `/`.
2. If `<path>` is `//`, the output will be `//`.
3. If `<path>` begins with `///` and consists entirely of `/` characters, the output will be `/`.

If `<path>` is not given, the input is read from standard input in CSF form. The output is always written to standard output in CSF form.

This function is unaffected by `errexit` suspension. Errors will always cause the current shell to exit with a nonzero exit status.

2.10. JSON functions

2.10.1. The `sst_json_escape` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_json_escape [<text>]...
```

The `sst_json_escape` function escapes the input to be suitable to use in a JSON string.

The input is the space-separated concatenation of the arguments. If no arguments are given, the input is read from standard input in CSF form instead. The output is always written to standard output in CSF form.

The input must be encoded with UTF-8. If the input contains an invalid sequence of UTF-8 code units, `sst_barf` is called. The output is encoded with UTF-8 and always consists of code points between U+0020 and U+007E. Any code point outside this range is always represented as `\b`, `\f`, `\n`, `\r`, `\t`, `\uXXXX`, or a UTF-16 surrogate pair `\uXXXX\uXXXX`. It is unspecified which representation is used for any occurrence of any code point that can be represented more than one way, including even for code points between U+0020 and U+007E.

This function never requires the use of `sst_csf`, as the output never ends with a newline character (before it is written to standard output in CSF form). In other words, a plain command substitution always correctly collects the output.

2.10.2. The `sst_json_quote` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_json_quote [<text>]...
```

The `sst_json_quote` function quotes the input as a JSON string.

The input is the space-separated concatenation of the arguments. If no arguments are given, the input is read from standard input in CSF form instead. The output is always written to standard output in CSF form.

The input must be encoded with UTF-8. If the input contains an invalid sequence of UTF-8 code units, `sst_barf` is called. The output is encoded with UTF-8 and always consists of code points between U+0020 and U+007E. Any code point outside this range is always represented as `\b`, `\f`, `\n`, `\r`, `\t`, `\uXXXX`, or a UTF-16 surrogate pair `\uXXXX\uXXXX`. It is unspecified which representation is used for any occurrence of any code point that can be represented more than one way, including even for code points between U+0020 and U+007E.

This function never requires the use of `sst_csf`, as the output never ends with a newline character (before it is written to standard output in CSF form). In other words, a plain command

substitution always correctly collects the output.

2.10.3. The `sst_jq_expect` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_jq_expect <condition> [<format> [<arg>...]]
```

The `sst_jq_expect` function checks a condition on JSON input using the `jq` utility and terminates the script if the condition is false.

The JSON input is read from standard input.

The `<condition>` argument specifies the condition to check. It should be a `jq` filter that outputs `true` or `false`.

If the condition is true, no action is taken. Otherwise, the script is terminated by calling `[bl_sst_barf] [<format> [<arg>…]]`.

2.10.4. The `sst_jq_get_string` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_jq_get_string <json> [<filter>] <&var>
```

The `sst_jq_get_string` function uses the `jq` utility to parse a JSON string into a variable.

`<json>` specifies the JSON value on which to run `jq`. If `<json>` is a single `-` character, the JSON value is read from standard input. Otherwise, if `<json>` begins with a `/`, `A-Z`, `a-z`, `0-9`, `.`, `_`, or `-` character, it is taken to be a path to a file from which to read the JSON value. Otherwise, `<json>` is taken literally as the JSON value. If the input is actually a sequence of JSON values, all but the first JSON value is ignored.

`<filter>` specifies the `jq` filter to run on `<json>`. The filter should produce either a null value or a string value. A null value will be implicitly adjusted to an empty string value. If `<filter>` is omitted, it is taken to be `.<&var>`.

The resulting string value is assigned to `<&var>`.

2.10.5. The `sst_jq_get_strings` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_jq_get_strings <json> <filter> <var>
```

The `sst_jq_get_strings` function parses an array of JSON strings into an indexed array.

`<json>` specifies the JSON value on which to run `jq`. If `<json>` is a single `-` character, the JSON value is read from standard input. Otherwise, if `<json>` begins with a `/`, `A-Z`, `a-z`, `0-9`, `.`, `_`, or `-` character, it is taken to be a path to a file from which to read the JSON value. Otherwise, `<json>` is taken literally as the JSON value. If the input is actually a sequence of JSON values, all but the first JSON value is ignored.

`<filter>` specifies the `jq` filter to run on `<json>`. The filter should produce either a single null value, a single string value, or an array of null and string values. A single value is implicitly adjusted to an array containing that value, an empty array is implicitly adjusted to an array containing an empty string value, and each null value is implicitly adjusted to an empty string value.

The resulting JSON array is parsed into the variable named `<var>` as an indexed array.

2.11. Command-line argument parsing

2.11.1. The `sst_parse_opt` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_parse_opt [<prefix=>] (<option>... | <matcher>())
              [<style=required>] (: "$@" | :<&array>[:<i=0>])
```

The `sst_parse_opt` function attempts to parse an option from the beginning of a list of command-line arguments.

When `sst_parse_opt` returns, the following variables will have been updated in the calling context:

`<prefix>got`

A boolean value indicating whether an option was parsed.

`<prefix>opt`

The option that was parsed, or unset if an option was not parsed.

`<prefix>arg`

The option argument that was parsed, or unset if an option argument was not parsed.

`<prefix>pop`

A code fragment that you should execute with `eval` immediately after the `sst_parse_opt` call returns. If an option was parsed, the code fragment will remove the option from the list of command-line arguments so that `sst_parse_opt` may be called again to attempt to parse another option. Otherwise, the code fragment will do nothing.

2.11.2. The `sst_unknown_opt` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_unknown_opt <arg>
```

2.12. Unit testing

2.12.1. The `sst_test` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_test [<status=0>[(:|=)<output=>]] <code> [<checks> [<show>]...]
```

2.13. Build system functions

2.13.1. The `build-dist-archive.bash` script

```
build-dist-archive.bash
  [--build-depends=<package>]...
```

The `build-dist-archive.bash` script builds the distribution archive of an Autotools project.

`--build-depends=<package>`

Specifies that `<package>` should be installed before running `./configure`. This option can be specified multiple times to install multiple packages.

2.13.2. The `sst_ag_call_defun_once_macros` function

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_ag_call_defun_once_macros
```

2.13.3. The `sst_ag_define_ordering_macros` function

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_ag_define_ordering_macros
```

The `sst_ag_define_ordering_macros` function generates every nonexistent macro y implied by the existence of a `GATBPS_BEFORE([$0], [`y`])` call. Each y is called an *ordering macro*.

Consider the set of all pairs (x, y) implied by the existence of any `GATBPS_BEFORE([$0], [`y`])` call in any macro x defined in `m4/`x`.m4`. For each unique y for which `m4/`y`.m4` does not exist, `m4/`y`.m4` will be generated, defining y to call `GATBPS_REQUIRE([`x`])` for every corresponding x.

2.13.4. The `sst_ag_include` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_ag_include [<file>...]
```

2.13.5. The `sst_ag_process_leaf` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_ag_process_leaf <target> <leaf> <&child>
```

2.13.6. The `sst_ac_include` function

Bash

```
#! /bin/sh -
```

```
set -e || exit $?
. sst.bash

sst_ac_include [<file>...]
```

2.13.7. The `sst_ac_config_file` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_ac_config_file [<file>...]
```

2.13.8. The `sst_am_include` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_am_include [<file>...]
```

2.13.9. The `sst_am_distribute` function

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_am_distribute [<path>]...
```

The `sst_am_distribute` function adds each `<path>` to the distributed files list.

`<path>` must either exist as a file, exist as a directory, or not exist. A repeated `<path>` will be skipped, even across multiple calls.

This function must not be called from a subshell.

2.13.10. The `sst_am_distribute_if_not_dir` function

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_am_distribute_if_not_dir [<path>]...
```

The `sst_am_distribute_if_not_dir` function adds each `<path>` to the distributed files list, but skips `<path>` if it exists as a directory.

`<path>` must either exist as a file, exist as a directory, or not exist. A repeated `<path>` will be skipped, even across multiple calls.

This function must not be called from a subshell.

2.13.11. The `sst_am_var_set` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_am_var_set <var> [<arg>]...
```

2.13.12. The `sst_am_var_set_const` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_am_var_set_const <var> [<arg>]...
```

2.13.13. The `sst_am_var_add` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_am_var_add <var> [<arg>]...
```

2.13.14. The `sst_am_var_add_unique_word` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_am_var_add_unique_word <var> [<word>...]
```

2.13.15. The `sst_am_var_add_unique_file` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_am_var_add_unique_file <var> [<file>...]
```

2.13.16. The `sst_am_if` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_am_if [!]<var>
```

The `sst_am_if` function starts an `if` block for the `AM_CONDITIONAL` variable `<var>`. If the `!` character is given, the condition will be negated. The block must be ended by a matching call to `sst_am_elif`, `sst_am_else`, or `sst_am_endif`.

2.13.17. The `sst_am_elif` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_am_elif <var> [<expected_comment>]
```

2.13.18. The `sst_am_else` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_am_else [<expected_comment>]
```

2.13.19. The `sst_am_endif` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_am_endif [<expected_comment>]
```

The `sst_am_endif` function ends the `if` block started by the matching call to `sst_am_if`, `sst_am_elif`, or `sst_am_else`. If `<expected_comment>` is given, it will be verified to match the condition of the block being ended.

2.14. Miscellaneous

2.14.1. The `sst_libdir` variable

The `sst_libdir` variable provides an absolute path to the directory that contains the SST Bash library. The path is guaranteed to begin with a / character and end with a non-/ character.

Example 12.

Script

```
#! /bin/sh -
set -e || exit $?
. sst.bash

printf '%s\n' "$sst_libdir"
```

Possible output

```
/usr/local/share/sst/bash
```

2.14.2. The `sst_isOatty` variable

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_isOatty=<boolean>
```

The `sst_isOatty` variable indicates whether standard input is connected to a terminal.

`sst_isOatty=<boolean>` means that `sst_isOatty` is set to either `1` or null to indicate true or false, respectively. Because arithmetic evaluation implicitly converts null variables to `0`, this means that `sst_isOatty` can be compared to true with either `[[ "$sst_isOatty" ]]` or `((sst_isOatty))`, and compared to false with either `[[ ! "$sst_isOatty" ]]` or

`((!sst_isOatty))`.

Example 13.

```
docker run --rm -i ${sst_isOatty:+-t} ubuntu echo hello world
```

2.14.3. The `sst_underscore_slug` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_underscore_slug [<arg>]▮
```

The `sst_underscore_slug` function replaces every non-alphanumeric character of the input with an underscore character.

The input is the space-separated concatenation of the arguments. If no arguments are given, the input is read from standard input in CSF form instead. The output is always written to standard output in CSF form.

Example 14.

```
sst_underscore_slug foo bar       # foo_bar
sst_underscore_slug lib/libfoo.la # lib_libfoo_la
```

2.14.4. The `sst_extract_archive` function

Bash

```
sst_extract_archive <file>
```

The `sst_extract_archive` function extracts the archive `<file>` into the current directory. The supported file extensions are `.tar`, `.tar.gz`, `.tar.xz`, and `.zip`.

2.14.5. The `sst_get_distro` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_get_distro
```

The `sst_get_distro` function identifies the system type and stores it in the `sst_distro` variable.

The following results are possible: `alpine`, `arch`, `centos`, `cygwin`, `debian`, `fedora`, `homebrew`,

`macports`, `msys2`, `rhel`, `ubuntu`, or `unknown`.

The first time this function is called, it determines the result and stores it in the `sst_distro` variable, and subsequent calls simply output the cached result.

All possible results, including those added in the future, are guaranteed to be spelled such that `$sst_distro` is unaffected by word splitting and pathname expansion.

2.14.6. The `sst_get_distro_version` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_get_distro_version
```

The `sst_get_distro_version` function identifies the system version and stores it in the `sst_distro_version` variable.

The following results are possible for the various `sst_get_distro` values:

`centos`

`6`, `7`, `8`, or `unknown`.

`ubuntu`

`16.04`, `18.04`, `20.04`, or `unknown`.

`unknown`

`unknown`.

The first time this function is called, it determines the result and stores it in the `sst_distro_version` variable, and subsequent calls simply output the cached result.

All possible results, including those added in the future, are guaranteed to be spelled such that `$sst_distro_version` is unaffected by word splitting and pathname expansion.

2.14.7. The `prepare-dist-repo.bash` script

Bash

```
prepare-dist-repo.bash <json>
```

The `prepare-dist-repo.bash` script makes a commit to a private [bs_distribution_repository] for preview purposes before the `[bl_publish_dist_repo_bash]` script is used to publish the preview commit to a corresponding public distribution repository.

`<json>` specifies the JSON value on which to run `jq`. If `<json>` is a single `-` character, the JSON value is read from standard input. Otherwise, if `<json>` begins with a `/`, `A-Z`, `a-z`, `0-9`, `.`, `_`, or `-` character, it is taken to be a path to a file from which to read the JSON value. Otherwise, `<json>` is taken literally as the JSON value. If the input is actually a sequence of JSON values, all but the first JSON value is ignored.

The following arguments may be specified in `<json>`:

`pull_repo`

> The URL of the repository to clone and make the preview commit in. The commit will be made on top of the commit specified by `pull_commit`.
>
> The URL will be expanded as if it appeared inside a double-quoted Bash string, allowing environment variables to be used. For example, a URL such as `https://gitlab-ci-token:${CI_JOB_TOKEN-}@gitlab.stealthsoftwareinc.com/stealth/sst.git` could be used to clone a repository while running in a GitLab CI job.
>
> More than one URL may be specified by providing an array of strings instead of a single string. The array will act as a list of mirrors from which to try clone the repository. The mirrors are attempted to be cloned in order, stopping after the first successful clone that contains `pull_commit`.

`pull_commit` (optional)

> The commit of `pull_repo` on top of which to make the preview commit. By default, `pull_commit` is the default branch of `pull_repo`.

`push_repo`

> A repository URL or an array of repository URLs to which to push the preview commit. The preview commit will always be pushed to a new branch `preview-<x>`, where `<x>` is derived from the source of the CI job. This argument is optional and defaults to `pull_repo`.

`update_script` (optional)

> A path to an update script to run instead of the default update script.

`post_update_script` (optional)

> A path to a post-update script to run instead of the default post-update script. The default post-update script does nothing.

`commit_script` (optional)

> A path to a commit script to run instead of the default commit script.

`post_commit_script` (optional)

> A path to a post-commit script to run instead of the default post-commit script. The default post-commit script does nothing.

`gitbundle`

: The name of the `.gitbundle` file to produce as an artifact to use as input to the `[bl_publish_dist_repo_bash]` script. This argument is optional and defaults to `preview.gitbundle`.

`dist_archive`

: A path to an archive to use instead of the default distribution archive, which is located using the `[bl_sst_find_dist_archive]` function. This argument may also be set to "`git-archive`" to create an ephemeral distribution archive that contains the content of `HEAD` inside a container directory with an unspecified name. This argument is optional.

`tag_transform`

: An optional `sed` script to use to transform the tag name when preparing from a tag.

2.14.8. The `publish-dist-repo.bash` script

Bash

```
publish-dist-repo.bash <json>
```

The `publish-dist-repo.bash` script publishes a preview commit made by the `[bl_prepare_dist_repo_bash]` script to a corresponding public [bs_distribution_repository].

2.14.9. The `sst_find_dist_archive` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_find_dist_archive
```

This function never requires the use of `sst_csf`, as the output never ends with a newline character (before it is written to standard output in CSF form). In other words, a plain command substitution always correctly collects the output.

2.14.10. The `sst_find_dist_date` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_find_dist_date
```

2.14.11. The `sst_find_dist_version` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_find_dist_version
```

2.14.12. The `sst_type` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_type [<arg>]
```

The `sst_type` function calls `type "$@"` and adjusts an exit status of 1 to 0.

Script

```
! /bin/sh -
set -e || exit $?
. sst.bash

g() {

  local x

  sst_expect_argument_count $ 1

  x=$(sst_type -p "$1")
  readonly x

  if [[ "$x" ]]; then
    sst_info "You have $1."
  else
    sst_info "You do not have $1."
  fi

}; readonly -f g

f() {

  g awk
  g sed
  g nonexistent-program

}; f "$@"
```

Possible output

```
a.bash: You have awk.
a.bash: You have sed.
a.bash: You do not have nonexistent-program.
```

2.14.13. The `sst_grep` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_grep [<arg>...]
```

The `sst_grep` function calls `grep "$@"` and adjusts an exit status of 1 to 0.

2.14.14. The `sst_curl_slurp` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_curl_slurp <url> [<arg>]...
```

2.14.15. The `sst_expect_utf8` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_expect_utf8 [<arg>]...
```

The `sst_expect_utf8` function validates that the input is valid UTF-8 and outputs the input unchanged.

The input is the space-separated concatenation of the arguments. If no arguments are given, the input is read from standard input in CSF form instead. The output is always written to standard output in CSF form.

2.14.16. The `sst_get_variables` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_get_variables [<regex>...]
```

The `sst_get_variables` function outputs all existing variable names that match any of the Bash regular expressions `<regex>`.

If more than one variable name matches, the names are separated by newline characters and the order of the names is unspecified. If no variable names match, the output is empty.

The output is written to standard output in CSF form.

2.14.17. The `sst_get_environment_variables` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_get_environment_variables [<regex>...]
```

The `sst_get_environment_variables` function outputs all existing environment variable names that match any of the Bash regular expressions `<regex>`.

If more than one environment variable name matches, the names are separated by newline characters and the order of the names is unspecified. If no environment variable names match, the output is empty.

The output is written to standard output in CSF form.

Example 15.

```
$ cat a.bash
#! /bin/sh -
set -e || exit $?
. sst.bash
sst_get_environment_variables "$@"

$ ./a.bash PWD
OLDPWD
PWD

$ ./a.bash '^PWD'
PWD

$ ./a.bash ALL
LC_ALL
```

```
$ ./a.bash ALL PWD
LC_ALL
OLDPWD
PWD

$ PARALLEL=1 ./a.bash ALL
LC_ALL
PARALLEL

$ ./a.bash && echo Note the blank line from CSF form.

Note the blank line from CSF form.
```

2.14.18. The `sst_human_list` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_human_list [<adjust>...] : [<item>...]
```

The `sst_human_list` function adjusts and outputs a list of items in human-readable form.

1. If `<adjust>...` is given, each `<item>` is adjusted by running and collecting the output of `<adjust>... <item>`.
2. If no `<item>`s are given, the output is "`none`".
3. If exactly one `<item>` is given, the output is that `<item>`.
4. If exactly two `<item>`s are given, the output is those two `<item>`s with " `and` " inbetween.
5. If three or more `<item>`s are given, the output is all `<item>`s separated by "`, `", and with "`and` " immediately preceding the last `<item>`.

The output is designed to appear after a `:` character in normal prose.

The output is written to standard output in CSF form.

Example 16.

```
$ cat a.bash
#! /bin/sh -
set -e || exit $?
. sst.bash
sst_human_list "$@"

$ ./a.bash :
none
```

```
$ ./a.bash : foo
foo

$ ./a.bash : foo bar
foo and bar

$ ./a.bash : foo bar baz
foo, bar, and baz

$ ./a.bash : 'foo bar baz'
foo bar baz

$ ./a.bash sst_quote :
none

$ ./a.bash sst_quote : foo
'foo'

$ ./a.bash sst_quote : foo bar
'foo' and 'bar'

$ ./a.bash sst_quote : foo bar baz
'foo', 'bar', and 'baz'

$ ./a.bash sst_quote : 'foo bar baz'
'foo bar baz'
```

2.14.19. The `sst_kill_all_jobs` function

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_kill_all_jobs [<arg>...]
```

The `sst_kill_all_jobs` function runs `kill [<arg>...] %j` for every currently existing background job `j`.

If all of the `kill` commands succeed, the function will return with exit status 0. Otherwise, the function will return with the exit status of the first failed `kill` command. The rest of the `kill` commands will still be run, but their exit statuses will be ignored.

Other errors are possible but extremely rare, and will cause the function to immediately return with the offending exit status. This differs from the usual fail-fast convention of the library, as the main purpose of this function is for best-effort cleanup. Because subshells do not inherit background jobs, the usual error handling technique of using a subshell does not work, i.e., `(sst_kill_all_jobs &>/dev/null) || :` will never find any background jobs to kill.

Note that the `kill` builtin may exit with exit status 0 in certain cases you may not expect it to. For example, on Bash 5.0, `(sleep 2 & kill foo %1 || echo bar)` will kill the `sleep` command and print an error message about `foo` being unrecognized, but it will not print `bar`.

Example 17.

sst-example-17.bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

# Kill all background jobs when this script exits.
sst_trap_append 'sst_kill_all_jobs &>/dev/null || :' EXIT

# Start a background job that runs for 30 seconds.
sleep 30 &

# After this script exits, you can run ps to verify that the sleep
# command isn't running. Then you can edit this script to remove the
# sst_kill_all_jobs command and try running it again. This time, you
# should see the sleep command in ps.
```

2.14.20. The `sst_copyright_notice` function

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_copyright_notice <comment1> [<comment2> <comment3>] [<prose>]
```

The `sst_copyright_notice` function formats a copyright notice for use in source code.

The prose of the copyright notice is specified by `<prose>` as follows:

1. If `<prose>` is omitted, it will be taken to be the prose of the SST copyright notice, which is as follows:

   ```
   Copyright (C) Stealth Software Technologies, Inc.

   For the complete copyright information, please see the
   associated README file.
   ```

2. Otherwise, if `<prose>` is `-`, it will be read from standard input in CSF form.

3. Otherwise, if `<prose>` begins with `/` or `./`, it will be read from the indicated file in CSF form.

4. Otherwise, `<prose>` will be taken directly in CSF form.

The comment structure of the copyright notice is specified by `<comment1>`, `<comment2>`, and `<comment3>` as follows:

1. If `<comment2>` and `<comment3>` are omitted, they are taken to be the same as `<comment1>`.

2. Each line of `<prose>` will be prefixed with `<comment2>` followed by a space character.

3. `<comment1>` followed by a newline character will be prepended to the beginning of the

copyright notice.

4. A newline character followed by `<comment3>` will be appended to the end of the copyright notice.

Example 18.

`sst-example-18.bash`

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_copyright_notice "$@"
```

Example terminal session

```
$ ./sst-example-18.bash //
//
// Copyright (C) Stealth Software Technologies, Inc.
//
// For the complete copyright information, please see the
// associated README file.
//

$ ./sst-example-18.bash // Example
//
// Example
//

$ ./sst-example-18.bash '/*' ' *' ' */' Example
/*
 * Example
 */
```

2.14.21. `sst_is_errexit_suspended`

Bash

```
sst_is_errexit_suspended
```

The `sst_is_errexit_suspended` function outputs `1` if `errexit` is suspended, `0` if `errexit` is not suspended, or nothing if an error occurs.

The output is written to standard output in CSF form. However, if an error occurs, the output may be completely empty instead of a single newline character.

Example 19.

`sst-example-19.bash`

```
#! /bin/sh -
```

```
set -e || exit $?
. sst.bash

sst_is_errexit_suspended                        # Outputs 0.
while sst_is_errexit_suspended; do break; done # Outputs 1.
until sst_is_errexit_suspended; do break; done # Outputs 1.
if sst_is_errexit_suspended; then :; fi        # Outputs 1.
sst_is_errexit_suspended && :                  # Outputs 1.
sst_is_errexit_suspended || :                  # Outputs 1.
! sst_is_errexit_suspended                     # Outputs 1.

exit 0
```

2.14.22. `sst_expect_errexit`

Bash

```
sst_expect_errexit
```

2.15. Filesystem utilities

2.15.1. The `sst_expect_any_file` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_expect_any_file [<path>]...
```

The `sst_expect_any_file` function verifies that at least one `<path>` exists as a file.

2.15.2. The `sst_expect_extension` function

Bash

```
sst_expect_extension <path> <extension>
```

The `sst_expect_extension` function verifies that the final component of `<path>` ends with `<extension>`, but is not solely `<extension>`.

Technically speaking, this function verifies that `<extension>` is a trailing substring of `<path>` that is preceded by at least one non-slash character.

2.15.3. The `sst_expect_file` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_expect_file [<path>]...
```

The `sst_expect_file` function verifies that each `<path>` exists as a file.

2.15.4. The `sst_expect_maybe_file` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_expect_maybe_file [<path>]...
```

The `sst_expect_maybe_file` function verifies that each `<path>` either exists as a file or does not exist.

2.15.5. The `sst_expect_not_exist` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_expect_not_exist [<path>...]
```

The `sst_expect_not_exist` function verifies that each `<path>` does not exist. If any verification fails, `sst_barf` will be called with an appropriate error message.

2.15.6. `sst_mkdir_p_new`

Bash

```
sst_mkdir_p_new [<path>...]
```

The `sst_mkdir_p_new` function iterates through the `<path>` parameters, taking the following actions for each `<path>`:

1. Create all parent directories of `<path>`. If a parent directory already exists as a directory, it is skipped. If a parent directory already exists as a non-directory, a fatal error occurs.

2. Create `<path>` as a directory. If `<path>` already exists, a fatal error occurs.

If no <path> parameters are given, no action is taken.

2.15.7. sst_mkdir_p_only

Bash

```
sst_mkdir_p_only [<path>...]
```

The sst_mkdir_p_only function iterates through the <path> parameters, taking the following actions for each <path>:

1. Create all parent directories of <path>. If a parent directory already exists as a directory, it is skipped. If a parent directory already exists as a non-directory, a fatal error occurs.

If no <path> parameters are given, no action is taken.

This function behaves correctly under errexit suspension.

Example 20.

```
sst_mkdir_p_only         # No action taken
sst_mkdir_p_only foo     # Same as mkdir -p .
sst_mkdir_p_only foo/bar # Same as mkdir -p foo
```

2.15.8. The sst_popd function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_popd [<arg>...]
```

The sst_popd function is equivalent to the popd builtin except it does not write anything to standard output.

This function is unaffected by errexit suspension. Errors will always cause the current shell to exit with a nonzero exit status.

2.15.9. The sst_pushd function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_pushd [<arg>...]
```

The `sst_pushd` function is equivalent to the `pushd` builtin except it does not write anything to standard output.

This function is unaffected by `errexit` suspension. Errors will always cause the current shell to exit with a nonzero exit status.

2.15.10. `sst_stmpdir`

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

readonly sst_stmpdir=d
```

The `sst_stmpdir` variable provides an absolute path to a directory that can be used to store sensitive temporary files. The path is guaranteed to begin with a "/" character and end with a non-"/" character. When the script exits, all files below the directory will be overwritten with zeros and the directory will be deleted.

2.15.11. `sst_tmpdir`

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

readonly sst_tmpdir=d
```

The `sst_tmpdir` variable provides an absolute path to a directory that can be used to store non-sensitive temporary files. The path is guaranteed to begin with a "/" character and end with a non-"/" character. When the script exits, the directory will be deleted.

2.16. Autogen JSON handlers

The `sst_ajh_build_tree_program_wrapper` function

```
sst_ajh_build_tree_program_wrapper [ag_json...]
```

2.16.1. The `sst_ajh_asciidoctor_document` function

Bash

```
sst_ajh_asciidoctor_document [<json>...]
```

2.16.1.1. Images

`asciidoctor` will be called with `-a imagesdir=images`, meaning that `image::foo.png[]` will reference `images/foo.png`, not `foo.png`. So, to use images, create an `images` directory, put your images in it, and reference them accordingly.

You can create subdirectories under the `images` directory to help organize your images. For example, if you create `images/foo/bar.png`, you can reference it with `image::foo/bar.png[]`.

Asciidoctor Diagram will output images to the `images` directory using the name pattern `diag-*`. You should avoid this name pattern for your own images, as the build system sometimes deletes all such images.

You can change the name of the `images` directory by setting `imagesdir` in `<json>`.

2.16.2. The `sst_ajh_java_library` function

Bash

```
#! /bin/sh -
set -e || exit $?
. sst.bash

sst_ajh_java_library [<json>]
```

`jardeps`

An optional array of `.jar` files that `dst` depends on. The following types of entries are supported:

1. A `.jar` file name. For example, you might list `netty-all-4.1.65.Final.jar`. The `.jar` file will be expected to already exist in `javadir`, `/usr/local/share/java`, or `/usr/share/java` at build time and after installation. If the `.jar` file exists in more than one of these directories, the first one will be preferred.

2. A makefile target for a `.jar` file. For example, you might list `build-aux/downloads/netty-all-4.1.65.Final.jar`. If an entry contains a `/` character, it is always assumed to be this type of entry. Otherwise, the entry is distinguished against a type 1 entry by the existence of a corresponding `.ag`, `.ag.json`, `.ac`, or `.am` file. This type of entry behaves the same as a type 1 entry containing only the `.jar` file name, except that additional code is generated for building and installing the `.jar` file to the system (see below).

3. An `.im` substitution that expands to a `.jar` file path (not merely a name). For example, you might list `{@}PATH_NETTY{@}`. This type of entry behaves the same as a type 1 entry except the path is determined at configure time instead of being semi-hardcoded to the previously described preference list.

4. An `.im` substitution that expands to a `.jar` file path, followed by a `:` character, followed by a type 1 or type 2 entry. For example, you might list `{@}PATH_NETTY{@}:build-aux/downloads/netty-all-4.1.65.Final.jar`. This combines the effects of a type 3 entry with a type 1 or 2 entry, with the type 3 entry being preferred. You'd want to use

this if you want to detect an already-installed newer version of a library at configure time, falling back to a known-good version if a newer version isn't found. When followed by a type 2 entry, the generated installation code will only run if the `.jar` file path does not exist as a file, i.e., the installation of the fallback will not occur if a newer version is already installed.

Several installation targets are generated to install any makefile target `.jar` files from entry types 2 and 4:

`install-java-$(slug)-jardeps-targets`

Makes all makefile target `.jar` files for `dst`.

`install-java-$(slug)-jardeps`

Installs all makefile target `.jar` files for `dst`.

`install-java-jardeps-targets`

Makes all makefile target `.jar` files every `dst`.

`install-java-jardeps`

Installs all makefile target `.jar` files for every `dst`.

Do not use `jardeps` to list `.jar` dependencies that are built by this project. Use `built_jardeps` instead.

`built_jardeps`

An optional array of `.jar` makefile targets that `dst` depends on that are built by this project.

3. C/C++ library

3.1. Reserved identifiers

Unless explicitly stated otherwise in this manual, the following rules must be followed in any file that directly or indirectly includes any SST header file:

1. Do not define the identifier `SST` or any identifiers that begin with `SST_` anywhere, not even in local contexts.
2. Do not define the identifier `sst` or any identifiers that begin with `sst_` at file scope or as macros.
3. Do not define any identifiers anywhere in the `::sst` namespace.

3.2. General development guidelines

This section provides guidelines for using the SST C/C++ library and writing C/C++ code in general.

3.3. Internal development guidelines

This section provides guidelines for working on the SST C/C++ library itself.

3.3.1. Header file extensions

The *public* header files of the SST C/C++ library are those that live below the `src/c-cpp/include/sst` directory in the SST Git repository, but excluding those that live below the `src/c-cpp/include/sst/private` directory.

All public SST header files use the `.h` file extension except for those that are intended to be used in C++ only, which use the `.hpp` file extension instead.

Sometimes you might find two copies of the same public SST header file alongside each other: one with the `.h` file extension, and one with the `.hpp` file extension. In this case, including the `.hpp` header file in C++ is equivalent to including the `.h` header file, but the `.hpp` header file is deprecated and will eventually be removed.

If a header file is intended to be used in C++ only, it should include the following code after its include guard:

C++ only

```
#ifndef __cplusplus
#error "This header file cannot be used in C"
#include "This header file cannot be used in C"
#endif
```

If a header file is intended to be used in C only, it should include the following code after its include guard:

C only

```
#ifdef __cplusplus
#error "This header file cannot be used in C++"
#include "This header file cannot be used in C++"
#endif
```

3.4. Creating a self-contained subset library

SST includes a script that can create a *self-contained subset library* of the C/C++ library.

Synopsis

```
src/c-cpp/subset.bash [<option>]... [<entity>]...
```

`subset.bash` will produce an archive file `sst.tar.gz` that contains all the necessary code for the given list of entities. Each `<entity>` should be the name of a C/C++ entity, such as `sst_getbit` or `sst::bignum`.

`subset.bash` must be run from the root of the SST project, i.e., the same way as written in the above synopsis. It will refuse to run if this is not the case.

For convenience, if the build system is not already initialized and configured, `subset.bash` will automatically run `./autogen` and `./configure` as needed.

The output archive file will contain a directory named `sst` that contains two subdirectories named `include` and `lib`. `sst/include` contains all the necessary public header files, and `sst/lib` contains all the necessary private implementation files.

The `sst/include/sst/config.h` file usually needs to be edited, as it will be populated with values deduced from the current system, which may not be correct for the ultimate destination of the subset library. You'll also need to remember to compile with `-lfoo` for the libraries you choose to enable in `config.h`.

SST freely makes use of the standard threading utilities of C/C++, so it's reasonably likely you'll also need to compile with `-pthread` (or equivalent for your compiler). If you fail to do so, the compilation can unfortunately sometimes still succeed and result in strange errors at run time, such as calls to `std::call_once` throwing exceptions.

`--namespace=<namespace>`

> Renames the `sst` C++ namespace to `<namespace>`. The `SST_` and `sst_` prefixes used for macros and C functions, the output archive file name, and all other names will also be renamed accordingly. It is permitted for `<namespace>` to have multiple components.
>
> As an example, `--namespace=foo` will rename the C++ namespace to `foo`, the `SST_` and `sst_` prefixes to `FOO_` and `foo_`, and the output archive file to `foo.tar.gz`. As another example, `--namespace=foo::bar` will rename the C++ namespace to `foo::bar`, the `SST_` and `sst_` prefixes to `FOO_BAR_` and `foo_bar_`, and the output archive file to `foo_bar.tar.gz`.

Example 21.

> The following commands will clone the SST distribution repository and produce a self-contained subset library of the C/C++ library that contains `sst::bignum`:
>
> ```
> git clone https://github.com/stealthsoftwareinc/sst.git
> cd sst
> src/c-cpp/subset.bash sst::bignum
> ```
>
> This will produce `sst.tar.gz`, which can be used in any other project. Here's how it could be used to compile a simple test program:
>
> ```
> $ tar xzf sst.tar.gz
>
> $ cat <<'EOF' >a.cpp
> #include <iostream>
> #include <sst/catalog/bignum.hpp>
> int main() {
>   std::cout << (sst::bignum(1) << 128).to_string() << "\n";
> ```

```
}
EOF

$ g++ -Isst/include -Isst/lib a.cpp $(find sst/lib -name '.c') -lcrypto

$ ./a.out
340282366920938463463374607431768211456
```

3.5. Algorithm categories

3.5.1. Unique-pass algorithms

An algorithm with an iterator parameter `p` is called *unique-pass* with respect to `p` if the algorithm satisfies all of the following properties:

1. For any two arithmetic operations A and B performed on two iterators derived from `p` (i.e., any of `++q`, `q++`, `--q`, `q--`, `q+n`, `n+q`, `q+=n`, `q-n`, or `q-=n` where `q` is derived from `p`), either A is sequenced before B or B is sequenced before A. In other words, all arithmetic operations on iterators derived from `p` are separated by sequence points.

2. Each time an arithmetic operation A is performed on an iterator derived from `p`, the next arithmetic operation is performed on the result of A or a copy of the result, even if A is trivial (e.g., `q+0`). This induces a unique chain of iterators derived from `p` that begins with `p` itself, where each successive iterator is the result of the next arithmetic operation on (a copy of) the previous iterator.

3. For any two dereference operations A and B performed on (copies of) two iterators in the chain, either A is sequenced before B or B is sequenced before A. In other words, all dereferences of (copies of) two iterators in the chain are separated by sequence points.

4. Each time an iterator derived from `p` is dereferenced, the iterator is either the newest iterator or the second newest iterator in the chain. Furthermore, if it is the second newest iterator, then the newest iterator has not yet been dereferenced. This permits `*p++` and `*p--` but forbids sequences such as `q=p++,*p,*q`.

3.6. Attributes

3.6.1. The `SST_MAYBE_UNUSED` attribute

C or C++

```
#include <sst/catalog/SST_MAYBE_UNUSED.h>

#define SST_MAYBE_UNUSED(...) /*...*/
```

3.6.2. The `SST_NODISCARD` attribute

C or C++

```
#include <sst/catalog/SST_NODISCARD.h>

#define SST_NODISCARD(...) /*...*/
```

3.6.3. The SST_NORETURN attribute

C or C++

```
#include <sst/catalog/SST_NORETURN.h>

#define SST_NORETURN(...) /*...*/
```

Depending on the language version being used, the SST_NORETURN macro expands to either nothing or an attribute that specifies that a function never returns.

Note that SST requires C++11, which supports the standard [[noreturn]] attribute, so SST_NORETURN is not necessary when using C++. However, for consistency, it is provided anyway.

Example 22. Using SST_NORETURN in C

sst-example-22.c

```
#include <stdlib.h>

#include <sst/catalog/SST_NORETURN.h>

SST_NORETURN() static void f() {
  abort();
}

int main(void) {
  f();
  return 0; /* unreachable */
}
```

Example 23. Using SST_NORETURN in C++

sst-example-23.cpp

```
#include <cstdlib>

#include <sst/catalog/SST_NORETURN.h>

namespace {

SST_NORETURN() void f() {
  std::abort();
}

}
```

```
int main() {
  f();
  return 0; // unreachable
}
```

3.7. Debugging utilities

3.7.1. SST_ASSERT (C) (1)

```
#include <sst/catalog/SST_ASSERT.h>

#define SST_ASSERT(CONDITION) (x)
```

If SST_DEBUG is nonzero, the SST_ASSERT (C) (1) macro expands to a parenthesized expression (x) of type void that evaluates the expression CONDITION exactly once. The following actions are then taken:

1. If the result of CONDITION compares unequal to 0, a message that indicates the assertion failed and includes the stringification of CONDITION is written to standard error, and abort is called.

2. Otherwise, no action is taken.

Any message written to standard error will also include the values of __FILE__ and __LINE__ from the location of the SST_ASSERT (C) (1) call.

If SST_DEBUG is zero, the SST_ASSERT (C) (1) macro expands to a parenthesized expression (x) of type void that takes no action. It does not evaluate CONDITION, except if the result of CONDITION is a variable length array.

3.7.2. SST_ASSERT (C) (2)

```
#include <sst/catalog/SST_ASSERT.h>

#define SST_ASSERT(CONDITION, MESSAGE) (x)
```

If SST_DEBUG is nonzero, the SST_ASSERT (C) (2) macro expands to a parenthesized expression (x) of type void that evaluates the expression CONDITION exactly once. The following actions are then taken:

1. If the result of CONDITION compares unequal to 0, the expression MESSAGE is evaluated exactly once, a message that indicates the assertion failed and includes MESSAGE is written to standard error, and abort is called. The result of MESSAGE must be implicitly convertible to char const *. If the result of MESSAGE is a null pointer, it is adjusted to be the stringification of CONDITION instead.

2. Otherwise, MESSAGE is not evaluated and no action is taken.

Any message written to standard error will also include the values of __FILE__ and __LINE__ from

the location of the SST_ASSERT (C) (2) call.

If SST_DEBUG is zero, the SST_ASSERT (C) (2) macro expands to a parenthesized expression (x) of type void that takes no action. It does not evaluate CONDITION, except if the result of CONDITION is a variable length array, and it does not evaluate MESSAGE.

3.7.3. SST_ASSERT (C++) (1)

```
#include <sst/catalog/SST_ASSERT.hpp>

#define SST_ASSERT(CONDITION) (x)
```

If SST_DEBUG is nonzero, the SST_ASSERT (C++) (1) macro expands to a parenthesized expression (x) of type void that evaluates the expression CONDITION exactly once. The following actions are then taken:

1. If CONDITION throws an exception e, a message that indicates the assertion was interrupted by an exception and includes e.what() is written to standard error, and std::abort is called.

2. Otherwise, the result of CONDITION is explicitly converted to bool. If the result is false, a message that indicates the assertion failed and includes the stringification of CONDITION is written to standard error, and std::abort is called.

3. Otherwise, no action is taken.

Any message written to standard error will also include the values of __FILE__ and __LINE__ from the location of the SST_ASSERT (C++) (1) call.

If SST_DEBUG is zero, the SST_ASSERT (C++) (1) macro expands to a parenthesized expression (x) of type void that takes no action. It does not evaluate CONDITION.

3.7.4. SST_ASSERT (C++) (2)

```
#include <sst/catalog/SST_ASSERT.hpp>

#define SST_ASSERT(CONDITION, MESSAGE) (x)
```

If SST_DEBUG is nonzero, the SST_ASSERT (C++) (2) macro expands to a parenthesized expression (x) of type void that evaluates the expression CONDITION exactly once. The following actions are then taken:

1. If CONDITION throws an exception e, the expression MESSAGE is not evaluated, a message that indicates the assertion was interrupted by an exception and includes e.what() is written to standard error, and std::abort is called.

2. Otherwise, the result of CONDITION is explicitly converted to bool. If the result is false, the expression MESSAGE is evaluated exactly once, a message that indicates the assertion failed and includes MESSAGE is written to standard error, and std::abort is called. The result of MESSAGE must be implicitly convertible to either char const * or std::string const &. If the result of MESSAGE is a null pointer, it is adjusted to be the stringification of CONDITION instead.

3. Otherwise, `MESSAGE` is not evaluated and no action is taken.

Any message written to standard error will also include the values of `__FILE__` and `__LINE__` from the location of the `SST_ASSERT` (C++) (2) call.

If `SST_DEBUG` is zero, the `SST_ASSERT` (C++) (2) macro expands to a parenthesized expression `(x)` of type `void` that takes no action. It does not evaluate `CONDITION`, and it does not evaluate `MESSAGE`.

3.7.5. SST_DEBUG

C or C++

```
#include <sst/catalog/SST_DEBUG.h>

#define SST_DEBUG (x)
```

The `SST_DEBUG` macro expands to a parenthesized integer constant expression `(x)` with type `int`. The expression is also suitable for use in the preprocessor.

3.8. Memory utilities

3.8.1. The `sst::overlap` function

```
#include <sst/catalog/overlap.hpp>
namespace sst {

template<class T1, std::integral N1, class T2, std::integral N2>
bool overlap(T1 const * p1, N1 n1, T2 const * p2, N2 n2);

template<class T1, std::integral N1, class T2>
bool overlap(T1 const * p1, N1 n1, T2 const * p2);

template<class T1, class T2, std::integral N2>
bool overlap(T1 const * p1, T2 const * p2, N2 n2);

template<class T1, class T2>
bool overlap(T1 const * p1, T2 const * p2);

}
```

The `sst::overlap` function attempts to determine whether two memory regions overlap.

Each memory region consists of the `n`i elements of type `T`i beginning at `p`i. `n`i must be nonnegative and `p`i must be a valid pointer, even if `n`i is zero. If `n`i is omitted, it is taken to be `1`. `T`i must be an object type or `void`. If `T`i is `void`, the behavior is as if it were `unsigned char`.

If this function returns `true`, the two memory regions overlap.

If this function returns `false`, the two memory regions may or may not overlap. False negatives are generally possible because some systems allow multiple memory regions to refer to the same underlying data. For example, a system may allow the same file to be mapped into two different memory regions at the same time, with any changes being reflected in both regions simultaneously. If no such functionality is being used, false negatives are not possible.

If `n1` or `n2` is zero, this function always returns `false`.

3.8.2. The `sst::pointer_eq` function

```
#include <sst/catalog/pointer_eq.hpp>
namespace sst {

template<class T1, class T2>
bool pointer_eq(T1 const * p1, T2 const * p2);

}
```

The `sst::pointer_eq` function returns `true` if `p1` is equal to `p2` with respect to the implementation-defined strict total order over pointers.

Each of `T1` and `T2` must be either `void` or an object type.

3.8.3. The `sst::pointer_ge` function

```
#include <sst/catalog/pointer_ge.hpp>
namespace sst {

template<class T1, class T2>
bool pointer_ge(T1 const * p1, T2 const * p2);

}
```

The `sst::pointer_ge` function returns `true` if `p1` is greater than or equal to `p2` with respect to the implementation-defined strict total order over pointers.

Each of `T1` and `T2` must be either `void` or an object type.

3.8.4. The `sst::pointer_gt` function

```
#include <sst/catalog/pointer_gt.hpp>
namespace sst {

template<class T1, class T2>
bool pointer_gt(T1 const * p1, T2 const * p2);

}
```

The `sst::pointer_gt` function returns `true` if `p1` is greater than `p2` with respect to the implementation-defined strict total order over pointers.

Each of `T1` and `T2` must be either `void` or an object type.

3.8.5. The `sst::pointer_le` function

```
#include <sst/catalog/pointer_le.hpp>
namespace sst {

template<class T1, class T2>
bool pointer_le(T1 const * p1, T2 const * p2);

}
```

The `sst::pointer_le` function returns `true` if `p1` is less than or equal to `p2` with respect to the implementation-defined strict total order over pointers.

Each of `T1` and `T2` must be either `void` or an object type.

3.8.6. The `sst::pointer_lt` function

```
#include <sst/catalog/pointer_lt.hpp>
namespace sst {

template<class T1, class T2>
bool pointer_lt(T1 const * p1, T2 const * p2);

}
```

The `sst::pointer_lt` function returns `true` if `p1` is less than `p2` with respect to the implementation-defined strict total order over pointers.

Each of `T1` and `T2` must be either `void` or an object type.

3.8.7. The `sst::pointer_ne` function

```
#include <sst/catalog/pointer_ne.hpp>
namespace sst {

template<class T1, class T2>
bool pointer_ne(T1 const * p1, T2 const * p2);

}
```

The `sst::pointer_ne` function returns `true` if `p1` is not equal to `p2` with respect to the implementation-defined strict total order over pointers.

Each of `T1` and `T2` must be either `void` or an object type.

3.9. Language shims

3.9.1. The SST_C_VALUE macros

C only

```
#include <sst/SST_C_VALUE.h>
// or:   <sst/language.h>

#define SST_C99_VALUE 199901L
#define SST_C11_VALUE 201112L
#define SST_C17_VALUE 201710L
#define SST_C18_VALUE SST_C17_VALUE
```

3.9.2. The SST_C_OR_LATER macros

C++

```
#include <sst/SST_C_OR_LATER.h>
// or:   <sst/language.h>

#define SST_C99_OR_LATER (__STDC_VERSION__ >= SST_C99_VALUE)
#define SST_C11_OR_LATER (__STDC_VERSION__ >= SST_C11_VALUE)
#define SST_C17_OR_LATER (__STDC_VERSION__ >= SST_C17_VALUE)
#define SST_C18_OR_LATER (__STDC_VERSION__ >= SST_C18_VALUE)
```

3.9.3. The SST_CPP_VALUE macros

C++

```
#include <sst/SST_CPP_VALUE.h>
// or:   <sst/language.h>

#define SST_CPP97_VALUE 199711L
#define SST_CPP03_VALUE SST_CPP97_VALUE
#define SST_CPP11_VALUE 201103L
#define SST_CPP14_VALUE 201402L
#define SST_CPP17_VALUE 201703L
#define SST_CPP20_VALUE 202002L
```

3.9.4. The SST_CPP_OR_LATER macros

C++

```
#include <sst/SST_CPP_OR_LATER.h>
// or:   <sst/language.h>

#define SST_CPP97_OR_LATER (__cplusplus >= SST_CPP97_VALUE)
#define SST_CPP03_OR_LATER (__cplusplus >= SST_CPP03_VALUE)
#define SST_CPP11_OR_LATER (__cplusplus >= SST_CPP11_VALUE)
#define SST_CPP14_OR_LATER (__cplusplus >= SST_CPP14_VALUE)
```

```
#define SST_CPP17_OR_LATER (__cplusplus >= SST_CPP17_VALUE)
#define SST_CPP20_OR_LATER (__cplusplus >= SST_CPP20_VALUE)
```

3.9.5. The SST_STATIC_ASSERT macro

C and C++

```
#include <sst/language.h>

#define SST_STATIC_ASSERT(constant_expression) ...
```

3.9.6. The SST_CPP_CONSTEXPR macros

C++

```
#include <sst/SST_CPP_CONSTEXPR.hpp>
// or:   <sst/language.h>

#if SST_CPP14_OR_LATER
  #define SST_CPP14_CONSTEXPR constexpr
#else
  #define SST_CPP14_CONSTEXPR
#endif

#if SST_CPP17_OR_LATER
  #define SST_CPP17_CONSTEXPR constexpr
#else
  #define SST_CPP17_CONSTEXPR
#endif

#if SST_CPP20_OR_LATER
  #define SST_CPP20_CONSTEXPR constexpr
#else
  #define SST_CPP20_CONSTEXPR
#endif
```

The SST_CPP14_CONSTEXPR macro expands to the constexpr keyword if C++14 or later is being used, or to nothing if not.

The SST_CPP17_CONSTEXPR macro expands to the constexpr keyword if C++17 or later is being used, or to nothing if not.

The SST_CPP20_CONSTEXPR macro expands to the constexpr keyword if C++20 or later is being used, or to nothing if not.

3.9.7. The SST_CPP_INLINE macros

C++

```
#include <sst/catalog/SST_CPP17_INLINE.hpp>
```

```
// or:   <sst/language.h>

#if SST_CPP17_OR_LATER
  #define SST_CPP17_INLINE inline
#else
  #define SST_CPP17_INLINE
#endif
```

The `SST_CPP17_INLINE` macro expands to the `inline` keyword if C++17 or later is being used, or to nothing if not.

3.9.8. The `SST_CONSTEXPR_ASSERT` macro

C++

```
#include <sst/catalog/SST_CONSTEXPR_ASSERT.hpp>
// or:   <sst/language.h>

#if SST_CPP14_OR_LATER
  #define SST_CONSTEXPR_ASSERT(x) assert(x)
#else
  #define SST_CONSTEXPR_ASSERT(x)
#endif
```

The `SST_CONSTEXPR_ASSERT` macro can be used to run dynamic assertions in `constexpr` functions.

3.9.9. The `SST_EXTERN_C` macro

C

```
#include <sst/catalog/SST_EXTERN_C.h>

#ifdef __cplusplus
#define SST_EXTERN_C extern "C"
#else
#define SST_EXTERN_C extern
#endif
```

3.10. Type traits

3.10.1. The `sst::bool_constant` alias

```
#include <sst/catalog/bool_constant.hpp>
namespace sst {

template<bool B>
using bool_constant = std::integral_constant<bool, B>;
```

```
}
```

The `sst::bool_constant` alias provides a convenient way to write `std::integral_constant<bool, B>`.

The `sst::bool_constant` alias was added to the library because the `std::bool_constant` alias is not available below C++17.

3.10.2. `sst::common_type_t`

C++

```
#include <sst/catalog/common_type_t.hpp>
namespace sst {

template<class... T>
using common_type_t =
    typename std::common_type<T...>::type;

}
```

The `sst::common_type_t` alias provides a convenient way to write `std::common_type<T...>::type` that never requires the `typename` keyword to be used to disambiguate a dependent type.

The `sst::common_type_t` alias was added to the library because the `std::common_type_t` alias is not available below C++14.

Example 24.

sst-example-24.cpp

```
#include <type_traits>

#include <sst/catalog/common_type_t.hpp>

template<class... T>
class foo {

  // OK.
  using type1 = typename std::common_type<T...>::type;

  // Compilation error in C++11: the typename keyword is required.
  using type2 = std::common_type<T...>::type;

  // OK.
  using type3 = sst::common_type_t<T...>;

};

int main() {
  return 0;
```

```
}
```

Compilation command

```
g++ sst-example-24.cpp -lsst
```

3.10.3. sst::conditional_t

C++

```
#include <sst/catalog/conditional_t.hpp>
namespace sst {

template<bool B, class T, class F>
using conditional_t =
    typename std::conditional<B, T, F>::type;

}
```

The `sst::conditional_t` alias provides a convenient way to write `std::conditional<B, T, F>::type` that never requires the `typename` keyword to be used to disambiguate a dependent type.

The `sst::conditional_t` alias was added to the library because the `std::conditional_t` alias is not available below C++14.

Example 25.

sst-example-25.cpp

```
#include <type_traits>

#include <sst/catalog/conditional_t.hpp>

template<bool B, class T, class F>
class foo {

  // OK.
  using type1 = typename std::conditional<B, T, F>::type;

  // Compilation error in C++11: the typename keyword is required.
  using type2 = std::conditional<B, T, F>::type;

  // OK.
  using type3 = sst::conditional_t<B, T, F>;

};

int main() {
  return 0;
}
```

Compilation command

```
g++ sst-example-25.cpp -lsst
```

3.10.4. The `sst::copy_cv` trait

C++

```
#include <sst/catalog/copy_cv.hpp>
namespace sst {

template<class Src, class Dst>
struct copy_cv
    : sst::type_identity<R> {};

}
```

The `sst::copy_cv` trait defines `R` to be `Dst` overwritten with the cv-qualifiers of `Src`. In other words, the following steps take place:

1. Begin with `R = Dst`.
2. Update `R` by removing any cv-qualifiers.
3. Update `R` by adding any cv-qualifiers from `Src`.

3.10.5. `sst::copy_cv_t`

C++

```
#include <sst/catalog/copy_cv_t.hpp>
namespace sst {

template<class Src, class Dst>
using copy_cv_t =
    typename sst::copy_cv<Src, Dst>::type;

}
```

The `sst::copy_cv_t` alias provides a convenient way to write `sst::copy_cv<Src, Dst>::type` that never requires the `typename` keyword to be used to disambiguate a dependent type.

Example 26.

`sst-example-26.cpp`

```
#include <sst/catalog/copy_cv.hpp>
#include <sst/catalog/copy_cv_t.hpp>

template<class Src, class Dst>
class foo {
```

```
  // OK.
  using type1 = typename sst::copy_cv<Src, Dst>::type;

  // Compilation error in C++11: the typename keyword is required.
  using type2 = sst::copy_cv<Src, Dst>::type;

  // OK.
  using type3 = sst::copy_cv_t<Src, Dst>;

};

int main() {
  return 0;
}
```

Compilation command

```
g++ sst-example-26.cpp -lsst
```

3.10.6. `sst::decay_t`

C++

```
#include <sst/catalog/decay_t.hpp>
namespace sst {

template<class T>
using decay_t =
    typename std::decay<T>::type;

}
```

The `sst::decay_t` alias provides a convenient way to write `std::decay<T>::type` that never requires the `typename` keyword to be used to disambiguate a dependent type.

The `sst::decay_t` alias was added to the library because the `std::decay_t` alias is not available below C++14.

Example 27.

sst-example-27.cpp

```
#include <type_traits>

#include <sst/catalog/decay_t.hpp>

template<class T>
class foo {

  // OK.
  using type1 = typename std::decay<T>::type;
```

```
  // Compilation error in C++11: the typename keyword is required.
  using type2 = std::decay<T>::type;

  // OK.
  using type3 = sst::decay_t<T>;

};

int main() {
  return 0;
}
```

Compilation command

```
g++ sst-example-27.cpp -lsst
```

3.10.7. The `sst::is_arithmetic` trait

```
#include <sst/catalog/is_arithmetic.hpp>
namespace sst {

template<class T>
struct is_arithmetic
    : sst::bool_constant<B> {};

}
```

3.10.8. The `sst::is_arithmetic_ish` trait

```
#include <sst/catalog/is_arithmetic_ish.hpp>
namespace sst {

template<class T>
struct is_arithmetic_ish
    : sst::bool_constant<B> {};

}
```

3.10.9. The `sst::is_arithmetic_like` trait

```
#include <sst/catalog/is_arithmetic_like.hpp>
namespace sst {

template<class T>
struct is_arithmetic_like
    : sst::bool_constant<B> {};

}
```

3.10.10. The `sst::is_big_endian` trait

C++

```
#include <sst/catalog/is_big_endian.hpp>
namespace sst {

template<class T>
struct is_big_endian : std::integral_constant<bool, b>;

}
```

The `sst::is_big_endian` trait determines whether the byte ordering of `T` is big endian.

Any cv-qualifiers on `T` are ignored.

The `sst::is_big_endian` trait may produce false negatives.

IMPORTANT Since `sst::is_big_endian` is publicly derived from `std::integral_constant<bool, b>`, there are two ways to extract the underlying value `b`. The first way is to write `sst::is_big_endian<T>::value`. This directly accesses the underlying value and always works as expected. The second way is to write `sst::is_big_endian<T>()`. This constructs an instance of `sst::is_big_endian`, which will be implicitly converted to the underlying value in most contexts, but not all. It is recommended to always use the first way.

3.10.11. The `sst::is_bool` type trait

C++

```
#include <sst/catalog/is_bool.hpp>
namespace sst {

template<class T>
struct sst::is_bool
    : std::integral_constant<bool, see below> {};

}
```

The `sst::is_bool` type trait determines whether `T` is the type `bool`, ignoring any references and cv-qualifiers.

IMPORTANT It is recommended to always extract the value of `sst::is_bool<T>` by writing `sst::is_bool<T>::value`. This always behaves as expected and ensures compatibility with C++11.

3.10.12. `sst::is_byte`

C++

```
#include <sst/catalog/is_byte.hpp>
```

```
namespace sst {

template<class T>
struct is_byte
    : std::integral_constant<bool, v> {};

}
```

The `sst::is_byte` trait determines whether `T` is one of the following types, ignoring any reference and cv-qualifiers:

- `char`
- `signed char`
- `unsigned char`
- `std::byte`

3.10.13. `sst::is_byte_input_iterable`

C++

```
#include <sst/catalog/is_byte_input_iterable.hpp>
namespace sst {

template<class C>
struct is_byte_input_iterable
    : std::integral_constant<bool, v> {};

}
```

3.10.14. `sst::is_byte_input_iterator`

C++

```
#include <sst/catalog/is_byte_input_iterator.hpp>
namespace sst {

template<class I>
struct is_byte_input_iterator
    : std::integral_constant<bool, v> {};

}
```

3.10.15. The `sst::is_enum` type trait

C++

```
#include <sst/catalog/is_enum.hpp>
```

```
namespace sst {

template<class T>
struct sst::is_enum
    : std::integral_constant<bool, see below> {};

}
```

The `sst::is_enum` type trait determines whether `T` is an enumeration type, ignoring any references and cv-qualifiers.

IMPORTANT It is recommended to always extract the value of `sst::is_enum<T>` by writing `sst::is_enum<T>::value`. This always behaves as expected and ensures compatibility with C++11.

3.10.16. The `sst::is_exact_width_integer` trait

C++

```
#include <sst/catalog/is_exact_width_integer.hpp>
namespace sst {

template<class T>
struct is_exact_width_integer : std::integral_constant<bool, b>;

}
```

The `sst::is_exact_width_integer` trait determines whether `T` is one of the exact-width integer types, i.e., one of `int8_t`, `uint8_t`, `int16_t`, `uint16_t`, `int32_t`, `uint32_t`, `int64_t`, or `uint64_t`, ignoring any cv-qualifiers.

IMPORTANT Since `sst::is_exact_width_integer` is publicly derived from `std::integral_constant<bool, b>`, there are two ways to extract the underlying value `b`. The first way is to write `sst::is_exact_width_integer<T>::value`. This directly accesses the underlying value and always works as expected. The second way is to write `sst::is_exact_width_integer<T>()`. This constructs an instance of `sst::is_exact_width_integer`, which will be implicitly converted to the underlying value in most contexts, but not all. It is recommended to always use the first way.

3.10.17. The `sst::is_floating` trait

```
#include <sst/catalog/is_floating.hpp>
namespace sst {

template<class T>
struct is_floating
    : sst::bool_constant<B> {};

}
```

3.10.18. The `sst::is_floating_ish` trait

```
#include <sst/catalog/is_floating_ish.hpp>
namespace sst {

template<class T>
struct is_floating_ish
    : sst::bool_constant<B> {};

}
```

3.10.19. The `sst::is_floating_like` trait

```
#include <sst/catalog/is_floating_like.hpp>
namespace sst {

template<class T>
struct is_floating_like
    : sst::bool_constant<B> {};

}
```

3.10.20. The `sst::is_input_iterator` trait

```
#include <sst/catalog/is_input_iterator.hpp>
namespace sst {

template<class I,
         template<class...> class heheT = sst::dependent_true,
         class... Args>
struct is_input_iterator
    : sst::bool_constant<b> {};

}
```

Let `i` be an object of type `sst::decay_t<I>`. The `sst::is_input_iterator` trait defines `b` to `true` if all of the following are true:

1. `sst::is_iterator<I>::value` is `true`.
2. `*i` is an lvalue reference to an object type.
3. `T<sst::decay_t<decltype(*i)>, Args...>::value` is `true`.

3.10.21. The `sst::is_integer` trait

```
#include <sst/catalog/is_integer.hpp>
namespace sst {
```

```
template<class T>
struct is_integer
    : sst::bool_constant<B> {};

}
```

The `sst::is_integer` trait defines `B` to `true` if `T` is a fundamental integer type[3], ignoring any cvref-qualifiers.

3.10.22. The `sst::is_integer_ish` trait

```
#include <sst/catalog/is_integer_ish.hpp>
namespace sst {

template<class T>
struct is_integer_ish
    : sst::bool_constant<B> {};

}
```

3.10.23. The `sst::is_integer_like` trait

```
#include <sst/catalog/is_integer_like.hpp>
namespace sst {

template<class T>
struct is_integer_like
    : sst::bool_constant<B> {};

}
```

3.10.24. The `sst::is_iterator` trait

```
#include <sst/catalog/is_iterator.hpp>
namespace sst {

template<class I>
struct is_iterator
    : sst::bool_constant<b> {};

}
```

Let `i` be an object of type `sst::decay_t<I>`. The `sst::is_iterator` trait defines `b` to `true` if all of the following are true:

1. `*i` compiles.
2. `++i` has type `sst::decay_t<I> &`.

3.10.25. The `sst::is_little_endian` trait

C++

```
#include <sst/catalog/is_little_endian.hpp>
namespace sst {

template<class T>
struct is_little_endian : std::integral_constant<bool, b>;

}
```

The `sst::is_little_endian` trait determines whether the byte ordering of `T` is little endian.

Any cv-qualifiers on `T` are ignored.

The `sst::is_little_endian` trait may produce false negatives.

IMPORTANT Since `sst::is_little_endian` is publicly derived from `std::integral_constant<bool, b>`, there are two ways to extract the underlying value `b`. The first way is to write `sst::is_little_endian<T>::value`. This directly accesses the underlying value and always works as expected. The second way is to write `sst::is_little_endian<T>()`. This constructs an instance of `sst::is_little_endian`, which will be implicitly converted to the underlying value in most contexts, but not all. It is recommended to always use the first way.

3.10.26. The `sst::is_non_bool_integer` trait

C++

```
#include <sst/type.h>
namespace sst {

template<class T>
struct is_non_bool_integer {
  constexpr bool value = /* ... */;
};

}
```

The `sst::is_non_bool_integer` class defines `value` to be `true` if `T` is a possibly cv-qualified integer type other than `bool`, or `false` if not.

3.10.27. The `sst::is_ones_complement` trait

C++

```
#include <sst/type.h>
namespace sst {
```

```
template<class T>
struct is_ones_complement : std::integral_constant<bool, b>;

}
```

The `sst::is_ones_complement` class determines whether `T` is an integer type that uses ones' complement representation.

In C++20 and later, signed integer types are required to use two's complement representation[4]. Before C++20, signed integer types are permitted to use ones' complement or sign-magnitude representation instead, in which case negative zero may be a trap representation[5].

Unsigned integer types are never considered to use ones' complement representation.

IMPORTANT Since `sst::is_ones_complement` is publicly derived from `std::integral_constant<bool, b>`, there are two ways to extract the underlying value `b`. The first way is to write `sst::is_ones_complement<T>::value`. This directly accesses the underlying value and always works as expected. The second way is to write `sst::is_ones_complement<T>()`. This constructs an instance of `sst::is_ones_complement`, which will be implicitly converted to the underlying value in most contexts, but not all. It is recommended to always use the first way.

3.10.28. The `sst::is_output_iterator` trait

```
#include <sst/catalog/is_output_iterator.hpp>
namespace sst {

template<class I, class T>
struct is_output_iterator
    : sst::bool_constant<b> {};

}
```

Let `i` be an object of type `sst::decay_t<I>`. The `sst::is_output_iterator` trait defines `b` to `true` if all of the following are true:

1. `sst::is_iterator<I>::value` is `true`.
2. `*i = std::forward<T>(std::declval<T>())` compiles.

3.10.29. `sst::is_sentinel`

C++

```
#include <sst/catalog/is_sentinel.hpp>
namespace sst {

template<class S, class I>
struct is_sentinel
    : std::integral_constant<bool, v> {};
```

```
}
```

3.10.30. `sst::is_value_sentinel`

C++

```
#include <sst/catalog/is_value_sentinel.hpp>
namespace sst {

template<class S, class I>
struct is_value_sentinel
    : std::integral_constant<bool, v> {};

}
```

3.10.31. The `sst::is_sign_magnitude` trait

C++

```
#include <sst/type.h>
namespace sst {

template<class T>
struct is_sign_magnitude : std::integral_constant<bool, b>;

}
```

The `sst::is_sign_magnitude` class determines whether `T` is an integer type that uses sign-magnitude representation.

In C++20 and later, signed integer types are required to use two's complement representation[4]. Before C++20, signed integer types are permitted to use ones' complement or sign-magnitude representation instead, in which case negative zero may be a trap representation[5].

Unsigned integer types are never considered to use sign-magnitude representation.

IMPORTANT Since `sst::is_sign_magnitude` is publicly derived from `std::integral_constant<bool, b>`, there are two ways to extract the underlying value `b`. The first way is to write `sst::is_sign_magnitude<T>::value`. This directly accesses the underlying value and always works as expected. The second way is to write `sst::is_sign_magnitude<T>()`. This constructs an instance of `sst::is_sign_magnitude`, which will be implicitly converted to the underlying value in most contexts, but not all. It is recommended to always use the first way.

3.10.32. The `sst::is_signed_integer` trait

C++

```
#include <sst/catalog/is_signed_integer.hpp>
// or:   <sst/type.h>
namespace sst {

template<class T>
struct is_signed_integer : std::integral_constant<bool, b>;

}
```

The `sst::is_signed_integer` trait determines whether `T` is a signed fundamental integer type[3].

IMPORTANT Since `sst::is_signed_integer` is publicly derived from `std::integral_constant<bool, b>`, there are two ways to extract the underlying value `b`. The first way is to write `sst::is_signed_integer<T>::value`. This directly accesses the underlying value and always works as expected. The second way is to write `sst::is_signed_integer<T>()`. This constructs an instance of `sst::is_signed_integer`, which will be implicitly converted to the underlying value in most contexts, but not all. It is recommended to always use the first way.

3.10.33. The `sst::is_twos_complement` trait

C++

```
#include <sst/type.h>
namespace sst {

template<class T>
struct is_twos_complement : std::integral_constant<bool, b>;

}
```

The `sst::is_twos_complement` class determines whether `T` is an integer type that uses two's complement representation.

In C++20 and later, signed integer types are required to use two's complement representation[4]. Before C++20, signed integer types are permitted to use ones' complement or sign-magnitude representation instead, in which case negative zero may be a trap representation[5].

Unsigned integer types are always considered to use two's complement representation.

IMPORTANT Since `sst::is_twos_complement` is publicly derived from `std::integral_constant<bool, b>`, there are two ways to extract the underlying value `b`. The first way is to write `sst::is_twos_complement<T>::value`. This directly accesses the underlying value and always works as expected. The second way is to write `sst::is_twos_complement<T>()`. This constructs an instance of `sst::is_twos_complement`, which will be implicitly converted to the underlying value in most contexts, but not all. It is recommended to always use the first way.

3.10.34. The `sst::is_unsigned_integer` trait

C++

```
#include <sst/type.h>
namespace sst {

template<class T>
struct is_unsigned_integer {
  constexpr bool value = /* ... */;
};

}
```

The `sst::is_unsigned_integer` class defines `value` to be `true` if `T` is a possibly cv-qualified unsigned integer type, or `false` if not.

For more information, see `std::is_unsigned` and `std::is_integral`.

3.10.35. `sst::remove_cv_t`

C++

```
#include <sst/catalog/remove_cv_t.hpp>
namespace sst {

template<class T>
using remove_cv_t =
    typename std::remove_cv<T>::type;

}
```

The `sst::remove_cv_t` alias provides a convenient way to write `std::remove_cv<T>::type` that never requires the `typename` keyword to be used to disambiguate a dependent type.

The `sst::remove_cv_t` alias was added to the library because the `std::remove_cv_t` alias is not available below C++14.

Example 28.

sst-example-28.cpp

```
#include <type_traits>

#include <sst/catalog/remove_cv_t.hpp>

template<class T>
class foo {

  // OK.
  using type1 = typename std::remove_cv<T>::type;
```

```
  // Compilation error in C++11: the typename keyword is required.
  using type2 = std::remove_cv<T>::type;

  // OK.
  using type3 = sst::remove_cv_t<T>;

};

int main() {
  return 0;
}
```

Compilation command

```
g++ sst-example-28.cpp -lsst
```

3.10.36. The `sst::remove_cvref` trait

C++

```
#include <sst/catalog/remove_cvref.hpp>
namespace sst {

template<class T>
struct remove_cvref
    : sst::type_identity<R> {};

}
```

The `sst::remove_cvref` trait defines `R` to be `T` with any reference and cv-qualifiers removed. In other words, the following steps take place:

1. Begin with `R = T`.
2. If `R` is a reference type, update `R` to be the referenced type.
3. Update `R` by removing any topmost cv-qualifiers.

The `sst::remove_cvref` trait was added to the library because the `std::remove_cvref` trait is not available below C++20.

Example 29.

```
sst::remove_cvref<int>::type          // int
sst::remove_cvref<int &>::type        // int
sst::remove_cvref<int const>::type    // int
sst::remove_cvref<int const &>::type  // int const
```

3.10.37. `sst::remove_cvref_t`

C++

```
#include <sst/catalog/remove_cvref_t.hpp>
namespace sst {

template<class T>
using remove_cvref_t =
    typename std::remove_cvref<T>::type;

}
```

The `sst::remove_cvref_t` alias provides a convenient way to write `std::remove_cvref<T>::type` that never requires the `typename` keyword to be used to disambiguate a dependent type.

Example 30.

`sst-example-30.cpp`

```
#include <sst/catalog/remove_cvref.hpp>
#include <sst/catalog/remove_cvref_t.hpp>

template<class T>
class foo {

  // OK.
  using type1 = typename std::remove_cvref<T>::type;

  // Compilation error in C++11: the typename keyword is required.
  using type2 = std::remove_cvref<T>::type;

  // OK.
  using type3 = sst::remove_cvref_t<T>;

};

int main() {
  return 0;
}
```

Compilation command

```
g++ sst-example-30.cpp -lsst
```

3.10.38. `sst::remove_reference_t`

C++

```
#include <sst/catalog/remove_reference_t.hpp>
namespace sst {

template<class T>
```

```
    using remove_reference_t =
        typename std::remove_reference<T>::type;

    }
```

The `sst::remove_reference_t` alias provides a convenient way to write `std::remove_reference<T>::type` that never requires the `typename` keyword to be used to disambiguate a dependent type.

The `sst::remove_reference_t` alias was added to the library because the `std::remove_reference_t` alias is not available below C++14.

Example 31.

sst-example-31.cpp

```
#include <type_traits>

#include <sst/catalog/remove_reference_t.hpp>

template<class T>
class foo {

  // OK.
  using type1 = typename std::remove_reference<T>::type;

  // Compilation error in C++11: the typename keyword is required.
  using type2 = std::remove_reference<T>::type;

  // OK.
  using type3 = sst::remove_reference_t<T>;

};

int main() {
  return 0;
}
```

Compilation command

```
g++ sst-example-31.cpp -lsst
```

3.10.39. The `sst::type_identity` trait

C++

```
#include <sst/catalog/type_identity.hpp>
namespace sst {

template<class T>
struct type_identity {
  using type = T;
```

```
};

}
```

The `sst::type_identity` trait defines `type` to be `T` itself.

The `sst::type_identity` trait was added to the library because the `std::type_identity` trait is not available below C++20.

3.10.40. `sst::type_identity_t`

C++

```
#include <sst/catalog/type_identity_t.hpp>
namespace sst {

template<class T>
using type_identity_t =
    typename sst::type_identity<T>::type;

}
```

The `sst::type_identity_t` alias provides a convenient way to write `sst::type_identity<T>::type` that never requires the `typename` keyword to be used to disambiguate a dependent type.

Example 32.

sst-example-32.cpp

```
#include <sst/catalog/type_identity.hpp>
#include <sst/catalog/type_identity_t.hpp>

template<class T>
class foo {

  // OK.
  using type1 = typename sst::type_identity<T>::type;

  // Compilation error in C++11: the typename keyword is required.
  using type2 = sst::type_identity<T>::type;

  // OK.
  using type3 = sst::type_identity_t<T>;

};

int main() {
  return 0;
}
```

Compilation command

```
g++ sst-example-32.cpp -lsst
```

3.10.41. The `sst::undeduced` trait

C++

```
#include <sst/catalog/undeduced.hpp>
namespace sst {

template<class T>
struct undeduced
    : sst::type_identity<R> {};

}
```

The `sst::undeduced` trait defines `R` to `T` itself, making it functionally equivalent to the `sst::type_identity` trait. The difference is only of style: both of these traits can be used to prevent template argument deduction from occurring, in which case using `sst::undeduced` is a better indication of intent than using `sst::type_identity`.

Example 33.

sst-example-33.cpp

```
#include <sst/catalog/undeduced_t.hpp>

template<class T>
void f(T, T) {}

template<class T>
void g(T, sst::undeduced_t<T>) {}

int main() {

  long x = 0;
  int y = 0;

  // Compilation error: it is ambiguous whether T should be deduced from
  // x as long or from y as int.
  f(x, y);

  // OK: T is deduced from x as long.
  g(x, y);

}
```

3.10.42. `sst::undeduced_t`

C++

```
#include <sst/catalog/undeduced_t.hpp>
namespace sst {

template<class T>
using undeduced_t =
    typename sst::undeduced<T>::type;

}
```

The `sst::undeduced_t` alias provides a convenient way to write `sst::undeduced<T>::type` that never requires the `typename` keyword to be used to disambiguate a dependent type.

Example 34.

sst-example-34.cpp

```
#include <sst/catalog/undeduced.hpp>
#include <sst/catalog/undeduced_t.hpp>

template<class T>
class foo {

  // OK.
  using type1 = typename sst::undeduced<T>::type;

  // Compilation error in C++11: the typename keyword is required.
  using type2 = sst::undeduced<T>::type;

  // OK.
  using type3 = sst::undeduced_t<T>;

};

int main() {
  return 0;
}
```

Compilation command

```
g++ sst-example-34.cpp -lsst
```

3.11. Iterator utilities

3.11.1. The `sstInputIterator` requirement

3.11.2. The `sstIterator` requirement

3.11.3. The `sst::count_it` function

C++

```
#include <sst/catalog/count_it.hpp>
namespace sst {

template<Iterator T, std::integral N>
R count_it(T x, N n);

}
```

The `sst::count_it` function returns an iterator that compares equal to the iterator that would be produced by `std::advance(x, n)`.

The type `R` of the returned iterator is unspecified. However, the following guarantees are made:

1. The return types of `sst::count_it(x, n)` and `sst::wrap_it(x, n)` are the same.
2. The return types of `sst::count_it(x, n)` and `sst::wrap_it(x, n)` are the same.

3.11.4. The `sst::track_it` function

The `sst::track_it` function returns an iterator that wraps another iterator with functionality for tracking and limiting the iterator's motion.

3.11.4.1. The `sst::track_it` function (overload 1)

```
#include <sst/catalog/track_it.hpp>
namespace sst {

template<
  class Count = int,
  class Iterator,
  sst::enable_if_t<
    sst::is_integer<Count>::value
  > = 0>
W track_it(Iterator & i);

}
```

The `sst::track_it` function (overload 1) returns an iterator that behaves like `i`, except it stores a pointer to `i` and updates `i` as it moves.

The returned iterator may only be used in a unique-pass algorithm.

The type `W` of the returned iterator is unspecified. However, the following guarantees are made:

1. Provided they are called with the same `Count` and `Iterator` types, the return types of `sst::track_it` (overload 1) and any `sst::count_it` overload are the same.

2. Provided they are called with the same `Count` and `Iterator` types, the return types of `sst::track_it` (overload 1) and any `sst::wrap_it` overload are the same.

3.11.4.2. The `sst::track_it` function (overload 2)

```
#include <sst/catalog/track_it.hpp>
namespace sst {

template<
  class Count,
  class Iterator,
  sst::enable_if_t<
    sst::is_integer<Count>::value
  > = 0>
W track_it(Iterator & i, Count n);

}
```

3.11.4.3. The `sst::track_it` function (overload 3)

```
#include <sst/catalog/track_it.hpp>
namespace sst {

template<
  class Count,
  class Iterator,
  sst::enable_if_t<
    sst::is_integer<Count>::value
  > = 0>
W track_it(Iterator & i, Count * n);

}
```

3.11.5. The `sst::wrap_it` function

The `sst::wrap_it` function returns an iterator that wraps another iterator with functionality for tracking and limiting the iterator's motion.

3.11.5.1. The `sst::wrap_it` function (overload 1)

```
#include <sst/catalog/wrap_it.hpp>
namespace sst {

template<
  class Count = int,
  class Iterator,
  sst::enable_if_t<
    sst::is_integer<Count>::value
  > = 0>
```

```
W wrap_it(Iterator i);

}
```

The `sst::wrap_it` function (overload 1) returns an iterator that behaves like `i`.

The returned iterator may only be used in a unique-pass algorithm.

The type `W` of the returned iterator is unspecified. However, the return types of all `sst::count_it`, `sst::track_it`, and `sst::wrap_it` overloads are the same provided they are called with the same `Count` and `Iterator` types.

The main purpose of this overload is to achieve type equality in cases where `i` is associated with another `sst::count_it`, `sst::track_it`, or `sst::wrap_it` iterator and the iterators are being passed to an algorithm that requires them to have the same type. For example, `std::copy` requires its first two arguments to have the same type, so passing `i` and `sst::count_it(i, 10)` will fail to compile, but passing `sst::wrap_it(i)` and `sst::count_it(i, 10)` will work. In this case, you must also be careful to ensure that both iterators have the same `Count` type. Continuing the example, if we were copying `n` elements instead of `10`, we could pass `sst::wrap_it<decltype(n)>(i)` and `sst::count_it(i, n)`.

3.11.5.2. The `sst::wrap_it` function (overload 2)

```
#include <sst/catalog/wrap_it.hpp>
namespace sst {

template<
  class Count,
  class Iterator,
  sst::enable_if_t<
    sst::is_integer<Count>::value
  > = 0>
W wrap_it(Iterator i, Count n);

}
```

3.11.5.3. The `sst::wrap_it` function (overload 3)

```
#include <sst/catalog/wrap_it.hpp>
namespace sst {

template<
  class Count,
  class Iterator,
  sst::enable_if_t<
    sst::is_integer<Count>::value
  > = 0>
W wrap_it(Iterator i, Count * n);

}
```

3.12. JSON utilities

The following functions are provided for convenience. They are equivalent to the corresponding `sst::json::get_to` functions except they return their results by value instead of having `dst` parameters:

C++

```
#include <link:{repo_browser_url}/src/c-
cpp/include/sst/catalog/json/get_as.hpp[sst/catalog/json/get_as.hpp,window=_b
lank]>
namespace sst::json {

template<class ValueType, class Json>
ValueType get_as(Json const & src) { ... }

template<class ValueType, class Json>
ValueType get_as(Json const & src,
    typename Json::object_t::key_type const & key) { ... }

template<class ValueType, class Json>
ValueType get_as(Json const & src,
    typename Json::object_t::key_type const & key,
    ValueType const & default_value) { ... }

}
```

3.12.1. The `sst::json::exception` class

C++

```
#include <sst/catalog/json/exception.hpp>
namespace sst::json {

class exception : public std::runtime_error {

};

}
```

3.12.2. The `sst::json::get_from_file` function

C++

```
#include <sst/catalog/json/get_from_file.hpp>
namespace sst::json {

template<class T, class NlohmannJson = T, class CharT = char>
T get_from_file(std::string const & src);
```

```
template<class T, class NlohmannJson = T, class CharT = char>
T get_from_file(char const * const src);

}
```

3.12.3. The `sst::json::get_as_file` function

C++

```
#include <sst/catalog/json/get_as_file.hpp>
namespace sst::json {

template<class CharT = char, class NlohmannJson>
void get_as_file(NlohmannJson const & src,
                 std::string const & dst,
                 int indent = 2);

template<class CharT = char, class NlohmannJson>
void get_as_file(NlohmannJson const & src,
                 char const * const dst,
                 int indent = 2);

}
```

3.12.4. The `sst::json::get_to` function

C++

```
#include <sst/catalog/json/get_to.hpp>
namespace sst::json {

//
// Integer destinations
//

// (1)
template<class ValueType,
         class NlohmannJson,
         typename sst::enable_if<
             std::is_integral<ValueType>::value>::type = 0>
ValueType & get_to(NlohmannJson const & src, ValueType & dst);

// (2)
template<class ValueType,
         class NlohmannJson,
         typename sst::enable_if<
             std::is_same<ValueType, BIGNUM>::value>::type = 0>
ValueType & get_to(NlohmannJson const & src, ValueType & dst);

// (3)
template<class ValueType,
```

```
            class NlohmannJson,
            typename sst::enable_if<
                std::is_same<ValueType, sst::bignum>::value>::type = 0>
    ValueType & get_to(NlohmannJson const & src, ValueType & dst);

    }
```

The `sst::json::get_to` function parses an `nlohmann::json` value into `dst` based on the type of `dst` and returns `dst`.

Integer destinations: (1) (2) (3)

Parses `src` into `dst`.

If the internal type of `src` is not `integer_t`, `unsigned_t`, `boolean_t`, or `string_t`, an `sst::json::exception` exception is thrown.

For `integer_t` and `unsigned_t`, the parsed value is simply the internal value. For `boolean_t`, the parsed values of `true` and `false` are 1 and 0, respectively. For `string_t`, the accepted syntax and parsed value is defined by the following grammar:

value:
    sign$_{opt}$ *magnitude*
    boolean

sign: one of
    `+` `-`

magnitude:
    `0`
    binary-magnitude
    octal-magnitude
    decimal-magnitude
    hexadecimal-magnitude

binary-magnitude:
    `0b` *binary-digit*
    `0B` *binary-digit*
    binary-magnitude `'`$_{opt}$ *binary-digit*

octal-magnitude:
    `0o` *octal-digit*
    `0O` *octal-digit*
    octal-magnitude `'`$_{opt}$ *octal-digit*

decimal-magnitude:
    nonzero-decimal-digit
    decimal-magnitude `'`$_{opt}$ *decimal-digit*

hexadecimal-magnitude:
    `0x` *hexadecimal-digit*
    `0X` *hexadecimal-digit*
    hexadecimal-magnitude `'`$_{opt}$ *hexadecimal-digit*

binary-digit: one of
`0 1`

octal-digit: one of
`0 1 2 3 4 5 6 7`

nonzero-decimal-digit: one of
`1 2 3 4 5 6 7 8 9`

decimal-digit: one of
`0 1 2 3 4 5 6 7 8 9`

hexadecimal-digit: one of
`0 1 2 3 4 5 6 7 8 9`
`a b c d e f`
`A B C D E F`

boolean: one of
`true false`

If the parsed value cannot be represented by `ValueType`, an `sst::json::exception` exception is thrown.

If `ValueType` is a `std::vector`-like type, then the internal type of `src` must be `array_t`. `dst` will first be cleared, and then each element of the array will be appended to `dst` after being parsed by calling `sst::json::get_to` recursively.

If `ValueType` is a `std::map`-like type, then the internal type of `src` must be `object_t`. `dst` will first be cleared, and then each member of the object will be inserted into `dst` after being parsed by calling `sst::json::get_to` recursively.

If `ValueType` is any other type, then the parsing will be forwarded to `nlohmann::json`'s built-in parsing logic by calling `src.get_to(dst)`. In particular, this means that `nlohmann::json`'s arbitrary type parsing feature (i.e., user-provided `from_json` functions) will work when calling `sst::json::get_to`.

Except perhaps when parsing is forwarded to `nlohmann::json`'s built-in parsing logic, this function will reliably throw an exception if any kind of error occurs. In particular, it will throw an exception if `ValueType` is an integer type that cannot represent a parsed integer value.

The following functions are provided for parsing JSON object members:

C++

```
#include <link:{repo_browser_url}/src/c-
cpp/include/sst/catalog/json/get_to.hpp[sst/catalog/json/get_to.hpp,window=_b
lank]>
namespace sst::json {

// (1)
template<class ValueType, class Json>
ValueType & get_to(Json const & src,
    typename Json::object_t::key_type const & key,
    ValueType & dst) { ... }
```

```
// (2)
template<class T, class {cl_nlohmann_json_Json}>
T & get_to({cl_nlohmann_json_Json} const & src,
           typename {cl_nlohmann_json_Json}::object_t::key_type const & key,
           T & dst,
           {cl_sst_undeduced_t}<T> const & default_value);

template<class T, class {cl_nlohmann_json_Json}>
T & get_to({cl_nlohmann_json_Json} const & src,
           typename {cl_nlohmann_json_Json}::object_t::key_type const & key,
           T & dst,
           {cl_sst_undeduced_t}<T> && default_value);

}
```

The internal type of `src` must be `object_t`. For (1), `src` must contain a member named `key`, whose value will be parsed by calling `sst::json::get_to` recursively. For (2), the behavior is the same as (1) except that if `src` does not contain a member named `key`, then `dst` will be set to `default_value`.

3.12.5. The `sst::json::get_as` function

C++

```
#include <sst/catalog/json/get_as.hpp>
namespace sst::json {

template<class T, class NlohmannJson, class... Args>
T get_as(NlohmannJson & src, typename NlohmannJson::object_t::key_type const
& key);

}
```

3.12.6. The `sst::json::remove_to` function

C++

```
#include <sst/catalog/json/remove_to.hpp>
namespace sst::json {

template<class T, class NlohmannJson, class... Args>
T & remove_to(NlohmannJson & src,
              typename NlohmannJson::object_t::key_type const & key,
              T & dst,
              Args &&... args);

}
```

Calling the `sst::json::remove_to` function is the same as calling the `sst::json::get_to` function with the same parameters, except `key` will also be removed from `src`, if present.

3.12.7. The `sst::json::remove_as` function

C++

```
#include <sst/catalog/json/remove_as.hpp>
namespace sst::json {

template<class T, class NlohmannJson, class... Args>
T remove_as(NlohmannJson & src,
            typename NlohmannJson::object_t::key_type const & key,
            Args &&... args);

}
```

Calling the `sst::json::remove_as` function is the same as calling the `sst::json::get_as` function with the same parameters, except `key` will also be removed from `src`, if present.

3.12.8. The `sst::json::unknown_key` function (overload 1)

C++

```
#include <sst/catalog/json/unknown_key.hpp>
namespace sst::json {

template<class NlohmannJson>
void unknown_key(NlohmannJson const & src);

}
```

The `sst::json::unknown_key` function (overload 1) checks for an unknown key in a JSON object.

If `src` is a JSON value other than an object, the function throws an `sst::json::exception`. Otherwise, if `src` does not contain any keys, the function takes no action. Otherwise, the function throws an `sst::json::exception` with a message that includes at least one of the keys.

This function is intended to be combined with the convention of deleting keys from a JSON object as they are parsed. After all known keys are parsed, any remaining keys are unknown, and calling this function will reject them.

3.12.9. The `sst::json::expect_string` function

C++

```
#include <sst/catalog/json/expect_string.hpp>
namespace sst::json {

template<class NlohmannJson>
NlohmannJson const & expect_string(NlohmannJson const & src);

template<class NlohmannJson>
NlohmannJson & expect_string(NlohmannJson & src);
```

```
template<class NlohmannJson>
NlohmannJson const &
expect_string(NlohmannJson const & src,
              typename NlohmannJson::object_t::key_type const & key);

template<class NlohmannJson>
NlohmannJson & expect_string(NlohmannJson & src,
                     typename NlohmannJson::object_t::key_type const & key);

}
```

3.12.10. The `sst::json::expect_object` function

C++

```
#include <sst/catalog/json/expect_object.hpp>
namespace sst::json {

template<class NlohmannJson>
NlohmannJson const & expect_object(NlohmannJson const & src);

template<class NlohmannJson>
NlohmannJson & expect_object(NlohmannJson & src);

template<class NlohmannJson>
NlohmannJson const &
expect_object(NlohmannJson const & src,
              typename NlohmannJson::object_t::key_type const & key);

template<class NlohmannJson>
NlohmannJson & expect_object(NlohmannJson & src,
                     typename NlohmannJson::object_t::key_type const & key);

}
```

3.13. Individual bit access

3.13.1. The `sst::get_bit` function

C++

```
#include <sst/bit.h>
namespace sst {

// (1)
template<class X, class Q, class R>
constexpr bool get_bit(X x, Q q, R r) noexcept;
```

```
// (2)
template<class X, class R>
constexpr bool get_bit(X x, R r) noexcept;

// (3)
template<class Q, class R>
bool get_bit(void const * x, Q q, R r) noexcept;

// (4)
template<class R>
bool get_bit(void const * x, R r) noexcept;

}
```

The `sst::get_bit` function returns the bit at index `q` · `CHAR_BIT` + `r` of the two's complement representation of `x` (for (1) and (2)) or the region of bytes pointed to by `x` (for (3) and (4)). For (1) and (2), the bits of `x` are zero indexed starting with the least significant bit (and the most significant bit is the sign bit if `X` is signed). For (3) and (4), the bits of the region of bytes pointed to by `x` are zero indexed starting with the least significant bit of the first byte.

`X`, `Q`, and `R` may be any integer types. If `x` is a null pointer, `q` is negative, `r` is negative, or `q` · `CHAR_BIT` + `r` is greater than or equal to the number of nonpadding bits in `X` or the number of bits in the region of bytes pointed to by `x`, the behavior is undefined. For (2) and (4), `q` is omitted and taken to be zero.

Prior to C++20, signed integers are permitted to use ones' complement or sign-magnitude representation instead of two's complement. This does not affect the behavior of (1) or (2).

3.13.2. The `sst::set_bit` function

C++

```
#include <sst/bit.h>
namespace sst {

// (1)
template<class X, class Q, class R>
constexpr X set_bit(X x, Q q, R r, bool b) noexcept;

// (2)
template<class X, class R>
constexpr X set_bit(X x, R r, bool b) noexcept;

// (3)
template<class Q, class R>
void * set_bit(void * x, Q q, R r, bool b) noexcept;

// (4)
template<class R>
void * set_bit(void * x, R r, bool b) noexcept;

}
```

The `sst::set_bit` function sets the bit at index `q` · `CHAR_BIT` + `r` of the two's complement representation of `x` (for (1) and (2)) or the region of bytes pointed to by `x` (for (3) and (4)) to `b`. For (1) and (2), the bits of `x` are zero indexed starting with the least significant bit (and the most significant bit is the sign bit if `X` is signed), and the function returns the resulting value. For (3) and (4), the bits of the region of bytes pointed to by `x` are zero indexed starting with the least significant bit of the first byte, and the function returns `x`.

`X`, `Q`, and `R` may be any integer types. If `x` is a null pointer, `q` is negative, `r` is negative, or `q` · `CHAR_BIT` + `r` is greater than or equal to the number of nonpadding bits in `X` or the number of bits in the region of bytes pointed to by `x`, the behavior is undefined. For (2) and (4), `q` is omitted and taken to be zero.

Prior to C++20, signed integers are permitted to use ones' complement or sign-magnitude representation instead of two's complement. This does not affect the behavior of (1) or (2), except that if the resulting value cannot be represented by `X` (i.e., if `X` is signed, `x` is zero, and the most significant bit is being set to 1), the behavior is undefined.

3.14. Integer packing

3.14.1. The `sst::from_bits` function

C++

```
#include <sst/catalog/from_bits.hpp>
// or:   <sst/representation.h>
namespace sst {

// (1)
template<class Y, bool SignExtend = std::is_signed<Y>::value,
         class Q, class R, class N>
Y from_bits(void const * x, Q q, R r, N n) noexcept;

// (2)
template<class Y, bool SignExtend = std::is_signed<Y>::value,
         class R, class N>
Y from_bits(void const * x, R r, N n) noexcept;

}
```

The `sst::from_bits` function loads an integer from a sequence of bits of memory. First, the `n` bits of memory starting at bit index `q` · `CHAR_BIT` + `r` of the region of bytes pointed to by `x` are interpreted as a two's complement integer if `SignExtend` is `true`, or as an unsigned integer if `SignExtend` is `false`. The first bit is taken to be the least significant bit of the integer, and the last bit is taken to be the most significant bit of the integer (which is the sign bit if `SignExtend` is `true`). The bits of the region of bytes pointed to by `x` are zero indexed starting with the least significant bit of the first byte. Next, the least significant w bits of the infinite two's complement representation of this integer are considered, where w is the number of nonpadding bits in `Y`. If `Y` is signed, these bits are interpreted as a two's complement integer (with the most significant bit being the sign bit) and the resulting value is returned. If `Y` is unsigned, these bits are interpreted as an unsigned integer and the resulting value is returned.

`Y`, `Q`, `R`, and `N` may be any integer types. If `x` is a null pointer, `q` is negative, `r` is negative, `n` is not positive, or `q` · `CHAR_BIT` + `r` + `n` is greater than the number of bits in the region of bytes pointed to by `x`, the behavior is undefined. For (2), `q` is omitted and taken to be zero.

Prior to C++20, signed integers are permitted to use ones' complement or sign-magnitude representation instead of two's complement. This does not affect the behavior of this function, except that if the resulting value cannot be represented by `Y` (i.e., if `Y` is signed and the resulting value is -2^{1-w}), the behavior is undefined.

3.14.2. The `sst::to_bits` function

C++

```
#include <sst/catalog/to_bits.hpp>
// or:   <sst/representation.h>
namespace sst {

// (1)
template<class Y, bool SignExtend = std::is_signed<Y>::value,
         class Q, class R, class N>
void * to_bits(void * x, Q q, R r, Y y, N n) noexcept;

// (2)
template<class Y, bool SignExtend = std::is_signed<Y>::value,
         class R, class N>
void * to_bits(void * x, R r, Y y, N n) noexcept;

}
```

The `sst::to_bits` function stores an integer to a sequence of bits of memory. First, the two's complement representation of `y` is considered, extending it with infinitely many copies of its most significant bit (which is the sign bit if `Y` is signed) if `SignExtend` is `true`, or with infinitely many 0's if `SignExtend` is `false`. Next, the least significant `n` bits of this infinite representation are stored to the `n` bits of memory starting at bit index `q` · `CHAR_BIT` + `r` of the region of bytes pointed to by `x`. The bits of the region of bytes pointed to by `x` are zero indexed starting with the least significant bit of the first byte. Finally, the function returns `x`.

`Y`, `Q`, `R`, and `N` may be any integer types. If `x` is a null pointer, `q` is negative, `r` is negative, `n` is not positive, or `q` · `CHAR_BIT` + `r` + `n` is greater than the number of bits in the region of bytes pointed to by `x`, the behavior is undefined. For (2), `q` is omitted and taken to be zero.

Prior to C++20, signed integers are permitted to use ones' complement or sign-magnitude representation instead of two's complement. This does not affect the behavior of this function.

3.15. Basic algorithms

3.15.1. The `sst::min` function

C++

```
#include <sst/catalog/min.hpp>
// or:   <sst/algorithm.h>
namespace sst {

template<class T>
constexpr T const & min(T const & a, T const & b);

}
```

The `sst::min` function returns `a < b ? a : b`.

This function was added to the library because `std::min` is not `constexpr` before C++14.

3.15.2. The `sst::max` function

C++

```
#include <sst/catalog/max.hpp>
// or:   <sst/algorithm.h>
namespace sst {

template<class T>
constexpr T const & max(T const & a, T const & b);

}
```

The `sst::max` function returns `a < b ? b : a`.

This function was added to the library because `std::max` is not `constexpr` before C++14.

3.15.3. The `sst::floor_sqrt` function

C++

```
#include <sst/catalog/floor_sqrt.hpp>
// or:   <sst/algorithm.h>
namespace sst {

template<std::integral T>
SST_CPP14_CONSTEXPR T floor_sqrt(T x) noexcept;

template<BigInteger T>
T floor_sqrt(T const & x);

template<std::floating_point T>
T floor_sqrt(T x);

}
```

The `sst::floor_sqrt` function returns $\lfloor\sqrt{\texttt{x}}\rfloor$. If `x` is negative, the behavior is undefined.

3.15.4. The `sst::ceil_sqrt` function

C++

```
#include <sst/catalog/ceil_sqrt.hpp>
// or:   <sst/algorithm.h>
namespace sst {

template<std::integral T>
SST_CPP14_CONSTEXPR T ceil_sqrt(T x) noexcept;

template<BigInteger T>
T ceil_sqrt(T const & x);

template<std::floating_point T>
T ceil_sqrt(T x);

}
```

The `sst::ceil_sqrt` function returns $\lceil\sqrt{\texttt{x}}\rceil$. If `x` is negative, the behavior is undefined.

3.15.5. The `sst::floor_lg` function

C++

```
#include <link:{repo_browser_url}/src/c-
cpp/include/sst/catalog/floor_lg.hpp[sst/catalog/floor_lg.hpp,window=_blank]>
namespace sst {

template<{cl_std_integral} T>
{cl_SST_CPP14_CONSTEXPR} T floor_lg(T x) noexcept;

template<BigInteger T>
T floor_lg(T const & x);

template<{cl_std_floating_point} T>
T floor_lg(T x);

}
```

The `sst::floor_lg` function returns $\lfloor\lg\texttt{x}\rfloor$. If `x` is negative or zero, the behavior is undefined.

3.15.6. The `sst::ceil_lg` function

C++

```
#include <link:{repo_browser_url}/src/c-
cpp/include/sst/catalog/ceil_lg.hpp[sst/catalog/ceil_lg.hpp,window=_blank]>
```

```
namespace sst {

template<{cl_std_integral} T>
{cl_SST_CPP14_CONSTEXPR} T ceil_lg(T x) noexcept;

template<BigInteger T>
T ceil_lg(T const & x);

template<{cl_std_floating_point} T>
T ceil_lg(T x);

}
```

The `sst::ceil_lg` function returns $\lceil \lg x \rceil$. If `x` is negative or zero, the behavior is undefined.

3.16. String utilities

3.16.1. The `sst::to_string` function

C++

```
#include <sst/catalog/to_string.hpp>
namespace sst {

// (1)
template<class CharT, class OutputIt, class... Args>
OutputIt to_string(std::basic_string<CharT, Args...> const & src,
                   OutputIt dst,
                   bool replace_unrepresentable = false);

// (2)
template<class CharT, class... Args>
std::string to_string(std::basic_string<CharT, Args...> const & src,
                      bool replace_unrepresentable = false);

// (3)
template<std::integral T, class OutputIt>
OutputIt to_string(T const & src, OutputIt dst);

// (4)
template<std::integral T>
std::string to_string(T const & src);

}
```

The `sst::to_string` function converts an object to a string.

3.16.1.1. String sources

Converts one type of string to another.

3.16.1.2. Integer sources

Converts an integer to a string.

3.17. Text encoding utilities

3.17.1. The `sst::pick_utf_encoding` function

```
#include <sst/catalog/pick_utf_encoding.hpp>
namespace sst {

template<sst::is_integer_ish T>
constexpr sst::text_encoding pick_utf_encoding() noexcept;

}
```

The `sst::pick_utf_encoding` function maps an integer-ish type `T` to a UTF encoding via the following algorithm:

1. If `sst::numeric_limits<T>::is_specialized` is true:
    a. If `sst::numeric_limits<T>::min()` is negative:
        i. If the range of `T` is a superset of $[-2^{31}, 2^{31})$, the result is `sst::text_encoding::utf_32`.
        ii. Otherwise, if the range of `T` is a superset of $[-2^{15}, 2^{15})$, the result is `sst::text_encoding::utf_16`.
        iii. Otherwise, the result is `sst::text_encoding::utf_8`.
    b. Otherwise:
        i. If the range of `T` is a superset of $[0, 2^{32})$, the result is `sst::text_encoding::utf_32`.
        ii. Otherwise, if the range of `T` is a superset of $[0, 2^{16})$, the result is `sst::text_encoding::utf_16`.
        iii. Otherwise, the result is `sst::text_encoding::utf_8`.
2. Otherwise, the result is `sst::text_encoding::utf_32`.

3.17.2. The `sst::text_encoding` type

```
#include <sst/catalog/text_encoding.hpp>
namespace sst {

enum class sst::text_encoding {
  utf_16,
```

```
  utf_32,
  utf_8,
};

}
```

3.18. Time utilities

3.18.1. The `sst::mono_time` function

C++

```
#include <sst/catalog/mono_time.hpp>
// or:   <sst/time.h>
namespace sst {

template<class Duration>
Duration mono_time();

}
```

The `sst::mono_time` function returns the current time of an implementation-defined monotonic clock with respect to `Duration`.

3.18.2. The `sst::mono_time_s` function

C++

```
#include <sst/catalog/mono_time_s.hpp>
// or:   <sst/time.h>
namespace sst {

inline sst::mono_time_s_t mono_time_s();

}
```

The `sst::mono_time_s` function returns the current time of the `sst::mono_time` clock in seconds.

Note that the return type, `sst::mono_time_s_t`, is an unspecified signed integer type with at least 35 width bits[6]. For more information, see `std::chrono::duration`, C++11 (N3337) §20.11.2, C++14 (N4140) §20.12.2, C++17 (N4659) §23.17.2, or C++20 (N4860) §27.2.

Example 35.

Program: `a.cpp`

```
#include <chrono>
#include <iostream>
#include <sst/catalog/mono_time_s.hpp>
```

```
#include <thread>

int main() {
  auto const t0 = sst::mono_time_s();
  std::this_thread::sleep_for(std::chrono::seconds(1));
  auto const t1 = sst::mono_time_s();
  std::cout << (t1 - t0) << "\n";
}
```

Compiling and running

```
g++ a.cpp -lsst
./a.out
```

Possible output

```
1
```

3.18.3. The `sst::mono_time_s_t` type

C++

```
#include <sst/catalog/mono_time_s_t.hpp>
// or:   <sst/time.h>
namespace sst {

using mono_time_s_t = std::chrono::seconds::rep;

}
```

3.18.4. The `sst::mono_time_ms` function

C++

```
#include <sst/catalog/mono_time_ms.hpp>
// or:   <sst/time.h>
namespace sst {

inline sst::mono_time_ms_t mono_time_ms();

}
```

The `sst::mono_time_ms` function returns the current time of the `sst::mono_time` clock in milliseconds.

Note that the return type, `sst::mono_time_ms_t`, is an unspecified signed integer type with at least 45 width bits[6]. For more information, see `std::chrono::duration`, C++11 (N3337) §20.11.2, C++14 (N4140) §20.12.2, C++17 (N4659) §23.17.2, or C++20 (N4860) §27.2.

Example 36.

Program: `a.cpp`

```
#include <chrono>
#include <iostream>
#include <sst/catalog/mono_time_ms.hpp>
#include <thread>

int main() {
  auto const t0 = sst::mono_time_ms();
  std::this_thread::sleep_for(std::chrono::seconds(1));
  auto const t1 = sst::mono_time_ms();
  std::cout << (t1 - t0) << "\n";
}
```

Compiling and running

```
g++ a.cpp -lsst
./a.out
```

Possible output

```
1000
```

3.18.5. The `sst::mono_time_ms_t` type

C++

```
#include <sst/catalog/mono_time_ms_t.hpp>
// or:   <sst/time.h>
namespace sst {

using mono_time_ms_t = std::chrono::milliseconds::rep;

}
```

3.18.6. The `sst::mono_time_us` function

C++

```
#include <sst/catalog/mono_time_us.hpp>
// or:   <sst/time.h>
namespace sst {

inline sst::mono_time_us_t mono_time_us();

}
```

The `sst::mono_time_us` function returns the current time of the `sst::mono_time` clock in microseconds.

Note that the return type, `sst::mono_time_us_t`, is an unspecified signed integer type with at least 55 width bits[6]. For more information, see `std::chrono::duration`, C++11 (N3337) §20.11.2, C++14 (N4140) §20.12.2, C++17 (N4659) §23.17.2, or C++20 (N4860) §27.2.

Example 37.

Program: `a.cpp`

```
#include <chrono>
#include <iostream>
#include <sst/catalog/mono_time_us.hpp>
#include <thread>

int main() {
  auto const t0 = sst::mono_time_us();
  std::this_thread::sleep_for(std::chrono::seconds(1));
  auto const t1 = sst::mono_time_us();
  std::cout << (t1 - t0) << "\n";
}
```

Compiling and running

```
g++ a.cpp -lsst
./a.out
```

Possible output

```
1000149
```

3.18.7. The `sst::mono_time_us_t` type

C++

```
#include <sst/catalog/mono_time_us_t.hpp>
// or:   <sst/time.h>
namespace sst {

using mono_time_us_t = std::chrono::microseconds::rep;

}
```

3.18.8. The `sst::mono_time_ns` function

C++

```
#include <sst/catalog/mono_time_ns.hpp>
// or:   <sst/time.h>
namespace sst {

inline sst::mono_time_ns_t mono_time_ns();
```

```
}
```

The `sst::mono_time_ns` function returns the current time of the `sst::mono_time` clock in nanoseconds.

Note that the return type, `sst::mono_time_ns_t`, is an unspecified signed integer type with at least 64 width bits[6]. For more information, see `std::chrono::duration`, C++11 (N3337) §20.11.2, C++14 (N4140) §20.12.2, C++17 (N4659) §23.17.2, or C++20 (N4860) §27.2.

Example 38.

Program: `a.cpp`

```
#include <chrono>
#include <iostream>
#include <sst/catalog/mono_time_ns.hpp>
#include <thread>

int main() {
  auto const t0 = sst::mono_time_ns();
  std::this_thread::sleep_for(std::chrono::seconds(1));
  auto const t1 = sst::mono_time_ns();
  std::cout << (t1 - t0) << "\n";
}
```

Compiling and running

```
g++ a.cpp -lsst
./a.out
```

Possible output

```
1000133270
```

3.18.9. The `sst::mono_time_ns_t` type

C++

```
#include <sst/catalog/mono_time_ns_t.hpp>
// or:   <sst/time.h>
namespace sst {

using mono_time_ns_t = std::chrono::nanoseconds::rep;

}
```

3.18.10. The `sst::mono_time_d` function

C++

```
#include <sst/catalog/mono_time_d.hpp>
```

```
// or:   <sst/time.h>
namespace sst {

inline double mono_time_d();

}
```

The `sst::mono_time_d` function returns the current time of the `sst::mono_time` clock in seconds with as much precision as possible.

Example 39.

Program: `a.cpp`

```
#include <chrono>
#include <iostream>
#include <sst/catalog/mono_time_d.hpp>
#include <thread>

int main() {
  auto const t0 = sst::mono_time_d();
  std::this_thread::sleep_for(std::chrono::seconds(1));
  auto const t1 = sst::mono_time_d();
  std::cout << (t1 - t0) << "\n";
}
```

Compiling and running

```
g++ a.cpp -lsst
./a.out
```

Possible output

```
1.00017
```

3.18.11. The `sst::unix_time` function

C++

```
#include <sst/catalog/unix_time.hpp>
// or:   <sst/time.h>
namespace sst {

template<class Duration>
Duration unix_time();

}
```

The `sst::unix_time` function returns the current time of the Unix clock with respect to `Duration`.

Prior to C++20, this function may use a clock other than the Unix clock, although there are no

known implementations that do so. For more information, see `std::chrono::system_clock` or compare C++17 (N4659) §23.17.7.1 with C++20 (N4860) §27.7.1.1.

3.18.12. The `sst::unix_time_s` function

C++

```
#include <sst/catalog/unix_time_s.hpp>
// or:   <sst/time.h>
namespace sst {

inline sst::unix_time_s_t unix_time_s();

}
```

The `sst::unix_time_s` function returns the current time of the Unix clock in seconds.

Note that the return type, `sst::unix_time_s_t`, is an unspecified signed integer type with at least 35 width bits[6]. For more information, see `std::chrono::duration`, C++11 (N3337) §20.11.2, C++14 (N4140) §20.12.2, C++17 (N4659) §23.17.2, or C++20 (N4860) §27.2.

Example 40.

Program: `a.cpp`

```
#include <iostream>
#include <sst/catalog/asctime.hpp>
#include <sst/catalog/gmtime.hpp>
#include <sst/catalog/unix_time_s.hpp>

int main() {
  std::cout << sst::asctime(sst::gmtime());
  std::cout << sst::unix_time_s() << "\n";
}
```

Compiling and running

```
g++ a.cpp -lsst
./a.out
```

Possible output

```
Wed Aug 25 05:04:19 2021
1629867859
```

3.18.13. The `sst::unix_time_s_t` type

C++

```
#include <sst/catalog/unix_time_s_t.hpp>
// or:   <sst/time.h>
```

```
namespace sst {

using unix_time_s_t = std::chrono::seconds::rep;

}
```

3.18.14. The `sst::unix_time_ms` function

C++

```
#include <sst/catalog/unix_time_ms.hpp>
// or:   <sst/time.h>
namespace sst {

inline sst::unix_time_ms_t unix_time_ms();

}
```

The `sst::unix_time_ms` function returns the current time of the Unix clock in milliseconds.

Note that the return type, `sst::unix_time_ms_t`, is an unspecified signed integer type with at least 45 width bits[6]. For more information, see `std::chrono::duration`, C++11 (N3337) §20.11.2, C++14 (N4140) §20.12.2, C++17 (N4659) §23.17.2, or C++20 (N4860) §27.2.

Example 41.

Program: `a.cpp`

```
#include <iostream>
#include <sst/catalog/asctime.hpp>
#include <sst/catalog/gmtime.hpp>
#include <sst/catalog/unix_time_ms.hpp>

int main() {
  std::cout << sst::asctime(sst::gmtime());
  std::cout << sst::unix_time_ms() << "\n";
}
```

Compiling and running

```
g++ a.cpp -lsst
./a.out
```

Possible output

```
Wed Aug 25 05:04:19 2021
1629867859036
```

3.18.15. The `sst::unix_time_ms_t` type

C++

```
#include <sst/catalog/unix_time_ms_t.hpp>
// or:   <sst/time.h>
namespace sst {

using unix_time_ms_t = std::chrono::milliseconds::rep;

}
```

3.18.16. The `sst::unix_time_us` function

C++

```
#include <sst/catalog/unix_time_us.hpp>
// or:   <sst/time.h>
namespace sst {

inline sst::unix_time_us_t unix_time_us();

}
```

The `sst::unix_time_us` function returns the current time of the Unix clock in microseconds.

Note that the return type, `sst::unix_time_us_t`, is an unspecified signed integer type with at least 55 width bits[6]. For more information, see `std::chrono::duration`, C++11 (N3337) §20.11.2, C++14 (N4140) §20.12.2, C++17 (N4659) §23.17.2, or C++20 (N4860) §27.2.

Example 42.

Program: `a.cpp`

```
#include <iostream>
#include <sst/catalog/asctime.hpp>
#include <sst/catalog/gmtime.hpp>
#include <sst/catalog/unix_time_us.hpp>

int main() {
  std::cout << sst::asctime(sst::gmtime());
  std::cout << sst::unix_time_us() << "\n";
}
```

Compiling and running

```
g++ a.cpp -lsst
./a.out
```

Possible output

```
Wed Aug 25 05:04:19 2021
1629867859037931
```

3.18.17. The `sst::unix_time_us_t` type

C++

```
#include <sst/catalog/unix_time_us_t.hpp>
// or:   <sst/time.h>
namespace sst {

using unix_time_us_t = std::chrono::microseconds::rep;

}
```

3.18.18. The `sst::unix_time_ns` function

C++

```
#include <sst/catalog/unix_time_ns.hpp>
// or:   <sst/time.h>
namespace sst {

inline sst::unix_time_ns_t unix_time_ns();

}
```

The `sst::unix_time_ns` function returns the current time of the Unix clock in nanoseconds.

Note that the return type, `sst::unix_time_ns_t`, is an unspecified signed integer type with at least 64 width bits[6]. For more information, see `std::chrono::duration`, C++11 (N3337) §20.11.2, C++14 (N4140) §20.12.2, C++17 (N4659) §23.17.2, or C++20 (N4860) §27.2.

Example 43.

Program: `a.cpp`

```
#include <iostream>
#include <sst/catalog/asctime.hpp>
#include <sst/catalog/gmtime.hpp>
#include <sst/catalog/unix_time_ns.hpp>

int main() {
  std::cout << sst::asctime(sst::gmtime());
  std::cout << sst::unix_time_ns() << "\n";
}
```

Compiling and running

```
g++ a.cpp -lsst
./a.out
```

Possible output

```
Wed Aug 25 05:04:19 2021
```

```
1629867859039736561
```

3.18.19. The `sst::unix_time_ns_t` type

C++

```
#include <sst/catalog/unix_time_ns_t.hpp>
// or:   <sst/time.h>
namespace sst {

using unix_time_ns_t = std::chrono::nanoseconds::rep;

}
```

3.18.20. The `sst::unix_time_d` function

C++

```
#include <sst/catalog/unix_time_d.hpp>
// or:   <sst/time.h>
namespace sst {

inline double unix_time_d();

}
```

The `sst::unix_time_d` function returns the current time of the Unix clock in seconds with as much precision as possible.

Example 44.

Program: `a.cpp`

```
#include <iostream>
#include <sst/catalog/asctime.hpp>
#include <sst/catalog/gmtime.hpp>
#include <sst/catalog/unix_time_d.hpp>

int main() {
  std::cout << sst::asctime(sst::gmtime());
  std::cout << sst::unix_time_d() << "\n";
}
```

Compiling and running

```
g++ a.cpp -lsst
./a.out
```

Possible output

```
Wed Aug 25 05:04:19 2021
```

```
1.62987e+09
```

3.19. Filesystem tools

3.19.1. The `sst::dir_it` class

The `sst::dir_it` class iterates through the contents of a directory.

C++

```
#include <sst/catalog/dir_it.hpp>
namespace sst {

class dir_it {
public:

  dir_it();

};

}
```

3.19.1.1. The `sst::dir_it` constructor (overload 1)

```
dir_it();
```

The `sst::dir_it` constructor (overload 1) creates a one-past-the-end iterator.

3.19.2. The `sst::mkdir` function

```
#include <sst/catalog/mkdir.hpp>
namespace sst {

template<
  class Path
  sst::enable_if_t<
    // TODO
  > = 0>
bool mkdir(
  Path const & path
);

}
```

3.19.3. The `sst::mkdir_p` function

```
#include <sst/catalog/mkdir_p.hpp>
```

```
namespace sst {

template<
  class Path
  sst::enable_if_t<
    // TODO
  > = 0>
bool mkdir_p(
  Path const & path
);

}
```

The `sst::mkdir_p` function ensures that `path` exists as a directory by creating any missing components as directories.

3.19.4. The `sst::read_whole_file` function

The `sst::read_whole_file` function reads the entire content of a file.

3.19.4.1. The `sst::read_whole_file` function (overload 1)

```
#include <sst/catalog/read_whole_file.hpp>
namespace sst {

template<
  class Dst,
  sst::enable_if_t<
    sst::is_output_iterator<Dst, unsigned char>::value
  > = 0
> Dst read_whole_file(
  char const * src,
  Dst dst
);

}
```

3.19.4.2. The `sst::read_whole_file` function (overload 2)

```
#include <sst/catalog/read_whole_file.hpp>
namespace sst {

template<
  class Dst = std::vector<unsigned char>
> Dst read_whole_file(
  char const * src
);

}
```

3.19.5. The `sst::write_whole_file` function

The `sst::write_whole_file` function writes a sequence of bytes to a file, overwriting the file if it already exists.

3.19.5.1. The `sst::write_whole_file` function (overload 1)

```
#include <sst/catalog/write_whole_file.hpp>
namespace sst {

template<
  class Src,
  class End,
  sst::enable_if_t<
    sst::is_byte_input_iterator<Src>::value
    && sst::is_sentinel<End, Src>::value
  > = 0
> void write_whole_file(
  Src src,
  End const & end,
  char const * dst,
  sst::write_whole_file_options const & options = {}
);

}
```

3.19.5.2. The `sst::write_whole_file` function (overload 2)

```
#include <sst/catalog/write_whole_file.hpp>
namespace sst {

template<
  class Src,
  sst::enable_if_t<
    sst::is_byte_input_iterable<Src>::value
  > = 0
> void write_whole_file(
  Src const & src,
  char const * dst,
  sst::write_whole_file_options const & options = {}
);

}
```

3.19.6. The `sst::write_whole_file_options` class

3.20. SFINAE tools

3.20.1. The `sst::enable_t` type

C++

```
#include <sst/catalog/enable_t.hpp>
namespace sst {

using enable_t = int;

}
```

3.20.2. The `sst::enable_if` metafunction

C++

```
#include <sst/catalog/enable_if.hpp>
namespace sst {

template<bool B, class T = sst::enable_t>
struct enable_if : std::enable_if<B, T> {};

template<bool B, class T = sst::enable_t>
using enable_if_t = typename enable_if<B, T>::type;

}
```

3.20.3. The `sst::first_t` alias template

C++

```
#include <sst/catalog/first_t.hpp>
namespace sst {

template<class T, class...>
using first_t = T;

}
```

The `sst::first_t` alias template yields its first template parameter.

3.20.4. `SST_COMPILES`

C++

```
#include <sst/catalog/SST_COMPILES.hpp>

#define SST_COMPILES(...) (x)
```

The `SST_COMPILES` macro expands to a parenthesized constant expression `(x)` that includes the

expression `__VA_ARGS__` as an unevaluated operand and always has value `true`.

3.20.5. The `SST_DEFINE_BOOLEAN_TRAIT_1` macro

C++

```
#include <sst/catalog/SST_DEFINE_BOOLEAN_TRAIT_1.hpp>

#define SST_DEFINE_BOOLEAN_TRAIT_1(name, T1, expr)                    \
                                                                       \
  template<class, class = sst::enable_t>                               \
  struct name : std::false_type {};                                    \
                                                                       \
  template<class T1>                                                   \
  struct name<T1, typename sst::enable_if<(expr)>::type>               \
      : std::true_type {};
```

3.20.6. The `SST_DEFINE_BOOLEAN_TRAIT_2` macro

C++

```
#include <sst/catalog/SST_DEFINE_BOOLEAN_TRAIT_2.hpp>

#define SST_DEFINE_BOOLEAN_TRAIT_2(name, T1, T2, expr)                \
                                                                       \
  template<class, class, class = sst::enable_t>                        \
  struct name : std::false_type {};                                    \
                                                                       \
  template<class T1, class T2>                                         \
  struct name<T1, T2, typename sst::enable_if<(expr)>::type>           \
      : std::true_type {};
```

3.20.7. `sst::dependent_true`

C++

```
#include <sst/catalog/dependent_true.hpp>
namespace sst {

template<class...>
struct dependent_true
    : std::integral_constant<bool, v> {};

}
```

The `sst::dependent_true` trait defines `v` to `true`. This is useful when the constant value `true` is needed but must be made dependent on one or more template parameters to delay evaluation.

3.20.8. `sst::dependent_false`

C++

```
#include <sst/catalog/dependent_false.hpp>
namespace sst {

template<class...>
struct dependent_false
    : std::integral_constant<bool, v> {};

}
```

The `sst::dependent_false` trait defines `v` to `false`. This is useful when the constant value `false` is needed but must be made dependent on one or more template parameters to delay evaluation.

3.21. Synchronization utilities

3.21.1. The `sst::cooldown_mutex` class

C++

```
#include <sst/catalog/cooldown_mutex.hpp>
namespace sst {

class cooldown_mutex;

}
```

The `sst::cooldown_mutex` class provides a mutex that, after being unlocked, cannot be locked again until a specified amount of time, called the cooldown, has passed. The cooldown does not apply to the first lock, i.e., after the mutex is constructed, it can be locked immediately.

3.21.2. The `sst::atomic_shared_ptr` class

C++

```
#include <sst/catalog/atomic_shared_ptr.hpp>
namespace sst {

template<class T>
class atomic_shared_ptr final {
public:
  atomic_shared_ptr() = default;
  ~atomic_shared_ptr() = default;

  atomic_shared_ptr(atomic_shared_ptr const &) = delete;
  atomic_shared_ptr(atomic_shared_ptr &&) = delete;
  void operator=(atomic_shared_ptr const &) = delete;
```

```
  void operator=(atomic_shared_ptr &&) = delete;

  atomic_shared_ptr(std::shared_ptr<T> const & ptr);
  void operator=(std::shared_ptr<T> const & ptr);
  operator std::shared_ptr<T>() const;

  void store(std::shared_ptr<T> const & ptr,
             std::memory_order mo = std::memory_order_seq_cst);

  std::shared_ptr<T>
  load(std::memory_order mo = std::memory_order_seq_cst) const;
};

}
```

The `sst::atomic_shared_ptr` class is an atomic wrapper for the `std::shared_ptr` class.

This class was added to the library because `std::atomic<std::shared_ptr>` is not available before C++20.

3.22. Random number utilities

3.22.1. The `sstByteRng` requirement

sstByteRng requirement A type `G` satisfies the `sstByteRng` requirement if all of the following hold:

1. `G` satisfies the `UniformRandomBitGenerator` requirement.
2. `G::result_type` is `unsigned char`.
3. `G::min()` is zero.
4. `G::max()` is `sst::uchar_max_v`.
5. If `g` is an object of type `G`, `p` is an output iterator that accepts `unsigned char`, and `n` is a nonnegative integer, then the expression `g(p, n)` writes `n` random bytes to `p` and returns the updated `p`.

   This functionality can always be implemented using `operator()()` as follows:

   ```
   template<class P, class N>
   P operator()(P p, N n) {
     while (n) {
       *p = operator()();
       ++p;
       --n;
     }
     return p;
   }
   ```

   However, it is typical that `G` will provide an implementation with higher performance.

3.22.2. The `sstCryptoRng` requirement

A type `G` satisfies the `sstCryptoRng` requirement if all of the following hold:

1. `G` satisfies the `UniformRandomBitGenerator` requirement.
2. If `g` is an object of type `G`, then the sequence of all random values produced by any overload of `g`'s `operator()` function throughout `g`'s lifetime is cryptographically secure.

3.22.3. The `sst::crypto_rng` function

3.22.3.1. Synopsis

```
#include <sst/catalog/crypto_rng.hpp>
namespace sst {

// Overload 1
sst::crypto_rng_t & crypto_rng();

// Overload 2
template<class... Args, sst::enable_if_t<(sizeof...(Args) > 0)> = 0>
R crypto_rng(Args &&... args);

}
```

3.22.3.2. Overload 1

```
#include <sst/catalog/crypto_rng.hpp>
namespace sst {

sst::crypto_rng_t & crypto_rng();

}
```

The `sst::crypto_rng` function (overload 1) returns a reference to a cryptographically secure random number generator `g` with thread storage duration.

Because the type of `g`, `sst::crypto_rng_t`, satisfies the `sstCryptoRng` requirement and the `sstCryptoRng` requirement is a superset of the `UniformRandomBitGenerator` requirement, `g` can be used in most situations where a standard random number generator, such as one of type `std::random_device` or `std::mt19937`, can be used.

Because `g` has thread storage duration, data races can be avoided regardless of multithreading by simply calling `sst::crypto_rng()` in every situation where a random number generator is needed.

Example 45.

sst-example-45.cpp

```
#include <iostream>
#include <random>

#include <sst/catalog/crypto_rng.hpp>

int main() {
  std::uniform_int_distribution<int> d(1, 100);
  std::cout << "Here is a random integer between ";
  std::cout << d.min() << " and " << d.max() << ": ";
  std::cout << d(sst::crypto_rng()) << "\n";
}
```

3.22.3.3. Overload 2

```
#include <sst/catalog/crypto_rng.hpp>
namespace sst {

template<
  class... Args,
  sst::enable_if_t<
    (sizeof...(Args) > 0U)
  > = 0>
auto crypto_rng(Args &&... args)
    -> decltype(sst::crypto_rng()(std::forward<Args>(args)...));

}
```

The `sst::crypto_rng` function (overload 2) is a convenience overload that is equivalent to `sst::crypto_rng()(std::forward<Args>(args)...)`.

Example 46.

The following program uses `sst::crypto_rng` (overload 2) to generate 16 random bytes.

sst-example-46.cpp

```
#include <iostream>
#include <vector>

#include <sst/catalog/crypto_rng.hpp>
#include <sst/catalog/to_hex.hpp>

int main() {
  std::vector<unsigned char> v(16);
  sst::crypto_rng(v.data(), v.size());
  std::cout << sst::to_hex(v) << "\n";
  return 0;
}
```

Compilation command

```
g++ sst-example-46.cpp -lsst
```

Example output

```
C4A3928E075C6536E9DA51DEC9B7F392
```

3.22.4. The `sst::crypto_rng_t` type

C++

```
#include <sst/catalog/crypto_rng_t.hpp>
namespace sst {

using crypto_rng_t = G;

}
```

The `sst::crypto_rng_t` type is an implementation-defined type `G` that satisfies the `sstByteRng` and `sstCryptoRng` requirements.

3.22.5. The `sst::openssl_rand_bytes_rng` class

C++

```
#include <sst/catalog/openssl_rand_bytes_rng.hpp>
namespace sst {

#if SST_WITH_OPENSSL_CRYPTO

class openssl_rand_bytes_rng final {
public:

  using result_type = unsigned char;

  static constexpr unsigned char min() noexcept {
    return 0;
  }

  static constexpr unsigned char max() noexcept {
    return sst::type_max<unsigned char>::value;
  }

  template<std::integral Size>
  explicit openssl_rand_bytes_rng(Size n);

  openssl_rand_bytes_rng();

  openssl_rand_bytes_rng(openssl_rand_bytes_rng &&) noexcept;
```

```
  openssl_rand_bytes_rng(openssl_rand_bytes_rng const &) = delete;
  openssl_rand_bytes_rng & operator=(openssl_rand_bytes_rng const &) =
delete;
  openssl_rand_bytes_rng & operator=(openssl_rand_bytes_rng &&) = delete;

  ~openssl_rand_bytes_rng() noexcept;

  void refill(bool eager = true);

  unsigned char operator()();

  template<class OutputIt, std::integral Size>
  OutputIt operator()(OutputIt dst, Size n);

};

#endif

}
```

3.23. Container utilities

3.23.1. The `sst::checked_reserve` function

C++

```
#include <sst/catalog/checked_reserve.hpp>
namespace sst {

template<linkhttps://en.cppreference.com/w/cpp/container[Container] C,
std::integral Size>
void checked_reserve(C & container, Size const size);

}
```

The `sst::checked_reserve` function is equivalent to calling `container.reserve(size)`, except that if `size` cannot be represented by `typename C::size_type`, an `sst::checked_overflow` will be thrown.

3.23.2. The `sst::checked_resize` function

C++

```
#include <sst/catalog/checked_resize.hpp>
namespace sst {

template<linkhttps://en.cppreference.com/w/cpp/container[Container] C,
std::integral Size>
void checked_resize(C & container, Size const size);
```

```
}
```

The `sst::checked_resize` function is equivalent to calling `container.resize(size)`, except that if `size` cannot be represented by `typename C::size_type`, an `sst::checked_overflow` will be thrown.

3.24. Networking utilities

3.24.1. The `sst::socket` class

3.24.1.1. Synopsis

```
#include <sst/catalog/socket.hpp>
namespace sst {

class socket {
public:

};

}
```

3.24.1.2. Thread safety

3.24.1.3. Examples

```
#include <future>
#include <iostream>
#include <ostream>
#include <string>

#include <sst/catalog/socket.hpp>
#include <sst/catalog/socket_shutdown.hpp>

int main(int argc, char ** argv) {

  // Initialize the socket library, if necessary.
  sst::socket_library_scope socket_library_scope;

  sst::socket socket;
  if (argc > 1) {
    // Run as the server if there's a command-line argument.
    sst::socket listener;
    listener.reuseaddr(true);
    listener.listen("127.0.0.1", 12345);
    socket = listener.accept();
```

```
  } else {
    // Run as the client if there's no command-line argument.
    socket.connect("127.0.0.1", 12345);
  }

  // In a background thread, read data from standard input and send it
  // over the socket.
  auto background = std::async(std::launch::async, [&socket] {
    std::string line;
    while (std::getline(std::cin, line)) {
      line += '\n';
      socket.send_exactly(&line[0], line.size());
    }
    socket.shutdown(sst::socket_shutdown::send());
  });

  // In the main thread, receive data from the socket and write it to
  // standard output.
  std::string buf;
  while (!socket.eof()) {
    buf.resize(100);
    auto const n = socket.recv_some(&buf[0], buf.size());
    buf.resize(n);
    std::cout << buf << std::flush;
  }

  // Wait for the background thread to finish.
  background.get();

}
```

3.24.2. The `sst::socket_library_scope` type

```
#include <sst/catalog/socket_library_scope.hpp>
namespace sst {

using socket_library_scope = implementation-defined;

}
```

The `sst::socket_library_scope` type can be used to initialize and shut down the underlying socket library, if the library requires it (e.g., Winsock).

When an `sst::socket_library_scope` object is constructed without arguments, it initializes the underlying socket library, throwing an exception if an error occurs. When the object is destructed, it shuts down the underlying socket library, ignoring any errors.

If the underlying socket library does not require any initialization or shutting down, the `sst::socket_library_scope` constructor and destructor are noops.

Example 47.

sst-example-47.cpp

```
#include <iostream>

#include <sst/catalog/socket.hpp>
#include <sst/catalog/socket_library_scope.hpp>

int main() {
  sst::socket_library_scope scope;
  std::cout << "Connecting to www.stealthsoftwareinc.com.\n";
  sst::socket s;
  s.connect("www.stealthsoftwareinc.com", 443);
  std::cout << "Connected.\n";
}
```

Compilation command

```
g++ sst-example-47.cpp -lsst
```

Possible output

```
Connecting to www.stealthsoftwareinc.com.
Connected.
```

3.24.3. The `sst::socket_poll_set` class

```
#include <sst/catalog/socket_poll_set.hpp>
namespace sst {

using socket_poll_set = implementation-defined;

}
```

Example 48.

This example implements an echo server.

sst-example-48.cpp

```
#include <cstring>
#include <iostream>
#include <map>
#include <memory>
#include <utility>
#include <vector>

#include <sst/catalog/socket.hpp>
#include <sst/catalog/socket_library_scope.hpp>
#include <sst/catalog/socket_poll_set.hpp>
#include <sst/catalog/socket_shutdown.hpp>
```

```
// TODO: Handle socket exceptions better.
int main() {
  sst::socket_library_scope socket_library_scope;
  sst::socket listener;
  listener.listen(12345);
  sst::socket_poll_set poll_set;
  poll_set.insert(&listener);
  struct context_t {
    sst::socket socket;
    std::vector<unsigned char> buf;
    std::vector<unsigned char>::size_type len = 0;
    context_t() : buf(1024) {
    }
  };
  std::map<sst::socket *, std::unique_ptr<context_t>> ctxs;
  while (true) {
    poll_set.poll();
    for (auto const & entry : poll_set) {
      if (&entry.socket() == &listener) {
        if (entry.can_recv()) {
          std::unique_ptr<context_t> ctx;
          {
            sst::socket s = entry.socket().accept();
            ctx.reset(new context_t);
            ctx->socket = std::move(s);
          }
          sst::socket & s = ctx->socket;
          s.blocking(false);
          ctxs.emplace(&s, std::move(ctx));
          poll_set.insert(&s);
        }
        continue;
      }
      context_t & ctx = *ctxs[&entry.socket()];
      if (entry.can_recv()) {
        ctx.len += ctx.socket.recv_some(&ctx.buf[ctx.len],
                                        ctx.buf.size() - ctx.len);
      }
      if (entry.can_send()) {
        auto const n = ctx.socket.send_some(&ctx.buf[0], ctx.len);
        std::memmove(&ctx.buf[0], &ctx.buf[n], ctx.len - n);
        ctx.len -= n;
        if (ctx.len == 0) {
          ctx.len += ctx.socket.recv_some(&ctx.buf[ctx.len],
                                          ctx.buf.size() - ctx.len);
          if (ctx.socket.eof()) {
            poll_set.erase(&ctx.socket);
            ctx.socket.shutdown(sst::socket_shutdown::both());
            ctx.socket.close();
            ctxs.erase(&ctx.socket);
          }
        }
      }
```

```
        }
      }
    }
```

Compilation command

```
g++ sst-example-48.cpp -lsst
```

3.25. Windows utilities

3.25.1. The `sst::gle_error_string` function

C++

```
#include <sst/catalog/gle_error_string.hpp>
namespace sst {
#if SST_WITH_WINDOWS_KERNEL32

std::string gle_error_string(sst_w32_DWORD code);

#endif
}
```

The `sst::gle_error_string` function returns a string representing the `GetLastError` code `code`.

3.25.2. The `sst::gle_exception` class

C++

```
#include <sst/catalog/gle_exception.hpp>
namespace sst {

class gle_exception : public std::runtime_error {
public:

  explicit gle_exception(std::string const & message,
                         sst_w32_DWORD code);
  explicit gle_exception(char const * message, sst_w32_DWORD code);
  explicit gle_exception(std::string const & message);
  explicit gle_exception(char const * message);
  explicit gle_exception(sst_w32_DWORD code);
  explicit gle_exception();

  sst_w32_DWORD code() const noexcept;

};

}
```

If `message` is a null pointer, the behavior is undefined. Otherwise, if `message` is empty, it will be adjusted to `sst::gle_error_string(code) + "."`. Otherwise, if `message` is a function name, it will be adjusted to `message + "() failed: " + sst::gle_error_string(code) + "."`. Otherwise, `message` will be adjusted to `message + ": " + sst::gle_error_string(code) + "."`.

Omitting `code` is equivalent to passing `GetLastError()`.

3.25.3. The `sst_w32_accept` function

C

```
#include <sst/catalog/sst_w32_accept.h>
#if SST_WITH_WINDOWS_WS2_32

sst_w32_SOCKET sst_w32_accept(
  sst_w32_SOCKET s,
  sst_w32_SOCKADDR * addr,
  int * addrlen
);

#endif
```

The `sst_w32_accept` function is equivalent to the `accept` function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.4. The `sst_w32_ADDRESS_FAMILY` type

C

```
#include <sst/catalog/sst_w32_ADDRESS_FAMILY.h>
#if SST_WITH_WINDOWS_WS2_32

typedef ADDRESS_FAMILY sst_w32_ADDRESS_FAMILY;

#endif
```

The `sst_w32_ADDRESS_FAMILY` type is equivalent to the `ADDRESS_FAMILY` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.5. The `sst_w32_ADDRINFOA` type

C

```
#include <sst/catalog/sst_w32_ADDRINFOA.h>
#if SST_WITH_WINDOWS_WS2_32

typedef ADDRINFOA sst_w32_ADDRINFOA;
```

```
#endif
```

The sst_w32_ADDRINFOA type is equivalent to the ADDRINFOA type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.6. The sst_w32_ADDRINFOA_unwrap function

C

```
#include <sst/catalog/sst_w32_ADDRINFOA_unwrap.h>
#if SST_WITH_WINDOWS_WS2_32

void sst_w32_ADDRINFOA_unwrap(
  sst_w32_ADDRINFOA_wrapper const * src,
  sst_w32_ADDRINFOA * dst
);

#endif
```

The sst_w32_ADDRINFOA_unwrap function converts an sst_w32_ADDRINFOA_wrapper to an sst_w32_ADDRINFOA.

All data in src that corresponds to data in dst must be fully initialized. In particular, array members must be fully initialized, even if they store sparse data such as null-terminated strings.

3.25.7. The sst_w32_ADDRINFOA_wrap function

C

```
#include <sst/catalog/sst_w32_ADDRINFOA_wrap.h>
#if SST_WITH_WINDOWS_WS2_32

void sst_w32_ADDRINFOA_wrap(
  sst_w32_ADDRINFOA const * src,
  sst_w32_ADDRINFOA_wrapper * dst
);

#endif
```

The sst_w32_ADDRINFOA_wrap function converts an sst_w32_ADDRINFOA to an sst_w32_ADDRINFOA_wrapper.

All data in src that corresponds to data in dst must be fully initialized. In particular, array members must be fully initialized, even if they store sparse data such as null-terminated strings.

3.25.8. The sst_w32_ADDRINFOA_wrapper type

C

```
#include <sst/catalog/sst_w32_ADDRINFOA_wrapper.h>
```

```
#if SST_WITH_WINDOWS_WS2_32

typedef struct sst_w32_ADDRINFOA_wrapper {
  int ai_flags;
  int ai_family;
  int ai_socktype;
  int ai_protocol;
  size_t ai_addrlen;
  char * ai_canonname;
  sst_w32_SOCKADDR * ai_addr;
  sst_w32_ADDRINFOA * ai_next;
} sst_w32_ADDRINFOA_wrapper;

#endif
```

The `sst_w32_ADDRINFOA_wrapper` type wraps the `ADDRINFOA` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

The `sst_w32_ADDRINFOA_wrap` and `sst_w32_ADDRINFOA_unwrap` functions can be used to convert back and forth between the `ADDRINFOA` and `sst_w32_ADDRINFOA_wrapper` types. The types do not necessarily have the same ABI, so they cannot be type punned.

3.25.9. The `SST_W32_AF_INET` constant

C

```
#include <sst/catalog/SST_W32_AF_INET.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_AF_INET (AF_INET)

#endif
```

The `SST_W32_AF_INET` constant is equivalent to the `AF_INET` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.10. The `SST_W32_AF_INET6` constant

C

```
#include <sst/catalog/SST_W32_AF_INET6.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_AF_INET6 (AF_INET6)

#endif
```

The `SST_W32_AF_INET6` constant is equivalent to the `AF_INET6` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.11. The SST_W32_AF_UNSPEC constant

C

```
#include <sst/catalog/SST_W32_AF_UNSPEC.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_AF_UNSPEC (AF_UNSPEC)

#endif
```

The SST_W32_AF_UNSPEC constant is equivalent to the AF_UNSPEC constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.12. The SST_W32_AI_PASSIVE constant

C

```
#include <sst/catalog/SST_W32_AI_PASSIVE.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_AI_PASSIVE (AI_PASSIVE)

#endif
```

The SST_W32_AI_PASSIVE constant is equivalent to the AI_PASSIVE constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.13. The sst_w32_bind function

C

```
#include <sst/catalog/sst_w32_bind.h>
#if SST_WITH_WINDOWS_WS2_32

int sst_w32_bind(
  sst_w32_SOCKET s,
  sst_w32_SOCKADDR const * name,
  int namelen
);

#endif
```

The sst_w32_bind function is equivalent to the bind function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.14. The sst_w32_CHAR type

C

```
#include <sst/catalog/sst_w32_CHAR.h>
#if SST_WITH_WINDOWS_KERNEL32

typedef CHAR sst_w32_CHAR;

#endif
```

The sst_w32_CHAR type is equivalent to the CHAR type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.15. The sst_w32_closesocket function

C

```
#include <sst/catalog/sst_w32_closesocket.h>
#if SST_WITH_WINDOWS_WS2_32

int sst_w32_closesocket(
  sst_w32_SOCKET s
);

#endif
```

The sst_w32_closesocket function is equivalent to the closesocket function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.16. The sst_w32_connect function

C

```
#include <sst/catalog/sst_w32_connect.h>
#if SST_WITH_WINDOWS_WS2_32

int sst_w32_connect(
  sst_w32_SOCKET s,
  sst_w32_SOCKADDR const * name,
  int namelen
);

#endif
```

The sst_w32_connect function is equivalent to the connect function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.17. The sst_w32_DWORD type

C

```
#include <sst/catalog/sst_w32_DWORD.h>
#if SST_WITH_WINDOWS_KERNEL32

typedef DWORD sst_w32_DWORD;

#endif
```

The `sst_w32_DWORD` type is equivalent to the `DWORD` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.18. The `SST_W32_FIONBIO` constant

C

```
#include <sst/catalog/SST_W32_FIONBIO.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_FIONBIO (FIONBIO)

#endif
```

The `SST_W32_FIONBIO` constant is equivalent to the `FIONBIO` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.19. The `sst_w32_freeaddrinfo` function

C

```
#include <sst/catalog/sst_w32_freeaddrinfo.h>
#if SST_WITH_WINDOWS_WS2_32

void sst_w32_freeaddrinfo(
  sst_w32_ADDRINFOA * pAddrInfo
);

#endif
```

The `sst_w32_freeaddrinfo` function is equivalent to the `freeaddrinfo` function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.20. The `sst_w32_getaddrinfo` function

C

```
#include <sst/catalog/sst_w32_getaddrinfo.h>
#if SST_WITH_WINDOWS_WS2_32
```

```
sst_w32_INT sst_w32_getaddrinfo(
  char const * pNodeName,
  char const * pServiceName,
  sst_w32_ADDRINFOA const * pHints,
  sst_w32_ADDRINFOA ** ppResult
);

#endif
```

The `sst_w32_getaddrinfo` function is equivalent to the `getaddrinfo` function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.21. The `sst_w32_getsockname` function

C

```
#include <sst/catalog/sst_w32_getsockname.h>
#if SST_WITH_WINDOWS_WS2_32

int sst_w32_getsockname(
  sst_w32_SOCKET s,
  sst_w32_SOCKADDR * name,
  int * namelen
);

#endif
```

The `sst_w32_getsockname` function is equivalent to the `getsockname` function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.22. The `sst_w32_getsockopt` function

C

```
#include <sst/catalog/sst_w32_getsockopt.h>
#if SST_WITH_WINDOWS_WS2_32

int sst_w32_getsockopt(
  sst_w32_SOCKET s,
  int level,
  int optname,
  char * optval,
  int * optlen
);

#endif
```

The `sst_w32_getsockopt` function is equivalent to the `getsockopt` function from the Windows

API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.23. The SST_W32_INET_ADDRSTRLEN constant

C

```
#include <sst/catalog/SST_W32_INET_ADDRSTRLEN.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_INET_ADDRSTRLEN (INET_ADDRSTRLEN)

#endif
```

The SST_W32_INET_ADDRSTRLEN constant is equivalent to the INET_ADDRSTRLEN constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.24. The sst_w32_inet_ntop function

C

```
#include <sst/catalog/sst_w32_inet_ntop.h>
#if SST_WITH_WINDOWS_WS2_32

sst_w32_PCSTR sst_w32_inet_ntop(
  sst_w32_INT Family,
  sst_w32_VOID const * pAddr,
  sst_w32_PSTR pStringBuf,
  size_t StringBufSize
);

#endif
```

The sst_w32_inet_ntop function is equivalent to the inet_ntop function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.25. The SST_W32_INET6_ADDRSTRLEN constant

C

```
#include <sst/catalog/SST_W32_INET6_ADDRSTRLEN.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_INET6_ADDRSTRLEN (INET6_ADDRSTRLEN)

#endif
```

The SST_W32_INET6_ADDRSTRLEN constant is equivalent to the INET6_ADDRSTRLEN constant from

the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.26. The `sst_w32_INT` type

C

```
#include <sst/catalog/sst_w32_INT.h>
#if SST_WITH_WINDOWS_KERNEL32

typedef INT sst_w32_INT;

#endif
```

The `sst_w32_INT` type is equivalent to the `INT` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.27. The `SST_W32_INVALID_SOCKET` constant

C

```
#include <sst/catalog/SST_W32_INVALID_SOCKET.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_INVALID_SOCKET (INVALID_SOCKET)

#endif
```

The `SST_W32_INVALID_SOCKET` constant is equivalent to the `INVALID_SOCKET` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.28. The `sst_w32_ioctlsocket` function

C

```
#include <sst/catalog/sst_w32_ioctlsocket.h>
#if SST_WITH_WINDOWS_WS2_32

int sst_w32_ioctlsocket(
  sst_w32_SOCKET s,
  long cmd,
  sst_w32_u_long * argp
);

#endif
```

The `sst_w32_ioctlsocket` function is equivalent to the `ioctlsocket` function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.29. The SST_W32_IPPROTO_TCP constant

C

```
#include <sst/catalog/SST_W32_IPPROTO_TCP.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_IPPROTO_TCP (IPPROTO_TCP)

#endif
```

The SST_W32_IPPROTO_TCP constant is equivalent to the IPPROTO_TCP constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.30. The SST_W32_IPPROTO_UDP constant

C

```
#include <sst/catalog/SST_W32_IPPROTO_UDP.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_IPPROTO_UDP (IPPROTO_UDP)

#endif
```

The SST_W32_IPPROTO_UDP constant is equivalent to the IPPROTO_UDP constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.31. The sst_w32_LANGID type

C

```
#include <sst/catalog/sst_w32_LANGID.h>
#if SST_WITH_WINDOWS_KERNEL32

typedef LANGID sst_w32_LANGID;

#endif
```

The sst_w32_LANGID type is equivalent to the LANGID type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.32. The sst_w32_listen function

C

```
#include <sst/catalog/sst_w32_listen.h>
```

```
#if SST_WITH_WINDOWS_WS2_32

int sst_w32_listen(
  sst_w32_SOCKET s,
  int backlog
);

#endif
```

The `sst_w32_listen` function is equivalent to the `listen` function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.33. The `sst_w32_ntohl` function

C

```
#include <sst/catalog/sst_w32_ntohl.h>
#if SST_WITH_WINDOWS_WS2_32

sst_w32_u_long sst_w32_ntohl(
  sst_w32_u_long netlong
);

#endif
```

The `sst_w32_ntohl` function is equivalent to the `ntohl` function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.34. The `sst_w32_ntohs` function

C

```
#include <sst/catalog/sst_w32_ntohs.h>
#if SST_WITH_WINDOWS_WS2_32

sst_w32_u_short sst_w32_ntohs(
  sst_w32_u_short netshort
);

#endif
```

The `sst_w32_ntohs` function is equivalent to the `ntohs` function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.35. The `sst_w32_offsetof_SOCKADDR_STORAGE_ss_family` constant

C

```
#include <sst/catalog/sst_w32_offsetof_SOCKADDR_STORAGE_ss_family.h>
#if SST_WITH_WINDOWS_WS2_32

size_t const sst_w32_offsetof_SOCKADDR_STORAGE_ss_family = offsetof
(SOCKADDR_STORAGE, ss_family);

#endif
```

The `sst_w32_offsetof_SOCKADDR_STORAGE_ss_family` constant is equivalent to the `offsetof (SOCKADDR_STORAGE, ss_family)` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.36. The `sst_w32_PCSTR` type

C

```
#include <sst/catalog/sst_w32_PCSTR.h>
#if SST_WITH_WINDOWS_KERNEL32

typedef PCSTR sst_w32_PCSTR;

#endif
```

The `sst_w32_PCSTR` type is equivalent to the `PCSTR` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.37. The `SST_W32_POLLRDBAND` constant

C

```
#include <sst/catalog/SST_W32_POLLRDBAND.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_POLLRDBAND (POLLRDBAND)

#endif
```

The `SST_W32_POLLRDBAND` constant is equivalent to the `POLLRDBAND` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.38. The `SST_W32_POLLRDNORM` constant

C

```
#include <sst/catalog/SST_W32_POLLRDNORM.h>
#if SST_WITH_WINDOWS_WS2_32
```

```
#define SST_W32_POLLRDNORM (POLLRDNORM)

#endif
```

The SST_W32_POLLRDNORM constant is equivalent to the POLLRDNORM constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.39. The SST_W32_POLLWRNORM constant

C

```
#include <sst/catalog/SST_W32_POLLWRNORM.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_POLLWRNORM (POLLWRNORM)

#endif
```

The SST_W32_POLLWRNORM constant is equivalent to the POLLWRNORM constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.40. The sst_w32_PSTR type

C

```
#include <sst/catalog/sst_w32_PSTR.h>
#if SST_WITH_WINDOWS_KERNEL32

typedef PSTR sst_w32_PSTR;

#endif
```

The sst_w32_PSTR type is equivalent to the PSTR type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.41. The sst_w32_recv function

C

```
#include <sst/catalog/sst_w32_recv.h>
#if SST_WITH_WINDOWS_WS2_32

int sst_w32_recv(
  sst_w32_SOCKET s,
  char * buf,
  int len,
  int flags
);
```

```
#endif
```

The sst_w32_recv function is equivalent to the recv function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.42. The SST_W32_SD_BOTH constant

C

```
#include <sst/catalog/SST_W32_SD_BOTH.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_SD_BOTH (SD_BOTH)

#endif
```

The SST_W32_SD_BOTH constant is equivalent to the SD_BOTH constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.43. The SST_W32_SD_RECEIVE constant

C

```
#include <sst/catalog/SST_W32_SD_RECEIVE.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_SD_RECEIVE (SD_RECEIVE)

#endif
```

The SST_W32_SD_RECEIVE constant is equivalent to the SD_RECEIVE constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.44. The SST_W32_SD_SEND constant

C

```
#include <sst/catalog/SST_W32_SD_SEND.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_SD_SEND (SD_SEND)

#endif
```

The SST_W32_SD_SEND constant is equivalent to the SD_SEND constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.45. The `sst_w32_send` function

C

```
#include <sst/catalog/sst_w32_send.h>
#if SST_WITH_WINDOWS_WS2_32

int sst_w32_send(
  sst_w32_SOCKET s,
  char const * buf,
  int len,
  int flags
);

#endif
```

The `sst_w32_send` function is equivalent to the `send` function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.46. The `sst_w32_setsockopt` function

C

```
#include <sst/catalog/sst_w32_setsockopt.h>
#if SST_WITH_WINDOWS_WS2_32

int sst_w32_setsockopt(
  sst_w32_SOCKET s,
  int level,
  int optname,
  const char * optval,
  int optlen
);

#endif
```

The `sst_w32_setsockopt` function is equivalent to the `setsockopt` function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.47. The `sst_w32_SHORT` type

C

```
#include <sst/catalog/sst_w32_SHORT.h>
#if SST_WITH_WINDOWS_KERNEL32

typedef SHORT sst_w32_SHORT;

#endif
```

The sst_w32_SHORT type is equivalent to the SHORT type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.48. The sst_w32_shutdown function

C

```
#include <sst/catalog/sst_w32_shutdown.h>
#if SST_WITH_WINDOWS_WS2_32

int sst_w32_shutdown(
  sst_w32_SOCKET s,
  int how
);

#endif
```

The sst_w32_shutdown function is equivalent to the shutdown function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.49. The SST_W32_SIZEOF_ADDRINFOA constant

C

```
#include <sst/catalog/SST_W32_SIZEOF_ADDRINFOA.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_SIZEOF_ADDRINFOA (/*...*/)

#endif
```

The SST_W32_SIZEOF_ADDRINFOA constant is equivalent to the sizeof(ADDRINFOA) constant from the Windows API, except it can also be used in the preprocessor, where it is an unsigned integer. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.50. The sst_w32_sizeof_SOCKADDR_IN constant

C

```
#include <sst/catalog/sst_w32_sizeof_SOCKADDR_IN.h>
#if SST_WITH_WINDOWS_WS2_32

size_t const sst_w32_sizeof_SOCKADDR_IN = sizeof(sockaddr_in);

#endif
```

The sst_w32_sizeof_SOCKADDR_IN constant is equivalent to the sizeof(sockaddr_in) constant from the Windows API. It does not require any Windows API headers to be included, which helps

reduce compilation difficulties.

3.25.51. The `sst_w32_sizeof_SOCKADDR_IN6` constant

C

```
#include <sst/catalog/sst_w32_sizeof_SOCKADDR_IN6.h>
#if SST_WITH_WINDOWS_WS2_32

size_t const sst_w32_sizeof_SOCKADDR_IN6 = sizeof(sockaddr_in6);

#endif
```

The `sst_w32_sizeof_SOCKADDR_IN6` constant is equivalent to the `sizeof(sockaddr_in6)` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.52. The `sst_w32_sizeof_SOCKADDR_STORAGE` constant

C

```
#include <sst/catalog/sst_w32_sizeof_SOCKADDR_STORAGE.h>
#if SST_WITH_WINDOWS_WS2_32

size_t const sst_w32_sizeof_SOCKADDR_STORAGE = sizeof(SOCKADDR_STORAGE);

#endif
```

The `sst_w32_sizeof_SOCKADDR_STORAGE` constant is equivalent to the `sizeof(SOCKADDR_STORAGE)` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.53. The `SST_W32_SOCKET_ERROR` constant

C

```
#include <sst/catalog/SST_W32_SOCKET_ERROR.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_SOCKET_ERROR (POLLRDBAND)

#endif
```

The `SST_W32_SOCKET_ERROR` constant is equivalent to the `POLLRDBAND` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.54. The `sst_w32_sizeof_WIN32_FILE_ATTRIBUTE_DATA` constant

C

```
#include <sst/catalog/sst_w32_sizeof_WIN32_FILE_ATTRIBUTE_DATA.h>
#if SST_WITH_WINDOWS_KERNEL32

size_t const sst_w32_sizeof_WIN32_FILE_ATTRIBUTE_DATA =
sizeof(WIN32_FILE_ATTRIBUTE_DATA);

#endif
```

The `sst_w32_sizeof_WIN32_FILE_ATTRIBUTE_DATA` constant is equivalent to the `sizeof(WIN32_FILE_ATTRIBUTE_DATA)` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.55. The `sst_w32_sizeof_WIN32_FIND_DATAA` constant

C

```
#include <sst/catalog/sst_w32_sizeof_WIN32_FIND_DATAA.h>
#if SST_WITH_WINDOWS_KERNEL32

size_t const sst_w32_sizeof_WIN32_FIND_DATAA = sizeof(WIN32_FIND_DATAA);

#endif
```

The `sst_w32_sizeof_WIN32_FIND_DATAA` constant is equivalent to the `sizeof(WIN32_FIND_DATAA)` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.56. The `sst_w32_sizeof_WIN32_FIND_DATAW` constant

C

```
#include <sst/catalog/sst_w32_sizeof_WIN32_FIND_DATAW.h>
#if SST_WITH_WINDOWS_KERNEL32

size_t const sst_w32_sizeof_WIN32_FIND_DATAW = sizeof(WIN32_FIND_DATAW);

#endif
```

The `sst_w32_sizeof_WIN32_FIND_DATAW` constant is equivalent to the `sizeof(WIN32_FIND_DATAW)` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.57. The `sst_w32_sizeof_WSADATA` constant

C

```
#include <sst/catalog/sst_w32_sizeof_WSADATA.h>
#if SST_WITH_WINDOWS_WS2_32

size_t const sst_w32_sizeof_WSADATA = sizeof(WSADATA);

#endif
```

The `sst_w32_sizeof_WSADATA` constant is equivalent to the `sizeof(WSADATA)` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.58. The `SST_W32_SIZEOF_WSAPOLLFD` constant

C

```
#include <sst/catalog/SST_W32_SIZEOF_WSAPOLLFD.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_SIZEOF_WSAPOLLFD (/*...*/)

#endif
```

The `SST_W32_SIZEOF_WSAPOLLFD` constant is equivalent to the `sizeof(WSAPOLLFD)` constant from the Windows API, except it can also be used in the preprocessor, where it is an unsigned integer. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.59. The `SST_W32_SO_REUSEADDR` constant

C

```
#include <sst/catalog/SST_W32_SO_REUSEADDR.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_SO_REUSEADDR (SO_REUSEADDR)

#endif
```

The `SST_W32_SO_REUSEADDR` constant is equivalent to the `SO_REUSEADDR` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.60. The `SST_W32_SOCK_DGRAM` constant

C

```
#include <sst/catalog/SST_W32_SOCK_DGRAM.h>
#if SST_WITH_WINDOWS_WS2_32
```

```
#define SST_W32_SOCK_DGRAM (SOCK_DGRAM)

#endif
```

The SST_W32_SOCK_DGRAM constant is equivalent to the SOCK_DGRAM constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.61. The SST_W32_SOCK_STREAM constant

C

```
#include <sst/catalog/SST_W32_SOCK_STREAM.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_SOCK_STREAM (SOCK_STREAM)

#endif
```

The SST_W32_SOCK_STREAM constant is equivalent to the SOCK_STREAM constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.62. The sst_w32_SOCKADDR type

C

```
#include <sst/catalog/sst_w32_SOCKADDR.h>
#if SST_WITH_WINDOWS_WS2_32

typedef sockaddr sst_w32_SOCKADDR;

#endif
```

The sst_w32_SOCKADDR type is equivalent to the sockaddr type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.63. The sst_w32_SOCKADDR_IN type

C

```
#include <sst/catalog/sst_w32_SOCKADDR_IN.h>
#if SST_WITH_WINDOWS_WS2_32

typedef sockaddr_in sst_w32_SOCKADDR_IN;

#endif
```

The sst_w32_SOCKADDR_IN type is equivalent to the sockaddr_in type from the Windows API. It

does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.64. The `sst_w32_SOCKADDR_IN_unwrap` function

C

```
#include <sst/catalog/sst_w32_SOCKADDR_IN_unwrap.h>
#if SST_WITH_WINDOWS_WS2_32

void sst_w32_SOCKADDR_IN_unwrap(
  sst_w32_SOCKADDR_IN_wrapper const * src,
  sst_w32_SOCKADDR_IN * dst
);

#endif
```

The `sst_w32_SOCKADDR_IN_unwrap` function converts an `sst_w32_SOCKADDR_IN_wrapper` to an `sst_w32_SOCKADDR_IN`.

All data in `src` that corresponds to data in `dst` must be fully initialized. In particular, array members must be fully initialized, even if they store sparse data such as null-terminated strings.

3.25.65. The `sst_w32_SOCKADDR_IN_wrap` function

C

```
#include <sst/catalog/sst_w32_SOCKADDR_IN_wrap.h>
#if SST_WITH_WINDOWS_WS2_32

void sst_w32_SOCKADDR_IN_wrap(
  sst_w32_SOCKADDR_IN const * src,
  sst_w32_SOCKADDR_IN_wrapper * dst
);

#endif
```

The `sst_w32_SOCKADDR_IN_wrap` function converts an `sst_w32_SOCKADDR_IN` to an `sst_w32_SOCKADDR_IN_wrapper`.

All data in `src` that corresponds to data in `dst` must be fully initialized. In particular, array members must be fully initialized, even if they store sparse data such as null-terminated strings.

3.25.66. The `sst_w32_SOCKADDR_IN_wrapper` type

C

```
#include <sst/catalog/sst_w32_SOCKADDR_IN_wrapper.h>

typedef struct sst_w32_SOCKADDR_IN_wrapper {
```

```
  sst_w32_ADDRESS_FAMILY sin_family;
  sst_w32_USHORT sin_port;
  sst_w32_ULONG sin_addr;
} sst_w32_SOCKADDR_IN_wrapper;
```

The `sst_w32_SOCKADDR_IN_wrapper` type wraps the `sockaddr_in` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

The `sst_w32_SOCKADDR_IN_wrap` and `sst_w32_SOCKADDR_IN_unwrap` functions can be used to convert back and forth between the `sockaddr_in` and `sst_w32_SOCKADDR_IN_wrapper` types. The types do not necessarily have the same ABI, so they cannot be type punned.

3.25.67. The `sst_w32_SOCKADDR_IN6` type

C

```
#include <sst/catalog/sst_w32_SOCKADDR_IN6.h>
#if SST_WITH_WINDOWS_WS2_32

typedef sockaddr_in6 sst_w32_SOCKADDR_IN6;

#endif
```

The `sst_w32_SOCKADDR_IN6` type is equivalent to the `sockaddr_in6` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.68. The `sst_w32_SOCKADDR_IN6_unwrap` function

C

```
#include <sst/catalog/sst_w32_SOCKADDR_IN6_unwrap.h>
#if SST_WITH_WINDOWS_WS2_32

void sst_w32_SOCKADDR_IN6_unwrap(
  sst_w32_SOCKADDR_IN6_wrapper const * src,
  sst_w32_SOCKADDR_IN6 * dst
);

#endif
```

The `sst_w32_SOCKADDR_IN6_unwrap` function converts an `sst_w32_SOCKADDR_IN6_wrapper` to an `sst_w32_SOCKADDR_IN6`.

All data in `src` that corresponds to data in `dst` must be fully initialized. In particular, array members must be fully initialized, even if they store sparse data such as null-terminated strings.

3.25.69. The `sst_w32_SOCKADDR_IN6_wrap` function

C

```
#include <sst/catalog/sst_w32_SOCKADDR_IN6_wrap.h>
#if SST_WITH_WINDOWS_WS2_32

void sst_w32_SOCKADDR_IN6_wrap(
  sst_w32_SOCKADDR_IN6 const * src,
  sst_w32_SOCKADDR_IN6_wrapper * dst
);

#endif
```

The `sst_w32_SOCKADDR_IN6_wrap` function converts an `sst_w32_SOCKADDR_IN6` to an `sst_w32_SOCKADDR_IN6_wrapper`.

All data in `src` that corresponds to data in `dst` must be fully initialized. In particular, array members must be fully initialized, even if they store sparse data such as null-terminated strings.

3.25.70. The `sst_w32_SOCKADDR_IN6_wrapper` type

C

```
#include <sst/catalog/sst_w32_SOCKADDR_IN6_wrapper.h>

typedef struct sst_w32_SOCKADDR_IN6_wrapper {
  sst_w32_ADDRESS_FAMILY sin6_family;
  sst_w32_USHORT sin6_port;
  sst_w32_ULONG sin6_flowinfo;
  unsigned char sin6_addr[16];
  sst_w32_ULONG sin6_scope_id;
} sst_w32_SOCKADDR_IN6_wrapper;
```

The `sst_w32_SOCKADDR_IN6_wrapper` type wraps the `sockaddr_in6` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

The `sst_w32_SOCKADDR_IN6_wrap` and `sst_w32_SOCKADDR_IN6_unwrap` functions can be used to convert back and forth between the `sockaddr_in6` and `sst_w32_SOCKADDR_IN6_wrapper` types. The types do not necessarily have the same ABI, so they cannot be type punned.

3.25.71. The `sst_w32_SOCKADDR_STORAGE` type

C

```
#include <sst/catalog/sst_w32_SOCKADDR_STORAGE.h>
#if SST_WITH_WINDOWS_WS2_32

typedef SOCKADDR_STORAGE sst_w32_SOCKADDR_STORAGE;

#endif
```

The `sst_w32_SOCKADDR_STORAGE` type is equivalent to the `SOCKADDR_STORAGE` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.72. The `sst_w32_SOCKET` type

C

```
#include <sst/catalog/sst_w32_SOCKET.h>
#if SST_WITH_WINDOWS_WS2_32

typedef SOCKET sst_w32_SOCKET;

#endif
```

The `sst_w32_SOCKET` type is equivalent to the `SOCKET` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.73. The `sst_w32_socket_f` function

C

```
#include <sst/catalog/sst_w32_socket_f.h>
#if SST_WITH_WINDOWS_WS2_32

sst_w32_SOCKET sst_w32_socket_f(
  int af,
  int type,
  int protocol
);

#endif
```

The `sst_w32_socket_f` function is equivalent to the `socket` function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.74. The `SST_W32_SOL_SOCKET` constant

C

```
#include <sst/catalog/SST_W32_SOL_SOCKET.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_SOL_SOCKET (SOL_SOCKET)

#endif
```

The `SST_W32_SOL_SOCKET` constant is equivalent to the `SOL_SOCKET` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation

difficulties.

3.25.75. The SST_W32_SOMAXCONN constant

C

```
#include <sst/catalog/SST_W32_SOMAXCONN.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_SOMAXCONN (SOMAXCONN)

#endif
```

The SST_W32_SOMAXCONN constant is equivalent to the SOMAXCONN constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.76. The SST_W32_TCP_NODELAY constant

C

```
#include <sst/catalog/SST_W32_TCP_NODELAY.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_TCP_NODELAY (TCP_NODELAY)

#endif
```

The SST_W32_TCP_NODELAY constant is equivalent to the TCP_NODELAY constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.77. The sst_w32_u_long type

C

```
#include <sst/catalog/sst_w32_u_long.h>
#if SST_WITH_WINDOWS_WS2_32

typedef u_long sst_w32_u_long;

#endif
```

The sst_w32_u_long type is equivalent to the u_long type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.78. The sst_w32_u_short type

C

```
#include <sst/catalog/sst_w32_u_short.h>
#if SST_WITH_WINDOWS_WS2_32

typedef u_short sst_w32_u_short;

#endif
```

The `sst_w32_u_short` type is equivalent to the `u_short` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.79. The `sst_w32_UINT_PTR` type

C

```
#include <sst/catalog/sst_w32_UINT_PTR.h>
#if SST_WITH_WINDOWS_KERNEL32

typedef UINT_PTR sst_w32_UINT_PTR;

#endif
```

The `sst_w32_UINT_PTR` type is equivalent to the `UINT_PTR` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.80. The `sst_w32_ULONG` type

C

```
#include <sst/catalog/sst_w32_ULONG.h>
#if SST_WITH_WINDOWS_KERNEL32

typedef ULONG sst_w32_ULONG;

#endif
```

The `sst_w32_ULONG` type is equivalent to the `ULONG` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.81. The `sst_w32_USHORT` type

C

```
#include <sst/catalog/sst_w32_USHORT.h>
#if SST_WITH_WINDOWS_KERNEL32

typedef USHORT sst_w32_USHORT;

#endif
```

The sst_w32_USHORT type is equivalent to the USHORT type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.82. The sst_w32_VOID type

C

```
#include <sst/catalog/sst_w32_VOID.h>
#if SST_WITH_WINDOWS_KERNEL32

typedef VOID sst_w32_VOID;

#endif
```

The sst_w32_VOID type is equivalent to the VOID type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.83. The sst_w32_WORD type

C

```
#include <sst/catalog/sst_w32_WORD.h>

typedef WORD sst_w32_WORD;
```

The sst_w32_WORD type is equivalent to the WORD type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.84. The sst_w32_WSACleanup function

C

```
#include <sst/catalog/sst_w32_WSACleanup.h>
#if SST_WITH_WINDOWS_WS2_32

int sst_w32_WSACleanup(
);

#endif
```

The sst_w32_WSACleanup function is equivalent to the WSACleanup function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.85. The sst_w32_WSADATA type

C

```
#include <sst/catalog/sst_w32_WSADATA.h>
#if SST_WITH_WINDOWS_WS2_32

typedef WSADATA sst_w32_WSADATA;

#endif
```

The `sst_w32_WSADATA` type is equivalent to the `WSADATA` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.86. The `sst_w32_WSADATA_unwrap` function

C

```
#include <sst/catalog/sst_w32_WSADATA_unwrap.h>
#if SST_WITH_WINDOWS_WS2_32

void sst_w32_WSADATA_unwrap(
  sst_w32_WSADATA_wrapper const * src,
  sst_w32_WSADATA * dst
);

#endif
```

The `sst_w32_WSADATA_unwrap` function converts an `sst_w32_WSADATA_wrapper` to an `sst_w32_WSADATA`.

All data in `src` that corresponds to data in `dst` must be fully initialized. In particular, array members must be fully initialized, even if they store sparse data such as null-terminated strings.

3.25.87. The `sst_w32_WSADATA_wrap` function

C

```
#include <sst/catalog/sst_w32_WSADATA_wrap.h>
#if SST_WITH_WINDOWS_WS2_32

void sst_w32_WSADATA_wrap(
  sst_w32_WSADATA const * src,
  sst_w32_WSADATA_wrapper * dst
);

#endif
```

The `sst_w32_WSADATA_wrap` function converts an `sst_w32_WSADATA` to an `sst_w32_WSADATA_wrapper`.

All data in `src` that corresponds to data in `dst` must be fully initialized. In particular, array members must be fully initialized, even if they store sparse data such as null-terminated strings.

3.25.88. The `sst_w32_WSADATA_wrapper` type

C

```
#include <sst/catalog/sst_w32_WSADATA_wrapper.h>

typedef struct sst_w32_WSADATA_wrapper {
  sst_w32_WORD wVersion;
  sst_w32_WORD wHighVersion;
  unsigned short iMaxSockets;
  unsigned short iMaxUdpDg;
  char * lpVendorInfo;
  char szDescription[SST_W32_WSADESCRIPTION_LEN + 1];
  char szSystemStatus[SST_W32_WSASYS_STATUS_LEN + 1];
} sst_w32_WSADATA_wrapper;
```

The `sst_w32_WSADATA_wrapper` type wraps the `WSADATA` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

The `sst_w32_WSADATA_wrap` and `sst_w32_WSADATA_unwrap` functions can be used to convert back and forth between the `WSADATA` and `sst_w32_WSADATA_wrapper` types. The types do not necessarily have the same ABI, so they cannot be type punned.

3.25.89. The `SST_W32_WSADESCRIPTION_LEN` constant

C

```
#include <sst/catalog/SST_W32_WSADESCRIPTION_LEN.h>

#define SST_W32_WSADESCRIPTION_LEN (WSADESCRIPTION_LEN)
```

The `SST_W32_WSADESCRIPTION_LEN` constant is equivalent to the `WSADESCRIPTION_LEN` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.90. The `SST_W32_WSAEALREADY` constant

C

```
#include <sst/catalog/SST_W32_WSAEALREADY.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_WSAEALREADY (WSAEALREADY)

#endif
```

The `SST_W32_WSAEALREADY` constant is equivalent to the `WSAEALREADY` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.91. The SST_W32_WSAEISCONN constant

C

```
#include <sst/catalog/SST_W32_WSAEISCONN.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_WSAEISCONN (WSAEISCONN)

#endif
```

The SST_W32_WSAEISCONN constant is equivalent to the WSAEISCONN constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.92. The SST_W32_WSAEWOULDBLOCK constant

C

```
#include <sst/catalog/SST_W32_WSAEWOULDBLOCK.h>
#if SST_WITH_WINDOWS_WS2_32

#define SST_W32_WSAEWOULDBLOCK (WSAEWOULDBLOCK)

#endif
```

The SST_W32_WSAEWOULDBLOCK constant is equivalent to the WSAEWOULDBLOCK constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.93. The sst_w32_WSAGetLastError function

C

```
#include <sst/catalog/sst_w32_WSAGetLastError.h>
#if SST_WITH_WINDOWS_WS2_32

int sst_w32_WSAGetLastError(
);

#endif
```

The sst_w32_WSAGetLastError function is equivalent to the WSAGetLastError function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.94. The sst_w32_WSAPoll function

C

```
#include <sst/catalog/sst_w32_WSAPoll.h>
#if SST_WITH_WINDOWS_WS2_32

int sst_w32_WSAPoll(
  sst_w32_WSAPOLLFD * fdArray,
  sst_w32_ULONG fds,
  sst_w32_INT timeout
);

#endif
```

The `sst_w32_WSAPoll` function is equivalent to the `WSAPoll` function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.95. The `sst_w32_WSAPOLLFD` type

C

```
#include <sst/catalog/sst_w32_WSAPOLLFD.h>
#if SST_WITH_WINDOWS_WS2_32

typedef WSAPOLLFD sst_w32_WSAPOLLFD;

#endif
```

The `sst_w32_WSAPOLLFD` type is equivalent to the `WSAPOLLFD` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.96. The `sst_w32_WSAPOLLFD_unwrap` function

C

```
#include <sst/catalog/sst_w32_WSAPOLLFD_unwrap.h>
#if SST_WITH_WINDOWS_WS2_32

void sst_w32_WSAPOLLFD_unwrap(
  sst_w32_WSAPOLLFD_wrapper const * src,
  sst_w32_WSAPOLLFD * dst
);

#endif
```

The `sst_w32_WSAPOLLFD_unwrap` function converts an `sst_w32_WSAPOLLFD_wrapper` to an `sst_w32_WSAPOLLFD`.

All data in `src` that corresponds to data in `dst` must be fully initialized. In particular, array members must be fully initialized, even if they store sparse data such as null-terminated strings.

3.25.97. The `sst_w32_WSAPOLLFD_wrap` function

C

```
#include <sst/catalog/sst_w32_WSAPOLLFD_wrap.h>
#if SST_WITH_WINDOWS_WS2_32

void sst_w32_WSAPOLLFD_wrap(
  sst_w32_WSAPOLLFD const * src,
  sst_w32_WSAPOLLFD_wrapper * dst
);

#endif
```

The `sst_w32_WSAPOLLFD_wrap` function converts an `sst_w32_WSAPOLLFD` to an `sst_w32_WSAPOLLFD_wrapper`.

All data in `src` that corresponds to data in `dst` must be fully initialized. In particular, array members must be fully initialized, even if they store sparse data such as null-terminated strings.

3.25.98. The `sst_w32_WSAPOLLFD_wrapper` type

C

```
#include <sst/catalog/sst_w32_WSAPOLLFD_wrapper.h>

typedef struct pollfd {
  sst_w32_SOCKET fd;
  sst_w32_SHORT events;
  sst_w32_SHORT revents;
} sst_w32_WSAPOLLFD_wrapper;
```

The `sst_w32_WSAPOLLFD_wrapper` type wraps the `WSAPOLLFD` type from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

The `sst_w32_WSAPOLLFD_wrap` and `sst_w32_WSAPOLLFD_unwrap` functions can be used to convert back and forth between the `WSAPOLLFD` and `sst_w32_WSAPOLLFD_wrapper` types. The types do not necessarily have the same ABI, so they cannot be type punned.

3.25.99. The `sst_w32_WSAStartup` function

C

```
#include <sst/catalog/sst_w32_WSAStartup.h>
#if SST_WITH_WINDOWS_WS2_32

int sst_w32_WSAStartup(
  sst_w32_WORD wVersionRequested,
  sst_w32_WSADATA * lpWSAData
);
```

```
#endif
```

The `sst_w32_WSAStartup` function is equivalent to the `WSAStartup` function from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.100. The `SST_W32_WSASYS_STATUS_LEN` constant

C

```
#include <sst/catalog/SST_W32_WSASYS_STATUS_LEN.h>

#define SST_W32_WSASYS_STATUS_LEN (WSASYS_STATUS_LEN)
```

The `SST_W32_WSASYS_STATUS_LEN` constant is equivalent to the `WSASYS_STATUS_LEN` constant from the Windows API. It does not require any Windows API headers to be included, which helps reduce compilation difficulties.

3.25.101. The `sst::windows_langid` enumeration

C++

```
#include <sst/catalog/windows_langid.hpp>
namespace sst {

enum class windows_langid : T {

  en_us = 0x0409,

};

}
```

The `sst::windows_langid` enumeration provides Windows language ID constants. The underlying type `T` is an unspecified unsigned fundamental integer type[3] with at least 16-bit range.

3.25.102. The `sst::wsa_error_string` function

C++

```
#include <sst/catalog/wsa_error_string.hpp>
namespace sst {
#if SST_WITH_WINDOWS_WS2_32

std::string wsa_error_string(int code);

#endif
}
```

The `sst::wsa_error_string` function returns a string representing the Windows Sockets error

code code.

3.25.103. The `sst::windows_socket_library_scope` class

C++

```
#include <sst/catalog/windows_socket_library_scope.hpp>
namespace sst {

class windows_socket_library_scope final {
public:

  explicit windows_socket_library_scope(int major, int minor);

  explicit windows_socket_library_scope();

  sst_w32_WSADATA_wrapper const & data() const;

  ~windows_socket_library_scope() noexcept;

  windows_socket_library_scope(windows_socket_library_scope const &) =
delete;
  windows_socket_library_scope(windows_socket_library_scope &&) = delete;
  windows_socket_library_scope & operator=(windows_socket_library_scope const
&) = delete;
  windows_socket_library_scope & operator=(windows_socket_library_scope &&) =
delete;
};

}
```

The `sst::windows_socket_library_scope` class calls the `WSAStartup` function when it is constructed and the `WSACleanup` function when it is destructed.

If Winsock is not available, i.e., if `SST_WITH_WINDOWS_WS2_32` is `0`, this class does nothing.

3.25.103.1. The `sst::windows_socket_library_scope` (overload 1) constructor

```
explicit windows_socket_library_scope(int major, int minor);
```

The `sst::windows_socket_library_scope` (overload 1) constructor calls the `WSAStartup` function and requests the Winsock version specified by `major` and `minor`.

If Winsock is not available, i.e., if `SST_WITH_WINDOWS_WS2_32` is `0`, this constructor does nothing.

3.25.103.2. The `sst::windows_socket_library_scope` (overload 2) constructor

```
explicit windows_socket_library_scope();
```

The `sst::windows_socket_library_scope` (overload 2) constructor calls the `WSAStartup` function and requests Winsock version 2.2.

If Winsock is not available, i.e., if `SST_WITH_WINDOWS_WS2_32` is `0`, this constructor does nothing.

3.25.103.3. The `sst::windows_socket_library_scope::data` function

```
sst_w32_WSADATA_wrapper const & data() const;
```

The `sst::windows_socket_library_scope::data` function returns a reference to the `sst_w32_WSADATA_wrapper` object that resulted from the `WSAStartup` call made by the constructor.

If Winsock is not available, i.e., if `SST_WITH_WINDOWS_WS2_32` is `0`, all data in the `sst_w32_WSADATA_wrapper` object will be zero.

3.25.103.4. The `sst::~windows_socket_library_scope` destructor

```
~windows_socket_library_scope() noexcept;
```

The `sst::~windows_socket_library_scope` destructor calls the `WSACleanup` function.

If Winsock is not available, i.e., if `SST_WITH_WINDOWS_WS2_32` is `0`, this destructor does nothing.

3.26. Miscellaneous

3.26.1. The `sst::value_bits` trait

C++

```
#include <sst/catalog/value_bits.hpp>
// or:   <sst/limits.h>
namespace sst {

template<class T>
class value_bits
  : public std::integral_constant<sst::remove_cvref_t<T>, /*...*/> {};

}
```

The `sst::value_bits` class provides the number of value bits[7] in the integer type `T`.

IMPORTANT Since `sst::value_bits` is publicly derived from `std::integral_constant <sst::remove_cvref_t<T>, v>`, there are two ways to extract the underlying value `v`. The first way is to write `sst::value_bits<T>::value`. This directly accesses the underlying value and always works as expected. The second way is to

write `sst::value_bits<T>()`. This constructs an instance of `sst::value_bits`, which will be implicitly converted to the underlying value in most contexts, but not all. It is recommended to always use the first way.

Example 49.

sst-example-49.cpp

```
#include <iomanip>
#include <ios>
#include <iostream>
#include <sst/limits.h>
#include <sst/catalog/to_string.hpp>
#include <string>

namespace {

template<class T>
void info(std::string const & T_name) {
  T const v = sst::value_bits<T>::value;
  std::cout << "sst::value_bits<";
  std::cout << std::left << std::setw(18) << T_name;
  std::cout << ">::value == ";
  std::cout << std::right << std::setw(2) << sst::to_string(v);
  std::cout << "\n";
}

#define INFO(T) info<T>(#T)

} // namespace

int main() {
  INFO(bool);
  INFO(char);
  INFO(signed char);
  INFO(unsigned char);
  INFO(short);
  INFO(unsigned short);
  INFO(int);
  INFO(unsigned int);
  INFO(long);
  INFO(unsigned long);
  INFO(long long);
  INFO(unsigned long long);
}
```

Compilation command

```
g++ sst-example-49.cpp -lsst
```

Possible output

```
sst::value_bits<bool               >::value ==  1
sst::value_bits<char               >::value ==  7
```

```
sst::value_bits<signed char       >::value ==  7
sst::value_bits<unsigned char     >::value ==  8
sst::value_bits<short             >::value == 15
sst::value_bits<unsigned short    >::value == 16
sst::value_bits<int               >::value == 31
sst::value_bits<unsigned int      >::value == 32
sst::value_bits<long              >::value == 63
sst::value_bits<unsigned long     >::value == 64
sst::value_bits<long long         >::value == 63
sst::value_bits<unsigned long long>::value == 64
```

3.26.2. The `sst::width_bits` trait

C++

```
#include <sst/catalog/width_bits.hpp>
// or:   <sst/limits.h>
namespace sst {

template<class T>
class width_bits
  : public std::integral_constant<sst::remove_cvref_t<T>, /*...*/> {};

}
```

The `sst::width_bits` class provides the number of width bits[6] in the integer type `T`.

IMPORTANT Since `sst::width_bits` is publicly derived from `std::integral_constant<sst::remove_cvref_t<T>, v>`, there are two ways to extract the underlying value `v`. The first way is to write `sst::width_bits<T>::value`. This directly accesses the underlying value and always works as expected. The second way is to write `sst::width_bits<T>()`. This constructs an instance of `sst::width_bits`, which will be implicitly converted to the underlying value in most contexts, but not all. It is recommended to always use the first way.

Example 50.

sst-example-50.cpp

```
#include <iomanip>
#include <ios>
#include <iostream>
#include <sst/limits.h>
#include <sst/catalog/to_string.hpp>
#include <string>

namespace {

template<class T>
void info(std::string const & T_name) {
  T const v = sst::width_bits<T>::value;
  std::cout << "sst::width_bits<";
```

```
  std::cout << std::left << std::setw(18) << T_name;
  std::cout << ">::value == ";
  std::cout << std::right << std::setw(2) << sst::to_string(v);
  std::cout << "\n";
}

#define INFO(T) info<T>(#T)

} // namespace

int main() {
  INFO(bool);
  INFO(char);
  INFO(signed char);
  INFO(unsigned char);
  INFO(short);
  INFO(unsigned short);
  INFO(int);
  INFO(unsigned int);
  INFO(long);
  INFO(unsigned long);
  INFO(long long);
  INFO(unsigned long long);
}
```

Compilation command

```
g++ sst-example-50.cpp -lsst
```

Possible output

```
sst::width_bits<bool               >::value ==  1
sst::width_bits<char               >::value ==  8
sst::width_bits<signed char        >::value ==  8
sst::width_bits<unsigned char      >::value ==  8
sst::width_bits<short              >::value == 16
sst::width_bits<unsigned short     >::value == 16
sst::width_bits<int                >::value == 32
sst::width_bits<unsigned int       >::value == 32
sst::width_bits<long               >::value == 64
sst::width_bits<unsigned long      >::value == 64
sst::width_bits<long long          >::value == 64
sst::width_bits<unsigned long long>::value == 64
```

3.26.3. The `sst::type_min` trait

C++

```
#include <sst/catalog/type_min.hpp>
// or:   <sst/limits.h>
namespace sst {
```

```
// (1)
template<std::integral T>
struct type_min : std::integral_constant<T, v> {};

}
```

The `sst::type_min` trait provides the minimum value of a type.

1. Provides the minimum value of a fundamental integer type[3].

IMPORTANT Since `sst::width_bits` is publicly derived from `std::integral_constant<sst::remove_cvref_t<T>, v>`, there are two ways to extract the underlying value `v`. The first way is to write `sst::width_bits<T>::value`. This directly accesses the underlying value and always works as expected. The second way is to write `sst::width_bits<T>()`. This constructs an instance of `sst::width_bits`, which will be implicitly converted to the underlying value in most contexts, but not all. It is recommended to always use the first way.

3.26.4. The `sst::type_max` trait

C++

```
#include <sst/catalog/type_max.hpp>
// or:   <sst/limits.h>
namespace sst {

// (1)
template<std::integral T>
struct type_max : std::integral_constant<T, v> {};

}
```

The `sst::type_max` trait provides the maximum value of a type.

1. Provides the maximum value of a fundamental integer type[3].

IMPORTANT Since `sst::width_bits` is publicly derived from `std::integral_constant<sst::remove_cvref_t<T>, v>`, there are two ways to extract the underlying value `v`. The first way is to write `sst::width_bits<T>::value`. This directly accesses the underlying value and always works as expected. The second way is to write `sst::width_bits<T>()`. This constructs an instance of `sst::width_bits`, which will be implicitly converted to the underlying value in most contexts, but not all. It is recommended to always use the first way.

3.26.5. The `sst::type_msb` trait

C++

```
#include <sst/catalog/type_msb.hpp>
// or:   <sst/bit.h>
namespace sst {
```

```
template<class IntType>
struct type_msb
  : std::integral_constant<IntType, /*...*/> {};

}
```

The `sst::type_msb` trait provides the value of integer type `IntType` whose value bits[7] are all set to 0 except for the most significant value bit, which is set to 1.

IMPORTANT Since `sst::type_msb` is publicly derived from `std::integral_constant<IntType, v>`, there are two ways to extract the underlying value `v`. The first way is to write `sst::type_msb<IntType>::value`. This directly accesses the underlying value and always works as expected. The second way is to write `sst::type_msb<IntType>()`. This constructs an instance of `sst::type_msb`, which will be implicitly converted to the underlying value in most contexts, but not all. It is recommended to always use the first way.

3.26.6. The `sst::char_bit` trait

C++

```
#include <sst/limits.h>
namespace sst {

using char_bit = sst::width_bits<unsigned char>;

}
```

The `sst::char_bit` class provides the value of `CHAR_BIT`.

3.26.7. The `sst::uchar_max` type alias

C++

```
#include <sst/catalog/uchar_max.hpp>
// or:   <sst/limits.h>
namespace sst {

using uchar_max = sst::type_max<unsigned char>;

}
```

The `sst::uchar_max` type alias provides an alternative name for `sst::type_max<unsigned char>`.

3.26.8. The `sst::promote` trait

C++

```
#include <sst/catalog/promote.hpp>
// or:   <sst/type.h>
namespace sst {

// (1)
template<std::integral T>
struct promote { using type = R; };

}
```

3.26.9. The `sst::promotes` trait

C++

```
#include <sst/catalog/promotes.hpp>
// or:   <sst/type.h>
namespace sst {

template<class T>
struct promotes : std::integral_constant<bool, b>;

}
```

The `sst::promotes` trait determines whether a type promotes to a type different from itself.

IMPORTANT Since `sst::promotes` is publicly derived from `std::integral_constant<bool, b>`, there are two ways to extract the underlying value `b`. The first way is to write `sst::promotes<T>::value`. This directly accesses the underlying value and always works as expected. The second way is to write `sst::promotes<T>()`. This constructs an instance of `sst::promotes`, which will be implicitly converted to the underlying value in most contexts, but not all. It is recommended to always use the first way.

3.26.10. The `sst::checked_t` class

C++

```
#include <sst/checked.h>
namespace sst {

template<class T>
class checked_t {

  // Underlying value retrieval
  constexpr T value() const noexcept;

};

}
```

The `sst::checked_t` class provides range-checked integer arithmetic.

Any value `x` of any integer type can be converted to an `sst::checked_t<T>` value via the converting constructor `sst::checked_t<T>(x)`, where `T` is also any integer type. If `x` cannot be represented by `T`, the constructor will throw an exception. Arbitrary integer arithmetic can then be performed on the `sst::checked_t<T>` value, which will cause the arithmetic to be range-checked_t: an exception will be thrown if there is any attempt to hold a value in a type that cannot represent the value, and every result that would have had type `R` will instead have type `sst::checked_t<R>`. After the arithmetic is complete, the underlying value can be retrieved with the `value()` function.

Unlike in normal integer arithmetic, unsigned integer types are not considered to be modular. If a value cannot be represented by an unsigned integer type, an exception will be thrown instead of reducing it modularly in that type.

Example 51.

```
[cl_sst_checked](INT_MIN) - 1; // throws
[cl_sst_checked](0U) - 1U;     // throws
```

3.26.11. The `sst::checked` function

C++

```
#include <sst/checked.h>
namespace sst {

template<class T>
checked_t<T> checked(T x) noexcept;

}
```

The `sst::checked` function converts a value of integer type `T` to an `[cl_sst_checked_t]<T>`.

3.26.12. The `sst::checked_cast` function

C++

```
#include <sst/checked.h>
namespace sst {

template<class T, class U>
[cl_SST_CPP14_CONSTEXPR] T checked_cast(U x);

}
```

The `sst::checked_cast` function converts a value of integer type `U` to a value of integer type `T`, throwing an exception if the value cannot be represented by `T`.

3.26.13. The `sst::perfect_lt` function

C++

```
#include <sst/catalog/perfect_lt.hpp>
// or:   <sst/integer.h>
namespace sst {

template<std::integral T1, std::integral T2>
constexpr bool perfect_lt(T1 x, T2 y) noexcept;

}
```

The `sst::perfect_lt` function determines whether `x` is less than `y` for any two values `x` and `y` of any two integer types `T1` and `T2`.

The usual arithmetic conversions can cause the plain < operator to yield an unexpected result when `T1` and `T2` have different signedness and the signed value is negative. Using the `sst::perfect_lt` function instead of the plain < operator will avoid this possibility.

The `sst::perfect_lt` function may have a small amount of overhead compared to the plain < operator, so it should be used judiciously.

3.26.14. The `sst::perfect_gt` function

C++

```
#include <sst/catalog/perfect_gt.hpp>
// or:   <sst/integer.h>
namespace sst {

template<std::integral T1, std::integral T2>
constexpr bool perfect_gt(T1 x, T2 y) noexcept;

}
```

The `sst::perfect_gt` function determines whether `x` is greater than `y` for any two values `x` and `y` of any two integer types `T1` and `T2`.

The usual arithmetic conversions can cause the plain > operator to yield an unexpected result when `T1` and `T2` have different signedness and the signed value is negative. Using the `sst::perfect_gt` function instead of the plain > operator will avoid this possibility.

The `sst::perfect_gt` function may have a small amount of overhead compared to the plain > operator, so it should be used judiciously.

3.26.15. The `sst::perfect_le` function

C++

```
#include <sst/catalog/perfect_le.hpp>
// or:   <sst/integer.h>
namespace sst {
```

```
template<std::integral T1, std::integral T2>
constexpr bool perfect_le(T1 x, T2 y) noexcept;

}
```

The `sst::perfect_le` function determines whether `x` is less than or equal to `y` for any two values `x` and `y` of any two integer types `T1` and `T2`.

The usual arithmetic conversions can cause the plain `<=` operator to yield an unexpected result when `T1` and `T2` have different signedness and the signed value is negative. Using the `sst::perfect_le` function instead of the plain `<=` operator will avoid this possibility.

The `sst::perfect_le` function may have a small amount of overhead compared to the plain `|` operator, so it should be used judiciously.

3.26.16. The `sst::perfect_ge` function

C++

```
#include <sst/catalog/perfect_ge.hpp>
// or:   <sst/integer.h>
namespace sst {

template<std::integral T1, std::integral T2>
constexpr bool perfect_ge(T1 x, T2 y) noexcept;

}
```

The `sst::perfect_ge` function determines whether `x` is greater than or equal to `y` for any two values `x` and `y` of any two integer types `T1` and `T2`.

The usual arithmetic conversions can cause the plain `>=` operator to yield an unexpected result when `T1` and `T2` have different signedness and the signed value is negative. Using the `sst::perfect_ge` function instead of the plain `>=` operator will avoid this possibility.

The `sst::perfect_ge` function may have a small amount of overhead compared to the plain `>=` operator, so it should be used judiciously.

3.26.17. The `sst::perfect_eq` function

C++

```
#include <sst/catalog/perfect_eq.hpp>
// or:   <sst/integer.h>
namespace sst {

template<std::integral T1, std::integral T2>
constexpr bool perfect_eq(T1 x, T2 y) noexcept;

}
```

The `sst::perfect_eq` function determines whether `x` is equal to `y` for any two values `x` and `y` of any two integer types `T1` and `T2`.

The usual arithmetic conversions can cause the plain == operator to yield an unexpected result when `T1` and `T2` have different signedness and the signed value is negative. Using the `sst::perfect_eq` function instead of the plain == operator will avoid this possibility.

The `sst::perfect_eq` function may have a small amount of overhead compared to the plain == operator, so it should be used judiciously.

3.26.18. The `sst::perfect_ne` function

C++

```
#include <sst/catalog/perfect_ne.hpp>
// or:   <sst/integer.h>
namespace sst {

template<std::integral T1, std::integral T2>
constexpr bool perfect_ne(T1 x, T2 y) noexcept;

}
```

The `sst::perfect_ne` function determines whether `x` is unequal to `y` for any two values `x` and `y` of any two integer types `T1` and `T2`.

The usual arithmetic conversions can cause the plain `!=` operator to yield an unexpected result when `T1` and `T2` have different signedness and the signed value is negative. Using the `sst::perfect_ne` function instead of the plain `!=` operator will avoid this possibility.

The `sst::perfect_ne` function may have a small amount of overhead compared to the plain `!=` operator, so it should be used judiciously.

3.26.19. The `sst::unsigned_lt` function

C++

```
#include <sst/catalog/unsigned_lt.hpp>
// or:   <sst/integer.h>
namespace sst {

template<class IntType1, class IntType2>
constexpr bool unsigned_lt(IntType1 a, IntType2 b) noexcept;

}
```

The `sst::unsigned_lt` function determines whether `a` is less than `b` for any two nonnegative values `a` and `b` of any two integer types `IntType1` and `IntType2` without generating any compiler warnings.

The usual arithmetic conversions guarantee that comparing `a` to `b` with the plain < operator always

yields the expected result if `a` and `b` are nonnegative, but it is common for compilers to generate warnings when `IntType1` and `IntType2` have different signedness. This can be annoying when writing template functions for arbitrary integer types, as certain type combinations can generate useless warnings. Using the `sst::unsigned_lt` function instead of the plain < operator will avoid these warnings.

The `sst::unsigned_lt` function does not have any overhead compared to the plain < operator.

If `a` is negative or `b` is negative, the behavior is undefined.

3.26.20. The `sst::unsigned_gt` function

C++

```
#include <sst/catalog/unsigned_gt.hpp>
// or:   <sst/integer.h>
namespace sst {

template<class IntType1, class IntType2>
constexpr bool unsigned_gt(IntType1 a, IntType2 b) noexcept;

}
```

The `sst::unsigned_gt` function determines whether `a` is greater than `b` for any two nonnegative values `a` and `b` of any two integer types `IntType1` and `IntType2` without generating any compiler warnings.

The usual arithmetic conversions guarantee that comparing `a` to `b` with the plain > operator always yields the expected result if `a` and `b` are nonnegative, but it is common for compilers to generate warnings when `IntType1` and `IntType2` have different signedness. This can be annoying when writing template functions for arbitrary integer types, as certain type combinations can generate useless warnings. Using the `sst::unsigned_gt` function instead of the plain > operator will avoid these warnings.

The `sst::unsigned_gt` function does not have any overhead compared to the plain > operator.

If `a` is negative or `b` is negative, the behavior is undefined.

3.26.21. The `sst::unsigned_le` function

C++

```
#include <sst/catalog/unsigned_le.hpp>
// or:   <sst/integer.h>
namespace sst {

template<class IntType1, class IntType2>
constexpr bool unsigned_le(IntType1 a, IntType2 b) noexcept;

}
```

The `sst::unsigned_le` function determines whether `a` is less than or equal to `b` for any two nonnegative values `a` and `b` of any two integer types `IntType1` and `IntType2` without generating any compiler warnings.

The usual arithmetic conversions guarantee that comparing `a` to `b` with the plain `<=` operator always yields the expected result if `a` and `b` are nonnegative, but it is common for compilers to generate warnings when `IntType1` and `IntType2` have different signedness. This can be annoying when writing template functions for arbitrary integer types, as certain type combinations can generate useless warnings. Using the `sst::unsigned_le` function instead of the plain `<=` operator will avoid these warnings.

The `sst::unsigned_le` function does not have any overhead compared to the plain `|` operator.

If `a` is negative or `b` is negative, the behavior is undefined.

3.26.22. The `sst::unsigned_ge` function

C++

```
#include <sst/catalog/unsigned_ge.hpp>
// or:   <sst/integer.h>
namespace sst {

template<class IntType1, class IntType2>
constexpr bool unsigned_ge(IntType1 a, IntType2 b) noexcept;

}
```

The `sst::unsigned_ge` function determines whether `a` is greater than or equal to `b` for any two nonnegative values `a` and `b` of any two integer types `IntType1` and `IntType2` without generating any compiler warnings.

The usual arithmetic conversions guarantee that comparing `a` to `b` with the plain `>=` operator always yields the expected result if `a` and `b` are nonnegative, but it is common for compilers to generate warnings when `IntType1` and `IntType2` have different signedness. This can be annoying when writing template functions for arbitrary integer types, as certain type combinations can generate useless warnings. Using the `sst::unsigned_ge` function instead of the plain `>=` operator will avoid these warnings.

The `sst::unsigned_ge` function does not have any overhead compared to the plain `>=` operator.

If `a` is negative or `b` is negative, the behavior is undefined.

3.26.23. The `sst::unsigned_eq` function

C++

```
#include <sst/catalog/unsigned_eq.hpp>
// or:   <sst/integer.h>
namespace sst {
```

```
template<class IntType1, class IntType2>
constexpr bool unsigned_eq(IntType1 a, IntType2 b) noexcept;

}
```

The `sst::unsigned_eq` function determines whether `a` is equal to `b` for any two nonnegative values `a` and `b` of any two integer types `IntType1` and `IntType2` without generating any compiler warnings.

The usual arithmetic conversions guarantee that comparing `a` to `b` with the plain `==` operator always yields the expected result if `a` and `b` are nonnegative, but it is common for compilers to generate warnings when `IntType1` and `IntType2` have different signedness. This can be annoying when writing template functions for arbitrary integer types, as certain type combinations can generate useless warnings. Using the `sst::unsigned_eq` function instead of the plain `==` operator will avoid these warnings.

The `sst::unsigned_eq` function does not have any overhead compared to the plain `==` operator.

If `a` is negative or `b` is negative, the behavior is undefined.

3.26.24. The `sst::unsigned_ne` function

C++

```
#include <sst/catalog/unsigned_ne.hpp>
// or:   <sst/integer.h>
namespace sst {

template<class IntType1, class IntType2>
constexpr bool unsigned_ne(IntType1 a, IntType2 b) noexcept;

}
```

The `sst::unsigned_ne` function determines whether `a` is unequal to `b` for any two nonnegative values `a` and `b` of any two integer types `IntType1` and `IntType2` without generating any compiler warnings.

The usual arithmetic conversions guarantee that comparing `a` to `b` with the plain `!=` operator always yields the expected result if `a` and `b` are nonnegative, but it is common for compilers to generate warnings when `IntType1` and `IntType2` have different signedness. This can be annoying when writing template functions for arbitrary integer types, as certain type combinations can generate useless warnings. Using the `sst::unsigned_ne` function instead of the plain `!=` operator will avoid these warnings.

The `sst::unsigned_ne` function does not have any overhead compared to the plain `!=` operator.

If `a` is negative or `b` is negative, the behavior is undefined.

3.26.25. `sst::strict_eq`

```
#include <sst/catalog/strict_eq.hpp>
```

```
sst::strict_eq(x, y)
```

The `sst::strict_eq` function returns `true` if all of the following conditions are satisfied, or `false` if not:

1. `x` and `y` have the same type ignoring any cvref-qualifiers.
2. `x == y` is `true`.

If the first condition is not satisfied, then the second condition will not be subject to compilation. In other words, if `x` and `y` have different types, then the function will return `false` even if `x == y` would not compile.

If the first condition is not satisfied, then the function will be `SST_CONSTEVAL`. Otherwise, the function will be `constexpr`.

If the first condition is not satisfied or `x == y` is `noexcept`, then the function will be `noexcept`.

3.26.26. The `sst::ones_mask` function

C++

```
#include <sst/catalog/ones_mask.hpp>
// or:   <sst/bit.h>
namespace sst {

// (1)
template<class IntType1, class IntType2>
constexpr IntType1 ones_mask(IntType2 n) noexcept;

// (2)
template<class IntType1, class IntType2, class IntType3>
constexpr IntType1 ones_mask(IntType2 n, IntType3 k) noexcept;

}
```

The `sst::ones_mask` function returns $(2^{n} - 1) \cdot 2^{k}$, where `k` is taken to be zero if it is omitted. This is the bit mask consisting of `n` one bits followed by `k` zero bits.

If `n` is negative or `k` is negative, the behavior is undefined.

For (1) and (2), if `n + k` is greater than the number of value bits[7] in `IntType1`, the behavior is undefined.

3.26.27. The `sst::uchar_msb` trait

C++

```
#include <sst/catalog/uchar_msb.hpp>
// or:   <sst/bit.h>
```

```
namespace sst {

using uchar_msb = [cl_sst_type_msb]<unsigned char>;

SST_CPP17_INLINE constexpr unsigned char uchar_msb_v = sst::uchar_msb::value;

}
```

The `sst::uchar_msb` trait provides the value of type `unsigned char` whose value bits[7] are all set to 0 except for the most significant value bit, which is set to 1.

3.26.28. The `sst::boxed` class template

The `sst::boxed` class template describes a class that stores a value of type `T` as a distinct type identified by type `Tag` in addition to type `T`, allowing different kinds of values of type `T` to be distinguished from each other.

C++

```
#include <sst/catalog/boxed.hpp>
namespace sst {

template<class T, class Tag>
class boxed {
public:

  using value_type = T;

  // Construction is forwarded to T.
  template<class... Args> explicit constexpr boxed(Args &&... args);

  // Copy construction, copy assignment, move construction, move
  // assignment, and destruction are intentionally implicit.

  // Value retrieval
  explicit SST_CPP14_CONSTEXPR operator T &() noexcept;
  explicit constexpr operator T const &() const noexcept;
  SST_CPP14_CONSTEXPR T & value() noexcept;
  constexpr T const & value() const noexcept;

  // Comparisons for aliases and publicly derived classes
  constexpr bool operator<(boxed const & b) const
      noexcept(noexcept(std::declval<T>() < std::declval<T>()));
  constexpr bool operator>(boxed const & b) const
      noexcept(noexcept(std::declval<T>() > std::declval<T>()));
  constexpr bool operator<=(boxed const & b) const
      noexcept(noexcept(std::declval<T>() <= std::declval<T>()));
  constexpr bool operator>=(boxed const & b) const
      noexcept(noexcept(std::declval<T>() >= std::declval<T>()));
  constexpr bool operator==(boxed const & b) const
      noexcept(noexcept(std::declval<T>() == std::declval<T>()));
  constexpr bool operator!=(boxed const & b) const
```

```
        noexcept(noexcept(std::declval<T>() != std::declval<T>()));

    // Comparisons for privately derived classes
    constexpr bool operator<(Tag const & b) const
        noexcept(noexcept(std::declval<T>() < std::declval<T>()));
    constexpr bool operator>(Tag const & b) const
        noexcept(noexcept(std::declval<T>() > std::declval<T>()));
    constexpr bool operator<=(Tag const & b) const
        noexcept(noexcept(std::declval<T>() <= std::declval<T>()));
    constexpr bool operator>=(Tag const & b) const
        noexcept(noexcept(std::declval<T>() >= std::declval<T>()));
    constexpr bool operator==(Tag const & b) const
        noexcept(noexcept(std::declval<T>() == std::declval<T>()));
    constexpr bool operator!=(Tag const & b) const
        noexcept(noexcept(std::declval<T>() != std::declval<T>()));

};

}
```

3.26.29. The `sst::bignum` class

C++

```
#include <sst/catalog/bignum.hpp>
namespace sst {

class bignum {

  template<std::integral T>
  explicit operator T() const;

};

}
```

3.26.29.1. Conversion to fundamental integer types

C++

```
template<std::integral T>
explicit operator T() const;
```

This operator converts the value of the big integer to a fundamental integer type[3] `T`. If `T` is unsigned, the conversion is modular.

3.26.30. The `sst::elgamal::cipher` class

C++

```
#include <sst/catalog/elgamal.hpp>
#include <sst/catalog/bignum.hpp>

#include <openssl/ec.h>

namespace sst {
namespace elgamal {
class cipher {

  cipher(EC_GROUP const * const group,
         sst::bignum const & skey,
         EC_POINT const * const pkey);

  cipher(EC_GROUP const * const group,
         EC_POINT const * const pkey);

  cipher(EC_GROUP const * const group);

  cipher(int const curve_nid);

  sst::elgamal::ciphertext
  encrypt_small_domain(std::vector<unsigned char> const & message) const;

  sst::elgamal::ciphertext
  encrypt_small_domain(std::size_t const & message) const;

  unsigned char
  decrypt_small_domain(sst::elgamal::ciphertext const & message_enc) const;

  std::vector<unsigned char> serialize_secret_key() const;

  std::vector<unsigned char> serialize_public_key() const;

  void deserialize_skey(std::vector<unsigned char> const & src);

  void deserialize_pkey(std::vector<unsigned char> const & src,
                        std::vector<unsigned char>::size_type & idx);

  void keygen();
};

}
}
```

The class `sst::elgamal::cipher` is meant to store the relevant information needed to perform Elgamal encryption, as well as some helper methods. The underlying elliptic curve `EC_GROUP` will be set at initialization, but both the `EC_POINT` public key and `sst::bignum` secret key can be set after initialization. The method `keygen` can be called to generate such a key pair.

The method `encrypt_small_domain` takes a message (as either a `std::size_t` number `m` or a `std::vector<unsigned char>` buffer which is encoded as a number `m`) and computes message : $= g^m$, where g is the generator of the elliptic curve group . Finally, Elgamal encryption

(using the public key pkey) is performed on message to produce the corresponding ciphertext:

$$x \xleftarrow{R} [0,\ \text{ord}(\mathbb{G}) - 1](g_1,\ g_2) = (x \cdot g,\ x \cdot \text{pkey} + \text{message})$$

The method `decrypt_small_domain` uses the secret key skey to convert the ciphertext to an `EC_POINT`, which is then decoded (one byte at a time) to recover the original message.

$$(g_1,\ g_2) \mapsto g_2 - \text{skey} \cdot g_1$$

There are also methods for serializing and deserializing both public and secret keys.

3.26.31. The `sst::str_cmp` function

C++

```
#include <sst/catalog/str_cmp.hpp>
namespace sst {

template<class IteratorA,
         class IteratorB,
         class ValueType =
             typename std::iterator_traits<IteratorA>::value_type>
constexpr int str_cmp(IteratorA a,
                      IteratorB b,
                      ValueType const & null_terminator = ValueType());

}
```

The `sst::str_cmp` function compares the two null-terminated sequences `a` and `b` and returns a value less than, equal to, or greater than zero if `a` is less than, equal to, or greater than `b`, respectively.

Example 52.

a.cpp

```
#include <iostream>
#include <vector>

#include <sst/catalog/str_cmp.hpp>

int main() {
  std::cout << sst::str_cmp("foo", "foo") << "\n";
  std::cout << sst::str_cmp("foo", "Foo") << "\n";
  std::cout << sst::str_cmp(L"Foo", L"foo") << "\n";
  std::vector<int> x{1, 2, 3, 4, 5, 6};
  std::vector<int> y{1, 2, 3, 6, 5};
  std::cout << sst::str_cmp(x.begin(), y.begin(), 3) << "\n";
  std::cout << sst::str_cmp(x.begin(), y.begin(), 5) << "\n";
  std::cout << sst::str_cmp(x.begin(), y.begin(), 6) << "\n";
}
```

```
$ g++ a.cpp -lsst

$ ./a.out
0
1
-1
0
-1
1
```

3.26.32. The `sst::opt_arg` enumeration

C++

```
#include <sst/catalog/opt_arg.hpp>
namespace sst {

enum class opt_arg {
  required,
  permitted,
  forbidden
};

}
```

3.26.33. The `sst::opt_exception` exception

C++

```
#include <sst/catalog/opt_exception.hpp>
namespace sst {

struct opt_exception : std::runtime_error {
  using std::runtime_error::runtime_error;
};

}
```

3.26.34. The `sst::parse_opt` function

C++

```
#include <sst/catalog/parse_opt.hpp>
namespace sst {

template<class StringList, class String>
auto parse_opt(StringList & args,
               String const & opt,
               opt_arg style = opt_arg::required,
```

```
                bool * has_arg = nullptr) ->
    typename std::enable_if<!std::is_pointer<String>::value,
                            bool>::type;

template<class StringList, class CharT>
bool parse_opt(StringList & args,
               CharT const * opt,
               opt_arg style = opt_arg::required,
               bool * has_arg = nullptr);

template<class StringList, class String1, class String2>
bool parse_opt(StringList & args,
               std::initializer_list<String1> opts,
               String2 * parsed_opt = nullptr,
               opt_arg style = opt_arg::required,
               bool * has_arg = nullptr);

template<class StringList, class String>
bool parse_opt(StringList & args,
               std::initializer_list<String> opts,
               opt_arg style = opt_arg::required,
               bool * has_arg = nullptr);

}
```

Example 53.

The following program demonstrates how `sst::parse_opt` could be used to parse the command-line arguments similarly to the `git init` command of Git 2.31.0 (see https://github.com/git/git/blob/v2.31.0/Documentation/git-init.txt):

cl_sst_parse_opt_git_init_example.cpp

```
//
// Copyright (C) Stealth Software Technologies, Inc.
//
// For the complete copyright information, please see the
// associated README file.
//

#include <cstdlib>
#include <exception>
#include <iostream>
#include <list>
#include <link:{repo_browser_url}/src/c-
cpp/include/sst/catalog/opt_arg.hpp[sst/catalog/opt_arg.hpp,window=_blank
]>
#include <link:{repo_browser_url}/src/c-
cpp/include/sst/catalog/parse_opt.hpp[sst/catalog/parse_opt.hpp,window=_b
lank]>
#include <link:{repo_browser_url}/src/c-
cpp/include/sst/catalog/unknown_opt.hpp[sst/catalog/unknown_opt.hpp,windo
w=_blank]>
#include <stdexcept>
```

```
#include <string>
#include <utility>

int main(int const argc, char ** const argv) {
  char const * const argv0 = argc > 0 ? argv[0] : "example";
  try {
    std::list<std::string> args(argv, argv + argc);
    if (args.empty()) {
      throw std::runtime_error("argc is zero");
    }
    bool parse_options = true;
    std::string directory = ".";
    bool directory_given = false;
    while (args.pop_front(), !args.empty()) {
      if (parse_options) {
        bool has_arg;
        if (sst::parse_opt(args, "--", sst::opt_arg::forbidden)) {
          parse_options = false;
          continue;
        }
        if (sst::parse_opt(args,
                           {"-q", "--quiet"},
                           sst::opt_arg::forbidden)) {
          // ...
          continue;
        }
        if (sst::parse_opt(args, "--bare", sst::opt_arg::forbidden)) {
          // ...
          continue;
        }
        if (sst::parse_opt(args, "--object-format")) {
          std::string & arg = args.front();
          // ...
          continue;
        }
        if (sst::parse_opt(args, "--template")) {
          std::string & arg = args.front();
          // ...
          continue;
        }
        if (sst::parse_opt(args, "--separate-git-dir")) {
          std::string & arg = args.front();
          // ...
          continue;
        }
        if (sst::parse_opt(args, {"-b", "--initial-branch"})) {
          std::string & arg = args.front();
          // ...
          continue;
        }
        if (sst::parse_opt(args,
                           "--shared",
                           sst::opt_arg::permitted,
                           &has_arg)) {
```

```
            std::string arg = has_arg ? std::move(args.front()) : "group";
            // ...
            continue;
          }
          sst::unknown_opt(args);
        }
        if (directory_given) {
          throw std::runtime_error("directory given twice");
        }
        directory = std::move(args.front());
        directory_given = true;
      }
      std::cout << "ok\n";
    } catch (std::exception const & e) {
      try {
        std::cerr << argv0 << ": " << e.what() << "\n";
      } catch (...) {
      }
      return EXIT_FAILURE;
    } catch (...) {
      try {
        std::cerr << argv0 << ": unknown error\n";
      } catch (...) {
      }
      return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
  }
```

Invocation examples

```
$ g++ cl_sst_parse_opt_git_init_example.cpp -lsst

$ ./a.out
ok

$ ./a.out --quiet --initial-branch foo
ok

$ ./a.out --quiet --initial-branch=foo
ok
# "--initial-branch foo" and "--initial-branch=foo" are equivalent
# because --initial-branch uses sst::opt_arg::required.

$ ./a.out --quiet --initial-branch
./a.out: option requires an argument: --initial-branch

$ ./a.out -q -b foo
ok

$ ./a.out -qbfoo
ok
# "-q -b foo", "-q -bfoo", and "-qbfoo" are equivalent because -q uses
# sst::opt_arg::forbidden and -b uses sst::opt_arg::required.
```

```
$ ./a.out -qb
./a.out: option requires an argument: -b

$ ./a.out -qbfoo --bar=baz
./a.out: unknown option: --bar

$ ./a.out -qbfoo -- --bar=baz
ok
# "--" is conventionally the options terminator (no further arguments
# are options, even if they begin with "-" or "--").

$ ./a.out --shared=all .
ok

$ ./a.out --shared all .
./a.out: directory given twice
# "--shared all" and "--shared=all" are not equivalent because --shared
# uses sst::opt_arg::permitted, not sst::opt_arg::required.
```

3.26.35. The `sst::unknown_opt` function

C++

```
#include <sst/catalog/unknown_opt.hpp>
namespace sst {

template<class StringList>
void unknown_opt(StringList const & args);

}
```

3.26.36. The `sst::monostate` class

C++

```
#include <sst/catalog/monostate.hpp>
namespace sst {

struct monostate {};

}
```

This class was added to the library because `std::monostate` is not available before C++17.

3.26.37. The `sst::indeterminate_t` tag class

C++

```
#include <sst/catalog/indeterminate_t.hpp>
```

```
namespace sst {

struct indeterminate_t {};

}
```

3.26.38. The `sst::indeterminate` tag

C++

```
#include <sst/catalog/indeterminate.hpp>
namespace sst {

SST_CPP17_INLINE constexpr sst::indeterminate_t indeterminate;

}
```

3.26.39. The `sst::errno_error_string` function

C++

```
#include <sst/catalog/errno_error_string.hpp>
namespace sst {

std::string errno_error_string(int code);
std::string errno_error_string();

}
```

The `sst::errno_error_string` function returns a string representing the `errno` value `code`. Omitting `code` is equivalent to calling `sst::errno_error_string(errno)`. If `code` is invalid, a generic error string will be returned.

3.26.40. The `sst::errno_exception` class

C++

```
#include <sst/catalog/errno_exception.hpp>
namespace sst {

class errno_exception : public std::runtime_error {
public:

  explicit errno_exception(std::string const & message, int code);
  explicit errno_exception(char const * message, int code);
  explicit errno_exception(std::string const & message);
  explicit errno_exception(char const * message);
  explicit errno_exception(int code);
  explicit errno_exception();
```

```
  int code() const noexcept;

};

}
```

1. If `message` is a null pointer, the behavior is undefined.
2. Otherwise, if `message` is empty, it will be adjusted to `sst::errno_error_string(code) + "."`.
3. Otherwise, if `message` ends with `()`, it will be adjusted to `message + " failed: " + sst::errno_error_string(code) + "."`.
4. Otherwise, `message` will be adjusted to `message + ": " + sst::errno_error_string(code) + "."`.

Omitting `code` is equivalent to passing `errno`.

3.26.41. The `sst::posix_gai_error_string` function

C++

```
#include <sst/catalog/posix_gai_error_string.hpp>
namespace sst {

std::string posix_gai_error_string(int code);

}
```

The `sst::posix_gai_error_string` function returns a string representing the `getaddrinfo` error code `code`.

3.26.42. `sst::is_negative`

C++

```
#include <sst/catalog/is_negative.hpp>
namespace sst {

template<sst::is_arithmetic_ish T>
constexpr bool is_negative(T const & x)
    noexcept(noexcept(x < sst::zero<T>()));

}
```

The `sst::is_negative` function returns `true` if `x` is negative.

If `T` is an unsigned fundamental integer type[3], no compiler warnings will be issued. This can be useful in generic code, where an expression such as `x < 0` can cause an "expression is always true" warning when the type of `x` happens to be unsigned. Writing `sst::is_negative(x)` instead of `x <`

0 will prevent such warnings.

3.26.43. The `sst::call_at_exit` function

C++

```
#include <sst/catalog/call_at_exit.hpp>
namespace sst {

void call_at_exit(void (*func)());

}
```

The `sst::call_at_exit` function registers the function pointed to by `func` to be called when the `main` function returns or the `std::exit` function is called. If `func` is a null pointer, the behavior is undefined.

The functions will be called in reverse order of registration. If the same function is registered multiple times, it will be called multiple times. The total number of registrations is only limited by the available memory.

The behavior is undefined if any of the following occurs:

1. A function calls `std::_Exit`, `std::exit`, `std::longjmp`, `std::quick_exit`, or `sst::call_at_exit`.

Any exceptions thrown by the functions will be caught and ignored.

The `sst::call_at_exit` function is thread-safe. If concurrent calls are made, the functions will be registered properly, but in an unspecified order.

Although the calls to the registered functions are sequential with respect to each other, they are not necessarily sequential with respect to any calls to static object destructors or functions registered with the `std::atexit` function.

Because of this weak sequencing guarantee, if you need to access a static data structure in `func`, you should not make the data structure itself static, as this may cause a data race between the call to `func` and the call to the data structure's destructor. Instead, you should keep a static pointer `p` to the data structure, lazily allocating and constructing it under `if (p == nullptr) {}` (if `p` is `thread_local`) or `std::call_once` (if `p` is not `thread_local`), and deallocating it in `func` under `if (p != nullptr) {}`.

3.26.44. The `sst::preinit` class

C++

```
#include <sst/catalog/preinit.hpp>
namespace sst {

struct preinit {
```

```
  template<class... Ts>
  preinit(Ts &&...) {
  }

};

}
```

The `sst::preinit` class accepts any arguments in its constructor and does nothing. This is primarily useful for running code at the beginning of a member initializer list.

Example 54.

sst-example-54.cpp

```
#include <sst/catalog/SST_ASSERT.h>
#include <sst/catalog/preinit.hpp>

class foo {
  sst::preinit preinit_;
  int x_;
public:
  foo(int const x) : preinit_((SST_ASSERT((x > 0)), 0)), x_(x) {
  }
};

int main() {
  foo f(0);
  return 0;
}
```

Compilation command

```
g++ sst-example-54.cpp -lsst
```

Possible output

```
terminate called after throwing an instance of 'std::logic_error'
  what():  sst-example-54.cpp:8: Assertion failed: (x > 0)
Aborted (core dumped)
```

3.26.45. The `sst::to_hex` function

The `sst::to_hex` function converts a sequence of bytes to a string.

3.26.45.1. The `sst::to_hex` function (overload 1)

```
#include <sst/catalog/to_hex.hpp>
namespace sst {

template<
```

```
  class Src,
  class End,
  class Dst,
  sst::enable_if_t<
    sst::is_byte_input_iterator<Src>::value
    && sst::is_sentinel<End, Src>::value
    && sst::is_output_iterator<Dst, char>::value
  > = 0>
Dst to_hex(
  Src src,
  End const & end,
  Dst dst,
  sst::to_hex_options const & options = {}
);

}
```

The `sst::to_hex` function (overload 1) converts a sequence of bytes to a hex string. Each byte will be converted to exactly two hex digits.

`Src` should satisfy `sst::is_byte_input_iterator<Src>`. In other words, `*src++` should read in successive elements of the input sequence.

`End` should satisfy `sst::is_sentinel<End, Src>`. In other words, `src == end` should indicate one-past-the-end of the input sequence.

`Dst` should satisfy `sst::is_output_iterator<Dst, char>`. In other words, `*dst++ = x` should write out successive elements of the output sequence, where `x` has type `char`.

The `options` parameter controls the output style.

1. `options.prefix()` indicates whether to add a `0x` prefix to the output. The prefix will be added even if the input sequence is empty. The default is `false`.

2. `options.uppercase_digits()` indicates whether to use `A` to `F` instead of `a` to `f` for digit values 10 to 15. The default is `true`.

3. `options.uppercase_prefix()` indicates whether to use `0X` instead of `0x` when `options.prefix()` is `true`. The default is `false`.

3.26.45.2. The `sst::to_hex` function (overload 2)

```
#include <sst/catalog/to_hex.hpp>
namespace sst {

template<
  class Dst = std::string,
  class Src,
  class End,
  sst::enable_if_t<
    sst::is_byte_input_iterator<Src>::value
    && sst::is_sentinel<End, Src>::value
  > = 0>
```

```
Dst to_hex(
  Src src,
  End const & end,
  sst::to_hex_options const & options = {}
);

}
```

3.26.45.3. The `sst::to_hex` function (overload 3)

```
#include <sst/catalog/to_hex.hpp>
namespace sst {

template<
  class Src,
  class End,
  class Dst,
  sst::enable_if_t<
    sst::is_byte_input_iterator<Src>::value
    && sst::is_value_sentinel<End, Src>::value
    && sst::is_output_iterator<Dst, char>::value
  > = 0>
Dst to_hex(
  Src src,
  End end,
  Dst dst,
  sst::to_hex_options const & options = {}
);

}
```

3.26.45.4. The `sst::to_hex` function (overload 4)

```
#include <sst/catalog/to_hex.hpp>
namespace sst {

template<
  class Dst = std::string,
  class Src,
  class End,
  sst::enable_if_t<
    sst::is_byte_input_iterator<Src>::value
    && sst::is_value_sentinel<End, Src>::value
  > = 0>
Dst to_hex(
  Src src,
  End const & end,
  sst::to_hex_options const & options = {}
);

}
```

3.26.45.5. The `sst::to_hex` function (overload 5)

```
#include <sst/catalog/to_hex.hpp>
namespace sst {

template<
  class Dst = std::string,
  class Src,
  sst::enable_if_t<
    sst::is_byte_input_iterator<Src>::value
  > = 0>
Dst to_hex(
  Src src,
  sst::to_hex_options const & options = {}
);

}
```

3.26.45.6. The `sst::to_hex` function (overload 6)

```
#include <sst/catalog/to_hex.hpp>
namespace sst {

template<
  class Src,
  class Dst,
  sst::enable_if_t<
    sst::is_byte_input_iterable<Src>::value
    && sst::is_output_iterator<Dst, char>::value
  > = 0>
Dst to_hex(
  Src const & src,
  Dst dst,
  sst::to_hex_options const & options = {}
);

}
```

3.26.45.7. The `sst::to_hex` function (overload 7)

```
#include <sst/catalog/to_hex.hpp>
namespace sst {

template<
  class Dst = std::string,
  class Src,
  sst::enable_if_t<
    sst::is_byte_input_iterable<Src>::value
  > = 0>
Dst to_hex(
  Src const & src,
```

```
  sst::to_hex_options const & options = {}
);

}
```

3.26.45.8. The `sst::to_hex` function (overload 8)

```
#include <sst/catalog/to_hex.hpp>
namespace sst {

template<
  class Src,
  class Dst,
  sst::enable_if_t<
    sst::is_byte<Src>::value
    && sst::is_output_iterator<Dst, char>::value
  > = 0>
Dst to_hex(
  Src src,
  Dst dst,
  sst::to_hex_options const & options = {}
);

}
```

3.26.45.9. The `sst::to_hex` function (overload 9)

```
#include <sst/catalog/to_hex.hpp>
namespace sst {

template<
  class Dst = std::string,
  class Src,
  sst::enable_if_t<
    sst::is_byte<Src>::value
  > = 0>
Dst to_hex(
  Src src,
  sst::to_hex_options const & options = {}
);

}
```

3.26.46. The `sst::integer_to_string_options` class

3.26.47. The `sst::to_hex_options` class

3.26.48. The `sst::unsigned_ceil_div` function

The `sst::unsigned_ceil_div` function returns $\lceil a / b \rceil$ for various type combinations of `a` and `b`. If `a` is negative, `b` is negative, or `b` is zero, the behavior is undefined.

3.26.48.1. Synopsis

```
#include <sst/catalog/unsigned_ceil_div.hpp>
namespace sst {

  constexpr R unsigned_ceil_div(sst::is_integer a,
                                sst::is_integer b) noexcept;

}
```

3.26.48.2. The `sst::unsigned_ceil_div` function (overload 1)

```
#include <sst/catalog/unsigned_ceil_div.hpp>
namespace sst {

  constexpr R unsigned_ceil_div(sst::is_integer a,
                                sst::is_integer b) noexcept;

}
```

The `sst::unsigned_ceil_div` function (overload 1) returns $\lceil a / b \rceil$, where `a` and `b` are fundamental integers[3]. If `a` is negative or `b` is not positive, the behavior is undefined.

By default, the return type `R` is the same as the type of the expression `a/b`, but it can be overridden by providing a type as a template parameter, in which case the return value will be explicitly converted to that type. For example, `sst::unsigned_ceil_div(9, 4L)` will return `3L`, while `sst::unsigned_ceil_div<int>(9, 4L)` will return `3`.

3.26.49. The `sst::unsigned_max` function

```
#include <sst/catalog/unsigned_max.hpp>
namespace sst {

template<
  class X = D,
  sst::is_arithmetic_ish Arg1,
  sst::is_arithmetic_ish... Args>
constexpr R unsigned_max(Arg1 && arg1, Args &&... args) noexcept(E);

}
```

The `sst::unsigned_max` function returns the maximum of its arguments. If any argument is negative, the behavior is undefined.

`D` is an unspecified type that is internal to the library. If the template parameter `X` is left defaulted to `D`, the return type `R` will be `sst::common_type_t<sst::remove_cvref_t<Args>...>`. Note that the result can always be represented in this type. Otherwise, the return type `R` will be `sst::remove_cvref_t<X>`, and if the result cannot be represented in this type, the behavior is undefined.

If all arguments are fundamental integers[3], `constexpr` will hold and `E` will be `true`.

3.26.50. The `sst::unsigned_min` function

```
#include <sst/catalog/unsigned_min.hpp>
namespace sst {

template<
  class X = D,
  sst::is_arithmetic_ish Arg1,
  sst::is_arithmetic_ish... Args>
constexpr R unsigned_min(Arg1 && arg1, Args &&... args) noexcept(E);

}
```

The `sst::unsigned_min` function returns the minimum of its arguments. If any argument is negative, the behavior is undefined.

`D` is an unspecified type that is internal to the library. If the template parameter `X` is left defaulted to `D`, the return type `R` will be `sst::common_type_t<sst::remove_cvref_t<Args>...>`. Note that the result can always be represented in this type. Otherwise, the return type `R` will be `sst::remove_cvref_t<X>`, and if the result cannot be represented in this type, the behavior is undefined.

If all arguments are fundamental integers[3], `constexpr` will hold and `E` will be `true`.

3.26.51. `sst::unsigned_rand_range_cc`

The `sst::unsigned_rand_range_cc` function returns a uniform random value from a left-closed, right-closed range of nonnegative integers.

3.26.51.1. Synopsis

```
#include <sst/catalog/unsigned_rand_range_cc.hpp>
namespace sst {

  template<sst::is_integer T>
  T unsigned_rand_range_cc(sst::is_urbg & g);

  template<sst::is_integer T>
  T unsigned_rand_range_cc();

}
```

3.26.51.2. Overload 1

```
#include <sst/catalog/unsigned_rand_range_cc.hpp>
namespace sst {

  template<sst::is_integer T>
  T unsigned_rand_range_cc(sst::is_urbg & g);

}
```

The `sst::unsigned_rand_range_cc` function (overload 1) returns a uniform random value from the complete nonnegative range of `T` using `g` as the randomness source.

Any cvref-qualifiers on `T` are removed.

3.26.51.3. Overload 2

```
#include <sst/catalog/unsigned_rand_range_cc.hpp>
namespace sst {

  template<sst::is_integer T>
  T unsigned_rand_range_cc();

}
```

The `sst::unsigned_rand_range_cc` function (overload 2) returns a uniform random value from the complete nonnegative range of `T` using `sst::crypto_rng()` as the randomness source.

Any cvref-qualifiers on `T` are removed.

Example 55.

sst-example-55.cpp

```
#include <iomanip>
#include <iostream>

#include <sst/catalog/unsigned_rand_range_cc.hpp>

int main() {
  for (int i = 0; i < 10; ++i) {
    for (int j = 0; j < 10; ++j) {
      if (j > 0) {
        std::cout << " ";
      }
      auto const x = sst::unsigned_rand_range_cc<unsigned char>();
      auto const y = static_cast<unsigned int>(x);
      std::cout << std::setw(3) << y;
    }
    std::cout << "\n";
  }
  return 0;
```

```
}
```

Possible output

```
136 147 227 221  40 234 203 127  49 188
110 192 192 116  29  62  27 132  75 140
 80  46 226 179  53 106  43  69 218  51
100 224 225 101 108 242  81 123 182  99
 90 194  62  87  84 193 130  20 144  24
 49  49   0 228  46 168  61 155  32  16
 42 158 107  69 247 212 229 154  36 237
 60 109 146 129 180 188 152 255  49  66
192 213 182 209 120  24  44 113 246 151
248 133 150 187  47 245 228 242  53 119
```

3.26.52. `sst::unsigned_rand_range_co`

The `sst::unsigned_rand_range_co` function returns a uniform random value from a left-closed, right-open range of nonnegative integers.

3.26.52.1. Synopsis

```
#include <sst/catalog/unsigned_rand_range_co.hpp>
namespace sst {

// TODO

}
```

3.26.53. `sst::unsigned_rand_range_oc`

The `sst::unsigned_rand_range_oc` function returns a uniform random value from a left-open, right-closed range of nonnegative integers.

3.26.53.1. Synopsis

```
#include <sst/catalog/unsigned_rand_range_oc.hpp>
namespace sst {

// TODO

}
```

3.26.54. `sst::unsigned_rand_range_oo`

The `sst::unsigned_rand_range_oo` function returns a uniform random value from a left-open, right-open range of nonnegative integers.

3.26.54.1. Synopsis

```
#include <sst/catalog/unsigned_rand_range_oo.hpp>
namespace sst {

// TODO

}
```

3.26.55. The `sst::last_arg` function

```
#include <sst/catalog/last_arg.hpp>
namespace sst {

template<std::size_t I = 1, class First, class... Rest>
constexpr R last_arg(First && first, Rest &&... rest);

}
```

The `sst::last_arg` function returns its `I`th argument, counting from right to left starting with 1. The behavior is as if the selected parameter were declared as `Arg && arg` and returned as `std::forward<Arg>(arg)`.

3.26.56. The `sst::numeric_limits` alias

C++

```
#include <sst/catalog/numeric_limits.hpp>
namespace sst {

template<class T>
using numeric_limits = std::numeric_limits<sst::remove_cvref_t<T>>;

}
```

The `sst::numeric_limits` alias is a convenience alias for `std::numeric_limits` that ignores any references and cv-qualifiers on `T`.

3.26.57. `sst::checked_overflow`

```
#include <sst/catalog/checked_overflow.hpp>
namespace sst {

class checked_overflow : public std::overflow_error {

public:

  using std::overflow_error::overflow_error;
```

```
}; //

}
```

3.26.58. `sst::zero`

C++

```
#include <sst/catalog/zero.hpp>
namespace sst {

template<
  class T,
  sst::enable_if_t<
    sst::is_arithmetic_like<T>::value
  > = 0>
consteval sst::remove_cvref_t<T> zero() noexcept;

template<
  class T,
  sst::enable_if_t<
    !sst::is_arithmetic_like<T>::value
    && sst::is_arithmetic_ish<T>::value
  > = 0>
constexpr R zero() noexcept(B);

}
```

3.26.59. The `sst::in_place_t` tag dispatch structure

```
#include <sst/catalog/in_place_t.hpp>
namespace sst {

struct in_place_t {};

}
```

The `sst::in_place_t` tag dispatch structure is used to indicate that an object should be constructed in-place.

The `sst::in_place_t` tag dispatch structure was added to the library because the `std::in_place_t` tag dispatch structure is not available below C++17.

3.26.60. The `sst::in_place` tag dispatch constant

```
#include <sst/catalog/in_place.hpp>
namespace sst {

SST_CPP17_INLINE constexpr sst::in_place_t in_place;
```

```
}
```

The `sst::in_place` tag dispatch constant corresponds to the `sst::in_place_t` tag dispatch structure.

3.26.61. SST_NARGS

C or C++

```
#include <sst/catalog/SST_NARGS.h>

#define SST_NARGS(...) n
```

The `SST_NARGS` macro expands to the number of arguments that were passed to it, formatted as an unsuffixed decimal integer constant.

If no arguments are given and the language version is below C23 or C++20, the behavior is undefined.

Example 56.

`sst-example-56.c` or `sst-example-56.cpp`

```
#include <sst/catalog/SST_C23_OR_LATER.h>
#include <sst/catalog/SST_CPP20_OR_LATER.h>
#include <sst/catalog/SST_NARGS.h>

#if __cplusplus
#include <cstdio>
using std::puts;
#else
#include <stdio.h>
#endif

#define STR2(X) #X
#define STR1(X) STR2(X)
#define F(...) \
  puts("SST_NARGS(" #__VA_ARGS__ ") = " STR1(SST_NARGS(__VA_ARGS__)))

int main() {
#if SST_C23_OR_LATER || SST_CPP20_OR_LATER
  F();
#endif
  F(a);
  F(a, b);
  F(a, b, c);
  return 0;
}
```

Output (below C23 or C++20)

```
SST_NARGS(a) = 1
```

```
SST_NARGS(a, b) = 2
SST_NARGS(a, b, c) = 3
```

Output (C23 or C++20 or later)

```
SST_NARGS() = 0
SST_NARGS(a) = 1
SST_NARGS(a, b) = 2
SST_NARGS(a, b, c) = 3
```

3.26.62. SST_DEFER

C or C++

```
#include <sst/catalog/SST_DEFER.h>

#define SST_DEFER /*...*/
```

The `SST_DEFER` macro defers the expansion of a function-like macro call until the expansion of the nearest enclosing call to the `SST_EXPAND` macro. To defer a function-like macro call `F(x)`, write the call as `SST_DEFER(F)(x)` instead. You must also define an auxiliary object-like macro named `F_DEFER` that simply expands to `F` (i.e., `#define F_DEFER F`).

3.26.63. The SST_EXPAND macro

C or C++

```
#include <sst/catalog/SST_EXPAND.h>

#define SST_EXPAND(...) __VA_ARGS__
```

The `SST_EXPAND` macro expands to its argument list unchanged.

Note that calling the `SST_EXPAND` macro without any arguments, i.e., writing `SST_EXPAND()`, is well-defined even when `__VA_OPT__` is not supported. In this case, the call is interpreted as having one argument that consists of no preprocessing tokens.

Example 57.

Because of the way the preprocessor works, macro arguments that contain commas must be enclosed in parentheses, otherwise the commas will be interpreted as macro argument separators. If the argument is an expression, then the extra parentheses are harmless, but if the argument is something else, then the extra parentheses will generally cause a syntax error.

For example, if `F(T)` is a macro that takes a type name `T`, then `F(std::map<int, int>)` won't work as expected, as the comma is interpreted as a macro argument separator and the call is therefore trying to pass two arguments: `std::map<int` and `int>`. Writing `F((std::map<int, int>))` fixes the argument count issue, but now the type name is enclosed in parentheses, which will generally cause a syntax error.

One way to resolve this is to declare a type alias for `std::map<int, int>` and pass the alias

to the macro. However, this is not a general solution, as it only works in the specific case where the argument is a type name.

A general way to resolve this is to require the arguments to always be enclosed in parentheses and use the SST_EXPAND macro to remove the parentheses.

sst-example-57.cpp

```
#include <iostream>
#include <map>
#include <type_traits>

#include <sst/catalog/SST_EXPAND.h>

// When calling this macro, each argument should always be enclosed in
// parentheses, which SST_EXPAND will then remove.
#define SAME(A, B) (std::is_same<SST_EXPAND A, SST_EXPAND B>::value)

int main() {
  std::cout << SAME((std::map<int, int>), (std::map<int, int>)) << "\n";
  return 0;
}
```

Output

```
1
```

3.26.64. sstSeqHashCore

A class H satisfies the sstSeqHashCore requirement if all of the following hold:

1. H satisfies the following class description:

```
class H {

public:

  // Default operations
  H() noexcept;                 // default constructor
  H(H const &);                 // copy constructor
  H & operator=(H const &);     // copy assignment
  H(H &&) noexcept;             // move constructor
  H & operator=(H &&) noexcept; // move assignment
  ~H() noexcept;                // destructor

  template<class T> class is_input;
  using output_t = /*...*/;

  // Mutators
  H & clear() noexcept;
  H & init();
  H & update(sst::is_input_iterator<is_input> src,
```

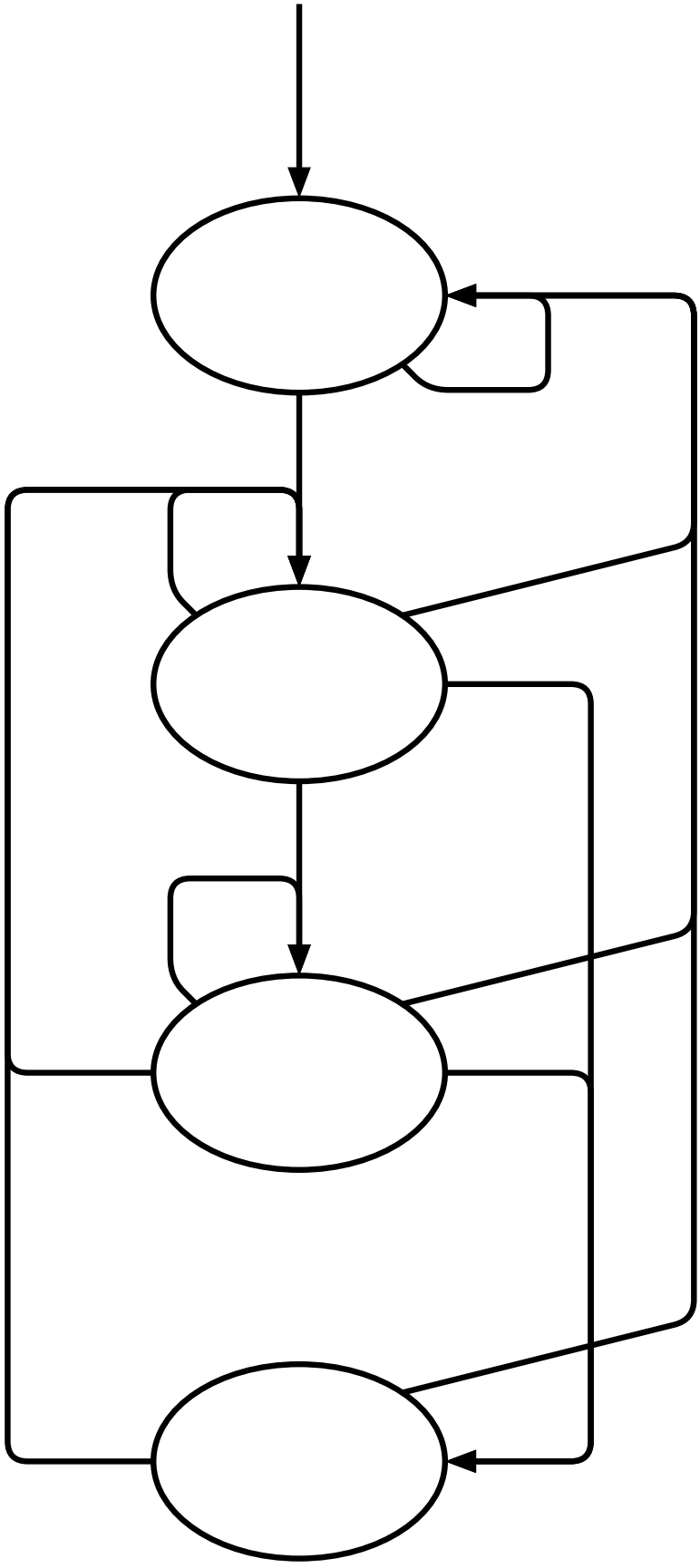
```
                sst::is_integer_ish const & len);
        H & update(sst::is_input_iterator<is_input> src,
                sst::is_sentinel<src> const & end);
        H & update(sst::is_input_iterator<is_input> src,
                sst::is_value_sentinel<src> const & end);
        H & finish();

        // Getters
        output_t const & output() const & noexcept;
        output_t && output() && noexcept;

    };
```

2. All mutator functions return `*this`.

The state of `H` changes according to the following diagram:



3.26.65. sst::value_sentinel

```
#include <sst/catalog/value_sentinel.hpp>
namespace sst {

template<class Value>
constexpr sst::value_sentinel_t<sst::remove_cvref_t<Value>>
value_sentinel(Value && value);

}
```

3.26.66. sst::value_sentinel_t

```
#include <sst/catalog/value_sentinel_t.hpp>
namespace sst {

template<class Value>
struct value_sentinel_t {

  Value value;

  friend constexpr bool operator==(
    value_sentinel_t const & a,
    Value const & b
  );

  friend constexpr bool operator==(
    Value const & a,
    value_sentinel_t const & b
  );

  friend constexpr bool operator!=(
    value_sentinel_t const & a,
    Value const & b
  );

  friend constexpr bool operator!=(
    Value const & a,
    value_sentinel_t const & b
  );

};

}
```

The `sst::value_sentinel_t` structure is used to disambiguate a count parameter vs. a value sentinel parameter. For example, consider the call `sst::sha256(p, 5)`. Does this mean hash 5 bytes from `p`, or hash bytes from `p` until we terminate by seeing a byte with value `5`? By convention, it means the former, and the latter must instead be written as `sst::sha256(p, sst::value_sentinel(5))`.

3.26.67. SST_REVERSE

C or C++

```
#include <sst/catalog/SST_REVERSE.h>

#define SST_REVERSE(...) /*...*/
```

The SST_REVERSE macro expands to a comma-separated list of its arguments in reverse order.

3.26.68. The SST_WITH_NOEXCEPT feature test macro

```
#include <sst/catalog/SST_WITH_NOEXCEPT.h>
// or:   <sst/catalog/SST_WITH_NOEXCEPT.hpp>

#define SST_WITH_NOEXCEPT /*...*/
```

The SST_WITH_NOEXCEPT feature test macro indicates whether the noexcept operator and the noexcept specifier are safe to use. Some versions of some compilers have bugs with these constructs, in which case it is best not to use them.

The macro expands to an integer constant expression with type int whose value is 1 to mean true and 0 to mean false. The expression is either parenthesized or exactly 1 or 0. The expression is also suitable for use in the preprocessor.

3.26.69. sst::unsigned_mod

```
#include <sst/catalog/unsigned_mod.hpp>

sst::unsigned_mod(x, m)
```

The sst::unsigned_mod function returns the least nonnegative residue of x modulo m.

x may have any integer-ish type X.

m may have any integer-ish type M.

m must be positive.

x must be nonnegative. If your x may be negative, you should use sst::mod instead of this function.

If X and M are not just integer-ish but in fact integer-like, then the function will be constexpr and noexcept.

The return value will have type M. Alternatively, you can specify a single template parameter to convert the return value to a specific type. For example, sst::unsigned_mod<int>(8L, 5L) will return 3 instead of 3L. If the conversion is not noexcept, it will remove any noexcept from the function.

3.26.70. `sst::mod`

```
#include <sst/catalog/mod.hpp>

sst::mod(x, m)
```

The `sst::mod` function returns the least nonnegative residue of `x` modulo `m`.

`x` may have any integer-ish type `X`.

`m` may have any integer-ish type `M`.

`m` must be positive.

If your `x` is always nonnegative, you may want to use `sst::unsigned_mod` instead of this function, as it may give better performance.

If `X` and `M` are not just integer-ish but in fact integer-like, then the function will be `constexpr` and `noexcept`.

The return value will have type `M`. Alternatively, you can specify a single template parameter to convert the return value to a specific type. For example, `sst::mod<int>(-8L, 5L)` will return `2` instead of `2L`. If the conversion is not `noexcept`, it will remove any `noexcept` from the function.

3.27. Hash functions

3.27.1. The `sst::sha256` function

```
#include <sst/catalog/sha256.hpp>
namespace sst {

// 1
template<
  class Src,
  sst::enable_if_t<
    (
         sst::is_iterable<Src, sst::is_byte>::value // a
      || sst::is_input_iterator<Src, sst::is_byte>::value // b
    )
  > = 0>
std::vector<unsigned char> sha256(Src && src);

// 2
template<
  class Src,
  sst::enable_if_t<
    (
         sst::is_iterable<Src, sst::is_byte>::value // a
      || sst::is_input_iterator<Src, sst::is_byte>::value // b
    ) && (
```

```
          sst::is_integer_ish<End>::value                  // c
        || sst::is_sentinel<End, Src>::value             // d
        || sst::is_value_sentinel<End, Src>::value       // e
      )
    > = 0>
  std::vector<unsigned char> sha256(Src && src, End const & end);

  }
```

The `sst::sha256` function computes and returns the SHA-256 hash of the byte sequence (a) stored by or (b) iterated by `src`. The byte sequence consists of:

(1a) All bytes stored by `src`.

(1b) All bytes iterated by `src` up to but not including the first zero byte.

(c) The first `end` bytes (a) stored by or (b) iterated by `src`.

(d) All bytes (a) stored by or (b) iterated by `src` up to but not including the first iterator that compares equal to `end`.

(e) All bytes (a) stored by or (b) iterated by `src` up to but not including the first byte that compares equal to `end`.

For (a), `src` will be treated as if it had type `sst::remove_cvref_t<Src> const &` regardless of its actual type. For (b), an iterator will be constructed from `src` via `sst::remove_cvref_t<Src>(std::forward<Src>(src))` and then iterated with no further copies or moves.

Example 58.

sst-example-58.cpp

```
#include <iostream>

#include <sst/catalog/sha256.hpp>
#include <sst/catalog/to_hex.hpp>

namespace {

template<class Label, class Hash>
void f(Label const & label, Hash const & hash) {
  std::cout << label;
  std::cout << sst::to_hex(hash);
  std::cout << "\n";
}

}

int main() {
  char const s[] = "Stealth";
  auto const v = sst::value_sentinel('h');
  f("1a  (Stealth): ", sst::sha256(s));
  f("2ac (Ste):     ", sst::sha256(s, 3));
  f("2ad (Steal):   ", sst::sha256(s, &s[5]));
```

```
  f("2ae (Stealt):  ", sst::sha256(s, v));
  f("2b  (tealth):  ", sst::sha256(&s[1]));
  f("2bc (tea):     ", sst::sha256(&s[1], 3));
  f("2bd (teal):    ", sst::sha256(&s[1], &s[5]));
  f("2be (tealt):   ", sst::sha256(&s[1], v));
  return 0;
}
```

Compilation command

```
g++ sst-example-58.cpp -lsst
```

Example output

```
1a  (Stealth):
526A40A447DDD7AA52D34BB62DBF1DA04DC4ABA8403769D239ADE80893FFEF7A
2ac (Ste):
25BD152797157817E497C23F90BA70D3D327F2F3721B030BFD9294CE10B61E85
2ad (Steal):
E2380ED2FB11F5D22C72FF91103478BBC2329BF16FAE4243F68FC796904B936F
2ae (Stealt):
D6973DF4D6C5CD8C3D368DD9CC8E1E8705215C2ED8D0353C697C3A0C3EF58CDE
2b  (tealth):
BC73F662B72CCC96E8AFD6182411CA385A00CE0AB3C99F746824A16FDA451BF4
2bc (tea):
A9F74D1EC36EBDEB2DA3F6E5868090CD2A2D20B3DCCA7B62F60304B1D3D9EF42
2bd (teal):
2655399ACF997BC2AD5346ECA205A8B1945E74831D4AAC71CAFBCFE2A73F093E
2be (tealt):
883B33A24036790D3DF3CC1900D5EB2B50EC20FA80DF803C8B4CE6DE132B6CCE
```

3.28. SQLite utilities

3.28.1. The `sst::sqlite::database` class

```
#include <sst/catalog/sqlite/database.hpp>
namespace sst {

class database {

public:

  database & operator=(
    database const & other
  );

  database & deserialize(
    char const *    schema,
    unsigned char * buffer,
    sqlite3_int64   size,
    bool            readonly = false,
```

```
    bool            copy = true,
    bool            resizable = true,
    sqlite3_int64   capacity = 0,
    bool            deallocate = false
  );

  database & deserialize(
    char const *          schema,
    unsigned char const * buffer,
    sqlite3_int64         size,
    bool                  readonly = false,
    bool                  copy = true,
    bool                  resizable = true,
    sqlite3_int64         capacity = 0,
    bool                  deallocate = false
  );

  template<
    class Size,
    sst::enable_if_t<
      sst::is_integer_ish<Size>::value
    > = 0>
  database & deserialize(
    char const *          schema,
    unsigned char const * buffer,
    Size const &          size
  );

  template<
    class Buffer,
    sst::enable_if_t<
      sst::is_iterable<Buffer, sst::is_byte>::value
      && sst::is_contiguous<Buffer>::value
    > = 0>
  database & deserialize(
    char const *   schema,
    Buffer const & buffer
  );

  std::vector<unsigned char> serialize(
    char const * schema
  );

};

}
```

3.28.1.1. Copy assignment

```
database & operator=(
  database const & other
);
```

Copy assignment overwrites `*this` with `other` as follows:

1. For each database that exists in both `*this` and `other`, the database in `*this` is overwritten with the database from `other`. The `main` database always exists, so this step always occurs for the `main` database. No other databases always exist, not even the `temp` database, which only comes into existence upon first use.

2. For each database that exists only in `other`, a new `:memory:` database is attached to `*this` and overwritten with the database from `other`. However, if the database in question is the `temp` database, then it cannot be attached, so it will instead be brought into existence by executing a `CREATE TEMP TABLE` statement before being overwritten.

3. For each database that exists only in `*this`, the database is detached from `*this`. However, if the database in question is the `temp` database, then it cannot be detached, so it will instead have all of its entities dropped.

3.28.1.2. The `deserialize` function (overload 1)

```
database & deserialize(
  char const *    schema,
  unsigned char * buffer,
  sqlite3_int64   size,
  bool            readonly = false,
  bool            copy = true,
  bool            resizable = true,
  sqlite3_int64   capacity = 0,
  bool            deallocate = false
);
```

The `deserialize` function (overload 1) attaches the `schema` name to the database serialization that consists of the first `size` bytes in `buffer`.

If `schema` is `"temp"`, then an exception will be thrown.

If `schema` already names a database, then it will be closed before attaching it to the database serialization. This will occur even if `schema` is `"main"`, which cannot normally be closed.

If `readonly` is `true`, then the database will be read-only.

If `copy` is `true`, then a copy of `buffer` will be allocated to underpin the database, and the copy will be automatically deallocated when the database is closed. Otherwise, `buffer` itself will underpin the database. In this case, the caller must ensure that `buffer` continues to exist and is not externally modified until the database is closed.

If `readonly` is `true`, then `resizable` is ignored. Otherwise, `resizable` specifies whether `buffer` will be automatically reallocated when more capacity is needed. In this case, either `copy` must be `true` or `buffer` must be allocated via SQLite.

If `readonly` is `true`, then `capacity` is ignored. Otherwise, `capacity` must be either zero or at least as large as `size`, and zero will be automatically adjusted to `size`. In this case, if `copy` is `true`, then `capacity` specifies how large the copy of `buffer` should be, although only the first `size` bytes will actually be copied. Otherwise, if `copy` is `false`, then `capacity` specifies how many bytes are actually available for use in `buffer`, which may be more than `size`.

If `copy` is `true`, then `deallocate` is ignored. Otherwise, `deallocate` specifies whether `buffer` will be automatically deallocated when the database is closed. If so, then `buffer` must be allocated via SQLite. When enabled, automatic deallocation will always occur, even if this function itself throws an exception.

This function always returns `*this`.

3.28.1.3. The `deserialize` function (overload 2)

```
database & deserialize(
  char const *          schema,
  unsigned char const * buffer,
  sqlite3_int64         size,
  bool                  readonly = false,
  bool                  copy = true,
  bool                  resizable = true,
  sqlite3_int64         capacity = 0,
  bool                  deallocate = false
);
```

The `deserialize` function (overload 2) is equivalent to the `deserialize` function (overload 1) except at least one of `readonly` and `copy` must be `true`.

3.28.1.4. The `deserialize` function (overload 3)

```
template<
  class Size,
  sst::enable_if_t<
    sst::is_integer_ish<Size>::value
  > = 0>
database & deserialize(
  char const *          schema,
  unsigned char const * buffer,
  Size const &          size
);
```

The `deserialize` function (overload 3) is equivalent to the `deserialize` function (overload 1) except the `size` parameter may be any integer-ish type.

3.28.1.5. The `deserialize` function (overload 4)

```
template<
  class Buffer,
  sst::enable_if_t<
    sst::is_iterable<Buffer, sst::is_byte>::value
    && sst::is_contiguous<Buffer>::value
  > = 0>
database & deserialize(
  char const *   schema,
  Buffer const & buffer
```

```
);
```

3.28.1.6. The `serialize` function (overload 1)

```
std::vector<unsigned char> serialize(
  char const * schema
);
```

3.29. Tracing utilities

3.29.1. The `SST_TEV_BOT` macro

C++

```
#include <sst/catalog/SST_TEV_BOT.hpp>

#define SST_TEV_BOT(tev) /*...*/
```

3.29.2. The `SST_TEV_TOP` macro

C++

```
#include <sst/catalog/SST_TEV_TOP.hpp>

#define SST_TEV_TOP(tev) /*...*/
```

Example 59.

`sst-example-59.cpp`

```
#include <iostream>
#include <stdexcept>

#include <sst/catalog/SST_TEV_ARG.hpp>
#include <sst/catalog/SST_TEV_BOT.hpp>
#include <sst/catalog/SST_TEV_DEF.hpp>
#include <sst/catalog/SST_TEV_TOP.hpp>
#include <sst/catalog/tracing_event.hpp>
#include <sst/catalog/tracing_exception.hpp>

#include <nlohmann/json.hpp>

namespace {

using tev_t = sst::tracing_event<nlohmann::json>;
using tex_t = sst::tracing_exception<tev_t>;

void f(tev_t tev) {
```

```
  SST_TEV_TOP(tev);
  throw std::runtime_error("Something bad happened");
  SST_TEV_BOT(tev);
}

} // namespace

int main() {
  tev_t SST_TEV_DEF(tev);
  try {
    f(SST_TEV_ARG(tev));
  } catch (tex_t const & e) {
    try {
      std::cerr << "Tracing event: ";
      if (e.tev().json()) {
        std::cerr << *e.tev().json();
      } else {
        std::cerr << "{}";
      }
      std::cerr << "\n";
    } catch (...) {
    }
  }
}
```

3.30. Serialization

3.30.1. `sst::i2osp`

```
#include <sst/catalog/i2osp.hpp>

sst::i2osp(src, len, [dst])
```

The `sst::i2osp` function converts a nonnegative integer `src` to a big endian octet string of length `len`. This is an implementation of the I2OSP (integer to octet string primitive) function from IETF RFC 8017.

`src` may have any integer-ish type.

`len` may have any integer-ish type.

`dst` may be any output iterator that accepts `unsigned char`.

If `dst` is given, the function writes the `len` octets to `dst` and returns one past the end of `dst`.

If `dst` is omitted, the function returns a `std::vector<unsigned char>` of length `len` that holds the octets.

If `len` octets are not enough to represent `src`, an exception will be thrown.

3.30.2. `sst::os2ip`

```
#include <sst/catalog/os2ip.hpp>

sst::os2ip<Dst>(src, [end])
```

The `sst::os2ip` function converts a big endian octet string `src` to a nonnegative integer. This is an implementation of the OS2IP (octet string to integer primitive) function from IETF RFC 8017.

`Dst` may be any integer-ish type.

`src` may be a byte input iterator or an iterable byte container. An iterable byte container will be implicitly adjusted to a byte input iterator starting at the first byte.

`end` may be a sentinel for `src`, a value sentinel for `src`, or an integer-ish count of the number of octets to read.

`end` may only be omitted if `src` is an iterable byte container, in which case `end` will be implicitly taken to be one past the end of the container.

The function returns the nonnegative integer value with type `Dst`.

If `Dst` does not have enough range to represent the integer value, an exception will be thrown.

4. Java library

4.1. Assertions

This is the simplest but always evaluates `g(x)` and always causes a call to `SST_ASSERT`:

```
void f(final Object x) {
  SST_ASSERT(g(x));
}
```

This omits the evaluation of `g(x)` but still always causes a call to `SST_ASSERT`:

```
void f(final Object x) {
  SST_ASSERT(SST_NDEBUG || g(x));
}
```

This omits the evaluation of `g(x)` and the call to `SST_ASSERT` since `SST_NDEBUG` is `static` and `final` (see JLS 7 §13.4.9):

```
void f(final Object x) {
  if (!SST_NDEBUG) {
    SST_ASSERT(g(x));
  }
}
```

In Java 7, you can make exceptions thrown by `g(x)` also trigger an assertion like this:

```
void f(final Object x) {
  if (!SST_NDEBUG) {
    try {
      SST_ASSERT(g(x));
    } catch (final Throwable e) {
      SST_ASSERT(e);
    }
  }
}
```

Or like this:

```
void f(final Object x) {
  if (!SST_NDEBUG) {
    SST_ASSERT(new ThrowingRunnable<Throwable>() {
      @Override
      public final void run() throws Throwable {
        SST_ASSERT(g(x));
      }
    });
  }
}
```

In Java 8 or later, you can use a lambda:

```
void f(final Object x) {
  if (!SST_NDEBUG) {
    SST_ASSERT(() -> {
      SST_ASSERT(g(x));
    });
  }
}
```

4.2. JSON processing

Example 60. Parsing JSON into a plain old data structure

This example uses Gson as the underlying JSON parsing library, but almost any library could be used. The only requirement is that the library can parse a JSON value into a hierarchy of `String`, `Boolean`, `Number`, `List`, and `Map` objects.

`Data.java`

```
import com.stealthsoftwareinc.sst.FromJson;
import com.stealthsoftwareinc.sst.Json;
import java.math.BigInteger;
import java.util.ArrayList;

public final class Data implements FromJson<Data> {
  public int x = 0;
  public BigInteger y = BigInteger.ZERO;
```

```
    @Override
    public final Data fromJson(final Object src) {
      x = Json.getAs(src, "x", x);
      y = Json.getAs(src, "y", y);
      return this;
    }
  }
```

Main.java

```
  import com.google.gson.Gson;
  import com.stealthsoftwareinc.sst.Json;
  import com.stealthsoftwareinc.sst.ParseFailureStatus;
  import java.io.InputStreamReader;
  import java.nio.charset.StandardCharsets;
  import java.util.Map;

  public final class Main {
    public static final void main(final String... args) throws Exception {
      try (final InputStreamReader stdin =
               new InputStreamReader(System.in, StandardCharsets.UTF_8)) {
        final Data d = new Data();
        try {
          Json.getTo(new Gson().fromJson(stdin, Map.class), d);
        } catch (final ParseFailureStatus e) {
          throw new ParseFailureStatus("<stdin>" + e.getMessage(), e);
        }
      }
    }
  }
```

4.3. The `Arith` class

Java

```
  package com.stealthsoftwareinc.sst;

  public final class Arith {
    private Arith();
  }
```

4.3.1. The `Arith.newtonSqrt` method

Java

```
  public static BigDecimal newtonSqrt(BigDecimal x, int scale);
```

The `Arith.newtonSqrt` method computes an approximation of $\sqrt{\texttt{x}}$ using Newton's method.

`scale` specifies the number of fractional digits to use for internal computations until a fixed point is reached. A larger `scale` produces a more accurate result, but there is no guarantee on how many

digits of the result match the true digits of $\sqrt{\texttt{x}}$.

4.4. The `Modulus` class

Java

```
package com.stealthsoftwareinc.sst;

public final class Modulus {

  // Constructors
  public Modulus(long m);
  public Modulus(BigInteger m);

  // Value sizing
  public boolean valuesFit(Byte T);
  public boolean valuesFit(Short T);
  public boolean valuesFit(Integer T);
  public boolean valuesFit(Long T);
  public boolean valuesFit(BigInteger T);

  // Modulus retrieval
  public byte get(Byte T);
  public short get(Short T);
  public int get(Integer T);
  public long get(Long T);
  public BigInteger get(BigInteger T);

}
```

The `Modulus` class represents a modulus that can be used for modular arithmetic.

4.4.1. Constructors

```
public Modulus(long m);
public Modulus(BigInteger m);
```

4.4.2. Value sizing

```
public boolean valuesFit(Byte T);
public boolean valuesFit(Short T);
public boolean valuesFit(Integer T);
public boolean valuesFit(Long T);
public boolean valuesFit(BigInteger T);
```

The `valuesFit` method determines whether type `T` can represent all modular values of m. The `BigInteger` overload always returns `true`. Every other overload returns `true` if and only if $m \leq 2^{\texttt{T.SIZE}}$.

Note that Java uses two's complement representation for `byte`, `short`, `int`, and `long`, and `T.SIZE` is the number of width bits in type `T`. See `Integer.SIZE`, for example. Because these types are signed, certain modular values are sometimes stored as negative values. For example, $2^{31} \pmod{2^{32}}$ and $2^{32} - 1 \pmod{2^{32}}$ are stored in type `int` as -2^{31} and -1, respectively.

4.4.3. Modulus retrieval

```
public byte get(Byte T);
public short get(Short T);
public int get(Integer T);
public long get(Long T);
public BigInteger get(BigInteger T);
```

The `get` method returns the value of m in type `T`.

Since `BigInteger` can represent any integer, `get((BigInteger)null)` always returns the exact value of m.

For `byte`, `short`, `int`, and `long`, if $m > 2^{\texttt{T.SIZE}}$, i.e., if `valuesFit(T)` returns `false`, the behavior of `get(T)` is undefined. Otherwise, if $m \geq 2^{\texttt{T.SIZE}-1}$, `get(T)` returns $-(2^{\texttt{T.SIZE}} - m)$. Otherwise, `get(T)` returns the exact value of m.

4.5. The `FixedPointModContext` class

Java

```
package com.stealthsoftwareinc.sst;

public final class FixedPointModContext {

}
```

The `FixedPointModContext` class converts between real numbers and fixed-point modular representation.

4.5.1. The `encode` method

If the encoding of `src` is not representable under the modulus and `checked` is `true`, an exception will be thrown. If the encoding of `src` is not representable under the modulus and `checked` is `false`, the behavior is undefined.

4.5.2. The `decode` method

If `src` is not a residue of the modulus, the behavior is undefined.

If the decoding of `src` is not representable in `dstType` and `checked` is `true`, an exception will be thrown.

If the decoding of `src` is not representable in `dstType` and `checked` is `false`, the behavior is undefined.

5. JavaScript library

5.1. Feature testing utilities

5.1.1. `sst.bootstrap_css_version`

JavaScript

```
import {} from "./sst/catalog/bootstrap_css_version.mjs";

async sst.bootstrap_css_version([options = {}])
```

The `sst.bootstrap_css_version` function searches for a Bootstrap CSS file and returns a `Promise` that resolves to the version number of the file.

If no Bootstrap CSS file is found, the returned `Promise` will resolve to `null`. Otherwise, it will resolve to a string that contains the version number of the first Bootstrap CSS file found. For example, for Bootstrap 5.3.2, the string will be `"5.3.2"`.

The first call to this function will cache its result, and subsequent calls will return the cached result. This behavior can be changed using `options.recache`.

The `options` parameter supports the following properties:

`recache`

> The function will behave as if this call was the first call. Any previously cached result will be overwritten with the new result.

5.2. Type traits

5.2.1. `sst.is_integer_ish`

JavaScript

```
import {} from "./sst/catalog/is_integer_ish.mjs";

sst.is_integer_ish(x)
```

The `sst.is_integer_ish` function returns `true` if `x` is a primitive safe integer or a primitive `BigInt`, or `false` if not.

5.2.2. `sst.is_json`

JavaScript

```
import {} from "./sst/catalog/is_json.mjs";

sst.is_json(x)
```

The `sst.is_json` function returns `true` if `x` is a JSON value, or `false` if not.

A JSON value is any of the following:

1. A primitive `String`.
2. A primitive finite `Number`.
3. A primitive `Boolean`.
4. The `null` value.
5. A plain array whose elements are also JSON values.
6. A plain object whose properties are also JSON values.

5.3. `sst.barf`

JavaScript

```
import {} from "./sst/catalog/barf.mjs";

sst.barf([message])
```

The `sst.barf` function outputs an error message and terminates the program to the greatest extent supported by the runtime environment.

5.4. `sst.chunkwise`

JavaScript

```
import {} from "./sst/catalog/chunkwise.mjs";

async sst.chunkwise(src, f, [options={}])
```

The `sst.chunkwise` function reads a stream of chunks from `src`, calling the `f` callback on each chunk. The return value is a `Promise` that resolves to `undefined`.

The `f` callback will be called as `f(chunk)` for each `chunk`. If `f` returns a `Promise`, it will be `await`ed.

The following types are supported for `src`:

`Blob`

The blob will be read as a stream of `Uint8Array` chunks.

Note that `Blob` is a superclass of `File`. When `src` is a `File`, it will be handled as a `File`, not a `Blob`.

`File` (browser only)

The file will be read as a stream of `Uint8Array` chunks.

If an exception `e` is thrown or a `Promise` is rejected with reason `e`, `sst.chunkwise` will reject its `Promise` with an `Error` reason whose `message` includes the file's `name` and whose `cause` is `e`.

Note that `File` is a subclass of `Blob`. When `src` is a `File`, it will be handled as a `File`, not a `Blob`.

`fsp.FileHandle` (Node.js only)

The file will be read as a stream of `Uint8Array` chunks.

`fsp.ReadStream` (Node.js only)

The file will be read as a stream of `Uint8Array` chunks.

`ReadableStream`

The stream of chunks will be read by calling `src.getReader()` to get a `reader`, then calling `reader.read()` repeatedly. Each `chunk` with type `ArrayBuffer` will be adjusted to a `Uint8Array(chunk)` view.

string (Node.js only)

`src` will be opened as a file via `fsp.open(src, "r")`, and the file will be read as a stream of `Uint8Array` chunks.

If an exception `e` is thrown or a `Promise` is rejected with reason `e`, `sst.chunkwise` will reject its `Promise` with an `Error` reason whose `message` includes `src` (the file's name) and whose `cause` is `e`.

Any other type

`src` itself will be considered to be the only chunk in the stream. If `src` has type `ArrayBuffer`, it will be adjusted to a `Uint8Array(src)` view.

The following `options` are supported:

`fetch` (optional)

Options to pass to the `sst.fetch` function. The presence of this option changes the behavior of the `sst.chunkwise` function as follows:

1. `src` must be a string.

2. `sst.fetch(src, options.fetch)` will be called, and the response body will be read as a stream of `Uint8Array` chunks.

5.5. `sst.crypto_rng`

JavaScript

```
import {} from "./sst/catalog/crypto_rng.mjs";

sst.crypto_rng(dst, [options = {}])
```

The `sst.crypto_rng` function generates cryptographically secure random bytes.

`dst` may be any of the following:

1. A nonnegative primitive safe integer `Number`.

   The function will create a new `Uint8Array` with length `dst` and fill it with random bytes.

   The return value will be the new `Uint8Array`.

2. A nonnegative primitive `BigInt`.

   The function will create a new `Uint8Array` with length `dst` and fill it with random bytes.

   The return value will be the new `Uint8Array`.

3. A `TypedArray`.

   The function will fill the array with random bytes.

   The return value will be `dst`.

4. An array `[x, i, n]` where `x` is a typed array and `i` and `n` are (big) integers.

   The function will fill the `n` elements of `x` beginning with `x[i]` with random bytes. If `n` is zero, no random bytes will be generated, but `i` must still be valid.

   The return value will be `x`.

`options` may contain the following members:

`async` (optional)

   A boolean that indicates whether the function should perform its work asynchronously.

   If this is `false`, the function will perform its work synchronously.

   If this is `true`, the function will perform its work asynchronously and return a `Promise` that resolves to the synchronous return value instead of returning the synchronous return value directly.

By default, this is `false`.

`crypto` (optional)

The instance of the Web Cryptography API to use.

By default, this is `window.crypto` in a browser, or `(await import("node:crypto")).webcrypto` in Node.js.

5.6. `sst.debounce`

JavaScript

```
import {} from "./sst/catalog/debounce.mjs";

sst.debounce([init], proc, [delay=1000])
```

The `sst.debounce` function returns an `async` function `f` that can be used as a debounced handler for an event.

The `proc` callback should perform the debounced work. Instead of being called once for each call to `f`, it will only be called after the most recent call to `f` becomes `delay` milliseconds old.

The `init` callback, if given, will be called by `f` each time `f` is called. It can be used to perform any non-debounced work.

No calls to `init` or `proc` will ever overlap, even if they use `await`. In other words, at any given moment, at most one execution context will exist for `init` and `proc`, even if that execution context is currently suspended by an `await`.

Conceptually, each call to `f` causes a direct call to `init` and possibly a delayed call to `proc`. If the `proc` call occurs, it is guaranteed to occur after the `init` call for the same call to `f`. `proc` will never be called directly by `f`, even if `delay` is 0. In other words, `proc` will only be called after `f` returns.

`init` and `proc` will be passed an initial parameter named `canceler`. In `init`, `canceler` will be `undefined`, as it is only passed to be consistent with `proc`. In `proc`, you can check `canceler.cancel` for truthiness after any use of `await`. If `canceler.cancel` is truthy, it is guaranteed that a future call to `proc` will occur, giving you a chance to return early from this call. If `canceler.cancel` is falsy, it is guaranteed that any future call to `proc` will be preceded by a future call to `init`, and this guarantee lasts until the next `await` or until this call returns. `canceler.cancel` always starts off falsy, and if you never use `await`, it will stay falsy, so you can ignore it entirely.

`f` will forward its parameters to both `init` and `proc` as the second and subsequent parameters, after the initial `canceler` parameter.

Any exceptions thrown by `init` or `proc` will be ignored. Any `Promise`s returned by `init` or `proc` will be `await`ed, and any rejections will be ignored.

`f` will never throw any exceptions. The `Promise` returned by `f` will always resolve to `undefined` and never be rejected.

5.7. `sst.fetch`

JavaScript

```
import {} from "./sst/catalog/fetch.mjs";

sst.fetch(resource, [options])
```

The `sst.fetch` function is a wrapper around the standard `fetch` function that provides additional functionality.

The behavior is the same as the standard `fetch` function, except the `options` parameter additionally supports the following options:

`insecure` (optional)

> If this is truthy and the request would use HTTPS, then HTTPS certificate validation will be skipped. However, this is only supported in Node.js, and the node-fetch package must be available. If the code is running in a browser, then an Error will be thrown, as HTTPS certificate validation cannot be skipped in a browser.
>
> If this is falsy or the request would use HTTP, then there is no effect, even if the code is running in a browser.
>
> By default, this is falsy.

5.8. The `sst.fetch_slurp` function

JavaScript

```
sst.fetch_slurp(url, init)
```

The `sst.fetch_slurp` function is similar to the `fetch` method, but performs multiple requests to gather multiple responses linked together by the `Link` header.

5.9. `sst.in_browser`

JavaScript

```
import {} from "./sst/catalog/in_browser.mjs";

sst.in_browser()
```

The `sst.in_browser` function returns `true` if the code is running inside a browser, or `false` if not.

5.10. `sst.is_stream_writable`

JavaScript

```
import {} from "./sst/catalog/is_stream_writable.mjs";

sst.is_stream_writable(x)
```

The `sst.is_stream_writable` function returns `true` if `x` implements the `stream.Writable` interface, or `false` if not.

This function has the following advantages over trying to write `x instanceof stream.Writable`:

- This function works in any runtime environment.
- This function does not require the `stream` package to be `imported`.
- This function works if the class of `x` is not a subclass of `stream.Writable`.

5.11. `sst.post_json`

JavaScript

```
import {} from "./sst/catalog/post_json.mjs";

sst.post_json(resource, body, [options={}])
```

The `sst.post_json` function is a wrapper around the `sst.fetch` function that is streamlined for `POST` endpoints that use JSON for both the request body and the response body.

The behavior is the same as `sst.fetch(resource, options)`, but with the following changes:

1. If `options.body` is omitted, then `sst.json.dump(body)` will be used. Otherwise, `body` will be ignored.
2. If `options.headers` is omitted or is missing `Content-Type`, then `Content-Type: application/json` will be added.
3. If `options.method` is omitted, then `POST` will be used.
4. The returned `Promise` will resolve to the response body parsed as JSON.
5. The returned `Promise` will be rejected if the response code is not in the 200-299 range. In this case, the rejection `Error` will include the `Response` as a member named `response`.

5.12. The `sst.sankey` function

JavaScript

```
sst.sankey(edges[, options])
```

The `sst.sankey` function draws a Sankey diagram and returns it as an SVG element.

A Sankey diagram is fundamentally a visualization of a weighted directed graph. As such, the minimum information from which a Sankey diagram can be created is the set of edges of the graph. The set of edges can be passed in via the `edges` parameter as an array of triples, where each triple `[A,B,W]` means there is an edge from node `A` to node `B` with weight `W`. `A` and `B` should be strings, and `W` should be a number.

Example 61.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<script src="d3.js"></script>
<script src="d3-sankey.js"></script>
</head>
<body>
<script type="module">
import {} from "./sst/catalog/sankey.mjs";
const edges = [
  ["A", "X", 4],
  ["A", "Y", 1],
  ["B", "X", 2],
  ["B", "Y", 3],
];
document.body.appendChild(sst.sankey(edges));
</script>
</body>
</html>
```

```
<script>{
  const currentScript = document.currentScript;
  window.addEventListener("DOMContentLoaded", function() {
    sst.iframeExampleOutput(currentScript, `
      <!DOCTYPE html>
      <html>
      <head>
      <meta charset="UTF-8">
      <script src="d3.js"><`+`/script>
      <script src="d3-sankey.js"><`+`/script>
      <`+`/head>
      <body>
      <script type="module">
      import {} from "./sst/catalog/sankey.mjs";
      const edges = [
        ["A", "X", 4],
        ["A", "Y", 1],
        ["B", "X", 2],
        ["B", "Y", 3],
      ];
      document.body.appendChild(sst.sankey(edges));
      <`+`/script>
      <`+`/body>
      <`+`/html>
    `);
```

```
  });
}</script>
```

5.12.1. `options` reference

`height`

Specifies the height, in pixels, of the diagram.

The default value is `480`.

`showNodeWeights`

Specifies whether the default node label will display the weight of the node.

The default value is **`false`**.

`width`

Specifies the width, in pixels, of the diagram.

The default value is `640`.

5.13. `sst.dom.bootstrap5.progress_bar`

JavaScript

```
import {} from "./sst/catalog/dom/bootstrap5/progress_bar.mjs";

class sst.dom.bootstrap5.progress_bar
```

Example 62.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="bootstrap-5.3.3.min.css">
    <link rel="stylesheet"
href="sst/catalog/dom/bootstrap5/progress_bar.css">
  </head>
  <body>
    <script type="module">

      import {} from "./sst/catalog/dom/bootstrap5/progress_bar.mjs";

      const v = 25;

      const progress_bar = new sst.dom.bootstrap5.progress_bar({
        max: 100,
        value: v,
```

```
      });
      document.body.appendChild(progress_bar.container());
      progress_bar.container().classList.add("m-2");

      const label_1 = "Set to 100%";
      const label_2 = "Set to " + v + "%";
      const full_button = document.createElement("button");
      full_button.type = "button";
      full_button.classList.add("btn", "btn-primary");
      full_button.innerText = label_1;
      full_button.addEventListener("click", () => {
        if (progress_bar.percent() < 100) {
          progress_bar.percent(100);
          full_button.innerText = label_2;
        } else {
          progress_bar.percent(v);
          full_button.innerText = label_1;
        }
      });
      document.body.appendChild(full_button);

      const spinning_button = document.createElement("button");
      spinning_button.type = "button";
      spinning_button.classList.add("btn", "btn-primary");
      spinning_button.innerText = "Toggle spinning";
      spinning_button.addEventListener("click", () => {
        const x = progress_bar.spinning();
        progress_bar.spinning(!x);
      });
      document.body.appendChild(spinning_button);

    </script>
  </body>
</html>
```

5.14. sst.dom.bootstrap5.text_input

JavaScript

```
import {} from "./sst/catalog/dom/bootstrap5/text_input.mjs";

class sst.dom.bootstrap5.text_input
```

Example 63.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="bootstrap-5.3.3.min.css">
    <link rel="stylesheet"
href="sst/catalog/dom/bootstrap5/text_input.css">
```

```
  </head>
  <body>
    <script type="module">

      import {} from "./sst/catalog/dom/bootstrap5/text_input.mjs";

      function create_button(text, callback) {
        const button = document.createElement("button");
        button.type = "button";
        button.classList.add("btn", "btn-primary", "mt-2", "me-1");
        button.innerText = text;
        button.addEventListener("click", callback);
        return button;
      }

      function create_demo(overlay) {
        let options = {
          label: "Value",
          validator: value => {
            if (value === "") {
              return "Value must not be empty";
            }
          },
        };
        options = Object.assign(options, overlay);
        const container = document.createElement("div");
        container.classList.add("m-2", "mb-4");
        const input = new sst.dom.bootstrap5.text_input(options);
        container.appendChild(input.container());
        container.appendChild(create_button("Disable", event => {
          const b = !input.disabled();
          input.disabled(b);
          event.target.innerText = b ? "Enable" : "Disable";
        }));
        container.appendChild(create_button("Commit", event => {
          const b = !input.committed();
          input.committed(b);
          event.target.innerText = b ? "Decommit" : "Commit";
        }));
        container.lastChild.classList.remove("me-1");
        document.body.appendChild(container);
      }

      create_demo({});

      create_demo({
        multiline: true,
      });

      document.body.lastChild.classList.remove("mb-4");

    </script>
  </body>
</html>
```

5.15. sst.dom.sidebar

JavaScript

```
import {} from "./sst/catalog/dom/sidebar.mjs";

class sst.dom.sidebar
```

Example 64.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="sst/catalog/dom/sidebar.css">
  </head>
  <body>
    <script type="module">

      import {} from "./sst/catalog/dom/sidebar.mjs";

      const container = document.createElement("div");
      document.body.appendChild(container);
      container.style.setProperty("display", "flex");

      const sidebar = new sst.dom.sidebar();
      container.appendChild(sidebar.container());
      sidebar.container().style.setProperty("flex-basis", "15em");
      sidebar.container().style.setProperty("flex-shrink", "0");
      sidebar.content().innerText = "Sidebar";

      const content = document.createElement("div");
      container.appendChild(content);
      content.innerText = "Content ".repeat(2000);

    </script>
  </body>
</html>
```

Example 65.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="sst/catalog/dom/sidebar.css">
  </head>
  <body>
    <script type="module">

      import {} from "./sst/catalog/dom/scrollspy.mjs";
      import {} from "./sst/catalog/dom/sidebar.mjs";
```

```
  const container = document.createElement("div");
  document.body.appendChild(container);
  container.style.setProperty("display", "flex");

  const sidebar = new sst.dom.sidebar();
  container.appendChild(sidebar.container());
  sidebar.container().style.setProperty("flex-basis", "15em");
  sidebar.container().style.setProperty("flex-shrink", "0");

  const ul = document.createElement("ul");
  sidebar.content().appendChild(ul);

  const sections = document.createElement("div");
  container.appendChild(sections);

  const items = [];

  for (let i = 0; i < 10; ++i) {

    const section = document.createElement("div");
    sections.appendChild(section);
    section.style.setProperty("flex-grow", "1");

    const title = document.createElement("h2");
    section.appendChild(title);
    title.innerText = "Section " + (i + 1);

    const content = document.createElement("div");
    section.appendChild(content);
    content.innerText = "Content ".repeat(200);

    const li = document.createElement("li");
    ul.appendChild(li);
    li.innerText = title.innerText;

    items.push({
      getBoundingClientRect: function() {
        return section.getBoundingClientRect();
      },
      li: li,
    });

  }

  sst.dom.scrollspy(items, (active, inactive) => {
    for (const item of active) {
      item.li.style.setProperty("font-weight", "bold");
    }
    for (const item of inactive) {
      item.li.style.removeProperty("font-weight");
    }
  });

</script>
```

```
  </body>
</html>
```

5.16. sst.dom.list

JavaScript

```
import {} from "./sst/catalog/dom/list.mjs";

class sst.dom.list
```

Example 66.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="sst/catalog/dom/list.css">
  </head>
  <body>
    <script type="module">

      import {} from "./sst/catalog/dom/list.mjs";

      const container = document.createElement("div");
      document.body.appendChild(container);

      const list = new sst.dom.list();

      let count = 0;

      for (; count < 5; count) {
        list.add().content().innerText = "Item " + count;
      }

      const add_button_div = document.createElement("div");
      container.appendChild(add_button_div);
      const add_button = document.createElement("button");
      add_button_div.appendChild(add_button);
      add_button.innerText = "Add item";
      add_button.addEventListener("click", () => {
        const n = list.size();
        const i = Math.round(Math.random() * n);
        list.add(i).content().innerText = "Item " + count;
      });

      const remove_button_div = document.createElement("div");
      container.appendChild(remove_button_div);
      const remove_button = document.createElement("button");
      remove_button_div.appendChild(remove_button);
      remove_button.innerText = "Remove item";
      remove_button.addEventListener("click", () => {
```

```
      const n = list.size();
      if (n > 0) {
        const i = Math.floor(Math.random() * n);
        list.remove(i);
      }
    });

    const move_button_div = document.createElement("div");
    container.appendChild(move_button_div);
    const move_button = document.createElement("button");
    move_button_div.appendChild(move_button);
    move_button.innerText = "Move item";
    move_button.addEventListener("click", () => {
      const n = list.size();
      if (n > 1) {
        const i = Math.floor(Math.random() * n);
        let j;
        do {
          j = Math.round(Math.random() * n);
        } while (i === j || i === j - 1);
        list.move(i, j);
      }
    });

    container.appendChild(list.container());

  </script>
 </body>
</html>
```

5.17. sst.dom.hideable

JavaScript

```
import {} from "./sst/catalog/dom/list.mjs";

class sst.dom.hideable
```

Example 67.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="sst/catalog/dom/hideable.css">
  </head>
  <body>
    <script type="module">

      import {} from "./sst/catalog/dom/hideable.mjs";

      const container = document.createElement("div");
```

```
        document.body.appendChild(container);

        const hideable = new sst.dom.hideable();
        hideable.content().innerText = "Hideable content";

        const button_div = document.createElement("div");
        container.appendChild(button_div);
        const button = document.createElement("button");
        button_div.appendChild(button);
        button.innerText = "Hide";
        button.addEventListener("click", () => {
          if (hideable.hidden()) {
            hideable.hidden(false);
            button.innerText = "Hide";
          } else {
            hideable.hidden(true);
            button.innerText = "Unhide";
          }
        });

        container.appendChild(hideable.container());

      </script>
    </body>
  </html>
```

5.18. `sst.reduce_motion`

JavaScript

```
import {} from "./sst/catalog/reduce_motion.mjs";

sst.reduce_motion()
```

The `sst.reduce_motion` function returns `true` if motion should be reduced in the user interface, or `false` if not.

This value may change over time.

In a browser, this function returns `true` if the current value of the `prefers-reduced-motion` media feature is `reduce`, or `false` if not.

6. CMD library

6.1. The `sst_find_java_home` function

CMD

```
sst_find_java_home
```

The `sst_find_java_home` function finds the best `JAVA_HOME` directory on the system. If the function exits with error level 0, then `JAVA_HOME` will have been set appropriately in the calling context.

6.2. The `sst_find_java` function

CMD

```
sst_find_java
```

The `sst_find_java` function finds the best `java` executable on the system. If the function exits with error level 0, then `JAVA` will have been set appropriately in the calling context.

7. jq library

7.1. The `sst_assert` function

jq

```
sst_assert($x; $message)
```

The `sst_assert` function calls `error($message)` if `$x` is a value other than `false` or `null`. Otherwise, the function produces its input unchanged.

7.2. The `sst_nwise` function

jq

```
sst_nwise($n)
```

The `sst_nwise` function splits an array into partitions of size `$n` and outputs the partitions in an array.

`$n` must be a positive integer.

If the input array is empty, the result is an empty array.

If the input array does not contain a multiple of `$n` elements, the last partition will only contain the last `length % $n` elements.

7.3. The `sst_adjacent_pairs` function

jq

```
sst_adjacent_pairs()
```

The `sst_adjacent_pairs` function enumerates all adjacent pairs of elements of an array and outputs the pairs in an array.

If the input array has fewer than two elements, the output array will be empty.

7.4. The `sst_graphviz_escape` function

jq

```
sst_graphviz_escape()
```

The `sst_graphviz_escape` function escapes a string for use in Graphviz, without surrounding it with quotes.

7.5. The `sst_graphviz_quote` function

jq

```
sst_graphviz_quote()
```

The `sst_graphviz_quote` function escapes a string for use in Graphviz, including surrounding it with quotes.

7.6. The `sst_html_escape` function

jq

```
sst_html_escape()
```

The `sst_html_escape` function escapes a string for use in HTML.

Index

[1] For more information on subshells, see POSIX.1-2001 (2004 Edition) Shell & Utilities §2.12, POSIX.1-2008 (2008 Edition) Shell & Utilities §2.12, POSIX.1-2008 (2013 Edition) Shell & Utilities §2.12, POSIX.1-2008 (2016 Edition) Shell & Utilities §2.12, and POSIX.1-2017 (2018 Edition) Shell & Utilities §2.12.
[2] See POSIX.1-2001 (2004 Edition) Shell & Utilities §2.12, POSIX.1-2008 (2008 Edition) Shell & Utilities §2.12, POSIX.1-2008 (2013 Edition) Shell & Utilities §2.12, POSIX.1-2008 (2016 Edition) Shell & Utilities §2.12, and POSIX.1-2017 (2018 Edition) Shell & Utilities §2.12.
[3] For more information about fundamental integer types in C/C++, see C99 (N1256) §6.2.5, C11 (N1570) §6.2.5, C18 (N2176) §6.2.5, C++11 (N3337) §3.9.1, C++14 (N4140) §3.9.1, C++17 (N4659) §6.9.1, and C++20 (N4860) §6.8.1.
[4] For more information about C++20 requiring two's complement representation for signed integers, compare C++17 (N4659) §6.9.1 with C++20 (N4860) §6.8.1.
[5] For more information about negative zero trap representations in C/C++, see C99 (N1256) §6.2.6.2p{2,3,4}, C11 (N1570) §6.2.6.2p{2,3,4}, and C18 (N2176) §6.2.6.2p{2,3,4}.
[6] For more information about width bits in C/C++, see C99 (N1256) §6.2.6.2p6, C11 (N1570) §6.2.6.2p6 and C18 (N2176) §6.2.6.2p6
[7] For more information about value bits in C/C++, see C99 (N1256) §6.2.6.2p{1,2}, C11 (N1570) §6.2.6.2p{1,2}, and C18 (N2176) §6.2.6.2p{1,2}.