Introduction to the Intel® Numeric String Conversion Library

Intel® Numeric String Conversion Library (libistrconv) is a new component introduced in Intel® C++ compiler version 14.0 Update 1. This library provides a collection of routines for converting between ASCII strings of decimal numbers and C numeric data types. These routines provide similar functionality as the GLIBC functions strtol, strtoll, strtof, strtod, and snprintf, but are highly optimized for performance. Tables below are a partial list of libistrconv functions. For a complete list and discussion, refer to the Intel® C++ compiler reference manual.

Number to string conversion

libistrconv functions	Notes	GLIBC equivalents
int __IML_float_to_string (char *str, size_t n, int prec, float x)	Convert a single-precision floating point number to an ASCII string.	snprintf (str, n, "%.*g", prec, x)
int __IML_double_to_string (char *str, size_t, n, int prec, double x)	Convert a double-precision floating point number to an ASCII string.	snprintf (str, n, "%.*g", prec, x)
int __IML_int_to_string (char *str, size_t n, int x)	Convert a 4-byte signed integer to an ASCII string.	snprintf (str, n, "%d", x)
int __IML_uint_to_string (char *str, size_t n, unsigned int x)	Convert a 4-byte unsigned integer to an ASCII string.	snprintf (str, n, "%u", x)
int __IML_int64_to_string (char *str, size_t n, long long int x)	Convert a 8-byte signed integer to an ASCII string	snprintf (str, n, "%lld", x)
int __IML_uint64_to_string (char *str, size_t n, unsigned long long int x)	Convert an 8-byte unsigned integer to an ASCII string	snprintf (str, n, "%llu", x)

String to number conversion

libistrconv functions	Notes	GLIBC equivalents
float __IML_string_to_float (const char *nptr, char **endptr)	Convert an ASCII string to a single-precision floating point numbers.	strtof (nptr, endptr)
double __IML_string_to_double (const char *nptr, char **endptr)	Convert an ASCII strings to a double-precision floating point numbers.	strtod (nptr, endptr)
float __IML_str_to_f (const char *significand, size_t n, int exponent, char **endptr)	Convert an ASCII string to the significand of a single-precision floating point number, then multiply it with 10 to the power of exponent.	N/A
double __IML_str_to_d (const char *significand, size_t n, int exponent, char **endptr)	Convert an ASCII string to the significand of a double-precision floating point number, then multiply it with 10 to the power of exponent.	N/A
int __IML_string_to_int (const char *nptr, char **endptr)	Convert an ASCII string to a 4-byte signed integer.	strtol (nptr, endptr, 10)
unsigned int __IML_string_to_uint (const char *nptr, char **endptr)	Convert an ASCII string to a 4-byte unsigned integer.	strtol (nptr, endptr, 10)
long long __IML_string_to_int64 (const char *nptr, char **endptr)	Convert an ASCII string to an 8-byte singed integer.	strtoll (nptr, endptr, 10)
unsigned long long __IML_string_to_uint64 (const char *nptr, char **endptr)	Convert an ASCII string to an 8-byte unsigned integer.	strtoll (nptr, endptr, 10)

These functions behave similarly to their GLIBC equivalents (except for *__IML_str_to_f* and *__IML_str_to_d*). Users can check the man pages of *snprintf*, *strtof*, *strtod*, *strtol*, and *strtoll* for explanations on the arguments and return values. There do exist some key differences in functionality, though. The libistrconv routines are limited in conversion types, string formats, error checking, and locales:

- *__IML_float_to_string* and *__IML_double_to_string* perform only the '%g' type of conversion.
- All integer to string conversion routines only produce decimal notation.
- All string to number (integer or floating point) conversion routines only work with base 10 (decimal notation).
- These functions do not set *errno* or return an error code.
- These functions support only US English locale (en_us).

Routines *__IML_str_to_f* and *__IML_str_to_d* do not have GLIBC equivalents. But they are just a different interface of *__IML_string_to_float* and *__IML_string_to_double*. Instead of treating the string argument as the complete number representation, these routines treat the string argument as the significand of some number, and then combine the information of the exponent argument to produce the final result.

Using libistrconv is straightforward (See a code sample in the Intel C++ Compiler 14.0 reference manual):

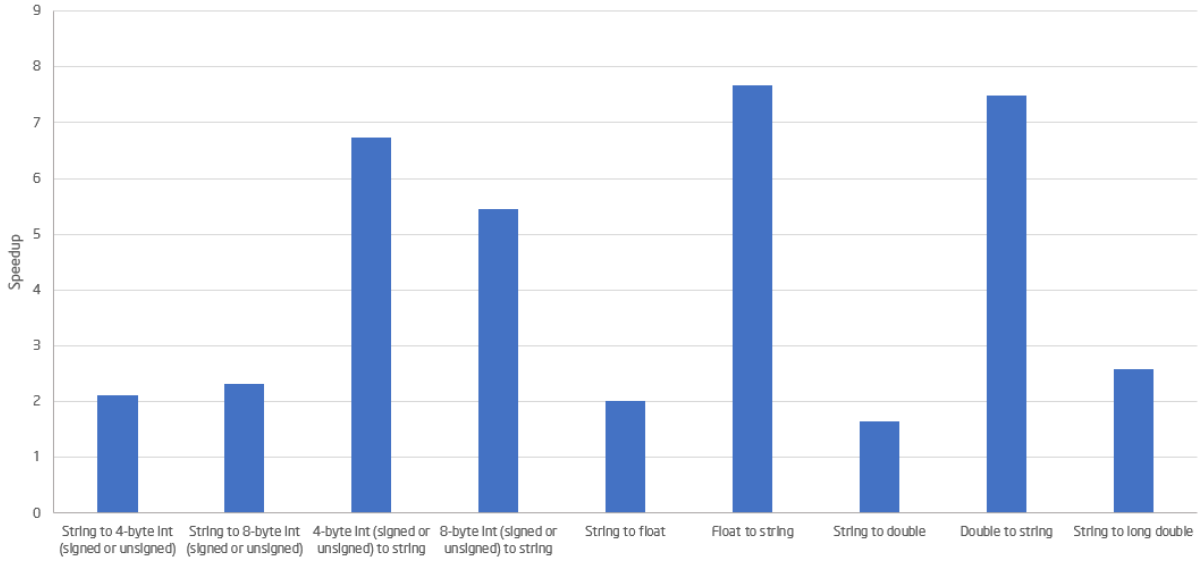
- Including the "istrconv.h" header file in the source code.
- Making calls to these routines as needed.
- Compiling the code using a C or C++ compiler.
- Linking with libistrconv library statically or dynamically. (On Windows* platforms, only static linking is supported).

Although libistrconv is included in the Intel® C++ compiler, you can use a different compiler, for example, GCC or Microsoft* C/C++ compiler, to link with it. But if a non-Intel compiler is used then you will need to have the CPU dispatcher library from the Intel® C++ compiler on the link line. The CPU dispatcher is libintlc.so (dynamic linking) or libintlc.a (static linking) on Linux*, and libirc.lib on Windows*.

Performance

Benchmarking results show that Intel® libistrconv routines significantly outperforms the equivalent routines provided by GLIBC and Microsoft* C/C++ compiler. See the performance comparison below on Linux*, Mac* OS X, and Windows*.

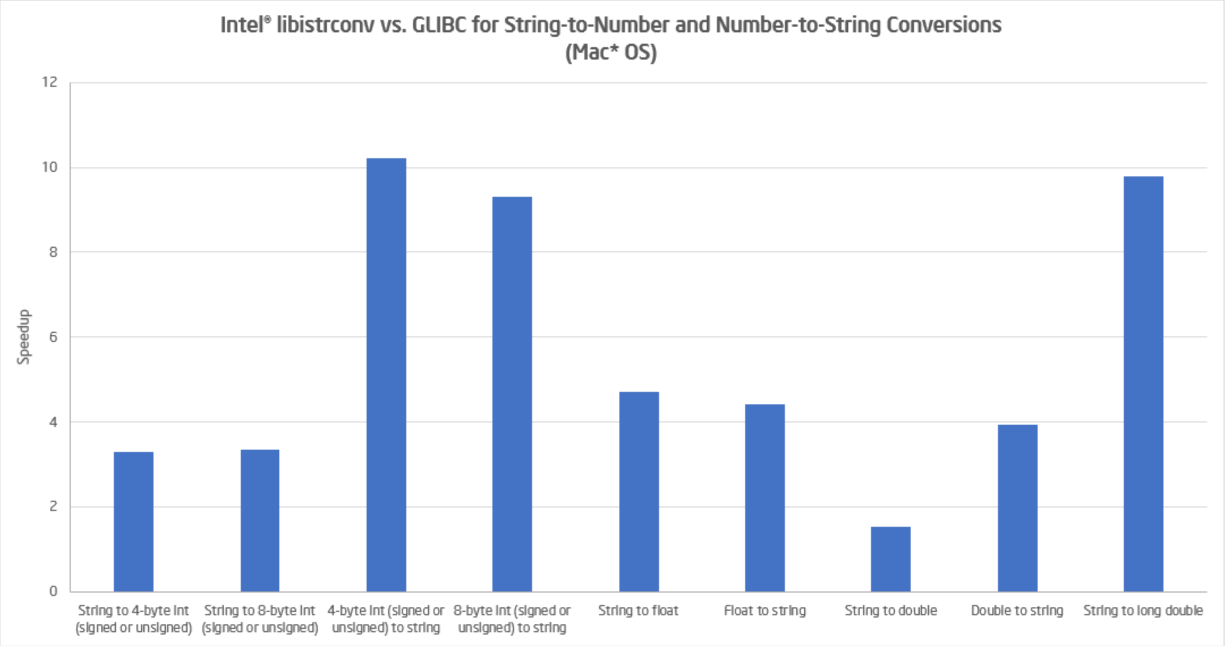
Intel® libistrconv vs. GLIBC for String-to-Number and Number-to-String Conversions (Linux* OS)



Configuration Info - Versions: Intel® C++ Compiler XE 2015 Beta, GCC 4.4.6; Hardware: Intel® Xeon® Processor E5-2687W, 2 Eight-Core CPUs (20MB LLC, 3.1GHz), 32GB of RAM; Operating System: RHEL 6 GA x86_64; Benchmark Source: Intel Corporation. The reported speedup for each conversion is the geomean of measurements for different value ranges.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation.

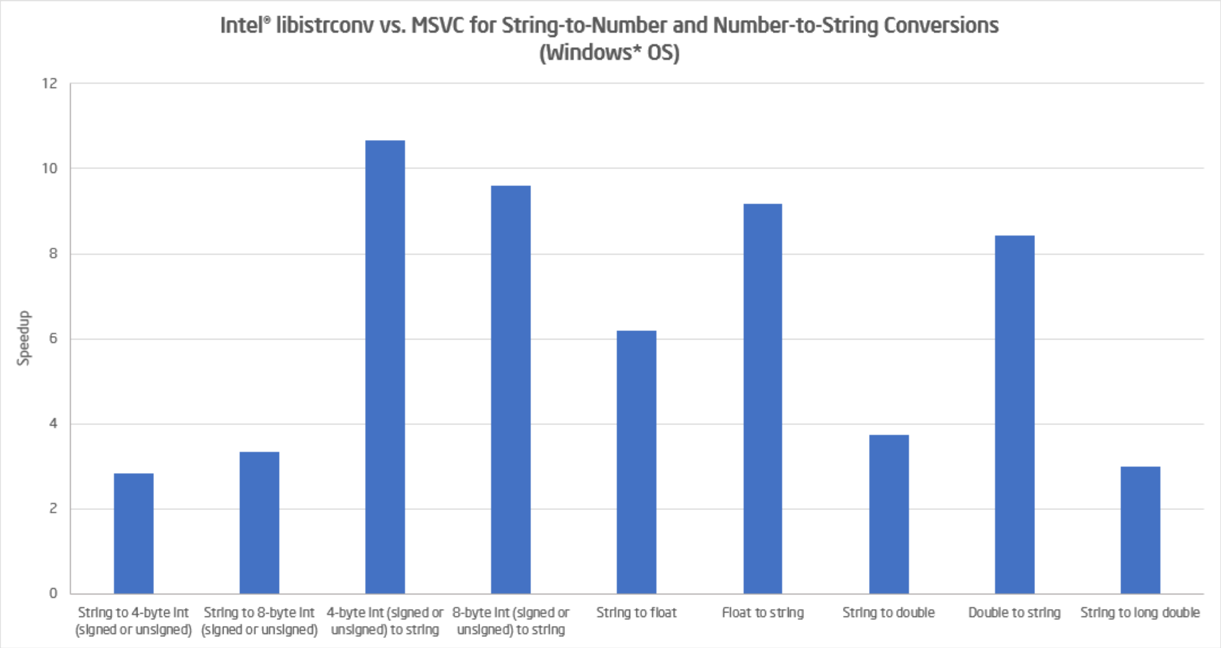
Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804



Configuration Info - Versions: Intel® C++ Compiler XE 2015 Beta, i686-apple-darwin11-llvm-gcc-4.2 (GCC) 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2336.11.00) ; Hardware: Intel® Core Processor i7-2635QM, Quad-Core CPU (6MB LLC, 2.0GHz), 8GB of RAM; Operating System: Mac* OS X 10.0 (Darwin); Benchmark Source: Intel Corporation. The reported speedup for each conversion is the geomean of measurements for different value ranges.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804



Configuration Info - Versions: Intel® C++ Compiler XE 2015 Beta, Microsoft* C/C++ Optimizing Compiler Version 15.00.21022.08 for x64 ; Hardware: Intel® Core Processor i7-2700K, Quad-core CPU (8MB LLC, 3.5GHz), 8GB of RAM; Operating System: Windows Server 2008 R2 x64; Benchmark Source: Intel Corporation. The reported speedup for each conversion is the geomean of measurements for different value ranges.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Notes

- Routines for conversions between strings and floating-point numbers were introduced in Intel® C++ Compiler 14.0 Update 1.
- Routines for conversions between strings and integers were introduced later in Intel® C++ Compiler 14.0 Update 3.