Release Notes

SAS/C® Compiler 7.50

The SAS/C and SAS/C++ team announces the availability of SAS/C Compiler 7.50. This release builds on the tradition of conformance with C/C++ standards, implementation of features requested by our software partners, and implementation of features that exploit advances in processor technology. The following are key features of this release:

- C/C++ Compiler and Library support for 64-bit z/Architecture
- C/C++ Compiler and Library support for IEEE Floating-point and C99 math libraries
- Architecture level compiler option to optimize programs for processor variants
- Multi-heap support for finer-grain control of storage allocation, oriented toward 64-bit and multitasking applications
- Debugger enhanced for IEEE floating-point and 64-bit addressing features

The release also targets optimizations intended to improve runtime performance, decrease storage requirements, and provide developers with a more robust feature set that may be tailored for the unique requirements of systems developers.

The following information highlights the new features of SAS/C Compiler 7.50. Developers should refer to the *SAS/C 7.50 Changes and Enhancements* publication for a complete description of the new features.

SAS/C Compiler and Utilities Enhancements

- Support for 64-bit addressing mode under z/OS: The `HUGEPTRS` compiler option directs the compiler to generate object code that runs in 64-bit addressing mode. At execution time, applications can allocate and utilize memory "above the bar" that is accessible with a 64-bit address. The implementation for 64-bit support is LP64, where the "long" data type is 64-bits. 64-bit and 31-bit object code may safely co-exist in the same load module, thereby allowing applications to be migrated gradually to a 64-bit environment.
- IEEE Floating-Point support: A binary floating-point compiler option allows the generation of code supporting IEEE floating-point, rather than the traditional mainframe hex floating-point. While most programs will use one floating-point format exclusively, the compiler and library support mixed-mode applications. This allows advanced applications, such as subroutine libraries and debuggers, to use whichever floating-point format is preferred by the end user. IEEE support provides greater portability for numeric programs to S/390 or z/Architecture environments.
- Architecture Level support: The `ARCHLEVEL` compiler option allows object code to be tailored to a specific execution environment. This feature allows applications to be optimized for processor variants and improved runtime performance.
  - `ARCHLEVEL(a)` allows the use of "logical string assist" instructions.
  - `ARCHLEVEL(b)` allows the use of the "relative and immediate" instruction facility.
  - `ARCHLEVEL(c)` allows the use of new floating-point registers and instructions defined by the "floating-point extensions" facility.
  - `ARCHLEVEL(d)` allows the use of z/Architecture instructions and 64-bit registers.

- Global Optimizer support: The optimizer supports z/Architecture instructions and sixty-four bit addressing:
  - Register variable assignment for 64-bit pointers and 64-bit integers
  - Constant propagation, folding and merging of sub-expressions for 64-bit computations
  - Loop transformations allowing the code generator to exploit BXLEG and BCTG instructions
- C99 support for the `#pragma` FENV_ACCESS directive. This enables the user to enable or disable compiler constant folding and defer execution of floating-point operations until runtime. This provides improved accuracy and error detection in IEEE math for non-default rounding modes and exception masks.
- C99 preprocessor enhancements: Support for variadic macros using the identifier `__VA_ARGS`. Variadic macros enable developers to write more readable and streamlined code sequences. Support for the `_Pragma` statement which allows `#pragma` directives to be produced by macro expansion.
- Implementation of C99 `<stdint.h>` header file: This support enhances portability of numeric programs and defines data types that make structures compatible with both 64-bit and 31-bit addressing.
- C99 support for "mixed declarations" anywhere within a code segment, as well as declaration support within a "for" statement when the `C99SUBSET` option is enabled.
- New compiler option, `(NO)ASYNSIG`, which informs the compiler if asynchronous signal support is required. When `NOASYNSIG` is enabled, the compiler can generate improved code sequences that result in reduced execution time.
- Redesigned `switch()` statement algorithms which improve program execution performance and reduce generated code segment size in some cases.
- Implementation of several new keywords and compiler options:
  - `stmap` compiler option that maps out offsets and lengths for structures in compiler listing.
  - #pragma `sname` directive that allows applications to define the `sname` value within source code. This directive permits the `SNAME` to be specified in the source code rather than in JCL or a makefile, for improved maintainability.
  - Implemented #pragma `options copts()` support for `INDEP` and `ARMODE`. These keywords allow the `INDEP` and `ARMODE` options to be specified on a per-function basis rather than for an entire source file.
- `PPONLY` option improvement: When the `PPONLY` option is used, #line directives are now maintained. This improves the usefulness of the generated source code.
- Updates to inline machine code interface:
  - Header files modified to define z/Architecture instructions
  - Added the new inline code function `_osa64svc` to generate SVC calls that execute in 64-bit addressing mode
  - Added `_ospc` to generate the PC instruction
- Instrumentation support for coverage analysis, which enables automation of testing and instrumentation of code sequences.
- Improved compiler diagnostics: Improved error processing reduces the number of diagnostics generated for erroneous statements, thereby facilitating identification and repair of errors.
- Improved diagnostics for erroneous uses of inline machine code. Also, new warnings for uses of inline machine code that may behave unexpectedly at execution time.
- Enhanced the `DSECT2C` utility to support 64-bit z/Architecture instructions.
- Enhancements to Object Module Dis-assembler (OMD) for support of C++ template objects.
- Cross-compiler support for user-defined translate tables for input source code.
- Cross-platform compiler enhancements to support Windows 2000, Windows registry enhancements, and improved help dialogues.

SAS/C C++ Development System Enhancements

- Support for new data types and runtime interfaces for z/OS. Specifically:
  - IEEE floating-point support
  - 64-bit addressing mode
  - Access Register mode
  - `ARCHLEVEL` options
- Language Enhancement and Extensions for C++
  - Nested template definitions and partial specializations of class templates are supported.
  - Members (including templates) of class templates may be defined outside the enclosing template.
  - Template partial ordering rules are applied to function overload resolution.
  - Template conversion functions and template member friend declarations are supported.
  - Virtual function overrides may have covariant return types.
  - Option `NOFRIENDINJECT` disables friend name injection.
  - Support `NOBITFIELD` option to apply C++ standard rules for bitfield widths.
- Enhanced pre-processor features and support
  - Implemented `#pragma options copts()` including support for `INDEP`, `ARMODE`, `SNAME` and associated pre-processor symbols.
  - Implemented `#pragma space`, `eject`, `title`.
- Standard Template and Tools++ library updated to support 7.50 C++ enhancements with 32-bit pointers and S/390 floating-point format.
- Usability Enhancements
  - Interleave errors and warnings in compilation listing output
  - Support `TERM` and `NOOPTIONS` options for listing output
- Optimization and performance enhancements
  - Support `INLINE` option to complement the global optimizer inline processing
  - Reduced memory requirements for processing initializer lists and template definitions
  - Reduced the size of object files generated when auto instantiation and debugging are enabled
  - Optimization of empty base class definitions and more efficient class object organization
  - Optimization added to elide user-defined destructors with empty bodies

SAS/C Runtime Library Enhancements

- IEEE floating-point support for `printf()`, `strtod()`, `scanf()` and related functions. Support has been added for the C99 function `vscanf()` (and related functions) for both IEEE and traditional mainframe floating-point.
- 64-bit library support: Support for 64-bit support has been added to much of the runtime library, including the string library and the standard I/O library (including `printf` and `scanf`). Support is also available for system interfaces such as `TPUT/TGET` and `WTO`. 64-bit support for non-I/O functions is available in `SPE`, as well as the full runtime library.
- malloc multi-heap support: This support permits creation of multiple heaps for flexibility in managing allocated storage. Heaps may be created with attributes, including addressing mode and subpool. Support is included for shared heaps, permitting storage management to be shared between tasks in the same address space.
- C99 Math Library: The SAS/C library has been augmented to include additional math functions defined in the C99 standard, such as `log2`, `remainder`, `round`, `nan` and many others. All functions are supported for traditional floating-point and IEEE floating-point. Additionally, the header file `<tgmath.h>`, providing type-generic math functions, is implemented.
- IEEE trapping support. The functions `fesettrapenable` and `fegettrapenable` have been added to allow control of trapping for IEEE exceptions. If trapping is enabled, either the standard signal `SIGFPE` or the non-standard signal `SIGBFPE` is raised. The signal handler can access the hardware status bits, and optionally correct the problem and return to the point of the exception.

- Implementation of an Operator Command Support interface. This interface allows the application to respond to operator `MODIFY` and `STOP` commands. The feature is enabled with the `opcmd()` function and the use of the `SIGOPER` asynchronous signal.
- Library Message Exit facility: This serviceability enhancement allows applications finer control over runtime diagnostic processing. This allows an application to selectively suppress, enhance or redirect library diagnostics for more precise control of runtime message processing.
- Implementation of an interface to OS/390 `NAME/TOKEN` services. Functions are provided to create, retrieve, and delete named tokens.
- A new runtime option, `=rsntrace` enhances diagnostic support to include low-level information about failing system macros. The additional information can be helpful in isolating problems with complex system interfaces.
- Modifications to the all-resident library to permit the use of `RMODE(SPLIT)` with all-resident programs.
- Enhancements to the `alarmd()` and `sleepd()` functions to improve execution runtime performance.
- Implementation of a C interface to the z/OS `IARV64` macro for allocation of storage above 2 gigabytes.
- Implementation of USS name resolution interfaces to the `gethostbyname()` and `gethostbyaddr()` services.
- Support for the z/OS System Resolver address space, announced in z/OS V1R2

Debugger Enhancements

- Support for IEEE and 64-bit data types within all combinations of debugger environments.
- Implementation of support for C++ namespace semantics for numerous debugger commands such as: `assign`, `break`, `run`, `monitor`, `print`, and `list`.
- Debugger `namespc` command: This command generates a listing of all or selected namespaces with associated details, including a breakdown of identifiers, variables, classes, and functions.

CICS Enhancements

- The CICS Command Language Translator supports the latest versions of CICS Transaction Server for z/OS V2R2. Transaction Server support includes processing for the Application Programming Interface, System Programming Interface, and Business Transaction Services.
- Implementation of new cross-compiler option to enable CICS pre-processing on user-specified source modules.

The following URLs may be used to access SAS/C and SAS/C++ Compiler information on the SAS Web server. Please contact the SAS/C Technical Support Division if you have additional questions:

- SAS Institute Home Page: `http://www.sas.com`
- SAS/C Compiler Products Home Page: `http://www.sas.com/sasc/`
- SAS Customer Support Center: `http://support.sas.com/`

March 16, 2004