DWARF3: Better than DWARF2

David B. Anderson

December 1,2005

Abstract

The Debugging Information Format DWARF Version 3 is an enhancement of DWARF Version 2. DWARF3 has new features for correctly representing everything in the current C++ and C and Fortran standards. DWARF Version 3 provides new features to allow significant space-compression and allows generation of debug-information larger than 4GBytes. Yet it is compatible with DWARF Version 2 in that a DWARF reader (such as a debugger) can easily read both DWARF Version 2 and DWARF Version 3. DWARF Version 3 provides some basic support for and eliminates obstacles to using DWARF for Ada and Java and COBOL.

1 Introduction

A debugger, such as dbx or gdb, requires debugging information and DWARF is an information format in wide current use. DWARF Version 2 (DWARF2) was published in 1993 and recent standards developments encouraged the DWARF committee to reform and to update DWARF.

Volunteers from various companies participated beginning in 1999, culminating in the January 2002 release of the DWARF Version 3 (DWARF3) document for public comment. In 2004 discussion resumed at http://dwarf.freestandards.org resulting in a revised public comment document (public review 18 October 2005 thru 1 December 2005). Committee membership was open to anyone throughout the process.

Here we describe the new features of DWARF3 and mention some corrections and clarifications. We are assuming familiarity with the terminology of DWARF2. We refer to the 1999 C standard as C99. We refer to the C++ Standard as C++. We refer to the Fortran 90 and 95 standards as Fortran.

2 Overriding Goal

The intent of the committee was to preserve compatibility with DWARF2. Consequently the recording format was not changed. By the end of the deliberations enough had been changed that the committee changed the DWARF version numbers and renamed it DWARF3. This was not an easy decision: there was considerable sentiment to keep the existing version number(s). However in the end consensus was reached that version numbers should change. An existing consumer (such as a debugger) will therefore not be able to use DWARF3. However it is easy for a slightly modified consumer to read DWARF2 and DWARF3 mixed into the same executable, so backward compatibility is maintained.

One impetus for the version change was that the C++ changes meant a DWARF2 consumer would be completely unable to get any useful info from a compilation unit which implemented DWARF C++ namespace support.

3 Major New Features

3.1 C++ , including Namespaces

DWARF2 was completed before the C++ Standard and before C++ namespaces were even considered. DWARF3 provides a complete set of features using DW_TAG_namespace, DW_TAG_imported_declaration, DW_AT_import, and DW_AT_extension that enables an implementation to represent the visible namespaces correctly in every function. Implementations may choose to emit a single namespace declaration showing the complete namespace at the end of the compilation unit as this is simpler, though it loses some of the details of some uses of C++ Namespaces.

3.2 Fortran 90 allocated and pointer data

Fortran 90 allocatable and pointer data could not be described in DWARF2. Such dynamically allocated arrays and pointers that can be associated at run time mean that there are run-time data structures pointing to the actual run-time data.

DWARF3 provides the DW_AT_data_location attribute and the expression operator DW_OP_push_object_address. DW_AT_data_location is a location expression that both defines this as having run-time structures and specifies the address of the run-time-structures (commonly called dope vectors and described in DWARF3 as descriptors). DW_OP_push_object_address provides the expressive capability in a location expression to describe the data as distinct from the run-time data structures.

DW_AT_associated and DW_AT_allocated attributes provide addresses or expressions that result in deriving a non-zero value if the array or pointer is actually associated or allocated at the time of the evaluation.

The run-time data structures that have to be there anyway for the run-time to work and for a debugger to work can be described directly in DWARF3 without a need for the debugger to have apriori knowledge of the run-time-data-structures.

3.3 Fortran subroutines

DW_AT_elemental, DW_AT_pure, DW_AT_recursive were added to allow these Fortran subroutine descriptive keywords to be represented.

3.4 Subroutine calls in expressions

Where DWARF2 spoke of Location Expressions, the DWARF3 document generalizes this somewhat to describe DWARF Expressions separately and then to define Location Expressions in terms of DWARF Expressions.

If there are many common sequences in DWARF expressions it can be a large space saving to use DW_OP_call2, DW_OP_call4, or DW_OP_call_ref to call a DWARF Expression subprogram. And this commonization can be carried across compilation units and across shared-libraries 3.6. Because in come situations there is no 'obvious' place to put the called DWARF Expression, DW_TAG_dwarf_procedure was defined as a TAG to hold a DW_AT_location expression to be called.

3.5 DWARF Compression

DWARF2 provided no recognizable means to avoid duplicating DWARF information. DWARF3 provides the means by defining DW_TAG_partial_unit and DW_TAG_imported_unit and providing an explanation and examples in an appendix. Because much of this involves object format issues and is outside of DWARF3, the explanation is a template offering means implementations can choose to use, not a detailed recipe.

An appendix to the DWARF3 document explains how a C or C++ implemenation could wind up with only a single copy of a header file in the debug information. It also demonstrates how the same basic approach allows eliminating duplicate functions (as might arise from C++ templates) and unused functions from the DWARF3 debug information for an executable or dynamic-shared-library.

The appendix also shows how Fortran common could be treated to eliminate duplicate DWARF3.

3.6 References Across Shared-Libraries

DWARF2 had DW_FORM_ref_addr for references between compilation units, but the documentation of it was difficult to interpret. Moreover the explicit specification of an address-size value of the reference was not useful. DWARF3 makes it clear that these references can be between compilation units even if the compilation units are in different dynamic-shared-objects. And DWARF3 specifies that the size of the field is an offset-size. References from one dynamic-shared-object to another requires relocations to be done by the debugger since only the debugger knows where each dynamic-shared-object is at run time. Defining these relocations (what they look like, how to implement them) is outside of DWARF, but the intent to allow such references is clearly specified.

3.7 64-Bit File Offsets

While few collections of debugging-information exceed a 32 bit offset today, real examples do come close (exceeding 30 bits of offset). Such a large debugging-information collection cannot be represented in DWARF2. So an extension was added, usurping 255 values as 'escape codes' and allowing vendors to emit DWARF3 with 32-bit-offsets when they are confident that is adequate and to emit DWARF3 with 64-bit-offsets when they think it advisable to do so. Mixing 32-bit-offset DWARF with 64-bit-offset DWARF is simple and requires no special action on the part of producers (compiler vendors) or consumers (debuggers). Producers and consumers that have no interest in 64-bit-offsets can completely ignore the 64-bit-offset extension and need not code for it.

This has nothing to do with 64-bit-addresses. DWARF2 was always perfectly capable of representing objects with 64-bit-addresses and DWARF3 retains that ability.

There are no specific TAGs or Attributes relating to 64-bit-offsets. If offsets do exceed 64-bits in an executable using 32-bit-offset-DWARF and some offset cannot be represented properly in DWARF it is a quality-of-implementation issue whether the static linker warns of the problem.

3.8 COBOL datatypes

New base types were added (example:DW_ATE_packed_decimal) along with attributes such as DW_AT_binary_scale, DW_AT_decimal_scale, DW_AT_small, DW_AT_decimal_sign, DW_AT_digit_count, and DW_AT_picture_string to allow normal COBOL datatypes to be described. New decimal sign attribute values such as 'DW_DS_trailing_overpunch' are also part of this COBOL type suppport. These are based on actual compiler implementations so are known to be adequate when combined with other existing DWARF3 features.

4 Minor Enhancements

4.1 Describing Void *

DWARF2 provided a specific means to describe a C 'Function Returning void' (which DWARF3 retains) but was silent about describing C 'void *'. DWARF3 provides a language-independent means to describe such, using DW_TAG_unspecified type to describe the language-notion and DW_AT_name of 'void' in the C/C++ case as the referent of DW_TAG_pointer_type.

4.2 Inlining information

An appendix gives examples and interpretations of how to represent inlines in messy cases.

DWARF2 provided no way to describe the *caller location* at the site of an inlined-function. DWARF3 provides DW_AT_call_file, DW_AT_call_line, DW_AT_call_column for those implementations wishing to provide this information.

4.3 New Data Type

C99 defines the data type _Imaginary and DWARF3 defines DW_AT_imaginary_float to describe this type.

The C++ keyword mutable is representable with DW_AT_mutable_type.

4.4 Function Prologue and Epilogue descriptions

In DWARF2, debuggers which wished to have function-entry-breakpoints set after the function prologue had run (copying incoming arguments to local storage, saving registers, etc) had to use heuristics to find a place to set such a breakpoint. For example, using line table information (which was dependent on the details the compiler used in emitting the line information, so it was compiler dependent). In DWARF3, the line table may contain a DW_LNS_set_prologue_end flag at the end of the prologue, providing debuggers a precise address to set the breakpoint.

In DWARF3 the line table may contain 1 or more DW_LNS_set_epilogue_begin flags per function. Each such identifies an address where a debugger may set a breakpoint 'just before the function returns', again providing a language- and compiler-independent means of describing such points (many compilers emit multiple return sequences for functions where such improves performance of the application).

4.5 ISA description

If an executable may contain instructions from distinct ISAs (perhaps some ISA for packing multiple fields into words, for example) the DW_LNS_set_isa flag in the line table may be used to describe exactly which ISA is in use at which addresses. ISA identities are vendor-defined, not specified in DWARF3.

4.6 New Languages

Specific codes DW_LANG_Java, DW_LANG_C99, DW_LANG_Ada95, DW_LANG_Fortran95, DW_LANG_PLI DW_LANG_ObjC, DW_LANG_ObjC_plus_plus, DW_LANG_UPC, and DW_LANG_D were added so vendors need not define extensions for these language names: implementations are known to be planning to use the last four.

4.7 Frame Description enhancements

There were two problems with DWARF2 frame descriptions.

First, DWARF2 provided no means for using DWARF expressions in a frame description, which was a problem for certain unusual architectures. DWARF3 provides DW_CFA_def_cfa_expression and DW_CFA_expression for those implementations that require it.

Second, DWARF2 provided no means for describing stack-frames with data both above and below the CFA (virtual frame pointer for the frame). DWARF3 provides DW_CFA_cfa_offset_extended_sf, DW_CFA_def_cfa_sf, and DW_CFA_def_cfa_offset_sf allowing a consise representation for such a stack frame description. These three operators are not strictly necessary since the DW_CFA_def_cfa_expression and DW_CFA_expression provide enough expressiveness, but the *_sf forms were sufficiently more space efficient that they were adopted.

4.8 Trampoline

DWARF2 provided no means to identify compiler-created code for calls to functions in dynamic-shared-libraries (often called 'stub code' or 'trampoline') or to identify code used to implement stack unwinding for exception handling. DWARF3 allows an implementation to emit the DW_AT_trampoline attribute to identify such code so a debugger can make a decision about how to deal with it.

4.9 UTF8

DWARF2 provided no means to deal with multibyte characters. DWARF3 provides DW_AT_use_UFT8 which is a flag telling the debugger that all strings in this compilation unit are UTF8 multibyte strings. This attribute only appears in the .debug_info section but applies to all strings for this compilation unit in all DWARF3 sections having strings.

4.10 Non-contiguous Functions

DWARF2 provided no means to deal with non-contiguous functions. Such functions might result from optimizations moving 'low frequency' code off away from the main high-frequency code. For example, many error-message situations never ever arise. One result of such an optimization is a reduced working set size.

DWARF3 provides DW_AT_ranges as an alternative to the simple contiguous-function DW_AT_low_pc DW_AT_high_pc attributes. DW_AT_ranges refers to the new DWARF3 object file section .debug_ranges, where the ranges are encoded. DW_AT_ranges can be used anywhere DW_AT_low_pc DW_AT_high_pc would appear. Compilers are known to mix together code from DW_TAG_lexical_block, DW_TAG_inlined_subroutine, DW_TAG_try_block, DW_TAG_catch_block so that these may contain non-contiguous code sequences.

4.11 Pubtypes

DWARF2 had no special means of mentioning globally-distinct types, such as C++ classes, which are guaranteed by the language to be unique and identical across compilation-units. DWARF3 defines the new section .debug_pubtypes (with a format identical to .debug_pubnames used for global variables) providing the debugger with a means for fast lookup (given a class name find the right compilation unit).

5 A selection of corrections

5.1 Spelling

DW_TAG_template_type_parameter and DW_TAG_template_value_parameter were also (inconsistently) spelled DW_TAG_template_type_param and DW_TAG_template_value_param in DWARF2. DWARF3 uniformly uses the longer spelling.

DW_AT_bit_stride is a new spelling for what used to be spelled DW_AT_stride_size: it always represented a bit stride, so the new name is simply clearer, and the (new) alternative attribute DW_AT_byte_stride is available when a byte stride suffices.

6 Format incompatibilities

There are only three changes directly affecting the format of the DWARF data , all mentioned in DWARF3 section 1.5.1.

6.1 Large Initial Length

Certain (large) values of the initial length field used in various DWARF sections were reserved as escape codes 3.7. Because no known instances of DWARF data with lengths within 255 bytes of the maximum offset recordable in 4 bytes exist this should have no practical impact on the interpretation of DWARF information in existing object files.

6.2 DW_FORM_ref_addr

DW_FORM_ref_addr was defined in DWARF2 as being the size of a target-machine address. DWARF3 defines this as a section offset, which can be 32 or 64 bits3.7.

6.3 CIE return address register field

The return-address-register field in the Common Information Entry(CIE) in the .debug_frame section was defined as an unsigned byte in DWARF2. This field is now defined as an unsigned LEB128 field. The change was made as the definition seemed pointlessly constraining to newer CPUs with large numbers of registers that might want a larger return-address-register designation. None of the implementations currently using dwarf frame-description information are known to have needed a number here greater than 127. The actual bits recorded are the same for both field definitions with the return-address-register value less than 128 (see the LEB128 definition in the DWARF 2 or DWARF3 documents) so all existing object files would show the same bit pattern with either definition. So no actual binary incompatibility applies to existing implementations.

7 New TAGs, Attributes, etc in brief

Some of the new features are not just TAGS and attributes, but all the new entities given assigned values in DWARF3 are listed here with a short description.

- DW_TAG_dwarf_procedure 0x36, used as a placeholder for DW_OP_call*ed subroutines.
- DW_TAG_restrict_type 0x37, restrict is a a C99 keyword.
- DW_TAG_interface_type 0x38, for Java interface types.
- DW_TAG_namespace 0x39, used with C++ namespaces
- DW_TAG_imported_module 0x3a, for example, used with Fortran modules.
- DW_TAG_unspecified_type 0x3b, used for C 'void' for example.
- DW_TAG_partial_unit 0x3c, used in compressing DWARF3 (eliminating duplicate DWARF).
- DW_TAG_imported_unit 0x3d, used to reference a normal or partial compilation unit that logically belongs 'inside' the referencing compilation unit at the point of the reference.
- DW_TAG_mutable_type 0x3e, was in DWARF3 but has been withdrawn as it was incorrect, DO NOT USE. See DW_AT_mutable instead.
- DW_TAG_condition 0x3f, describes a COBOL level-88 condition.
- DW_TAG_shared_type 0x40, indicates the UPC 'shared' qualifier applies to a type.
- DW_AT_bit_stride 0x2e, is a changed spelling: used to be spelled DW_AT_stride_size, the new name indicates the true meaning better.
- DW_AT_allocated 0x4e, used with Fortran allocated data.
- DW_AT_associated 0x4f, used with Fortran associated data.
- DW_AT_data_location 0x50, used with Fortran allocated and associated data.

- DW_AT_stride 0x51, has been removed from DWARF3 (it was never in DWARF2). DO NOT USE. See DW_AT_bit_stride and DW_AT_byte_stride instead.
- DW_AT_entry_pc 0x52, for functions whose entry point is not the lowest address in the function.
- DW_AT_use_UTF8 0x53, to signal that all strings in the compilation unit are UTF8 multibyte form (the only clue UTF* is in use).
- DW_AT_extension 0x54, a general purpose attribute, contents vendor defined.
- DW_AT_ranges 0x55, reference to a new section allowing function code to be non-contiguous.
- DW_AT_trampoline 0x56, identifies a function as being compiler generated such as dynamic-shared-library stub code or exception-handling code.
- DW_AT_call_column 0x57, identifies the column of the call site (not called routine) for more precise debugging of inlined functions.
- DW_AT_call_file 0x58 , identifies the file of the call site (not called routine) for more precise debugging of inlined functions.
- DW_AT_call_line 0x59, identifies the line of the call site (not called routine) for more precise debugging of inlined functions.
- DW_AT_description 0x5a, for compiler augmented descriptions of an entity.
- DW_AT_binary_scale 0x5b, for types with a binary scale factor.
- DW_AT_decimal_scale 0x5c, used with DW_ATE_packed_decimal and DW_ATE_numeric_string.
- DW_AT_small 0x5d, for data types with arbitrary scale factors (defined with the Ada 'small' attribute in mind).
- DW_AT_decimal_sign 0x5e, used with DW_ATE_packed_decimal and DW_ATE_numeric_string.
- DW_AT_digit_count 0x5f, used with DW_ATE_packed_decimal and DW_ATE_numeric_string.
- DW_AT_picture_string 0x60, used with DW_ATE_edited base type.
- DW_AT_mutable 0x61, represents the C++ 'mutable' keyword.
- DW_AT_threads_scaled 0x62, used to indicate a subrange is scaled by a thread number (in UPC).

- DW_AT_explicit 0x63, indicates a C++ member function has the 'explicit' property.
- DW_AT_object_pointer 0x64, a reference to a non-static C++ member function parameter representing the object the member applies to.
- DW_AT_endianity 0x65, indicates the endian-ness of a variable or compile-unit is different than the object file and ABI would normally suggest.
- DW_AT_elemental 0x66, indicates a Fortran function has the 'elemental' property.
- DW_AT_pure 0x67, indicates a Fortran function has the 'pure' property.
- DW_AT_recursive 0x68, indicates a Fortran function has the 'recursive' property (the flag is not used for C or C++ as those language functions default to being recursive).
- DW_AT_pointer 0x69, indicates a Fortran function with the 'pointer' property.
- DW_OP_push_object_address 0x97, used with Fortran allocated and pointer types to correctly calculate addresses of data.
- DW_OP_call2 0x98, call allows expression subroutines.
- DW_OP_call4 0x99, call allows expression subroutines.
- DW_OP_call_ref 0x9a, call allows expression subroutines
- DW_OP_form_tls_address 0x9b, is an operation whose implementation is left to user code: forming an address that depends on a value on the operation stack (needed for thread local variables).
- DW_OP_call_frame_cfa 0x9c, pushes the value of the CFA onto the operation stack.
- DW_OP_bit_piece 0x9d, like DW_OP_piece, but giving a bit offset.
- DW_ATE_imaginary_float 0x9, imaginary float is a new C99 data type.
- DW_ATE_packed_decimal 0xa, represents decimal string numeric data types, such as in COBOL.
- DW_ATE_numeric_string 0xb, represents decimal string numeric data types, such as in COBOL.
- DW_ATE_edited 0xc, represents a COBOL edited numeric or alphanumberic data type.

- DW_ATE_signed_fixed 0xd, represents a signed fixed point binary data type.
- DW_ATE_unsigned_fixed 0xe, represents an unsigned fixed point binary data type.
- DW_ATE_decimal_float 0xf, represents a decimal floating point number, such as may be specified in COBOL.
- DW_DS_unsigned 0x01, signifies decimal sign base type attribute 'unsigned' (all DW_DS_* used in COBOL, for example).
- DW_DS_leading_overpunch 0x02, signifies decimal sign is encoded in the most significant digit.
- DW_DS_trailing_overpunch 0x03, signifies decimal sign is encoded in the least significant digit.
- DW_DS_leading_separate 0x04, signifies decimal sign is a '+' or '-' to the left of the most significant digit.
- DW_DS_trailing_separate 0x05,signifies decimal sign is a '+' or '-' to the right of the most significant digit.
- DW_END_default 0x00, value of DW_AT_endianity, indicating default endian-ness.
- DW_END_big 0x01, value of DW_AT_endianity, indicating implementation defined big-endian.
- DW_END_little 0x02, value of DW_AT_endianity, indicating implementation defined little-endian.
- DW_LANG_Java 0x000b
- DW_LANG_C99 0x000c
- DW_LANG_Ada95 0x000d
- DW_LANG_Fortran95 0x000e
- DW_LANG_PLI 0x000f
- DW_LANG_ObjC 0x0010
- DW_LANG_ObjC_plus_plus 0x0011
- DW_LANG_UPC 0x0012
- DW_LANG_D 0x0013
- DW_LNS_set_prologue_end 10, used to precisely identify the end of the function prologue and the beginning of user code in a function.

- DW_LNS_set_epilogue_begin 11, used to precisely identify the end of user code and the beginning of the return point (multiple points if there is code generated for multiple returns) so a debugger can easily set a breakpoint before function return.
- DW_LNS_set_isa 12, allows precise description of which instructions are what instruction set architecture in systems using multiple instruction set architectures in a single executable.
- DW_LNE_lo_user 128, identify range of codes usable by the compiler implementor for vendor extensions.
- DW_LNE_hi_user 255, identify range of codes usable by the compiler implementor for vendor extensions.
- DW_CFA_def_cfa_expression 0x0f, allow for general expressions in frame descriptions.
- DW_CFA_expression 0x10, allow for general expressions in frame descriptions.
- DW_CFA_cfa_offset_extended_sf 0x11, allow more flexible frame descriptions, compactly.
- DW_CFA_def_cfa_sf 0x12, allow more flexible frame descriptions, compactly.
- DW_CFA def_cfa_offset_sf 0x13, allow more flexible frame descriptions, compactly
- DW_CFA_val_offset 0x14, uses unsigned factored offset to calculate the address of a location where a register is stored.
- DW_CFA_val_offset_sf 0x15, uses signed factored offset to calculate the address of a location where a register is stored.
- DW_CFA_val_expression 0x16, Like DW_CFA_expression, but computes the actual value, not the address where a value is stored.

8 Acknowledgements

While many people have participated in the development of DWARF3 and many have made significant contributions, two people have been instrumental in getting DWARF3 completed and deserve special mention.

Michael Eager, eager@eagercon.com, http://www.eagercon.com chaired the committee and provided a web site for distribution of the drafts.

Ron Brender edited the DWARF3 document, producing 9+ drafts and adding 100 pages of new/clarified content to the document.

9 Changes

November 2,2011: Fixed a spelling error.