

A COMPILER LEGACY SUPPORT

Overview

The VisualDSP++ environment and tools provide several types of support for legacy code that was developed with previous releases of DSP development software.

Tools Differences

VisualDSP++ 2.0 includes a new C Compiler, linker, debugger, and a binary file format, ELF. It also introduces an Integrated Development and Debugging Environment (IDDE) for easy management of your ADSP-218x projects.

There are a number of enhancements to other tools as well. As a result, VisualDSP++, Release 2.0 has some differences from Release 6.1 that you will need to be aware of. In some cases, you will need to modify your sources to use the new tools.

Of the new features and enhancements, the following have the most impact on your existing projects:

- Some tools' switches have changed. If you use any of the modified or obsolete switches, you must revise your command-line scripts or batch files in order to rebuild your project.
- The code generation tools no longer support AEXE-format DSP executables (.exe). They now generate ELF-format DSP executables (.dxe), and the debugger requires DSP executables to be in the

Tools Differences

ELF/DWARF-2 format. As a result, AEXE-formatted files must be recompiled or reassembled in order to be debugged under VisualDSP++ 2.0. An ELF/DWARF-to-AEXE conversion utility is available in VisualDSP++ 2.0 to perform back-conversion. An AEXE-to-ELF conversion utility performs forward conversion.

- Some assembly instructions and directives have changed from the VisualDSP 6.1 syntax, but a `-legacy` assembler switch has been provided to assemble files in the old syntax. You may need to review diagnostic messages and revise your source code in order to reassemble your source. Legacy syntax and the new syntax under VisualDSP++ 2.0 cannot be used together in the same source file. They can be mixed together within the same project, as long as they are assembled in different source files.
- Some C compiler extensions to the ISO/ANSI standard have changed. If you use any of the modified or removed extensions, you must revise your code in order to rebuild your project.
- The run-time model has changed. If you call a Release 6.1 assembly-language subroutine from your C program, you must revise the assembly code to comply with the new rules for the C run-time environment.
- The Architecture File (`.ach`) is no longer supported. If you re-link using your Release 6.1 object files or object libraries, you must create a Linker Description File for each object or object library before using the new linker.

The remainder of this section describes these and other known differences between VisualDSP Releases 6.1 and VisualDSP++ 2.0. It also provides assistance when possible to make transformation to the new software easier.

C Compiler and C Run-Time Library

The new cc218x compiler provided in VisualDSP++ 2.0 does not support some switches and extensions that were available in the g21 compiler. As a result, the compiler supports a set of new rules for the run-time environment. This section lists the extensions and switches that have been removed, replaced, or whose function works differently than in Release 6.1. For further details about the cc218x compiler, see [“Compiler” on page 2-1](#).

Segment Placement Support Keyword Changed to Section

The `segment()` placement keyword has changed to `section()`. The `section()` construct now precedes the variable declaration, and its argument is a string; for example:

```
section("my_sec") int myvar;
```

For more information about the `section()` construct, see [“Placement Support Keyword \(section\)” on page 2-55](#).

G21 Compatibility Call

The C compiler provides a special g21 compatibility call that enables use of existing libraries with the new compiler. The extern `OldAsmCall` declaration can be added to the prototype(s) of the functions developed under Release 6.1. Your programs will be faster, smaller, and more reliable after the C code is upgraded to use the new compiler.



This convention is similar to the C++/C linkage specification.

Support for G21-Based Options and Extensions

The C compiler supports most of the switches and extensions of the previous GNU-based compiler release. For a list of absolute or modified options, see [“Compiler Switch Modifications” on page A-5](#).

Indexed Initializers

The syntax for indexed initializers may change in a future release. This change may be incompatible with the existing syntax. For more information about indexed initializers, see [“Indexed Initializer Support” on page 2-59](#).

Compiler Diagnostics

Compiler diagnostics now go to `stderr` on the PC, rather than to `stdout`.

ANSI C Extensions

The following extensions are no longer supported:

- `typeof` — This extension was used to obtain the type of an expression.
- **complex types:** `complex`, `creal`, `cimag`, and `conj` — These extensions were used to define complex numbers. Although you cannot write complex number literals, you can have a complex type defined with real and imaginary components. These types need to be managed by the programmer.
- **compound statements within expressions** — This extension was used to declare variables within an expression. You can achieve similar results using in-line functions.
- **iterator types:** `iter`, `sum` — These extensions created loop expressions that were used as a shorthand for working with arrays.
- **assigning variables to specific registers:** `asm` — This extension was used to declare a variable and specify a machine register in which to store it.

If you used any of these extensions in your C source code, you must remove them if you modify or re-compile that source.

Compiler Switch Modifications

Compiler switches listed in [Table A-1](#) have been removed or their action has been modified. If you used any of these switches to compile your C code, you should remove or replace the switch before recompiling the code with the new C compiler.

Table A-1. Obsolete and Replaced Switches

Switch Name	Operation under Release 6.1	Change for VisualDSP++ 2.0
-a	Specify Architecture File.	Replaced with -T and used with a Linker Description File.
-dD, -dM, and -dN	Output the results of preprocessing and/or list of # defines.	Removed.
-deps	Generate dependencies list.	Removed.
-fcond-mismatch	Allow conditional expression mismatch.	Removed.
-finline-functions	Force function inlining.	Removed.
-fkeep-inline-functions	Force keeping of inlined functions.	Removed.
-fno-asm	Don't recognize asm as a keyword.	Replaced with -no-extra-keywords.
-fno-builtin	Don't recognize builtin functions.	Replaced with -no-builtin.
-fsigned-bitfields -funsigned-bitfields	Control whether bit field is signed or unsigned.	Removed. Bitfield is signed or unsigned based on the sign of the type definition declaring the bitfield.

Tools Differences

Table A-1. Obsolete and Replaced Switches (Cont'd)

Switch Name	Operation under Release 6.1	Change for VisualDSP++ 2.0
-fno-signed-bit-fields -fno-unsigned-bitfields	Negative form of the -fsigned-bitfield and -funsigned-bitfield.	Removed. Bitfield is signed or unsigned based on the sign of the type definition declaring the bitfield.
-fsigned-char -funsigned-char	Specify whether to default to signed or unsigned char type.	Replaced with -signed-char and -unsigned-char.
-fsyntax_only	Check syntax only; no output.	Replaced with -syntax-only.
-fwritable-strings	Store string constants in the writable data segment.	Removed. String constants are placed in the seg_data1 section. Definition in the .ldf can place this section in RAM or ROM.
-I-	Modifies directory search order.	Removed.
-imacros	Process macro file.	Removed.
-MD, -MM, and -MMD	Output rules for the make utility; used with -E.	Removed.
-mboot-page=	Specify boot page.	Removed. The ADSP-218x processor does not support paging.
-mdmdata= -mpmdata= -mdcode=	Specify architecture file segments.	Removed. You can control placement of object file segments using the .SECTION directive in the Linker Description File.

Table A-1. Obsolete and Replaced Switches (Cont'd)

Switch Name	Operation under Release 6.1	Change for VisualDSP++ 2.0
-mlstm	Merge C code with assembler-generated code.	Removed.
-mno-doloops	Don't generate loop structures in assembled code.	Removed.
-mno-inits	Don't initialize variables in assembled code.	Removed.
-mpjump	Place the jump table in pm memory.	Replaced with <code>-jump-{dm pm}</code> .
-mreserved=	Instructs the compiler not to use specified registers.	Removed.
-mrom	Make the module a ROM module.	Removed.
-msmall-code	Optimize for size, not for speed.	Removed.
-mstatic-spill	Use dm memory when all register are used.	Removed.
-nostdinc	Do not search standard system directories for header files.	Replaced with <code>-no-std-inc</code> .
-nostdlib	Do not use standard system libraries and startup files when linking.	Replaced with <code>-no-std-lib</code> .
-runhdr	Specify a particular run-time header.	Removed; this feature can now be defined in the <code>.ldf</code> .
-traditional-cpp	Support some preprocessing features.	Removed.

Warnings

The VisualDSP++ 2.0 compiler includes several new warning switches. These switches control the number and type of messages reported during a given compilation. They are described in [Table A-2](#).

Table A-2.C Compiler: New Warning Switches

VisualDSP ++ 2.0 Warning Switch	Description
-warn-protos	Produce a warning when a function is called without a full prototype.
-Wdriver-limit <i>number</i>	Set a maximum <i>number</i> of driver errors.
-Werror-limit <i>number</i>	Set a maximum <i>number</i> of compiler errors.
-Wremarks	Indicates that the compiler may issue remarks, which are diagnostic messages even milder than warnings.
-Wterse	Enable terse warnings.
-pedantic	Causes the compiler to generate warnings for any constructs in a C source file that does not conform to the ANSI standard.
-W{error remark suppress warn} < <i>num</i> > [, <i>num</i> ...]	Overrides the severity of specific compilation diagnostic messages, where <i>num</i> is the number representing the message to override.

The Release 6.1 warning switches that are no longer supported are listed in [Table A-3](#).

Table A-3. Obsolete Warning Switches

-W	-Wformat	-Wtrigraphs
-Wall	-Wimplicit	-Wuninitialized
-Wchar-subscripts	-Wparentheses	-Wunused
-Wcomment	-Wreturn-type	
-Werror	-Wswitch	

See [“Compiler Switch Descriptions” on page 2-7](#) for complete descriptions of C compiler switches.

Run-Time Model

The cc218x Compiler in VisualDSP++ 2.0 produces code that is not fully compatible with the Release 6.1 run-time model. However, the new compiler is superior in many ways to the old one, and your programs will be faster, smaller, and more reliable after the C code is converted to the new system.

A compatibility call is provided to enable you to use existing libraries and special-purpose assembly language subroutines with the new compiler. For more information about this topic, see [“Compatibility Call” on page 2-91](#).

VisualDSP++ 2.0 includes significant changes to the registers, stack usage, and other features of the run-time model; these changes are especially important if you wish to call assembly language subroutines from C programs, or C functions from assembly language programs. For complete details about the VisualDSP++ 2.0 run-time model, see [“C Run-Time Model” on page 2-70](#).

