# iASL: ACPI Source Language Optimizing Compiler and Disassembler

# User Guide

**iASL Overview and Compiler Operation**

**Revision 4.02**

***April 27, 2010***

# Contents

# 1 Introduction

The iASL compiler is a translator for the ACPI Source Language (ASL). As part of the Intel ACPI Component Architecture, the Intel ASL compiler implements translation for the ACPI Source Language (ASL) to the ACPI Machine Language (AML). The disassembler feature will disassemble compiled AML code back to (near-original) ASL source code.

The major features of the iASL compiler include:

- Full support for the ACPI 4.0a Specification including ASL grammar elements and operators.
- Extensive compiler syntax and semantic error checking, especially in the area of control methods. This reduces the number of errors that are not discovered until the AML code is actually interpreted (i.e., the compile-time error checking reduces the number of run-time errors.)
- Multiple types of output files. Besides binary ACPI tables output options include formatted listing files with intermixed source, several types of AML files, and error messages.
- Portable code (ANSI C) and source code availability allows the compiler to be easily ported and run on multiple execution platforms.
- Support for integration with the Microsoft Visual C++ (or similar) development environment.
- Disassembly of all ACPI tables, including tables that contain AML (DSDT, SSDT) as well as ACPI “data” tables such as the FADT, MADT, SRAT, etc.

## 1.1 Document Structure

This document consists of these major sections:

1. ***Introduction***: Contains a brief overview of the iASL compiler/disassembler, document structure, and related reference documents.
2. ***Compiler Overview:*** Compiler inputs, outputs, and supported system environments.
3. ***Compiler Analysis Phases:*** The various source code analysis phases, error and typechecking.
4. ***Compiler Optimizations:*** Optimizations performed by the compiler during AML code generation.
5. ***Compiler Operation Reference:*** Guide for compiler options and general operation.
6. ***ASL-to-AML Disassembler:*** Operation of the disassembler feature.
7. ***Generating iASL from Source Code***: Instructions for building the iASL compiler.

## 1.2 Reference Documents

ACPI documents are available at http://www.acpi.info

- *Advanced Configuration and Power Interface Specification*, Revision 1.0, December 1, 1996
- *Advanced Configuration and Power Interface Specification*, Revision 1.0a, July 1, 1998
- *Advanced Configuration and Power Interface Specification*, Revision 1.0b, February 8, 1999
- *Advanced Configuration and Power Interface Specification*, Revision 2.0, July 27, 2000
- *Advanced Configuration and Power Interface Specification*, Revision 2.0a, March 32, 2002
- *Advanced Configuration and Power Interface Specification*, Revision 2.0b, October 11, 2002
- *Advanced Configuration and Power Interface Specification*, Revision 2.0c, August 23, 2003
- *Advanced Configuration and Power Interface Specification*, Revision 3.0, September 2, 2004
- *Advanced Configuration and Power Interface Specification*, Revision 3.0a, December 30, 2005
- *Advanced Configuration and Power Interface Specification*, Revision 3.0b, October 10, 2006
- *Advanced Configuration and Power Interface Specification*, Revision 4.0, June 16, 2009
- *Advanced Configuration and Power Interface Specification*, Revision 4.0a, April 5, 2010

ACPICA documents are available at http://www.acpica.org/documentation/

- *ACPI Component Architecture User Guide and Programmer Reference*

# 2 Compiler Overview

## 2.1 Input Files

- Existing ACPI ASL source files are fully supported. Enhanced compiler error checking will often uncover unknown problems in these files.
- All ACPI 4.0 ASL additions are supported. The compiler fully supports ACPI 4.0.

## 2.2 Output File Options

- AML binary output file
- AML code in C source code form for inclusion into a BIOS project
- AML code in x86 assembly code form for inclusion into a BIOS project
- AML Hex Table output file in either C, ASL, or x86 assembly code as a table initialization statement.
- Listing file with source file line number, source statements, and intermixed generated AML code. Include files named in the original source ASL file are expanded within the listing file
- Namespace output file — shows the ACPI namespace that corresponds to the input ASL file (and all include files.)
- Debug parse trace output file — gives a trace of the parser and namespace during the compile. Used to debug problems in the compiler, or to help add new compiler features.

## 2.3 Environments Supported

- Runs on multiple platforms as a 32-bit application. Generation and operation as a 64-bit operation has not been tested and is not supported at this time.
- Portable code – requires only ANSI C and a compiler generation package such as Bison/Flex or Yacc/Lex.
- Error and warning messages are compatible with Microsoft Visual C++, allowing for integration of the compiler into the development environment to simplify project building and debug.
- The iASL source code is distributed with the compiler binaries under the ACPICA source license.

# 3 Compiler Analysis Phases

## 3.1 General ASL Syntax Analysis

- Enhanced ASL syntax checking. Multiple errors and warnings are reported in one compile – the compiler recovers to the next ASL statement upon detection of a syntax error.
- Constants larger than the target data size are flagged as errors. For example, if the target data type is a BYTE, the compiler will reject any constants larger than 0xFF (255). The same error checking is performed for WORD and DWORD constants.

## 3.2 General Semantic Analysis

- All named references to objects are checked for validity. All names (both full Namepaths and 4-character Namesegs) must refer to valid declared objects.
- All Fields created within Operation Regions and Buffers are checked for out-of-bounds offset and length. The minimum access width specified for the field is used when performing this check to ensure that the field can be properly accessed.

## 3.3 Control Method Semantic Analysis

- Method local variables are checked for initialization before use. All locals (LOCAL0 – LOCAL7) must be initialized before use. This prevents fatal run-time errors for uninitialized ASL arguments.
- Method arguments are checked for validity. For example, a control method defined with 1 argument can't use ARG4. Again, this prevents fatal run-time errors for uninitialized ASL arguments.
- Control method execution paths are analyzed to determine if all return statements are of the same type — to ensure that either all return statements return a value, or all do not. This includes an analysis to determine if execution can possibly fall through to the default implicit return (which does not return a value) at the end of the method. A warning is issued if some method control paths return a value and others do not

## 3.4 Control Method Invocation Analysis

- All control method invocations (method calls) are checked for the correct number of arguments in all cases, regardless of whether the method is invoked with argument parentheses or not (e.g. both ABCD() and ABCD). Prevents run-time errors caused by non-existent arguments.
- All control methods and invocations are checked to ensure that if a return value is expected and used by the method caller, the target method actually returns a value.

## 3.5 Predefined ACPI Names

For all ACPI reserved control methods (such as _STA, _TMP, etc.), both the number of arguments and return types (whether the method must return a value or not) are checked. This prevents missing operand run-time errors that may not be detected until after the product is shipped.

Predefined names that are defined with arguments or return no value must be implemented as control methods and are flagged if they are not. Predefined names that may be implemented as static objects via the ASL Name() operator are typechecked.

Reserved names (all names that begin with an underscore are reserved) that are not currently defined by ACPI are flagged with a warning.

## 3.6 Resource Descriptors

Validation of values for Resource Descriptors is performed wherever possible.

Address Descriptors: Values for AddressMin, AddressMax, Length, and Granularity are validated:

1. AddressMax must be greater than or equal to AddressMin
2. Length must be less than or equal to (Max-Min+1)
3. If Granularity is non-zero, it must be a power-of-two minus one.

The IsMinFixed and IsMaxFixed parameters are validated against the values given for the AddressMin, AddressMax, Length, and Granularity. This implements the rules given in Table 6-40 of the ACPI 4.0a specification.

# 4 Compiler Optimizations

The compiler implements several optimizations whose primary intent is to reduce the size of the resulting AML output.

## 4.1 Named References

Namepaths within the ASL can often be optimized to shorter strings than specified by the ASL programmer. For example, a full pathname can be optimized to a single 4-character ACPI name if the final name in the path is within the local scope or is along the upward search path to the root from the local scope. In addition, the carat (^) operator can often be used to optimize namepaths.

## 4.2 Integers

Certain integers can be optimized to single-byte AML opcodes. These are: 0, 1, and -1. The opcodes used are Zero, One, and Ones. All other integers are described in AML code using the smallest representation necessary – either Byte, Word, Dword, or Qword.

## 4.3 Constant Folding

All expressions that can be evaluated at compile-time rather than run time are executed and reduced to the simplified value. The ASL operators that are supported in this manner are the Type3, Type4, and Type5 operators defined in the ACPI specification.

The iASL compiler contains the ACPICA AML interpreter which is used to evaluate these expressions at compile time.

# 5 Compiler Operation

The iASL compiler is a command line utility that is invoked to translate one ASL source file to a corresponding AML binary file or the reverse. The syntax of the various command line options is identical across all platforms.

## 5.1 Command Line Invocation

The general command line syntax is as follows:

```
iasl [options] file1, file2, … fileN
```

## 5.2 Wildcard Support

On Windows, wildcard support is implemented within the compiler. For other platforms, it is expected that the shell or command line interpreter will automatically expand wildcards into the **argv** array that is passed to the compiler **main().**

## 5.3 Command Line Options

All compiler options are specified using the '-' (minus) prefix. These options are summarized below, and described in detail after.

```
Global:
  -@<file>        Specify command file
  -I<dir>         Specify additional include directory

General Output:
  -p<prefix>      Specify path/filename prefix for all output files
  -va             Disable all errors and warnings (summary only)
  -vi             Less verbose errors and warnings for use with IDEs
  -vo             Enable optimization comments
  -vr             Disable remarks
  -vs             Disable signon
  -w<1|2|3>       Set warning reporting level

AML Output Files:
  -s<a|c>         Create AML in assembler or C source file (*.asm or *.c)
  -i<a|c>         Create assembler or C include file (*.inc or *.h)
  -t<a|c|s>       Create AML in assembler, C, or ASL hex table (*.hex)

AML Code Generation:
  -oa             Disable all optimizations (compatibility mode)
  -of             Disable constant folding
  -oi             Disable integer optimization to Zero/One/Ones
  -on             Disable named reference string optimization
  -r<Revision>    Override table header Revision (1-255)
```

```
Listings:
  -l            Create mixed listing file (ASL source and AML) (*.lst)
  -ln           Create namespace file (*.nsp)
  -ls           Create combined source file (expanded includes) (*.src)

AML Disassembler:
  -d  [file]    Disassemble or decode binary ACPI table to file (*.dsl)
  -dc [file]    Disassemble AML and immediately compile it
                (Obtain DSDT from current system if no input file)
  -e  [f1,f2]   Include ACPI table(s) for external symbol resolution
  -2            Emit ACPI 2.0 compatible ASL code
  -g            Get ACPI tables and write to files (*.dat)

Help:
  -h            Additional help and compiler debug options
  -hc           Display operators allowed in constant expressions
  -hr           Display ACPI reserved method names
```

### 5.3.1 Global Options

These options affect the compiler globally.

| | |
|---|---|
| -@<file> | Read additional command line options from a command file. The format of this text file is one complete option per line. |
| -I<dir> | Specify an additional directory for include files. The directory that contains the source ASL file is searched first. Then, any additional directories specified via this option are searched. This option may be invoked an unlimited number of times. Directories are searched in the order they appear on the command line. |

### 5.3.2 General Output

These options affect the output of errors and warnings.

| | |
|---|---|
| -p<prefix> | Specify the filename prefix used for all output files, including the .AML file. (This option overrides the output filename specified in the DefinitionBlock of the ASL.) |
| -va | Disable all errors/warnings/remarks. The compiler signon and compilation summary information are the only messages. |
| -vi | Provide less verbose errors and warnings in the format required by the MS VC++ environment. This allows the automatic mapping of errors and warnings to the line of ASL source code that caused the message. |
| -vo | Enable optimization comments in the listing file. A remark/comment is made wherever an optimization has been performed. |
| -vr | Disable all remark messages. |
| -vs | Disable the compiler signon. |
| -w<1\|2\|3> | Set the warning reporting level. |

### 5.3.3 AML Text Output Files

The compiler always emits a binary AML table. These options allow the compiler to create various text versions of the AML code to simplify the inclusion of the code into a BIOS project.

#### 5.3.3.1 Source Code Files (-s)

These options create files that contain the AML in hex format, with a unique label for each line of the original ASL code. This allows the BIOS to easily dynamically access/modify the ACPI table.

| | |
|---|---|
| -sa | Create AML in an x86 assembly source code file with the extension .ASM. This option creates a file with a unique label on the AML code for each line of ASL code. |
| -sc | Create AML in a C source code file with the extension .C. This option creates a file with a unique label on the AML code for each line of ASL code. |

#### 5.3.3.2 Source External Declaration Files (-i)

These options create files that contain external declarations for the symbols created by the options in the previous section.

| | |
|---|---|
| -ia | Create an ASM include file (.INC) that contains external declarations for the symbols produced by the –sa option above. |
| -ic | Create a C header file (.H) that contains external declarations for the symbols produced by the –sc option above. |

#### 5.3.3.3 Hex Source Code Files (-t)

These options create files that contain the AML code in hex format, in a single array.

| | |
|---|---|
| -ta | Create a hex table file with the extension .HEX. This file contains raw AML byte data in hex table format suitable for inclusion into an ASM file. |
| -tc | Create a hex table file with the extension .HEX. This file contains raw AML byte data in hex table format suitable for inclusion into a C file. |
| -ts | Create a hex table file with the extension .HEX. This file contains raw AML byte data in an ASL Buffer object format suitable for inclusion into a ASL file. |

### 5.3.4 AML Bytecode Generation

These options affect the actual AML code that is generated by the compiler.

| | |
|---|---|
| -oa | Disable all optimizations. |
| -of | Disable the constant folding feature. |
| -oi | Disable integer optimizations to the Zero/One/Ones AML opcodes. |
| -on | Disable named reference string optimizations. |
| -r<Rev> | Set the revision number of the table header, overriding the existing revision. |

### 5.3.5 Listings

These options control the listings that are produced by the compiler

| | |
|---|---|
| -l | Create a listing file with the extension .LST. This file contains intermixed ASL source code and AML byte code so that the AML corresponding to each ASL statement can be examined. |
| -ln | Create a namespace file with a dump of the ACPI namespace and the extension .NSP |
| -ls | Create a combined source file with the extension .SRC. This file combines all include files into a single, large source file. |

### 5.3.6 AML Disassembler

These options are used to invoke and control the behavior of the AML disassembler.

| | |
|---|---|
| -d [file] | Disassemble or decode a binary ACPI to a file (.DSL). Tables that contain AML code are disassembled back to ASL code. Tables that do not contain AML code are decoded and displayed with a description of each field within the table. |
| -dc [file] | Disassemble a binary AML file and immediately compile it. |
| -e [f1,f2] | Include these extra binary AML tables to assist with external symbol resolution. This option is very useful when attempting to disassemble a table that contains cross-table control method invocations. In these cases, it is difficult or impossible to properly disassemble the method invocation without having the definition of the method present (the important missing data is the number of arguments.) |
| -2 | Emit ACPI 2.0 compatible ASL code. |
| -g | Get ACPI tables from the local machine (availability depends on the host implementation.) |

### 5.3.7 Help

| | |
|---|---|
| -h | Additional help |
| -hc | Display a complete list of all ASL operators that are allowed in constant expressions that can be evaluated at compile time. (This is a list of the Type 3, 4, and 5 operators.) |
| -hr | Display a list of the ACPI predefined names (reserved names.) |

## 5.4 Examples

### 5.4.1 Input ASL

Example input ASL:

```
DefinitionBlock ("", "DSDT", 2, "Intel", "EXAMPLE", 1)
{
    Name (BSTP, Package() {0,1,2,3})

    Method (_BST)
    {
        Store (BSTP, Debug)
        Return (BSTP)
    }
}
```

### 5.4.2 Output of –tc (make C hex table) Option

This is the output of the –tc option. The entire table is emitted in a single C array.

```
/*
 *
 * Intel ACPI Component Architecture
 * ASL Optimizing Compiler version 20100331 [Mar 31 2010]
 * Copyright (c) 2000 - 2010 Intel Corporation
 * Supports ACPI Specification Revision 4.0
 *
 * Compilation of "dsdt.asl" - Tue Apr 27 14:20:41 2010
 *
 * C source code output
 * AML code block contains 0x45 bytes
 *
 */
unsigned char AmlCode[] =
{
    0x44,0x53,0x44,0x54,0x45,0x00,0x00,0x00,  /* 00000000    "DSDTE..." */
    0x02,0xED,0x49,0x6E,0x74,0x65,0x6C,0x00,  /* 00000008    "..Intel." */
    0x45,0x58,0x41,0x4D,0x50,0x4C,0x45,0x00,  /* 00000010    "EXAMPLE." */
    0x01,0x00,0x00,0x00,0x49,0x4E,0x54,0x4C,  /* 00000018    "....INTL" */
    0x31,0x03,0x10,0x20,0x08,0x42,0x53,0x54,  /* 00000020    "1.. .BST" */
    0x50,0x12,0x08,0x04,0x00,0x01,0x0A,0x02,  /* 00000028    "P......." */
    0x0A,0x03,0x14,0x12,0x5F,0x42,0x53,0x54,  /* 00000030    "...._BST" */
    0x00,0x70,0x42,0x53,0x54,0x50,0x5B,0x31,  /* 00000038    ".pBSTP[1" */
    0xA4,0x42,0x53,0x54,0x50                  /* 00000040    ".BSTP"    */
};
```

### 5.4.3 Output of –sc (make C source) Option

This is the output of the –sc option. The table is emitted in multiple C arrays, approximatly one array per "block" of ASL code. For example, one array is emitted per control method.

```
/*
 *
 * Intel ACPI Component Architecture
 * ASL Optimizing Compiler version 20090730 [Aug 14 2009]
 * Copyright (C) 2000 - 2009 Intel Corporation
 * Supports ACPI Specification Revision 4.0
 *
 * Compilation of "dsdt.asl" - Fri Aug 14 14:59:46 2009
 *
 */
    /*
     *       1....
     *       2....DefinitionBlock ("", "DSDT", 2, "Intel", "EXAMPLE", 1)
     */
    unsigned char    DSDT_EXAMPLE_Header [] =
    {
        0x44,0x53,0x44,0x54,0x45,0x00,0x00,0x00, /* 00000000    "DSDTE..." */
        0x02,0xF1,0x49,0x6E,0x74,0x65,0x6C,0x00, /* 00000008    "..Intel." */
        0x45,0x58,0x41,0x4D,0x50,0x4C,0x45,0x00, /* 00000010    "EXAMPLE." */
        0x01,0x00,0x00,0x00,0x49,0x4E,0x54,0x4C, /* 00000018    "....INTL" */
        0x30,0x07,0x09,0x20,                     /* 0000001C    "0.. " */
    };

    /*
     *       3....{
     *       4....    Name (BSTP, Package() {0,1,2,3})
     */
    unsigned char    DSDT_EXAMPLE_BSTP [] =
    {
        0x08,0x42,0x53,0x54,0x50,                /* 00000021    ".BSTP" */
        0x12,0x08,0x04,0x00,0x01,0x0A,0x02,0x0A, /* 00000029    "........" */
        0x03,                                    /* 0000002A    "." */
    };

    /*
     *       5....
     *       6....    Method (_BST)
     */
    unsigned char    DSDT_EXAMPLE__BST [] =
    {
        0x14,0x12,0x5F,0x42,0x53,0x54,0x00,      /* 00000031    ".._BST." */

    /*
     *       7....    {
     *       8....        Store (BSTP, Debug)
     */
        0x70,0x42,0x53,0x54,0x50,0x5B,0x31,      /* 00000038    "pBSTP[1" */
    /*
     *       9....        Return (BSTP)
     */
        0xA4,0x42,0x53,0x54,0x50,                /* 0000003D    ".BSTP" */
    /*
     *      10....    }
     *      11....}
     *      12....
     */
    };
```

### 5.4.4 Output of –ic (make include file) Option

This is the output of the –ic option. It creates external declarations for all of the arrays created by the –sc option above.

```
/*
 *
 * Intel ACPI Component Architecture
 * ASL Optimizing Compiler version 20090730 [Aug 14 2009]
 * Copyright (C) 2000 - 2009 Intel Corporation
 * Supports ACPI Specification Revision 4.0
 *
 * Compilation of "dsdt.asl" - Fri Aug 14 15:05:34 2009
 *
 */
extern unsigned char    DSDT_EXAMPLE_Header [];
extern unsigned char    DSDT_EXAMPLE_BSTP [];
extern unsigned char    DSDT_EXAMPLE__BST [];
```

### 5.4.5 Output of –l (Listing) Option

This is a standard listing file with intermixed ASL and AML code.

```
Intel ACPI Component Architecture
ASL Optimizing Compiler version 20090730 [Aug 14 2009]
Copyright (C) 2000 - 2009 Intel Corporation
Supports ACPI Specification Revision 4.0

Compilation of "dsdt.asl" - Fri Aug 14 15:08:30 2009

       1....
       2....DefinitionBlock ("", "DSDT", 2, "Intel", "EXAMPLE", 1)

00000000....44 53 44 54 45 00 00 00     "DSDTE..."
00000008....02 F1 49 6E 74 65 6C 00     "..Intel."
00000010....45 58 41 4D 50 4C 45 00     "EXAMPLE."
00000018....01 00 00 00 49 4E 54 4C     "....INTL"
00000020....30 07 09 20 ............    "0.. "
```

```
       3....{
       4....    Name (BSTP, Package() {0,1,2,3})

[****iasl****]
dsdt.asl     4:     Name (BSTP, Package() {0,1,2,3})
Optimize 6033 -                          ^ Integer optimized to single-byte AML
opcode (Zero)

 [****iasl****]
dsdt.asl     4:     Name (BSTP, Package() {0,1,2,3})
Optimize 6033 -                            ^ Integer optimized to single-byte AML
opcode (One)

00000024....08 42 53 54 50 .........    ".BSTP"
00000029....12 08 04 00 01 0A 02 0A     "........"
00000031....03 ....................     "."

       5....
       6....    Method (_BST)

00000032....14 12 5F 42 53 54 00 ...    ".._BST."

       7....    {
       8....        Store (BSTP, Debug)

00000039....70 42 53 54 50 5B 31 ...    "pBSTP[1"

       9....        Return (BSTP)

00000040....A4 42 53 54 50 .........    ".BSTP"
      10....    }
      11....}
      12....

Summary of errors and warnings

ASL Optimizing Compiler version 20090730 [Aug 14 2009]
ASL Input:  dsdt.asl - 13 lines, 178 bytes, 4 keywords
AML Output: dsdt.aml - 69 bytes, 2 named objects, 2 executable opcodes

Compilation complete. 0 Errors, 0 Warnings, 0 Remarks, 2 Optimizations
```

## 5.4.6 Output of –ln (Namespace Listing) Option

This is a namespace listing file.

```
Intel ACPI Component Architecture
ASL Optimizing Compiler version 20090730 [Aug 14 2009]
Copyright (C) 2000 - 2009 Intel Corporation
Supports ACPI Specification Revision 4.0

Compilation of "dsdt.asl" - Fri Aug 14 15:08:30 2009

Contents of ACPI Namespace

Count  Depth    Name - Type

    1  [1]      _GPE - Scope
    2  [1]      _PR_ - Scope
    3  [1]      _SB_ - Device
    4  [1]      _SI_ - Scope
    5  [1]      _TZ_ - Thermal
    6  [1]      _REV - Integer
    7  [1]      _OS_ - String
    8  [1]      _GL_ - Mutex
    9  [1]      _OSI - Method
   10  [1]      BSTP - Package       [Initial Length  0x04 elements]
   11  [1]      _BST - Method        [Code Length     0x0011 bytes]

Namespace pathnames

\_GPE
\_PR_
\_SB_
\_SI_
\_TZ_
\_REV
\_OS_
\_GL_
\_OSI
\BSTP
\_BST
```

## 5.5 Integration Into MS VC++ Environment

This section contains instructions for integrating the iASL compiler into MS VC++ 6.0 development environment.

### 5.5.1 Integration as a Custom Tool

This procedure adds the iASL compiler as a custom tool that can be used to compile ASL source files. The output is sent to the VC output window.

a) Select Tools->Customize.

b) Select the "Tools" tab.

c) Scroll down to the bottom of the "Menu Contents" window. There you will see an empty rectangle. Click in the rectangle to enter a name for this tool.

d) Type "iASL Compiler" in the box and hit enter. You can now edit the other fields for this new custom tool.

e) Enter the following into the fields:

| | |
|---|---|
| Command: | `C:\Acpi\iasl.exe` |
| Arguments: | `-e "$(FilePath)"` |
| Initial Directory: | `"$(FileDir)"` |
| Use Output Window: | <Check this option> |

"Command" must be the path to wherever you copied the compiler.

"-e" instructs the compiler to produce messages appropriate for VC.

Quotes around FilePath and FileDir enable spaces in filenames.

f) Select "Close".

These steps will add the compiler to the tools menu as a custom tool. By enabling "Use Output Window", you can click on error messages in the output window and the source file and source line will be automatically displayed by VC. Also, you can use F4 to step through the messages and the corresponding source line(s).

### 5.5.2 Integration into a Project Build

The compiler can be integrated into a project build by using it in the “custom build” step of the project generation. The commands and arguments should be similar to those described above.

# 6 ASL-to-AML Disassembler

## 6.1 AML ACPI Tables (DSDT, SSDT, etc.)

Because the AML contains all of the orginal symbols from the ASL, the AML byte code can be disassembled back to nearly the original ASL code with only a few caveats. There is a known difficulty in disassembling control method invocations for methods that are external to the table being disassembled. This is because there is often insufficient information within the AML to properly disassemble these method invocations. Wherever possible, all DSDTs and SSDTs for a given machine should be disassembled together using the –e option.

## 6.2 ACPI Data Tables (FADT, MADT, SRAT, etc.)

These non-AML ACPI tables can be "disassembled", meaning that they are formatted with the individual fields and data.

Example disassembly of an FADT.

```
/*
 * Intel ACPI Component Architecture
 * AML Disassembler version 20090730
 *
 * Disassembly of FACP.dat, Wed Aug 12 14:18:28 2009
 *
 * ACPI Data Table [FACP]
 *
 * Format: [HexOffset DecimalOffset ByteLength]  FieldName : FieldValue
 */
[000h 0000  4]                    Signature : "FACP
[004h 0004  4]                 Table Length : 00000084
[008h 0008  1]                     Revision : 02
[009h 0009  1]                     Checksum : E3
[00Ah 0010  6]                       Oem ID : "INTEL "
[010h 0016  8]                 Oem Table ID : "0944    "
[018h 0024  4]                 Oem Revision : 00000002
[01Ch 0028  4]              Asl Compiler ID : "INTL"
[020h 0032  4]        Asl Compiler Revision : 00000001

[024h 0036  4]                 FACS Address : 37FF7E80
[028h 0040  4]                 DSDT Address : 37FE58DC
[02Ch 0044  1]                        Model : 00
[02Dh 0045  1]                   PM Profile : 00 (Unspecified)
[02Eh 0046  2]                SCI Interrupt : 0009
[030h 0048  4]             SMI Command Port : 000000B0
[034h 0052  1]            ACPI Enable Value : F1
[035h 0053  1]           ACPI Disable Value : F0
[036h 0054  1]               S4BIOS Command : F2
[037h 0055  1]              P-State Control : 00
[038h 0056  4]     PM1A Event Block Address : 00008000
[03Ch 0060  4]     PM1B Event Block Address : 00008104
[040h 0064  4]   PM1A Control Block Address : 00008004
```

```
[044h 0068  4]   PM1B Control Block Address : 00000000
[048h 0072  4]    PM2 Control Block Address : 00008100
[04Ch 0076  4]       PM Timer Block Address : 00008008
[050h 0080  4]           GPE0 Block Address : 00008020
[054h 0084  4]           GPE1 Block Address : 00000000
[058h 0088  1]       PM1 Event Block Length : 04
[059h 0089  1]     PM1 Control Block Length : 02
[05Ah 0090  1]     PM2 Control Block Length : 01
[05Bh 0091  1]        PM Timer Block Length : 04
[05Ch 0092  1]            GPE0 Block Length : 08
[05Dh 0093  1]            GPE1 Block Length : 00
[05Eh 0094  1]             GPE1 Base Offset : 00
[05Fh 0095  1]                 _CST Support : 00
[060h 0096  2]                   C2 Latency : 03E7
[062h 0098  2]                   C3 Latency : 03E9
[064h 0100  2]               CPU Cache Size : 0000
[066h 0102  2]           Cache Flush Stride : 0000
[068h 0104  1]            Duty Cycle Offset : 01
[069h 0105  1]             Duty Cycle Width : 03
[06Ah 0106  1]          RTC Day Alarm Index : 0D
[06Bh 0107  1]        RTC Month Alarm Index : 00
[06Ch 0108  1]            RTC Century Index : 32
[06Dh 0109  2]   Boot Flags (decoded below) : 0003
              Legacy Devices Supported (V2) : 1
           8042 Present on ports 60/64 (V2) : 1
                       VGA Not Present (V4) : 0
                     MSI Not Supported (V4) : 0
               PCIe ASPM Not Supported (V4) : 0
[06Fh 0111  1]                     Reserved : 00
[070h 0112  4]        Flags (decoded below) : 000001A5
     WBINVD instruction is operational (V1) : 1
             WBINVD flushes all caches (V1) : 1
                   All CPUs support C1 (V1) : 1
                 C2 works on MP system (V1) : 0
           Control Method Power Button (V1) : 0
           Control Method Sleep Button (V1) : 1
       RTC wake not in fixed reg space (V1) : 0
           RTC can wake system from S4 (V1) : 1
                       32-bit PM Timer (V1) : 1
                     Docking Supported (V1) : 0
              Reset Register Supported (V2) : 0
                           Sealed Case (V3) : 0
                   Headless - No Video (V3) : 0
       Use native instr after SLP_TYPx (V3) : 0
             PCIEXP_WAK Bits Supported (V4) : 0
                    Use Platform Timer (V4) : 0
              RTC_STS valid on S4 wake (V4) : 0
               Remote Power-on capable (V4) : 0
                Use APIC Cluster Model (V4) : 0
    Use APIC Physical Destination Mode (V4) : 0

[074h 0116 12]               Reset Register : <Generic Address Structure>
[074h 0116  1]                     Space ID : 01 (SystemIO)
[075h 0117  1]                    Bit Width : 08
[076h 0118  1]                   Bit Offset : 00
[077h 0119  1]                 Access Width : 00
[078h 0120  8]                      Address : 0000000000000000
```

```
[080h 0128  1]         Value to cause reset : 00
[081h 0129  3]                     Reserved : 000000
```

# 7 Generating iASL from Source Code

Generation of the ASL compiler from source code requires these items:

## 7.1 Required Tools

1) The *flex* (or *Lex*) lexical analyzer generator
2) The *Bison (Yacc* replacement) parser generator
3) An ANSI C compiler

## 7.2 Required Source Code

There are three major source code components that are required to generate the compiler

1. The ASL compiler source
2. The ACPICA Core Subsystem source. In particular, the Namespace Manager component is used to create an internal ACPI namespace and symbol table.), and the AML Interpreter is used to evaluate constant expressions.
3. The Common source for all ACPI components

The source files appear in these directories by default:

| | |
|---|---|
| Compiler Source: | **Acpica/Source/Compiler** |
| Common Source: | **Acpica/Source//Common** |
| Subsystem Source: | **Acpica/Source/Components/** |

This page intentionally left blank.