iASL Compiler User Reference

Revision 1.01

May 3, 2002

<Classification>

Contents

1 Introduction 4

2 Compiler Overview 4

2.1 Input Files 4
2.2 Output File Options 4
2.3 Environments Supported 5

3 Compiler Analysis Phases 5

3.1 General ASL Syntax Analysis 5
3.2 General Semantic Analysis 5
3.3 Control Method Semantic Analysis 5
3.4 Control Method Invocation Analysis 6
3.5 Constant Folding 6

4 Compiler Operation 6

4.1 Command Line Invocation 6
4.2 Options 6
4.3 Integration Into MS VC++ Environment 7
4.3.1 Integration as a Custom Tool 7
4.3.2 Integration into a Project Build 8

5 Compiler Generation From Source 8

5.1 Required Tools 8
5.2 Required Source Code 8

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).

Major features of the iASL compiler include:

- Full support for the ACPI 2.0a 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, including 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++ development environment.

2 Compiler Overview

2.1 Input Files

- Existing ACPI 1.0 ASL source files are fully supported. Enhanced compiler error checking will often uncover unknown problems in these files.
- All ACPI 2.0 ASL additions are supported. The compiler fully supports ACPI 2.0a and all errata documents as of the time of this writing,

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 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 (Currently Windows, FreeBSD, and Linux).
- 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.
- Source code is distributed with the compiler binaries

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 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.

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

4 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. The syntax of the various command line options is the same across all platforms.

4.1 Command Line Invocation

The general command line syntax is as follows:

```
iasl [options] <Input Filename>
```

4.2 Options

The compiler options are specified using the '-' (minus) prefix. These options include:

-a 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.

-c 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.

-e 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.

-f Disable the constant folding feature.

-h Additional help

-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.

-n Create a namespace file with a dump of the ACPI namespace and the extension .NSP

-o 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.)

-qc 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.)

-s Create a combined source file with the extension .SRC. This file combines all include files into a single, large source file.

-t 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 (or optionally, ASM) file.

4.3 Integration Into MS VC++ Environment

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

4.3.1 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).

4.3.2 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.

5 Compiler Generation From Source

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

5.1 Required Tools

1) The *flex* (or *Lex*) lexical analyzer generator

2) The *Bison (Yacc* replacement) parser generator

3) An ANSI C compiler

5.2 Required Source Code

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

1. The ASL compiler source
2. The ACPI CA 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:	**Acpi/Components/AslCompiler**
Common Source:	**Acpi/Components/Common**
Subsystem Source:	**Acpi/Components/Subsystem**

This page intentionally left blank.