Multi-String .UNI File Format Specification

February 2014

Revision 1.0

Contents

Tables

Revision History

Revision Number	Description	Revision Date
1.0	Initial Release.	February 2014

Introduction

§

1 Introduction

1.1 Overview

This document describes file format for Unicode string files. This file format supports multiple layouts and formats in the Unicode file. This versatility allows strings to be grouped either by language or by string identifier.

1.2 Related Information

The following publications and sources of information may be useful to you or are referred to by this specification:

- *Extensible Firmware Interface Specification*, Version 1.10, Intel, 2001, http://developer.intel.com/technology/efi.
- *Unified Extensible Firmware Interface Specification*, Version 2.0, Unified EFI, Inc., 2006, http://www.uefi.org.
- *Unified Extensible Firmware Interface Specification*, Version 2.1, Unified EFI, Inc., 2007, http://www.uefi.org.
- *Intel® Platform Innovation Framework for EFI Specifications*, Intel, 2006, http://www.intel.com/technology/framework/.
- *EDK II Module Development Environment Package Specification*, Version 0.51. Intel, 2006, https://edk2.tianocore.org/source/browse/edk2/trunk/docs/.
- *EDK II Build and Packaging Architecture Specification*, Version 0.53, Intel, 2006, https://edk2.tianocore.org/source/browse/edk2/trunk/docs/.
- *EDK II Module Surface Area Specification*, Version 0.51, Intel, 2006, https://edk2.tianocore.org/source/browse/edk2/trunk/docs/.
- *EDK II Module Development Environment Library Specification*, Version 0.58, Intel, 2006, https://edk2.tianocore.org/servlets/ProjectDocumentList?folderID=82&expandFolder=82&folderID=0.
- *EDK II Platform Configuration Database Infrastructure Description*, Version 0.54, Intel, 2006, https://edk2.tianocore.org/source/browse/edk2/trunk/docs/.
- *EDK II C Coding Standards Specification*, Version 0.51, Intel, 2006, https://edk2.tianocore.org/source/browse/edk2/trunk/docs/.

1.3 Terms

The following terms are used throughout this document to describe varying aspects of input localization:

BDS

Framework Boot Device Selection phase.

BNF

BNF is an acronym for “Backus Naur Form.” John Backus and Peter Naur introduced for the first time a formal notation to describe the syntax of a given language.

Component

An executable image. Components defined in this specification support one of the defined module types.

DXE

Framework Driver Execution Environment phase.

DXE SAL

A special class of DXE module that produces SAL Runtime Services. DXE SAL modules differ from DXE Runtime modules in that the DXE Runtime modules support Virtual mode OS calls at OS runtime and DXE SAL modules support intermixing Virtual or Physical mode OS calls.

DXE SMM

A special class of DXE module that is loaded into the System Management Mode memory.

DXE Runtime

Special class of DXE module that provides Runtime Services

EFI

Generic term that refers to one of the versions of the EFI specification: EFI 1.02, EFI 1.10, or UEFI 2.0.

EFI 1.10 Specification

Intel Corporation published the Extensible Firmware Interface Specification. Intel donated the EFI specification to the Unified EFI Forum, and the UEFI now owns future updates of the EFI specification. See UEFI Specifications.

Foundation

The set of code and interfaces that glue implementations of EFI together.

Framework

Intel® Platform Innovation Framework for EFI consists of the Foundation, plus other modular components that characterize the portability surface for modular components designed to work on any implementation of the Tiano architecture.

GUID

Globally Unique Identifier. A 128-bit value used to name entities uniquely. An individual without the help of a centralized authority can generate a unique GUID. This allows the generation of names that will never conflict, even among multiple, unrelated parties.

HII

Human Interface Infrastructure. This generally refers to the database that contains string, font, and IFR information along with other pieces that use one of the database components.

IFR

Internal Forms Representation. This is the binary encoding that is used for the representation of user interface pages.

Library Class

A library class defines the API or interface set for a library. The consumer of the library is coded to the library class definition. Library classes are defined via a library class .h file that is published by a package. See the *EDK 2.0 Module Development Environment Library Specification* for a list of libraries defined in this package.

Library Instance

An implementation of one or more library classes. See the *EDK 2.0 Module Development Environment Library Specification* for a list of library defined in this package.

Module

A module is either an executable image or a library instance. For a list of module types supported by this package, see module type.

Module Type

All libraries and components belong to one of the following module types: BASE, SEC, PEI_CORE, PEIM, DXE_CORE, DXE_DRIVER, DXE_RUNTIME_DRIVER, DXE_SMM_DRIVER, DXE_SAL_DRIVER, UEFI_DRIVER, or UEFI_APPLICATION. These definitions provide a framework that is consistent with a similar set of requirements. A module that is of module type BASE, depends only on headers and libraries provided in the MDE, while a module that is of module type DXE_DRIVER depends on common DXE components. For a definition of the various module types, see module type.

Module Surface Area (MSA)

The MSA is an XML description of how the module is coded. The MSA contains information about the different construction options for the module. After the module is constructed the MSA can describe the interoperability requirements of a module.

Package

A package is a container. It can hold a collection of files for any given set of modules. Packages may be described as one of the following types of modules:

• Source modules, containing all source files and descriptions of a module

• Binary modules, containing EFI Sections or a Framework File System and a description file specific to linking and binary editing of features and attributes specified in a Platform Configuration Database (PCD,)

• Mixed modules, with both binary and source modules

Multiple modules can be combined into a package, and multiple packages can be combined into a single package.

Protocol

An API named by a GUID as defined by the EFI specification.

PCD

Platform Configuration Database.

PEI

Pre-EFI Initialization Phase.

PPI

A PEIM-to-PEIM Interface that is named by a GUID as defined by the PEI CIS.

SAL

System Abstraction Layer. A firmware interface specification used on Intel® Itanium® Processor based systems.

Runtime Services

Interfaces that provide access to underlying platform-specific hardware that might be useful during OS runtime, such as time and date services. These services become active during the boot process but also persist after the OS loader terminates boot services.

SEC

Security Phase is the code in the Framework that contains the processor reset vector and launches PEI. This phase is separate from PEI because some security schemes require ownership of the reset vector.

UEFI Application

An application that follows the UEFI specification. The only difference between a UEFI application and a UEFI driver is that an application is unloaded from memory when it exits regardless of return status, while a driver that returns a successful return status is not unloaded when its entry point exits.

UEFI Driver

A driver that follows the UEFI specification.

UEFI Specification Version 2.0

Current version of the EFI specification released by the Unified EFI Forum. This specification builds on the EFI 1.10 specification and transfers ownership of the EFI specification from Intel to a non-profit, industry trade organization.

Unified EFI Forum

A non-profit collaborative trade organization formed to promote and manage the UEFI standard. For more information, see www.uefi.org.

1.4 Additional subheadings (optional)

Put additional introductory sections here as needed.

1.5 Conventions used in this document

This document uses the typographic and illustrative conventions described below.

1.5.1 Pseudo-code conventions

Pseudo code is presented to describe algorithms in a more concise form. None of the algorithms in this document are intended to be compiled directly. The code is presented at a level corresponding to the surrounding text.

In describing variables, a list is an unordered collection of homogeneous objects. A queue is an ordered list of homogeneous objects. Unless otherwise noted, the ordering is assumed to be First In First Out (FIFO).

Pseudo code is presented in a C-like format, using C conventions where appropriate. The coding style, particularly the indentation style, is used for readability and does not necessarily comply with an implementation of the Extensible Firmware Interface Specification.

1.5.2 Typographic conventions

This document uses the typographic and illustrative conventions described below:

Plain text (Body)	The normal text typeface is used for the vast majority of the descriptive text in a specification.
Plain text [blue] (Cross-Reference)	Any plain text that is underlined and in blue indicates an active link to the cross-reference. Click on the word to follow the hyperlink. USE ONLY IF YOU MAKE AN ACTUAL CROSS-REFERENCE LINK.

Bold **(Bold or GlossTerm)**	In text, a Bold typeface identifies a processor register name. In other instances, a Bold typeface can be used as a running head within a paragraph. or as a definition heading (GlossTerm)
Italic	In text, an Italic typeface can be used as emphasis to introduce a new term or to indicate a manual or specification name.

`BOLD Monospace`
 	
`(CodeCharacter and CodePargraph)`
	Computer code, example code segments, and all prototype code segments use a **`BOLD Monospace`** typeface with a dark red color. These code listings normally appear in one or more separate paragraphs, though words or segments can also be embedded in a normal text paragraph.

`Bold Monospace`	Words in a **`Bold Monospace`** typeface that is underlined and in blue indicate an active hyperlink to the code definition for that function or type definition. Click on the word to follow the hyperlink. USE ONLY IF YOU MAKE AN ACTUAL HYPERLINK.

`Italic Monospace`
 	
`(ArgCharacter, ArgCharacter+ and ArgParagraph)`
	In code or in text, words in Italic Monospace indicate placeholder names for variable information that must be supplied (i.e., arguments).
	
	Arguments or Parameters in a CodePargraph should use *`ArgCharacter+`* (to prevent Parameters being in bold). ArgCharacter is fine for

`Plain Monospace`
 	
`(CodeCharacter + Not Bold)`
	In code, words in a `Plain Monospace` typeface that is a dark red color but is not bold or italicized indicate pseudo code or example code. These code segments typically occur in one or more separate paragraphs.

text text text	In the PDF of this specification, text that is highlighted in yellow indicates that a change was made to that text since the previous revision of the PDF. The highlighting indicates only that a change was made since the previous version; it does not specify what changed. If text was deleted and thus cannot be highlighted, a note in red and highlighted in yellow (that looks like *(Note: text text text.)*) appears where the deletion occurred.

2 Unicode Strings File Format

This file format is a tool-agnostic method for storing localizable strings that are indexed by tokens.

Strings ends are determined by the first of the following items found:

- double quotes
- a control character in the first column of a new line
- a comment
- the end of the file
- a blank line

Comments may appear anywhere within the string file.

All the files must begin with the binary character, `0xFEFF` (big-endian).

Note: *These file may be created either by using a text editor program that will do conversions or by creating an ASCII text file and processing the file to create the UCS-2LE encoded file.*

All strings must be null terminated upon extraction from the file.

2.1 Extended Backus–Naur Form (EBNF)

The following EBNF shows quoted (double quotes) ASCII characters preceded by an upper case "L" character. This format is used in the document following the C coding method, however, the HII UCS-2LE encoded string file actually uses char16 data characters. In the actual UCS-2LE encoded file, the "L" character is not present.

```
<StringFileFormat>  ::=  <CommentLine>*
                           <LanguageDefs>
                           <Content>+

<US>                ::=  L" "

<MS>                ::=  <US>+

<ME>                ::=  {<MS>} {<EOL>}

<LanguageDefs>      ::=  <CtrlChar> L"langdef" <MS> <LangCode> <MS>
                           <LangDesc> <EOL>

<LangDesc>          ::=  L" <DoubleQuote> <Chars> <DoubleQuote> "

<Content>           ::=  {<CommentLine>}  {<BlankLine>}
                           {<UnicodeLines>} {<ControlRefactor>}
                           {<LanguageDefs>} {<SecurityLines>}
                           {<IncludeLines>}

<CommentLine>       ::=  L"//" <US>* <PCHars> <EOL>

<BlankLine>         ::=  <EOL>

<Chars>             ::=  (0x0001-0xF6FF)

<PChars>            ::=  (0x0020-0xF6FF)

<VChars>            ::=  (0x0021-0xF6FF)

<DefaultCtrlChar>   ::=  L"/"

<ControlRefactor>   ::=  <CtrlChar> L"=" <NewCtrlChar> <EOL>

<NewCtrlChar>       ::=  (0x0021 - 0xF6FF)

<UnicodeLines>      ::=  <Token> <ME>
                           [<CtrlChar> L"language" <MS> <LangCode>
                           <ME> <String> <ME>]+

<SecurityLines>     ::=  L"security" L"(" <CFormatGUID> L")" <EOL>

<CFormatGUID>       ::=  L"{ 0x" <HexDigitU>{8} L", 0x" <HexDigitU>{4}
                           L", 0x" <HexDigitU>{4} L", { 0x" <HexDigitU>{2}
                           L", 0x" <HexDigitU>{2} L", 0x" <HexDigitU>{2}
                           L", 0x" <HexDigitU>{2} L", 0x" <HexDigitU>{2}
                           L", 0x" <HexDigitU>{2} L", 0x" <HexDigitU>{2}
                           L", 0x" <HexDigitU>{2} L" }}"

<HexDigitU>         ::=  (\ua-\uf\uA-\uF\u0-\u9)

<Token>             ::=  <CtrlChar> <MS> L"string" <MS>
                           (\uA-\uZ\ua\uz)(\uA-\uZ\ua\uz\u0-\u9\u_)*

<LangCode>          ::=  <RFC4646>


<RFC4646>           ::=  (\ua-\uz\uA-\uZ){2,8}
                           [L"-" (\ua-\uz\uA-\uZ\u0-\u9){2,8}]

<UDblQuote>         ::=  0x0022

<String>            ::=  <UDblQuote> <SContent>* <UDblQuote>

<SContent>          ::=  {<PChars>} {<Attributes>} {<CtrlCode>}

<Attributes>        ::=  <StartAttribute> <SContent>* [<StopAttribute>]
```

```
<StartAttribute>    ::=  <AttrCtrlChar> <FontAttr>

<AttrCtrlChar>      ::=  L"\"

<StopAttribute>     ::=  <AttrCtrlChar> L"end" <FontAttr>

<FontAttr>          ::=  {<SimpleAttrs>} {<StandardAttrs>}

<SimpleAttrs>       ::=  {L"narrow">} {<L"wide">}

<StandardAttrs>     ::=  {L"normal"} {L"bold">} {<L"italic">}
                         {<L"emboss">}
                         {L"shadow"} {L"underline"} {L"dblunder"}

<CtrlCode>          ::=  <EscChar> {L"n"} {L"f"} {L"r"} {L"p"}
                         {L"ospace"} {L"enquad"} {L"emquad"}
                         {L"ensp"} {L"emsp"} {L"em3sp"} {L"em4sp"}
                         {L"em6sp"} {L"usp"} {L"tsp"} {L"hsp"}
                         {L"msp"} {L"!bsp"} {L"!nbsp"}
                         {L"zsp"} {L"ah"} {L"hy"} {L"df"} {L"den"}
                         {L"dem"} {L"!bh"} {L"g"} {L"osp"} {L"k"}

<EscChar>           ::=  L"\"

<IncludeLines>      ::=  <CtrlChar> L”include” <UniFile> <EOL>

<UniFile>           ::=  <UDblQuote> <UniFilename> <UDblQuote>

<UnifFilename>      ::=  <FNameChars> <MoreFNameChars>* {L".uni"}
                         {L".UNI"}

<FilenameChars>     ::=  (\ua-\uz\uA-\uZ\u0-\u9)

<MoreFnameChars>    ::=  (\ua-\uz\uA-\uZ\u0-\u9\u_\u-)
```

2.1.1 Definitions

LanguageCodes

The language code must be a valid RFC4646 language code.

EscChar

In order to include some standard characters, such as the "\" back-slash character within a string, the character must be prefixed with the escape character. Characters that may require a prefixed escape character include the following, back slash "\" character, single-quote "'" character, double-quote '"' character and the forward slash "/" character. The back slash always requires the escape character.

StandardAttrs

The standard font attribute, "normal" was not defined in the *UEFI Specification*, however it has been proposed and is included here. Additional attributes defined in the *UEFI Specification*, such as double underline (dblunder), did not have the double-byte encoding for the character mapping, however recommendations have been given for these characters (see

below).

Token

The token (strong identifier) may only contain numbers, upper and lower case letters, underscore character, and dash character.

Include

An include line is used to parse another file, also compliant with this specification, as if it was in the file. The tokens should not overlap between the file for the same language.

2.2 Example file:

The following example uses ASCII characters. The actual content would be UCS-2LE encoded.

```
//
// Cpu I/O Strings
//
// Copyright (c) 2006, Intel Corporation. All rights reserved.<BR>
//
// This program and the accompanying materials are licensed and made
// available under the terms and conditions of the BSD License which
// accompanies this distribution.  The full text of the license may be
// found at:
//    http://opensource.org/licenses/bsd-license.php
//
// THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
// WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS
// OR IMPLIED.
//

/=#

#langdef  en-US "English, US"
#langdef  fr-FR "Français"

#string STR_PROCESSOR_VERSION
#language en-US
"NT32 Emulated Processor"
#language fr-FR
"Processeur Émulé par NT32"
```

Table 1 HII Double-Byte Encoding Map

String	Double-Byte Encoding	String	Double-Byte Encoding
\bold	0xF620	\endbold	0xF621
\italic	0xF622	\enditalic	0xF623
\underline	0xF624	\endunderline	0xF625
\dblunder	0xF62A	\enddblunder	0xF62B5
\emboss	0xF6265	\endemboss	0xF6275
\shadow	0xF6285	\endshadow	0xF6295
\n (newline)	0x2028	\f (formfeed)	0x000C
\r (carriage return)	0x000D	\p (paragraph separator)	0x2029

String	Double-Byte Encoding	String	Double-Byte Encoding
\ospace (ogham space mark)	0x1680	\enquad	0x2000
\emquad	0x2001	\ensp (en space)	0x2002
\emsp	0x2003	\em3sp (three-per-em space)	0x2004
\em4sp	0x2005	\em6sp	0x2006
\usp (punctuation space)	0x2008	\tsp (thin space)	0x2009
\hsp (hair space)	0x200A	\msp (medium math space)	0x205F
\!bsp (no-break space)	0x00A0	\!nbsp (narrow no-break space)	0x0202F
\zsp (zero width space)	0x200B	\ah (Armenian hyphen)	0x058A
\hy (hyphen)	0x2010	\df (figure dash)	0x2012
\den (en dash)	0x2013	\dem (em dash)	0x2014
\!bh (non-breaking hyphen)	0x2011	\g (Tibetan mark intersyllabic tsheg)	0x0F0B
\osp (Ethiopic wordspace)	0x1361	\k (Khmer sign bariyoosan)	0x17D5