SDL Extentions for OpenVMS Installation and User Guide

June 2006

This guide describes the SDLEXT SDL backends. It covers installation, release notes and use.

Revision/Update Information:	This is a new manual
Operating System/Version:	OpenVMS VAX V7.3
	OpenVMS Alpha V7.3-2
	OpenVMS I64 V8.2
Software Version:	ALPHA_SDL EV1-65
	SDL V2.1-5
	SDLEXT V1.0

November 2008

Contents

TABLES

Preface

This document is the definitive source for information on the SDLEXT collection of SDL backends. It contains release notes, installation information and a general user guide.

Intended Audience

This manual is intended for anyone looking to install or use the the SDLEXT collection. It is expected that readers will have a working knowledge of SDL.

Associated Documents

All SDL documentation is available from the Kednos website here:

```
http://www.kednos.com/kednos/Integration/SDL
```

Conventions

Table 1 lists the conventions used in this manual.

Table 1 Conventions Used in this Manual

Conventions	Meaning
ALPHA_SDL	This name refers to the port of the original VAX SDL (not publically available) product to Alpha. The last release was EV1-65 and is available on the OpenVMS Freeware distribution. It comes with full PL/I and BLISS source code
SDL	This name referes to the most recent release of the SDL compiler. The latest release is V2.3-0. This is a binary-only release and at present this product is closed source. It too is available on the OpenVMS Freeware distribution.

1 Getting Started

The SDLEXT product offers a growing collection of backend code generators for the Structure Definition Language compilers (SDL and ALPHA_SDL) available from HP.

This release of SDLEXT includes the following SDL backends:

- **Java**: The Java backend can be used to generate external routine definitions and structure declarations that can be used with the J2VMS product.
- **XSD**: The XML Schema Definition generator constructs XML schema documents that accurately describe SDL data structures. This can be very useful, particularly in parsing configuration data.

The SDL backends contained in this product are supported under ALPHA_SDL EV1-65 (possibly earlier versions) and HP SDL V2.1-5 and higher.

The rest of this chapter covers installation and removal of the SDLEXT software product.

1.1 Installation

Installation of the SDLEXT product is quite simple. The product is distributed in a PCSI kit that can be installed with the command PRODUCT INSTALL. Table 1–1 details the different options available at the time of installation. Example 1–1 demonstrates how to install the software product and the expected output.

Table 1–1 Installation Options

Options	Meaning
Documentation	The SDLEXT manual is installed by default to SYS$HELP. To prevent installation answer 'No' when prompted.
Examples	SDLEXT provides an example procedure that can be used to demonstrate the output of a backend. This output can then be used when comparing results from different backends. The particular example is adapted from the one that appeared in the VAX SDL manual.

Example 1–1 Installing SDLEXT

```
$ PRODUCT INSTALL SDLEXT

The following product has been selected:
    KEDNOS VMS SDLEXT V2.0                Layered Product

Do you want to continue? [YES]

Configuration phase starting ...

You will be asked to choose options, if any, for each selected product and for
any products that may be installed to satisfy software dependency requirements.

KEDNOS VMS SDLEXT V2.0: SDL Extentions for OpenVMS

    (C) Copyright 2003-2008 Tim E. Sneddon

    This software is distributed by Kednos Enterprises

    This product uses the PAK: SDLEXT

Do you want the defaults for all options? [YES] NO

    Install SDLEXT Documentation? [YES] YES

    Do you want the defaults for all suboptions? [YES] NO

      Install SDLEXT User Guide & Release Notes in HTML format? [YES] YES

      Install SDLEXT User Guide & Release Notes in PS format? [YES] YES

      Install SDLEXT User Guide & Release Notes in PDF format? [YES] YES

Do you want to review the options? [NO] NO

Execution phase starting ...

The following product will be installed to destination:
    KEDNOS VMS SDLEXT V2.0                DISK$AXP082:[VMS$COMMON.]

Portion done: 0%...10%...20%...30%...40%...80%...100%

The following product has been installed:
    KEDNOS VMS SDLEXT V2.0                Layered Product

KEDNOS VMS SDLEXT V2.0: SDL Extentions for OpenVMS

    Release Notes are included in the User Guide.

    SDLEXT release notes are included in the User Guide & Release Notes
    manual. To install these locally ensure that at least one
    documentation option is selected. Otherwise the manual can be
    found at the Kednos website.

    Insert the following lines in SYS$MANAGER:SYSTARTUP_VMS.COM:
        @SYS$STARTUP:SDLEXT_STARTUP.COM
```

1.2 How To Install SDLEXT Without PCSI

Sometimes it may not be feasible to install SDLEXT from the PCSI kit. This could be for any number of reasons, including a lack of privilege. It is possible to perform the installation manually by removing the backend images and procedures from the PCSI kit using the PRODUCT commands.

To get a listing of the files included in the kit use the PRODUCT LIST command and then use the PRODUCT EXTRACT FILE command to extract the necessary files. The example Example 1–2 demonstrates extracting the Java and XSD backends.

Example 1–2 Manual Product Installation

```
$ PRODUCT EXTRACT SDLEXT -
_$ /SELECT=(SDLEXT_JAVA.EXE,SDLEXT_XSD.EXE)

The following product has been selected:
    KEDNOS VMS SDLEXT V2.0                 Layered Product

Do you want to continue? [YES]

Portion done: 0%...100%
$ DEFINE SDL$JAVA SYS$DISK:[]SDLEXT_JAVA.EXE
$ DEFINE SDL$XSD SYS$DISK:[]SDLEXT_XSD.EXE
```

1.3 Post Installation

Following a successful installation the startup procedure, SYS$STARTUP:SDLEXT_STARTUP.COM should be added to the system startup procedure, most commonly SYS$STARTUP:SYSTARTUP_VMS.COM.

Executing this procedure at system startup ensures that the logicals necessary to use the backends provided by SDLEXT are setup. Table 1–2 shows the logicals that will be declared and the images the point to. Two sets of logicals are defined to ensure that the backends will work with both ALPHA_SDL and SDL.

Table 1–2 SDL Backend Logicals

Backend	ALPHA_SDL Logicals	SDL Logicals	Location
Java	ALPHA_SDLJAVA	SDL$JAVA	SYS$LIBRARY:SDLEXT_SDLJAVA.EXE
XSD	ALPHA_SDLXSD	SDL$XSD	SYS$LIBRARY:SDLEXT_SDLXSD.EXE

1.4 Removal

Removal of the SDLEXT software product is performed using the PCSI command PRODUCT REMOVE. Example 1–3 demonstrates the removal of SDLEXT and the expected output.

Example 1–3 Removing SDLEXT

```
$ PRODUCT REMOVE SDLEXT

The following product has been selected:
    KEDNOS VMS SDLEXT V2.0                Layered Product

Do you want to continue? [YES]

The following product will be removed from destination:
    KEDNOS VMS SDLEXT V2.0                DISK$AXP082:[VMS$COMMON.]

Portion done: 0%...50%...60%...70%...80%...100%

The following product has been removed:
    KEDNOS VMS SDLEXT V2.0                Layered Product
```

2 User Guide

This section of the manual covers how to use the SDLEXT backends, this includes example output as well as a summary of the code generated by the various backends.

2.1 Invoking The Backends

Invoking the backends is quite simple. They conform to the requirements set forth by the SDL compiler and so can be called as any other SDL code generator might be.

Under ALPHA_SDL the following command could be used to generate Java code suitable for used with J2VMS from the module EXAMPLE.SDL that ships with the product.

```
$ SDL/ALPHA/LANGUAGE=JAVA EXAMPLE.SDL/VMS_DEVELOPMENT
```

Under SDL the above command will suffice. However, it is nolonger necessary to specify either /VAX or /ALPHA.

The remainder of this chapter covers the different backends provided as part of the SDLEXT software product and the output the generate.

2.2 Java

The Java backend can be used to generate structure and external routine declarations that can then be used with the product J2VMS. The structure declarations mimic those generated by the BLISS backend as both languages use similar mechanisms when accessing native data structures. The following sections describe the output in close detail.

2.2.1 Translation Summary

This sections details what Java code is generated by which SDL statements. See Table 2–1 for details.

Table 2–1 Java Translation Summary

SDL Declaration	Java Output
MODULE **name**	public class **name** {
IDENT **string**	// IDENT **string**
/* **comment**	// **comment**
CONSTANT **x** EQUALS **n**	public static final int **x** = **n**;

Table 2–1 (Cont.) Java Translation Summary

SDL Declaration	Java Output

ENTRY **name**	private static SystemCall nullclass;
	private static final String libname = "libname";
	private static SystemCall sdl$**name**;
	private static int sdl$**name**(VMSparam[] args) {
	if (sdl$**name**_return == nullclass) {
	sdl$**name**_return = new SystemCall("sdl$**name**", libname);
	}
	return sdl$**name**_return.call(args);
	}

PARAMETER (**type**, . . .)	n/a
ANY	n/a
DESCRIPTOR	n/a
IN	n/a
OUT	n/a
NAMED **param-name**	n/a
VALUE	n/a
REF	n/a
DEFAULT	n/a
OPTIONAL	n/a
TYPENAME **type-name**	n/a
RETURNS **return-type**	n/a
VARIABLE	n/a
ALIAS	n/a
LINKAGE	n/a
name STRUCTURE	Each aggregate or member declaration in SDL produces a J2VMS FieldDescriptor definition of the form:
name UNION	public static final FieldDescriptor **name** = new FieldDescriptor(off,pos,size,ext);
BYTE	
WORD	off
LONGWORD	Byte offset of this aggregate or item
QUADWORD	within the current aggregate.
OCTAWORD	
BYTE UNSIGNED	pos
WORD UNSIGNED	The bit position from the offset
LONGWORD UNSIGNED	
QUADWORD UNSIGNED	size
OCTAWORD UNSIGNED	The size of the aggregate or item,

Table 2–1 (Cont.) Java Translation Summary

SDL Declaration	Java Output
F_FLOATING	in bits, if the size is 4 bytes
D_FLOATING	or less. Otherwise, this field contains
DECIMAL PRECISION (**p**, **q**)	0, and SDLJAVA generates the size declaration
BITFIELD LENGTH **n**	
MASK	S_**name** = size;
SIGNED	
BOOLEAN	where the size is given in bytes
CHARACTER LENGTH **n**	
VARYING	
ADDRESS	ext Contains 0 if the value is zero extended, or 1 if the value is sign extended or SIGNED bit.
COMMON **storage class**	n/a
GLOBAL **storage class**	n/a
with /GLOBALREF	n/a
BASED **pointer-name**	n/a
DIMENSION [**lbound**:]**hbound**	
ORIGIN **member-name**	n/a

Note: Field and routine names can change when using the /VMS_DEVELOPMENT qualifier. Read the following section carfully to determine if this is right for you.

2.2.2 Qualifiers

The architecture specific qualifiers /ALPHA and /VAX have no bearing on the the output generated by this backend. The Java VM only offers a 32-bit virtual machine and has no support for architecture specific features. The JVM is not even available on OpenVMS VAX. However, it is still possible to run the Java backend on OpenVMS VAX.

The /VMS_DEVELOPMENT qualifier causes names to be normalized much in the same way as the CC backend. The only major difference here is that both all-upper and all-lower case names are provided. This is to provide names that conform to the CC backend, as well as names that conform to the Starlet library provided with J2VMS V1.2.

2.2.3 Feature Logicals

The Java backend relies on a couple of logicals to ensure that some Java specific details, not covered by the SDL compiler, make it into the output. Table 2–2 describes these logicals and their purposes.

Table 2–2 Java Feature Logicals

Logicals	Description
SDLJAVA_LIBNAME	Use this logical to define the name of the shareable image containing all the ENTRY definitions

SDLJAVA_PACKAGE	This logical controls the name of the Java package the module is associated with. It can be defined like so:
	`$ DEFINE/USER SDLJAVA_LIBNAME -`
	`_$ "org.tes.sdlext"`This will generate code similar to:
	`package org.tes.sdlext;`

2.3 XSD

The section documents the behaviour of the XSD backend.

2.3.1 Translation Summary

The following SDL output summary is based on those found in the "VAX SDL (Structure Definition Language)", Software Version VAX SDL 3.0 manual.

Table 2–3 XSD Translation Summary

SDL Declaration	XSD Output
MODULE **name**	<!– *** MODULE **name** *** –>
IDENT **string**	<!– **string** –>
/* **comment**	<!– **comment** –>
CONSTANT **x** EQUALS **n**	n/a
ENTRY **name**	n/a
PARAMETER (**type**, . . .)	n/a
ANY	n/a
DESCRIPTOR	n/a
IN	n/a
OUT	n/a
NAMED **param-name**	n/a
VALUE	n/a
REF	n/a

Table 2–3 (Cont.) XSD Translation Summary

SDL Declaration	XSD Output
DEFAULT	n/a
OPTIONAL	n/a
TYPENAME **type-name**	n/a
RETURNS **return-type**	n/a
VARIABLE	n/a
ALIAS	n/a
LINKAGE	n/a
name STRUCTURE	<xsd:element name="**name**"> <xsd:complexType> <xsd:sequence> . . . </xsd:sequence> </xsd:complexType> </xsd:element>
name UNION	<xsd:element name="**name**"> <xsd:complexType> <xsd:sequence> <xsd:choice> <xsd:element> . . . <xsd:element> . . . </xsd:choice> </xsd:sequence> </xsd:complexType> </xsd:element>
COMMON **storage class**	n/a
GLOBAL **storage class**	n/a
with /GLOBALREF	n/a
BASED **pointer-name**	n/a
DIMENSION [**lbound**:]**hbound**	Arrays are handled a bit differently. An array will generate an element of the array name containing an element with a "maxOccurs" of **hbound**. This element is named prefix + tag + "_item" with the base type of the array.
ORIGIN **member-name**	n/a

Table 2–4 shows the SDL data types and their correspondence to XSD data types. The code example below shows the code template used to generate field definitions.

```
<xsd:element name="field-name">
  <xsd:simpleType>
    <xsd:restriction base="type-name"/>
  </xsd:simpleType>
</xsd:element>
```

Table 2–4 XSD Type Translations

SDL Type Constraints	XSD Type Constraints
BYTE	xsd:byte
WORD	xsd:short
LONGWORD	xsd:int
QUADWORD	xsd:long
OCTAWORD	xsd:integer
BYTE UNSIGNED	xsd:unsignedByte
WORD UNSIGNED	xsd:unsignedShort
LONGWORD UNSIGNED	xsd:unsignedInt
QUADWORD UNSIGNED	xsd:unsignedLong
OCTAWORD UNSIGNED	xsd:integer xsd:minInclusive="0"
F_FLOATING	xsd:float
D_FLOATING	xsd:double
G_FLOATING	xsd:anyType
H_FLOATING	xsd:anyType
DECIMAL PRECISION	xsd:decimal
(**p**, **q**)	xsd:totalDigits="**p**" xsd:fractionDigits="**q**"
BITFIELD LENGTH **n**	xsd:integer xsd:minInclusive="0" xsd:maxInclusive="(2^**n** - 1")
MASK	n/a
SIGNED	n/a
BOOLEAN	xsd:boolean
CHARACTER LENGTH **n**	xsd:string
VARYING	xsd:maxLength="**n**"
ADDRESS	xsd:unsignedInt[1] xsd:unsignedLong[2]

[1]VAX only

[2]Alpha and I64 only

Note: If an element is within a structure or union then the facet "maxOccurs" on the element will be set to 0.

2.3.2 Qualifiers

The XSD output in not effected by the /xxx_DEVELOPMENT qualifiers. All other qualifiers, such as /SUPPRESS, apply as normal.

2.3.3 Feature Logicals

The SDLXSD_OPTIONS feature logical has been deprecated. However, it is still available and although documented below it is recommended that /SUPPRESS be used instead.

This logical was introduced specifically for generating configuration file schemas. This allows the XML to have the same fields and value restrictions as the configuration block, just without the prefix and/or tag.

SDLXSD_OPTIONS translates to a comman delimited list. Currently the only two options are:

Table 2–5 SDLXSD_OPTIONS Keyword

Keyword	Description
NOPREFIX	Disables the prefix set in the SDL source module. It is equivalent to /SUPPRESS=PREFIX.
NOTAG	Disables the tag set in the SDL source module. It is equivalent to /SUPPRESS=TAG.

2.3.4 Names

Unfortunately, the XML standard says that the dollar sign is not acceptable in a name so SDLXSD converts this to an underscore. There is no command line qualifier or feature logical to switch this on or off as no correctly conforming schema based XML parser would allow it.