Advanced Design System 2011.01

Feburary 2011
AEL

Acknowledgments

7-zip - 7-Zip Copyright: Copyright (C) 1999-2009 Igor Pavlov. Licenses for files are: 7z.dll: GNU LGPL + unRAR restriction, All other files: GNU LGPL. 7-zip License: This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful,but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA. unRAR copyright: The decompression engine for RAR archives was developed using source code of unRAR program.All copyrights to original unRAR code are owned by Alexander Roshal. unRAR License: The unRAR sources cannot be used to re-create the RAR compression algorithm, which is proprietary. Distribution of modified unRAR sources in separate form or as a part of other software is permitted, provided that it is clearly stated in the documentation and source comments that the code may not be used to develop a RAR (WinRAR) compatible archiver. 7-zip Availability: http://www.7-zip.org/

AMD Version 2.2 - AMD Notice: The AMD code was modified. Used by permission. AMD copyright: AMD Version 2.2, Copyright © 2007 by Timothy A. Davis, Patrick R. Amestoy, and Iain S. Duff. All Rights Reserved. AMD License: Your use or distribution of AMD or any modified version of AMD implies that you agree to this License. This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Permission is hereby granted to use or copy this program under the terms of the GNU LGPL, provided that the Copyright, this License, and the Availability of the original version is retained on all copies.User documentation of any code that uses this code or any modified version of this code must cite the Copyright, this License, the Availability note, and "Used by permission." Permission to modify the code and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included. AMD Availability: http://www.cise.ufl.edu/research/sparse/amd

UMFPACK 5.0.2 - UMFPACK Notice: The UMFPACK code was modified. Used by permission. UMFPACK Copyright: UMFPACK Copyright © 1995-2006 by Timothy A. Davis. All Rights Reserved. UMFPACK License: Your use or distribution of UMFPACK or any modified version of UMFPACK implies that you agree to this License. This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Permission is hereby granted to use or copy this program under the terms of the GNU LGPL, provided that the Copyright, this License, and the Availability of the original version is retained on all copies. User documentation of any code that uses this code or any modified version of this code must cite the Copyright, this License, the Availability note, and "Used by permission." Permission to modify the code

and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included. UMFPACK Availability: http://www.cise.ufl.edu/research/sparse/umfpack UMFPACK (including versions 2.2.1 and earlier, in FORTRAN) is available at http://www.cise.ufl.edu/research/sparse . MA38 is available in the Harwell Subroutine Library. This version of UMFPACK includes a modified form of COLAMD Version 2.0, originally released on Jan. 31, 2000, also available at http://www.cise.ufl.edu/research/sparse . COLAMD V2.0 is also incorporated as a built-in function in MATLAB version 6.1, by The MathWorks, Inc. http://www.mathworks.com . COLAMD V1.0 appears as a column-preordering in SuperLU (SuperLU is available at http://www.netlib.org). UMFPACK v4.0 is a built-in routine in MATLAB 6.5. UMFPACK v4.3 is a built-in routine in MATLAB 7.1.

Qt Version 4.6.3 - Qt Notice: The Qt code was modified. Used by permission. Qt copyright: Qt Version 4.6.3, Copyright (c) 2010 by Nokia Corporation. All Rights Reserved. Qt License: Your use or distribution of Qt or any modified version of Qt implies that you agree to this License. This library is free software; you can redistribute it and/or modify it under the
terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Permission is hereby granted to use or copy this program under the terms of the GNU LGPL, provided that the Copyright, this License, and the Availability of the original version is retained on all copies.User
documentation of any code that uses this code or any modified version of this code must cite the Copyright, this License, the Availability note, and "Used by permission." Permission to modify the code and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included. Qt Availability: http://www.qtsoftware.com/downloads Patches Applied to Qt can be found in the installation at:
$HPEESOF_DIR/prod/licenses/thirdparty/qt/patches. You may also contact Brian Buchanan at Agilent Inc. at brian_buchanan@agilent.com for more information.

The HiSIM_HV source code, and all copyrights, trade secrets or other intellectual property rights in and to the source code, is owned by Hiroshima University and/or STARC.

Errata The ADS product may contain references to "HP" or "HPEESOF" such as in file names and directory names. The business entity formerly known as "HP EEsof" is now part of Agilent Technologies and is known as "Agilent EEsof". To avoid broken functionality and to maintain backward compatibility for our customers, we did not change all the names and labels that contain "HP" or "HPEESOF" references.

Warranty The material contained in this document is provided "as is", and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this documentation and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with

these terms, the warranty terms in the separate agreement shall control.

Technology Licenses The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license. Portions of this product include the SystemC software licensed under Open Source terms, which are available for download at http://systemc.org/ . This software is redistributed by Agilent. The Contributors of the SystemC software provide this software "as is" and offer no warranty of any kind, express or implied, including without limitation warranties or conditions or title and non-infringement, and implied warranties or conditions merchantability and fitness for a particular purpose. Contributors shall not be liable for any damages of any kind including without limitation direct, indirect, special, incidental and consequential damages, such as lost profits. Any provisions that differ from this disclaimer are offered by Agilent only.

Restricted Rights Legend U.S. Government Restricted Rights. Software and technical data rights granted to the federal government include only those rights customarily provided to end user customers. Agilent provides this customary commercial license in Software and technical data pursuant to FAR 12.211 (Technical Data) and 12.212 (Computer Software) and, for the Department of Defense, DFARS 252.227-7015 (Technical Data - Commercial Items) and DFARS 227.7202-3 (Rights in Commercial Computer Software or Computer Software Documentation).

Introduction to AEL

Application Extension Language (AEL) programming language, modeled after the popular C programming language, is used to configure, customize and extend the capabilities of the design environment. Like C, AEL has an extensive set of built-in function libraries, including functions for file input/output, math, string manipulation, list handling, and database query.

AEL can be used for:

- Organizing libraries and palettes of components.
- Defining the interface to new user-defined components.
- Creating new components with layout artwork.
- Defining custom layout artwork functions.
- Defining the interface to discrete-valued simulation components.
- Creating custom utility functions, such as parts list generators and bill of materials.
- Automating routine tasks, such as repetitive command sequences, batch analysis, or optimizations.

For a table that shows where each function can be used, see *Where to Use AEL Functions* (ael).

Note
Only a small subset of the language is used for component, library, or palette definitions. If you just want to customize a library or palette, you can skip the general language description and start at the section *Using AEL for Library Definition* (ael).

General AEL Structure

The AEL language is similar to the C language in many ways. AEL is a procedural language, with many of the same language components as C, such as the use of variables and functions, arithmetic expressions, program flow, and logic statements. However, there are significant differences between AEL and the C language:

- AEL has no pre-processor; therefore, the *#if*, *#ifdef*, *#ifndef*, *#endif*, *#define*, *#undef* and *#include* directives are not supported. However, AEL files can reference other AEL files using the load() function, which has an effect similar to *#include*.
- Variables, pass parameters and function return values are not typed; therefore, the C keywords *int, char, float, double, long, short, unsigned, auto, register,* and *typedef* are not supported. Although a value has a particular type, the type of a variable, pass parameter, or function return is dependent only on the most recently associated value, not on the declaration of the variable, parameter, or function. Supported values are long, double, string, and boolean.
- AEL also supports complex numbers and lists which are not supported by C.
- AEL does not support structures or unions which are part of standard C, nor does it support external or static variables. All variables defined outside of a function are global.
- Variables can be declared globally (outside a function definition), local to a function, or declared within a compound statement.
- AEL supports a large subset of the C operators and logic and control constructs. However, AEL does not support the right arrow (->) and period (.). Brackets ([ ]) are supported for list access only. For example:

```
a=list("yes", "no")
b=[0] - "yes"
b=[1] - "no"
```

The *switch, case,* and *default* logic and control constructs are supported.

- AEL does not support casting; however, there are several type conversion functions.
- AEL features automatic *garbage collection*, a language feature that automatically reclaims storage when it is no longer needed.

> **Note**
> If you are unfamiliar with the C language, refer to *The C Programming Language* by Kernighan and Ritchie.

Language Specifics

The AEL language can be described in terms of the six classes of tokens: comments, identifiers, keywords, constants, operators, and other separators. Like the C language, blanks, tabs, new lines, and comments are ignored except as separators for other tokens.

Comments

The characters /* or // introduce a comment. If the characters /* are used, the comment ends with the next occurrence of the characters */. If the characters // are used, the comment continues to the end of the line.

Naming Convention for Variables

Letter	Variable Type
b	Boolean
c	Complex
d	Double
f	Float
h	Handle
i	Integer
r	Real
s	String

Lists

All lists are 0 based, starting at 0, not 1.

Identifiers

An identifier is a sequence of letters, digits, and underscore, where the first character must not be a digit. Case is significant for identifiers. There is no limit to the length of an

identifier. Identifiers are used to represent function names, variable names, or statement labels.

Keywords

These identifiers are reserved for use as keywords:

Keyword	Description
break	leave a loop, exit switch statement
continue	start next iteration of a loop
decl	define a variable
defun	define a function
do	begin a do {} while (); loop
else	begin alternate action of an if()-conditional or of an inline if()-then-else conditional
for	begin for (;;) loop
goto	goto a labelled statement
if	begin if() conditional
NULL	constant NULL
return	return from a function
while	begin while(){} loop, or end do{} while(); loop
*_FILE_*	current filename
*_LINE_*	current line number
default	defines default case for switch(){} statement
switch	begin a switch (){} statement
case	defines a case 1A switch (){case: }
elseif	begin alternate if() conditional to an if() conditional
then	defines the action for inline if() then else statements
AND	logical and
OR	logical or
EQUALS	logical equal operator
NOT	logical not
list	define a list

Constants

AEL supports two constant forms not supported by C: the null value and the imaginary number. These constants represent internal data types not found in C. The list is also an internal data type, but there is no constant expression for a list. The supported forms of constants are:

Supported Constant	Description
null value	NULL
decimal integer constant	13
hexadecimal integer constant	0x3E (case is not significant)
octal integer constant	0377
string constant	"a string"
real number constant	10.3 or 25.4e-3 (case is not significant)
imaginary number constant	3.5i or 4+3.5i

A string constant consists of one or more characters enclosed in quotation marks (" "). Unlike C, AEL does not allow you to treat strings as arrays of characters. A string constant can have embedded non-printable characters, expressed using the backslash:

Non-Printable Character	Description
\n	new line
\r	return
\f	form feed
\b	backspace
\t	tab
\"	double quote
\ \	backslash
\xNN	character in hexadecimal notation (N is 0 - 9 or A - F; case is not significant)
\0NNN	character in octal notation (N is 0 - 7)

Note
If you do not want to convert control characters, use 2 single quotes around the string instead of quotation marks; for example: `''` \usr\local\nim.abc `''`.

Predefined Global Constants

To simplify the use of the database query functions, a set of symbolic constants, called global variables, have been predefined. The global boolean definitions are: TRUE, FALSE. Other global variables are shown with the applicable function definition.

Global Variable	Definition
TRUE	boolean true
FALSE	boolean false
stdin	standard in
stdout	standard out
stderr	standard error
hugeReal	3.4e +38
tinyReal	2.2e -308
e	2.718281828
ln10	ln(10) 2.302585093
c0	speed of light 2.997 924 58 e+08 m/s
e0	vacuum permittivity 8.8541878176204e-12 F/m
u0	vacuum permittivity 1.2566370614359e-06 H/m
boltzmann	Boltzmann's constant 1.380658e-23 J/K
qelectron	charge of an electron 1.60217733e-19 C
planck	Planck's constant 6.6260755e-34 J-sec
PI	= pi = 3.1415926535898
on_PC	TRUE if running on PC, else FALSE

Note that literally thousands of functions and global variables are defined in the Advanced Design System Design Environment (DE). To ensure that your functions and global variables do not interfere with those provided in the program, you should add a special prefix to your functions and variables. Further, different setups or users at your site should use unique definitions to distinguish newly-created sets of functions and variables (such as, by using a special prefix). For example:

mc1_varname
mimic_abc

Caution
Do not use a dollar sign ($) as a special operator, since this character is used in expressions.
2pi is not equal to 2 * pi. 2pi is interpreted as 2 scale factor p for pico and i for imaginary part (= 2.000E-12).

Operators

Constants and identifiers can be combined, by using operators, to form expressions. Expressions can be combined again, by using operators, to form more complex expressions. Expressions are the building blocks of AEL. Expressions can be used in assignments to variables and passed as parameters when calling AEL functions. AEL expressions are described and evaluated in the same way as in the C language, where operators are applied in order of their precedence. As with C, the default precedence applied in evaluating an expression can be overridden by grouping operands with parentheses. The following table lists (by operator precedence) the mathematical operators that are recognized in expressions.

Mathematical Operators by Operator Precedence

Operator	Description
()	Expression grouping
!	Logical NOT (unary)

~	Bitwise one's complement (unary)
++	Pre- and post-increment operator
-	Pre- and post-decrement operator
-	Negates a value (unary)
*	Used preceding an identifier to calculate value stored at an address (unary)
&	Used preceding an identifier to specify address of a variable (unary)
%	Integer remainder after division (modulus)
/	Division
+	Addition
-	Subtraction
>=	Test for first value greater than or equal to second
<=	Test for first value less than or equal to second
>	Test for first value greater than second
<	Test for first value less than second
==	Test for equal values
!=	Test for not equal values
&	Bitwise AND
^	Bitwise exclusive OR
|	Bitwise inclusive OR
&&	Logical AND
||	Logical OR
?:	Conditional evaluation operator (a ternary operator)
=	Assignment operator
*=	Multiplication assignment operator
/=	Division assignment operator
%=	Integer remainder assignment operator
+=	Addition assignment operator
^=	Bitwise exclusion or assignment operator
−=	Subtraction assignment operator
|=	OR assignment operator
&=	AND assignment operator
<<	Bitwise left shift
>>	Bitwise right shift
<<=	Left shift assignment operator
>>=	Right shift assignment operator
,	Sequential evaluation operator
**	Power operator
::	Used to define sequences; for example, 1::10 or 1::10::2
$	Indicates substituting the variable for its value
AND	Logical AND
OR	Logical OR
EQUALS	Test for equal values (same as ==)
NOT	Logical NOT
.	Apply this operation to all elements of the array operand. See the following section The Apply-to-All-Elements (.) Math Operator.

Operators which have one operand are called unary operators. These include !, ~, ++, −, −, *, and &. The operand follows the operator except for ++ and −. The operator follows the operand in the post-increment and post-decrement use of ++ and −. Unlike C, the

unary * and & operators can be used only with identifiers. However, they generate the address or dereference an address much like C does.

Binary operators have two operands which are separated by the operator. Most of the operators fall into this category. Most of the binary operators perform mathematical operations on real numbers, but there are a few exceptional operators.

The operators !, &&, and || work with the logical values. The operands are tested for the values TRUE and FALSE, and the result of the operation is either TRUE or FALSE. In AEL a logical test of a value is TRUE for non-zero numbers or strings with non-zero length, and FALSE for 0.0 (real), 0 (integer), NULL or empty strings. Note that the right hand operand of && is only evaluated if the left hand operand tests TRUE, and the right hand operand of || is only evaluated if the left hand operand tests FALSE.

The operators >=, <=, >, <, ==, !=, AND, OR, EQUALS, and NOT EQUALS also produce logical results, producing a logical TRUE or FALSE upon comparing the values of two expressions. These operators are most often used to compare two real numbers or integers. These operators operate differently in AEL than C with string expressions in that they actually perform the equivalent of *strcmp()* between the first and second operands, and test the return value against 0 using the specified operator.

The operators ~, &, ^, |, <<, and >> are considered bitwise operators and work only with integers, manipulating the binary bits of the value.

There are several assignment operators: =, +=, -=. *=, /=, %=, ^=, |=, &=, <<=, and >>=. These operators always have an identifier as the left hand operand, and modify the value of the variable as well as returning its value after modification. Multiple assignments are supported (e.g., a=b=4) which evaluate right to left.

As in C, there is only one ternary operator, the conditional evaluation operator, which has the form:

```
expression ? expression : expression
```

This operator evaluates the first expression and tests it. If the value of the first expression is zero it evaluates the third expression, and if not zero it evaluates the second expression.

An expression can also be a function call, which takes the following form:

```
identifier( argument_list )
```

The *identifier* is the name of the function and *argument_list* is a list of comma separated expressions representing the values of the parameters passed to the function. The function can return a value for use as an operand in evaluating additional expressions.

The sequential evaluation operator is used to cause several comma separated expressions to be evaluated in left-to-right order, with only the last value returned.

The Apply-to-All-Elements (.) Math Operator

Any math operator prefixed by a "." (for example, .* .+ ./ .-) applies the following operation to all elements of the array operand or a dataset data operand.

If one operand is an array/dataset data operand and the other is a single value, the single value and the operation will be executed on every element of the array/dataset data operand, thereby creating a new array/dataset data variable.

If both operands are arrays/dataset data operands, then their dimensions are ensured to be the same, and each element of the array/dataset data operand is operated on the element in the same position in the other array/dataset data operand. A new array/dataset data variable is created of the same size with the new values.

Other Separators

In addition to the operators, several other separators are used in AEL to form statements:

Separator	Description
;	Terminates an AEL statement.
:	Terminates an identifier to define a statement label.
{}	Groups one or more statements together to form a compound statement. Variables declared within braces are defined only within the scope of the braces.

Building AEL Programs

AEL programs consist of a sequence of one or more of the following forms of AEL statements:

- Comment
- Simple statement
- Compound statement
- Variable declaration
- Function definition
- Control and logic statements

Comment

Comments are allowed anywhere, as long as the contents of the comment are not required to complete a statement or function definition. Comments begin with // or /*. In the first case, the comment continues to the end of the line, and in the second case the comment ends with */.

```
// single line comment
/* multi-line comment */
```

Simple Statement

A simple AEL statement is an AEL expression terminated by a semicolon (;) where the resulting value of the expression is discarded. Expressions often have side effects caused by evaluation which can be more important than the resulting value. Some examples of

simple AEL statements are:

AEL Statement	Description
a=5;	Value of a is set to 5
my_fun(1, 2);	Function *my_fun* is called, with the values 1 and 2 passed as parameters, return value of the function is discarded
b++;	Value of *b* is incremented by one

Compound Statement

A compound statement is several statements grouped together inside braces { }. Any type of statement can be contained in a compound statement. Compound statements are used to define the body of a function, switch statement, or the action of a conditional or loop statement.

Variable Declaration

The keyword *decl* begins a variable declaration, followed by a comma-separated identifier list, and terminated by a semicolon. Each identifier can be given an initial value using the = assignment operator. An example of a variable declaration is:

```
decl a, b=5, cosA=cos(A);
```

A variable declaration without an initial value has the value NULL. A variable can be declared more than once, in which case the first declaration prevails and the identity of the variable is maintained through subsequent declarations.

Note
If a variable is declared with an initial value, the 2nd declaration prevails. That is, if a global variable is re-declared *and* initialized, then the global variable writes over the previous value; if the global variable is declared *but is not initialized*, the previous value is retained.

A variable declaration outside of a function definition is considered global. Inside a function definition a variable is considered local to the function. Variable declarations contained inside a compound statement are hidden from expressions outside the compound statement.

AEL Lists

Declaration and Initialization of an AEL List:

An AEL list is a collection of AEL values. It is created by:

```
list(<val1>, <val2>, etc...);
```

where <valx> is any valid AEL value, including another AEL list.

Example:

```
decl a = list(1,2,3);
decl b = -3.44;
decl c = list(1,1.5, "hello",a, b);
```

Printing Lists:

The value of a list can be printed using the *identify_value()* function.

Example:

```
fputs(stderr,identify_value(c));
// output is:  list(1,1.5,"hello", list(1,2,3), -3.44)
```

See *List Management Functions* (ael) for documentation on traversing lists and other operations on lists.

Function Definitions

The general form of a function definition is:

```
defun identifier (parameter_list)
{
     statements (body)
}
```

The keyword *defun* begins a function definition. The name of the function is the *identifier*. The *parameter_list* is a list of comma-separated identifiers defining the parameters of the functions. A return statement (consisting of a return keyword optionally followed by an expression and terminated by a semicolon) ends the function and specifies the return value. All AEL functions return a value. If the return statement is missing or no value was given for the return statement, the return value is NULL. If a function returning a NULL value is used in the context of an expression, the evaluation of the expression can fail, causing an AEL error.

Note
AEL functions must be defined before they can be called.

Control and Logic Statements

AEL supports many of the C control and logic capabilities. The control and logic statements which can be used are:

goto label;
if (expression) block
else if (expression) block
else block
for (statement; expression; statement) block
do block *while* (expression);
while (expression) block
switch (expression) {case statements}

where:

label is a defined statement label;
expression is a combination of values, functions, and operators evaluating to a single value;
statement is also a single expression where the value is discarded; and
block is a single statement, or several statements enclosed in braces.

If *goto* is used within a function definition, the label must be defined in the same function definition. Switch and case statements are the same as case statements in C language.

AEL Arrays

Declaration and Initialization

AEL supports multidimensional arrays. An array must be initialized in a declaration or in an assignment statement.

Note
The number of elements in the initialization statement MUST be the same as the loop. You can initialize an array to one element, and then resize it later using the resize_array() function.

Note
In AEL arrays, {} and [] are treated the same. If you are using data from datasets, {} is a row vector and [] is a column vector.

Array elements are separated by ",". Array elements can be integers, doubles, or complex numbers. Sequences can also be used to initialize an integer or double array.

When specifying values in a dimension, you can use the following inside of either {}'s or []'s:

- scalar values separated by commas
- a sequence
- a sequences with steps

If you are specifying an array with more than one dimension, you can specify an inner dimension WITHOUT using {}'s or []'s by using a ";" to separate the dimensions.

Examples:

Row vectors or one dimensional array:

```
decl x = {1+3i, 4-5i, 9+3i, 10i};
```

This array consists of the following elements:

x[0] = 1+3i
x[1] = 4-5i
x[2] = 9+3i

x[3] = 0+10i

```
x = [1+3i, 4-5i, 9+3i, 10i];
```

This array consists of the following elements:

x[0] = 1+3i
x[1] = 4-5i
x[2] = 9+3i
x[3] = 0+10i

```
x = {1::10)
```

This array shows initializing with a sequence and consists of the following elements:

x[0] = 1
x[1] = 2
x[2] = 3
x[4] = 4
x[5] = 5
...
x[9] = 10

```
x = {1::0.5::2)
```

This array shows initializing with a sequence *and* a step (how much to increment) and consists of the following elements:

x[0] = 1
x[1] = 1.5
x[2] = 2

Column vector or two dimensional array with row dimension being 0:

```
decl z;
z = {{1},{2},{3}};
z = [1;2;3];
```

Both examples produce the same array and consists of the following elements:

z[0] = {1} (one dimensional array)
z[1] = {2}
z[2] = {3}
z[0,0] = 1 (scalar values)
z[1,0] = 2
z[2,0] = 3

Matrix or a two dimensional array:

```
decl y;
y = { {1,2}, {3,4}, {5,6}, {7,8} };
y = [ 1,2; 3,4; 5,6; 7,8 ];
```

Both examples produce the same array and consists of the following elements:

y[0] = {1,2} (one dimensional array)
y[1] = {3,4}
y[2] = {5,6}
y[3] = {7,8}
y[0,0] = 1 (scalar values)
y[0,1] = 2
y[1,0] = 3
y[1,1] = 4
y[2,0] = 5
y[2,1] = 6
y[3,0] = 7
y[3,1] = 8

Three dimensional array:

```
decl s;
s = {[0,1;2,3],[4,5;6,7]};
s = {{{0,1},{2,3}},{{4,5},{6,7}}}
```

Both examples produce the same array and consists of the following elements:

s[0] = `0,1},{2,3` (two dimensional array)
s[1] = `4,5},{6,7`
s[0,0] = {0,1} (one dimensional array)
s[0,1] = {2,3}
s[1,0] = {4,5}
s[1,1] = {6,7}
s[0,0,0] = 0 (scalar values)
s[0,0,1] = 1
s[0,1,0] = 2
s[0,1,1] = 3
s[1,0,0] = 4
s[1,0,1] = 5
s[1,2,0] = 6
s[1,1,1] = 7

Printing Arrays

The value of an array can be printed using the identify_value() function.

Example:

```
// this would output "{1+3i, 4-5i, 9+3i, 10i\}"
fprintf(stdout, "%s\n", identify_value(x));
// this would output "{{1,2},{3,4},{5,6},{7,8}}"
fprintf\(stdout, "%s\n", identify_value(y));
```

Operations Between Arrays

Some operations on a whole array can be executed. These include assignments, additions, subtraction, multiplication (which is essentially matrix multiplication), division, and negation.

When assignments are made, the receiver of the assignment will point to the same array as the original array. However, when a modification is made to an array, a new array is created so that other variables pointing to that array will not be modified. This is a significant difference between AEL arrays and C arrays.

Example:

```
decl x = {1,2,3,4};
// z = {1,2,3,4}
decl z = x;
// x = {1,2,3,4} and z = {5,5}
z = {5,5};
```

When addition or subtraction is executed between whole arrays, the math is executed one element at a time. Therefore, the array bounds must be exactly the same. Element by element multiplication, division and power are also supported by using the operators: **.**, **./, and .***.

Example:

```
decl x = {1,2,3,4};
decl y = {9,8,7,6};
// z = {10,10,10,10};
decl z = x + y;
x = { {1,2}, {3,4} };
y = { {10,9}, {8,7} };
// z = { {-9,-7}, {-5,-3} };
z = x - y;
// z = { {10,18}, {24,28} };
z = x .* y;
```

When multiplication is executed between whole arrays, matrix multiplication is executed. Therefore, the arrays must be 2 dimensional and the array bounds must be compatible for matrix multiplication, which is row1 x col1 * row2 x col2 where the dimension of col1 must equal the dimension of row2. The new array will have the dimensions row1 x col2; that is, 4 x 2 * 2 x 8 = 4 x 8.

Example:

```
x = { {1,2}, {3,4} };
y = { {10,9}, {8,7} };
// z = { {26,23}, {62,55} };
z = x * y;
```

Note
Currently, the following short hand notations are supported: +=, -=, *=.

Operations Between Scalar and Arrays

Any math operation between an array and scalar values is legal. The result of such expressions is an array.

Example:

```
decl x = {1,2,3,4};
// x = "{6,7,8,9}"
x += 5;
// x = "{32,28,24,22}"
x = 100 / x * 2;
```

Transpose

Transposition of arrays is supported. The syntax is:

Example:

```
decl x = {1,2,3,4};
// y = {{1},{2},{3},{4}}
y = x';
```

Access and Operations on Array Elements

Individual elements are accessed by specifying the array name followed by the index(s) enclosed in square brackets "[]". There are three ways to access elements of multidimensional arrays:

- The indices can be separated by commas and the whole sequence enclosed in square brackets; for example, y[1,3]
- Each index can be enclosed in square brackets; for example, y[1][3]
- An index can be an AEL sequence; for example, y[1::3]

When individual elements are accessed, they are treated as any other scalar variable. That is they can be assigned, printed, parameters, or simply accessed. Note that the indices begin with 0.

Example:

```
// x's 2nd element is modified to a new value
x[1] = 10-10i;
// an element in y is printed, the value printed is "4"
fprintf(stderr, "%s",identify_value(y[1,1]));
// an element in y is changed from "7" to "50"
y[3][0] = 50;
// this loop produces the sum of all elements of x
sum = 0;
for ( i=0; i<4; i++)
sum += x\[i];
```

Looping

It is possible to loop with for and while statements, like in C. The first three examples below will print out 0 1 2 3 4 5 6 7 8 9. The 4th example prints out 1 2 3 4 6 7 8 9 10.

For Loop Example

```
decl i;
for (i=0; i<10; i++)
  fprintf(stderr,"%i ",i);
```

While Loop Example

```
decl stop, cnt;
stop = FALSE;
cnt = 0;
while (!stop)
{
  fprintf(stderr, "%i ",cnt++);
  if (cnt == 10)
    stop = TRUE;
}
```

While Loop Example

```
decl i;
i = 0;
while (TRUE)
{
  fprintf(stderr, "%i ",i++);
  if (i == 10)
    break;
}
```

While Loop with Continue Example

```
decl i;
```

```
i = 0;
while (TRUE)
{
  i++;
  if (i == 5)
    continue;
  fprintf(stderr, "%i ",i);
  if (i == 10)
    break;
}
```

AEL Array Functions (Used in AEL Scripts)

The AEL array functions used in AEL scripts are:
offset_array() (ael), *resize_array()* (ael), *array_size()* (ael), *array_type()* (ael), *array_lowerBound()* (ael), *array_upperBound()* (ael), and *convert_array()* (ael).

Writing, Loading and Testing AEL Functions

For component and artwork definitions, you can refer to the AEL files supplied with the program. These files, which define the entire element sets for each simulator, contain hundreds of examples. The files are located in the installation directory for your application:

$HPEESOF_DIR/circuit/ael (for Analog/Rf applications)
$HPEESOF_DIR/adsptolemy/ael (for Signal Processing applications)
$HPEESOF_DIR/de/ael (for the Design Environment, all applications)

You can use any text editor to write AEL files.

Note
There is a special procedure to install an AEL file so that the functions defined in the file can be used in a circuit simulation. Refer to *Introduction to Measurement Expressions* (expmeas) for more information.

Playing an AEL Macro

In the ADS Main window, you can replay an AEL macro using the Tools > Playback Macro command. You can execute AEL statements interactively by typing the statement in the Tools > Command Line dialog box. For example, the first value in a list could be examined by typing:

```
fputs(stderr, car(list(1,2,3)));
```

The value *1* would be displayed in stderr.

Loading AEL Files

You can load AEL files by executing the *load()* command, either from the Command Line

dialog box or as a statement in an AEL file. For example:

```
load("myfunc.ael");
```

Any number of functions can be declared in a single AEL file and any function in the loaded file can then be executed by entering the function name and parameters. For example:

```
testit(1,2, "myfunc");
```

You can load AEL files automatically when you start the program in one of these ways:

- By adding the name of the file containing the functions to one of the configuration variables LOCAL_AEL or USER_AEL and specifying the path to the file in the AEL_PATH variable. LOCAL_AEL is a workspace-specific variable that is reloaded every time a workspace is loaded (once per workspace). USER_AEL is a user-specific variable that is loaded once per program session (once per user). For details on configuring these variables, see *Customizing the ADS Environment* (custom).
- By using the *-m* option; for example, *ads -m startup.ael*.
- By placing the file in the networks subdirectory of a workspace, which automatically loads the file when opening the workspace.

When files are loaded automatically by the program, these files are also compiled automatically. Any file loaded is re-compiled if it is modified or if its compiled version does not exist. Compiling greatly increases the speed in which the function is loaded and it also performs a simplified syntax check. When creating AEL files, you can perform a quick check of their syntax by pre-compiling the AEL file. For details on manually compiling AEL files, refer to the section Using the AEL Compiler.

Testing AEL Functions

You can test most functions by loading the AEL file from the Command Line dialog box. Debugging of the program is limited to adding *fputs()* and *fprintf()* statements in the file to determine function values and code execution. When using *fputs()* and *fprintf()* statements be sure to direct the *fputs()* and *fprintf()* output to standard error, stderr, or to a file, since stdout is used by the workspace design environment to communicate with other programs.

On UNIX, the results of an *fputs()* or *fprintf()* statement sent to stderr will be displayed in the terminal window that the program was started in. On a PC, the program needs to be started as follows:

```
ads -d daemon.log
```

This command opens the program with a window that contains all debugging statements and the results of *fputs()* and *fprintf()* statements sent to stderr. On UNIX, this method creates a logging file called daemon.log. For example, to see something print to a UNIX terminal or PC window:

- In the Main window, choose **Tools** > **Command Line**.
- In the dialog, type:
  ```
  fputs(stderr, "hello world!");
  ```
- Click Apply.

AEL Loading Context

When an AEL file is loaded, functions and variable references are resolved according to the loading context, which specifies the AEL vocabulary to search. New definitions in the AEL file are also added to the context vocabulary.

CmdOp and SimCmd Loading Contexts

The Advanced Design System contains two important contexts, called *SimCmd* and CmdOp. When AEL files are loaded at program start-up or when you enter AEL from the Command Line, the context is CmdOp. When the dialog callbacks are invoked, the context is SimCmd. SimCmd is a superset of CmdOp; that is, when the context is SimCmd, all of the definitions in CmdOp are accessible. But when the context is CmdOp, the definitions in SimCmd cannot be accessed and fail with the message: "could not find global word".

An AEL file that contains new AEL functions or variables can be loaded automatically at program start up. The functions or variables are then accessible from the Command Line and menu commands.

The default loading context is CmdOp, which is also the context in which all AEL definitions in ADS get loaded. The normal context for project's networks .ael files is a workspace-specific context. This context is a superset of *SimCmd*. When the definitions in the file must be accessed while running the program, the context must be CmdOp.

The default loading context can be overwritten by specifying the optional second argument (the context name) for the load() function. This can be specified when loading the AEL file from another startup AEL file, or from the command line. For example, to load an AEL file "test" in the SimCmd, you can use the following load command:

```
load ("test", "SimCmd");
```

The loading context within a particular AEL file can be fixed using the AEL directive #voc(). The argument to #voc() specifies the new loading context. If no argument is specified, the loading context is reset to the default for the file. The loading context fixed using #voc() overrides that set by the load() function. For example, when placed in an AEL file that is loaded in the CmdOp context, #voc(SimCmd) causes the lines in the AEL file following it to be loaded in SimCmd context. When a #voc() directive is encountered, or the end of file is reached, the loading context reverts back to CmdOp.

```
....          //Loading context is CmdOp
#voc(SimCmd) //Loading context is set to SimCmd
....          //AEL code is loaded in SimCmd
#voc()        //Loading context is reset to CmdOp
```

Using the AEL Compiler

When creating AEL functions, you can pre-compile the AEL file by using the AEL compiler, aelcomp, in a stand-alone mode. Any errors in syntax are reported by the compiler and a

compiled atf AEL file is produced. The compiler is invoked automatically by the Design Environment whenever an AEL file is loaded that has been modified since the last compile or is missing its compiled atf counterpart. To run the AEL compiler, enter the following instruction at the command line:

```
aelcomp <input>.ael <output>.atf
```

- The input file and output file names are required and must be specified with their extensions. Normally, the extensions ael and atf are used for the input and output files, respectively.
- Aelcomp has one option, -version, which prints out version info on aelcomp itself. To use this option at the command line, type

  ```
  aelcomp -version
  ```

On UNIX, you must set the shared library path before you run the AEL compiler. Using a bourne shell (sh) or k shell (ksh), enter the following at the command line to set the shared library path:

```
$HPEESOF_DIR=<your complete installation path>
export HPEESOF_DIR
PATH=$HPEESOF_DIR/bin:$PATH
export PATH
. bootscript.sh
```

Creating New Component Definitions

The program learns about new components through component definitions in AEL files. An example of a new component definition is shown below. The example is a definition of a microstrip bend component that has five parameters.

```
/* MBEND */
create_item(              "MBEND",              // name
            "Microstrip Bend (Arbitrary Angle/Miter",       // label
            "BEND",                             // prefix
            0,                                  // attribute
            NULL,                               // priority
            "MBEND",                            // iconName
            standard_dialog,                    // dialogName
            "*",                                // dialogData
            ComponentNetlistFmt,                // netlistFormat
            "MBEND",                            // netlistData
            ComponentAnnotFmt,                  // displayFormat
            "SYM_BendAngle",                    // symbolName
            macro_artwork,                      // artworkType
            "MBendAngle",                       // artworkData
            ITEM_PRIMITIVE_EX,                  // extraAttrib
           create_parm ("Subst", "Substrate instance name", 0, "StringAndReferenceFormSet",
UNITLESS_UNIT, prm("StringAndReference", "\"MSub1\"")),
           create_parm ("W", "Conductor width", PARM_OPTIMIZABLE | PARM_STATISTICAL,
"StdFileFormSet", LENGTH_UNIT, prm("StdForm", "25.0 mil"), list(dm_create_cb(PARM_DEFAULT_VALUE_CB,
"get_default_length_value_cb", "25.0", TRUE))),
           create_parm ("Angle", "Angle of bend", PARM_OPTIMIZABLE | PARM_STATISTICAL,
"StdFileFormSet", ANGLE_UNIT, prm("StdForm", "90")),
           create_parm ("M", "Miter fraction", PARM_OPTIMIZABLE | PARM_STATISTICAL,
"StdFileFormSet", UNITLESS_UNIT, prm("StdForm", "0.6")),
           create_parm ("Temp", "Physical temperature", PARM_NO_DISPLAY PARM_OPTIMIZABLE |
PARM_STATISTICAL, "StdFileFormSet", TEMPERATURE_UNIT, prm("StdForm", "27.0")));
```

New Component Definition Example

The example definition uses the AEL functions *create_item()* and *create_parm()*.

The *create_item()* function creates a component definition with a specific name, associated parameters and other characteristics.

The *create_parm()* function creates and returns a parameter definition and is used to create parameter definitions for *create_item()*, where the *create_parm()* function return values become pass parameters for *create_item()*, with each use of *create_parm()* defining a parameter for the component.

The function *prm()* creates and returns a parameter value and is used to generate a default value where the form of the value is something other than a normal number or string. The forms of a parameter value are explained more completely in the section Creating and Using Custom Forms.

For details on these functions, see *Component Definition Functions* (ael).

Form Sets

When components and commands are defined with their associated parameters, the allowable forms for each parameter are specified through a named form set. A form set describes the *types* of values a parameter can take.

A form set is composed of one or more forms, each representing a value option for a parameter. Several forms are associated together in sets for use with a parameter, where a parameter value can assume any of the forms in the set. To be valid, the parameter definitions must specify the name of a defined form set. In some cases, this can require the definition of a form set of one form.

For example, the Parameter Entry Mode menu in the Component Parameters dialog box is configured by the form set for the selected parameter. The name of the form is used to configure the standard Component Parameters dialog box.

Forms and form sets are all defined using these AEL functions:

```
create_form_set()
create_text_form()
create_constant_form()
create_compound_form()
```

The following example defines a form set:

```
create_form_set("StdFileFormSet", "StdForm", "FileBasedForm");
```

Certain primitive forms are pre-defined and do not need to be created explicitly. Some of the pre-defined forms are:

- StdFormSet
- StdFileFormSet
- ReadFileFormSet
- StringAndReferenceFormSet
- SingleTextLineFormSet
- SingleTextLineIntegerFormSet
- SingleTextLineDoubleFormSet

The files *stdforms.ael*, *stditems.ael*, and *pde_gemini.ael* contain many of the standard form and component definitions used by the program. These files can be found in the *<$HPEESOF_DIR>/de/ael* directory.

For details on forms and form set functions, see *Component Definition Functions* (ael).

Creating and Using Custom Forms

In some cases a non-standard form is required to support unique syntax or to limit the options presented in a dialog box. Some of the necessary forms are unique to a simulator or to a particular design type. The program provides the ability to define new forms and their characteristics.

The characteristics of a form govern its handling in a Component Parameters dialog box, on-screen editing in the schematic and in the netlist. The form contains the instructions for formatting an instance parameter value for use in the display annotation and in the netlist. It also contains information to generate a list of value options for selection from a list in a Component Parameters dialog box. Also, the form can check the typed input for validity using the validate function.

Forms come in several classes that are provided with the program. A new form can be defined using one of these classes, where the new definition gives the form characteristics specific to the simulator/syntax requirements. The following built-in classes are characterized.

Constant Class

Used where the value is a constant and cannot be changed. Examples of this class are the forms for the SinE parameter of the LPF_RaisedCos component, where the allowed values are YES and NO. Two forms would be created, one for each of the possible values, and a form set would be created that referenced both forms. The definition for the SquareRoot parameter of the LPF_RaisedCosineTimed component would reference the form set defined. Selection would be made between the two allowed forms from an option menu in a dialog box, with no typing allowed.

Compound Class

Used to define a hierarchical parameter, where the parameter value is defined by several sub parameter values combined. Examples of this class are VarFormStdForm and DACForm. For this class, parameters are defined for the form using create_parm(), each of which references a form set. Since this form is indirectly referenced through another parameter definition, it creates a recursive definition (only one level deep). The parameter's value is composed of the multiple fields set in the form's parameter values.

Text Class

Used where a text value is required, but an arbitrary string is not appropriate. This occurs for parameters where a list of possible values should be generated at run time and presented in a dialog box, or where additional verification must be performed on the typed response in the dialog box. An example of this class is the msub parameter of the mlin element, where the text required is the instance name of an msub element. Optionally, a dialog box would provide a list of msub components available, and the selection could be made from the list or a name could be entered. In this case, checking would not be necessary, but a check function could be applied to verify that the entry was indeed a valid msub name. Although not supported by the user interface anymore, most parameters in the program are using this class of form.

Two rules should be applied to the naming of a form:

- All form names should be unique.
- If the form needs to be identified in the netlist, the name of the form should be the same as the identification string desired in the netlist.

The AEL functions that create new forms are *create_constant_form()*, *create_compound_form()*, and *create_text_form()*.

The forms created by create_constant_form() represent a fixed value, which is selected from a list of the possible values (normally from an option box) but does not edit

textually. An example of this might be the definition of the Yes form:

```
create_constant_form("Yes", "YES", 0, "yes", "yes");
```

The forms created by create_compound_form() represent values that contain one or more parameters, some of which represent a value more complicated than a string. Each parameter has its own set of forms for the values it can accept. For example:

```
create_compound_form("LinearStart", "LinearStart", 0,
"Start=%0s, Stop=%1s Step=%2s Lin=%3s",
"Start=%0s, Stop=%1s Step=%2s Lin=%3s"

create_parm("Start", "Start Value", 0 "FSTextForms",
FREQUENCY_UNIT, prm("FSTextForm", "0")),
create_parm("Stop", "Stop Value", 0 "FSTextForms",
FREQUENCY_UNIT, prm("FSTextForm", "0")),

create_parm("Step", "Step Value", 0 "FSTextForms",
FREQUENCY_UNIT, prm("FSTextForm", "0")),

create_parm("Lin", "Points in a Linear Sweep", 0 "FSTextForms",
UNITLESS_UNIT, prm("FSTextForm", "0")));
```

The forms created by create_text_form() represent values that accept a string, but not just any string will do. A dialog box to present options for the string and additional checking of the typed string can be specified. A data string can be provided for use in the option list generation and value verification. An example might be the definition of the SingleTextLineIntegerForm form, used to represent a value that is an integer.

The value stdforms_validate_integer specifies a function that checks a value typed in by the user. If the value is an integer, the function returns 1 and the user-value is acceptable. If the value is not an integer, the function returns 0 and the user-value is not acceptable.

```
create_text_form("SingleTextLineInteger", "Integer Value",
"SingleTextLine", 0, "%v", "%v", NULL, stdforms_validate_integer,
NULL);
```

For details on these functions, see *Component Definition Functions* (ael).

Format Strings

Format strings are used by a number of functions to construct a single string out of a number of different components. They are modeled on the basic printf capability in the C programming language, but include a large number of additional conversion specifiers. These strings provide a flexible way to control the formatting of complex value strings. Format strings control the way component parameters are displayed on the schematic. A standard format string is used by most components. A standard format string, called *ComponentNetlistFmt*, is used for the formatting of a component's netlist line.

The basic form of these strings is a pair of quotation marks enclosing any number of strings and conversion specifiers. Conversion specifiers are preceded by the percent sign (%) and take the following form:

%nc

where n represents an optional integer value and c represents the instruction code. The default value of the optional integer value is 0.

Available Specifiers

The available specifiers are Loops, Conditionals, and Operators. The available specifiers and what each does (with optional integer after % and before the single letter instruction code) are listed in the following sections.

Loops

- %b: Begin parameter iteration loop, which shifts through each parameter in the list of parameters of a component instance or a compound form. The optional integer value (the *n* in %nc) indicates the initial parameter index, which sets the value of j counter. The j counter is incremented at the end of each parameter iteration.
- %r: Begin repeated parameter iteration loop, which checks if the current parameter (j counter) is a repeated parameter and iterates through each value in the list of values for the repeated parameter. The optional integer value (the *n* in %nc) indicates the initial repeat parameter index, which sets the value of i counter. The i counter is incremented at the end of each repeated parameter iteration. The value of a repeated parameter is stored as a list of values, where a normal parameter has a single value. For details on how to define a repeated parameter, refer to *create_parm()* (ael). An example of a component with repeated parameters is SDD1P.
- %h: Begin hierarchical parameter iteration loop for compound form, where the parameter list is retrieved from the current parameter's compound form. The optional integer value (the *n* in %nc) indicates the initial parameter index, which sets the value of j counter for the hierarchical loop. The j counter for the hierarchical loop is incremented at the end of each hierarchical parameter iteration. If this is in a nested loop, the j counter value for the original parameter iteration loop (%b) is stored and retrieved as a top counter value.
- %#: Begin node iteration loop, which shifts through each node in the list of nodes of a component instance. The optional integer value (the *n* in %nc) indicates the initial node index, which sets the value of l counter. The l counter is incremented at the end of each node iteration.
- %i: Transfer the sum of repeated parameter iteration index (current i counter value) and the optional integer value (the *n* in %nc), to the output string.
- %j: Transfer the sum of parameter iteration loop index (current j counter value) and the optional integer value (the *n* in %nc) to the output string.
- %l: Transfer the sum of node iteration loop index (current l counter value) and the optional integer value (the *n* in %nc) to the output string.
- %e: End the most recent iteration loop. The iteration loop ends when the iteration counter for the loop is the same as the optional integer value (the *n* in %nc), or when the end of the list being iterated on is reached. The appropriate iteration counter is modified depending on the type of loop in progress. In addition, the format string is modified to facilitate the next iteration of the current loop or to end the current loop. Loops can be nested.

Conditionals

%F Set the fieldIndx so the next %29? test will apply to that field, which is a piece of the value for a compound form. The optional integer value (the *n* in %nc) indicates the field number. For example, %2F ... %29? will test if the third field in a compound form is an empty value. After %29? the fieldIndx is reset to -1.

%? Begin a conditional test, where the optional integer value (the *n* in %nc) indicates the type of test performed:

0 = TRUE if inside a repeated parameter loop (%r)
1 = TRUE if form of current parameter value is tunable
2 = TRUE if form of current parameter value has sub parameters (compound form)
3 = TRUE if form of current parameter value has special attributes
5 = TRUE if form of current parameter value is an equation
6 = TRUE if nominal value of current parameter is non-zero
8 = TRUE if parameter is netlistable
20 = TRUE if design is a network
24 = TRUE if form of current parameter value is binary
25 = TRUE if form of current parameter value is octal
26 = TRUE if form of current parameter value is hexadecimal
27 = TRUE if form of current parameter value is numeric
28 = TRUE if form of current parameter value is string
29 = TRUE if current parameter has no value
30 = TRUE if PARM_RIGHT_HAND_ONLY attribute is set for the parameter.
31 = TRUE if node name for the specified node index (l counter) exists in the node name list.
32 = TRUE if external port name for the specified external port index (k counter) exists in the external port name list.
37 = TRUE if the number of parameters for the instance is not zero.
38 = TRUE if PARM_NO_PLOT attribute is set for the parameter.
39 = TRUE if the global parameter AllParams is set for the instance.
41 = TRUE if INST_SCOPE_LOCAL attribute is set for the instance.
42 = TRUE if INST_SCOPE_NESTED attribute is set for the instance.
43 = TRUE if INST_SCOPE_GLOBAL attribute is set for the instance.
44 = TRUE if the node is a ground node (node number = 0)
45 = TRUE if the value string starts with ";"

The format string following a test is optionally interpreted depending on a TRUE result of the test. The %: and %; delimit the end of the conditional part of the format string.

%: End TRUE branch of conditional format begun with %? And begin FALSE branch.

%; End of conditional format begun with %?

Operators

- %g: Transfer the current design name to the output string. The optional integer value (the *n* in %nc) is ignored.
- %n: Transfer the component name to the output string. The optional integer value

(the *n* in %nc) is ignored.

- %t: Transfer the instance name (formerly instance tag) to the output string. The optional integer value (the *n* in %nc) is ignored.
- %d: Transfer the data string to the output string. Data string can be component netlist data (*netlistData* argument in *create_item()*) or an array of form data strings (*dataValue* argument in *create_text_form()*). The optional integer value (the *n* in %nc) specifies the index into the array of form data. Index value other than 0 is invalid for netlist data. The default index value is 0.
- %k: Transfer the parameter name to the output string. The optional integer value (the *n* in %nc) indicates the desired parameter index relative to the current parameter index in the parameter list, with 0 for the current parameter. The current parameter index is determined by the value of j counter.
- %p: Transfer the parameter value, formatted using the netlistFormat string from the parameter's form definition, to the output string. This works like %s except that the netlistFormat is used. The optional integer value (the *n* in %nc) indicates the desired parameter index relative to the current parameter index in the parameter list, with 0 for the current parameter. The current parameter index is determined by the value of j counter.
- %s: Transfer the parameter value, formatted using the displayFormat string from the parameter's form definition, to the output string. This works like %p except that the displayFormat is used. The optional integer value (the *n* in %nc) indicates the desired parameter index relative to the current parameter index in the parameter list, with 0 for the current parameter. The current parameter index is determined by the value of j counter.
- %v: Transfer the current parameter value to the output string. The optional integer value (the *n* in %nc) indicates the field index for forms having more than one field, with 0 for the first field (see example later). For constant forms, special control over formatting may be exercised by calling the dataFunction specified in *create_text_form()*. This format is used only in netlistFormat string or displayFormat string of form definitions.
- %o: Transfer the nominal value of a parameter that can be optimized. The optional integer value (the *n* in %nc) indicates the desired parameter index relative to the current parameter index in the parameter list, with 0 for the current parameter. The current parameter index is determined by the value of j counter.
- %q: Transfer the nominal value, evaluating all variable references, of a parameter that can be optimized. This works like %o except that variable references are evaluated. The optional integer value (the *n* in %nc) indicates the desired parameter index relative to the current parameter index in the parameter list, with 0 for the current parameter. The current parameter index is determined by the value of j counter.
- %f: Transfer the parameter value form name to the output string. The optional integer value (the *n* in %nc) is ignored. (Use after %p or %s operators).
- %u: Transfer the parameter value unit name to the output string. The optional integer value (the *n* in %nc) indicates the desired parameter index relative to the current parameter index in the parameter list, with 0 for the current parameter. The current parameter index is determined by the value of j counter.
- %z: Transfer unit conversion factors, standard usage of optional integer value (the *n* in %nc):

  0 = FREQUENCY_UNIT to MKS
  1 = RESISTANCE_UNIT to MKS
  2 = CONDUCTANCE_UNIT to MKS
  3 = INDUCTANCE_UNIT to MKS
  4 = CAPACITANCE_UNIT to MKS
  5 = LENGTH_UNIT to MKS

6 = TIME_UNIT to MKS
7 = ANGLE_UNIT to MKS
8 = POWER_UNIT to MKS
9 = VOLTAGE_UNIT to MKS
10 = CURRENT_UNIT to MKS
11 = DISTANCE_UNIT to MKS
12 = UNIT length to layout database
13 = User length to layout database
20 = MKS to FREQUENCY_UNIT
21 = MKS to RESISTANCE_UNIT
22 = MKS to CONDUCTANCE_UNIT
23 = MKS to INDUCTANCE_UNIT
24 = MKS to CAPACITANCE_UNIT
25 = MKS to LENGTH_UNIT
26 = MKS to TIME_UNIT
27 = MKS to ANGLE_UNIT
28 = MKS to POWER_UNIT
29 = MKS to VOLTAGE_UNIT
30 = MKS to CURRENT_UNIT
31 = MKS to DISTANCE_UNIT
32 = Layout database to UNIT length
33 = Layout database to User length

- %c: Transfer the node number to the output string. The optional integer value (the *n* in %nc) indicates the desired node index relative to the current node index in the node number list for the instance, with 0 for the current node number. The current node index is determined by the value of l counter.
- %C: Transfer the node name to the output string. The optional integer value (the *n* in %nc) indicates the desired node index relative to the current node index in the node name list for the instance, with 0 for the current node name. The current node index is determined by the value of l counter.
- %m: Transfer the external port number to the output string. The optional integer value (the *n* in %nc) indicates the desired external port index relative to the current external port index in the external port number list for the design, with 0 for the current external port number. The current external port index is determined by the value of k counter.
- %M: Transfer the external port name to the output string. The optional integer value (the *n* in %nc) indicates the desired external port index relative to the current external port index in the external port name list for the design, with 0 for the current external port name. The current external port index is determined by the value of k counter.
- %%: Transfer a single % character to the output string, integer value (the *n* in %nc) ignored.

Netlist Format String Examples

These examples illustrate the use of netlist format string in components and forms.

Example 1: Simple format string

A Compound form netlist format string "Start=%0v Stop=%1v Step=%2v"

Can be interpreted as follows:

Transfer the string "Start=" to output string (Start=)
Transfer the first field in parameter value to output string (%0v).
Transfer the string "Stop=" to output string (Stop=)
Transfer the second field in parameter value to output string (%1v).
Transfer the string "Step=" to output string (Step=)
Transfer the third field in parameter value to output string (%2v).

An example output string with this format string, for a compound form component parameter value list of 10, 20, 2 is:

"Start=10 Stop=20 Step=2"

Example 2: Simple iteration loop and conditionals

A node iteration loop format string %# %44?0%:%31?%C%:_net%c%;%;%e

Can be interpreted as follows:

```
Start node iteration loop at node index 0 (%#)
(For each node iteration perform all of the following)
If the node is a ground node (%44?)
  Transfer "0" to output string ( 0 )
Else ( %: )
  If there is a node name entry for the node index ( %31? )
      Transfer the node name to output string ( %C )
  Else ( %: )
      Transfer the string "_net" and the node number to the output string (_net%c)
  End condition ( %; )
End condition ( %; )
End loop ( %e )
```

An example output string with this format string for a component with four nodes, one connected to the ground and another named "Vcc" is:

"0 _net2 Vcc _net4"

Example 3: Nested iteration loops

A simplified nested parameter iteration loop(%b) and repeated parameter loop(%r) format string " %b%r%k%?[%1i]%;=%p %e%e"

Can be interpreted as follows:

```
Start parameter iteration loop at parameter index 0 ( %b )
(For each parameter iteration, perform all of the following)
Start a repeated parameter iteration loop at repeated parameter index
0 ( %r )
(For each repeated parameter iteration perform all of the following)
  Transfer the parameter name to the output string ( %k )
  If the parameter is a repeated parameter ( %? )
       Transfer "[" string to the output string ( [ )
       Transfer the sum of 1 and the repeated parameter index to
```

```
the output string (%1i)
       Transfer "]" string to the output string ( ] )
  End condition ( %; )
  Transfer the string "=" to output string ( = )
  Transfer the parameter value to the output string ( %p )
End loop ( %e )
End loop ( %e )
```

An example output string with this format string for a component with a repeated parameter S and another parameter Num is:

"S[1]=0 S[2]=1 Num=2"

Assigning Components to Groups

A new component must be assigned to a library group or palette group for placement. Library group and palette groups are associated with a particular type of design. The group definitions for each design type is distinct. Most likely, you will assign the new components you create to groups for the current network design, which are displayed in the schematic and layout windows in the program.

The functions used to assign components to groups are *set_design_type()*, *library_group()*, and *de_define_palette_group()*. For details on these functions, see *Component Definition Functions* (ael).

An example of assignment of an component to a group is:

```
set_design_type(analogRFnet);
library_group("mstrip", "Microstrip components", "MLIN", "MBEND");
```

Two AEL functions are demonstrated in the example. The example assigns the components MLIN and MBEND to the library group *mstrip* and the library group is associated with analog/RF network designs.

Designing Component Schematic Symbols

Beginning with ADS 2011, the symbol for a component must be stored in a symbol view in the component's cell. The symbolName parameter in create_ite is obsolete. Symbols from other cells cannot be referenced. Instead, the sybol should be copied into the new component. See *Working with Symbols* (usrguide) for more information.

Designing Component Palette Buttons

When new components are defined, a bitmap name can be specified to represent the component button in a palette. If the named icon does not exist, the component name is displayed on the button. To add a customized button to the component palette, a graphic to represent the component needs to be created, saved in the correct form, and placed in the appropriate directory.

On UNIX systems, any visual tool such as Icon Editor or xv will suffice and the bitmaps are saved as X bitmap files. On a PC, they can be created using any visual tool such as Paint or Microsoft Visual C++ and are saved in PC bitmap format. Regardless of the platform type or the tool used to create them, all bitmaps must have the following attributes:

- Size of 32 x 32 pixels
- 16 colors
- .bmp filename extension

The bitmap search path is defined at runtime by ADS via the environment variables XBMLANGPATH for UNIX platforms and WBMLANGPATH for PCs. Users can place bitmaps in predefined directories in the search path where they will be automatically found and displayed.
On UNIX platforms, the predefined directories are:

$HOME/hpeesof/custom/bitmaps - for individual user customization
$HPEESOF_DIR/custom/bitmaps - for site-wide customization
$HOME/hpeesof/circuit/bitmaps - for analog/RF customization
$HOME/hpeesof/de/bitmaps - for general PDE customization
$HOME/hpeesof/adsptolemy/bitmaps - for DSP related customization

On PCs the predefined directories are:

%HOME%\hpeesof\custom\bitmaps - for individual user customization
%HPEESOF_DIR%\custom\bitmaps - for site-wide customization
%HOME%\hpeesof\circuit\bitmaps - for analog/RF customization
%HOME%\hpeesof\de/bitmaps - for general PDE customization
%HOME%\hpeesof\adsptolemy\bitmaps - for DSP related customization

Developing Measured Component Libraries

A set of linear S-parameter measurement files can represent a library of parts for the simulator. This is accomplished by representing these as a library of parts that can be placed in a schematic. The complete set of S-parameter files would reside together in a directory. For example, the directory *<home directory>/hpeesof/myparts/parts/circuit* would hold a set of S-parameter files. In the same directory an AEL definition file, possibly named *circuitcomponents*, defines the components for the workspace environment, one component for each part, and the library group. The name of the directory is added to the AEL path for the SIMULATOR_AEL, directing the program to search and read the definition file when attaching to a workspace. The next time program is run, the new group name appears in the list of libraries in the schematic and layout windows. Any parts library would be handled in the same way.

The environment variable value for CIRCUIT_AEL is:

HOME_CIRCUIT_AEL={$HOME}/hpeesof/{%PROJECT1}/ael
where: PROJECT1=circuit and $HOME is your home directory on a UNIX platform.

Circuit components are defined in the file gemini.ael.

The circuit example, shown below, creates a library of S-parameter file components. The AEL definition file and the S-parameter files can be stored in the same directory and the

directory is added to the CIRCUIT_AEL path. First, a message that the library is being loaded is displayed on the terminal window. Then a form set is created specifying that the value of the S-parameter component will be an S-parameter file name. Finally, a component definition is created for each S-parameter file. The standard EEFET1 symbol is used for the schematic symbol artwork. The component is presented to the simulator as an S2P element with the name of the S-parameter file as the value of the FILE parameter. The component description field is used to display the *Vds* and *Ids* values for the S-parameter file. This line, which clearly identifies these values, appears in the library list dialog box. The contents of the *linear.ael* file are shown below and the contents of the _d_0777.s2p_ file are shown in Measured Component Example Database.

```
fputs(stderr,"Reading S-Parameter Library: sp_lin_40");

create_file_name_form_set("sp_lin_40","s-parameter files", "SP_LIN_40", "s2p" );

create_item( "sp40_lin_d_0777", "Vds=3.5 Ids=15", "S",NULL, NULL, NULL, standard_dialog, "d_0777
fet", special_netlist, "ELEMENT S2P", standard_symbol, "EEFET1", no_artwork, NULL, create_parm(
"File", "Tdata model file", 2, "sp_lin_40", -2, prm( "sp_lin_40" , "d_0777" ) ), create_parm( "TEMP",
"TEMP name", 0, "tempname", -2, prm( "def*" )));

create_item( "sp40_lin_d_1503", "Vds=3.5 Ids=15", "S", NULL, NULL, NULL, standard_dialog, "d_1503
fet", special_netlist, "ELEMENT S2P", standard_symbol, "EEFET1", no_artwork, NULL, create_parm(
"File", "Tdata model file", 2,"sp_lin_40", -2, prm( "sp_lin_40", "d_1503")), create_parm( "TEMP",
"TEMP name", 0, "tempname", -2, prm("def*")));

create_item( "sp40_lin_d_2501", "Vds=3.5 Ids=15", "S", NULL, NULL, NULL, standard_dialog, "d_2501
fet", special_netlist, "ELEMENT S2P", standard_symbol, "EEFET1", no_artwork, NULL, create_parm(
"File", "Tdata model file", 2, "sp_lin_40", -2, prm( "sp_lin_40", "d_2501")), create_parm(
"TEMP","TEMP name", 0, "tempname", -2, prm("def*")));

create_item( "sp40_lin_d_2502", "Vds=3.5 Ids=15", "S", NULL, NULL, NULL, standard_dialog, "d_2502
fet", special_netlist, "ELEMENT S2P", standard_symbol, "EEFET1", no_artwork, NULL,
create_parm("File","Tdata model file", 2, "sp_lin_40", -2, prm( "sp_lin_40", "d_2502")),
create_parm("TEMP", "TEMP name", 0, "tempname", -2, prm("def*")));

 library_group("sp_lin_40","S-parameters",
"sp40_lin_d_0777",
"sp40_lin_d_1503",
"sp40_lin_d_2501",
"sp40_lin_d_2502"
```

Measured Component Library Example

```
D-0777 (CHIP) ULTRA LOW NOISE FET
! VDS= 3.5V, IDS= 15mA
! NOTE: S-PARAMETERS INCLUDE 0.7 mil BOND WIRES
# GHZ S MA R 50
! FREQ      S11           S21           S12           S22
! GHZ    MAG    ANG     MAG    ANG     MAG    ANG    MAG    ANG
  2     .97     -22     3.40   161     .03    77     .64    -13
  4     .93     -41     3.11   145     .06    66     .62    -21
  6     .88     -55     2.84   133     .08    60     .59    -26
  8     .82     -67     2.66   122     .09    54     .56    -32
  10    .75     -82     2.55   110     .10    48     .51    -38
  12    .68     -100    2.42    97     .10    40     .45    -48
  14    .61     -118    2.22    84     .10    38     .43    -59
  16    .61     -141    2.02    71     .10    35     .43    -75
  18    .61     -157    1.74    59     .10    32     .43    -84
```

Developing Discrete Valued Part Libraries

The Advanced System Design simulators have the capability of using discrete measurement data to simulate manufactured components with discrete values. Discrete measured data is provided in a Microwave Data Interchange Format (MDIF) file. The file organization allows the description of named sets of data, where each set represents a discrete value of a part. You can create a component definition in AEL that allows you to pick only valid discrete values for a part. For example, you can have measured data for a capacitor in the form of capacitance and series resistance for several nominal values of the capacitor, such as 20, 22, 24, 27, and 30 pf. These values are organized into a MDIF file, a subnetwork is created to model the physical part, and an AEL component definition is created that allows only the discrete values to be chosen for analysis in the subnetwork. Also, the simulator's discrete value optimizer uses only the valid discrete values.

To create a set of discrete values components for simulation:

- Develop a model for the part and measure the model parameters for the desired nominal values of the part.
- Format the measured values into a MDIF file.
- Build a parametric subnetwork for the part model, where one of the pass parameters represents the nominal value.
- Create the AEL component definition for use of the parametric subnetwork as a discrete valued part.

Developing Discrete Valued Parts MDIF File

The values available for discrete valued parts are defined in a file using a subset of the Microwave Data Interchange Format (MDIF). This file format is used to define a variety of data for use by the simulators. The file to define discrete values contains a table of values, which are accessed by row index and column name. An example of a simple MDIF file format for discrete valued parts is:

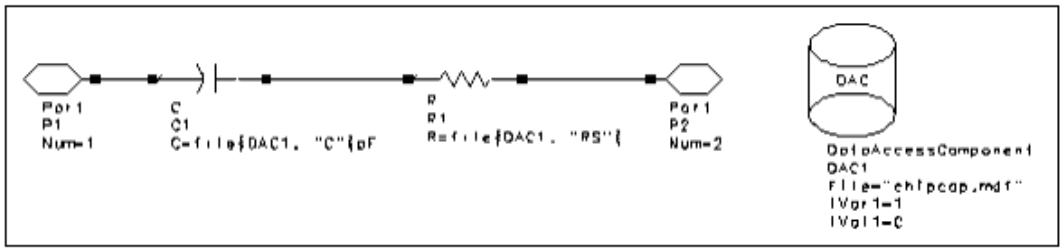
```
REM This is an example file for discrete capacitors
BEGIN DSCRDATA
% PART C RS
20pF 20 2.1
22pF 22 1.9
24pF 24 1.75
27pF 27 1.56
30pF 30 1.4
END DSCRDATA
```

This example has five sets of values, where each set consists of two values. The REM line at the top of the file is an optional comment. The BEGIN DSCRDATA line marks the beginning of the data table and gives the table the required name DSCRDATA. The next line, beginning with a percent sign, names the columns in the data table. The name for the first column (PART) is not significant. This column contains the names for the rows used by the program when accessing a particular set of values. The names of the

remaining columns are used to access the individual data values for a particular row. There must be at least one column of data values beyond the first column. The lines following define sets of data values, one row at a time, until the END DSCRDATA line ends the table. Only one data table is allowed in a file. For use by the local workspace, the MDIF file should be placed in the `data` subdirectory.

Designing a Discrete Valued Parts Parametric Subnetwork

The model for the discrete valued part is contained in a parametric subnetwork. The following figure shows the schematic for the discrete valued parts parametric subnetwork.



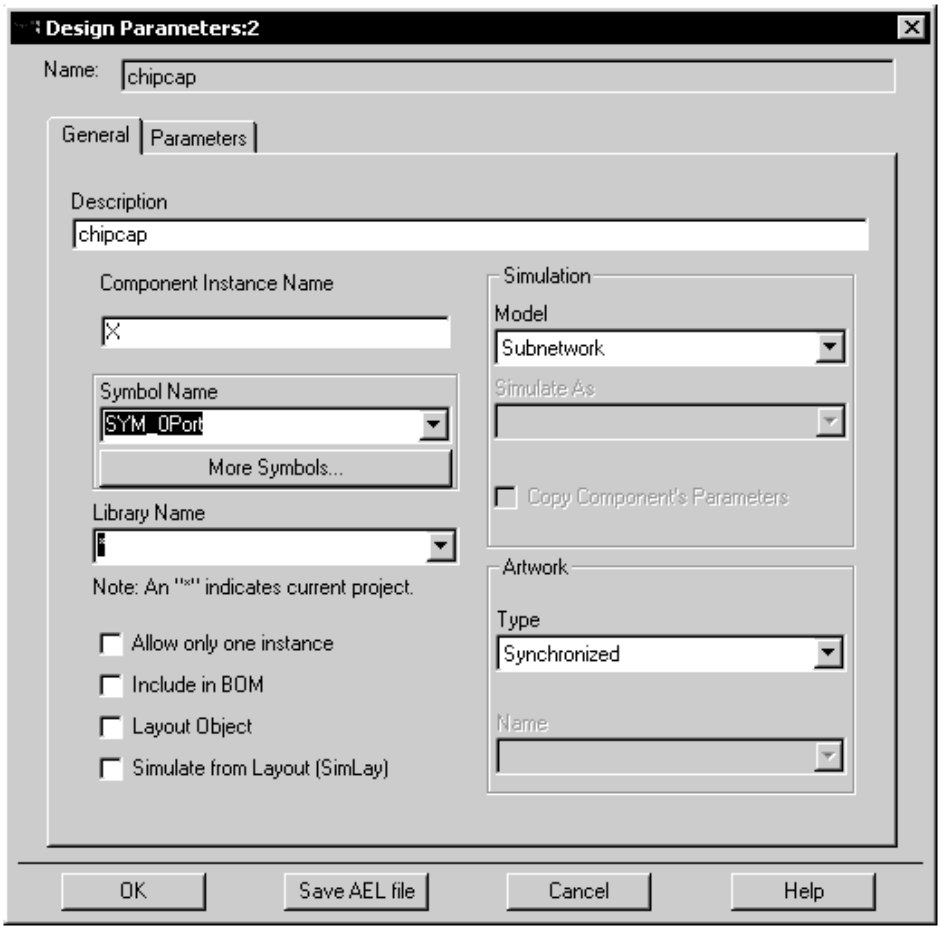
Discrete Valued Parts Parametric Subnetwork Schematic

Model components are placed and connected in the network in the usual way, except for the addition of references to the discrete data file. This involves adding a DataAccessComponent (DAC) and setting certain model component parameter values to refer to the DAC. One parameter for the subnetwork is used to represent the nominal value of the part.

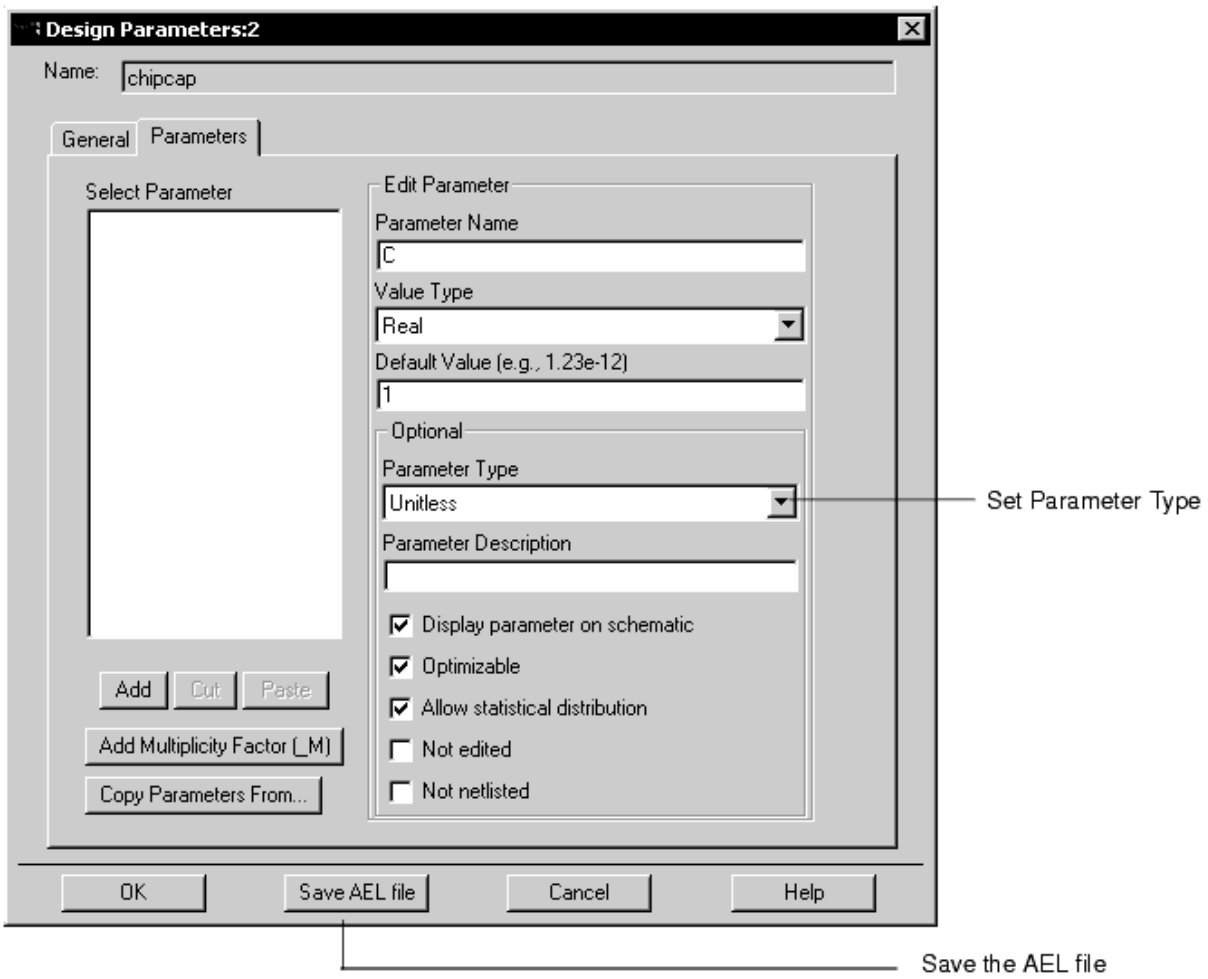
Note
The units of the components placed in the schematic are assumed to be in SI units. To enter the value of capacitance in pF, you must edit the unit in the schematic and add pF to "file{DAC1, "C"}", as shown in the previous figure.

The basic requirements for designing a discrete valued parts parametric subnetwork are to define the subnetwork component, the symbol, and the subnetwork parameter for use as the nominal value.

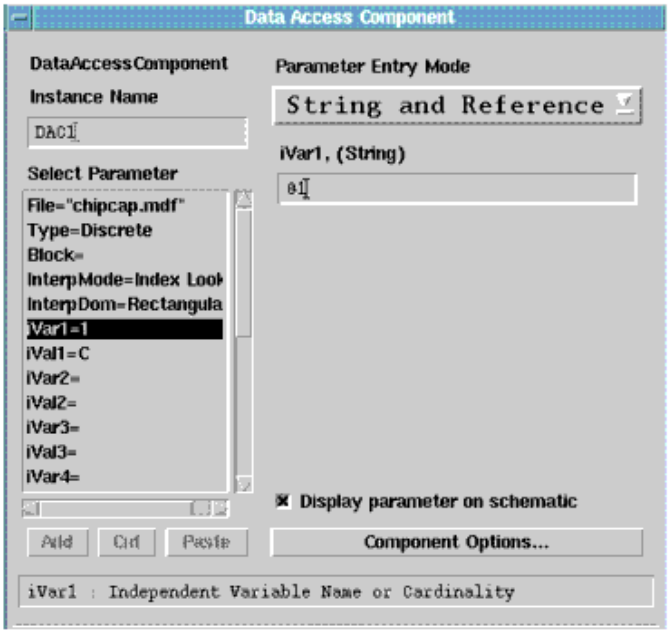
1. Create the subnetwork by placing and connecting the model components and the ports, as shown in in the previous figure. Save the design. In this example, chipcap is used as the design name.
2. Select **File** > **Design/Parameters** and click **General**.
   - To save the new item in an existing custom library group, select the group from the **Library Name** list.
   - To save the new item in a new custom library group, select the Library Name field and enter the name. The default Library Name is "*", which indicates the current project. In this example, **Sample Discrete Library Group** is entered as the custom library.



3. Select **Parameters**.
   - Select the Parameter Name field and enter the name. In this example C is entered as a parameter.
   - Set Parameter Type to **Unitless**.
   - The Value Type and Default Value are not important.
   - Save the AEL file by clicking **Save AEL File**. Then close the dialog by clicking OK.



4. In the schematic window, place a **DataAccessComponent (DAC)** from the Data Items library. Open the Edit Component dialog by double-clicking on the DataAccessComponent (DAC) instance.
   - Set the **File** parameter to the name of the MDIF data file. Include the *mdf* extension.
   - Confirm that **File Type** = **Discrete,** to set the parameter values for certain components of the model which reference individual values from the data file. When this type is used, the instance name of the DataAccessComponent (DAC) and the name of the data column provide the link to the data file.
   - Confirm that **InterpMode=Index Lookup** and **InterpDom=Rectangular**.
   - Set the index parameter (**iVal1**) to the subnetwork parameter used for the nominal value. When entering the Independent Variable Name or Cardinality value of iVar1, enter as @1 so that it will not appear in quotes.



5. Open the Edit Component dialogs for components of the model and reference individual values from the data file, by selecting File Based Parameter Entry Mode and entering the appropriate Dependent Parameter Name, as shown in Discrete Valued Parts Parametric Subnetwork Schematic.
6. Save the design.

After the parametric subnetwork has been created and saved, you must modify the AEL definition to make the part usable. When any design is saved, an AEL definition is created for the subnetwork so that it can be placed in another design. For parametric subnetworks that represent discrete parts, the AEL definition created above does not allow the part to be handled as a discrete valued part. You must modify the AEL definition each time the parameters for the network are modified. The modification is described in the section Discrete Valued Parts Required AEL Definitions.

Discrete Valued Parts Required AEL Definitions

The AEL component definition created when saving a parametric subnetwork must be modified in order for the network to be used as a discrete valued part. The following figure shows the modified AEL component definition for the subnetwork design in Designing a Discrete Valued Parts Parametric Subnetwork. In the figure, the lines you must modify are highlighted. In addition, you must remove the line:

```
library_group("*", "*", 1, "chipcap");
```

If you do not want the item to appear in the current workspace, but only to appear in the designated library group (in this example, Sample Discrete Library Group).

```
set_simulator_type(-1);
set_design_type(1);

create_constant_form ("chipcap_20pF", "20pF", 68, "0", "20pF");
create_constant_form ("chipcap_22pF", "22pF", 68, "1", "22pF");
create_constant_form ("chipcap_24pF", "24pF", 68, "2", "24pF");
create_constant_form ("chipcap_27pF", "27pF", 68, "3", "27pF");
create_constant_form ("chipcap_30pF", "30pF", 68, "4", "30pF");

create_form_set ("chipcap_values", "chipcap_20pF", "chipcap_22pF", "chipcap_24pF",
"chipcap_27pF", "chipcap_30pF");

create_compound_form ("chipcap_index", "Discrete optimize", "DistLibForm", 68, "%p opt{discrete
%1p to %2p by 1}", "%s,%1s to %2s",
   create_parm ("NOM", "Nominal", PARM_DISCRETE_VALUE, "chipcap_values", UNITLESS_UNIT,
prm("chipcap_20pF")),
         create_parm ("MIN", "Minimum", PARM_DISCRETE_VALUE, "chipcap_values", UNITLESS_UNIT,
prm("chipcap_20pF")),
   create_parm ("MAX", "Maximum", PARM_DISCRETE_VALUE, "chipcap_values", UNITLESS_UNIT,
prm("chipcap_30pF")));

create_form_set ("chipcap", "chipcap_20pF", "chipcap_22pF", "chipcap_24pF", "chipcap_27pF",
"chipcap_30pF", "chipcap_index", "value");

create_item("chipcap", "chipcap", "X", 16, -1, NULL, "Component Parameters", "",
    "%l:%t %f %44?0%:%31?%0%:_net%c%;%;%e %b%r%8?%29?%:%30?%p %:%k%?[%1i]%;=%p %;%;%;%e%e",
    "chipcap", "%t%b%r%38?%:\n%39?all_parm%A%:%30?%s%:%k%?[%1i]%;=%s%;%;%;%e%e%;",
    "SYM_C", 3, NULL, 0,
   create_parm("C", "capacitance", PARM_DISCRETE_VALUE, "chipcap", UNITLESS_UNIT,
prm("chipcap_20pF")));
```

AEL Definition for Discrete Value Component

Supporting User-Defined Simulator Components

The Advanced Design System simulators support the capability for augmenting the built-in simulator element set with new components that you define by writing C language functions that are linked with the simulator object archive to produce a simulator executable.

Note
For details, refer to *Building User-Compiled Analog Models* (modbuild).

After the user-defined modules have been linked to the simulator, you need to create an AEL definition to place and use the element in a schematic. You accomplish this by creating a component for each user-defined element.

In AEL, user-defined components are described in the same way as built-in components. For each parameter of the element, a parameter definition is created using the *create_parm()* function. The parameters are added to the component using *create_item()*, along with the basic component definitions. Finally, the component is specified in a library group, or optionally, assigned to a palette group and given a bitmap.

An important consideration in describing the user-defined component is assuring the correct parameters. The parameters should match in order, name and type, between the AEL component description and those being used in the user-defined module. In particular

the *UserParamType* array definition needs to match the parameters described by *create_parm()*. Also, the netlist string provided in the *create_item()* call needs to produce the proper simulator syntax. Unless the component has unusual parameters, the pre-defined *standard_netlist* string can be used.

Note
The netlist data string for all user-defined components should be either ELEMENT or DATA. ELEMENT should be used for user-defined components with one or more ports/pins, and DATA used for components with none.

In the following example, the standard *U2PA* element is defined. Only the definitions for the user-defined element are described here:

```
LOCAL UserParamType
U2PA[] =
{
 {"R1", REAL_data}, {"R2", REAL_data}, {"R3", REAL_data}
};
LOCAL UserElemDef user_components[] =
{
 {"U2PA", 2, sizeof(U2PA), U2PA, NULL, u2pa_y, thermal_n, NULL, NULL, NULL},
};
```

The corresponding AEL component definition for this element would be:

```
create_item( "U2PA", "User Defined Element", "I", NULL,NULL, "U2PA",
     standard_dialog, "*", standard_netlist, "ELEMENT", standard_symbol,
     "ARES", no_artwork, NULL,
create_parm( "R1", "Resistance of R1", 0, "rvopt",1,1.000000),
create_parm( "R2", "Resistance of R2", 0, "rvopt",1,1.000000),
create_parm( "R3", "Resistance of R3", 0, "rvopt",1,1.000000),
));
```

In the component definition file, the component is defined as U2PA and the component ID prefix is I. The component uses the standard netlist format for components and the ARES symbol, which needs to be created. It has no associated artwork and has three optimizable parameters with the default value of 1.0.

Note
For details on creating a symbol, refer to *Working with Symbols* (usrguide) in *Schematic Capture and Layout* (usrguide).

Defining Artwork Creation Functions

If you have purchased the Layout option, you have the ability to define parameterized custom layout functions using AEL. The full power of the language is available, including artwork generation functions, math libraries, and data query functions.

Several AEL functions are especially useful for generating layout artwork:

db_factor()
de_ang_factor()
de_set_layer(n)
de_set_global_db_factor()
de_draw_rect(x, y, l, h)

```
de_draw_circ(x, y, r)
de_draw_text(f, x, y, h, a, string)
de_draw_port(x, y, a, n)
de_add_path()
de_add_polygon()
de_add_polyline()
de_end()
de_end_command()
de_draw_point(x, y)
de_draw_arc(x, y, a, f)
```

Artwork functions are assigned in the *create_item()* function. The name of the artwork function is set as the artwork data and the artwork type is set to *artwork_macro* or the integer 2.

Artwork functions use parameters passed into them to control the dimensions and features of the layout geometry. The basic requirement in all AEL artwork creation macros is that the parameters defined for the macro match the exact order of the parameters defined for the component. That is, the names of the parameters used in the macros *defun* statement do not need to match the component's, but the order of the parameters needs to match the order in which they are defined in the *create_item()* function. If W is defined before L in *create_item()*, then W is the first parameter passed in and L the second.

An AEL artwork macro can use all available AEL commands, including file I/O, string manipulation, data base query, lists, math functions, and can also call other AEL artwork functions. If you are constructing a library of artwork generation routines, you can build a set of AEL utilities to simplify tasks that are repeated in more than one function.

The basic format of the artwork function is similar to any AEL function. It is composed of a declaration section (*defun*, followed by a list of parameters), followed by the function body. The function body can contain any number of AEL statements and usually contains a call to a draw function, as well as calls to draw port to create the component's ports. The number of ports needs to match that of any schematic symbol defined for the component. Further, port 1 should always be placed at the 0,0 point.

When using the polygon or polyline command functions to draw complex shapes, follow these by two or more *de_draw_point()* functions. The *de_draw_point()* functions should always be terminated with a call to the *de_end()* function, followed by the *de_end_command()*. This example draws a simple polygon:

```
.
.
.
de_set_layer(iDrawLayer);
de_draw_rect(0, -fWidth * 0.5, fLength, fWidth * 0.5 );
de_draw_port(0, 0, -90.0);
de_draw_port(fLength, 0, 90.0);
.
.
.
```

The simple artwork macro shown next creates a microstrip transmission line. The function is passed in the width and length of the transmission line as the second and third arguments (note the order of the parameter's *fWidth* is the same as in the *create_item()*

function for the MLIN). The width and length parameters are then used to create a rectangle and specify the port 2 location. A more complete version of this macro can be found for in the <installation directory>/circuit/ael/ckt_linear_art.ael file.

```
defun MLIN(w,l)
{
de_set_layer(1);
de_draw_rect(0,-w*.5,l,w*.5);
de_draw_port(0,0,-90.0);
de_draw_port(l,0,90.0);
}
```

Creating a Library of Artwork Objects

Typically, many users have developed a library of artwork to use in designs, such as simple shapes used for drill holes, cover alignment, and mask alignment or parts developed for resistors, capacitors, FETs and other components.

Often, the former are not part of the active circuitry and are not simulated; however, the latter are active components that need to be simulated. The AEL definitions for these two types of objects differ, and the method for creating the supporting AEL can differ as well.

AEL for Simulated Components with Artwork

Generally, you would create new components that can be simulated by following the directions in *Schematic Capture and Layout* (usrguide). The method outlined there involves creating a schematic to represent the simulation model for the artwork.

A number of components have a one-to-one correspondence between a single simulator element and the artwork for the element. This is especially true with FETs, but is often true with lumped components and others. In this case it is simpler to create a new component definition using AEL, than to create a schematic with just this one element placed in it.

The basic idea is to copy the AEL *create_item()* definition for the built-in simulator element, rename and modify the definition to associate it with a artwork file or AEL artwork generation macro. Though the *create_item()* definition for a simulated component can look imposing, you only need to modify a few fields. Then, you add your new AEL files to the appropriate AEL configuration variable so the new components can appear in the library.

An example of the *create_item()* definition for a capacitor is described in the next section, Example Artwork Creation Functions.

Note the changed parameters in *create_item()* : the name, label, artwork type and artwork name in the fixed artwork example, plus the addition of a new parameter in the macro example. The artwork types are fixed, macro and none, the type can be set with the pre-defined variables *no_artwork*, *fixed_artwork* or *macro_artwork*.

If the type is fixed or macro, you need to supply the design file name (minus the *.dsn*

extension) or the macro function name. If the type is a macro, be sure that you load the AEL file containing the macro, or you can include it with the component definition, as is done here.

For details on creating artwork macros, refer to *Schematic Capture and Layout* (usrguide). Note that the macro's parameters are exactly the same as the capacitor's. The names do not have to match, but the position of each parameter (their order in the list) does. You can use this example parameter definition for all layout-only parameters you wish to add to an component. Most likely, you will only need to change the parameter name and default value.

Finally, a *library_group* statement is added to associate the new components with a new library group. You can associate new components with an existing group or name a new group. You can have any number of library groups. If you want the components to show up in a palette as well, you can use the *de_define_palette_group* function to define a similar group for palettes.

All existing AEL definitions for a simulator are stored in the install directory. You should not modify these files directly, but copy the information you need to a new file and add the new file to the AEL search path.

Example Artwork Creation Functions

This example is a more complex artwork generation function that creates the CAPP2_Pad1 component. A number of supporting routines are included to demonstrate the use of generalized utility functions when creating a library of components. This function and the supporting code for Analog/RF designs can be found in these files:

- $HPEESOF_DIR/circuit/ael/examples/ckt_lumped_item.ael file (section of CAPP2_PAD1)
- $HPEESOF_DIR>/circuit/ael/ckt_linear_art.ael file
- $HPEESOF_DIR/de/ael/destdart.ael (section of defun pad1)
  where $HPEESOF_DIR is your installation directory (on UNIX platform).

The sample code is shown in the following figures.

```
// sample code in $HPEESOF_DIR/circuit/ael/ckt_lumped_item.ael
// CAPP2 - Pad1
create_item ("CAPP2_Pad1",              // name
             "Chip Capacitor (Pad Artwork)",          // label
             "C",                       // prefix
             0,                         // attribute
             -1,                        // priority
             "CAPP2_Pad1",              // iconName
             standard_dialog,           // dialogName
             "*",                       // dialogData
             ComponentNetlistFmt,       // netlistFormat
             "CAPP2",                   // netlistData
             ComponentAnnotFmt,         // displayFormat
             "SYM_C",                   // symbolName
             macro_artwork,             // artWorkType
             "CP2_Pad1",                // artworkData
             ITEM_PRIMITIVE_EX,         // extraAttrib
             create_parm("C", "Capacitance", PARM_OPTIMIZABLE | PARM_STATISTICAL,
"StdFileFormSet", CAPACITANCE_UNIT, prm("StdForm", "1.0 pF")),
             create_parm ("TanD", "dielectric loss tangent", PARM_OPTIMIZABLE | PARM_STATISTICAL,
"StdFileFormSet", UNITLESS_UNIT, prm("StdForm","0.001")),
             create_parm ("Q", "quality factor", PARM_OPTIMIZABLE | PARM_STATISTICAL,
"StdFileFormSet", UNITLESS_UNIT, prm("StdForm","50.0")),
             create_parm ("FreqQ", "reference frequency for q", PARM_OPTIMIZABLE |
PARM_STATISTICAL, "StdFileFormSet", FREQUENCY_UNIT, prm("StdForm","300.0 MHz")),
             create_parm ("FreqRes", "resonance frequency", PARM_OPTIMIZABLE | PARM_STATISTICAL,
"StdFileFormSet", FREQUENCY_UNIT, prm("StdForm","500.0 MHz")),
             create_parm ("Exp", "exponent for frequency dependance of q", PARM_OPTIMIZABLE |
PARM_STATISTICAL, "StdFileFormSet", UNITLESS_UNIT, prm("StdForm","2.0")),
             create_parm ("W",  "W", PARM_NOT_NETLISTED,  "StdFormSet", LENGTH_UNIT,
prm("StdForm","25.0 mil"), list(dm_create_cb(PARM_DEFAULT_VALUE_CB,
"get_default_length_value_cb", "25.0", TRUE))),
             create_parm ("S",  "S", PARM_NOT_NETLISTED,  "StdFormSet", LENGTH_UNIT,
prm("StdForm","10.0 mil"), list(dm_create_cb(PARM_DEFAULT_VALUE_CB,
"get_default_length_value_cb", "10.0", TRUE))),
             create_parm ("L1", "L", PARM_NOT_NETLISTED,  "StdFormSet", LENGTH_UNIT,
prm("StdForm","50.0 mil"), list(dm_create_cb(PARM_DEFAULT_VALUE_CB,
"get_default_length_value_cb", "50.0", TRUE))));
```

Artwork Generation Function Example (ckt_lumped_item.ael)

```
// sample code in $HPEESOF_DIR/circuit/ael/ckt_lumpedart_def.ael
library_group("lumpedart",
              "Lumped Components (with artwork)",
              "SMT_Pad",
              "C_Pad1",
              "C_Space",
              "C_Conn",
              "CAPP2_Pad1",
              "CAPP2_Space",
              "CAPP2_Conn",
              "CQ_Pad1",
              "CQ_Space",
              "CQ_Conn",
              "L_Pad1",
              "L_Space",
              "L_Conn",
              "LQ_Pad1",
              "LQ_Space",
              "LQ_Conn",
              "R_Pad1",
              "R_Space",
              "R_Conn");
```

Artwork Generation Function Example (ckt_lumped_item.ael)

```
// sample code in $HPEESOF_DIR/circuit/ael/ckt_linear_art.ael
defun CP2_Pad1 (c,tand,q,fq,fr,exp,w,s,l)
{
    elem = "CP2";
    pad1(w,s,l,xlatorLayer);
}
```

Artwork Generation Data Example (ckt_linear_art.ael)

```
// sample code in $HPEESOF_DIR/de/ael/destdart.ael
//PAD1
defun pad1(w,s,l,layer)
{
  de_set_global_db_factor();
  if(w <= 0.0)
    error(DeStdArtErrClass,2,fmt_tokens(list(DeStdArtElem,"Width <= 0.0
:",w)),"");
  if(s <= 0.0)
    error(DeStdArtErrClass,3,fmt_tokens(list(DeStdArtElem,"Spacing <= 0.0
:",s)),"");
  if(l <= 0.0)
    error(DeStdArtErrClass,1,fmt_tokens(list(DeStdArtElem,"Length <= 0.0
:",l)),"");
  if (s + s > l )
    error(DeStdArtErrClass,4,fmt_tokens(list(DeStdArtElem,"Two Spacings >
Length :",s,l)),"");

  de_set_layer(layer);
  de_draw_rect(0,-w*.5, s,w*.5);
  de_draw_rect(l-s,-w*.5, l,w*.5);
  de_draw_port(0,0, -90, FALSE,1);
  de_draw_port(l, 0, 90, FALSE,2);
}
```

Artwork Data Example (destdart.ael)

Modifying AEL Configuration Variables

After you have created AEL definitions for a new library, you must modify the appropriate configuration variables for the file with the new definitions to be loaded automatically. Which variable you modify depends on who will use the definitions.

Using Pointers to find variable addresses

It is possible to access the address of a variable through the "&" operator. To access the contents of an address of the variable, you must use the "*" operator. These operators are the same as C. "&" signifies the address of a variable, and "*" signifies the contents of the variable.

This is most useful when passing parameters to AEL functions. When the address is passed to a variable rather than the variable itself, you can modify that parameter in the function, and the parameter will remain modified after the function. This functionality is the same as in C.

Below is an example of using the "&" and "*" in both a parameter and in a global context. Using the "&" and "*" in the global context if you wish to manipulate addresses of variables. This will not be discussed here, but can be studied from C documentation.

Example:

```
defun modify(addr)
{
fprintf(stderr,"addr = %s\n",identify_value(addr));
// prints "addr =
fprintf(stderr,"original value of addr = %s\n",*addr);
// prints "value of addr
= 100
*addr = *addr + 100;
}
decl val = 100;
fprintf(stderr,"address of val = %s\n",identify_value(&al));
// prints
"address of val =
fprintf(stderr,"original value of val = %s\n",*(&al));
// prints "value of
val = 100
modify(&al);
fprintf(stderr,"new value = %d\n",val);
// prints "new value = 200"
```

Using AEL with Component Libraries

Each design type comes with a pre-defined component set. The set includes all the basic components that a simulator understands, such as transmission line components, lumped components, filters, active devices, etc. For each component, an AEL description exists which describes the component to the design environment and defines the component's interface to the simulator.

In the design environment, the simulator and the front-end schematic and layout capabilities are separate programs which communicate by sending data back and forth through an Interprocess Process Communication (IPC) protocol. The data that is simulated by the simulator is sent as a netlist. A netlist describes the schematic and its simulation controls in a form understood by the simulator. When you create new parts, you must create a netlist format string for the component that corresponds to the syntax expected by the simulator. This syntax is described in *Format Strings* (ael).

Some of the new parts that you can create are:

- Components linked to the simulator using the user-compiled model feature of the simulator.
- Components that reference specific measured data files, such as in-house or custom active devices.
- Components with specific artwork requirements, such as chip capacitors, printed resistors and etched inductors.
- Parametric subnetworks, which model commonly-used devices.

In order to access the new components through the program, you must add the components to an existing library or create a new library. The design environment becomes aware of new parts through a combination of configuration file changes.

Note
For details, refer to the section *Building User-Compiled Analog Models* (modbuild).

Using AEL for Library Definition

The libraries of simulation components are defined for the program through AEL. Adding new libraries to the program requires creating or changing AEL files. AEL definition files are interpreted when the program is invoked or after attaching to a project. The AEL files contain the definitions for the new library parts and for the groups used to present the parts through the palettes and library lists.

The design environment refers to parts as *components*. The definition for a new component specifies information such as:

- Component name
- Descriptive label
- Component parameters
- Symbol used to represent the component in the schematic
- Artwork used to represent the component in the layout
- Syntax used to represent the component in the netlist
- How the components parameters are displayed in the schematic
- Image (icon) used to represent the component in a palette (if the part is used in a

palette)

The program refers to libraries or palettes as *groups*. The definition of a group specifies:

- Group name
- Descriptive label
- List of components in the group

The AEL definitions for components and groups take the form of function calls that are interpreted sequentially from the AEL file. The AEL functions required to define the program components and groups are described in a later section.

You should not modify the AEL files that are shipped with your product. Instead, create new AEL files to hold the AEL definitions for new or altered parts and libraries. You can use any text editor to create and modify the AEL files. New files can have any name you choose, but you must use the extension ael; for example *steves.ael* is an appropriate name for a new AEL file.

Note
If you do modify the shipped AEL files, create backups of the original files first, in case you need to restore them.

Although AEL files can be located anywhere in the file system, usually these files are located in a directory convenient to all users. Since the $HPEESOF_DIR/custom directory is not modified by a new release installation, all system-wide customized component definitions can be stored there. You can locate the files in an application-specific sub-directory, such as $HPEESOF_DIR/custom/circuit/ael, $HPEESOF_DIR/custom/adsptolemy/ael, or $HPEESOF_DIR/custom/de/ael.

If the files are used by a particular user (or all users of a particular workspace), locate the files in the $HOME/hpeesof/circuit/ael, $HOME/hpeesof/adsptolemy/ael, or $HOME/hpeesof/de/ael directory. For a particular workspace, locate the files in the project's networks directory.

The environment variables must be set properly to allow the program to find and read the new AEL files. For details on the environment variables that are used, refer to the section, Environment Configuration Directories.

When adding parametric subnetworks as new parts, the design environment does much of the AEL work for you. The program creates an AEL definition file in the current workspace when a network design is saved. This file defines the parametric subnetwork component and assigns it to a library. New parametric subnetworks automatically become part of the current working project. Assignment of the symbol, artwork, and library list group can be made from within the program before the network is saved. Parameters can be added to a network from within the program. However, you can not define palettes for parametric subnetworks from within the program. If you want to add your parametric subnetworks to a palette, you must alter the AEL files created by the program or create your own AEL definition file.

Environment Configuration Directories

The environment configuration file, de_sim.cfg, contains many definitions required by the program. Directories that contain the environment file are:

$HPEESOF_DIR/config Environment variables defined at this level are effective for all sessions unless overridden by custom, user or workspace definitions.

$HPEESOF_DIR/custom/config Environment variables defined at this level are effective for all sessions unless overridden by user or workspace definitions. This directory allows site-wide modifications without changing the installation directory $HPEESOF_DIR/config.

$HOME/hpeesof/config Environment variables defined at this level override those defined in the system environment variable files, but only for the current user.

Workspace Environment variables defined at this level override those defined in the both the user and system environment variable files.

Using Environment Variables

Environment variables tell the program which AEL files to read, the directories where they are located, and directories where associated design files are located.

AEL files are located and read by the program according to the values of environment variables that are defined in environment file, de_sim.cfg. The names of new AEL files must be added to the definitions in de_sim.cfg for the program to know about them and read them. These environment variable definitions can be changed by editing the de_sim.cfg file.

Each definition has the form of a variable name followed by an equal sign and a value (MY_NAME=my_value). The value is text, possibly a single name or a list of names separated by colons or semi-colons. Values should not have any embedded spaces.

The important configuration variables are: SIMULATOR_AEL, AEL_PATH, USER_AEL, LOCAL_AEL, USER_DSN_PATH, and SITE_AEL. For detailed information on these variables, refer to *Customizing Configuration Variables* (custom) in *Customization and Configuration* (custom).

AEL Debugger

A new AEL debugger has been added to Advanced Design System.

Basic Capabilities

The AEL debugger:

- Steps through syntax-highlighted AEL source code
- Has an interactive call stack display
- Shows variable values in tool tips
- Allows setting of breakpoints
- Can be used with ADS.

How to use the debugger

To start the debugger

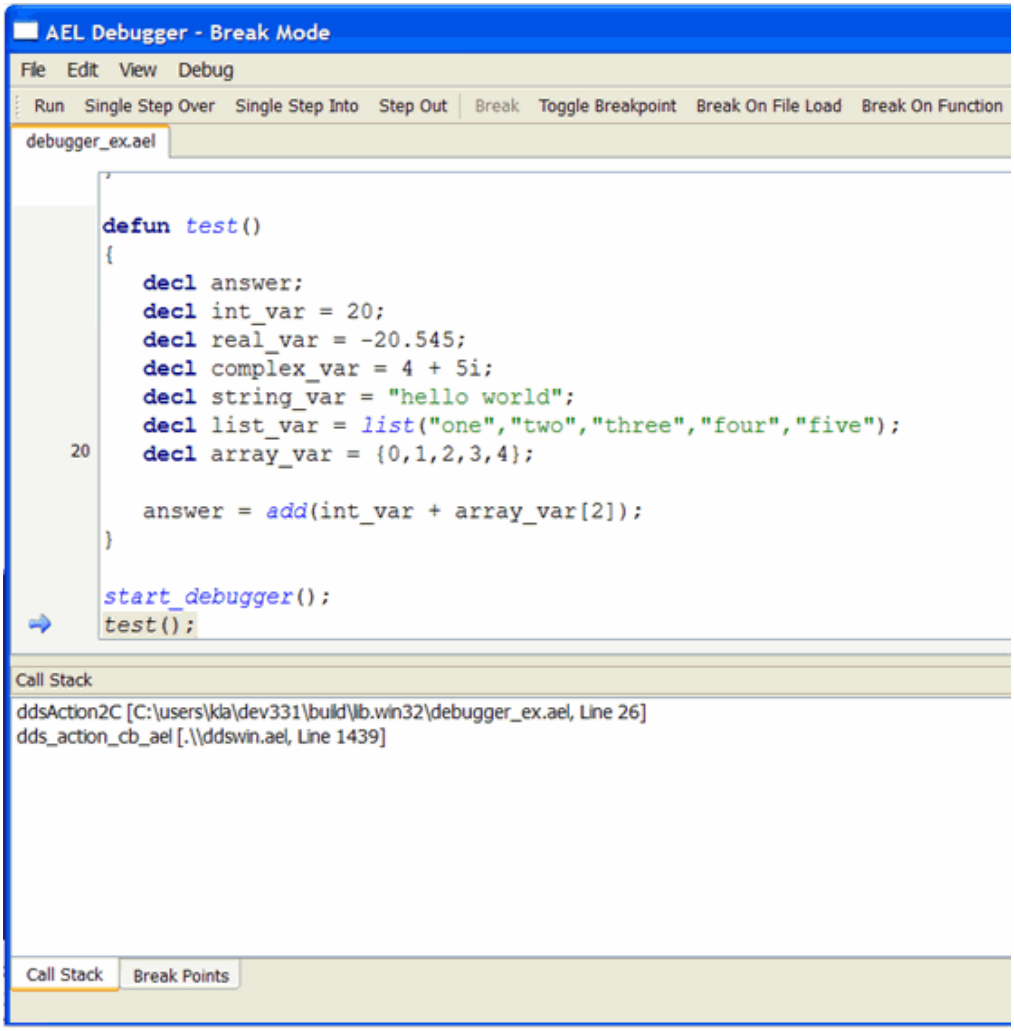
The AEL statement to start the AEL debugger is:

```
start_debugger();
```

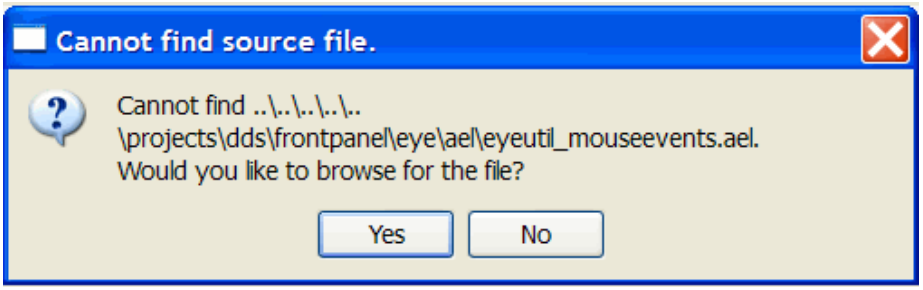
There are three ways to execute this statement:

- Insert the statement in any AEL script.
- In ADS, open a command line dialog (use Tools->Command Line... menu), and insert the statement at the "Command>>" prompt.
- Start ADS with the command line argument "-debugAel".

When this statement is executed, the AEL debugger in invoked. ADS will halted until the "Run" menu is selected or the F5 key is pressed. ADS will be halted in the AEL source from where the start_debugger() call was executed. If the AEL source is available and can be found, the debugger will show the AEL source in the main window.



If the AEL source is not found, then a dialog will appear asking the user if he wishes to browse for the file.



To use the debugger

1. Open the AEL script you wish to debug. You have 2 choices:
    1. If you want to debug an AEL file that has not yet been loaded, use the "Break on File Load" or "Break on Function" commands on the toolbar or in the "Debug" menu. When specifying a file name, use the exact leaf name. When the specified file is loaded, the debugger will break. Note that you cannot load a file from within the debugger. The application must load the file. If the file cannot be found, then a dialog will appear providing the user the opportunity to browse for the file. When specifying a function name, the debugger will break the first time the function is executed after setting the breakpoint. In both cases, the breakpoint is removed after the 1st time its encountered.

    2. If the AEL file has been loaded, use the menu "File->Open". A dialog will appear that has a list of all the AEL files loaded. Upon selection, if the file actually exist at the specified location, then it will be opened. If it is not found, then a dialog will appear that allows the user to browse for the file. Note that many files may appear in this list that do not have the source available to you. You cannot debug these files.
2. Set a breakpoint. There are 3 choices:
    1. If you want to break when an AEL file that has not yet been loaded, use the "Break on File Load". See Step 1a.
    2. If you want to break on a function when it is invoked, use "Break on Function". See Step 1a.
    3. Once a file is open, you can set a breakpoint by clicking the left mouse button anywhere on the left side of the file area. When a breakpoint is set, a stop sign will appear at the point of break.
    All breakpoints can be viewed by selecting the tab along the bottom of the debugger called "Break Points".
3. Clear a breakpoint. You have 2 choices:
    1. Click the left mouse button on top of a stop sign that appears on the left side of the file area.
    2. Select the tab "Break Points" along the bottom of the debugger. Select any breakpoint from that list, and click the right mouse button and select the menu "Delete".
4. View all breakpoints.
    Select the tab "Break Points" along the bottom of the debugger. If you double click on any breakpoint in the list, the location of the breakpoint in the file area is highlighted.
5. View a variable. There is no way to view global variables. You cannot view variable while in run mode.
    Hover the mouse over the variable you wish to view in the file area. A tool tip will appear the prints the name, type, and value of a variable.
6. Continue Execution.
    1. Press "Run" tool button.
    2. Press "Debug->Run" menu.
    3. Press F5.
7. Single step over one statement. NOTE: Sometimes it will take more than one step to get through in line that has embedded statements.
    1. Press "Single Step Over" tool button.
    2. Press "Debug->Single Step Over" menu.
    3. Press F10.
8. Single step into a statement.
    1. Press "Single Step Into" tool button.
    2. Press "Debug->Single Step Into" menu.
    3. Press F11.
9. Step out of a statement. This will take you back to where the statment was called.
    1. Press "Step out" tool button.
    2. Press "Debug->Step Out" menu.
    3. Press Shift+F11.
10. View the callstack.
    Select the tab "Call Stack" along the bottom of the debugger. If you double click on any of the entries, the file area will change to that context and highlight where that call is made. There are a couple of conditions in which this will not work:
    1. If the entry has a line number of -1.
    2. The source of that function is not available to you.
11. Find a word.
    1. Press "Edit->Find" menu. Respond to the dialog that appears.

2. Press Ctrl+F. Respond to the dialog that appears.
3. Press "Edit->Find next" menu to find the word again.
4. Press "Edit->Find next" menu to find the word again.

12. Go to a particular line number.
    1. Press "Edit->Goto Line...". Respond to the dialog that appears.
    2. Press Ctrl+G. Respond to the dialog that appears.

Important notes

- There is no checksum/timestamp check betweek the ael and the atf files, so you must be sure to get the correct version of the ael file.
- Currently, the debugger window does not pop up when a break-point is hit. Its nice to put the debugger on a second monitor if possible.
- ADS system functions may appear in the Call Stack. However, they are not debuggable.

Walkthrough

For instance, suppose you have to following AEL script called debugger_example.ael.

```
defun add(a,b)
{
  decl sum;
  sum = a + b;
  return sum;
}
defun test()
{
  decl answer;
  decl int_var = 20;
  decl real_var = -20.545;
  decl complex_var = 4 + 5i;
  decl string_var = "hello world";
  decl list_var = list("one","two","three","four","five");
  decl array_var = {0,1,2,3,4};
  answer = add(int_var, array_var[2]);
}
test();
```

If you want to stop on the test() function inside ADS.

1. Start the debugger with one of the following methods:
   - Add the -debugAel command line argument.
   - Or, run the command start_debugger(); from the command window.
2. Use one of the following methods to get a breakpoint set.
   - If the file hasn't yet been loaded, then select "Break on File Load", and enter debugger_example.ael.
   - Or, select "Break on Function" and enter "test".
3. If you are in Break mode, select "Run" from the toolbar or Debug menu (Or press F5).
   - In ADS, in the command line dialog, type: load("debugger_example.ael");
   - Switch to the debugger window which should now be in "Break" mode.
   - You will see a blue arrow on the left showing the current statement.
4. If you used "Break on File Load" or "Break on Function", then the breakpoint you set was temporary (one-time only). If you want to set a permanent breakpoint in this file, click on the left margin (like in visual studio), or press F9.

5. Press F10 to step to the next line (going over any functions). Note: If you are on a multi-line statement, sometimes the arrow will go back and forth between the lines before proceeding.
6. Press F11 to step in, going into any functions.
7. Press Shift+F11 to step out of the function you are in.
8. Note that you can double-click on the call stack window to navigate up the stack.

DesignContext in AEL

This section describes what is a DesignContext and how you can extract design information from the Design Environment (DE) by using the AEL DesignContext based functions.

Note
The AEL DesignContext object and these DesignContext based functions were introduced in ADS 2009 Update 1.

DesignContext Overview

A DesignContext is an AEL object that holds information about a particular design being edited. The DesignContext is not associated with a window, but knows the type of view (i.e. schematic,symbol or layout) or if its associated with an artwork macro evaluation.

It is intended to represent design data in a generic way, with the same API or set of DesignContext based functions being used throughout. In this way, the same piece of code may work on a number of different design data, if the design context is used as a parameter.

Try to write code so it is parameterized by DesignContext AEL objects - if you do this, the same piece of code may write/read/query/traverse on a number of different design data, by passing a different design context.

Database traversal using a DesignContext

To traverse the design data of a DesignContext, a DesignContext can be used to create an instance iterator and/or a shape iterator. An instance iterator is useful in traversing the instances contained in a context's design data, see the function *db_create_inst_iter()* (ael) on how to travese the instances of a DesignContext. A shape iterator is useful in traversing the shapes contained in a context's design data, see the function *db_create_shape_iter()* (ael) on how to travese the shapes of a DesignContext.

Database query/retrieval using a DesignContext

A DesignContext can be used to retrieve/traverse instance and shape design data using instance and shape iterators. From the instance iterators and shape iterators, there is more design data information that can be queried/retrieved.

Example use of DesignContext AEL objects.

```
/// Example how to traverse the instances of the design context's design data.
///  and deselect those instances that are already selected.
// Get current design context.
decl context = de_get_current_design_context();
// Deselect any selected instances in the current design context.
decl instIter = db_create_inst_iter(context);
instIter = db_inst_iter_limit_selected(instIter);
for ( ; db_inst_iter_is_valid(instIter); instIter = db_inst_iter_get_next(instIter))
   db_select(instIter, FALSE);
...
/// Example how to get the first selected shape of the design context's design data
///  and get the bounding box of that first selected shape.
```

```
decl context = de_get_current_design_context();
// Get first selected shape in the current design context.
decl shapeIter = db_create_shape_iter(context);
shapeIter = db_shape_iter_limit_selected(shapeIter);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
  return;
// Get bounding box of first selected shape.
decl bboxH = db_get_shape_bbox(shapeIter);
...
/// Example how to find all the rectangular shapes that have selected points
///  in a layout design context's design data and get the selected points coordinates.
decl winInst = api_get_current_window();
// Get DesignContext of the current window.
decl context = de_get_design_context(winInst);
// Only work on a layout DesignContext.
if (!de_is_layout_context(context))
  return FALSE;
// For all selected rectangles in the DesignContext get the
//  selected points coordinates and do something with the coordinates.
decl shapeIter = db_create_shape_iter(context);
shapeIter = db_shape_iter_limit_selected(shapeIter);
for( ; db_shape_iter_is_valid(shapeIter); shapeIter = db_shape_iter_get_next(shapeIter))
{
   // For all rectangles, process selected points
   if (db_shape_is_rectangle(shapeIter))
   {
       decl selCoordLst = db_get_shape_selected_points(shapeIter);

       // For any selected points on rectangles do something
       decl selIdx = 0;
       for (selIdx=0; selIdx < listlen(selCoordLst); selIdx++)
       {
            decl coordH = nth(selIdx, selCoordLst);
            decl x = db_get_x(coordH);
            decl y = db_get_y(coordH);
            ...
            // Do something here with each selected point.
            ...
       }
   }
}
return TRUE;
```

AEL functions

See AEL *DesignContext Functions* (ael).

See also *Connectivity Objects* (ael) and *ADS Compatibility Examples* (ael)

Version Introduced

ADS 2009 Update 1

Connectivity Objects

ADS 2011 and newer versions have adopted a standard terminology for connectivity objects.

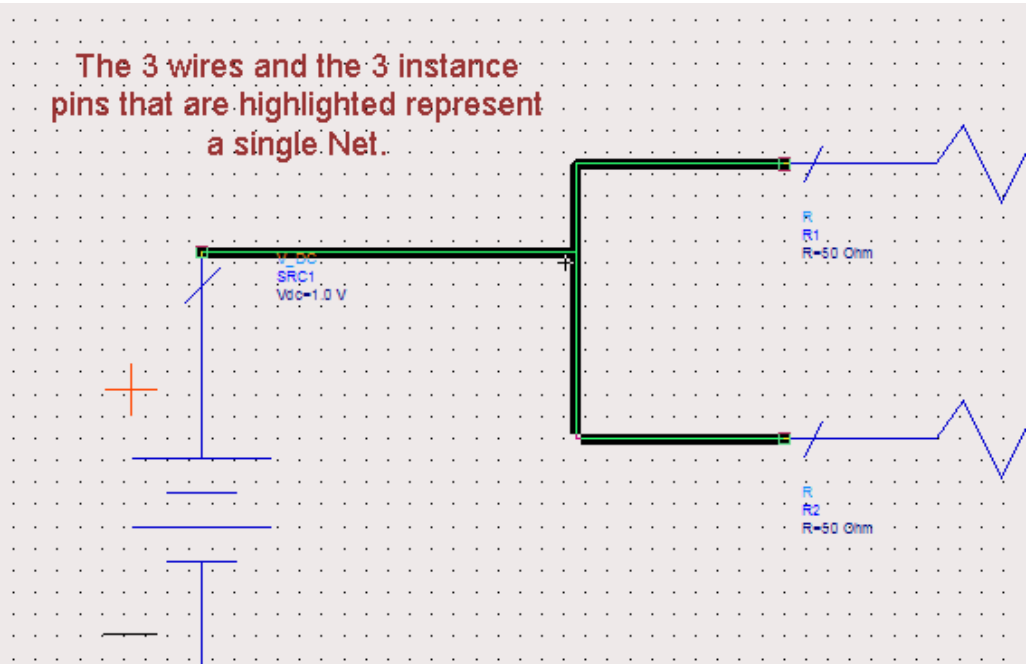
This section describes the following connectivity objects available within an ADS design:

- Nets
- Terms
- Pins
- InstTerms
- InstPins

Net

A **Net** represents the logical connectivity within a design. A net logically represents an electrical path in a circuit.

- Nets connect to Terms (terminals), which are the logical connection points on instances.
- Nets also connect to InstTerms (instance terminals) that represent the logical connections to the lower-level instances.
- A collection of wires or interconnects that carry the same electrical signal is considered to be in the same net.
- A net can be physically implemented using connection figure objects such as shapes (polygons, paths, lines, etc.).



The above diagram displays three highlighted wire lines and instance pins are the physical representation of a single Net.
The single Net is a logical connection of three instance terminals.

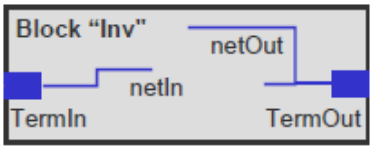
Term

A **Term**, or terminal, represents the logical connection points for a design. A terminal is an

interface to a design. Nets are made available from one design to a higher-level design through a terminal.

- A Term holds information on the terminal name, terminal number and the Net it is contained within.
- Pins associated with terminals represent the physical implementation of the interface to the design, or in other words they represent the physical connection points.

This following figure displays the conceptual view of a Term where the Terms termIn and termOut are each a point where a Net connects to external logic.

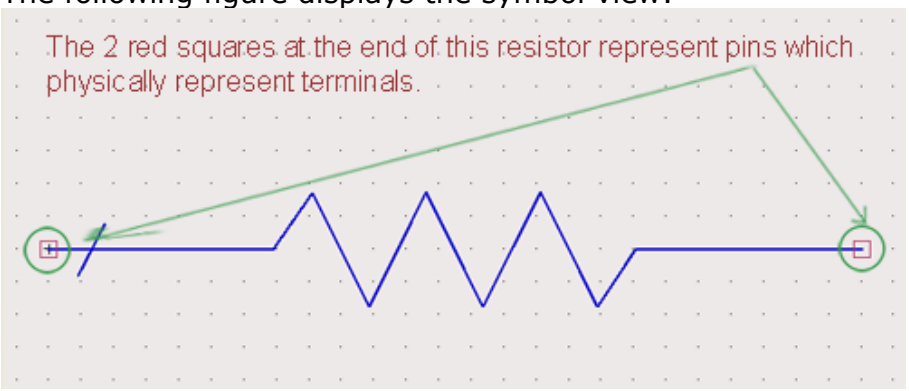


Pin

A **Pin** represents the physical connection of a terminal to a net.

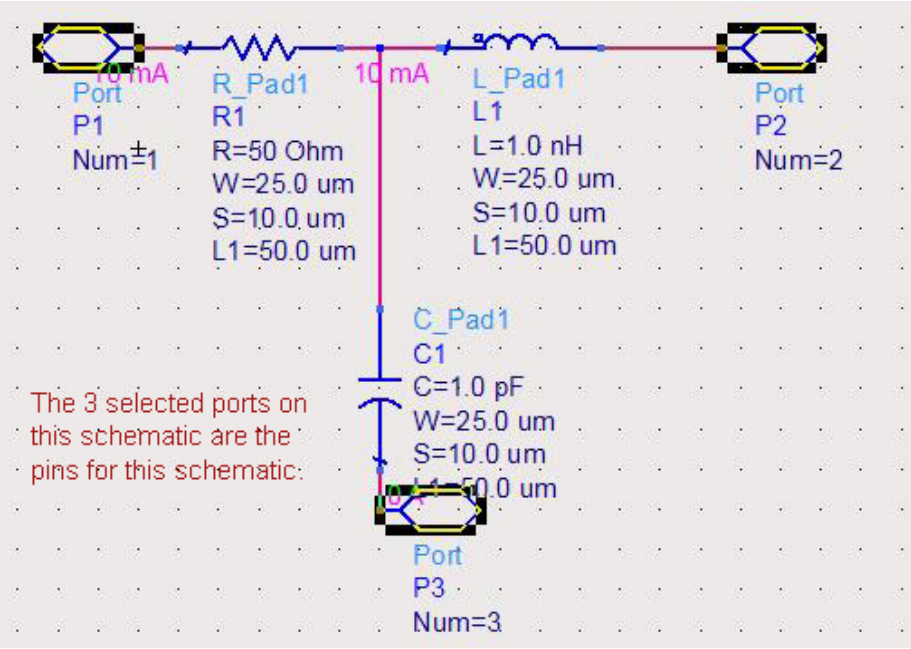
- A terminal can have multiple pins where multiple physical connections can correspond to one logical connection.
- A Pin is associated with one or more physical figures.
- Pins have names.
- A Pin holds information on the term object it represents and physical pin location information such as angle and bounding box information.

The following figure displays the symbol view:



The resistor symbol diagram's two red squares are Pins which are physical representations of corresponding terminals.

The following figure displays the schematic view:



The schematic diagram's three selected ports are Pins which are the physical physical representations of corresponding terminals.

Instance

An **Instance** is a reference to another design from the context of the design in which it is placed. The design referenced by an instance is referred to as the Master Design.

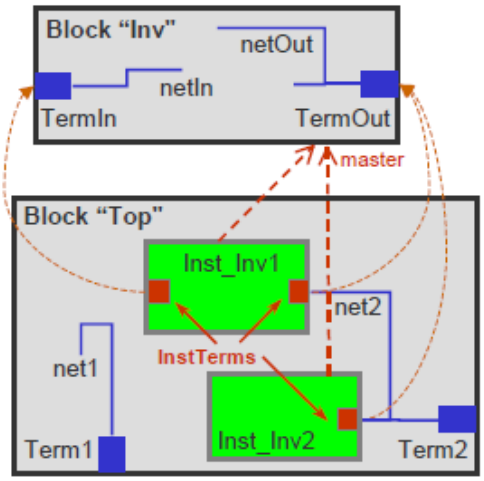
- An instance represents the inclusion of one design as part of the contents of another.
- You can use instances to create hierarchical designs because the master of an instance can contain instances of other master designs.
- The design containing the instance is the parent design and the design that is included is the master of the instance.

InstTerm

An **InstTerm**, or instance terminal, represents a connection between a net and a terminal in the master of an instance.

- The Instance terminals make connections between nets in one level of a design to terminals (and their nets) in a different level of the design in an instance master.
- A net specification (which can be NULL), instance, and terminal specification are required for creating an InstTerm (instance terminal).

The following image displays the conceptual view of an InstTerm which is a point on an Inst where a Net in the parent/owner Design may connect to a Term in the Inst's master.



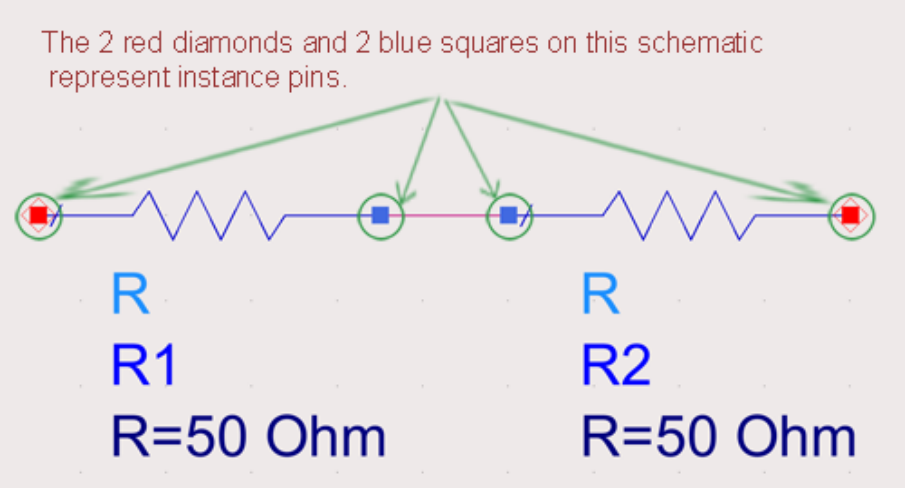
In the above diagram, InstInv1 has InstTerms for both Terms on its master, while only the InstTerm for TermOut was created for InstInv2. InstInv1's InstTerm corresponding to TermIn in the "Inv" master has not yet been connected to any Net.

InstPin

An **InstPin**, or instance pin, represents a pin in the master of an instance mapped into the parent design. When a wire (the physical implementation of a net) connects to an InstPin, the net associated with the wire connects to the InstTerm associated with the InstPin.

- All InstPins (instance pins) are associated with a physical figure.

The following image displays two blue boxes and two red diamonds (instance pins).



Design

An ADS **Design** holds the design data that describes a part of a design. It can hold design geometry, design connectivity, and design hierarchy information about a particular design.

In ADS 2011 and newer releases, a design consists of a library, cell and view. A schematic is a design, a symbol is a design, and a layout is a design. The design contains only one view.

In ADS 2009 Update 1 and older releases, a design lives in only one global library and can contain all the three views:

- schematic
- layout
- symbol view

Some Notes About Connectivity Objects

- Terms (terminals) and InstTerms (instance terminals), the logical objects, are most commonly used in code unless your code may have a need for location information (e.g. x,y coordinates in schematic/layout view) and then Pins or InstPins, the physical objects, would be necessary.
- A Symbol pin is the physical representation of a Term in a symbol view.
- A port in an Artwork macro is represented by a Pin that physically represents a Term (terminal).

Connectivity Object AEL Functions

- *Net Functions* (ael)
  - *Net Iterator Functions* (ael)
- *Term Functions* (ael)
  - *Term Iterator Functions* (ael)
- *InstTerm Functions* (ael)
  - *InstTerm Iterator Functions* (ael)
- *Pin Functions* (ael)
  - *Pin Iterator Functions* (ael)
- *InstPin Functions* (ael)
  - *InstPin Iterator Functions* (ael)

See also *DesignContext in AEL* (ael) and *ADS Compatibility Examples* (ael)

AEL Functions (by category)

(For alphabetized list of Functions, *click here* (ael))

Contents

Command Functions

Name	Description
db_add_arc() (ael)	Adds a clockwise or counterclockwise circular arc to a polygon or polyline under construction in the given design context.
db_add_arc1() (ael)	Creates a standalone arc in the given design context.
db_add_arc2() (ael)	Creates a clockwise or counterclockwise standalone arc in the given design context.
db_add_arc3() (ael)	Creates a clockwise or counterclockwise standalone arc in the given design context.
db_add_arc4() (ael)	Creates a clockwise or counterclockwise standalone arc in the given design context, that is created with a user defined starting point coordinate, center point coordinate, and a chord length for an arc angle expressed in terms of circumference.
db_add_circle() (ael)	Adds a circle in the given design context in user-defined units.
db_add_construction_line() (ael)	Adds a construction line to the given design context using user-defined units.
db_add_path() (ael)	Starts a path command sequence. Adds a path to the given design context.
db_add_point() (ael)	Adds a construction line to the given design context using user-defined units.
db_add_polygon() (ael)	Adds a point to a polygon, polyline, or path using user units
db_add_polyline() (ael)	Starts a polygon command sequence.
db_add_rectangle() (ael)	Adds a rectangle to the given design context.
db_end() (ael)	Completes a polygon, polyline, wire or trace command sequence.
de_activate() (ael)	Activates an instance by setting the deactivate flag of an instance to false.
de_add_arc() (ael)	Adds a clockwise or counterclockwise, circular arc to a polygon or polyline in user units in the current representation.
de_add_arc1() (ael)	Adds an arc using user-defined units.
de_add_arc2() (ael)	Adds an arc using user-defined units.
de_add_arc3() (ael)	Adds an arc using user-defined units.
de_add_arc4() (ael)	Adds an arc using user-defined units.
de_add_circle() (ael)	Draws a circle in the current representation in user-defined units.
de_add_construction_line() (ael)	Adds a construction line using user-defined units.
de_add_path() (ael)	Starts a path command sequence.
de_add_point() (ael)	Adds a point to a polygon, polyline, or path using user-defined units.
de_add_polygon() (ael)	Adds a polygon to the current representation.
de_add_polyline() (ael)	Adds a polyline to the current representation.
de_add_property() (ael)	Adds property to data group, port, or instance.
de_add_rectangle() (ael)	Adds a rectangle to the current representation.
de_add_text() (ael)	Adds a text string to the current representation.
de_add_trace() (ael)	Starts trace command sequence for adding a trace to the current representation.
de_add_vertex() (ael)	Adds a vertex to an existing polygon, polyline, wire or trace in the current representation.
de_add_wire() (ael)	Adds a wire vertex to a wire or trace connection in the current

	representation.
de_add_wire label() (ael)	Adds a label to a wire or pin at the x,y location.
db_highlight_instance_ex() (ael)	Highlights an instance using the highlight color.
db_unhighlight_instances_ex() (ael)	Function to unhighlight all instances in a given design context.
de_bom() (ael)	Generates a bill of material file for the current representation.
de_boolean logical() (ael)	Performs boolean operations on shapes placed on different layers in a layout presentation.
de_break connection() (ael)	Breaks connection to attached wire or trace from the selected instances in the current representation.
de_cell_exists() (ael)	Returns TRUE if a cell with the given cell name exists in the given open library.
de_cellview_exists() (ael)	Returns TRUE if a cellview with the given cell name and view name exists within a given open library.
de_change_annotation_layer() (ael)	Changes the layer of the nearest annotation (within the select region) to the current layer.
de_check_rep_options() (ael)	Generates a check representation report.
de_chop() (ael)	Removes the defined rectangular region from selected polygons, rectangles, circles, or paths.
de_clear_dc_annotation() (ael)	Removes the annotated DC node voltages and branch currents on the current schematic.
de_clear_highlighting() (ael)	Clears any highlighting from a design representation in a current window.
de_close_all() (ael)	Closes all designs from memory.
de_close_all_designs_and_windows()	Closes all design windows and all designs.
de_close_window() (ael)	Closes the current window instance.
de_close_workspace_without_prompting() (ael)	Closes the current open workspace.
de_connect() (ael)	Starts a wire or trace connection using autorouting.
de_convert_path_to_trace() (ael)	Converts selected paths to traces.
de_convert_to_polygon() (ael)	Converts selected closed shapes to polygons.
de_convert_trace_to_path() (ael)	Converts selected traces to paths.
de_convert_traces_to_instances() (ael)	Converts selected traces to transmission line elements.
de_copy() (ael)	Copies the selected components in the current representation and places them at delta from the original position.
de_copy_cell() (ael)	Function to copy a cell. Returns TRUE if cell was successfully copied, FALSE if not.
de_copy_cellview() (ael)	Function to copy a cellview. Returns TRUE if cellview was successfully copied, FALSE if not.
de_copy_file() (ael)	Copies a file.
de_copy_to_buffer() (ael)	Copies the selected group in the current representation to the copy buffer.
de_copy_to_layer() (ael)	Copies the selected group in the current representation to the current active layer.
de_create_window() (ael)	Creates and opens a window.
de_create_polygon() (ael)	Returns a Shape polygon object and adds the polygon to the given design context and refreshes the design context's view.
de_crop() (ael)	Preserves the defined rectangular region from selected polygons, rectangles, circles, or paths while removing all areas outside of the region.
de_dc_annotation() (ael)	Display DC node voltages and branch currents on the current schematic after a DC or Transient simulation.
de_deactivate() (ael)	Sets instance's deactivate flag to true (makes the instance deactivated).

de_define_edge_area_port() (ael)	Modifies a circle, path, polygon, rectangle to become a area port or modifies a polyline or arc to become a edge port.
de_define_npport() (ael)	Places a non-preferred port/pin in an artwork instance.
de_define_port() (ael)	Places a single port, with a unique port number, in an artwork instance using user units.
de_delete() (ael)	Deletes selected objects, or object near given point in current representation.
de_delete_all_orphaned_instances() (ael)	Deletes all the orphaned instances in the current representation.
de_delete_cell() (ael)	Function to delete a cell and all of its contents. Returns TRUE if cell was successfully deleted, FALSE if not.
de_delete_cellview() (ael)	Function to delete a cellview. Returns TRUE if cellview was successfully deleted, FALSE if not.
de_delete_view() (ael)	Deletes a stored view with the matching name of view.
de_delete_workspace() (ael)	Deletes a specified ADS workspace from the given workspace directory path.
de_deselect_all() (ael)	Deselects all objects in the current representation.
de_deselect_all_force() (ael)	Deselects all objects regardless of layer protection status.
de_deselect_by_name() (ael)	Deselects instances by instance name or ID.
de_deselect_window() (ael)	Deselects all objects in given window in current representation.
de_difference() (ael)	Creates new polygons by subtracting the intersections of the selected shapes (e.g., polygon, rectangle, circle, or path) from the union of the selected shapes on the same layer.
de_draw_arc() (ael)	Adds a circular arc to a polygon or polyline in simulator units.
de_draw_arc1() (ael)	Adds an arc using simulator units.
de_draw_arc2() (ael)	Adds an arc using simulator units.
de_draw_arc3() (ael)	Adds an arc using simulator units.
de_draw_arc4() (ael)	Adds an arc using simulator units.
de_draw_circ() (ael)	Draws a circle in simulator units.
de_draw_point() (ael)	Adds a point to a polygon or polyline to the current representation using simulator units.
de_draw_port() (ael)	Adds a port to an art work instance.
de_draw_rect() (ael)	Draws a rectangle in simulator units.
de_draw_text() (ael)	Draws text at given location, with given font, height and angle, height and angle are in simulator units.
de_dse_l2s() (ael)	Synchronizes the schematic with the layout, using the layout as the reference representation.
de_dse_s2l() (ael)	Synchronizes the schematic with the layout, using the layout as the reference representation.
de_edit_annotation_attribute() (ael)	Edits attributes of selected instance's parameter annotation.
de_edit_item() (ael)	Allows a given item to be selected for editing.
de_edit_item_ex() (ael)	Allows a given item to be selected for editing. Returns an item info handle for the instance.
de_edit_path_trace() (ael)	Modifies attributes of a selected path or trace in the current representation.
de_edit_symbol_pin() (ael)	Sets up the editing of a symbol pin.
de_edit_text_attribute() (ael)	Edits the attributes of selected text components in the current representation.
de_edit_text_string() (ael)	Edits text string in the current representation that was set with de_set_edit_text().
de_empty() (ael)	Empties an enclosed, filled shape in the current representation, creating a hole.
de_end() (ael)	Completes a polygon, polyline, wire or trace command sequence.

	This command completes a shape.
de_end_command() (ael)	Terminates a repeating command sequence.
de_end_edit_item() (ael)	Commits and displays the editing changes of a given instance.
de_export_design() (ael)	Exports the current layout or schematic in given format.
de_fill() (ael)	Converts an empty shape (a hole) in the current representation into a filled shape (normal closed shape).
de_find_arc_center() (ael)	Moves the cursor to the center of the arc or circle selected within the select region and enters the coordinates to the event that was previously installed.
de_find_line_center() (ael)	Moves the cursor to the pin closest to the center of the line selected within the select region and enters that point to the event that was previously installed.
de_find_pin() (ael)	Moves the cursor to the pin closest to the point selected within the select region and enters that point to the event that was previously installed.
de_find_vertex() (ael)	Moves the cursor to the pin closest to the vertex selected within the select region and enters that point to the event that was previously installed.
de_fix_instances() (ael)	Fixes the position of an instance in the current representation and prevents design synchronization from re-positioning it.
de_flatten() (ael)	Removes a single level of hierarchy.
de_format_lib_cell_view_name() (ael)	Returns a formatted string ADS design name constructed from a given libraryName, cellName, and viewName.
de_free_instances() (ael)	Frees the position of an instance in the current representation and allows design synchronization to re-position it.
de_free_item() (ael)	Frees the data structure created by de_init_item() when it is no longer needed.
de_generate_symbol_ex() (ael)	This function generates a symbol into the given symbol context based on the design within the given DesignContext.
de_get_cells_in_library() (ael)	Returns an AEL list of string names of all cells in a given open library.
de_get_data_parm() (ael)	Returns a string, the value of a given data reference or a named parameter belonging to a default component.
de_get_open_libraries() (ael)	Returns an AEL list of the current open library names.
de_get_views_in_library_cell() (ael)	Returns an AEL list of string names of all views in a given library, cell. With the optional windowType argument you can limit the returned list of view names to a certain window type "LAYOUT_WINDOW, SCHEMATIC_WINDOW, or SYMBOL_WINDOW".
de_get_windows_with_same_library() (ael)	Returns an AEL list of window instances that are currently displaying designs stored in a given library.
de_group_edit_parameter value() (ael)	Modifies selected instances with the parameter name to the value given.
de_hide_or_display_component_name() (ael)	Toggles the instance component name in annotation to hide or display.
de_import_design() (ael)	Imports a foreign design format.
de_init_item() (ael)	Initializes the instance, readying it for placement.
de_insert_arrow() (ael)	Inserts an arrow draw as a polyline or polygon item.
de_insert_dimlin() (ael)	Inserts a dimension line.
de_instantiate() (ael)	Creates a new design from the selected group.
de_intersection() (ael)	Creates new polygons formed by the intersections of selected shapes (e.g., polygon, rectangle, circle, or path) on the same layer.
de_is_cellview_open() (ael)	Function to test if a cellview is open. Returns TRUE if the cellview is open, returns FALSE if it is not.

de_is_library_open() (ael)	Returns TRUE if any library with the given library name is open; Returns FALSE otherwise.
de_is_project_or_workspace_open() (ael)	This function returns TRUE if any ADS Workspace (ADS2011 and later) or ADS Project(used in ADS2009 Update 1 and prior releases) is open, otherwise FALSE.
de_last_view() (ael)	Recalls last view into the current window.
de_mirror_x() (ael)	Mirrors selected objects around the X-axis, given a reference point.
de_mirror_y() (ael)	Mirrors selected objects around the Y-axis, given a reference point.
de_miter_vertex() (ael)	Creates a mitered edge on a polygon or polyline.
de_modify_arc_resolution() (ael)	Modifies the resolution of an arc.
de_modify_break() (ael)	Converts selected polygons into polylines; that is, breaks a closed shape into an open shape.
de_modify_circle_radius() (ael)	Modifies the radius of a circle.
de_modify_explode() (ael)	Converts selected polygons and polylines into two-point polyline segments.
de_modify_join() (ael)	Joins selected polylines with coincident end points into a single polyline or polygon.
de_move() (ael)	Moves selected items in the current representation by a given amount.
de_move_annotation() (ael)	Moves parameter annotation of a selected instance to a new location.
de_move_break() (ael)	Moves selected instances in the current representation, breaking any wire or trace connection.
de_move_to_layer() (ael)	Moves selected objects in the current representation to the current entry layer.
de_net() (ael)	Creates a netlist report file.
de_new_datadisplay() (ael)	Sends the new window command to the data display server.
de_open_workspace() (ael)	Opens a specified workspace directory path.
de_oversize() (ael)	Creates a new oversized or undersized shape from the selected closed shapes in the current representation.
de_pan_window() (ael)	Re-centers the viewing window to the given point.
de_parse_lib_cell_view_name() (ael)	Returns a parsed libraryName, cellName, and viewName from a given ADS design name.
de_parts() (ael)	Generates a parts list for the current representation and stores in the specified file.
de_parts_option_add_exclusion_items() (ael)	Adds an Exclusion List to the parts list options.
de_parts_option_add_inclusion_items() (ael)	Adds an Inclusion List to the parts list options.
de_parts_option_check_bom() (ael)	Checks the BOM Flag.
de_parts_option_include_header() (ael)	Sets the parts list option to Include Header.
de_parts_option_set_attribute_columns() (ael)	Sets the User Attribute Columns parts list options.
de_parts_option_set_center_placement() (ael)	Sets the Component Placement X,Y Coordinates parts list options.
de_parts_option_set_delimeter() (ael)	Sets the Delimeter Character parts list options.
de_parts_option_set_hierarchical() (ael)	Sets Hierarchical Reporting parts list option.
de_parts_option_set_package_offset() (ael)	Sets the Package Offsets parts list option.
de_parts_option_sort_by_component() (ael)	Sets the Sort by Component Name parts list option.

de_paste_from_buffer() (ael)	Copies the contents of a buffer to the current representation.
de_place_design_template() (ael)	Inserts the design template setup with de_set_design_template() into current design.
de_place_item() (ael)	Places the instance that was initialized by de_init_item().
de_place_port() (ael)	Places a symbol pin in the current representation's symbol view.
de_place_unplaced() (ael)	Places an instance that has not yet been placed in one representation.
de_playback_macro() (ael)	Executes an AEL or DEM file.
de_plot() (ael)	Creates a plot of the active design in the current window and sends it to the default printer or plotter.
de_plot_to_file() (ael)	Creates a plot of the active design and saves it to a file in a format that can be sent to a printer or plotter.
de_pop_outof_instance() (ael)	Returns to the previous design before a de_push_into_instance() command.
de_push_into_instance() (ael)	Pushes into hierarchical instance reference.
de_refresh_view() (ael)	Redraws the screen of the active window.
de_release_simulator() (ael)	Cancels the simulation and makes the simulator license available for other users on the network.
de_remove_properties() (ael)	Removes properties of design group, port, or instance.
de_rename_cell() (ael)	Function to rename a cell to a specified new name. Returns TRUE if cell was successfully renamed, FALSE if not.
de_rename_cellview() (ael)	Function to rename a cellview to a specified new name. Returns TRUE if cellview was successfully renamed, FALSE if not.
de_restore_view() (ael)	Recalls specified view into the current window.
de_rotate() (ael)	Rotates selected items in the current representation around a given point.
de_rotate_90() (ael)	Rotates an item being placed 90 degrees clockwise around pin 1.
de_rotate_center() (ael)	Rotates selected items in the current representation around the center of gravity if more than one object is selected.
de_rotate_image() (ael)	Rotates an instance (element) being placed in a specified direction around pin 1.
de_save_all_designs() (ael)	Saves all designs in memory.
de_save_design_template() (ael)	Saves the schematic representation of a design (or a design template) as a design template.
de_scale() (ael)	Scales items in the current representation by a scale factor.
de_search_and_replace() (ael)	Finds all references (depending on refType), given a variable or item reference, and highlights the item.
de_select_all() (ael)	Selects everything matching the select filter in the current window.
de_select_all_force() (ael)	Selects everything on a specified layer matching the select filter in the current window.
de_select_all_on_layer() (ael)	Selects everything on a specified layer matching the select filter in the current window.
de_select_by_name() (ael)	Selects instances by component name or instance name.
de_select_item() (ael)	Toggles a selection of an item within the select region in the current representation.
de_select_range() (ael)	Selects all items in the current representation enclosed by a given window range.
de_select_unplaced() (ael)	Selects an unplaced instance in the current representation to place in the other representation (from schematic to layout or layout to schematic).
de_select_window() (ael)	Selects all items (matching the select filter) in the current representation enclosed by given window.

de_set_design_template() (ael)	Sets up the design template for insertion into current design by de_place_design_template().
de_set_edit_property() (ael)	Sets the data group, port, or instance for property editing.
de_set_edit_symbol_pin() (ael)	Sets the symbol pin whose attributes are to be edited.
de_set_edit_text() (ael)	Sets the text string in the current representation to be edited by de_edit_text_string().
de_set_hierarchy_instance_path() (ael)	Function to set the instance path for the design within given design context.
de_set_hierarchy_root() (ael)	Function to set the hierarchy root for the current design within a given design context.
de_set_inst_pin_order_property() (ael)	
de_set_item_id() (ael)	Sets the ID of the item that is either being readied to be placed (from de_init_item()) or is being placed (from de_edit_item()).
de_set_item_parameters() (ael)	Sets the parameter values for a given item that has been selected by the de_init_item() or de_edit_item().
de_set_move_annotation() (ael)	Selects an instance to have its annotation moved.
de_set_origin() (ael)	Resets the origin of a representation (resets the 0,0 point).
de_set_port() (ael)	Sets attributes of a symbol port (pin) before it is created for the current window.
de_set_simulation_dataset() (ael)	Sets the name of the dataset used for simulation.
de_set_simulation_host() (ael)	Sets the name of the host computer to be used for simulation.
de_set_swap_template_instance() (ael)	Sets the name of an item to swap within the function de_swap_instance().
de_shove() (ael)	Shove by a distance, vertices or whole polygons, whole text, and whole instances that are selected and lie on vector side of dividing line formed perpendicular to the vector.
de_show_equiv_inst() (ael)	Highlights the equivalent instance in another representation to a given instance.
de_snap() (ael)	Force the vertices of selected shapes to the nearest snap grid; forces pin 1 of selected instances to nearest grid point.
de_split() (ael)	Divides selected polygons, rectangles, circles, or paths into multiple shapes using a defined rectangular region.
de_split_tlin() (ael)	(For Layout only.) Splits a transmission line element into two of the same elements at a given point, adjusting the new elements length parameters.
de_step_and_repeat() (ael)	Copies and places selected items multiple times in rows and columns.
de_store_current_view() (ael)	Assigns a name to a view and stores view window coordinates.
de_stretch() (ael)	Stretches or moves an edge of a given shape in the current representation.
de_stretch_dimlin() (ael)	Stretches a dimension line.
de_stretch_tlin() (ael)	(For Layout only) Stretches (increases or decreases its length) a given transmission line element in the current representation.
de_swap_instances() (ael)	Replaces all selected instances in the current window with the instance set with the function de_set_swap_template_instance().
de_tap_tlin() (ael)	(For Layout only) Taps a transmission line element (MLIN or SLIN) in the current representation, and inserts a tee element (MTEE or STEE).
de_undo() (ael)	Undoes the effect of the last draw or edit command in the current window.
de_undo_vertex() (ael)	Undoes or removes the last vertex entered with the de_add_point() command.
de_union() (ael)	Creates new polygons formed by the union of selected shapes (e.g., polygon, rectangle, circle, or path) on the same layer.

db_update_parameters_ex() (ael)	Modifies parameter values.
de_update_tune_parameters() (ael)	Modifies tuned parameter values.
de_vertex_to_arc() (ael)	Converts a vertex on a shape to an arc.
de_view_all() (ael)	Expands the view window to view all data in the current window.
de_zoom_in_point() (ael)	Zooms in (double magnification) at the location specified on the current window.
de_zoom_in_scale() (ael)	Zooms in by factor specified on the current window.
de_zoom_out_point() (ael)	Zooms out (double magnification) at the location specified on the current window.
de_zoom_out_scale() (ael)	Zooms out by factor specified on the current window.
de_zoom_window() (ael)	Describes a region to display in the current window.

Component Definition Functions

Name	Description
create_compound_form() (ael)	Creates a new compound form and stores the form in the dictionary for the current simulator type.
create_constant_form() (ael)	Creates a new constant form and stores the form in the dictionary for the current simulator type.
create_form_set() (ael)	Creates a set of forms for use by the create_parm() function.
create_item() (ael)	Creates a new component definition and stores it with the current dictionary.
create_parm() (ael)	Creates a parameter definition for a component.
create_text_form() (ael)	Creates a form whose set of possible values are a set of strings.
de_define_library_palette() (ael)	Defines a palette group of ADS Components for a particular library that contains components from the same library or different libraries.
de_define_palette_group() (ael)	Defines a palette group.
de_update_design_definition_ex() (ael)	Updates the item definitions for all the instances within the given design context.
dm_create_cb()	Defines a callback function.
library_group() (ael)	Defines a new library group composed of the listed components.
prm() (ael)	Creates and returns a default parameter value.
set_design_choices() (ael)	Defines choices for characteristics allowed for parametric subnetwork components.
set_design_sub_choices() (ael)	Defines sub-choices for each characteristic choice for parametric subnetwork components.
set_design_type() (ael)	Sets the current type of design for group definitions.
set_netlist_info() (ael)	Sets special netlisting characteristics for current type of design.
set_simulator_type() (ael)	Sets the current simulator for ADS 2003A and earlier.

Component Definition Query Functions

Name	Description
dm_find_form_definition() (ael)	Returns a handle to a form definition given the form name and, optionally, a simulator ID.
dm_find_item_definition() (ael)	Returns an handle to an item definition given the item's name.
dm_first_parm_definition() (ael)	Returns the handle to the first parameter definition of an item, given a parameter definition handle as returned from dm_get_item_definition_attribute().
dm_get_design_class_code() (ael)	Returns a code representing the design class
dm_get_design_name() (ael)	Returns a string, the generic group name defined in AEL for a particular type of design.
dm_get_form_definition_attribute() (ael)	Returns the value of a form definition attribute, given a handle to the form definition.
dm_get_item_definition_attribute() (ael)	Returns the value of an item definition attribute given a handle to the item.
dm_get_parm_definition_attribute() (ael)	Returns a value of a parameter definition attribute, as defined with create_parm().
dm_get_simcode_from_designcode() (ael)	Returns a simcode, which is required by functions such as dm_find_form_definition(),
dm_index_parm_definition() (ael)	Returns a handle to the indexed parameter definition.
dm_next_parm_definition() (ael)	Returns the next handle to the parameter definition of an item.
dm_num_parm_definitions() (ael)	Returns the number of parameter definitions in a parameter list, given a parameter definition handle.

Construction Line Functions

Name	Description
db_get_const_line_layerid() (ael)	Returns the LayerId for the given construction line object.

Database Query and Manipulation Functions

Name	Description
db_clear_map() (ael)	Clears the transformation mapping established by db_setup_transform(), db_set_map(), or db_setup_map().
db_find_instance_ex() (ael)	Finds and returns an instance with a given instance name in the given DesignContext.
db_first_parm() (ael)	Returns a handle to a parameter or instance identified by an instance handle or the first parameter of a list of parameters identified by a parameter handle.
db_get_bbox_x1() (ael)	Returns an integer, the x component of the lower-left corner of an object's bounding box.
db_get_bbox_x2() (ael)	Returns an integer, the x component of the upper-right corner of an object's bounding box.
db_get_bbox_y1() (ael)	Returns an integer, the y component of the lower-left corner of an object's bounding box.
db_get_bbox_y2() (ael)	Returns an integer, the y component of the upper-right corner of an object's bounding box.
db_get_component_name() (ael)	Returns a string component name of a given DesignContext.
db_get_design_name() (ael)	Returns a string design name of a given DesignContext.
db_get_design_type() (ael)	Returns an integer design type of a given DesignContext.
db_get_instance_bbox() (ael)	Returns a handle to the bounding box of an instance.
db_get_instance_component_name() (ael)	Returns a string component name of a given instance.
db_get_instance_description() (ael)	Returns a string, the AEL component description string for an instance.

db_get_instance_item_definition() (ael)	Returns an item definition for a given instance.
db_get_instance_name() (ael)	Returns a string instance name of a given instance.
db_get_instance_parm() (ael)	Returns a real number, the nominal value of an instance parameter given the parameter name or index (position in list of parameters).
db_get_item_definition() (ael)	Returns an item definition for a given DesignContext.
db_get_location_angle() (ael)	Returns an integer, the angle given a location handle in .001 of a degree.
db_get_location_x() (ael)	Returns an integer, the x position given a location handle.
db_get_location_y() (ael)	Returns an integer, the y position given a location handle.
db_get_map() (ael)	Returns a handle to the current transformation mapping.
db_get_map_attribute() (ael)	Returns an attribute of a transformation mapping, as retrieved by db_get_map() .
db_get_node_wires() (ael)	Returns a handle to the segment representing a wire for a given node.
db_get_parm_attribute() (ael)	Returns an attribute of a parameter.
db_get_path_trace_bend_type() (ael)	Returns the corner (bend) type for a given path, trace, or wire shape.
db_get_path_trace_miter_radius() (ael)	Returns the miter cutoff percentage or curve radius for a given path, trace, or wire shape.
db_get_path_trace_width() (ael)	Returns the width in database units for a given path, trace, or wire shape.
db_get_pin_attribute()	Returns the attribute value of a given pin.
db_get_transform_angle() (ael)	Returns the angle of a transform object as returned from the function db_get_instance_attribute().
db_get_transform_mirror_x() (ael)	Returns the x axis mirror flag of a transform object, as returned from the function db_get_instance_attribute(),
db_get_transform_mirror_y() (ael)	Returns the y axis mirror flag of a transform object, as returned from the function db_get_instance_attribute()
db_get_transform_x() (ael)	Returns the x coordinate of an instance's transformation record.
db_get_transform_y() (ael)	Returns the y coordinate of an instance's transformation record.
db_get_x() (ael)	Returns an integer, the x coordinate in data base units.
db_get_y() (ael)	Returns an integer; the y coordinate in data base units.
db_next_parm() (ael)	Returns a handle to the next parameter of the instance.
db_num_parms() (ael)	Returns the number of parameters in a parameter list.
db_mks_to_uu_factor() (ael)	Returns a real value conversion factor for converting circuit units to database user units for a given DesignContext.
db_set_map() (ael)	Sets the current transformation mapping.
db_setup_map() (ael)	Modifies the current transformation mapping set
db_setup_transform() (ael)	Sets up the instance transformation mapping (rotation, translation and mirror).
db_transform_angle() (ael)	Transforms an angle value by rotation and mirror values from the current transformation as set by db_set_map(), db_setup_transformation(), or db_setup_map().
db_transform_bbox_ex() (ael)	Transforms the given bounding box to be the smallest possible bounding box which completely contains the transformed input bounding box.
db_transform_coord() (ael)	Transforms a single coordinate value using the current transformation as set by db_set_map(), db_setup_transformation(), or db_setup_map().

Design Functions

Name	Description
db_get_appropriate_base_name() (ael)	Returns a suitable filename for the given design context.
db_get_appropriate_base_name_from_design_name() (ael)	Returns a suitable filename from the given design name or component name.

DesignContext Functions

Name	Description
db_clear_context() (ael)	Clears everything in a given design context.
db_copy_context() (ael)	Copies the shapes, pins, application objects, instances, and properties from a given source design context into a given destination design context.
db_get_cell_name() (ael)	Returns the cell name of a design context.
db_get_context_bbox() (ael)	Returns the bounding box of the given DesignContext.
db_get_context_db_factor() (ael)	Returns a real value conversion factor for converting user units to database units for a given DesignContext.
db_get_context_unit_name() (ael)	Returns the string unit name used by the given DesignContext.
db_design_is_modified() (ael)	Returns TRUE if the given design context has been modified, otherwise returns FALSE.
db_get_dbu_to_uu_factor() (ael)	Returns the real value conversion factor for converting database units to user units for a given DesignContext.
db_get_library_name() (ael)	Returns the library name of a design context.
db_get_mks_to_uu_factor() (ael)	Returns the real value conversion factor for converting circuit MKS units to database user units for a given DesignContext.
db_get_view_name() (ael)	Returns the view name of a design context.
db_get_user_units_significant_digits() (ael)	Returns the integer number of user unit significant digits in the given DesignContext.
db_get_uu_to_dbu_factor() (ael)	Returns the real value conversion factor for converting user units to database units for a given DesignContext.
db_get_uu_to_mks_factor() (ael)	Returns the real value conversion factor for converting database user units to circuit MKS units for a given DesignContext
db_is_same_context() (ael)	Returns TRUE if the two contexts are equivalent.
db_save_design_without_prompting() (ael)	Saves the given design without notifying or prompting the user for permission.
de_create_new_layout_view() (ael)	Function to create a new layout view. Returns the design context to the created layout view.
de_create_new_schematic_view() (ael)	Function to create a new schematic view. Returns the design context to the created schematic view.
de_create_new_symbol_view() (ael)	Function to create a new symbol view. Returns the design context to the created symbol view.
de_current_context_is_valid() (ael)	Returns TRUE if there is a valid current DesignContext.
de_find_design_context_from_name() (ael)	Function to find the DesignContext from a design name and a rep type.
de_get_current_design_context() (ael)	Returns the current DesignContext.
de_get_design_context() (ael)	Returns the DesignContext of the given window.
de_get_design_context_from_name() (ael)	Function to find the DesignContext from a design name.
de_get_windows_with_same_context() (ael)	Returns a list of windows that are currently displaying the same design and view as the given window or DesignContext.
de_is_artwork_macro_context() (ael)	Returns TRUE if the given window or DesignContext is an artwork macro evaluation.
de_is_layout_context() (ael)	Returns TRUE if the given window or DesignContext is a layout window.

de_is_schematic_context() (ael)	Returns TRUE if the given window or DesignContext is a schematic window.
de_is_schematic_or_layout_context() (ael)	Returns TRUE if the given DesignContext or window instance's DesignContext is a schematic design context or is a layout design context; Returns FALSE otherwise.
de_is_symbol_context() (ael)	Returns TRUE if the given window or DesignContext is a symbol window.
de_show_context_in_new_window() (ael)	Opens the given design in a new window and returns the window.
de_window_has_valid_context() (ael)	Returns TRUE if the given window has a valid DesignContext.

Design Environment Query Functions

Name	Description
db_factor() (ael)	Returns a real value, a conversion factor from layout user units to meters.
de_ang_factor() (ael)	Returns a real number, a conversion factor to convert a value from simulator units to degrees.
de_current_design_type() (ael)	Returns the current design type (the design opened in the active window).
de_get_alternate_window() (ael)	Returns the alternate window instance handle and sets the current window to this alternate.
de_get_design_instances() (ael)	Returns a list of instance names of a given element, belonging to the named design.
de_get_file_names() (ael)	Returns a list of files with given file extension.
de_get_variable_names() (ael)	Returns a list of all variable names declared in a design.
de_get_window() (ael)	Returns the currently active window
de_invoke_help() (ael)	Accepts two arguments and returns a command to the help server
de_post_help() (ael)	Accepts one argument and returns a command to the help server
de_retrieve_version_info() (ael)	Returns a long string of product version information for the Design Environment.
de_short_design_name() (ael)	Strips off the fully qualified path to the design and any extension, returning just the name of the design.
de_version_number_int() (ael)	Returns the current version number as an integer.
de_window_is_open() (ael)	Determines the status of a window.
get_eqn_list() (ael)	Returns a list of variables and equations for the current design.
get_item_list() (ael)	Returns a string list of all components (instance names) matching a component name placed in the current design.
get_measurement_list() (ael)	Returns a string list of all measurement components active for the current design.

ExpressionContext Functions

- ***ExpressionContext Overview*** **(ael)**

Name	Description
db_setup_expression_context() (ael)	Setup an expression context to use for expression evaluation. The returned expression context defines the scope for expressions to be evaluated.
db_evaluate_param_expression() (ael)	Evaluate an expression in a given expression context.
db_evaluate_param_value() (ael)	Returns the evaluated expression value from a given parameter within the scope of the given ExpressionContext.
db_evaluate_param_value_no_expr() (ael)	Evaluates a parameter value without expression evaluation.
db_evaluate_inst_param_value() (ael)	Returns the evaluated expression value for an instance's parameter with a given parameter name within the scope of the given ExpressionContext.
db_evaluate_inst_param_value_no_expr() (ael)	Evaluates an instance parameter value with the given parameter name without expression evaluation.

File Handling Functions

Name	Description
ael_gfile_hasext() (ael)	Takes a string parameter and returns an integer that indicates where an extension exists in that string.
chdir() (ael)	Changes the current directory to the specified directory.
fclose() (ael)	Closes a file opened with fopen() command.
fflush() (ael)	Flushes buffers to disk when writing a file opened with fopen() for write or append.
fgets() (ael)	Reads a line from a file.
fopen() (ael)	Opens a file and returns a file identifier.
fprintf() (ael)	Print formatted text to a file.
fputs() (ael)	Writes a value to the file or device specified.
freopen() (ael)	Opens the named file for reading or writing, attaching an existing file handle to it.
remove() (ael)	Deletes a given file.
rename() (ael)	Renames a file.

Form Definition Functions

- ***Form Definition - Overview*** **(ael)**

Name	Description
dm_form_get_name() (ael)	Returns a string with the name of the given form.
dm_form_get_label() (ael)	Returns a string descriptive label of the given form.
dm_form_get_net_format() (ael)	Returns the netlist format string of the given form.
dm_form_get_disp_format() (ael)	Returns the display format string of the given form.
dm_form_get_class() (ael)	Returns the integer class code that represents the given form’s class type.
dm_form_get_attr() (ael)	Returns the form attribute code for the given form.
dm_form_get_parms() (ael)	Returns the list of parameter definitions for the form.
dm_form_is_constant_form() (ael)	Returns TRUE if the given form is a constant form.
dm_form_is_discrete() (ael)	Returns TRUE if the given form is discrete.
dm_get_string_form_class_selection() (ael)	Returns the list of string form names in the given form set.

Form Set Functions

- ***Form Set - Overview*** **(ael)**

Name	Description
dm_get_form_set_form_names() (ael)	Returns the list of string form names in the given form set.

GDSII Functions

Name	Description
de_get_gds_number() (ael)	Returns the GDS number corrosponding to a layer number from a GDSII layer mapping file.

Hierarchy Context Functions

- ***HierarchyContext Overview* (ael)**

Name	Description
db_create_hierarchy_context() (ael)	create a HierarchyContext for a given design with a given descent policy.
db_is_primitive_instance_in_hierarchy() (ael)	Returns TRUE if the instance is a primitive in the given HierarchyContext.
db_get_design_name_for_instance_in_hierarchy() (ael)	Returns the sub-design name for a given instance in the given HierarchyContext.
db_get_hierarchy_context_for_instance() (ael)	Returns the HierarchyContext for the given instance's sub-design in the given HierarchyContext.
db_get_design_context_from_hierarchy() (ael)	Returns the DesignContext from the given HierarchyContext.
db_is_hierarchy_context_at_root() (ael)	Returns TRUE if the given HierarchyContext is at the root.
db_get_hierarchy_context_for_parent() (ael)	Returns the HierarchyContext for the parent of the current design in the given HierarchyContext.
db_get_parent_inst_name_in_hierarchy() (ael)	Returns the instance name of the parent instance of the current design in the given HierarchyContext.

Highlight Selection Functions

Name	Description
db_highlight() (ael)	Function to highlight or unhighlight a given shape, instance, or symbol port.
db_highlight_net() (ael)	This function highlights or unhighlights a given Net within a given design context.
db_is_highlighted() (ael)	Returns TRUE if the given instance, symbol port, or shape is highlighted.
db_is_selected() (ael)	Returns TRUE if the passed in instance, symbol port, or shape is selected.
db_select() (ael)	Function to select or deselect an instance, symbol port, or shape.

Instance Iterator Functions

Name	Description
db_create_inst_iter() (ael)	Returns an iterator to the first instance belonging to a given design context.
db_inst_iter_enable_caching() (ael)	Function to make the iterator cache its list.
db_inst_iter_exclude_port_insts() (ael)	Limits instance iteration to exclude port instances for a given instance iterator.
db_inst_iter_get_instance() (ael)	Returns an instance object that is referenced by a given instance iterator
db_inst_iter_get_next() (ael)	Returns an iterator to the next instance after the given instance iterator.
db_inst_iter_is_valid() (ael)	Returns TRUE if the instance iterator references a valid instance object.
db_inst_iter_limit_region() (ael)	Function to limit the instance iteration to a region.
db_inst_iter_limit_selected() (ael)	Function to limit the iteration to selected instances.

Instance Functions

Name	Description
db_get_instance_cell_name() (ael)	Returns the string cell name of the given instance.
db_get_instance_description() (ael)	Returns the item description string for the given instance. Returns NULL if no item definition exists for the given instance.
db_get_instance_design_name() (ael)	Returns the string design name of the instance's master design. The design name string value returned from this function is in the format "<library name>:<cell name>:<view name>".
db_get_instance_library_name() (ael)	Returns the string library name of the given instance.
db_get_instance_owner_design() (ael)	Returns a design context to the instance's owner design. For a PSN, this is the Master design.
db_get_instance_placement_transform() (ael)	Returns the transform of the given instance.
db_get_instance_special() (ael)	Returns the special flag code of an instance.
db_get_instance_subcontext_transform() (ael)	Returns the subcontext transform of the given instance.
db_is_instance_deactivated() (ael)	Returns TRUE if instance is deactivated.
db_is_instance_deactivated_and_shorted() (ael)	Returns TRUE if the given instance is shorted.
db_is_instance_ground() (ael)	Returns TRUE if the given instance is a ground component. Returns FALSE if the given instance is not a ground component.

InstTerm Functions

Name	Description
db_get_inst_term_name() (ael)	Returns the name of a given instance terminal.
db_get_inst_term_number() (ael)	Returns the number of a given instance terminal.
db_get_inst_term_net() (ael)	Returns the Net of a given instance terminal.
db_get_inst_term_type() (ael)	Returns an integer terminal type of a given instance term object.
db_get_inst_term_instance() (ael)	Returns the instance of a given instance terminal.
db_is_inst_term_grounded() (ael)	Returns TRUE if the given instance terminal is grounded.

Term Functions

Name	Description
db_get_term_name() (ael)	Returns the name of a given terminal.
db_get_term_number() (ael)	Returns the number of a given terminal.
db_get_term_net() (ael)	Returns the Net of a given terminal.

InstTerm Iterator Functions

Name	Description
db_create_inst_term_iter() (ael)	Returns an instance terminal iterator from a given Net or Instance object.
db_inst_term_iter_is_valid() (ael)	Returns TRUE if the given instance terminal iterator is referencing a valid instance terminal object.
db_inst_term_iter_get_next() (ael)	Returns the next instance terminal iterator of the given instance terminal iterator.
db_inst_term_iter_get_inst_term() (ael)	Returns the instance terminal object that is referenced by the given instance terminal iterator.

InstPin Functions

Name	Description
db_get_inst_pin_angle_normalized() (ael)	Returns the angle of a given InstPin, normalized (zero degrees is right).
db_get_inst_pin_bbox() (ael)	Returns the bounding box of the given instance pin.
db_get_inst_pin_instance() (ael)	Returns the instance of the given instance pin.
db_get_inst_pin_inst_term() (ael)	Returns the InstTerm (instance terminal) of the given InstPin (instance pin) object.
db_get_inst_pin_net() (ael)	Returns the Net connected to the given instance pin.
db_get_inst_pin_snap_layerid() (ael)	Returns the LayerId for the snap layer of an instance pin.
db_get_inst_pin_snap_point() (ael)	Returns the snap coordinate point of a given instance pin object.
db_get_inst_pin_term_number() (ael)	Returns the term number of a given Instance Pin.
db_get_inst_pin_term_type() (ael)	Returns an integer terminal type of a given instance pin object.
db_is_inst_pin_connected_to_wire() (ael)	Returns TRUE if the given instance pin is connected to a wire. Returns FALSE if the given instance pin is not connected to a wire.

InstPin Iterator Functions

Name	Description
db_create_inst_pin_iter() (ael)	Returns an instance pin iterator of the given instance or of InstPins connected to the given Net.
db_inst_pin_iter_is_valid() (ael)	Returns TRUE if the instance pin iterator is referencing a valid instance pin.
db_inst_pin_iter_get_next() (ael)	Returns the next instance pin iterator from the given instance pin iterator.
db_inst_pin_iter_get_inst_pin() (ael)	Returns the instance pin object referred to by the given instance pin iterator.

Item Definition Functions

- ***Item Definition - Overview*** **(ael)**

Name	Description
dm_item_get_name() (ael)	Returns the string name of an item definition.
dm_item_get_label() (ael)	Returns the string descriptive label of an item definition.
dm_item_get_library_name() (ael)	Returns the string library name of an item definition.
dm_item_get_attr() (ael)	Returns the item attribute code of an item definition.
dm_item_get_ex_attr() (ael)	Returns the item extra attribute code of an item definition.
dm_item_get_priority() (ael)	Returns the integer priority for an item definition.
dm_item_get_prefix() (ael)	Returns the string instance prefix name for an item definition.
dm_item_get_netlist_format() (ael)	Returns the netlist format string for an item definition.
dm_item_get_netlist_data() (ael)	Returns the netlist data string for an item definition.
dm_item_get_symbol_format() (ael)	Returns the symbol format string for an item definition.
dm_item_get_symbol_name() (ael)	Returns the string symbol name for an item definition.
dm_item_get_icon_name() (ael)	Returns the string icon name for an item definition.
dm_item_get_dialog_name() (ael)	Returns the string dialog name for an item definition.
dm_item_get_dialog_data() (ael)	Returns the string dialog data for an item definition.
dm_item_get_artwork_type() (ael)	Returns the integer artwork type for an item definition.
dm_item_get_artwork_data() (ael)	Returns the string artwork data for an item definition.
dm_item_get_parms() (ael)	Returns the parameter definition list for an item definition.
dm_item_get_parm_names() (ael)	Returns a list of string parameter names for an item definition.
dm_item_is_bom_item() (ael)	Returns TRUE if an item has the isBom flag set.
dm_item_is_netlist_from_layout() (ael)	Returns TRUE if an item is marked to be netlisted from layout
dm_item_is_variable() (ael)	Returns TRUE if an item is marked as a variable (VAR) component
dm_item_is_unique() (ael)	Returns TRUE if an item is marked as a unique component
dm_item_is_sub_design() (ael)	Returns TRUE if an item is a sub-circuit
dm_item_is_port() (ael)	Returns TRUE if an item is a Port
dm_item_no_special_characters() (ael)	Returns TRUE if the item disallows special characters in the component's instance name

Layer and LayerId Functions

- ***Overview - LayerId* (ael)**

Name	Description
db_get_layerid() (ael)	Returns a new LayerId object of a given layer/purpose pair, specified by their layer name and purpose name, of a given design context.
db_layerid() (ael)	Returns a new LayerId object of a given layer/purpose pair, specified by the layer number and optional purpose number.
db_get_layerid_name() (ael)	Returns a LayerId name for a specified LayerId within a given design context.
db_get_layerid_names() (ael)	Returns an ordered list of LayerId names for a given design context.
db_get_layerid_index() (ael)	Returns the LayerId integer index into the design context's layer list given a LayerId.
db_get_layout_layerid_name() (ael)	Returns the layout LayerId name for the given design context from the given LayerId.
db_get_layout_layerid_names() (ael)	Returns an ordered list of the layout LayerId names for the given design context.
db_create_layer() (ael)	Creates a new physical layer for the given design context, given the layer's name and number.
db_find_layerid_by_name() (ael)	Returns a LayerId of a given design context that has a given

	LayerId name.
db_find_layout_layerid_by_name() (ael)	Finds and returns the layout LayerId for the given design context from the given LayerId name.
db_find_layer_name_by_number() (ael)	Finds and returns the name of a layer in a design context with the given layer number.
db_find_layer_number_by_name() (ael)	Finds and returns the number of a layer in a design context with the given layer name.
db_get_layer_binding() (ael)	Returns the layer binding string for a given layer number in the given design context's library's technology.
db_get_layer_number() (ael)	Returns the layer number for a given LayerId.
db_get_purpose_number() (ael)	Returns the purpose number for a given LayerId.
db_get_layer_process_role() (ael)	Returns the layer process role for a given layer.
db_set_layer_process_role() (ael)	Sets the layer process role for a given layer.
db_is_layerid_invisible() (ael)	Returns TRUE if the given LayerId is marked invisible, FALSE otherwise.
db_is_layerid_protected() (ael)	Returns TRUE if the given LayerId is marked protected, FALSE otherwise.
db_set_layerid_invisible() (ael)	Function to mark a LayerId to be invisible or visible.
db_set_layerid_protected() (ael)	Function to mark a LayerId to be protected or unprotected.
db_get_layerid_alpha() (ael)	Returns the alpha integer value for a given LayerId.
db_get_layerid_fill_mode() (ael)	Returns the fill mode integer identifier for a given LayerId.
db_get_layerid_fill_pattern() (ael)	Returns the fill Patern integer identifier for a given LayerId.
db_get_layerid_line_style() (ael)	Returns the line style integer identifier for a given LayerId.
db_get_layerid_rgb() (ael)	Returns the RGB color integer value for a given LayerId.
db_set_layerid_alpha() (ael)	Sets an alpha integer value (0-255) for a particular LayerId in a given library or a given design context's library's technology. Returns none.
db_set_layerid_fill_mode() (ael)	Sets the fill mode for a particular LayerId in a given library or a given design context's library's technology. Returns none.
db_set_layerid_fill_pattern() (ael)	Sets an integer fill pattern for a particular LayerId in a given library or a given design context's library's technology. Returns none.
db_set_layerid_line_style() (ael)	Sets the line style for a particular LayerId in a given library or a given design context's library's technology. Returns none.
db_set_layerid_rgb() (ael)	Sets a RGB color for a particular LayerId in a given library or a given design context's library's technology. Returns none.
db_get_layerid_for_comp_name() (ael)	Returns the LayerId for the component name annotation used for component instances from a given design context.
db_get_layerid_for_inst_name() (ael)	Returns the LayerId for the instance name annotation used for component instances from a given design context.
db_get_layerid_for_parameters() (ael)	Returns the LayerId for the parameter annotation used for component instances from a given design context.
db_get_layerid_for_symbol_body() (ael)	Returns the LayerId for the symbol body from a given design context.
db_get_layerid_for_symbol_text() (ael)	Returns the LayerId for the symbol text from a given design context.
db_get_layerid_for_schematic_wires() (ael)	Returns the LayerId for the schematic wires from a given schematic design context.

List Management Functions

Name	Description
append() (ael)	Appends a new list to the end of an existing list.
car() (ael)	Returns the first item from a given list.
cdr() (ael)	Returns the remainder of the list, after the first item is removed.
cons() (ael)	Constructs a list cell from the member value to be returned by car() and the next element value to be returned by cdr().
delete_nth() (ael)	Deletes an item in a list, given an integer index into the list and an existing list.
insert() (ael)	Returns a new list with an inserted item, given an existing list and item to insert into the beginning of the list.
insert_nth() (ael)	Returns a new list with an inserted item, given an index into the list, an existing list and an item to insert into the list.
list() Function (ael)	Creates a list of items.
listlen() (ael)	Returns an integer representing the length of a list
member() (ael)	Tests whether an item is a member of a list.
nth() (ael)	Takes a list and an integer index into the list.
nthcdr() (ael)	Takes a list, and an integer index into the list.
remov() (ael)	Returns a new list with all the items copied from the original list except for the item specified in the first parameter.
repla() (ael)	Replaces an item in a list.
sort_list() (ael)	Returns a new list with members of a given list in sorted order.

Math Functions For AEL

Name	Description
abs() Function (ael)	Returns the absolute value of a real number or an integer.
acos() Function (ael)	Returns the inverse cosine, in radians, of a real number or complex number.
acosh() Function (ael)	Returns the inverse hyperbolic cosine of an integer, real, or complex number.
acot() Function (ael)	Returns the inverse hyperbolic cosine of an integer, real, or complex number.
acoth() Function (ael)	Returns the inverse hyperbolic cotangent of an integer, real, or complex number.
asin() Function (ael)	Returns the inverse sine, in radians, of a real number, an integer, or complex number.
asinh() Function (ael)	Returns the inverse hyperbolic sine of an integer, real, or complex number.
atan() Function (ael)	Returns the inverse tangent, in radians, of a real number, ±/2, an integer, or complex number.
atan2() Function (ael)	Returns the arc tangent of the rectangular coordinates y and x.
atanh() Function (ael)	Returns the inverse hyperbolic tangent of an integer, real, or complex number.
ceil() Function (ael)	Given a real number, returns the smallest integer not less than its argument
chr() Function (ael)	Returns the character representation of an integer.
cint() Function (ael)	Given a noninteger real number, returns a rounded integer value.
cmplx() Function (ael)	Given two real numbers representing the real and imaginary components of a complex number, returns a complex number.
conj() Function (ael)	Returns the conjugate of a complex number.
convBin() Function (ael)	Returns a binary string of an integer with n-digits.
convHex() Function (ael)	Returns a hexidecimal string of an integer with n-digits.

convOct() Function (ael)	Returns an octal string of an integer with n-digits
cos() Function (ael)	Returns the cosine of a real number (in radians).
cosh() Function (ael)	Returns the hyperbolic cosine of an integer, real, or complex number.
cot() Function (ael)	Returns the cotangent of an integer, real, or complex number.
coth() Function (ael)	Returns the hyperbolic cotangent of an integer, real, or complex number.
dB() Function (ael)	Converts a voltage ratio to decibels.
dBm() Function (ael)	Converts a voltage to decibels referenced to milliwatt.
deg() Function (ael)	Converts an angle from radians to degrees.
exp() Function (ael)	Given an integer or real number as an exponent, returns e (~2.7183) raised to that exponent.
fix() Function (ael)	Takes a real number argument, truncates it, and returns an integer value.
float() Function (ael)	Converts an integer to a floating decimal number. Returns a real (floating-point) number.
floor() Function (ael)	Returns the largest integer not more than its argument from a real number.
im() Function (ael)	Returns a real value, the imaginary component of a complex number.
imag() Function (ael)	Returns a real value, the imaginary component of a complex number.
int() Function (ael)	Returns the largest integer not greater than its real-number argument.
ln() Function (ael)	Formerly log() in Series IV.
log() Function (ael)	Returns the base 10 logarithm of an integer or real number.
log10() Function (ael)	Returns the base 10 logarithm of an integer or real number.
mag() Function (ael)	Returns the absolute/magnitude value of an integer, real, or complex number.
max2() Function (ael)	Returns the larger value of two numeric values, or NULL if parameters are invalid.
min2() Function (ael)	Returns the lesser value of two numeric values, or NULL if parameters are invalid.
num() Function (ael)	Returns an integer that represents an ASCII numeric value of the first character in the specified string.
phase() Function (ael)	Returns a real number that represents the phase in degrees of the argument.
phasedeg() Function (ael)	Returns a real number that represents the phase in degrees of the argument.
phaserad() Function (ael)	Returns a real number that represents the phase in radians of the argument.
polar() Function (ael)	Converts polar magnitude and angle into a complex number.
pow() Function (ael)	Returns an integer or real number raised to given power.
rad() Function (ael)	Converts angle from degrees to radians.
re() Function (ael)	Returns a real number, the real part of a complex value.
real() Function (ael)	Returns a real number, the real part of a complex value.
round() Function (ael)	Returns an integer, a real number rounded to nearest integer value.
sgn() Function (ael)	Returns the integer sign of an integer or real number, as either 1 or -1.
sin() Function (ael)	Returns the sine of a real number (in radians).
sinc() Function (ael)	Returns a real number that represents sin(parm)/parm in radians.
sinh() Function (ael)	Returns the hyperbolic sine of an integer, real, or complex number.
sqrt() Function (ael)	Returns the square root of a positive integer or real number.
step() Function (ael)	A step function that returns 0, 0.5, or 1.
tan() Function (ael)	Returns the tangent of a real number (in radians).

tanh() Function (ael)	Returns the hyperbolic tangent of an integer, real, or complex number.
xor() Function (ael)	Returns an integer that represents the exclusive OR between arguments.

Net Functions

Name	Description
db_get_net_name() (ael)	Returns the name of the given Net.
db_is_net_named_by_user() (ael)	Returns TRUE if the given Net was named by the user.
db_is_net_grounded() (ael)	Returns TRUE if the given Net is grounded.

Net Iterator Functions

Name	Description
db_create_net_iter() (ael)	Returns an iterator to the first Net belonging to a given design context.
db_net_iter_is_valid() (ael)	Returns TRUE if the given net iterator references a valid net object
db_net_iter_get_next() (ael)	Returns the next net iterator of the given net iterator.
db_net_iter_get_net() (ael)	Returns the net object that is referenced by the given net iterator.

Netlist Functions

Name	Description
de_netlist_interpret_format_string() (ael)	Given a netlist format string, this function will return a string representing an outline of the structure of the format string.
de_netlist_create() (ael)	Generates a netlist for the given DesignContext.
de_netlist_get_text() (ael)	Returns the full text of a netlist as an AEL string.
de_netlist_save() (ael)	Saves the given netlist object's text to an indicated file.

Object Functions

Name	Description
db_is_instance() (ael)	Returns TRUE if the passed in object is an instance.
db_is_shape() (ael)	Returns TRUE if the passed in object is a shape.
db_is_pin() (ael)	Returns TRUE if the passed in object is a pin.

Preference Functions

Name	Description
db_set_curve_radius() (ael)	Sets the curve radius for paths and traces used when adding paths and traces with curved corners in the given design context.
db_set_miter_cutoff() (ael)	Sets the miter cutoff angle used when adding paths and traces with mitered corners in the given design context.
db_set_path_corner() (ael)	Sets the corner type used when adding paths and traces in the given design context.
db_set_path_width() (ael)	Sets the width for paths and traces used when adding paths and traces in the given design context.
de_get_preference() (ael)	Returns the value of a current preference, the preference setting.
de_read_preferences() (ael)	Reads a preference file from disk, re-setting the preferences in the current window.
de_refresh_layers() (ael)	Refreshes the layer information in the windows displaying the current design representation.
de_set_annotation_font() (ael)	Sets the font for instance parameter annotation for the current window.
de_set_annotation_height() (ael)	Sets the text height used for component parameter annotation.

de_set_annotation_id_layer() (ael)	Sets the layer number for instance name (ID) parameter annotation for the current window.
de_set_annotation_name_layer() (ael)	Sets the layer number for instance parameter annotation for the current window.
de_set_annotation_parameters_layer() (ael)	Sets the layer number for instance parameter annotation for the current window.
de_set_annotation_precision() (ael)	Sets the display precision of real numbers displayed in schematic annotation.
de_set_annotation_rows() (ael)	Sets the number of rows for instance parameter annotation for the current window.
de_set_arc_radius() (ael)	Set the radius for any arcs created using the de_vertex_to_arc() command
de_set_background_color() (ael)	Sets the background color of the drawing area in the current window.
de_set_backup_count() (ael)	Sets the number of edits that must be performed before a backup file is made.
de_set_coord_entry_popup() (ael)	Sets whether the Coordinate Entry dialog will be displayed for draw and place commands.
de_set_curve_radius() (ael)	Sets the curve radius in the current window used for paths and traces with rounded corners.
de_set_drag_move() (ael)	Sets the ability to drag and move objects with the left mouse button.
de_set_drag_move_size() (ael)	Sets the minimum distance an object must be dragged before the object is moved.
de_set_drag_move_units() (ael)	Sets the units of the drag move.
de_set_dse_start() (ael)	Sets the starting instance for design synchronization in the current representation.
de_set_dual_placement() (ael)	Sets the mode so that when enabled, causes any component placed in one representation to be automatically placed in the other representation.
de_set_foreground_color() (ael)	Sets the sketch color of the active window.
de_set_global_db_factor() (ael)	Sets the simulator units to layout user unit conversion factor used during rendering of AEL artwork objects.
de_set_grid_color() (ael)	Sets the color of the visible grid in the current window.
de_set_grid_display_type() (ael)	Sets the grid display to dots or dotted line for the current window.
de_set_grid_snap() (ael)	Sets grid snap increments in X and Y direction and turns grid snapping for the current window on or off.
de_set_grid_snap_mode() (ael)	Turns grid snapping for the current window on or off.
de_set_grid_snap_type() (ael)	Sets the snap type to one or more of grid, pin, or vertex for the current window.
de_set_highlight_color() (ael)	Sets the highlight color for the current window (used for errors, and show commands).
de_set_layer() (ael)	Sets the entry layer for the current layer for the current window.
de_set_major_grid_display() (ael)	Sets the major grid display factor.
de_set_minor_grid_display() (ael)	Sets the minor grid display factor to xFactor multiple of the snap grid in X and yFactor multiple of the snap grid in Y.
de_set_miter_cutoff() (ael)	Sets the miter cutoff angle for the path and trace command.
de_set_miter_length() (ael)	Sets the miter length.
de_set_oversize()	Selects or rejects polygons on one layer (inLayer1) in relation to edges of polygons on another layer (inLayer2) if TouchCondition is true.
de_set_path_corner() (ael)	Sets the corner type used for paths and traces in the current window.
de_set_path_width() (ael)	Sets the width for paths and traces used by the de_add_path() and

	de_add_trace() entry commands for the current window.
de_set_pin_color() (ael)	Sets the color used for instance pins for the current window.
de_set_pin_size() (ael)	Sets the size used to display instance pins of the current window.
de_set_pin_size_units() (ael)	Sets the units for the pin size set by de_set_pin_size().
de_set_pin_snap() (ael)	Sets the maximum distance at which the cursor snaps to a pin when snap type includes PREF_SNAP_TO_PIN.
de_set_pin_snap_units() (ael)	Sets the units for the pin snap distance set by de_set_pin_snap().
de_set_place_popup_mode() (ael)	Sets mode for displaying the component Parameter dialog box for the current window.
de_set_place_popup_on_zero_parm() (ael)	Sets mode for the current window for displaying the Component Parameter dialog box for components with no parameters.
de_set_plot_pin_names() (ael)	Turns the display of pin names on or off.
de_set_plot_pin_numbers() (ael)	Turns the plotting of pin numbers on and off for the current window.
de_set_plot_pins() (ael)	Turns plotting of connected pins on or off for the current window.
de_set_plotting_depth() (ael)	Sets the hierarchical level at which component instances in Layout are drawn in outline.
de_set_port_size() (ael)	Sets the size of ports in the Layout representation.
de_set_port_size_units() (ael)	Sets the units for the port size set by de_set_port_size().
de_set_preference() (ael)	Sets the value of any preference.
de_set_reroute_wires() (ael)	Sets wire or trace editing route mode for the current window.
de_set_resolution_for_arc() (ael)	Sets the resolution that is used for approximating an arc
de_set_rotation_increment() (ael)	Sets the step increment used by the de_rotate() command for the current window.
de_set_route_around_annot() (ael)	When the Wire Route command is in progress, this mode determines if the wire should route around or through annotations.
de_set_scale() (ael)	Set the X- and Y-scale factors used by the de_scale() command in the current window.
de_set_select_box_size() (ael)	Sets the select region size for the current window.
de_set_select_box_units() (ael)	Sets the units for the select region size, set by de_set_select_box_size(), at the current window.
de_set_select_color() (ael)	Sets the color used to display selected objects for the current window.
de_set_select_filter() (ael)	Sets the select filter mask for the current window.
de_set_select_inside_polygon() (ael)	Sets the selection mode.
de_set_select_point_size() (ael)	Sets the size of the selected vertices in the current window.
de_set_select_point_size_units() (ael)	Sets the units for the selected vertices in the current window.
de_set_self_intersect() (ael)	Sets the polygon self intersection checking for the current window.
de_set_shape_entry_mode() (ael)	Sets the entry mode in the current window for polygons and polylines to orthogonal or non-orthogonal.
de_set_step_and_repeat() (ael)	Sets variables for the Step and Repeat command.
de_set_tap_length() (ael)	Sets the length (w3 parameters) of the tee element produced when a transmission line element is tapped with the de_tap_tlin() command.
de_set_tee_color() (ael)	Sets the color of schematic tees (interconnect dots) for the current window.
de_set_tee_size() (ael)	Sets the size of tees (interconnect dots) used in schematics (for the current window).
de_set_tee_size_units() (ael)	Sets the type of units of the tee size.
de_set_text_absolute() (ael)	Sets absolute characteristics for text.
de_set_text_angle() (ael)	Sets the angle in which text is placed for the current window.
de_set_text_font() (ael)	Sets the font type when placing text for the current window.

de_set_text_height() (ael)	Sets the height for placed text (in the current window).
de_set_text_justification() (ael)	Sets the default text justification.
de_set_text_string() (ael)	Sets the text string used by the de_add_text() command for the current window.
de_set_trace_sim_mode() (ael)	Sets the trace simulation mode for the current window.
de_set_trace_single_elem() (ael)	Sets the name of the element used when trace simulation is set to a single element.
de_set_trace_tech() (ael)	Sets the technology type for converting/simulating traces to microstrip, stripline, or PCB for the current window.
de_set_trace_traverse() (ael)	Sets the trace traversal mode.
de_set_undo_edit_count() (ael)	Sets the number of edit commands that can be undone.
de_touch() (ael)	Selects or rejects polygons on one layer (inLayer1) in relation to edges of polygons on another layer (inLayer2) if TouchCondition is true.
de_write_preferences() (ael)	Writes the current preference settings of the active window to a file.
ly_find_layer_by_name() (ael)	Retrieves a layer number given its name, for active layer set.
ly_find_layer_name_by_num() (ael)	Retrieves a layer name by its layer number, for the active layer set.

Parameter Definition Functions

- ***Parameter Definition - Overview* (ael)**

Name	Description
dm_parm_get_name() (ael)	Returns the name of the parameter.
dm_parm_get_label() (ael)	Returns the label of the given parameter definition.
dm_parm_get_attr() (ael)	Returns the attribute code of the given parameter definition.
dm_parm_get_unit() (ael)	Returns the unit code of the given parameter definition.
dm_parm_get_formset_name() (ael)	Returns the string name of the form set used for the given parameter definition.
dm_parm_get_defvalue() (ael)	Returns the default parameter value for the given parameter definition.
dm_get_parm_definition_forms() (ael)	Returns a list of form definitions for the given parameter definition.

Parameter Value Functions

- ***Parameter Value - Overview* (ael)**

Name	Description
db_get_param_name() (ael)	Returns the string parameter name for the given parameter value.
db_get_param_value_code() (ael)	Returns an integer parameter value code that represent's the parameter value form's class type.
db_is_param_repeatable() (ael)	Returns TRUE if the parameter is repeatable.
db_param_value_uses_string_form() (ael)	Returns TRUE if the parameter is string valued.
db_param_value_uses_constant_form() (ael)	Returns TRUE if the parameter is constant valued.
db_param_value_uses_compound_form() (ael)	Returns TRUE if the parameter is compound valued.
db_get_param_form_name() (ael)	Returns the string form name of the given parameter value's form.
db_set_param_form_name() (ael)	Sets the given parameter value's form to be the form of the given string form name.
db_get_param_string_value() (ael)	Returns the string value of the given parameter value.
db_set_param_string_value() (ael)	Set the string value of the given parameter value.
db_is_param_displayed() (ael)	Returns TRUE if the parameter value is marked to be displayed in the current view.
db_set_param_displayed() (ael)	Function to mark a parameter value to be displayed or not displayed in the current view.

Parameter Iterator Functions

- ***Parameter Iterator - Overview*** **(ael)**

Name	Description
db_create_param_iter() (ael)	Returns an iterator to the first parameter in the collection.
db_param_iter_is_valid() (ael)	Returns TRUE if the parameter iterator is referencing a valid parameter.
db_param_iter_get_next() (ael)	Returns the next parameter iterator from the given parameter iterator.
db_param_iter_get_first() (ael)	Returns the first parameter iterator from the given parameter iterator.
db_param_iter_flatten_repeats() (ael)	Function to flatten the repeated parameters while iterating.
db_param_iter_get_param() (ael)	Returns the parameter value referred to by the given parameter iterator.
db_param_iter_get_parm_def() (ael)	Returns the parameter definition referred to by the given parameter iterator.
db_param_iter_is_repeatable() (ael)	Returns TRUE if the parameter referenced by the given parameter iterator is a top-level parameter that is repeatable.
db_param_iter_is_repeated() (ael)	Returns TRUE if the parameter referenced by the given parameter iterator is a repeat of a top-level repeatable parameter.
db_param_iter_get_repeat_index() (ael)	Returns the integer repeated index of the parameter referenced by the given parameter iterator
db_param_iter_get_sub_parameters() (ael)	Returns a parameter iterator to the sub-parameter values of the parameter referenced by the given parameter iterator
db_param_iter_find_form() (ael)	Returns the form definition for the parameter referenced by the given parameter iterator.
db_param_iter_get_display_value() (ael)	Returns a string with the displayed value of the parameter referenced by the given parameter iterator.
db_param_iter_get_netlist_value() (ael)	Returns a string with the netlist value of the parameter referenced by the given parameter iterator.
db_param_iter_find_by_name() (ael)	Returns a new parameter iterator for the parameter with the given parameter name within the given parameter iterator's list of parameters.
db_find_inst_param_by_name() (ael)	Returns a new parameter iterator for the parameter with the given parameter name within the given parameter iterator's list of parameters.
db_param_iter_find_by_index() (ael)	Returns a new parameter iterator for the parameter with the given parameter integer index within the given parameter iterator's list of parameters.
db_param_iter_find_by_name_and_repeat() (ael)	Returns a new parameter iterator for the parameter with the given parameter name and the given parameter repeat index within the given parameter iterator's list of parameters.
db_param_iter_find_by_index_and_repeat() (ael)	Returns a new parameter iterator for the parameter with the given parameter index and the given parameter's repeat index within the given parameter iterator's list of parameters.

Pin Functions

Name	Description
db_create_pin() (ael)	Creates a new pin in a given context and returns the new pin object.
db_edit_pin_attributes() (ael)	Returns the modified pin for a given pin and the given modified pin attributes.
db_get_pin_angle_normalized() (ael)	Returns the angle of a given Pin, normalized (zero degrees is right).
db_get_pin_name() (ael)	Returns the name of a given Pin object.
db_get_pin_term() (ael)	Returns the term of a given Pin object.
db_get_pin_term_name() (ael)	Returns the term name of a given Pin object's term.
db_get_pin_term_number() (ael)	Returns the term number of a given Pin object's term.
db_get_pin_term_type() (ael)	Returns an integer terminal type of a given Pin object.
db_get_pin_bbox() (ael)	Returns the bounding box of a given pin.
db_get_pin_angle() (ael)	Returns the angle of a given pin.
db_get_pin_snap_point() (ael)	Returns the snap coordinate point of a Pin object's.
db_get_pin_snap_layerid() (ael)	Returns the LayerId for the snap layer of a pin.
db_is_pin_selected() (ael)	Returns TRUE if the passed in pin is selected.
db_select_pin() (ael)	Function to select or deselect a pin.

Pin Iterator Functions

Name	Description
db_create_pin_iter() (ael)	Returns a pin iterator from a given Net or DesignContext.
db_pin_iter_is_valid() (ael)	Returns TRUE if the pin iterator is referencing a valid pin.
db_pin_iter_get_next() (ael)	Returns the next pin iterator from the given pin iterator.
db_pin_iter_get_pin() (ael)	Returns the pin object referred to by the given pin iterator.
db_pin_iter_limit_selected() (ael)	Function to limit the pin iteration to selected pins.
db_pin_iter_limit_region() (ael)	Function to limit the pin iteration to a region.

Primitive Polygon Functions

- ***Primitive Polygon Overview*** **(ael)**

Name	Description
db_create_primitive_polygon() (ael)	Returns a new primitive polygon from a given list of outer boundary points, and a given optional list of point index to edge arc bulge factors.
db_create_shape_from_primitive_polygon() (ael)	Creates and returns a new database shape in a given context on a given layerId from a given source primitive polygon.
db_get_shape_primitive_polygon() (ael)	Returns a primitive polygon for a given shape. Works for all shapes.
db_get_shape_control_polygon() (ael)	Returns a primitive polygon containing the shape's control points. This polygon is guaranteed to not contain arcs or holes.
db_get_primitive_polygon_outer() (ael)	Returns an outer primitive polygon of a given primitive polygon.
db_get_primitive_polygon_num_points() (ael)	Returns the number of points in the outer boundary of the polygon.
db_get_primitive_polygon_point() (ael)	Returns the coordinate point at a specified point index in the given primitive polygon.
db_get_primitive_polygon_num_holes() (ael)	Returns the integer number of holes in the given primitive polygon.
db_get_primitive_polygon_hole() (ael)	Returns a new primitive polygon equivalent to the specified hole at a specified hole index in the given primitive polygon.
db_add_hole_to_primitive_polygon() (ael)	Returns a new primitive polygon of a given primitive polygon with a given hole polygon added into it.
db_get_primitive_polygon_with_linked_holes() (ael)	Returns a new primitive polygon with holes linked from a given primitive polygon.
db_primitive_polygon_has_arcs() (ael)	Returns TRUE if a given primitive polygon contains arcs; returns FALSE otherwise.
db_get_primitive_polygon_with_arcs_removed() (ael)	Returns a new primitive polygon with arcs removed from a given primitive polygon and given arc resolution to use for arc removal.
db_get_primitive_polygon_edge_is_arc() (ael)	Returns TRUE if the edge following the given point index in a given primitive polygon is an arc segment; returns FALSE otherwise.
db_get_edge_arc_angle() (ael)	Returns the arc sweep angle for a given primitive polygon's given point index of an edge that is an arc.
db_get_edge_arc_bulge() (ael)	Returns the arc bulge factor for a given primitive polygon given point index of an edge.
db_get_edge_arc_center() (ael)	Returns the center coordinate point for a given primitive polygon's given point index of an edge that is an arc.

Property Functions

- ***Property Overview*** **(ael)**

Name	Description
db_add_property() (ael)	Adds a property to an ADS database objecta
db_remove_property() (ael)	Removes a property from an ADS database object.
db_remove_properties_with_prefix() (ael)	Removes all properties from an ADS database object.
db_get_property() (ael)	Get the property’s double, string, or long value from an ADS database object.
db_get_property_as_string() (ael)	Get the property’s double, string, or long value returned as a string value from an ADS database object.

Property Iterator Functions

- ***Property Iterator - Overview* (ael)**

Name	Description
db_create_prop_iter() (ael)	Returns an iterator to the first property of the given database object.
db_prop_iter_is_valid() (ael)	Returns TRUE if the property iterator is referencing a valid property.
db_prop_iter_get_next() (ael)	Returns the next property iterator from the given property iterator.
db_prop_iter_get_name() (ael)	Returns the name of the property referenced the given property iterator.
db_prop_iter_get_type() (ael)	Returns the integer value type of the iterator's current property.
db_prop_iter_get_value() (ael)	Returns the string, long or real value, if the optional type argument is not specified.
db_prop_iter_remove_prop() (ael)	Removes the given iterator's current property and returns an iterator to the next property.

Selection Functions

Name	Description
db_cycle_select_item() (ael)	Select one item at a given location in the DesignContext.
db_deselect_all() (ael)	Deselect all items in a DesignContext.
db_deselect_area() (ael)	Deselect all items in a given rectangular area of a DesignContext.
db_get_objects_selected_at() (ael)	Returns a list of shape objects, instance objects, and/or port objects that are selected at a particular location in a DesignContext.
db_get_selection_tolerance() (ael)	Returns the user's selection tolerance for a given window.
db_get_shape_selected_points() (ael)	Returns a list of the selected points' coordinates for a given shape.
db_pick_instance_at() (ael)	Returns the instance object which would be selected if the user clicked at the location.
db_pick_object_at() (ael)	Returns the shape object, instance object, or port object which would be selected if the user clicked at the location.
db_pick_shape_at() (ael)	Returns the shape object which would be selected if the user clicked at the location.
db_select_all() (ael)	Select all items in a DesignContext.
db_select_all_on_layerid() (ael)	Function to select or deselect all shapes on a particular LayerId in the given design context.
db_select_area() (ael)	Select all items in a given rectangular area of a DesignContext.

Shape Functions

Name	Description
db_get_arc_angle() (ael)	Returns the sweep angle of a given arc shape.
db_get_arc_center() (ael)	Returns the center coordinate of a given arc shape.
db_get_arc_start() (ael)	Returns the start coordinate of a given arc shape.
db_get_ellipse_center() (ael)	Returns the center coordinate of a given arc shape.
db_get_ellipse_x_radius() (ael)	Returns the x radius of a given ellipse shape.
db_get_ellipse_y_radius() (ael)	Returns the y radius of a given ellipse shape.
db_get_shape_bbox() (ael)	Returns a handle to the bounding box of a shape.
db_get_shape_layer() (ael)	Returns the layer number of a given shape.
db_get_shape_layerid() (ael)	Returns the LayerId for the given shape.
db_get_shape_net() (ael)	This functions returns the Net associated with the given shape. Returns NULL if there is no Net associated with the given shape.
db_get_shape_text_string() (ael)	Returns a string containing the text within the given text shape or annotation shape.
db_get_shape_type_description() (ael)	Returns a string that describes the type of the given shape.
db_get_text_absolute() (ael)	Function returns the boolean (TRUE,FALSE) absolute flag from a given text or annotation shape object.
db_get_text_angle() (ael)	Returns the text angle in 0.001 degree units of the given text or annotation shape object.
db_get_text_font_name() (ael)	Function returns the string font name for a given text or annotation shape object.
db_get_text_height() (ael)	Function returns the text height in database units for a given text or annotation shape object.
db_get_text_justification() (ael)	Function to return the justification attribute code from a text or annotation object.
db_get_text_origin() (ael)	Returns the origin's (anchor point) coordinate in database units of the given text or annotation shape object.
db_is_cell_name_display() (ael)	Returns TRUE if the given annotation shape object is a cell name display.
db_is_inst_name_display() (ael)	Returns TRUE if the given annotation shape object is an instance name display.
db_shape_has_selected_points() (ael)	Returns TRUE if a given shape has any point(s) selected. If the shape has no points selected, then FALSE is returned.
db_shape_is_annotation() (ael)	Returns TRUE if the given shape is an annotation object.
db_shape_is_arc() (ael)	Returns TRUE if the given shape is an arc.
db_shape_is_circle() (ael)	Returns TRUE if the given shape is a circle.
db_shape_is_const line() (ael)	Returns TRUE if the given shape is a construction line.
db_shape_is_dot() (ael)	Returns TRUE if the given shape is a dot.
db_shape_is_ellipse() (ael)	Returns TRUE if the given shape is an ellipse.
db_shape_is_path() (ael)	Returns TRUE if the given shape is a path.
db_shape_is_polygon() (ael)	Returns TRUE if the given shape is a polygon.
db_shape_is_polyline() (ael)	Returns TRUE if the given shape is a polyline.
db_shape_is_rectangle() (ael)	Returns TRUE if the given shape is a rectangle.
db_shape_is_text() (ael)	Returns TRUE if the given shape is a text object.
db_shape_is_text_or_annotation() (ael)	Returns TRUE if the given shape is a text or annotation object.
db_shape_is_wire() (ael)	Returns TRUE if the given shape is a wire.
db_shape_is_wire_label() (ael)	Returns TRUE if the given shape is a wire label.
db_shape_is_trace() (ael)	Returns TRUE if the given shape is a trace.
db_shape_is_wire_or_trace() (ael)	Returns TRUE if the given shape is a wire or trace.

Shape Iterator Functions

Name	Description
db_create_shape_iter() (ael)	Returns an iterator to the first shape belonging to a given design context.
db_shape_iter_enable_caching() (ael)	Function to make the iterator cache its list.
db_shape_iter_exclude_invisible_layers() (ael)	Function to exclude invisible layers in the shape iteration.
db_shape_iter_exclude_protected_layers() (ael)	Function to exclude protected layers in the shape iteration.
db_shape_iter_get_next() (ael)	Returns an iterator to the next shape after the given shape iterator.
db_shape_iter_get_shape() (ael)	Returns a shape object that is referenced by a given shape iterator
db_shape_iter_include_invisible_layers() (ael)	Function to include invisible layers in the iteration
db_shape_iter_include_protected_layers() (ael)	Function to include protected layers in the iteration
db_shape_iter_is valid() (ael)	Returns TRUE if the shape iterator references a valid shape object.
db_shape_iter_limit_layer() (ael)	Function to limit the shape iteration to a layer.
db_shape_iter_limit_layerid() (ael)	Function to limit the shape iteration to a particular LayerId.
db_shape_iter_limit_purpose() (ael)	Function to limit the shape iteration to a purpose layer.
db_shape_iter_limit_region() (ael)	Function to limit the shape iteration to a region.
db_shape_iter_limit_selected() (ael)	Function to limit the iteration to selected shapes.

Simulation Functions

Name	Description
db_get_controller_design_name() (ael)	Returns the string design name for the controller design of the given design context.
db_get_datadisp_name() (ael)	Returns the string (DDS) Data Display file name for the given design context.
db_get_dataset_name() (ael)	Returns the string dataset name for the given design context.
db_get_hierarchy_policy_name() (ael)	Returns the string name of the HierarchyPolicy that will be used for simulation of the given design context.
db_get_opendds_option() (ael)	Returns TRUE if the given design context is specified to automatically open the Data Display after simulation. Returns FALSE otherwise.
db_has_explicit_hierarchy_policy() (ael)	Returns TRUE if the given design context has a specific HierarchyPolicy name set that will be used for simulation of the design.
db_set_controller_design_name() (ael)	Function to set the string design name of the controller design for the given design context.
db_set_datadisp_name() (ael)	Function to specify the custom (DDS) Data Display name to use for the given design context.
db_set_dataset_name() (ael)	Function to specify the custom dataset name to use for the given design context.
db_set_hierarchy_policy_name() (ael)	Function to set the HierarchyPolicy name to use specifically for simulation of the given design context.
db_set_opendds_option() (ael)	Function to specify for the given design if the Data Display window should open automatically after simulation of the design occurs.
de_get_default_hierarchy_policy_name() (ael)	Returns the name of the default HierarchyPolicy that is used for simulating designs.
de_set_default_hierarchy_policy_name() (ael)	Sets the name of the default HierarchyPolicy that is used for simulating designs.

Simulator Command Functions

Name	Description
clear_server() (ael)	Clears the busy status state of the active server (simulator).
create_server() (ael)	Creates a handle to a server process that can be controlled through AEL.
de_analyze() (ael)	Invokes the simulator and sends it a new, updated netlist of the current design.
de_analyze_tune() (ael)	Invokes the simulator, sends the simulator a new, updated netlist of the current design, and prepares the simulator to receive de_tune() commands.
de_close_all_datadisplay() (ael)	Hides all data display windows that have been opened.
de_open_datadisplay() (ael)	Opens a saved data display file.
de_restore_all_datadisplay() (ael)	Opens all data display windows that have been hidden, using de_close_all_datadisplay().
de_restore_status() (ael)	Opens the closed status server window.
de_tune() (ael)	Activates a tune analysis after updating parameters to the values specified in the paramList.
de_tune_deinit() (ael)	Terminates the tuning operation in the simulator.
de_update_optimization_values() (ael)	Requests updated optimization/yield results from any optimization or yield analysis performed by the current server (simulator).
invoke_process() (ael)	Function to invoke a child process.
pop_message_handler() (ael)	Restores a saved message handler for a server to the state previous to the last call to push_message_handler().
push_message_handler() (ael)	Installs a message handler function to receive messages from a server.
send_server_command() (ael)	Sends a command to the server identified by server_handle.
send_server_data() (ael)	Sends data statement to the server identified by server_handle.
send_server_interrupt() (ael)	Sends an interrupt to the current server identified by server_handle.
send_server_kill() (ael)	Sends the special kill command ("KILL") to the server identified by server_handle.
server_running() (ael)	Returns the status of the server identified by serverHandle
start_server() (ael)	Causes the server identified by serverHandle to be spawned.

String Functions

Name	Description
fmt() (ael)	Converts a float to a string.
fmt_tokens() (ael)	Creates a string from a list of tokens.
index() Function (ael)	Returns the position of the first occurrence of a string or character in a string.
leftstr() (ael)	Returns the left portion of a string.
midstr() (ael)	Returns a piece of a string (the dissected string).
parse() (ael)	Parses a string into tokens, where each token is delimited by a blank, tab, or operator.
parse_blank() (ael)	Parses a string of tokens separated by spaces or tabs into a list of strings.
rightstr() (ael)	Returns the right portion of a string value.
sprintf() Function (ael)	Format text values into a string.
strcasecmp() (ael)	Compares two strings, ignoring differences in case between them.
strcat() Function (ael)	Appends two or more strings to the end of the first, resulting in a single string.
strcmp() (ael)	Compares two strings.
stripstr() (ael)	Returns a string with leading and trailing blanks and tabs removed.
strlen() (ael)	Returns the length of a string, as an integer.
tolower() (ael)	Converts a string to lowercase and return the converted string.
toupper() (ael)	Converts a string to uppercase and return the converted string.
val() (ael)	Returns a real number, the numeric value of an ASCII string.

Term Iterator Functions

db_create_term_iter() (ael)	Returns an iterator to the first Term belonging to a given design context or given net.
db_term_iter_is_valid() (ael)	Returns TRUE if the given terminal iterator is referencing a valid terminal object.
db_term_iter_get_next() (ael)	Returns the next terminal iterator of the given terminal iterator.
db_term_iter_get_term() (ael)	Returns the terminal object that is referenced by the given terminal iterator.

Transaction Functions

Name	Description
db_commit_transaction() (ael)	Commit the given transaction.
db_create_transaction() (ael)	Create a new transaction object for a given DesignContext.
db_rollback_transaction() (ael)	Rollback the given transaction.
db_transaction_is_empty() (ael)	Returns TRUE if no changes have occured since the start of the transaction.

User Interface Functions

Name	Description
add_menu() (ael)	Adds an item, menuItemName, to the user-defined menu and associates the AEL function aelFuncName with the item.
add_separator() (ael)	Adds a separator in the menu.
api_dlg_unmanage() (ael)	Unmanages the specified dialog.
api_get_current_window() (ael)	Returns the handle to the currently active window.
api_get_graph_color_index_by_name() (ael)	Returns the index of the color for the given color name
api_get_graph_color_name_by_index() (ael)	Returns the name of the color for the given color index
api_get_graph_pattern_index_by_name() (ael)	Returns the index of the pattern for the given pattern name
api_get_graph_pattern_name_by_index() (ael)	Returns the name of the pattern for the given pattern index
api_get_window_graph() (ael)	Returns a handle to the graphics area of the given window.
api_set_current_window() (ael)	Sets a window to be currently active.
api_set_current_window_by_seq_num() (ael)	Makes the window instance having the given sequence number as the current window.
check_user_menu() (ael)	This function can be used to determine if a user menu is being used by another application.
clear_user_menu() (ael)	Removes and clears all entries in the user-defined menu.
de_data_dialog() (ael)	Displays a scrollable multiline text editor.
de_info() (ael)	Displays an information dialog box.
de_install_get_xy() (ael)	Installs a user-defined event handler for prompting in the current window.
de_install_get_xy_pair() (ael)	Installs a user-defined event handler for prompting in the current window.
de_list_select() (ael)	Pops up a list selection dialog box.
de_open_hierarchy_dialog() (ael)	Opens dialog box to display hierarchy of current design.
de_open_info_dialog() (ael)	Opens dialog box for displaying information.
de_print_info() (ael)	Prints the contents of the following dialog boxes: data, new features, info, hierarchy, editor, DSE schematic status, DSE layout status, and check rep report.
de_prompt() (ael)	Invokes the prompt dialog box in the current window with user-supplied prompt string
de_question() (ael)	Invokes question dialog box in the current window.
set_user_menu_label() (ael)	Sets the label for the user menu.

Utility Functions for AEL

Name	Description
ael_file_exists() (ael)	Returns TRUE if the given file path string or directory path string exists. Returns FALSE if the given file path string or directory path string does not exist.
ael_is_file_writable() (ael)	Returns TRUE if the given directory path string or file path string is writable. Returns FALSE if the given directory path string or file path string does not have write permissions.
arg() (ael)	Returns the nth argument passed into a function.
arg_list() (ael)	Returns the arguments passed to a function as a list.
array_lowerBound() (ael)	Returns the lower bound of the specified dimension in an AEL array.
array_size() (ael)	Returns the size of each dimension in an AEL array.
array_type() (ael)	Returns the type of elements in an AEL array.
array_upperBound() (ael)	Returns the upper bound of the specified dimension in an AEL array.

call() (ael)	Invokes or calls a function, gives a function value and an argument list (as returned from arglist() or list()).
call_depth() (ael)	Returns the depth of the active call stack.
check_syntax() (ael)	Interprets a string as an AEL command and returns whether or not the syntax is valid.
chmod() (ael)	Changes the mode of the specified file.
convert_array() (ael)	Returns a new AEL array created from an existing array, but with the elements of the existing array converted to the specified type.
date_time() (ael)	Returns the date and time as a formatted string.
de_error_bell() (ael)	Sounds the error bell.
de_exit() (ael)	Exits Advanced Design System.
de_get_env() (ael)	Retrieves environment file name.
de_set_error_bell() (ael)	Enables or disables the error bell (on or off).
de_simple_dialog_cancel_cb() (ael)	A general purpose cancel callback.
de_valid_name() (ael)	Returns TRUE or FALSE
de_set_warning_bell() (ael)	Enables or disables the warning bell (on or off).
de_warning_bell() (ael)	Sounds the warning bell.
delete_word() (ael)	Deletes a word from the current vocabulary, or from a specified vocabulary.
error() (ael)	Reports an error.
evaluate() (ael)	Evaluates an AEL expression and returns the result.
execute() (ael)	Interprets text as an AEL program.
expandenv() (ael)	Returns a string with the environment variables and EEsof configuration variables
file_loaded() (ael)	Tests to see whether an AEL file has been read.
filedate() (ael)	Returns an integer that represents a time stamp of a file.
find_word() (ael)	Returns the address of an AEL word if it exists
find_word_voc() (ael)	Returns the address of a vocabulary in which a specified AEL word is found if it exists
fix_path() (ael)	Adds the appropriate backslashes needed in a string to handle control characters correctly.
format_date_time() (ael)	Returns the date and time as a formatted string
generic_netlist_cb() (ael)	This function allows the creation of a list of lists.
get_dir_files() (ael)	Returns an AEL list of files found in a specified directory.
getcwd() (ael)	Returns a string representing the path name of the current working directory.
getenv() (ael)	Returns a string representing the configuration variable value.
geterror() (ael)	Returns a string that represents the last error message that occurred.
getppid() (ael)	Returns an integer that represents the parent process ID of the current process.
getsysenv() (ael)	Returns a string representing a system environment variable value.
identify_value() (ael)	Returns a single string representing the value(s) of any valid AEL object.
is_complex() (ael)	Identifies whether a given value is a complex number.
is_dir() (ael)	Determines if a given path is a directory.
is_file() (ael)	Identifies whether a given value is a file identifier.
is_function() (ael)	Identifies whether a given value is a function.
is_function_defined() (ael)	Identifies whether a given value is a defined function.
is_integer() (ael)	Identifies whether a given value is an integer number.
is_list() (ael)	Identifies whether a given value is a list.
is_real() (ael)	Identifies whether a given value is a real number.

is_string() (ael)	Identifies whether a given value is a string.
is_type() (ael)	Identifies whether a given value is of a given type.
is_voc() (ael)	Identifies whether a given value is a vocabulary reference.
list_undefined() (ael)	Interprets a string as an AEL command and returns an AEL list of undefined identifiers in that command.
load() (ael)	Loads an AEL file and interprets its statements.
mkdir() (ael)	Creates a directory.
num_args() (ael)	Returns the number of arguments passed into a function.
offset_array() (ael)	Creates a new array by copying an existing AEL array
on_error() (ael)	Sets the AEL function to be called in case of an error.
pcb_get_form_value() (ael)	Retrieves the value of a parameter that is a constant form.
pcb_get_mks() (ael)	Retrieves the value of a parameter in MKS (unscaled) units.
pcb_get_parm_type() (ael)	Returns parameter type.
pcb_get_string() (ael)	Retrieves the value of a parameter that is a constant form.
pcb_set_form_value() (ael)	Sets the value of a parameter that is a constant form.
pcb_set_mks() (ael)	Sets the value of a parameter.
pcb_set_string() (ael)	Sets the value of a parameter that is a constant form.
rename_word() (ael)	Finds an AEL word and renames it.
resize_array() (ael)	Creates a new array by copying an existing AEL array with resizing of the specified dimension.
setenv() (ael)	Sets the value of a named configuration variable.
sleep() (ael)	Sets an idle time for the program.
start_timer() (ael)	Records the current timestamp internally.
system() (ael)	Executes a system command and returns the command exit status.
tmpnam() (ael)	Generates the name of a unique temporary file.
total_elapsed_time() (ael)	Returns the time elapsed since the last start_timer() was called as an AEL list of length 2.
validate_name() (ael)	Verifies that a string is a valid program file name
warning() (ael)	Issues a warning message which is then printed in the warning dialog.
what_col() (ael)	Returns the column number of the caller of the current function as an integer.
what_file() (ael)	Returns the file name of the caller of the current function as a string
what_function() (ael)	Takes an optional integer parameter to indicate what level of the function stack to extract.
what_line() (ael)	Returns the line number of the caller of the current function as an integer.

Command Functions

This section describes each Command function in detail. Command functions are listed in alphabetical order.

db_a	
db_add_arc() (ael) *db_add_arc1()* (ael) *db_add_arc2()* (ael) *db_add_arc3()* (ael) *db_add_arc4()* (ael) *db_add_circle()* (ael) *db_add_construction_line()* (ael)	*db_add_path()* (ael) *db_add_point()* (ael) *db_add_polygon()* (ael) *db_add_polyline()* (ael) *db_add_rectangle()* (ael) *de_create_polygon()* (ael) *db_end()* (ael)
de_a-b	
de_activate() (ael) *de_add_arc()* (ael) *de_add_arc1()* (ael) *de_add_arc2()* (ael) *de_add_arc3()* (ael) *de_add_arc4()* (ael) *de_add_circle()* (ael) *de_add_construction_line()* (ael) *de_add_path()* (ael) *de_add_point()* (ael) *de_add_polygon()* (ael) *de_add_polyline()* (ael)	*de_add_property()* (ael) *de_add rectangle()* (ael) *de_add text()* (ael) *de add trace()* (ael) *de add vertex()* (ael) *de add wire()* (ael) *de add wire label()* (ael) *de bom()* (ael) *de boolean logical()* (ael) *de break connection()* (ael)
de_c	
de_cell_exists() (ael) *de_cellview_exists()* (ael) *de_change_annotation_layer()* (ael) *de_check_rep_options()* (ael) *de_chop()* (ael) *de_clear_dc_annotation()* (ael) *de_clear_highlighting()* (ael) *de_close_all()* (ael) *de_close_window()* (ael) *de_close_workspace_without_prompting()* (ael) *de_connect()* (ael) *de_convert_path_to_trace()* (ael) *de convert to polygon()* (ael) *de convert trace to path()* (ael) *de convert traces to instances()* (ael) *de copy()* (ael) *de_copy_cell()* (ael) *de_copy_cellview()* (ael) *de copy file()* (ael) *de copy to buffer()* (ael) *de copy to layer()* (ael) *de create window()* (ael) *de crop()* (ael)	
de_d	
de_dc_annotation() (ael) *de_deactivate()* (ael) *de_define_edge_area_port()* (ael) *de_define_npport()* (ael) *de_define_port()* (ael) *de_delete()* (ael) *de_delete_all_orphaned_instances()* (ael) *de_delete_cell()* (ael) *de_delete_cellview()* (ael)	*de_deselect_window()* (ael) *de difference()* (ael) *de draw arc()* (ael) *de draw arc1()* (ael) *de draw arc2()* (ael) *de draw arc3()* (ael) *de draw arc4()* (ael) *de draw circ()* (ael) *de draw point()* (ael)

de_delete_view() (ael) *de_deselect_all()* (ael) *de_deselect_all_force()* (ael) *de_deselect_by_name()* (ael)	*de draw port()* (ael) *de draw rect()* (ael) *de draw text()* (ael) *de dse l2s()* (ael) *de dse s2l()* (ael)
de_e	
de_edit_annotation_attribute() (ael) *de_edit_item()* (ael) *de_edit_path_trace()* (ael) *de_edit_symbol_pin()* (ael) *de_edit_text_attribute()* (ael) *de_edit_text_string()* (ael)	*de empty()* (ael) *de end()* (ael) *de end command()* (ael) *de end edit item()* (ael) *de export design()* (ael)
de_f-i	
de_fill() (ael) *de_find_arc_center()* (ael) *de_find_line_center()* (ael) *de_find_pin()* (ael) *de_find_vertex()* (ael) *de_fix_instances()* (ael) *de_flatten()* (ael) *de_format_lib_cell_view_name()* (ael) *de_free_instances()* (ael) *de_free_item()* (ael) *de_get_cells_in_library()* (ael) *de_get_data_parm()* (ael) *de group edit parameter value()* (ael)	*de_get_open_libraries()* (ael) *de_get_views_in_library_cell()* (ael) *de_get_windows_with_same_library()* (ael) *de hide or display component name()* (ael) *de import design()* (ael) *de init item()* (ael) *de insert arrow()* (ael) *de insert dimlin()* (ael) *de instantiate()* (ael) *de intersection()* (ael) *de_is_cellview_open()* (ael) *de_is_library_open()* (ael) *de_is_project_or_workspace_open()* (ael)
de_l-m	
de_last_view() (ael) *de_mirror_x()* (ael) *de_mirror_y()* (ael) *de_miter_vertex()* (ael) *de_modify_arc_resolution()* (ael) *de_modify_break()* (ael) *de_modify_circle_radius()* (ael)	*de_modify_explode()* (ael) *de modify join()* (ael) *de move()* (ael) *de move annotation()* (ael) *de move break()* (ael) *de move to layer()* (ael)
de_n-p	
de_net() (ael) *de_new_datadisplay()* (ael) *de_open_workspace()* (ael) *de_oversize()* (ael) *de_pan_window()* (ael) *de_parse_lib_cell_view_name()* (ael) *de_parts()* (ael) *de_parts_option_add_exclusion_items()* (ael) *de_parts_option_add_inclusion_items()* (ael) *de_parts_option_check_bom()* (ael)	*de_parts_option_include_header()* (ael) *de_parts_option_set_attribute_columns()* (ael) *de parts option set center placement()* (ael) *de parts option set delimeter()* (ael) *de parts option set hierarchical()* (ael) *de parts option set package offset()* (ael) *de parts option sort by component()* (ael) *de paste from buffer()* (ael) *de place design template()* (ael) *de place item()* (ael) *de place port()* (ael) *de place unplaced()* (ael) *de playback macro()* (ael) *de plot()* (ael) *de plot to file()* (ael) *de pop outof instance()* (ael) *de push into instance()* (ael)
de_r	
de_refresh_view() (ael) *de_release_simulator()* (ael) *de_remove_properties()* (ael) *de_rename_cell()* (ael) *de_rename_cellview()* (ael)	*de_restore_view()* (ael) *de rotate()* (ael) *de rotate 90()* (ael) *de rotate center()* (ael) *de rotate image()* (ael)
de_s	
de_save_all_designs() (ael) *de_save_design_template()* (ael)	*de set origin()* (ael) *de set port()* (ael)

db_add_arc1_ex()

Creates a standalone arc in the given design context, that is created with a user defined starting point coordinate, circumference point coordinate, and an end point coordinate for the arc.

Syntax

```
arcShape = db_add_arc1_ex(context, layerid, x1,y1, x2,y2, x3,y3 );
```

Where,

- *context* is a design context to add the arc into.
- *layerid* is the layerid to add the arc into.
- *x1*,*y1* is the x,y coordinate starting point location of the arc in user units.
- *x2*,*y2* is the x,y coordinate circumference point location of the arc in user units.
- *x3*,*y3* is the x,y coordinate end point location of the arc in user units.

Example

```
decl sx=100.0, sy=0.0;
decl cirx=150.0, ciry=50.0;
decl ex=100.0, ey=100.0;
decl context = de_get_current_design_context();
decl layerid = db_get_layerid(context, "Metal1");
decl shape = db_add_arc1_ex(context, layerid, sx,sy, cirx,ciry, ex,ey);
```

Return Value(s)

The new arc shape object if it was successfully created, NULL otherwise.

Version Introduced

ADS 2011

Version Compatible

ADS 2011

See also

db_add_arc_ex() (ael)
db_add_arc2_ex() (ael)
db_add_arc3_ex() (ael)
db_add_arc4_ex() (ael)

Where Used (ael)

Schematic, Layout

db_add_arc1()

Creates a standalone arc in the given design context, that is created with a user defined starting point coordinate, circumference point coordinate, and an end point coordinate for the arc.

Syntax

```
decl arcShape = db_add_arc1(context, layerid, x1,y1, x2,y2, x3,y3 );
```

Where

- *context* is a design context to add the arc into.
- *layerid* is the layerid to add the arc into.
- *x1*,*y1* is the x,y coordinate starting point location of the arc in user units.
- *x2*,*y2* is the x,y coordinate circumference point location of the arc in user units.
- *x3*,*y3* is the x,y coordinate end point location of the arc in user units.

Example

```
decl sx=100.0, sy=0.0;
decl cirx=150.0, ciry=50.0;
decl ex=100.0, ey=100.0;
decl context = de_get_current_design_context();
decl layerid = db_get_layerid(context, "Metal1");
decl shape = db_add_arc1(context, layerid, sx,sy, cirx,ciry, ex,ey);
```

Return Value(s)

The new arc shape object if it was successfully created, NULL otherwise.

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_add_arc() (ael)
db_add_arc2() (ael)
db_add_arc3() (ael)
db_add_arc4() (ael)

Where Used (ael)

Schematic, Layout

db_add_arc2_ex()

This function creates a clockwise or counterclockwise standalone arc in the given design context, that is created with a user defined starting point coordinate, center point coordinate, and an approximate end point coordinate for the arc.

Syntax

```
arcShape = db_add_arc2_ex(context, layerid, x1,y1, x2,y2, x3,y3, cw
[,thickness] );
```

Where,

- *context* is a design context to add the arc into.
- *layerid* is the layerid to add the arc into.
- *x1*,*y1* is the x,y coordinate starting point location of the arc in user units.
- *x2*,*y2* is the x,y coordinate center point location of the arc in user units.
- *x3*,*y3* is the x,y coordinate approx. end point location of the arc in user units.
- *cw* is an integer representing the direction of the arc:
  1 = clockwise
  0 = counterclockwise
- *thickness* is an optional integer line thickness to draw the arc with. Possible line thickness values are:
  1 = thin, default
  2 = medium
  3 = thick

Example

```
decl sx=0.0, sy=0.0;
decl centerx=0.0, centery=50.0;
decl ex=0.0, ey=100.0;
decl context = de_get_current_design_context();
decl layerid = db_get_layerid(context, "cond", "drawing");
decl shape = db_add_arc2_ex(context, layerid, sx,sy, centerx,centery, ex,ey, 1);
```

Return Value(s)

The new arc shape object if it was successfully created, NULL otherwise.

Version Introduced

ADS 2011

Version Compatible

ADS 2011

See also

db_add_arc_ex() (ael)
db_add_arc1_ex() (ael)
db_add_arc3_ex() (ael)
db_add_arc4_ex() (ael)

Where Used (ael)

Schematic, Layout

db_add_arc2()

This function creates a clockwise or counterclockwise standalone arc in the given design context, that is created with a user defined starting point coordinate, center point coordinate, and an approximate end point coordinate for the arc.

Syntax

```
decl arcShape = db_add_arc2(context, layerid, x1,y1, x2,y2, x3,y3, cw
[,thickness] );
```

Where

- *context* is a design context to add the arc into.
- *layerid* is the layerid to add the arc into.
- *x1*,*y1* is the x,y coordinate starting point location of the arc in user units.
- *x2*,*y2* is the x,y coordinate center point location of the arc in user units.
- *x3*,*y3* is the x,y coordinate approx. end point location of the arc in user units.
- *cw* is an integer representing the direction of the arc:
  1 = clockwise
  0 = counterclockwise
- *thickness* is an optional integer line thickness to draw the arc with. Possible line thickness values are:
  1 = thin, default
  2 = medium
  3 = thick

Example

```
decl sx=0.0, sy=0.0;
decl centerx=0.0, centery=50.0;
decl ex=0.0, ey=100.0;
decl context = de_get_current_design_context();
decl layerid = db_get_layerid(context, "cond", "drawing");
decl shape = db_add_arc2(context, layerid, sx,sy, centerx,centery, ex,ey, 1);
```

Return Value(s)

The new arc shape object if it was successfully created, NULL otherwise.

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_add_arc() (ael)
db_add_arc1() (ael)
db_add_arc3() (ael)
db_add_arc4() (ael)

Where Used (ael)

Schematic, Layout

db_add_arc3_ex()

This function creates a clockwise or counterclockwise standalone arc in the given design context, that is created with a user defined starting point coordinate, center point coordinate, and an arc angle in degrees for the arc.

Syntax

```
arcShape = db_add_arc3_ex(context,layerid,x1,y1,x2,y2,sweepAngle,cw);
```

Where,

- *context* is a design context to add the arc into.
- *layerid* is the layerid to add the arc into.
- *x1*,*y1* is the x,y coordinate starting point location of the arc in user units.
- *x2*,*y2* is the x,y coordinate center point location of the arc in user units.
- *sweepAng* is a positive real value arc angle in degrees.
- *cw* is an integer representing the direction of the arc:
  1 = clockwise
  0 = counterclockwise

Example

```
decl sx=0.0, sy=0.0;
decl centerx=0.0, centery=50.0;
decl sweepAng = 180.0;
decl context = de_get_current_design_context();
decl layerid = db_get_layerid(context, "Metal2");
decl shape = db_add_arc3_ex(context,layerid,sx,sy,centerx,centery,sweepAng,1);
```

Return Value(s)

The new arc shape object if it was successfully created, NULL otherwise.

Version Introduced

ADS 2011

Version Compatible

ADS 2011

See also

db_add_arc_ex() (ael)
db_add_arc1_ex() (ael)

db_add_arc2_ex() (ael)
db_add_arc4_ex() (ael)

Where Used (ael)

Schematic, Layout

db_add_arc3()

This function creates a clockwise or counterclockwise standalone arc in the given design context, that is created with a user defined starting point coordinate, center point coordinate, and an arc angle in degrees for the arc.

Syntax

```
arcShape = db_add_arc3(context,layerid,x1,y1,x2,y2,sweepAngle,cw);
```

Where

- *context* is a design context to add the arc into.
- *layerid* is the layerid to add the arc into.
- *x1*,*y1* is the x,y coordinate starting point location of the arc in user units.
- *x2*,*y2* is the x,y coordinate center point location of the arc in user units.
- *sweepAng* is a positive real value arc angle in degrees.
- *cw* is an integer representing the direction of the arc:
  1 = clockwise
  0 = counterclockwise

Example

```
decl sx=0.0, sy=0.0;
decl centerx=0.0, centery=50.0;
decl sweepAng = 180.0;
decl context = de_get_current_design_context();
decl layerid = db_get_layerid(context, "Metal2");
decl shape = db_add_arc3(context,layerid,sx,sy,centerx,centery,sweepAng,1);
```

Return Value(s)

The new arc shape object if it was successfully created, NULL otherwise.

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_add_arc() (ael)
db_add_arc1() (ael)
db_add_arc2() (ael)
db_add_arc4() (ael)

Where Used (ael)

Schematic, Layout

db_add_arc4_ex()

This function creates a clockwise or counterclockwise standalone arc in the given design context, that is created with a user defined starting point coordinate, center point coordinate, and a chord length for an arc angle expressed in terms of circumference.

Syntax

```
arcShape = db_add_arc4_ex(context,layerid,x1,y1,x2,y2,chordLength,cw);
```

Where,

- *context* is a design context to add the arc into.
- *layerid* is the layerid to add the arc into.
- *x1*,*y1* is the x,y coordinate starting point location of the arc in user units.
- *x2*,*y2* is the x,y coordinate center point location of the arc in user units.
- *chordLength* is the arc angle expressed in terms of circumference.
- *cw* is an integer representing the direction of the arc:
  1 = clockwise
  0 = counterclockwise

Example

```
decl sx=0.0, sy=0.0;
decl cx=0.0, cy=50.0;
decl chordLength= 100.0;
decl context = de_get_current_design_context();
decl layerid = de_get_layerid(context, "cond2", "drawing");
decl shape = db_add_arc4_ex(context,layerid,sx,sy,cx,cy,chordLength,0);
```

Return Value(s)

The new arc shape object if it was successfully created, NULL otherwise.

Version Introduced

ADS 2011

Version Compatible

ADS 2011

See also

db_add_arc_ex() (ael)
db_add_arc1_ex() (ael)
db_add_arc2_ex() (ael)
db_add_arc3_ex() (ael)

Where Used (ael)

Schematic, Layout

db_add_arc4()

This function creates a clockwise or counterclockwise standalone arc in the given design context, that is created with a user defined starting point coordinate, center point

coordinate, and a chord length for an arc angle expressed in terms of circumference.

Syntax

```
arcShape = db_add_arc4(context,layerid,x1,y1,x2,y2,chordLength,cw);
```

Where

- *context* is a design context to add the arc into.
- *layerid* is the layerid to add the arc into.
- *x1*,*y1* is the x,y coordinate starting point location of the arc in user units.
- *x2*,*y2* is the x,y coordinate center point location of the arc in user units.
- *chordLength* is the arc angle expressed in terms of circumference.
- *cw* is an integer representing the direction of the arc:
  1 = clockwise
  0 = counterclockwise

Example

```
decl sx=0.0, sy=0.0;
decl cx=0.0, cy=50.0;
decl chordLength= 100.0;
decl context = de_get_current_design_context();
decl layerid = de_get_layerid(context, "cond2", "drawing");
decl shape = db_add_arc4(context,layerid,sx,sy,cx,cy,chordLength,0);
```

Return Value(s)

The new arc shape object if it was successfully created, NULL otherwise.

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_add_arc() (ael)
db_add_arc1() (ael)
db_add_arc2() (ael)
db_add_arc3() (ael)

Where Used (ael)

Schematic, Layout

db_add_arc_ex()

This function adds a clockwise or counterclockwise circular arc to a polygon or polyline under construction in the given design context. Before adding the arc, you must define the starting point of a polygon or polyline shape.

Syntax

```
db_add_arc_ex(context, x,y, angle);
```

Where,

- *context* is a design context.
- *x*,*y* is the x,y coordinate center point location of the arc in user units; this becomes the vertex in the polygon or polyline.
- *angle* is a real value sweep angle of the arc in degrees, where: negative value = clockwise, positive value = counterclockwise.

Example

The following code adds polygon with arcs.

```
db_add_polygon_ex(context);
db_add_point_ex(context, 3.25, 4.75);
db_add_point_ex(context, 4.125, 5.625);
db_add_point_ex(context, 5.25, 4.75);
db_add_point_ex(context, 6.25, 4.75);
db_add_arc_ex(context, 7.125, 4.75, 180.0);
db_add_point_ex(context, 8.75, 4.75);
db_add_point_ex(context, 8.75, 3.875);
db_add_arc_ex(context, 8.75, 3.125, -180.0);
db_add_point_ex(context, 3.75, 3.125);
db_add_point_ex(context, 3.75, 3.125);
db_end_ex(context, layerId);
```

Return Value(s)

None

Version Introduced

ADS 2011

Version Compatible

ADS 2011

See also

db_add_arc1_ex() (ael)
db_add_arc2_ex() (ael)
db_add_arc3_ex() (ael)
db_add_arc4_ex() (ael)
db_add_point_ex() (ael)
db_add_polygon_ex() (ael)
db_add_polyline_ex() (ael)
db_end_ex() (ael)

Where Used (ael)

Schematic, Layout

db_add_arc()

This function adds a clockwise or counterclockwise circular arc to a polygon or polyline under construction in the given design context. Before adding the arc, you must define the starting point of a polygon or polyline shape.

Syntax

```
db_add_arc(context, x,y, angle);
```

Where,

- *context* is a design context.
- *x*,*y* is the x,y coordinate center point location of the arc in user units; this becomes the vertex in the polygon or polyline.
- *angle* is a real value sweep angle of the arc in degrees, where: negative value = clockwise, positive value = counterclockwise.

Example

The following code adds polygon with arcs.

```
db_add_polygon(context);
db_add_point(context, 3.25, 4.75);
db_add_point(context, 4.125, 5.625);
db_add_point(context, 5.25, 4.75);
db_add_point(context, 6.25, 4.75);
db_add_arc(context, 7.125, 4.75, 180.0);
db_add_point(context, 8.75, 4.75);
db_add_point(context, 8.75, 3.875);
db_add_arc(context, 8.75, 3.125, -180.0);
db_add_point(context, 3.75, 3.125);
db_add_point(context, 3.75, 3.125);
db_end(context, layerId);
```

Return Value(s)

None

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_add_arc1() (ael)
db_add_arc2() (ael)
db_add_arc3() (ael)
db_add_arc4() (ael)
db_add_point() (ael)
db_add_polygon() (ael)
db_add_polyline() (ael)
db_end() (ael)

Where Used (ael)

Schematic, Layout

db_add_circle_ex()

This function adds a circle in the given design context in user-defined units. Returns the shape that was just created if successful, otherwise returns NULL.

Syntax

```
dbShape = db_add_circle_ex(context, layerid, x1, y1, radius ,thickness );
```

Where,

- *context* is a design context to add the circle into.
- *layerid* is the layerid to add the circle into.
- *x1*,*y1* is the x,y coordinate center point location of the circle in user units.
- *radius* is the radius of the circle.
- *thickness* is optional, where:
  1 = thin, default
  2 = medium
  3 = thick

Example

```
decl dbShape = db_add_circle_ex(context, layerId, 10.0,10.0, 25.5);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011

See also

db_add_construction_line_ex() (ael)
db_add_rectangle_ex() (ael)

Where Used (ael)

Schematic, Layout

db_add_circle()

This function adds a circle in the given design context in user-defined units. Returns the shape that was just created if successful, otherwise returns NULL.

Syntax

```
decl dbShape = db_add_circle(context, layerid, x1, y1, radius [,thickness]);
```

Where,

- *context* is a design context to add the circle into.
- *layerid* is the layerid to add the circle into.
- *x1*,*y1* is the x,y coordinate center point location of the circle in user units.
- *radius* is the radius of the circle.
- *thickness* is optional, where:
  1 = thin, default
  2 = medium
  3 = thick

Example

```
decl dbShape = db_add_circle(context, layerId, 10.0,10.0, 25.5);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_add_construction_line() (ael)
db_add_rectangle() (ael)

***Where Used* (ael)**

Schematic, Layout

db_add_construction_line_ex()

This function adds a construction line to the given design context using user-defined units. The angle of a construction line is defined by any two distinct points. Construction lines have no bounding box and extend out to infinity.

Syntax

```
dbShape = db_add_construction_line_ex(context, layerId, x1, y1, x2, y2);
```

Where,

- *context* is a design context to add the construction line into.
- *layerId* is the layerid to add the construction line into.
- *x1*, *y1* is the first x,y coordinate point location in user units that defines the line.
- *x2*, *y2* is the second x,y coordinate point location in user units that defines the line.

Return Value(s)

On successful execution, this function returns the shape that was created, otherwise returns NULL.

Example

```
// The example creates two construction lines that intersect at 0,0.
  db_add_construction_line_ex(context, layerid, 0, 0, 100, 0);
  db_add_construction_line_ex(context, layerid, 0, 0, 0, 100);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011

See also

db_add_circle_ex() (ael)
db_add_rectangle_ex() (ael)

Where Used (ael)

Schematic, Layout

db_add_construction_line()

This function adds a construction line to the given design context using user-defined units. The angle of a construction line is defined by any two distinct points. Construction lines have no bounding box and extend out to infinity.

Syntax

```
decl dbShape = db_add_construction_line(context, layerId, x1, y1, x2, y2);
```

Where,

- *context* is a design context to add the construction line into.
- *layerId* is the layerid to add the construction line into.
- *x1*, *y1* is the first x,y coordinate point location in user units that defines the line.
- *x2*, *y2* is the second x,y coordinate point location in user units that defines the line.

Return Value(s)

On successful execution, this function returns the shape that was created, otherwise returns NULL.

Example

```
// The example creates two construction lines that intersect at 0,0.
db_add_construction_line(context, layerid, 0, 0, 100, 0);
db_add_construction_line(context, layerid, 0, 0, 0, 100);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_add_circle() (ael)
db_add_rectangle() (ael)

Where Used (ael)

Schematic, Layout

db_add_path_ex()

Starts a path command sequence. Adds a path to the given design context.

Note
Use *db_end_ex()* (ael) to complete the path shape for the add path command sequence.

Syntax

```
db_add_path_ex(context);
```

Where,

- *context* is a design context to add the path into.

Example

The following code adds a path:

```
db_add_path_ex(context);
db_add_point_ex(context, 10, 20);
db_add_point_ex(context, 30, 40);
decl dbShape = db_end_ex(context, layerId);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011

See also

db_add_point_ex() (ael)
db_end_ex() (ael)

Where Used (ael)

Schematic, Layout

db_add_path()

Starts a path command sequence. Adds a path to the given design context.

Note
Use *db_end()* (ael) to complete the path shape for the add path command sequence.

Syntax

```
db_add_path(context);
```

Where,

- *context* is a design context to add the path into.

Example

The following code adds a path.

```
db_add_path(context);
db_add_point(context, 10, 20);
db_add_point(context, 30, 40);
decl dbShape = db_end(context, layerId);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_add_point() (ael)
db_end() (ael)

Where Used (ael)

Schematic, Layout

db_add_point_ex()

Adds a point to a polygon, polyline, or path using user units. Returns: TRUE if point was successfully added, FALSE otherwise.

Syntax

```
bOk = db_add_point_ex(context, x, y);
```

Where,

- *context* is a design context to add the point into.
- *x*,*y* is the x,y coordinate point location to add in user units.

Example

The following code adds a polygon with arcs.

```
db_add_polygon_ex(context);
db_add_point_ex(context, 3.25, 4.75);
db_add_point_ex(context, 4.125, 5.625);
db_add_point_ex(context, 5.25, 4.75);
db_add_point_ex(context, 6.25, 4.75);
db_add_arc_ex(context, 7.125, 4.75, 180.0);
db_add_point_ex(context, 8.75, 4.75);
db_add_point_ex(context, 8.75, 3.875);
db_add_arc_ex(context, 8.75, 3.125, -180.0);
db_add_point_ex(context, 3.75, 3.125);
db_add_point_ex(context, 3.75, 3.125);
decl dbShape = db_end_ex(context, layerId);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011

See also

db_add_path_ex() (ael)
db_add_polygon_ex() (ael)
db_add_polyline_ex() (ael)
db_end_ex() (ael)

Where Used (ael)

Where Used (ael)

Schematic, Layout

db_add_point()

Adds a point to a polygon, polyline, or path using user units.
Returns TRUE if point was successfully added; FALSE otherwise.

Syntax

```
decl bOk = db_add_point(context, x, y);
```

Where,

- *context* is a design context to add the point into.
- *x*,*y* is the x,y coordinate point location to add in user units.

Example

The following code adds a polygon with arcs.

```
db_add_polygon(context);
db_add_point(context, 3.25, 4.75);
db_add_point(context, 4.125, 5.625);
db_add_point(context, 5.25, 4.75);
db_add_point(context, 6.25, 4.75);
db_add_arc(context, 7.125, 4.75, 180.0);
db_add_point(context, 8.75, 4.75);
db_add_point(context, 8.75, 3.875);
db_add_arc(context, 8.75, 3.125, -180.0);
db_add_point(context, 3.75, 3.125);
db_add_point(context, 3.75, 3.125);
decl dbShape = db_end(context, layerId);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_add_path() (ael)
db_add_polygon() (ael)
db_add_polyline() (ael)
db_end() (ael)

Where Used (ael)

Schematic, Layout

db_add_polygon_ex()

Starts a polygon command sequence. Adds a polygon to the given design context.
Returns: none.

Note
Use *db_end_ex()* (ael) to complete the polygon shape for the add polygon command sequence.

Syntax

```
db_add_polygon_ex(context);
```

Where,

- *context* is a design context to add the polygon into.

Example

The following code adds a polygon with arcs:

```
db_add_polygon_ex(context);
db_add_point_ex(context, 3.25, 4.75);
db_add_point_ex(context, 4.125, 5.625);
db_add_point_ex(context, 5.25, 4.75);
db_add_point_ex(context, 6.25, 4.75);
db_add_arc_ex(context, 7.125, 4.75, 180.0);
db_add_point_ex(context, 8.75, 4.75);
db_add_point_ex(context, 8.75, 3.875);
db_add_arc_ex(context, 8.75, 3.125, -180.0);
db_add_point_ex(context, 3.75, 3.125);
db_add_point_ex(context, 3.75, 3.125);
decl dbShape = db_end_ex(context, layerId);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011

See also

db_add_arc_ex() (ael)
db_add_point_ex() (ael)
db_add_polyline_ex() (ael)
db_end_ex() (ael)

Where Used (ael)

Schematic, Layout

db_add_polygon()

Starts a polygon command sequence. Adds a polygon to the given design context.
Returns: none.

Note
Use *db_end()* (ael) to complete the polygon shape for the add polygon command sequence.

Syntax

```
db_add_polygon(context);
```

Where,

- *context* is a design context to add the polygon into.

Example

The following code adds a polygon with arcs:

```
db_add_polygon(context);
db_add_point(context, 3.25, 4.75);
db_add_point(context, 4.125, 5.625);
db_add_point(context, 5.25, 4.75);
db_add_point(context, 6.25, 4.75);
db_add_arc(context, 7.125, 4.75, 180.0);
db_add_point(context, 8.75, 4.75);
db_add_point(context, 8.75, 3.875);
db_add_arc(context, 8.75, 3.125, -180.0);
db_add_point(context, 3.75, 3.125);
db_add_point(context, 3.75, 3.125);
decl dbShape = db_end(context, layerId);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_add_arc() (ael)
db_add_point() (ael)
db_add_polyline() (ael)
db_end() (ael)

Where Used (ael)

Schematic, Layout

db_add_polyline_ex()

This function starts a polyline command sequence and adds a polyline to the given design context. This function does not return any value.

Note
Use db_end_ex() to complete the polyline shape for the add polyline command sequence.

Syntax

```
db_add_polyline_ex(context);
```

Where,

- *context* is a design context to add the polyline into.

Example

```
db_add_polyline_ex(context);
db_add_point_ex(context,0,0);
```

```
db_add_point_ex(context,10,20);
decl dbShape = db_end_ex(context,layerId);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011

See also

db_add_arc_ex() (ael)
db_add_point_ex() (ael)
db_add_polygon_ex() (ael)
db_end_ex() (ael)

***Where Used* (ael)**

Schematic, Layout

db_add_polyline()

This function starts a polyline command sequence and adds a polyline to the given design context. This function does not return any value.

Note
Use *db_end()* (ael) to complete the polyline shape for the add polyline command sequence.

Syntax

```
db_add_polyline(context);
```

Where,

- *context* is a design context to add the polyline into.

Example

```
db_add_polyline(context);
db_add_point(context,0,0);
db_add_point(context,10,20);
decl dbShape = db_end(context,layerId);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_add_arc() (ael)
db_add_point() (ael)
db_add_polygon() (ael)
db_end() (ael)

Where Used (ael)

Schematic, Layout

db_add_rectangle_ex()

This function adds a rectangle to the given design context. Takes the lower left and upper right coordinate point locations in user units. Returns the shape that was just created if successful, otherwise returns NULL.

Syntax

```
dbShape = db_add_rectangle_ex(context, layerid, x1,y1, x2,y2 [, thickness]);
```

Where,

- *context* is a design context to add the rectangle into.
- *layerid* is the layerid to add the rectangle into.
- *x1*,*y1* is the x,y coordinate lower left point location of the rectangle in user units.
- *x2*,*y2* is the x,y coordinate upper right point location of the rectangle in user units.
- *thickness* is optional and can have following values:
  1 = thin, default
  2 = medium
  3 = thick

Example

```
decl dbShape = db_add_rectangle_ex(context, layerId, 10,20, 30,40);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011

See also

db_add_circle_ex() (ael)
db_add_construction_line_ex() (ael)

Where Used (ael)

Schematic, Layout

db_add_rectangle()

This function adds a rectangle to the given design context. Takes the lower left and upper right coordinate point locations in user units.
Returns the shape that was just created if successful, otherwise returns NULL.

Syntax

```
decl dbShape = db_add_rectangle(context, layerid, x1,y1, x2,y2 [, thickness]);
```

Where,

- *context* is a design context to add the rectangle into.
- *layerid* is the layerid to add the rectangle into.
- *x1*,*y1* is the x,y coordinate lower left point location of the rectangle in user units.
- *x2*,*y2* is the x,y coordinate upper right point location of the rectangle in user units.
- *thickness* is optional and can have following values:
  1 = thin, default
  2 = medium
  3 = thick

Example

```
decl dbShape = db_add_rectangle(context, layerId, 10,20, 30,40);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_add_circle() (ael)
db_add_construction_line() (ael)

Where Used (ael)

Schematic, Layout

db_end_ex()

This function completes a polygon, polyline, wire or trace command sequence. This command completes a shape. Use the de_end_command() to terminate the repeating command. Returns a the shape that was just created if successful, NULL otherwise

Syntax

```
decl dbShape = db_end_ex(context, layerid [, thickness]);
```

Where,

- *context* is a design context to end editing the shape into.
- *layerid* is the layerid to end editing the shape into.
- *thickness* is optional and can have following values:
  1 = thin, default
  2 = medium
  3 = thick

Example

The following code adds a polygon with arcs:

```
db_add_polygon_ex(context);
db_add_point_ex(context, 3.25, 4.75);
db_add_point_ex(context, 4.125, 5.625);
db_add_point_ex(context, 5.25, 4.75);
```

```
db_add_point_ex(context, 6.25, 4.75);
db_add_arc_ex(context, 7.125, 4.75, 180.0);
db_add_point_ex(context, 8.75, 4.75);
db_add_point_ex(context, 8.75, 3.875);
db_add_arc_ex(context, 8.75, 3.125, -180.0);
db_add_point_ex(context, 3.75, 3.125);
db_add_point_ex(context, 3.75, 3.125);
decl dbShape = db_end_ex(context, layerId);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011

See also

db_add_arc_ex() (ael)
db_add_path_ex() (ael)
db_add_point_ex() (ael)
db_add_polygon_ex() (ael)
db_add_polyline_ex() (ael)

Where Used (ael)

Schematic, Layout

db_end()

This function completes a polygon, polyline, wire or trace command sequence. This command completes a shape. Use the de_end_command() to terminate the repeating command. Returns a the shape that was just created if successful, NULL otherwise

Syntax

```
decl dbShape = db_end(context, layerid [, thickness]);
```

Where,

- *context* is a design context to end editing the shape into.
- *layerid* is the layerid to end editing the shape into.
- *thickness* is optional and can have following values:
  1 = thin, default
  2 = medium
  3 = thick

Example

The following code adds a polygon with arcs:

```
db_add_polygon(context);
db_add_point(context, 3.25, 4.75);
db_add_point(context, 4.125, 5.625);
db_add_point(context, 5.25, 4.75);
db_add_point(context, 6.25, 4.75);
db_add_arc(context, 7.125, 4.75, 180.0);
db_add_point(context, 8.75, 4.75);
db_add_point(context, 8.75, 3.875);
```

```
db_add_arc(context, 8.75, 3.125, -180.0);
db_add_point(context, 3.75, 3.125);
db_add_point(context, 3.75, 3.125);
decl dbShape = db_end(context, layerId);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_add_arc() (ael)
db_add_path() (ael)
db_add_point() (ael)
db_add_polygon() (ael)
db_add_polyline() (ael)

***Where Used* (ael)**

Schematic, Layout

db_highlight_instance_ex()

Highlights an instance using the highlight color.
Returns: none.

Syntax

```
db_highlight_instance_ex(context, instName, toggle);
```

Where,

- *context* is the design context the instance is in.
- *instName* is the unique instance name.

- *toggle* is optional. If present toggle highlight state, otherwise set highlighting.

Example

```
db_highlight_instance_ex(context, "M1");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_highlight() (ael)
db_is_highlighted() (ael)
db_unhighlight_instances_ex() (ael)

***Where Used* (ael)**

Schematic, Layout

db_unhighlight_instances_ex()

Function to unhighlight all instances in a given *design context* (ael).

Syntax

```
db_unhighlight_instances_ex(context);
```

Where,

- *context* is the design context the instances are within.

Example

```
db_unhighlight_instances_ex(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_highlight() (ael)
db_is_highlighted() (ael)
db_highlight_instance_ex() (ael)

Where Used (ael)

Schematic, Layout

db_update_parameters_ex()

This function is used to modify the parameter values.

Returns

None

Syntax

```
db_update_parameters_ex(context, valueList list(instName1, paramName1, value1,
...));
```

where

- *context* is the design context.

- *valueList* is a list of triplets (e.g, list(instName1, paramName1, value1, ...)) where:
  - instName1 = instance ID
  - paramName1 = parameter name
  - value1 = new value of parameter as string or real

Example

```
decl context= de_get_current_design_context();
  if (context != NULL)
     db_update_parameters_ex(context, list("TL3", "W", "50 mil", "Term1", "Z", "50 ohm"));
```

de_cell_exists()

Returns TRUE if a cell with the given cell name exists in the given open library. Returns FALSE if the given library is not open and if the library or cell is not found. Returns an AEL error if the given library is not open and not found and if the library does not have read permissions.

Syntax

```
decl bCellExists = de_cell_exists(libraryName, cellName);
```

Where,

- *libraryName* is a string library name of an open library to find if a cell exists within.
- *cellName* is a string cell name of a cell to find if it exists within a given open library

Example

```
decl bCellExists = de_cell_exists("my_sources", "my_cell");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_is_library_open() (ael)
de_cellview_exists() (ael)

Where Used (ael)

Schematic, Layout

de_cellview_exists()

Returns TRUE if a cellview with the given cell name and view name exists within a given open library.

Returns FALSE if the given cell name's view name does not exist in an open library with the given library name.
Returns an AEL error if the given library is not open and not found and if the library does not have read permissions.

Syntax

```
decl bCellViewExists = de_cellview_exists (libraryName, cellName, viewName);
```

Where,

- *libraryName* is a string library name of an open library to find if a cellview exists within.
- *cellName* is a string cell name of a cellview to find if it exists within a given open library.
- *viewName* is a string view name of a cellview to find if it exists within a given open library.

Example

```
decl bCellViewExists = de_cellview_exists("my_sources", "my_cell", "schematic");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

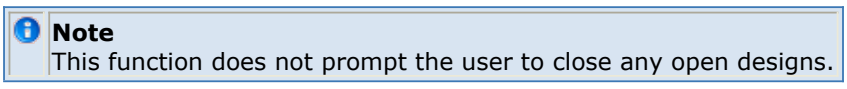
de_is_cellview_open() (ael)
de_is_library_open() (ael)
de_cell_exists() (ael)

Where Used (ael)

Schematic, Layout

de_close_workspace_without_prompting()

Closes the current open workspace. The close workspace routine saves the workspace state and preferences, closes all designs in windows, closes all open libraries, closes all running simulations, and changes the ADS current directory to the parent directory of the workspace directory.
Returns TRUE if the current workspace is closed successfully.
An AEL Error occurs and returns FALSE if the if there is no current workspace open or if the current workspace fails to be closed.



Syntax

```
decl bOk = de_close_workspace_without_prompting();
```

Example

```
decl bOk= de_close_workspace_without_prompting();
if (bOk == FALSE)
   return FALSE;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_open_workspace() (ael)

***Where Used* (ael)**

Schematic, Layout

de_copy_cell()

This function is used to copy a cell. The function returns TRUE if cell is successfully copied, otherwsie FALSE.

Syntax

```
decl stat = de_copy_cell(fromLibName, fromCellName, toLibName, toCellName);
```

Where,

- *fromLibName* is a string library name of the cell to copy from.
- *fromCellName* is a string cell name of the cell to copy from.
- *toLibName* is a string libray name to copy the cell into.
- *toCellName* is a new string cell name to copy the cell into..

Example

```
if ( !de_copy_cell( "myLibrary", "myCell", "myLibrary", "newMyCell") )
   return FALSE;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_delete_cell() (ael)
de_rename_cell() (ael)
de_copy_cellview() (ael)

***Where Used* (ael)**

Schematic, Layout

de_copy_cellview()

This function is used to copy a cellview. The function returns TRUE if cellview is successfully copied, otherwise FALSE.

Note
The library for the copy to cellview must have write permission or an AEL error will occur.

Syntax

```
decl stat = de_copy_cellview(fromLibName, fromCellName, fromViewName,
toLibName, toCellName, toViewName);
```

Where,

- *fromLibName* is the copy from cellview's library string name.
- *fromCellName* is the copy from cellview's cell string name.
- *fromViewName* is the copy from cellview's view string name.
- *toLibName* is the copy to cellview's library string name.
- *toCellName* is the copy to cellview's cell string name.
- *toViewName* is the copy to cellview's view string name.

Example

```
if ( !de_copy_cellview("myLibrary","myCell","schematic",
"myNewLibrary","myNewCell","schematic_alt") )
   return FALSE;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_delete_cellview() (ael)
de_is_cellview_open() (ael)
de_rename_cellview() (ael)
de_copy_cell() (ael)

Where Used (ael)

Schematic, Layout

de_create_polygon()

This function creates a Shape polygon object and adds the polygon to the given design context and refreshes the design context's view.

Syntax

```
decl dbShape = de_create_polygon(context, listOfPoints, [offset_x, offset_y [,
scaleFactor]]);
```

Where,

- *context* is a DesignContext returned from a function such as *de_get_current_design_context()* (ael).
- *listOfPoints* is a list of x,y coordinate pairs, such as: list(x1,y1,x2,y2,...,xN,yN); or is a list of lists of x,y coordinate pairs, such as: list(list(x1,y1),list(x2,y2),...,list(xN,yN)).
- *offset_x* is an optional real value to offset the polygon's x position by.
- *offset_y* is an optional real value to offset the polygon's y position by.
- *scaleFactor* is an optional real value scale factor to scale the dimensions of polygon by.

Example

```
decl context = de_get_current_design_context();

// test using a list of lists of x,y coordinate pairs.
decl inputList1 = list(list(-3,0),list(-13,0),list(-18,12), list(-3,12));
decl dbShape1 = de_create_polygon(context, inputList1);

// test using a list of x,y coordinate pairs.
decl inputList2 = list(0,0,10,0,15,15,15,0);
decl dbShape2 = de_create_polygon(context, inputList2);

// test the optional x,y offsets.
decl xOffset3 = 7.7;
decl yOffset3 = 8.8;
decl dbShape3 = de_create_polygon(context, inputList2, xOffset3, yOffset3);

// test the optional scale factor.
decl xOffset4 = 26.3;
decl yOffset4 = 29.5;
decl scaleFactor = 2.5;
decl dbShape4 = de_create_polygon(context, inputList2, xOffset4, yOffset4, scaleFactor);
de_view_all();
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions.

See also

***Where Used* (ael)**

Schematic, Layout

de_delete_cell()

This function is used to delete a cell and all of its contents. The function returns TRUE if cell was successfully deleted, otherwise FALSE.

Note
The library for the cell should be open. Designs from the cell must not be open. This function does not check for and resolve other dependent design references to the cell.

Syntax

```
decl stat = de_delete_cell(libName, cellName);
```

Where,

- *libName* is the string library string name of the cell to delete.
- *cellName* is the string cell name of the cell to delete.

Example

```
if ( !de_delete_cell( "myLibrary", "myCell") )
   return FALSE;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_copy_cell() (ael)
de_rename_cell() (ael)
de_delete_cellview() (ael)

Where Used (ael)

Schematic, Layout

de_delete_cellview()

This function is used to delete a cellview from ADS workspace. Function returns TRUE if cellview is successfully deleted, otherwise FALSE.

Note
The design to delete must not be open. This function does not check for and resolve any references to the cellview.

Syntax

```
decl stat = de_delete_cellview(libName, cellName, viewName);
```

Where,

- *libName* is the library string name of the cellView to delete.
- *cellName* is the cell string name of the cellView to delete.
- *viewName* is the view string name of the cellView to delete

Example

```
if ( !de_delete_cellview( "myLibrary", "myCell", "layout") )
   return FALSE;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_copy_cellview() (ael)
de_rename_cellview() (ael)
de_is_cellview_open() (ael)
de_delete_cell() (ael)

Where Used (ael)

Schematic, Layout

de_delete_workspace()

Deletes a specified ADS workspace from the given workspace directory path. This function can also be used to delete an ADS project from a given ADS project's directory path.

Syntax

```
de_delete_workspace(workspaceDirPath);
```

Where,

- *workspaceDirPath* is a string directory path to the workspace directory or project directory to delete.

Example

```
de_delete_workspace("C:/tmp/my_workspace_wk");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_close_workspace_without_prompting() (ael)
de_is_project_or_workspace_open() (ael)
de_open_workspace() (ael)

Where Used (ael)

Schematic, Layout

de_edit_item_ex()

Allows a given item to be selected for editing. Returns an item info handle for the instance.

Syntax

```
decl itemInfoP = db_edit_item_ex(context, instName);
```

Where,

- *context* is the design context the instance is in.
- *instName* is the unique name of an existing instance.

Example

```
decl itemInfoP = db_edit_item_ex(context, "M1");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

***Where Used* (ael)**

Schematic, Layout

de_format_lib_cell_view_name()

Returns a formatted string ADS design name constructed from a given libraryName, cellName, and viewName.

Note
An ADS Design name is in the format of "<libraryName>:<cellName>:<viewName>".

Syntax

```
decl designName = de_format_lib_cell_view_name(libraryName, cellName,
viewName);
```

Where,

- *libraryName* is the string library name to use for the generated ADS design name
- *cellName* is the string cell name to use for the generated ADS design name.
- *viewName* is the string view name to use for the generated ADS design name.

Example

```
decl libName, cellName, viewName;
decl designName = de_format_lib_cell_view_name("my_sources", "my_cell_1", "schematic");
// Where after the above call the designName value is:
//  designName == "my_sources:my_cell_1:schematic"
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_parse_lib_cell_view_name() (ael)
db_get_library_name() (ael)

db_get_cell_name() (ael)
db_get_view_name() (ael)
db_get_design_name() (ael)
db_get_component_name() (ael)

Where Used (ael)

Schematic, Layout

de_generate_symbol_ex()

This function generates a symbol into the given symbol context based on the design within the given *DesignContext* (ael). The generated symbol's number of pins is determined from the number of pins in the given design in the reference design context. Returns: none.

Syntax

```
de_generate_symbol_ex(symbolContext, designContext, leadLen, leadSpacing[,
symbolType, replace, order]);
```

Where,

- *symbolContext* is a symbol *DesignContext* (ael) to generate the symbol into.
- *designContext* is a *DesignContext* (ael) to generate the symbol from.
- *leadLen* is the lead length, usually 0.25.
- *leadSpacing* is the distance between pins, usually 0.25.

- *symbolType* is optional; TRUE if a Dual line symbol is desired; FALSE if a Quad line symbol is desired
- *replace* is optional; TRUE if the existing symbol should be replaced; FALSE for otherwise.
- *order* is optional; FALSE if the pins should be ordered by location; TRUE if the pins should be ordered by their pin numbers.

Example

```
de_generate_symbol_ex(symbolContext, designContext, 0.25, 0.25);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions.

See also

Where Used (ael)

Schematic, Layout

de_get_cells_in_library()

Returns an AEL list of string names of all cells in a given open library.
Returns an empty AEL list if no cells exist for the given open library’s name.
An AEL Error occurs if the library does not exist or if there is an error that occurs in

retrieving the cells for the given library.

Syntax

```
decl cellNameList = de_get_cells_in_library(libraryName);
```

Where,

- *libraryName* is a string library name for the library to get the cells names list for.

Example

```
decl myCellNameList = de_get_cells_in_library("my_sources");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_get_views_in_library_cell() (ael)

Where Used (ael)

Schematic, Layout

de_get_open_libraries()

Returns an AEL list of the current open library names.
If the optional argument bUserOnly is set TRUE, the open library name list returned will exclude any currently open system libraries.
Returns an empty AEL list if there currently are no open libraries.

Syntax

```
decl libraryNameList = de_get_open_libraries([bUserOnly]);
```

Where,

- *bUserOnly* is an optional TRUE or FALSE argument.
  If bUserOnly is specified as TRUE, the function will only return the list of open library names that are not system libraries.
  If this optional argument is not specified, the default value is FALSE; all open libraries' names will be returned.

Example

```
decl libraryList = de_get_open_libraries();
decl listsize = listlen(libraryList);
for ( i=0; I < listsize; i++)
{
   decl openLibName = nth(i, libraryList);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_is_library_open() (ael)

Where Used (ael)

Schematic, Layout

de_get_views_in_library_cell()

Returns an AEL list of string names of all views in a given library, cell. With the optional windowType argument you can limit the returned list of view names to a certain window type "LAYOUT_WINDOW, SCHEMATIC_WINDOW, or SYMBOL_WINDOW".
Returns an empty AEL list if no views exist for the given library name, cell name.
An AEL Error occurs if the library or the library's cell do not exist or if there is an error that occurs retrieving the cells for the given library's cell.

Syntax

```
decl viewNameList = de_get_views_in_library_cell (libraryName, cellName[,
windowtype]);
```

Where,

- *libraryName* is a string library name for the library's cell to get the view names list for.
- *cellname* is a string cell name for the library's cell to get the view names list for.
- *windowType* is an optional constant integer value which can be set to one of the following 4 constant values:
  - **SCHEMATIC_WINDOW** = 1 - The list of view names will be limited to views that are of type schematic.
  - **LAYOUT_WINDOW** = 2 - The list of view names will be limited to views that are of type layout.
  - **SYMBOL_WINDOW** = 3 - The list of view names will be limited to the views that are of type symbol.
  - **0** = 0 – Returns all views for all window types. This is the default value if this argument is not specified

Example

```
decl myCell1ViewNameList = de_get_views_in_library_cell("my_sources","my_cell_1");
decl myCell2SchematicViewNameList = de_get_views_in_library_cell("my_sources","my_cell_2",
SCHEMATIC_WINDOW);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_get_cells_in_library() (ael)

Where Used (ael)

Schematic, Layout

de_get_windows_with_same_library()

Returns an AEL list of window instances that are currently displaying designs stored in a given library.
Returns an empty AEL list if no windows are currently displaying designs from the given library.

Syntax

```
decl winInstanceList = de_get_windows_with_same_library(libraryName);
```

Where,

- *libraryName* is a string library name to return list of open windows for.

Example

```
decl winInstanceList = de_get_windows_with_same_library("my_sources");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_get_windows_with_same_context() (ael)

Where Used (ael)

Schematic, Layout

de_is_cellview_open()

Function to test if a cellview is open. Returns TRUE if the cellview is open, returns FALSE if it is not.

Syntax

```
decl isOpen = de_is_cellview_open(libName, cellName, viewName);
```

Where,

- *libName* is the library string name of the cellView to test is open.
- *cellName* is the cell string name of the cellView to test is open.

- *viewName* is the view string name of the cellView to test is open.

Example

```
// Ensure my view is open.
if (de_is_cellview_open("mylibrary", "myCell", "myview"))
   return TRUE;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_cellview_exists() (ael)
de_copy_cellview() (ael)
de_delete_cellview() (ael)
de_is_library_open() (ael)
de_rename_cellview() (ael)

***Where Used* (ael)**

Schematic, Layout

de_is_library_open()

Returns TRUE if any library with the given library name is open; Returns FALSE otherwise.

Syntax

```
decl bIsOpen = de_is_library_open(libraryName);
```

Where,

- *libraryName* is a string library name to check is opened.

Example

```
decl isOpen = de_is_library_open("ads_sources");
if (isOpen == FALSE)
   return FALSE;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_get_open_libraries() (ael)
de_is_cellview_open() (ael)

***Where Used* (ael)**

Schematic, Layout

de_is_project_or_workspace_open()

This function returns TRUE if any ADS Workspace (ADS2011 and later) or ADS Project(used in ADS2009 Update 1 and prior releases) is open, otherwise FALSE.

Returns

TRUE or FALSE

Syntax

```
decl bIsOpen = de_is_project_or_workspace_open();
```

Example

```
decl bIsOpen = de_is_project_or_workspace_open();
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_close_workspace_without_prompting() (ael)
de_open_workspace() (ael)

Where Used (ael)

Schematic, Layout

de_open_workspace()

Opens a specified workspace directory path. The open workspace routine will change the current ADS directory to be the workspace's directory, will open the libraries for the workspace, open up the workspace's state, and read in the workspaces preferences. Returns TRUE if the given workspace directory path exists and is opened successfully. An AEL Error occurs and returns FALSE if the given workspace directory path is not found or if the workspace fails to be opened.

Syntax

```
decl bOk = de_open_workspace(workspaceDirPath);
```

Where,

- *workspaceDirPath* is a string directory path to the workspace directory.

Example

```
decl bOk = de_open_workspace("C:/tmp/my_workspace_wk");
if (bOk == FALSE)
   return FALSE;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_close_workspace_without_prompting() (ael)
de_is_project_or_workspace_open() (ael)

Where Used (ael)

Schematic, Layout

de_parse_lib_cell_view_name()

Returns a parsed libraryName, cellName, and viewName from a given ADS design name.

Note
An ADS Design name is in the format of "<libraryName>:<cellName>:<viewName>".

Syntax

```
de_parse_lib_cell_view_name (designName, &libraryName, &cellName, &viewName);
```

Where,

- *designName* is a string ADS design name. An ADS Design name is in the format of "<libraryName>:<cellName>:<viewName>".
- *libraryName* is an AEL pointer return argument that will have its value set to the parsed string library name for the given ADS design name
- *cellName* is an AEL pointer return argument that will have its value set to the parsed string cell name for the given ADS design name.
- *viewName* is an AEL pointer return argument that will have its value set to the parsed string view name for the given ADS design name.

Example

```
decl libName, cellName, viewName;
de_parse_lib_cell_view_name("my_sources:my_cell_1:schematic", &ibName, &ellName, &iewName);
// Where after the above call the libName, cellName, and viewName values are:
//  libName == "my_sources" , cellName == "my_cell_1", & viewName == "schematic"
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_format_lib_cell_view_name() (ael)
db_get_library_name() (ael)
db_get_cell_name() (ael)
db_get_view_name() (ael)
db_get_design_name() (ael)
db_get_component_name() (ael)

Where Used (ael)

Schematic, Layout

de_rename_cell()

This function is used to rename a cell to a specified new name. This function searchs for any design dependents for this cell and update their instances to use the new cell name appropriately. The function returns TRUE if cell is successfully renamed, otherwise FALSE.

Note
An AEL Error will occur if a cell with the new name already exists, if the cell to rename has any cellviews currently open, or the cell to rename does not exist.

Syntax

```
decl stat = de_rename_cell(libName, fromCellName, toCellName);
```

Where,

- *libName* is a string library name containing the cell to rename.
- *fromCellName* is a string cell name of the cell to rename.
- *toCellName* is a new string cell name to rename the cell to.

Example

```
if ( !de_rename_cell("myLibrary", "myCell", "myNewCell") )
   return FALSE;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_copy_cell() (ael)
de_delete_cell() (ael)
de_rename_cellview() (ael)

Where Used (ael)

Schematic, Layout

de_rename_cellview()

This function is used to rename a cellview to a specified new name. This function will search for any design dependents for this cellview and update their instances to use the new cellview name appropriately. The function returns TRUE if cellview is successfully renamed, otherwise FALSE.

Note
An AEL Error will occur if a cellview with the new name already exists, if the cellview to rename is open, or the cellview to rename does not exist.

Syntax

```
decl stat = de_rename_cellview(libName, cellName, fromViewName, toViewName);
```

Where,

- *libName* is the name of the library containing the cellview to rename.
- *cellName* is the name of the cell containing the cellview to rename.
- *fromViewName* is the cellview's old view name.
- *toViewName* is the cellview's new view name.

Example

```
if ( !de_rename_cellview( "myLibrary","myCell","schematic","schematic_alt") )
   return FALSE;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_copy_cellview() (ael)
de_delete_cellview() (ael)
de_is_cellview_open() (ael)
de_rename_cell() (ael)

Where Used (ael)

Schematic, Layout

de_set_hierarchy_instance_path()

Function to set the instance path for the design within given *design context* (ael).
The instance path starts at the top design and is used to specify the variable hierarchy.

Syntax

```
de_set_hierarchy_instance_path(context, pathFromRoot );
```

Where,

- *context* is the design context to set the hierarchy root for.
- *pathFromRoot* is the the instance path from the top design.
  It should only be specified if you want the current design to take parameters from a

specific path.
If you set pathFromRoot blank with an empty string "" it will unset the hierarchy instance path for the design context.

Example

```
de_set_hierarchy_instance_path(context, pathFromRoot);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_set_hierarchy_root() (ael)

***Where Used* (ael)**

Schematic, Layout

de_set_hierarchy_root()

Function to set the hierarchy root for the current design within a given *design context* (ael). The root needs to be specified in any layout that has expressions containing variables from a higher level design.

Syntax

```
de_set_hierarchy_root(context, rootDsnName);
```

Where,

- *context* is the design context to set the hierarchy root for.
- *rootDsnName* is the root design name. The ADS design name is in the format: "libName:cellName:viewName".

Example

```
de_set_hierarchy_root(context, rootDsnName);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_set_hierarchy_instance_path() (ael)

***Where Used* (ael)**

Schematic, Layout

de_activate()

Activates an instance by setting the deactivate flag of an instance to false. Deactivated instances (which are commented out) are not simulated and are displayed with a box with an x-mark through them. Returns: none.

See also: *de_deactivate()* (ael).

Syntax:

```
de_activate([x,y]);
```

where

x,y is optional. Coordinates within the select region of an instance to activate. If not given, activates previously-selected instances.

Example:

```
de_activate (10,20);
```
or
```
de_activate();
```

Where Used: **(ael)**

Schematic, Layout

de_add_arc1()

Adds an arc using user-defined units.

- To create an arc using simulator units, use the *de_draw_arc1()* function.
- To create an arc embedded in a polygon or polyline, use the *de_add_arc()* function.

Returns a handle to the data group shape that was just created.

See also: *de_draw_arc1()* (ael), *de_add_arc()* (ael).

Syntax:

```
de_add_arc1(x1, y1, x2, y2, x3, y3);
```

where

x1, y1 is the start point of the arc.

x2, y2 is the circumference point of the arc.

x3, y3 is the end point of the arc.

Example:

```
// A 180 degree clockwise standalone arc
de_add_arc1 (0.0, 100.0, -50.0, 50.0, 0.0, 0.0);
// A 180 degree counterclockwise standalone arc
de_add_arc1 (200.0, 0.0, 250.0, 50.0, 200.0, 100.0);
// A 180 degree counterclockwise standalone arc and save data group
decl dgHangle = de_add_arc1 (200.0, 0.0, 250.0, 50.0, 200.0, 100.0);
```

***Where Used:* (ael)**

Schematic, Layout

de_add_arc2()

Adds an arc using user-defined units.

- To create an arc using simulator units, use the *de_draw_arc2()* function.
- To create an arc embedded in a polygon or polyline, use the *de_add_arc()* function.

Returns a handle to the data group shape that was just created.

See also: *de_draw_arc2()* (ael), *de_add_arc()* (ael).

Syntax:

```
de_add_arc2(x1, y1, x2, y2, x3, y3, direction[, thickness]);
```

where

x1, y1 is the start point of the arc.

x2, y2 is the center point of the arc.

x3, y3 is the end point of the arc.

direction is the direction of arc, where:

- 1 = clockwise
- 0 = counter clockwise

thickness is optional, where:

- 1 = thin, default

- 2 = medium
- 3 = thick

Example:

```
// A 180 degree clockwise standalone arc
de_add_arc2 (0.0, 0.0, 0.0, 50.0, 0.0, 100.0, 1);
// A 180 degree counter clockwise standalone arc
de_add_arc2 (200.0, 0.0, 200.0, 50.0, 200.0, 100.0, 0);
// A 180 degree counterclockwise standalone arc and save data group
decl dgHangle = de_add_arc2 (200.0, 0.0, 200.0, 50.0, 200.0, 100.0, 0);
```

Where Used: **(ael)**

Schematic, Layout

de_add_arc3()

Adds an arc using user-defined units.

- To create an arc using simulator units, use the *de_draw_arc3()* function.
- To create an arc embedded in a polygon or polyline, use the *de_add_arc()* function.

Returns a handle to the data group shape that was just created.

See also: *de_draw_arc3()* (ael), *de_add_arc()* (ael).

Syntax:

```
de_add_arc3(x1, y1, x2, y2, sweepAngle, direction);
```

where

x1, y1 is the start point of the arc.

x2, y2 is the center point of the arc.

sweepAngle is the arc angle in degrees.

direction is the direction of arc, where:

- 1 = clockwise
- 0 = counter clockwise

Example:

```
// A 180 degree clockwise stand alone arc
```

```
de_add_arc3 (0.0, 0.0, 0.0, 50.0, 180.0, 1);
// A 180 degree counter clockwise stand alone arc
de_add_arc3 (200.0, 0.0, 200.0, 50.0, 180.0, 0);
// A 180 degree counterclockwise standalone arc and save data group
decl dgHangle = de_add_arc3 (200.0, 0.0, 200.0, 50.0, 180.0, 0);
```

Where Used: **(ael)**

Schematic, Layout

de_add_arc4()

Adds an arc using user-defined units.

- To create an arc using simulator units, use the *de_draw_arc4()* function.
- To create an arc embedded in a polygon or polyline, use the *de_add_arc()* function.

Returns a handle to the data group shape that was just created.

See also: *de_draw_arc4()* (ael), *de_add_arc()* (ael).

Syntax:

```
de_add_arc4(x1, y1, x2, y2, chordLength, direction);
```

where

x1, y1 is the start point of the arc.

x2, y2 is the center point of the arc.

direction is the direction of arc, where:

- 1 = clockwise
- 0 = counter clockwise

chordLength is the arc angle expressed in terms of circumference.

Example:

```
// A 180 degree clockwise standalone arc
de_add_arc4 (0.0, 0.0, 0.0, 50.0, 157.08, 1);
// A 180 degree counter clockwise standalone arc
de_add_arc4 (200.0, 0.0, 200.0, 50.0, 157.08, 0);
// A 180 degree counterclockwise standalone arc and save data group
decl dgHangle = de_add_arc4 (200.0, 0.0, 200.0, 50.0, 157.08, 0);
```

***Where Used:* (ael)**

Schematic, Layout

de_add_arc()

Adds a clockwise or counterclockwise, circular arc to a polygon or polyline in user units in the current representation. Before using an arc, you must define the starting point of a polygon or polyline. Returns: none.

See also: *de_add_arc1()* (ael) , *de_add_arc2()* (ael), *de_add_arc3()* (ael), *de_add_arc4()* (ael).

Syntax:

```
de_add_arc(x,y, angle);
```

where

x,y is the center point of the arc; this becomes a vertex in the polygon or polyline.

angle is the sweep angle of the arc, where:

- negative = clockwise
- positive = counterclockwise

Example:

The example creates a polygon with a clockwise and counterclockwise arc.

```
de_add_polygon();
de_add_point(3.25,4.75);
de_add_point(4.125,5.625);
de_add_point(5.25,4.75);
de_add_point(6.25,4.75);
de_add_arc(7.125,4.75,180.0);
de_add_point(8.75,4.75);
de_add_point(8.75,3.875);
de_add_arc(8.75,3.125,-180.0);
de_add_point(3.75,3.125);
de_add_point(3.75,3.125);
de_end();
```

***Where Used:* (ael)**

Schematic, Layout

de_add_circle()

Draws a circle in the current representation in user-defined units. To use simulator units, use the *de_draw_circ()* function. The *de_draw_circ()* function is used in artwork creation functions. Returns a handle to the data group shape that was just created.

See also: *de_add_arc()* (ael), *de_add_arc1()* (ael), *de_add_arc2()* (ael), *de_add_arc3()* (ael), *de_add_arc4()* (ael).

Syntax:

```
de_add_circle(x,y, radius[, thickness]);
```

where

x,y is the coordinates for the circle's center point.

radius is the radius of the circle.

thickness is optional, where:

- 1 = thin, default
- 2 = medium
- 3 = thick

Example:

```
api_set_current_window(1);
de_add_circle(10.0,10.0, 25.5);
// Or to capture the data group handle
decl dgHandle = de_add_circle(10.0,10.0, 25.5);
```

***Where Used:* (ael)**

Schematic, Layout

de_add_construction_line()

Adds a construction line using user-defined units. The angle of a construction line is defined by any two distinct points. Construction lines have no bounding box and extend out to infinity. Returns: none.

Syntax:

```
de_add_construction_line(x1, y1, x2, y2);
```

where

x1, y1 is the first point that defines the line.

x2, y2 is the second point that defines the line.

Example:

The example creates two construction lines that intersect at 0,0.

```
de_add_construction_line(0, 0, 100, 0);
de_add_construction_line(0, 0, 0, 100);
```

***Where Used:* (ael)**

Schematic, Layout

de_add_path()

Starts a path command sequence. Adds a path to the current representation. Returns a handle to the data group shape that was just created.

See also: *de_add_trace()* (ael), *de_add_point()* (ael), *de_end()* (ael).

Syntax:

```
de_add_path();
```

Example:

```
de_add_path();
de_add_point(10, 20);
de_add_point(30, 40);
de_end();
// Or to capture the data group handle
de_add_path();
de_add_point(10, 20);
de_add_point(30, 40);
decl dgHandle = de_end();
```

***Where Used:* (ael)**

Schematic, Layout

de_add_point()

Adds a point to a polygon, polyline, or path using user-defined units. To draw a shape using simulator units, use the function *de_draw_point()*. The *de_draw_point()* function is typically used in artwork generation functions. Returns: none.

See also: *de_add_polyline()* (ael), *de_add_polygon()* (ael), *de_add_path()* (ael).

Syntax:

```
de_add_point(x,y);
```

where

x,y is the coordinates, in user-defined units, for the new vertex.

Example:

```
//draws a polygon in the Schematic window
api_set_current_window(1);
de_add_polygon();
de_add_point(0.0, 5.625);
de_add_point(1.25, 6.125);
de_add_point(1.375, 5.0);
de_add_point(-0.125, 5.125);
de_add_point(-0.125, 5.125);
de_end();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Generate SnP Component
Polylines to Polygons

Where Used: (ael)

Schematic, Layout

de_add_polygon()

Starts a polygon command sequence. Adds a polygon to the current representation. Returns: none.

See also: *de_add_rectangle()* (ael), *de_add_circle()* (ael), *de_add_arc()* (ael), *de_add_point()* (ael), *de_end()* (ael), *de_add_polyline()* (ael).

Syntax:

```
de_add_polygon();
```

Example:

```
de_add_polygon();
de_add_point(0,0);
de_add_point(20,0);
de_add_point(20,40);
de_add_point(0,0);
de_end();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Polylines to Polygons

Where Used: **(ael)**

Schematic, Layout

de_add_polyline()

Starts a polyline command sequence. Adds a polyline to the current representation. Returns: none.

See also: *de_add_polygon()* (ael), *de_add_rectangle()* (ael), *de_add_circle()* (ael), *de_add_arc()* (ael), *de_add_point()* (ael), *de_end()* (ael).

Syntax:

```
de_add_polyline();
```

Example:

```
de_add_polyline();
de_add_point(0,0);
de_add_point(10,20);
de_end();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Generate SnP Component

***Where Used:* (ael)**

Schematic, Layout

de_add_property()

Adds property to data group, port, or instance. Returns: none.

See also: *de_remove_properties()* (ael), *de_set_edit_property()* (ael).

Syntax:

```
de_add_property(propName, propValue, editSelected);
```

where

propName is the name of property to be added.

propValue is the value of property to be added.

editSelected is the mode for selecting components to add, where:

- TRUE = edit only selected components
- FALSE = add to component selected by de_set_edit_property

Example:

```
de_add_property ("Prop1",5,TRUE);
```

***Where Used:* (ael)**

Schematic, Layout

de_add_rectangle()

Adds a rectangle to the current representation. Takes the lower left and upper right coordinate points. Returns a handle to the data group shape that was just created.

See also: *de_add_polygon()* (ael), *de_add_circle()* (ael).

Syntax:

```
de_add_rectangle(x1,y1, x2,y2[, thickness]);
```

where

x1,y1 is the first corner describing window.

x2,y2 is the second (opposite) selection corner.

thickness is optional, where:

- 1 = thin, default
- 2 = medium
- 3 = thick

Example:

```
de_add_rectangle(10,20, 30,40);
// Or to capture the data group handle
decl dgHangle = de_add_rectangle(10,20,30,40);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Generate SnP Component

***Where Used:* (ael)**

Schematic, Layout

de_add_text()

Adds a text string to the current representation. The string can be set with the function *de_set_text_string()* is passed into this function; the text attributes are set with the functions *de_set_text_height(), de_set_text_font(), de_set_text_angle()*. Returns: none.

See also: *de_set_text_string()* (ael).

Syntax:

```
de_add_text(x,y[, text]);
```

where

x,y is the point for the lower-left corner of first character in the string.

text is a text string.

Example:

```
de_set_text_string("This is a text string");
de_add_text(10,12);
de_add_text (13, 14);
```

or

```
de_add_text(10, 12, "This is a text string");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Generate SnP Component

***Where Used:* (ael)**

Schematic, Layout

de_add_trace()

Starts trace command sequence for adding a trace to the current representation. This command is followed by two or more *de_add_point()* commands to describe the trace vertices and the *de_end()* command to terminate the trace. Traces must start and end at a pin or another trace. Returns: none.

See also: *de_add_point()* (ael), *de_end()* (ael), *de_add_path()* (ael).

Syntax:

```
de_add_trace();
```

Example:

```
de_add_trace();
de_add_wire(10,10);
de_add_wire(30, 10);
de_end();
```

***Where Used:* (ael)**

Schematic, Layout

de_add_vertex()

Adds a vertex to an existing polygon, polyline, wire or trace in the current representation. Returns: none.

Syntax:

```
de_add_vertex(x,y, nx, ny);
```

where

x,y is the point between two existing vertices, between which a new vertex is added.

nx, ny is the coordinates for a new vertex.

Example:

```
de_add_vertex(5.0, 9.0, 5.0, 7.9);
```

***Where Used:* (ael)**

Schematic, Layout

de_add_wire()

Adds a wire vertex to a wire or trace connection in the current representation. Wires and traces must begin and end at another wire, trace or pin. Returns: none.

See also: *de_connect()* (ael).

Syntax:

```
de_add_wire(x,y);
```

where

x,y is the coordinates of the wire vertex in user units.

Example:

```
de_connect();
de_add_wire(-0.25, 3.125);
de_add_wire(1.25, 3.25);
de_end();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
PI Attenuator

***Where Used:* (ael)**

Schematic, Layout

de_add_wire_label()

Adds a label to a wire or pin at the x,y location. Any pins or wires with the same name are connected for simulations. If the label is in proper bus syntax, the wire or pin will be iterated appropriately.

Syntax:

```
de_add_wire_label(x, y, label);
```

where

x is the x coordinate of the label location

y is the y coordinate of the label location

label is label to give a wire

Example:

```
de_add_wire_label(1.0, 1.0, "A<0>");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Knowledge center website. You will need to register at this site with a valid support contract to download an example file.
Replace all Grounds with Node Name

***Where Used:* (ael)**

Schematic, Layout

de_bom()

Generates a bill of material file for the current representation. Returns: none.

See also: *de_parts()* (ael), *de_net()* (ael).

Syntax:

```
de_bom(fileName, showDlg);
```

where

fileName is the name of file to store Bill of Materials list.

showDlg is whether or not to show the Bill of Materials list, where:

- TRUE is show the list.
- FALSE is do not show the list (it still records to the file).

Example:

```
de_bom ("my BOMList", FALSE);
```

***Where Used:* (ael)**

Schematic, Layout

de_boolean_logical()

Performs boolean operations on shapes placed on different layers in a layout presentation. Returns: none.

Syntax:

```
de_boolean_logical(operator, inLayer1, inLayer2, outLayer, rmLayer1, rmLayer2,
applyToSelect[, winInstH]);
```

where

operator is a string that specifies the boolean operation to be performed. The choices are "AND", "OR", "XOR", and "DIFF".

inLayer1 and *inLayer2* are integers that specify the layers containing the shapes to perform the boolean operation on.

outLayer is an integer that specifies the layer to contain the results of the boolean operation.

rmLayer1 and *rmLayer2* are boolean flags that specify whether the input shapes are to be deleted or not. Choices are:

- TRUE = delete the shapes.
- FALSE = do not delete the shapes.

applyToSelect is a boolean flag that specifies whether the boolean operation is only to be performed on the input layers that have been selected. Choices are:

- TRUE = use only those shapes in the operation that have been selected in the input layers.
- FALSE = use all shapes in the input layers.

winInstH is an optional parameter. It is the handle to the window containing the layout representation to use for the boolean operation. If unspecified, the current window is used.

Example:

```
de_set_layer(1);
de_add_rectangle(-200, -50, 0, 50);
de_set_layer(2);
de_add_circle(0, 0, 50);
de_boolean_logical("AND", 1, 2, 3, TRUE, TRUE, FALSE);
```

Where Used: **(ael)**

Layout

de_break_connection()

Breaks connection to attached wire or trace from the selected instances in the current representation. Returns: none.

See also: *de_move_break()* (ael), *de_select_range()* (ael).

Syntax:

```
de_break_connection();
```

Example:

```
de_break_connection();
```

Where Used: **(ael)**

Schematic, Layout

de_change_annotation_layer()

Changes the layer of the nearest annotation (within the select region) to the current layer. Returns: none.

See also: *de_edit_annotation_attribute()* (ael).

Syntax:

```
de_change_annotation_layer(x,y);
```

where

x,y is the coordinates within the select region of an instance's annotation.

Example:

```
de_set_layer(5);
de_change_annotation_layer(10.4,3.4);
```

Where Used: **(ael)**

Schematic

de_check_rep_options()

Generates a check representation report. Checks for unconnected pins and wires, bus connectivity errors (schematic only), nodal mismatches (schematic vs. layout), wires in

layout (layout only), pin vs. port mismatches (schematic only), parameter values mismatches (schematic vs. layout), overlapping wires (layout only), and overlaid items. Returns: none.

Syntax:

```
de_check_rep_options(dispMode);
```

where

dispMode is the sum of desired checks, as one of the following modes:

- DEINFO_DISP_CHECK_NONE
- DEINFO_DISP_CHECK_UNCONNECTEDPIN
- DEINFO_DISP_CHECK_BUSCONN (Schematic only)
- DEINFO_DISP_CHECK_NODALMISMATCH
- DEINFO_DISP_CHECK_WIRELAYOUT (Layout only)
- DEINFO_DISP_CHECK_PINVSPORT (Schematic only)
- DEINFO_DISP_CHECK_OVERLAIDITEMS
- DEINFO_DISP_CHECK_OVERLAPWIRE (Layout only)
- DEINFO_DISP_CHECK_PARAMS

Example:

```
decl dispMODE;
dispMODE=DEINFO_DISP_CHECK_UNCONNECTEDPIN
       +DEINFO_DISP_CHECK_NODALMISMATCH;
de_check_rep_options(dispMODE);
```

***Where Used:* (ael)**

Schematic, Layout

de_chop()

Removes the defined rectangular region from selected polygons, rectangles, circles, or paths. This command will fail and display an error if traces or instances are selected. All other types are silently ignored. Returns: 1 if successful, otherwise 0.

Syntax:

```
de_chop(x1, y1, x2, y2);
```

where

x1, y1 is the first corner of the rectangular region.

x2, y2 is the second, opposite corner of the rectangular region.

Example:

```
de_select_all();
de_chop(0, 0, 10, 10);
```

Where Used:

Layout

de_clear_dc_annotation()

Removes the annotated DC node voltages and branch currents on the current schematic.

See also: *de_dc_annotation()* (ael).

Syntax:

```
de_clear_dc_annotation();
```

Example:

```
de_clear_dc_annotation();
```

Where Used: **(ael)**

Schematic

de_clear_highlighting()

Clears any highlighting from a design representation in a current window. Returns: none.

Syntax:

```
de_clear_highlighting();
```

Example:

```
de_clear_highlighting();
```

Where Used: **(ael)**

Schematic, Layout

de_close_all()

Closes all designs from memory. Closes all windows. Does not prompt for unsaved changes. Returns: none.

Syntax:

```
de_close_all();
```

Example:

```
de_close_all();
```

Where Used: **(ael)**

Schematic, Layout

de_close_window()

Closes the current window instance. Returns: none.

See also: *de_create_window()* (ael), *api_set_current_window_by_seq_num()* (ael).

Syntax:

```
de_close_window();
```

Example:

```
// Closes the current window instance
api_set_current_window_by_seq_num (1);
de_close_window();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Layer/Process Manager

Where Used: **(ael)**

Schematic, Layout

de_connect()

Starts a wire or trace connection using autorouting. This command is followed by two or more calls to *de_add_wire()* and is terminated with the *de_end()* function. Returns: none.

Syntax:

```
de_connect();
```

Example:

```
de_connect();
de_add_wire(-0.25,3.125);
de_add_wire(1.25,3.25);
de_end();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
PI Attenuator

Where Used: **(ael)**

Schematic, Layout

de_convert_path_to_trace()

Converts selected paths to traces. Returns: none.
See also: *de_convert_trace_to_path()* (ael), *de_edit_path_trace()* (ael).

Syntax:

```
de_convert_path_to_trace();
```

Example:

```
de_convert_path_to_trace();
```

***Where Used:* (ael)**

Layout

de_convert_to_polygon()

Converts selected closed shapes to polygons. All closed shapes except text are converted to polygons. Returns: none.

Syntax:

```
de_convert_to_polygon([x,y]);
```

where

x,y is optional. Point within select region of object to convert.

Example:

```
de_convert_to_polygon();
```

or

```
de_convert_to_polygon(10,20);
```

***Where Used:* (ael)**

Schematic, Layout

de_convert_traces_to_instances()

Converts selected traces to transmission line elements. The elements used are set using *de_set_trace()* commands. Returns: none.

Syntax:

```
de_convert_traces_to_instances();
```

Example:

```
de_convert_traces_to_instances();
```

Where Used: **(ael)**

Layout

de_convert_trace_to_path()

Converts selected traces to paths. Returns: none.

See also: *de_convert_path_to_trace()* (ael), *de_edit_path_trace()* (ael).

Syntax:

```
de_convert_trace_to_path();
```

Example:

```
de_convert_trace_to_path();
```

Where Used: **(ael)**

Layout

de_copy()

Copies the selected components in the current representation and places them at delta from the original position. Returns: none.

See also: *de_copy_to_buffer()* (ael).

Syntax:

```
de_copy(dx,dy);
```

where

dx,dy is the delta from original position.

Example:

```
de_copy(15.9,17);
```

***Where Used:* (ael)**

Schematic, Layout

de_copy_file()

Copies a file. Returns TRUE if successful, FALSE if error.

Syntax:

```
de_copy_file(fromFileName, toFileName);
```

where

fromFileName is the name of the file to be copied.

toFileNmee is the copy of fromFileName.

Example:

```
de_copy_file("fromFile", "toFile");
```

***Where Used:* (ael)**

Schematic, Layout

de_copy_to_buffer()

Copies the selected group in the current representation to the copy buffer. Returns: none.

Syntax:

```
de_copy_to_buffer([x,y]);
```

where

x,y is optional. Coordinates used as the paste origin reference point when the Paste From Buffer command is executed. If no (x,y) value is specified, the paste origin defaults to the first unconnected pin or lower left corner of bounding box.

Example:

```
de_select_all();
de_copy_to_buffer();
de_paste_from_buffer(20.0,30.9);
```

Where Used: **(ael)**

Schematic, Layout

de_copy_to_layer()

Copies the selected group in the current representation to the current active layer.
Returns: none.

See also: *de_move_to_layer()* (ael).

Syntax:

```
de_copy_to_layer([layerNo]);
```

where

layerNo is optional; layer to copy selected objects to. If layer not specified, object is copied to current layer.

Example:

```
de_select_all();
de_set_layer(4);
de_copy_to_layer();
```

Where Used: **(ael)**

Schematic, Layout

de_create_window()

Creates and opens a window. Returns the handle to the opened window.

See also: *de_close_window()* (ael).

Syntax

```
de_create_window(window_code[, winInstP, xLoc, yLoc, width, height]);
```

where

window_code is the code for type of window; one of the following:

- SCHEM_WIN = Schematic window
- LAYOUT_WIN = Layout window

winInstP is optional. NULL if no parent window; otherwise, winInst of parent window with design to show in newly-created window. The cfg env vars are:

- SCHEMATIC/LAYOUT_WINDOW_X_LOC
- SCHEMATIC/LAYOUT_WINDOW_Y_LOC
- SCHEMATIC/LAYOUT_WINDOW_HEIGHT
- SCHEMATIC/LAYOUT_WINDOW_WIDTH

If you don't specify the xLoc, yLoc, width, and height, then:

- If there are no windows open, then open the window with the coordinates, width, and height specified by the cfg env vars.
- If there are windows open, then instead of using the coordinates specified by the cfg env vars, open the window to the lower right of the most-recently opened window, but still use the width and height specified by the cfg env vars.
- If you are reattaching to a workspace which had windows open before you reattached to it (that is, save workspace state), then restore the windows, to the coordinates, width and height, that they had before.

xLoc, yLoc is optional. Initial position for the window.

width is optional. Width of window in user units.

height is optional. Height of window in user units.

Example:

```
// sets the layout window active and creates a rectangle in it.
de_create_window(LAYOUT_WIN);
rectangle (10.3,0,14.5,11);
```

***Where Used:* (ael)**

Schematic, Layout

de_crop()

Preserves the defined rectangular region from selected polygons, rectangles, circles, or paths while removing all areas outside of the region. This command will fail and will display an error if traces or instances are selected. All other types are silently ignored. Returns: 1 if successful, otherwise 0.

Syntax:

```
de_crop(x1, y1, x2, y2);
```

where

x1, *y1* is the first corner of the rectangular region.

x2, *y2* is the second, opposite corner of the rectangular region.

Example:

```
de_select_all();
de_crop(0, 0, 10, 10);
```

Where Used:

Layout

de_dc_annotation()

Display DC node voltages and branch currents on the current schematic after a DC or Transient simulation.

See also: *de_clear_dc_annotation()* (ael).

Syntax:

```
de_dc_annotation();
```

Example:

```
de_dc_annotation();
```

Where Used: **(ael)**

Schematic

de_deactivate()

Sets instance's deactivate flag to true (makes the instance deactivated). Deactivated instances are not simulated (they are commented out) and are displayed with a box and x-mark over them. Returns: none.

See also: *de_activate()* (ael).

Syntax:

```
de_deactivate([x,y]);
```

where

x,y is optional. Coordinates within the select region of an instance to deactivate. If not given, deactivates selected instances.

Example:

```
decl itemInfoOSP = de_init_item("R");
de_place_item(itemInfoOSP, -1, 1);
de_deactivate(-1,1);
```

***Where Used:* (ael)**

Schematic, Layout

de_define_edge_area_port()

Modifies a circle, path, polygon, rectangle to become a area port or modifies a polyline or arc to become a edge port. Edge and area ports must be associated with a Port. To create a Port, see *de_draw_port()* (ael) or *de_define_port()* (ael). Returns: none.

Syntax:

```
de_define_edge_area_port(dgHandle, portNumber);
```

where

dgHandle is the handle to a data group, as returned from the creation of a circle, path, polygon, rectangle, polyline, or arc. See *de_add_arc1()* (ael), *de_add_arc2()* (ael), *de_add_arc3()* (ael), *de_add_arc4()* (ael), *de_add_path()* (ael), *de_add_circle()* (ael), *de_add_rectangle()* (ael), *de_draw_arc1()* (ael), *de_draw_arc2()* (ael), *de_draw_arc3()* (ael), *de_draw_arc4()* (ael), *de_draw_circ()* (ael), *de_draw_rect()* (ael), and *de_end()* (ael).

portNumber is the port number of the corresponding Port that this edge or area port is associated with.

Example:

```
// Place a Port 1 and an associated area pin on layer 2
de_set_layer(2);
de_draw_port(0,0,-90,NULL,1);
decl dgHandle = de_draw_rect(0,0, 10,20);
de_define_edge_area_port(dgHandle,1);
```

***Where Used:* (ael)**

Layout

de_define_npport()

Places a non-preferred port/pin in an artwork instance. This is used in layouts with multiple connection points per pin to define connection points. The preferred connection point is used by design synchronization to connect to by default. Returns: none.

Syntax:

```
de_define_npport(x,y, angle, portNo, pinName);
```

where

x,y is the location of the port (in user-defined units).

angle is the angle of the port (used to determine abutting interconnected instance).

portNo is the port/pin number.

pinName is optional. The pin/part name; a string.

Example:

```
de_define_npport(10.2, 14.5, 90.0, 2, "input_sig");
```

***Where Used:* (ael)**

Layout

de_define_port()

Places a single port, with a unique port number, in an artwork instance using user units. To specify location and angle in simulator units, use *de_draw_port()*. Returns: none.

de_define_port() is an AEL function used mostly in AEL artwork macro AEL. It adds a pin to an artwork at a given location and orientation. This function is not related to the regular Port that is placeable from the toolbar, and is operational only when the program is in INST_VIEW mode which can only be set within the program. INST_VIEW mode is the mode you are in when you place a component in a design.

Syntax:

```
de_define_port(x,y[, angle, portNo, portName, portPower]);
```

where

x,y is the port location (in user-defined units).

angle is optional; the angle of the port (used to determine abutting interconnected instance); a real number. Default = 0.0.

portNo is optional; the unique port number; an integer. Default = -1.

portName is optional; name of the pin; a string. Default = NULL.

portPower is optional; power of the pin; a string. Default = NULL.

Example:

```
de_define_port(10, 20, 0, 4,"input_sig");
```

Where Used: **(ael)**

Layout

de_delete()

Deletes selected objects, or object near given point in current representation. Returns: none.

Syntax:

```
de_delete([x,y]);
```

where

x,y is optional. Point within select region of object to delete.

Example:

```
de_delete();
```

or

```
de_delete(10.0, 34.68);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Export all Variables
Replace all GROUND's with Node Name

Where Used: **(ael)**

Schematic, Layout

de_delete_all_orphaned_instances()

Deletes all the orphaned instances in the current representation. Returns: none.

Syntax:

```
de_delete_all_orphaned_instances();
```

Example:

```
de_delete_all_orphaned_instances();
```

Where Used: **(ael)**

Schematic, Layout

de_delete_view()

Deletes a stored view with the matching name of view. Returns: none.

See also: *de_store_current_view()* (ael), *de_restore_view()* (ael).

Syntax:

```
de_delete_view(viewName);
```

where

viewName is the name of view to delete.

Example:

```
de_delete_view("myView");
```

Where Used: **(ael)**

Schematic, Layout

de_deselect_all()

Deselects all objects in the current representation. Returns: none.

Syntax:

```
de_deselect_all();
```

Example:

```
de_deselect_all();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Flatten Design
Polylines to Polygons
Set Variable Value 2

Where Used: **(ael)**

Schematic, Layout

de_deselect_all_force()

Deselects all objects regardless of layer protection status. Returns: none.

Syntax:

```
de_deselect_all_force();
```

Example:

```
de_deselect_all_force();
```

Where Used: **(ael)**

Schematic, Layout

de_deselect_by_name()

Deselects instances by instance name or ID. Returns: none.

Syntax:

```
de_deselect_by_name(instName, selectType);
```

where

instName is the instance name or ID.

selectType is the selection type code, where:

- 0 = selects by instance name
- 1 = selects by instance ID

Example:

```
de_deselect_by_name("MLIN",0);
```

or

```
de_deselect_by_name("TL1", 1);
```

Where Used: **(ael)**

Schematic, Layout

de_deselect_window()

Deselects all objects in given window in current representation. Returns: none.

Syntax:

```
de_deselect_window(x1,y1, x2,y2);
```

where

x1,y1 is the first corner describing window.

x2,y2 is the second (opposite) selection corner.

Example:

```
de_deselect_window(10,15, 22,87);
```

Where Used: **(ael)**

Schematic, Layout

de_difference()

Creates new polygons by subtracting the intersections of the selected shapes (e.g., polygon, rectangle, circle, or path) from the union of the selected shapes on the same layer. Returns: true or false, which indicates if the command was successful.

See also: *de_union()* (ael), *de_intersection()* (ael).

Syntax:

```
de_difference();
```

Example:

```
de_difference();
```

Where Used: **(ael)**

Layout

de_draw_arc1()

Adds an arc using simulator units. Typically, used in artwork creation macros.

- To create an arc using user-defined units, use the *de_add_arc1()* function.
- To create an arc embedded in a polygon or polyline, use the *de_draw_arc()* function.

Returns a handle to the data group shape that was just created.

See also: *de_add_arc1()* (ael), *de_add_arc()* (ael).

Syntax:

```
de_draw_arc1(x1, y1, x2, y2, x3, y3);
```

where

x1, y1 is the start point of the arc.

x2, y2 is the circumference point of the arc.

x3, y3 is the end point of the arc.

Example:

```
// A 180 degree clockwise standalone arc
de_draw_arc1(0, 100, -50, 50, 0, 0);
// A 180 degree counterclockwise standalone arc
de_draw_arc1(200, 0, 250, 50, 200, 100);
// A 180 degree counterclockwise standalone arc and save data group
decl dgHangle = de_draw_arc1(200, 0, 250, 50, 200, 100);
```

***Where Used:* (ael)**

Schematic, Layout

de_draw_arc2()

Adds an arc using simulator units. Typically, used in artwork creation macros.

- To create an arc using user-defined units, use the *de_add_arc2()* function.
- To create an arc embedded in a polygon or polyline, use the *de_draw_arc()* function.

Returns a handle to the data group shape that was just created.

See also: *de_add_arc2()* (ael), *de_draw_arc()* (ael).

Syntax:

```
de_draw_arc2(x1, y1, x2, y2, x3, y3, direction);
```

where

x1, y1 is the start point of the arc.

x2, y2 is the center point of the arc.

x3, y3 is the end point of the arc.

direction is the arc direction code, where:

- 1 = clockwise
- 0 = counter clockwise

Example:

```
// A 180 degree clockwise standalone arc
de_draw_arc2 (0.0, 0.0, 0.0, 50.0, 0.0, 100.0, 1);
// A 180 degree counter clockwise standalone arc
de_draw_arc2 (200.0, 0.0, 200.0, 50.0, 200.0, 100.0, 0);
// A 180 degree counterclockwise standalone arc and save data group
decl dgHangle = de_draw_arc2 (200.0, 0.0, 200.0, 50.0, 200.0, 100.0, 0);
```

Where Used: **(ael)**

Schematic, Layout

de_draw_arc3()

Adds an arc using simulator units. Typically, used in artwork creation macros.

- To create an arc using user-defined units, use the *de_add_arc3()* function.
- To create an arc embedded in a polygon or polyline, use the *de_draw_arc()* function.

Returns a handle to the data group shape that was just created.

See also: *de_add_arc3()* (ael), *de_draw_arc()* (ael).

Syntax:

```
de_draw_arc3(x1, y1, x2, y2, sweepAngle, direction);
```

where

x1, y1 is the start point of the arc.

x2, y2 is the center point of the arc.

sweepAngle is the arc angle in degrees.

direction is the arc direction code, where:

- 1 = clockwise
- 0 = counter clockwise

Example:

```
// A 180 degree clockwise stand alone arc
de_draw_arc3 (0.0, 0.0, 0.0, 50.0, 180.0, 1);
// A 180 degree counter clockwise stand alone arc
de_draw_arc3 (200.0, 0.0, 200.0, 50.0, 180.0, 0);
// A 180 degree counterclockwise standalone arc and save data group
decl dgHangle = de_draw_arc3 (200.0, 0.0, 200.0, 50.0, 180.0, 0);
```

***Where Used:* (ael)**

Schematic, Layout

de_draw_arc4()

Adds an arc using simulator units. Typically, used in artwork creation macros.

- To create an arc using user-defined units, use the *de_add_arc4()* function.
- To create an arc embedded in a polygon or polyline, use the *de_draw_arc()* function.

Returns a handle to the data group shape that was just created.

See also: *de_add_arc4()* (ael), *de_draw_arc()* (ael).

Syntax:

```
de_draw_arc4(x1, y1, x2, y2, chordLength, direction);
```

where

x1, y1 is the start point of the arc.

x2, y2 is the center point of the arc.

chordLength is the arc angle expressed in terms of circumference.

direction is the arc direction code, where:

- 1 = clockwise
- 0 = counterclockwise

Example:

```
// A 180 degree clockwise standalone arc
de_draw_arc4 (0.0, 0.0, 0.0, 50.0, 157.08, 1);
// A 180 degree counter clockwise standalone arc
de_draw_arc4 (200.0, 0.0, 200.0, 50.0, 157.08, 0);
// A 180 degree counterclockwise standalone arc and save data group
decl dgHangle = de_draw_arc4 (200.0, 0.0, 200.0, 50.0, 157.08, 0);
```

Where Used: **(ael)**

Schematic, Layout

de_draw_arc()

Adds a circular arc to a polygon or polyline in simulator units. Typically, this is used in artwork creation macros rather than in the *de_add_arc()* function. Returns: none.

See also: *de_draw_arc1()* (ael), *de_draw_arc2()* (ael), *de_draw_arc3()* (ael), *de_draw_arc4()* (ael).

Syntax:

```
de_draw_arc(x,y, angle);
```

where

x,y is the center of the arc.

angle is the sweep angle of the arc.

- Negative = clockwise
- Positive = counterclockwise

Example:

```
de_add_polyline();
de_draw_point(0,0);
de_draw_arc(20,0,90.0);
de_draw_point(40,60);
de_end();
```

Where Used: **(ael)**

Schematic, Layout

de_draw_circ()

Draws a circle in simulator units. Typically, used in artwork creation macros rather than in the *de_add_circle()* function. Returns a handle to the data group shape that was just created.

Syntax:

```
de_draw_circ(x,y,radius);
```

where

x,y is the center of the circle.

radius is the radius of the circle.

Example:

```
de_draw_circ(10,45, 36.7);
// Or to capture the data group handle
decl dgHandle = de_draw_circ(10,45, 36.7);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Polygons to Circles
Polylines to Circles

***Where Used:* (ael)**

Schematic, Layout

de_draw_point()

Adds a point to a polygon or polyline to the current representation using simulator units. The polygon or polyline is started with the *de_add_polygon()* or *de_add_polyline()* command and it must be terminated with an *de_end()*. Typically, used in artwork creation macros rather than in the *de_add_point()* function. Returns: none.

Syntax:

```
de_draw_point(x,y);
```

where

x,y is the coordinates of the point to add.

Example:

```
de_add_polyline();
de_draw_point(0,0);
de_draw_point(25,56);
de_end();
```

***Where Used:* (ael)**

Schematic, Layout

de_draw_port()

Adds a port to an art work instance. Returns: none.

de_draw_port() is an AEL function used mostly in AEL artwork macro AEL. It adds a pin to an artwork at a given location and orientation. This function is not related to the regular Port that is placeable from the toolbar, and is operational only when the program is in INST_VIEW mode which can only be set within the program. INST_VIEW mode is the mode you are in when you place a component in a design.

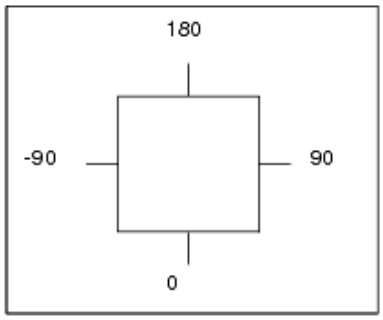
Syntax:

```
de_draw_port(x, y, angle[, power, portNumber, portName]);
```

where

x, y is the port location.

angle is the port angle; used to determine the angle of a connecting instance. The following chart shows the angle to specify for the pin so that the next component will flow away from the current component in the correct direction. Two pin components generally have pin 1 on the left-hand side and pin 2 on the right. In this case, pin 1 would have an angle of -90 degrees and pin 2 would have an angle of 90 degrees.



power is optional; power of pin.

portNumber is optional; next available unique port number used if not supplied.

portName is optional; unique port name.

Example:

```
de_draw_port(0, 0, -90);
```

***Where Used:* (ael)**

Layout

de_draw_rect()

Draws a rectangle in simulator units. Typically, used in artwork creation macros rather than in the *de_add_rectangle()* function. Returns a handle to the data group shape that was just created.

Syntax:

```
de_draw_rect(x1,y1, x2,y2);
```

where

x1,y1 is the lower left corner of the rectangle.

x2,y2 is the upper right corner of the rectangle.

Example:

```
de_draw_rect(0,0, 10,20);
// Or to capture the data group handle
decl dgHandle = de_draw_rect(0,0, 10,20);
```

***Where Used:* (ael)**

Schematic, Layout

de_draw_text()

Draws text at given location, with given font, height and angle, height and angle are in simulator units. Typically, used in artwork creation macros rather than in the *de_add_text()* function. Returns: none.

Syntax:

```
de_draw_text(font, x,y, height, angle, string);
```

where

font is an integer indicating text font number.

- 0 = Hershey Roman
- 1 = Hershey Roman Narrow

x,y is the location for the text string (lower left corner of first character).

height is the height of the text in user units.

angle is an angle in simulator angle units.

string is a text string.

Example:

```
de_draw_text(1, 12.3,45.6, 44, 90.0, "this is a text string");
```

***Where Used:* (ael)**

Schematic, Layout

de_dse_l2s()

Synchronizes the schematic with the layout, using the layout as the reference representation. Returns: none.

See also: *de_dse_s2l()* (ael).

Syntax:

```
de_dse_l2s(starting_item, x,y, ang, x_spac,y_spac );
```

where

starting_item is the name of the component to start the synchronization from. Usually set to "P1" for port 1.

x,y is the location to place the starting element when creating a layout or schematic the first time.

ang is the angle to place the starting element when creating a layout or schematic the first time.

x_spac, *y_spa* c is the spacing to use between components. A wire will be drawn to connect components that are spaced > 0 units apart. Schematic components are usually spaced 0.5 inches apart, layout 0.

Example:

```
de_dse_l2s("P1", -1, 1, 30, 0,0 ,0,0);
```

***Where Used:* (ael)**

Layout

de_dse_s2l()

Synchronizes the layout with the schematic, using the schematic as the reference representation. Returns: none.

See also: *de_dse_l2s()* (ael).

Syntax:

```
de_dse_s2l(starting_item, x,y, ang, x_spac, y_spac, layoutWinInstP);
```

where

starting_item is the name of the component to start the synchronization from. Usually set to "P1" for port 1.

x,y is the location to place the starting element when creating a layout or schematic the first time.

ang is the angle to place the starting element when creating a layout or schematic the first time.

x_spac, *y_spac* is the spacing to use between components. A wire will be drawn to connect components that are spaced > 0 units apart. Schematic components are usually spaced 0.5 inches apart, layout 0.

layoutWinInstP is the handle to the layout window to synchronize.

Example:

```
de_dse_s2l("P1", -1, 1, 30, 0,0, 0,0, layoutWinInstP);
```

Where Used: **(ael)**

Schematic

de_edit_annotation_attribute()

Edits attributes of selected instance's parameter annotation. Returns: none.

See also: *de_change_annotation_layer()* (ael).

Syntax:

```
de_edit_annotation_attribute(font, height, maxRows, isPoint);
```

where

font is a string indicating text font.

height is the height of text in user units or points.

maxRows is the number of rows per column of text.

isPoint informs whether height is in user units or point size, where:

- TRUE = height is in points
- FALSE = height is in user units

Example:

```
de_edit_annotation_attribute("Arial",10,15,TRUE);
```

Where Used: **(ael)**

Schematic

de_edit_item()

Allows a given item to be selected for editing. Returns a pointer to the instance.

See also: *de_end_edit_item()* (ael), *de_set_item_id()* (ael), *de_set_item_parameters()* (ael).

Syntax:

```
de_edit_item(id);
```

where

id is a string indicating unique id of the instance to edit.

Example:

```
// Select the instance with id R1
decl itemInfoOSP = de_edit_item ("R1");
// Change the instance id to R11
de_set_item_id(itemInfoOSP, "R11");
// Finish the editing (commit and display the editing changes)
de_end_edit_item(itemInfoOSP);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Set Meas Equation Variable String
Set Variable Value 2

***Where Used:* (ael)**

Schematic, Layout

de_edit_path_trace()

Modifies attributes of a selected path or trace in the current representation. Returns: none.

Syntax:

```
de_edit_path_trace(cornerType, width, cutoffRatio, curveRadius);
```

where

cornerType is the corner type.

- 1 = mitered
- 2 = square
- 3 = curved

width is the width of path or trace in user units.

cutoffRatio is the angle of miter (used only if cornerType is mitered).

curveRadius is the radius of the curve (used only if cornerType is curve).

Example:

```
de_edit_path_trace(1, 12.4, 20.0, 0);
```

***Where Used:* (ael)**

Layout

de_edit_symbol_pin()

Sets up the editing of a symbol pin. Returns: none.

Syntax:

```
de_edit_symbol_pin(name,number, angle, type[, pinPower]);
```

where

name is the name of pin.

number is an integer pin number. Each pin of a symbol must have a unique number.

angle is the pin angle. Used only in design synchronization when creating/updating schematic from layout. Determines the angle of the connecting part.

type is the pin direction: Input, Output, or Input/Output.

pinPower is optional; power of the pin.

Example:

```
de_set_edit_symbol_pin(10, 20);
de_edit_symbol_pin("Input", 1, 90.0, 1);
```

***Where Used:* (ael)**

Schematic, Layout

de_edit_text_attribute()

Edits the attributes of selected text components in the current representation. Returns: none.

Syntax:

```
de_edit_text_attribute(font, height, angle, just, absolute, isPoint);
```

where

font is a string indicating text font.

height is the height of the text in user units or points.

angle is the angle of text in degrees.

just is a field composed of the *or* of the first four bits of an integer. The first two bits represent the top, center, bottom vertical justification, while the next two represent the left, middle, right horizontal justification. The default justification is left, bottom (9 or 1001 binary).

absolute defines mode of text rotation, where:

- TRUE = text on a symbol or design does not rotate when the symbol or design is rotated
- FALSE = text on a symbol or design rotates when the symbol or design is rotated; default

isPoint informs whether height is in user units or point size, where:

- TRUE = height is in points
- FALSE = height is in user units

Example:

```
de_edit_text_attribute("Ariel For CAE", 10, 0, 9, 0, TRUE);
```

***Where Used:* (ael)**

Schematic, Layout

de_edit_text_string()

Edits text string in the current representation that was set with *de_set_edit_text()*.
Returns: none.

Syntax:

```
de_edit_text_string(newString);
```

where

newString is the new text string which replaces selected text strings.

Example:

```
de_set_edit_text(0,0,1);
de_edit_text_string("this is the new string");
```

Where Used: **(ael)**

Schematic, Layout

de_empty()

Empties an enclosed, filled shape in the current representation, creating a hole. Returns: none.

See also: *de_fill()* (ael).

Syntax:

```
de_empty(innerX, innerY, outerX, outerY);
```

where

innerX, innerY is the coordinate within select region of inner shape (hole).

outerX, outerY is the coordinate within select region of outer shape (enclosing shape).

Example:

```
de_empty(10, 20, 30, 60);
```

Where Used: **(ael)**

Schematic, Layout

de_end()

Completes a polygon, polyline, wire or trace command sequence. This command completes a shape. Use the *de_end_command()* to terminate the repeating command. Returns a handle to the data group shape that was just created.

Syntax:

```
de_end([thickness]);
```

thickness is optional, where:

- 1 = thin, default
- 2 = medium
- 3 = thick

Example:

```
de_add_polyline();
de_add_point(10,20);
de_add_point(40, 60);
de_add_point(20, 30);
de_end();
de_end_command();
// Or to capture the data group handle
de_add_polyline();
de_add_point(10,20);
de_add_point(40, 60);
de_add_point(20, 30);
decl dgHandle = de_end();
de_end_command();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Generate SnP Component
PI Attenuator
Polylines to Polygons

***Where Used:* (ael)**

Schematic, Layout

de_end_command()

Terminates a repeating command sequence. To properly playback any macro with repeating commands, the macro needs to terminate the command sequence with this function. Returns: none.

Syntax:

```
de_end_command();
```

Example:

```
de_add_polygon();
de_add_point(2.75,3.5);
de_add_point(3.5,3.5);
de_add_point(3.5,3.75);
de_add_point(4.0,3.75);
de_add_point(4.0,3.0);
de_add_point(2.75,3.0);
de_add_point(2.75,3.0);
de_end();
de_end_command();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Swap all Components with Themselves

Where Used: **(ael)**

Schematic, Layout

de_end_edit_item()

Commits and displays the editing changes of a given instance. Returns: none.

See also: *de_edit_item()* (ael).

Syntax:

```
de_end_edit_item(itemP);
```

where

itemP is a pointer to the item structure (the return value of de_edit_item()).

Example:

```
// Select the instance with id R1
decl itemInfoOSP = de_edit_item ("R1");
// Change the instance id to R11
de_set_item_id(itemInfoOSP, "R11");
// Finish the editing (commit and display the editing changes)
de_end_edit_item(itemInfoOSP);
```

***Where Used:* (ael)**

Schematic, Layout

de_export_design()

Exports the current layout or schematic in given format. Options are set via appropriate options file for the export format. Returns: none.

See also: *de_import_design()* (ael).

Syntax:

```
de_export_design(designType, "outputFileName");
```

where

designType is a numerical value indicating type of output design, where:

designType	Numeric Equivalent	Description
DE_HPIFF_FILE	11	IFF
DE_GDSII_FILE	1	GDSII Stream Format
DE_IGES_FILE	2	IGES
DE_HPGL2_FILE	3	HPGL/2
DE_MASK_FILE	4	Mask File (.msk)
DE_EGSGEN_FILE	9	EGS Generate Format
DE_EGSARC_FILE	10	EGS Archive Format
DE_GERBER_FILE	12	ACS MTOOLS Gerber
DE_GERBER_VIEWER	16	ACS MTOOLS Gerber Viewer
DE_DXF_FILE	13	ACS MTOOLS DXF
DE_DXF_ASM800_FILE	19	DXF (hierarchical)
DE_MGCPCB_FILE	-16	MGC/PCB

outputFileName is the file name where output will be written

Example:

The example outputs the current design in GDSII format.

```
de_export_design(DE_GDSII_FILE, "mmicAmp");
```

***Where Used:* (ael)**

Schematic, Layout

de_fill()

This function is used to convert a polygon with holes to a polygon without holes, and converts each hole in the original polygon to a polygon without holes. Returns: none.

Syntax

de_fill(x,y);

where,

- x,y is the point within select region of the polygon to operate on.

Example:

```
de_fill(20,30);
```

See also

de_empty() (ael).

***Where Used:* (ael)**

Schematic, Layout

de_find_arc_center()

Moves the cursor to the center of the arc or circle selected within the select region and enters the coordinates to the event that was previously installed. Returns: none.

Syntax:

```
de_find_arc_center(x,y);
```

where

x,y is the point within select region of arc center to find.

Example:

```
de_find_arc_center(20,30);
```

Where Used: *Where to Use AEL Functions* (ael)

Schematic, Layout

de_find_line_center()

Moves the cursor to the pin closest to the center of the line selected within the select region and enters that point to the event that was previously installed. Returns: none.

Syntax:

```
de_find_line_center(x,y);
```

where

x,y is the point within select region of line center to find.

Example:

```
de_find_line_center(20,30);
```

***Where Used:* (ael)**

Schematic, Layout

de_find_pin()

de_find_pin()

Moves the cursor to the pin closest to the point selected within the select region and enters that point to the event that was previously installed. Returns list (x,y), the coordinates of the pin within the pick region, else NULL.

Syntax:

de_find_pin(x,y);
where
x,y is the point within select region of pin to find.

Example:

```
de_find_pin(20,30);
```

Where Used: (ael)

***Where Used:* (ael)**

Schematic, Layout

de_find_vertex()

Moves the cursor to the pin closest to the vertex selected within the select region and enters that point to the event that was previously installed. Returns: none.

Syntax:

```
de_find_vertex(x,y);
```

where

x,y is the point within select region of vertex to find.

Example:

```
de_find_vertex(20,30);
```

***Where Used:* (ael)**

Schematic, Layout

de_fix_instances()

Fixes the position of an instance in the current representation and prevents design synchronization from re-positioning it. Returns: none.

See also: *de_free_instances()* (ael).

Syntax:

```
de_fix_instances([x,y]);
```

where

x,y is optional. Point within select region of instance to fix; if not specified, uses selected instances.

Example:

```
de_fix_instances(15, 20);
```

***Where Used:* (ael)**

Schematic, Layout

de_flatten()

Removes a single level of hierarchy. Returns: none.

See also: *de_instantiate()* (ael).

Syntax:

```
de_flatten();
```

Example:

```
de_flatten();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Flatten Design
Design has Layout?

***Where Used:* (ael)**

Schematic, Layout

de_free_instances()

Frees the position of an instance in the current representation and allows design synchronization to re-position it. Returns: none.

See also: *de_fix_instances()* (ael).

Syntax:

```
de_free_instances([x,y]);
```

where

x,y is optional. Point within select region of instance to free; if not specified, uses selected instances.

Example:

```
de_free_instances(15, 20);
```

Where Used: **(ael)**

Schematic, Layout

de_free_item()

Frees the data structure created by *de_init_item()* when it is no longer needed. Returns: NULL.

See also: *de_init_item()* (ael), *de_place_item()* (ael), *de_set_item_id()* (ael), *de_set_item_parameters()* (ael), *de_edit_item()* (ael), *de_end_edit_item()* (ael).

Syntax:

```
de_free_item(itemP);
```

where

itemP is a pointer to the item structure (the return value of *de_init_item()*).

Example:

```
// Initialize a resistor item
decl itemInfoOSP = de_init_item("R");
```

```
// Place the resistor
de_place_item(itemInfoOSP, 0.125, -1.625);
// Free the resistor item
itemInfoOSP = de_free_item(itemInfoOSP);
/* You should set the variable initialized in the de-init_item step to the
return value of the de_free_item so it resets to NULL. This allows future
de_place_item()'s with the no longer valid variable to behave as NOPs (not do
anything). */
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Export all Variables
PI Attenuator

***Where Used:* (ael)**

Schematic, Layout

de_get_data_parm()

Returns a string, the value of a given data reference or a named parameter belonging to a default component. Returns NULL if reference is not located.

Syntax:

```
de_get_data_parm(name, key, type);
```

where

name is the name of the default component (instance name).

key is the name of the component's parameter.

type is the default component's prefix, such as: MSUB, TEMP, TAND, PERM, etc.

Example:

The example retrieves the height parameter of the default MSUB.

```
decl hm;
hm = de_get_data_parm("MSUB10", "h", "MSUB");
```

***Where Used:* (ael)**

Schematic, Layout

de_group_edit_parameter_value()

Modifies selected instances with the parameter name to the value given. Returns: none.

Syntax:

```
de_group_edit_parameter_value(paramName. paramValue);
```

where

paramName is the name of the parameter.

paramValue is a string or real value of the parameter.

Example:

```
de_group_edit_parameter_value("L", "25 mil");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Export all Variables
Set Variable Values

Where Used: **(ael)**

Schematic, Layout

de_hide_or_display_component_name()

Toggles the instance component name in annotation to hide or display. This command is only available from the schematic window. Returns: none.

Syntax:

```
de_hide_or_display_component_name([x, y]);
```

where

x,y is optional. Coordinates within the select region of an instance to change the visibility of the component name. If not given, select instances will be changed to all hide or all display the component name.

Example:

```
de_select_all();
de_hide_or_display_component_name();
```

Where Used: **(ael)**

Schematic

de_import_design()

Imports a foreign design format. Translates a foreign file into a design file. Returns: none.

See also: *de_export_design()* (ael).

Syntax:

```
de_import_design(designType, overwrite, "inputFileName", "designFileName" [,
"defaultDesignName"]);
```

where

designType is a numerical value indicating type of output design, where: DE_SPICE_FILE = Spice File. The spice dialect for translation is set using the SpiceDialect Option in the options file *spice.opt* , where:

- 1 = SPICE2G
- 2 = SPICE3
- 3 = PSPICE (default)
- 4 = HSPICE
- 5 = HPSPICE
- DE_HPIFF_FILE = IFF
- DE_GDSII_FILE = GDSII Stream Format
- DE_IGES_FILE = IGES
- DE_HPGL2_FILE = HPGL/2
- DE_MASK_FILE = Mask File (.msk)
- DE_EGSGEN_FILE = EGS Generate Format
- DE_EGSARC_FILE = EGS Archive Format

overwrite is a numerical value indicating whether design file is overwritten.

- 0 = do not overwrite design file
- 1 = overwrite design file

inputFileName is a string, enclosed in quotes, indicating the file name of the file to be translated.

designFileName is a string, enclosed in quotes, indicating the file name of the design file to be generated.

defaultDesignName is a string, enclosed in quotes, indicating the default design name. Optional.

Example:

The example imports the current design in GDSII format.

```
de_import_design(DE_GDSII_FILE, 1, "graph1.hpg", "graph1");
```

***Where Used:* (ael)**

Schematic, Layout

de_init_item()

Initializes the instance, readying it for placement. Returns a pointer to the instance.

See also: *de_place_item()* (ael), *de_free_item()* (ael), *de_set_item_id()* (ael), *de_set_item_parameters()* (ael), *de_edit_item()* (ael), *de_end_edit_item()* (ael).

Syntax:

```
de_init_item(itemName);
```

where

itemName is the name of the instance to initialize.

Example:

```
// Initialize a resistor
decl itemInfoOSP = de_init_item("R");
// Place the resistor
de_place_item(itemInfoOSP, 0.125, -1.625);
// Free the resistor item
itemInfoOSP = de_free_item(itemInfoOSP);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and

Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Export all Variables
Insert Equations from File
PI Attenuator

***Where Used:* (ael)**

Schematic, Layout

de_insert_arrow()

Inserts an arrow draw as a polyline or polygon item.

Syntax:

```
de_insert_arrow(arrowNum, length, width, solid, x2, y2, x1, y1);
```

where

arrowNum is the number of arrowheads to be drawn.

length is the length of the arrowhead.

width is the width of the arrowhead.

solid is TRUE if the arrow should be drawn as a polygon; FALSE if the arrow should be drawn as a polyline.

x1, y1 and *x2, y2* are the coordinates of the end points of the arrows.

Example:

```
de_insert_arrow(2, 20, 6.67, FALSE, -220, -165, 20, 10);
```

***Where Used:* (ael)**

Schematic, Layout

de_insert_dimlin()

Inserts a dimension line.

Syntax:

```
de_insert_dimlin(x1, y1, x2, y2);
```

where

x1, y1 and *x2, y2* are the coordinates of the end points of the dimension line.

Example:

```
de_insert_dimlin(-220, -155, -70, 50);
```

Where Used: **(ael)**

Layout

de_instantiate()

Creates a new design from the selected group. The design is not saved to disk and it is not placed. This is the opposite of *de_flatten()*. Returns: none.

See also: *de_flatten()* (ael).

Syntax:

```
de_instantiate(newDesignName);
```

where

newDesignName is the name of the design to create.

Example:

```
de_instantiate("sub_amp");
```

Where Used: **(ael)**

Schematic, Layout

de_intersection()

Creates new polygons formed by the intersections of selected shapes (e.g., polygon, rectangle, circle, or path) on the same layer. Returns: true or false, which indicates if the

command was successful.

See also: *de_union()* (ael), *de_difference()* (ael).

Syntax:

```
de_intersection();
```

Example:

```
de_intersection();
```

Where Used: **(ael)**

Layout

de_last_view()

Recalls last view into the current window. Returns: none.

See also: *de_restore_view()* (ael), *de_store_current_view()* (ael), *de_delete_view()* (ael).

Syntax:

```
de_last_view();
```

Example:

```
de_last_view();
```

Where Used: **(ael)**

Schematic, Layout

de_mirror_x()

Mirrors selected objects around the X-axis, given a reference point. Returns: none.

Syntax:

```
de_mirror_x(x,y);
```

where

x,y is the reference point to mirror around.

Example:

```
de_mirror_x(10,20);
```

Where Used: **(ael)**

Schematic, Layout

de_mirror_y()

Mirrors selected objects around the Y-axis, given a reference point. Returns: none.

Syntax:

```
de_mirror_y(x,y);
```

where

x,y is the reference point to mirror around.

Example:

```
de_mirror_y(10,30);
```

Where Used: **(ael)**

Schematic, Layout

de_miter_vertex()

Creates a mitered edge on a polygon or polyline. Returns: none.

Syntax:

```
de_miter_vertex([x,y]);
```

where

x,y is optional. The point to convert into a mitered corner. If not specified, previously-selected point is converted.

Example:

```
de_set_miter_length(10);
de_miter_vertex();
```

***Where Used:* (ael)**

Layout

de_modify_arc_resolution()

Modifies the resolution of an arc. Returns boolean true or false.

Syntax:

```
de_modify_arc_resolution(resolution);
```

where

resolution is the resolution in degrees.

Example:

```
de_modify_arc_resolution(5);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Modify Circle Resolution

***Where Used:* (ael)**

Schematic, Layout

de_modify_break()

Converts selected polygons into polylines; that is, breaks a closed shape into an open shape. Returns: none.

See also: *de_modify_join()* (ael), *de_modify_explode()* (ael).

Syntax:

```
de_modify_break([x,y]);
```

where

x,y is optional. The point within selected region of object to break.

Example:

```
de_modify_break();
```

or

```
de_modify_break(10,20);
```

Where Used: **(ael)**

Schematic, Layout

de_modify_circle_radius()

Modifies the radius of a circle. If the absolute radius is less than or equal to zero, use delta radius instead. Returns boolean true or false.

Syntax:

```
de_modify_circle_radius(absRadius, deltaRadius);
```

where

absRadius is the radius as given.

deltaRadius is the change in radius.

Example:

```
de_modify_circle_radius(0, 0.5);
```

Where Used: **(ael)**

Schematic, Layout

de_modify_explode()

Converts selected polygons and polylines into two-point polyline segments. Each pair of vertices form a new shape. Returns: none.

See also: *de_modify_join()* (ael), *de_modify_break()* (ael).

Syntax:

```
de_modify_explode([x,y]);
```

where

x,y is optional. The point within selected region of object to explode.

Example:

```
de_modify_explode();
```
or

```
de_modify_explode(10,20);
```

Where Used: **(ael)**

Schematic, Layout

de_modify_join()

Joins selected polylines with coincident end points into a single polyline or polygon. If selected polylines form a closed shape, a polygon will be created. Returns: none. It also joins selected paths with coincident end points into a single path.

See also: *de_modify_explode()* (ael), *de_modify_break()* (ael).

Syntax:

```
de_modify_join([x,y]);
```

where

x,y is optional. The point within selected region of object to join.

Example:

```
de_modify_join();
```

or

```
de_modify_join(10,20);
```

Where Used: **(ael)**

Schematic, Layout

de_move()

Moves selected items in the current representation by a given amount. Returns: none.

See also: *de_move_break()* (ael).

Syntax:

```
de_move(dx, dy);
```

where

dx,dy is the delta to move the selected objects by.

Example:

```
de_move(20,-30);
```

Where Used: **(ael)**

Schematic, Layout

de_move_annotation()

Moves parameter annotation of a selected instance to a new location. Returns: none.

See also: *de_set_move_annotation()* (ael), *de_change_annotation_layer()* (ael), *de_edit_annotation_attribute()* (ael).

Syntax:

```
de_move_annotation(dx,dy);
```

where

dx,dy is the delta to move the annotation by.

Example:

```
de_set_move_annotation(10,10);
de_move_annotation(10,20);
```

Where Used: **(ael)**

Schematic

de_move_break()

Moves selected instances in the current representation, breaking any wire or trace connection. Returns: none.

See also: *de_break_connection()* (ael), *de_move()* (ael).

Syntax:

```
de_move_break(dx,dy);
```

where

dx,dy is the delta to move the selected instances.

Example:

```
de_move_break(10.5,12.3);
```

Where Used: **(ael)**

Schematic, Layout

de_move_to_layer()

Moves selected objects in the current representation to the current entry layer.

Note
The destination layer must be selectable.

Returns: none.

See also: *de_copy_to_layer()* (ael).

Syntax:

```
de_move_to_layer([layerNo]);
```

where

layerNo is optional; layer to move selected objects to. If layer is not specified, object is moved to current layer.

Example:

```
de_select_all();
de_move_to_layer();
```

***Where Used:* (ael)**

Schematic, Layout

de_net()

Creates a netlist report file. This is similar to a parts list in netlist format. Returns: none.

See also: *de_bom()* (ael), *de_parts()* (ael).

Syntax:

```
de_net(fileName, showDlg);
```

where

fileName is the name of report file to be created.

showDlg is whether or not to show the parts list report where:

- TRUE = show the report
- FALSE = do not show the report (the file is still recorded)

Example:

```
de_net("net");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Netlist with Node Names

***Where Used:* (ael)**

Schematic, Layout

de_new_datadisplay()

Sends the new window command to the data display server. Opens a data display window.
Returns: none.

Syntax:

```
de_new_datadisplay([dataSetName]);
```

where

dataSetName is the name of the dataset that will most likely be used. Optional.

Example:

```
de_new_datadisplay();
```

***Where Used:* (ael)**

Schematic, Layout

de_oversize()

Creates a new oversized or undersized shape from the selected closed shapes in the current representation. The shape will be oversized or undersized by the amount specified by *de_set_oversize()*. Returns TRUE if the operation is successful and FALSE if it fails.

See also: *de_set_oversize()* (ael), *de_scale()* (ael), *de_set_scale()* (ael).

Syntax:

```
de_oversize([copyFlag]);
```

where

copyFlag is optional where

- 0 = Copy before oversize (default)
- 1 = Oversize only

Example:

```
de_set_oversize(50,45);
de_oversize();  //Copy before Oversize
de_oversize(0); //Copy before Oversize
de_oversize(1); //Oversize
if ( !de_oversize() )
  return;
```

***Where Used:* (ael)**

Schematic, Layout

de_pan_window()

Re-centers the viewing window to the given point. Returns: none.

See also: *de_zoom_window()* (ael), *de_zoom_in_point()* (ael), *de_zoom_out_point()* (ael), *de_view_all()* (ael), *de_zoom_in_scale()* (ael), *de_zoom_out_scale()* (ael).

Syntax:

```
de_pan_window(x,y);
```

where

x,y is the point for new viewing window center.

Example:

```
de_pan_window(10, 20);
```

Where Used: **(ael)**

Schematic, Layout

de_parts()

Generates a parts list for the current representation and stores in the specified file. Opens the Parts List UI. Returns: none.

See also: *de_net()* (ael), *de_bom()* (ael).

Syntax:

```
de_parts(fileName, showDlg);
```

where

fileName is the name of file for storing parts list.

showDlg is whether or not to show the parts list report where:

- TRUE = show the report
- FALSE = do not show the report (the file is still recorded)

Example:

```
de_parts ("myParts");              //creates file myParts.p1
```

Where Used: **(ael)**

Schematic, Layout

de_parts_option_add_exclusion_items()

Adds an Exclusion List to the parts list options.

Syntax:

```
de_parts_option_add_exclusion_items (list ("MLIN"));
```

where

list is a list of items that will not appear in the parts list. This list is useful if parts have not been consistently flagged as BOM items. For this case, you wish to include everything except items in the exclusion list. In order to include everything, do not check the BOM flag.

Example:

```
de_parts_option_check_bom (FALSE);
```

or

```
de_parts_option_add_exclusion_items (DePartsLumpedWithArtworkElements);
```

***Where Used:* (ael)**

Schematic, Layout

de_parts_option_add_inclusion_items()

Adds an Inclusion List to the parts list options.

Syntax:

```
de_parts_option_add_inclusion_items (list ("res_smt"));
```

where

list is a list of items that will appear in the parts list. This list is useful if parts have not been consistently flagged as BOM items. For this case, specify to include only items flagged as BOM items, and add additional items in the inclusion list.

Inclusion items are treated as leaf-level parts and do not get flattened. For example, if an inclusion item is a hierarchical part, its subelements will not be included in the parts list.

Example:

```
de_parts_option_check_bom (TRUE);
```

or

```
de_parts_option_add_inclusion_items (list ("res_smt"));
```

***Where Used:* (ael)**

Schematic, Layout

de_parts_option_check_bom()

Checks the BOM Flag.

Syntax:

```
de_parts_option_check_bom (TRUE|FALSE);
```

where

TRUE Only include instances with attribute INST_SPECIAL set as ITEM_BOM_ITEM

FALSE Do not test for ITEM_BOM_ITEM (default)

Example:

```
de_parts_option_check_bom (TRUE);
```

***Where Used:* (ael)**

Schematic, Layout

de_parts_option_include_header()

Sets the parts list option to Include Header.

Syntax:

```
de_parts_option_include_header (TRUE|FALSE);
```

where

TRUE is the output header information (default).

FALSE is the output part data only.

Example:

```
de_parts_option_include_header (TRUE);
```

Where Used: **(ael)**

Schematic, Layout

de_parts_option_set_attribute_columns()

Sets the User Attribute Columns parts list options.

Syntax:

```
de_parts_option_set_attribute_columns (list ("INST_SPECIAL", "PART_NUM",
"Price"));
```

where

list is the list of attributes that will appear as columns in the parts list. The attributes can be user properties, user parameters, or instance attributes. The following instance attributes can appear in the report:

- INST_TYPE
- INST_SPECIAL
- INST_NAME
- INST_DESIGN_NAME
- INST_SYMBOL_NAME
- INST_BBOX
- INST_PROPERTY

Example:

```
de_parts_option_set_attribute_columns(list("PART_NUM"));
```

Where Used: **(ael)**

Schematic, Layout

de_parts_option_set_center_placement()

Sets the Component Placement X,Y Coordinates parts list options.

Syntax:

```
de_parts_option_set_center_placement (TRUE|FALSE);
```

where

TRUE Coordinates represent the center point of the instance bounding box. The bounding box does not include the annotation text.(default)

FALSE Coordinates represent the location of the instance origin. This is the origin of the associated symbol in the schematic or the footprint in the layout.

Example:

```
de_parts_option_set_center_placement(TRUE);
```

***Where Used:* (ael)**

Schematic, Layout

de_parts_option_set_delimeter()

Sets the Delimeter Character parts list options.

Syntax:

```
de_parts_option_set_delimeter (delimeter);
```

where

delimeter is used to separate column data (i.e. " ", ","). The default is NULL. If a NULL delimeter is specified, column widths will be determined by the longest data field and all data will be left justified.

Example:

```
/* Separate columns with commas */
de_parts_option_set_delimeter (",");
```

or, for example:

```
/* Auto-format */
de_parts_option_set_delimeter (NULL);
```

***Where Used:* (ael)**

***Where Used:* (ael)**

Schematic, Layout

de_parts_option_set_hierarchical()

Sets Hierarchical Reporting parts list option.

Syntax:

```
de_parts_option_set_hierarchical (TRUE|FALSE);
```

where

TRUE Produces a parts list containing instances from all levels of the hierarchy (default).

FALSE Produces a parts list containing instances from only the top level of hierarchy.

Example:

```
de_parts_option_set_hierarchical(FALSE);
```

***Where Used:* (ael)**

Schematic, Layout

de_parts_option_set_package_offset()

Sets the Package Offsets parts list option.

Syntax:

```
de_parts_option_set_package_offset (packageAttributeName, packageName, xOffset,
yOffset);
```

where

packageAttributeName is the name of the user attribute.

packageName is the value of the user attribute.

xOffset is the amount the origin on the x coordinate will be offset to make the placement coordinate.

yOffset is the amount the origin on the y coordinate will be offset to make the placement coordinate.

Example:

For each instance which has a user attribute named "Package", with attribute value "P1", the placement coordinate will be the origin offset by xOffset, yOffset.

```
de_parts_option_set_package_offset ("Package", "P1", 15, 0);
```

Where Used: **(ael)**

Schematic, Layout

de_parts_option_sort_by_component()

Sets the Sort by Component Name parts list option.

Syntax:

```
de_parts_option_sort_by_component (TRUE|FALSE);
```

where

TRUE Sort the parts list by the component name (default).

FALSE Parts are listed as they appear in the database.

Example:

```
de_parts_option_sort_by_component(FALSE);
```

Where Used: **(ael)**

Schematic, Layout

de_paste_from_buffer()

Copies the contents of a buffer to the current representation. Note that schematic instances cannot be copied to layout and vice-versa. Returns: none.

Syntax:

```
de_paste_from_buffer(x,y);
```

where

x,y is the point to paste origin of paste buffer.

Example:

```
de_paste_from_buffer(23.4, 56.98);
```

Where Used: **(ael)**

Schematic, Layout

de_place_design_template()

Inserts the design template setup with *de_set_design_template()* into current design. Returns: none.

See also: *de_set_design_template()* (ael).

Syntax:

```
de_place_design_template(x,y);
```

where

x,y is the X and Y location to place the template.

Example:

```
de_place_design_template(1.2, -1.2);
```

Where Used: **(ael)**

Schematic

de_place_item()

Places the instance that was initialized by *de_init_item()*. Returns: none.

See also: *de_init_item()* (ael), *de_free_item()* (ael), *de_set_item_id()* (ael), *de_set_item_parameters()* (ael), *de_edit_item()* (ael), *de_end_edit_item()* (ael).

Syntax:

```
de_place_item(itemP, xLoc, yLoc);
```

where

itemP is a pointer to the item structure (the return value of *de_init_item()*).

xLoc and *yLoc* are the x and y locations on the layout or schematic drawing area where the instance is to be placed.

Example:

```
// Initialize a resistor item
decl itemInfoOSP = de_init_item("R");
// Place three resistors in a row
de_place_item(itemInfoOSP, 0, 0);
de_place_item(itemInfoOSP, 1, 0);
de_place_item(itemInfoOSP, 1, 0);
// Free the resistor item
itemInfoOSP = de_free_item(itemInfoOSP);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Export all Variables
Insert Equations from File
PI Attenuator

***Where Used:* (ael)**

Schematic, Layout

de_place_port()

Places a symbol pin in the current representation's symbol view. Adds a pin to a symbol. The pin/ports attributes are set with *de_set_port()* command. Returns: none.

See also: *de_set_port()* (ael).

Syntax:

```
de_place_port(x,y);
```

where

x,y is the port location.

Example:

```
de_switch_view(SYMBOL_VIEW);
de_set_port("",1,0.0);
de_place_port(0.75,3.25);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Generate SnP Component

***Where Used:* (ael)**

Schematic, Layout

de_place_unplaced()

Places an instance that has not yet been placed in one representation. Returns: none.

Syntax:

```
de_place_unplaced(x,y);
```

where

x,y is the location in the other representation to place the instance.

Example:

```
de_select_unplaced(15, 25);
api_set_current_window(LAYOUT_WINDOW);
de_place_unplaced(55.0,-45.0);
```

***Where Used:* (ael)**

Schematic, Layout

de_playback_macro()

Executes an AEL or DEM file. Returns: none.

Syntax:

```
de_playback_macro(macroName);
```

where

macroName is the name of the macro file.

Example:

```
de_playback_macro("abc.ael");
de_playback_macro("def.dem");
```

Where Used: **(ael)**

Schematic, Layout

de_plot()

Creates a plot of the active design in the current window and sends it to the default printer or plotter. Returns: none.

Syntax:

```
de_plot();
```

Example:

```
api_set_current_window(SCHEM_WIN); // set schematic window
de_plot();
```

Where Used: **(ael)**

Schematic, Layout

de_plot_to_file()

Creates a plot of the active design and saves it to a file in a format that can be sent to a printer or plotter. The format of the file is HPGL2. The file adds the extension *.hgl* . Returns: none.

Syntax:

```
de_plot_to_file(fileName);
```

where

fileName is the filename for saving the plot.

Example:

```
api_set_current_window(SCHEM_WIN);
de_plot_to_file("myFile");
```

***Where Used:* (ael)**

Schematic, Layout

de_pop_outof_instance()

Returns to the previous design before a *de_push_into_instance()* command. Returns: none.

See also: *de_push_into_instance()* (ael).

Syntax:

```
de_pop_outof_instance();
```

Example:

```
de_pop_outof_instance();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.

Replace all GROUND's with Node Name
Swap all Components with Themselves

Where Used: **(ael)**

Schematic, Layout

de_push_into_instance()

Pushes into hierarchical instance reference. Opens the design referred to by the instance and sets it active. Returns: none.

See also: *de_pop_outof_instance()* (ael).

Syntax:

```
de_push_into_instance(x,y);
```

where

x,y is the point within select region of instance to push into.

Example:

```
de_push_into_instance(10,20);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Replace all GROUND's with Node Name
Swap all Components with Themselves

Where Used: **(ael)**

Schematic, Layout

de_refresh_view()

Redraws the screen of the active window. Returns: none.

Syntax:

```
de_refresh_view();
```

Example:

```
de_refresh_view();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Flatten Design
Polygons to Circles
Polylines to Polygons

Where Used: **(ael)**

Schematic, Layout

de_release_simulator()

Cancels the simulation and makes the simulator license available for other users on the network. Returns: none.

See also: *de_analyze()* (ael).

Syntax:

```
de_release_simulator();
```

Example:

```
de_release_simulator();
```

Where Used: **(ael)**

Schematic

de_remove_properties()

Removes properties of design group, port, or instance. Returns: none.

See also: *de_set_edit_property()* (ael), *de_add_property()* (ael).

Syntax:

```
de_remove_properties(editSelected);
```

where

editSelected is the mode for selecting items to remove.

- TRUE = edit only selected items
- FALSE = edit item selected by de_set_edit_property()

Example:

```
de_remove_properties(FALSE);
```

Where Used: **(ael)**

Schematic, Layout

de_restore_view()

Recalls specified view into the current window. Returns: none.

See also: *de_store_current_view()* (ael), *de_delete_view()* (ael).

Syntax:

```
de_restore_view(viewName);
```

where

viewName is the name of the view to recall.

Example:

```
de_restore_view("myView");
```

***Where Used:* (ael)**

Schematic, Layout

de_rotate_90()

Rotates an item being placed 90 degrees clockwise around pin 1. Returns: none.

Syntax:

```
de_rotate_90();
```

Example:

```
de_rotate_90();
```

***Where Used:* (ael)**

Schematic, Layout

de_rotate()

Rotates selected items in the current design context around a given point.
Returns: none.

Syntax:

```
de_rotate(x,y, angle[, keepConnected]);
```

Where,

- *x,y* is the point to rotate around.
- *angle* is the angle to rotate.
- *keepConnected* is an optional boolean TRUE or FALSE value that denotes if rotation should move wires to keep connectivity intact. This argument is FALSE by default if it is not provided.

The optional *keepConnected* argument is introduced in ADS 2011. This argument is ignored in ADS 2009 Update 1 and earlier version and the default behavior of not moving wires to keep connectivity intact is used.

Example:

```
de_rotate(10, 20, 45);
```

Where Used: **(ael)**

Schematic, Layout

de_rotate_center()

Rotates selected items in the current design context around the center of gravity if more than one object is selected. If only one object is selected, rotates around pin 1. Returns: none.

Syntax:

```
de_rotate_center(angle[, keepConnected]);
```

Where,

- *angle* is the angle in degrees.
- *keepConnected* is an optional boolean TRUE or FALSE value that denotes if rotation should move wires to keep connectivity intact. This argument is FALSE by default if it is not provided.

The optional *keepConnected* argument is introduced in ADS 2011. This argument is ignored in ADS 2009 Update 1 or earlier ADS releases and the default behavior of not moving wires to keep connectivity intact is used.

Example:

```
de_rotate_center(45);
```

Where Used: **(ael)**

Schematic, Layout

de_rotate_image()

Rotates an instance (element) being placed in a specified direction around pin 1. This command must be specified between the function *de_init_item()* and the function *de _ place_item()*. Returns: none.

Syntax:

```
de_rotate_image(direction);
```

where

direction is the string "UP", "DOWN", "LEFT", "RIGHT".

Example:

```
decl itemInfoOSP=de_init_item("R");
de_rotate_image("UP");
de_place_item(itemInfoOSP, 10, 20);
itemInfoOSP=de_free_item(itemInfoOSP);
```

Download Example Fie:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Export all Variables
Insert Equations from File
PI Attenuator

Where Used: **(ael)**

Schematic, Layout

de_save_all_designs()

Saves all designs in memory. Does not save untitled designs. Returns: none.

Syntax:

```
de_save_all_designs();
```

Example:

```
de_save_all_designs();
```

Where Used: **(ael)**

Schematic, Layout

de_save_design_template()

Saves the schematic representation of a design (or a design template) as a design template. The name of design template can be the name the design (or design template) currently has or can be a different name. Design templates are stored in special directories for access from any workspace. These directories are: CIRCUIT_TEMPLATE_DIR for Analog/RF Designs and ADSPTOLEMY_TEMPLATE_DIR for DSP Designs. Returns the name the design template was saved as.

Syntax

```
de_save_design_template(templateName, designHandle);
```

Where,

- *templateName* is the name to save the design template as; NULL if saving template as same name.
- *designHandle* is the handle of the design (or design template) to save.

Example

```
decl designHandle;
de_save_design_template("test1", designHandle);
```

Where Used (ael)

Schematic

de_scale()

Scales items in the current representation by a scale factor. Returns: none.

See also: *de_set_scale()* (ael), *de_set_oversize()* (ael), *de_oversize()* (ael).

Syntax:

```
de_scale(x,y);
```

where

x,y is the X and Y location to scale around. The scale factor is set with *de_set_scale()*.

Example:

```
de_scale(10, 15);
```

Where Used: **(ael)**

Schematic, Layout

de_search_and_replace()

Finds all references (depending on refType), given a variable or item reference, and highlights the item. Optionally, if searchOnlyFlag is set to 0, replaces the reference with the reference specified by replaceName. Returns: none.

Syntax:

```
de_search_and_replace(searchName, replaceName, refType, searchOnlyFlag);
```

where

searchName is the name of variable or item reference.

replaceName is the name of variable or item to search for or replace.

refType is the choice of reference type, where:

- 0 = item reference
- 1 = variable reference

searchOnlyFlag is the signal for searching and replace options, where:

- 0 = search and replace reference
- 1 = search for only

Example:

```
de_search_and_replace("TL1", "TL14", 0, 0);
```

Where Used: **(ael)**

Schematic, Layout

de_select_all()

Selects everything matching the select filter in the current window. Returns: none.

Syntax:

```
de_select_all();
```

Example:

```
de_select_all();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Flatten Design
Select All on Layer

Where Used: **(ael)**

Schematic, Layout

de_select_all_force()

Selects all items regardless of a layers protection status. Used in macros that need to reliably modify selected items regardless of the layer state. Returns: none.

Syntax:

```
de_select_all_force();
```

Example:

```
de_select_all_force();
```

Where Used: **(ael)**

Schematic, Layout

de_select_all_on_layer()

Selects everything on a specified layer matching the select filter in the current window. Returns: none.

Syntax:

```
de_select_all_on_layer(layerNameOrNum);
```

where

layerNameOrNum is the name of the layer or the layer number.

Example:

```
de_select_all_on_layer("cond");
de_select_all_on_layer(1);
```

***Where Used:* (ael)**

Schematic, Layout

de_select_by_name()

Selects instances by component name or instance name. Returns: none.

Syntax:

```
de_select_by_name(name, selectType);
```

where

name is the component name or instance name.

selectType is the choice for type of selection, where:

- 0 = selects by component name
- 1 = selects by instance name

Example:

```
de_select_by_name("MLIN", 0);
```

or

```
de_select_by_name("TL1", 1);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Export all Variables
Set Variable Values
Swap all Components with Themselves

Where Used: **(ael)**

Schematic, Layout

de_select_item()

Toggles a selection of an item within the select region in the current representation. Returns: none.

Syntax:

```
de_select_item(x,y);
```

where

x,y is the location of item to toggle.

Example:

```
de_select_item(10,20);
```

Where Used: **(ael)**

Schematic, Layout

de_select_range()

Selects all items in the current representation enclosed by a given window range. Returns: none.

Syntax:

```
de_select_range(x1,y1, x2,y2, [deselectFlag, buttonState]);
```

where

x1,y1 is the first corner of the select window.

x2,y2 is the second, opposite corner.

deselectFlag is optional. Signals whether to deselect all objects first, where:

- 0 = default; do not deselect all objects first
- 1 = deselect all objects first

buttonState is optional. Specification of select mode, where:

- 0 = unknown. If x1==x2 and y1==y2 a single item is selected; otherwise, all objects in window range are selected
- 1 = Single click to select a single object
- 2 = Double click to select a single object; deselect flag forced to 1
- 3 = Press and drag to select all objects in window range

Example:

```
de_select_range(3.25,2.75, 3.25,2.75);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Find Close Vertexes
Modify Circle Resolution
Select XY on Layer

Where Used: **(ael)**

Schematic, Layout

de_select_unplaced()

Selects an unplaced instance in the current representation to place in the other representation (from schematic to layout or layout to schematic). Returns: none.

Syntax:

```
de_select_unplaced(x,y);
```

where

x,y is the point within select region of an unplaced instance.

Example:

```
de_select_unplaced(3.375,4.125);
api_set_current_window(LAYOUT_WINDOW);
de_place_unplaced(35.0,-40.0);
```

Where Used: **(ael)**

Schematic, Layout

de_select_window()

Selects all items (matching the select filter) in the current representation enclosed by given window. Returns: none.

Syntax:

```
de_select_window(x1,y1, x2,y2);
```

where

x1,y1 is the point describing first corner selection window.

x2,y2 is the point describing opposite corner of selection window.

Example:

```
de_select_window(10,20, 40,60);
```

Where Used: **(ael)**

Schematic, Layout

de_set_design_template()

Sets up the design template for insertion into current design by *de_place_design_template()*. Returns: none.

See also: *de_place_design_template()* (ael).

Syntax:

```
de_set_design_template(templateName);
```

where

templateName is the name of the design template to be inserted.

Example:

```
de_set_design_template("mytemplate");
```

***Where Used:* (ael)**

Schematic, Layout

de_set_edit_property()

Sets the data group, port, or instance for property editing. Returns: none.

See also: *de_remove_properties()* (ael), *de_add_property()* (ael).

Syntax:

```
de_set_edit_property([x,y]);
```

where

x,y is optional. Location coordinates of data group, port, or instance for property editing. If if not specified, uses first selected data group, port, or instance that is found.

Example:

```
de_set_edit_property();
```

***Where Used:* (ael)**

Schematic, Layout

de_set_edit_symbol_pin()

Sets the symbol pin whose attributes are to be edited. Returns TRUE if symbol pin is found, FALSE if symbol pin is not found.

See also: *de_edit_symbol_pin()* (ael).

Syntax:

```
de_set_edit_symbol_pin(x,y);
```

where

x,y is the location of symbol whose attributes are to be edited.

Example:

```
de_set_edit_symbol_pin(3.5, 4.125);
```

Where Used: **(ael)**

Schematic, Layout

de_set_edit_text()}

Sets the text string in the current representation to be edited by *de_edit_text_string()*. Returns: none.

Syntax:

```
de_set_edit_text(x,y, useSelected);
```

where

x,y is the point within select region of text to edit. If not given, first selected text is used.

useSelected is TRUE to set selected text, FALSE to use x,y location entered.

Example:

```
de_set_edit_text(10,20, 0);
```

```
de_edit_text_string("this is the new string");
```

***Where Used:* (ael)**

Schematic, Layout

de_set_inst_pin_order_property()

Syntax:

```
de_set_inst_pin_order_property(instP, pinListString);
```

where

instP is the handle to the instance.

pinListString is a string which contains the pin numbers separated by spaces.

Example:

```
de_set_inst_pin_order_property(instP, "3, 1, 2")
```

***Where Used:* (ael)**

Schematic, Layout

de_set_item_id()

Sets the ID of the item that is either being readied to be placed (from *de_init_item()*) or is being placed (from *de_edit_item()*). Returns: none.

See also: *de_init_item()* (ael), *de_place_item()* (ael), *de_free_item()* (ael), *de_set_item_parameters()* (ael), *de_edit_item()* (ael), *de_end_edit_item()* (ael).

Syntax:

```
de_set_item_id(itemP, id);
```

where

itemP is a pointer to the item structure (the return value of *de_init_item()* and *de_edit_item()*).

id is the new unique id for the instance.

Example:

```
// Initialize a resistor
decl itemInfoOSP = de_init_item("R");
// Set the resistor id
de_set_item_id(itemInfoOSP, "R99");
// Place the resistor
de_place_item(itemInfoOSP, 0.125, -1.625);
// Free the resistor item
itemInfoOSP = de_free_item(itemInfoOSP);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
PI Attenuator

Where Used: **(ael)**

Schematic, Layout

de_set_item_parameters()

Sets the parameter values for a given item that has been selected by the *de_init_item()* or *de_edit_item()*. Returns: none.

See also: *de_init_item()* (ael), *de_place_item()* (ael), *de_free_item()* (ael), *de_set_item_id()* (ael), *de_edit_item()* (ael), *de_end_edit_item()* (ael).

Syntax:

```
de_set_item_parameters(itemP, listOfParameters);
```

where

itemP is a pointer to the item structure (the return value of *de_init_item()* and *de_edit_item()*).

listOfParameters is a complete list of parameters in the order they are to be set.

Example:

```
// Select the instance with id R1
```

```
decl itemInfoOSP = de_edit_item ("R1");
// Change the instance parameters to 99 Ohm (from 50 Ohm)
de_set_item_parameters(itemInfoOSP, list(prm("StdForm", "99 Ohm"),
prm("StdForm", ""), prm("Yes"), prm("StdForm", ""), prm("StdForm", ""),
prm("StdForm", ""), prm("StdForm", "")));
// Finish the editing (commit and display the editing changes)
de_end_edit_item(itemInfoOSP);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Export all Variables
Insert Equations from File
PI Attenuator

Where Used: **(ael)**

Schematic, Layout

de_set_move_annotation()

Selects an instance to have its annotation moved. Returns: none.

See also: *de_move_annotation()* (ael).

Syntax:

```
de_set_move_annotation(x,y);
```

where

x,y is the point within the instance to have its annotation moved.

Example:

```
de_set_move_annotation(10,10);
de_move_annotation(10,20);
```

Where Used: **(ael)**

Schematic

de_set_origin()

Resets the origin of a representation (resets the 0,0 point). Returns: none.

Syntax:

```
de_set_origin(x,y);
```

where

x,y describes the new origin point for the representation.

Example:

```
de_set_origin(25, 30);
```

Where Used: **(ael)**

Schematic, Layout

de_set_port()

Sets attributes of a symbol port (pin) before it is created for the current window. Returns: none.

See also: *de_place_port()* (ael).

Syntax:

```
de_set_port(portName, portNum, portAngle[, type, power]);
```

where

portName is the port name (unused-reserved for future use).

portNum is the port number (integer > 0).

portAngle is the angle to use for instance connected by abutment to this pin.

type is optional; INPUT=0, OUTPUT=1, IN/OUT=2 (default is 0)

power is optional; power of pin.

Example:

```
de_set_port("", 1, 0.0);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Generate SnP Component

***Where Used:* (ael)**

Schematic, Layout

de_set_simulation_dataset()

Sets the name of the dataset used for simulation. Returns: none.

See also: *de_set_simulation_host()* (ael).

Syntax:

```
de_set_simulation_dataset(datasetName);
```

where

datasetName is the dataset name.

Example:

```
de_set_simulation_dataset("myDataSet");
de_set_simulation_host("local");
de_analyze();
```

***Where Used:* (ael)**

Schematic

de_set_simulation_host()

Sets the name of the host computer to be used for simulation. Returns: none.

See also: *de_set_simulation_dataset()* (ael).

Syntax:

```
de_set_simulation_host(hostName);
```

where

hostName is the host computer name; "local" for local computer.

Example:

```
de_set_simulation_dataset("myDataSet");
de_set_simulation_host("local");
de_analyze();
```

***Where Used:* (ael)**

Schematic

de_set_swap_template_instance()

Sets the name of an item to swap within the function *de_swap_instance()*. Returns: none.

Syntax:

```
de_set_swap_template_instance(itemName);
```

where

itemName is a string; the name of an item.

Example:

```
//Select all MLINs, replace with TLIN.
de_select_by_name("MLIN", 0);
de_set_swap_template_instance("TLIN");
de_swap_instances(TRUE);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Swap all Components with Themselves

Where Used: **(ael)**

Schematic, Layout

de_shove()

Shove by a distance, vertices or whole polygons, whole text, and whole instances that are selected and lie on vector side of dividing line formed perpendicular to the vector. Returns: none.

Syntax:

```
de_shove(x, y, angle, dist);
```

where

x,y is the point on dividing line.

angle is the angle of vector that indicates direction or movement.

dist is the distance to shove.

Example:

```
de_shove(0, 0, 90, 10);
```

Where Used: **(ael)**

Schematic, Layout

de_show_equiv_inst()

Highlights the equivalent instance in another representation to a given instance. Returns: none.

See also: *de_show_connected()* , *de_show_fixed()* , de_show_unplaced() .

Syntax:

```
de_show_equiv_inst(x,y);
```

where

x,y is the point within select region of instance.

Example:

```
de_show_equiv_inst(10,20);
```

Where Used: **(ael)**

Schematic, Layout

de_snap()

Force the vertices of selected shapes to the nearest snap grid; forces pin 1 of selected instances to nearest grid point. Returns: none.

Syntax:

```
de_snap([x,y]);
```

where

x,y is optional. Point within select region of an object to snap.

Example:

```
de_snap();
```

or

```
de_snap(10,20);
```

Where Used: **(ael)**

Schematic, Layout

de_split()

Divides selected polygons, rectangles, circles, or paths into multiple shapes using a defined rectangular region. This command will fail and display an error if traces or instances are selected. All other types are silently ignored. Returns: 1 if successful, otherwise 0.

Syntax:

```
de_split(x1, y1, x2, y2);
```

where

x1, y1 is the first corner of the rectangular region.

x2, y2 is the second, opposite corner of the rectangular region.

Example:

```
de_select_all();
de_split(0, 0, 10, 10);
```

Where Used: **(ael)**

Layout

de_split_tlin()

(For Layout only.) Splits a transmission line element into two of the same elements at a given point, adjusting the new elements length parameters. Works with MLIN and SLIN components. Returns: none.

See also: *de_tap_tlin()* (ael), *de_stretch_tlin()* (ael).

Syntax:

```
de_split_tlin(x,y);
```

where

x,y is the point enclosed by a transmission line element (SLIN or MLIN) indicating the element to split as well as the point where the split should occur.

Example:

```
de_split_tlin(10,20);
```

Where Used: **(ael)**

Layout

de_step_and_repeat()

Copies and places selected items multiple times in rows and columns. Returns: none.

Note
X and Y spacing is defined as the distance between the outer edges of the respective bounding boxes.

Syntax:

```
de_step_and_repeat(x,y);
```

where

x,y is the location to place copies.

Example:

```
de_set_step_and_repeat(0.5, 0.5, 2, 2, FALSE);
de_step_and_repeat (10, 20);
```

Where Used: **(ael)**

Schematic, Layout

de_store_current_view()

Assigns a name to a view and stores view window coordinates. The view may be recalled by *de_restore_view()*. Returns: none.

See also: *de_restore_view()* (ael), *de_delete_view()* (ael).

Syntax:

```
de_store_current_view(viewName);
```

where

viewName is the name assigned to view.

Example:

```
de_store_current_view("myView");
```

Where Used: **(ael)**

Schematic, Layout

de_stretch()

Stretches or moves an edge of a given shape in the current representation. Returns: none.

Syntax:

```
de_stretch(oldX, oldY, newX, newY, maintainAngle);
```

where

oldX,oldY is the point on the edge to stretch.

newX,newY indicates the delta and direction to stretch the edge.

maintainAngle is TRUE if the adjacent angles of the edge to stretch are to be kept, FALSE otherwise.

Example:

```
de_stretch(0,0, 20,20, TRUE);
```

Where Used: **(ael)**

Schematic, Layout

de_stretch_dimlin()

Stretches a dimension line. Works with MLIN and SLIN elements. Returns: none.

See also: *de_tap_tlin()* (ael), *de_split_tlin()* (ael).

Syntax:

```
de_stretch_dimlin(x1, y1, x2, y2);
```

where

x1, y1 is the coordinate of the endline to stretch from.

x2, y2 is the coordinate of the new position for the endline to stretch to.

Example:

```
de_stretch_dimlin(-65, 55, -13.9933, 19.2953);
```

Where Used: **(ael)**

Layout

de_stretch_tlin()

(For Layout only) Stretches (increases or decreases its length) a given transmission line element in the current representation. Works with MLIN and SLIN elements. Returns: none.

See also: *de_tap_tlin()* (ael), *de_split_tlin()* (ael).

Syntax:

```
de_stretch_tlin(oldX, oldY, newX, newY);
```

where

oldX, oldY is the point on edge of transmission line element to stretch.

newX, newY is the delta and direction to stretch the edge is determined from this point.

Example:

```
de_stretch_tlin(19.8,18.45, 30.4, 18.45);
```

Where Used: **(ael)**

Layout

de_swap_instances()

Replaces all selected instances in the current window with the instance set with the function *de_set_swap_template_instance()*. Returns: none.

See also: *de_set_swap_template_instance()* (ael).

Syntax:

```
de_swap_instances(replaceID);
```

where

replaceID signals whether to replace ID of selected instances, where:

- FALSE = preserve the item ID
- TRUE = create a new ID

Example:

```
//Select all MLINs, replace with TLIN with the same instance name
de_select_by_name("MLIN", 1);
de_set_swap_template_instance("TLIN");
de_swap_instances(FALSE);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Swap all Components with Themselves

***Where Used:* (ael)**

Schematic, Layout

de_tap_tlin()

(For Layout only) Taps a transmission line element (MLIN or SLIN) in the current representation, and inserts a tee element (MTEE or STEE). The W3 parameter's value is set with the function *de_set_tap_length()* . Returns: none.

See also: *de_stretch_tlin()* (ael), *de_split_tlin()* (ael).

Syntax:

```
de_tap_tlin(x,y);
```

where

x,y is the point on a transmission line to split element and insert tee.

Example:

```
de_tap_tlin(23,34);
```

***Where Used:* (ael)**

Layout

de_undo()

Undoes the effect of the last draw or edit command in the current window. Returns: none.

Syntax:

```
de_undo();
```

Example:

```
de_undo();
```

Where Used: **(ael)**

Schematic, Layout

de_undo_vertex()

Undoes or removes the last vertex entered with the *de_add_point()* command. Returns: none.

Syntax:

```
de_undo_vertex();
```

Example:

```
de_undo_vertex();
```

Where Used: **(ael)**

Schematic, Layout

de_unhighlight_instances()

Removes the highlighting of any highlighted instances in the given design. Opposite of

de_highlight_instance(). Returns: none.

Syntax:

```
de_unhighlight_instances(designName, repType);
```

where

designName is the name of the design.

repType is the type of representation, where:

- REP_SCHEM = Schematic representation
- REP_LAY = Layout representation

Example:

```
de_unhighlight_instances("myamp", 1);
```

Where Used: **(ael)**

Schematic, Layout

de_union()

Creates new polygons formed by the union of selected shapes (e.g., polygon, rectangle, circle, or path) on the same layer. Returns: true or false, which indicates if the command was successful.

See also: *de_intersection()* (ael), *de_difference()* (ael).

Syntax:

```
de_union();
```

Example:

```
de_union();
```

Where Used: **(ael)**

Layout

de_update_tune_parameters()

Modifies tuned parameter values. Returns: none.

See also: *de_analyze_tune()* (ael), *de_tune_deinit()* (ael).

Syntax:

```
de_update_tune_parameters(designName, repType, valueList list(instName1,
paramName1, value1, ...));
```

where

designName is the name of the design.

repType is the type of representation where:

- REP_SCHEM = Schematic representation
- REP_LAY = Layout representation

valueList is a list of triplets (e.g, list(instName1, paramName1, value1, ...)) where:

- instName1 = instance ID
- paramName1 = parameter name
- value1 = new value of parameter as string or real

Example:

```
de_update_tune_parameters("MyDesign", REP_SCHEM, list("TL3", "W", "50 mil",
"Term1", "Z", "50 ohm"));
```

***Where Used:* (ael)**

Schematic, Layout

de_vertex_to_arc()

Converts a vertex on a shape to an arc. The arc radius is set with the *de_set_arc_radius()* command. Returns: none.

Syntax:

```
de_vertex_to_arc([x,y]);
```

where

x,y is optional. Point within select region of the vertex to convert.

Example:

```
de_vertex_to_arc(23,45);
```

***Where Used:* (ael)**

Schematic, Layout

de_view_all()

Expands the view window to view all data in the current window. Returns: none.

Syntax:

```
de_view_all();
```

Example:

```
de_view_all();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
PI Attenuator
Replace all GROUND's with Node Name
Swap all Components with Themselves

***Where Used:* (ael)**

Schematic, Layout

de_zoom_in_point()

Zooms in (double magnification) at the location specified on the current window. Returns: none.

See also: *de_zoom_out_point()* (ael).

Syntax:

```
de_zoom_in_point(x,y);
```

where
x,y is the point to zoom in to.

Example:

```
de_zoom_in_point(0,0);
```

***Where Used:* (ael)**

Schematic, Layout

de_zoom_in_scale()

Zooms in by factor specified on the current window. Returns: none.

See also: *de_zoom_out_scale()* (ael).

Syntax:

```
de_zoom_in_scale(scaleFactor);
```

where

scaleFactor is the integer specifying zoom factor

Example:

```
de_zoom_in_scale(2);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Find Close Vertexes

***Where Used:* (ael)**

Schematic, Layout

de_zoom_out_point()

Zooms out (double magnification) at the location specified on the current window.
Returns: none.

See also: *de_zoom_in_point()* (ael).

Syntax:

```
de_zoom_out_point(x,y);
```

where

x,y is the point to zoom out from.

Example:

```
de_zoom_out_point(0,0);
```

***Where Used:* (ael)**

Schematic, Layout

de_zoom_out_scale()

Zooms out by factor specified on the current window. Returns: none.

See also: *de_zoom_in_scale()* (ael).

Syntax:

```
de_zoom_out_scale(scaleFactor);
```

where

scaleFactor is the integer specifying zoom factor

Example:

```
de_zoom_out_scale(2);
```

Where Used: **(ael)**

Schematic, Layout

de_zoom_window()

Describes a region to display in the current window. Returns: none.

Syntax:

```
de_zoom_window(x1,y1, x2,y2);
```

where

x1,y1 is the first corner of zoom window.

x2,y2 is the second, opposite corner of zoom window.

Example:

```
de_zoom_window(-20, -30, 50, 60);
```

Where Used: **(ael)**

Schematic, Layout

Component Definition Functions

This section describes each Component Definition function in detail. The functions are listed in alphabetical order.

create_compound_form()

Creates a new compound form and stores the form in the dictionary for the current simulator type. Compound forms represent parameter values more complex than just strings. A compound form contains one or more parameter values. Returns: none.

Syntax:

```
create_compound_form(formName, label[, dialogDataStr], attribute,
netlistFormat, displayFormat, parameter1, parameter2,..., parameterN);
```

where

formName is a string, the unique form name.

label is a string, a descriptive string for the form.

dialogDataStr is optional; a string, that can be any dialog data. For the standard Edit Component dialog box, this field can be used as the name of the table entry field. This argument indicates to the dialog the gui component to use to breakdown the parm value input. For example, the standard Edit Component dialog supports these gui configurations:

- "StdForm"
- "StringAndReference"
- "SingleTextLine"
- "InstSelectionForm"
- "Node SetForm"
- "FileBasedForm"
- "ReadFileForm"

Note
This argument can be omitted if dialogDataStr is the same as formName. For complete listing, see *$HPEESOF_DIR/de/ael/pde_gemini.ael*.

attribute is an integer and should be set to 0.

netlistFormat is the format string to netlist the parameter value as. Refer to

Format Strings (ael).

displayFormat is the format string to display in a schematic. Refer to *Format Strings* (ael).

parameterN is one or more parameters, created with *create_parm()*.

Example:

This example creates a form for describing variables. The variable has a default name *X* and a default value *1.0*.

```
create_text_form("VarNameForm", "Variable Name", 0, "%v", "%v", NULL, validate_var_name, NULL);
create_text_form("VarValueForm", "Variable Value", 0, "%v", "%v", NULL, validate_var_value,
UNITLESS_UNIT);
create_form_set("VarNameForms", "VarNameForm");
create_form_set("VarValueForms", "VarValueForm");
create_compound_form("VarFormStdForm", "Standard", "StdForm", 0,
   "%0s=%1s", "%0s=%1s",
   create_parm("VarName","Variable Name",0,"VarNameForms",UNITLESS_UNIT),
       prm("VarNameForm", "X")),
   create_parm("VarValue","Variable Value",0,"VarValueForms",UNITLESS_UNIT,
       prm("VarValueForm", "1.0")));
```

Where Used: **(ael)**

Schematic

create_constant_form()

Creates a new constant form and stores the form in the dictionary for the current simulator type. Constant forms are used to describe one or more constant values. A parameter using constant form displays a list of values from which to choose, rather than a field for typing in the value. This is used by components with parameters whose allowable values are a list of constants. Returns: none.

Syntax:

create_constant_form(name, label, attribute, netlistFormat, displayFormat);

where

name is a string representing the form name.

label is a descriptive label for the form.

attribute is an integer and is normally set to 0. For discrete valued parts this value is set to 68, which is the sum of the FORM_DISCRETE attribute (value = 64) and the FORM_TUNABLE attribute (value = 4). For more information about discrete valued parts, see *Discrete Valued Parts Required AEL Definitions* (ael).

netlistFormat is the format string to netlist the parameter value as. Refer to

Format Strings (ael).

displayFormat is the format string to display in a schematic. Refer to *Format Strings* (ael).

Example:

This example creates forms for a parameter that has either *yes* or *no* values. *Yes* is netlisted as a 1 and *no* as a 0, but the strings *yes* and *no* are displayed in the schematic and the dialog box. The forms are combined into a single formset with the *create_form_set()* function.

```
create_constant_form("y_n1", "YES", 0, "1", "yes");
create_constant_form("y_n0", "NO", 0, "0", "no");
create_form_set("yes1_no0", "y_n1", "y_n0");
```

***Where Used:* (ael)**

Schematic

create_form_set()

Creates a set of forms for use by the *create_parm()* function. A form set describes the set of allowable forms the value of a parameter may take on. A number of predefined form sets exist. Returns: none.

See also *Connectivity Objects* (ael) and *DesignContext in AEL* (ael).

Syntax:

```
create_form_set(name, formName1, formName2,.... formNameN);
```

where

name is the name of the form set. Used by the *create_parm()* function.

formName is one or more form names.

Note
If you redefine an existing format to include different forms, then there may be backwards-compatibility problems when you open a design with components whose parameters were created with the old form set definitions; for example:
(1) Going from a non-compound to a compound form (by way of the *create_compound_form()* function)
(2) Going from a compound form to a non-compound form

Example:

This example creates a form set composed of the _y_n1_ and _y_n0_ forms. These allow a parameter value to take on a *yes* or *no* value.

```
create_constant_form("y_n1", "YES", 0, 1, "YES");
create_constant_form("y_n0", "NO", 0, 0, "NO");
create_form_set("yes1_no0", "y_n1", "y_n0");
```

***Where Used:* (ael)**

Schematic

create_item()

Creates a new component definition and stores it in the library's dictionary. This function defines the attributes and parameters that are common to all views of the cell. Returns: none.

Note
Some parameters of create_item are obsolete and are only used when converting ADS 2009 Update 1 and older designs.

Syntax

create_item(name, label, prefix, attrib, priority, iconName, dialogName, dialogData, netlistFormat, netlistData, displayFormat, symbolName, artworkType, artworkData[, extraAttribute, cbList, parameterN]);

Where,

name is a string; a unique name of the component.

label is a string; a descriptive label for the component. This is also used as the balloon help for the component palette selection. The label should not be longer than 80 characters (a Win32 restriction).

prefix is a string; the prefix for the instance name (e.g., "TL" for name "TL1") used when the component is placed.

attrib is an attribute code-the attribute choices are listed in *create_item() Attribute Choices*. To set multiple attributes, sum their numeric equivalent values. For example, to set both ITEM_BEND (2048) and ITEM_BOM_ITEM (262144), add their numeric equivalent values and set *attrib* to the combined value (264192).

priority is an integer; indicates special netlist handling. Specified as NULL or -1 for all components except the netlist include components. For netlist include components, set the netlisting priority to zero, to avoid illegal nested subcircuits.

iconName is a string; name of the bitmap file used for component button in a palette.

dialogName is a string; name of the dialog, usually *standard_dialog*. The dialog is used to edit the parameters and/or some other attributes of the component.

dialogData is a string; passed to the dialog creating function. Usually "*". Not

really used.

netlistFormat is a string; used to control the netlist format for the component. Usually specified with the variable *standard_netlist* or *ComponentNetlistFmt*.

netlistData is a string; used in conjunction with the netlist format string. %d.

displayFormat is a string; used to control the display of the component annotation in the schematic. Usually specified with the variable standard_symbol. Currently this format string also dictates how on-screen editing works.

symbolName is a string (Only used during translation); the name of the schematic symbol for the component.

artworkType is an integer (Only used during translation); indicates the type of artwork for the component, where:

- 0 = no artwork
- 1 = fixed artwork
- 2 = ael generated artwork
- 3 = synchronized

artworkData is a string (Only used during translation); the name of the artwork function or design containing the artwork.

- For fixed artwork, it should be a layout artwork design name.
- For macro artwork, it should be an AEL artwork generation function.
- If artwork Type = 1, string is set to the name of the design file containing the artwork

extraAttribute is optional; an extension to the attribute code. Use the function *dm_get_item_definition()* to retrieve this attribute. Attribute choices are listed in *create_item() Extra Attribute Choices*.

cbList is optional; the list of callbacks. For example, `list( dm_create_cb ("...", "..."), ...);`
Currently supported callbacks are: ITEM_NETLIST_CB

parameterN is optional; the list of parameters. Return value from the *create_parm()* command.

Note
If you redefine a component to use different parameters (by way of the *create_parm()* command), and then you open a design that contains components created with the older component definition, it will not work for the following cases:
(1) Going from a non-PARM_REPEATED to a PARM_REPEATED parameter
(2) Going from a PARM_REPEATED to a non-PARM_REPEATED parameter

Example

```
/* TLIN */
create_item("TLIN",                                    // name
```

```
"Ideal 2-Terminal Transmission Line",   // label
"TL",                                   // prefix
0,                                      // attribute
NULL,                                   // priority
"TLIN",                                 // iconName
standard_dialog,                        // dialogName
"*",                                    // dialogData
ComponentNetlistFmt,                    // netlistFormat
"TLIN",                                 // netlistData
ComponentAnnotFmt,                      // displayFormat
"SYM_TLin",                             // symbolName
no_artwork,                             // artworkType
NULL,                                   // artworkData
ITEM_PRIMITIVE_EX,                      // extraAttrib
create_parm ("Z", "Characteristic Impedance", PARM_OPTIMIZABLE |
             PARM_STATISTICAL, "StdFileFormSet",
             RESISTANCE_UNIT, prm("StdForm","50.0")),
create_parm ("E", "Electrical Length", PARM_OPTIMIZABLE |
             PARM_STATISTICAL, "StdFileFormSet", ANGLE_UNIT,
             prm("StdForm","90")),
create_parm ("F", "Reference Frequency for Electrical Length",
             PARM_OPTIMIZABLE | PARM_STATISTICAL, "StdFileFormSet",
             FREQUENCY_UNIT, prm("StdForm","1 GHz")));
```

create_item() Attribute Choices

Attribute	Numeric Equivalent	Description
No attribute set	0	Only the instance line is netlisted; the simulation behavior is defined by a user-compiled model, model card, or subcircuit. For the latter two options, a Netlist Include must be employed to reference the model card or subcircuit.
ITEM_BEND	2048	Component is a transmission line bend.
ITEM_BOM_ITEM	262144	Component is included in a BOM.
ITEM_DESIGN_INST	16	Component refers to a subdesign. When set, the component is netlisted as a subcircuit with both a subcircuit definition and an instance of the component. When not set, the component netlists only as a instance of the component.
ITEM_DRAWSHEET	16384	Component is a drawing sheet or PCB outline. Placed in the fourth field on the create_item() statement.
ITEM_GLOBAL	512	Component has global scope, such as MSUB.
ITEM_GLOBAL_NODE	536870912	Component is a global node.
ITEM_GROUND	1	Component is a ground.
ITEM_HAS_UNKNOWN_MODEL	65536	Component does not represent a model built in to the simulator.
ITEM_NO_LAYOUT_ANNOT	131072	Component does not generate annotation.
ITEM_NOT_ALL_PARM	1073741824	Component does not have an "all parameters" selection in the standard dialog box.
ITEM_NOT_NETLISTED	268435456	Component is not netlisted.
ITEM_NOT_PLACABLE	134217728	Component is not placable.
ITEM_PORT	32	Component is a port.
ITEM_SCOPE_GLOBAL	16777216	Component has global scope.
ITEM_SCOPE_LOCAL	4194304	Component has local scope.
ITEM_SCOPE_NESTED	8388608	Component has nested scope.
ITEM_TEE	4096	Component is a transmission line connection tee.
ITEM_UNIQUE	8	Only one instance of this component can be placed in a design.
ITEM_VARIABLE	2	Component is an equation/variable definition.

create_item() Extra Attribute Choices

Attribute	Numeric Equivalent	Description
ITEM_ANALYSIS_EX	16384	The component is an analysis item. Placed in the fourteenth field of the create_item() statement.
ITEM_CKT_MODEL_EX	64	Identifies the model as Analog/RF type.
ITEM_CROSS_EX	8	
ITEM_CURVE_EX	1	
ITEM_FILE_EX	4096	Identifies file component for the Edit Component dialog.
ITEM_FREQUENCY_EX	1024	Identifies frequency component for the sparam and freqplan dialog.
ITEM_GOAL_EX	262144	The component is a goal item.
ITEM_INITIAL_ORIENTATION_DOWN_EX	524288	Place -90.
ITEM_INITIAL_ORIENTATION_LEFT_EX	2097152	Place 180.
ITEM_INITIAL_ORIENTATION_RIGHT_EX	4194304	Place 0.
ITEM_INITIAL_ORIENTATION_UP_EX	1048576	Place 90.
ITEM_MEASUREMENT_EX	32768	The component is a measurement item.
ITEM_MSTRIP_EX	2	
ITEM_NO_SPL_CHAR_EX	268435456	Only allow alpha-numeric or underscore characters in the component instance name.
ITEM_NOT_AUTO_REPEAT_PLACEMENT_EX	8388608	Place one component at a time.
ITEM_PCB_EX	16	
ITEM_PRIMITIVE_EX	32	Identifies the associated item as a compiled component contained in the simulator.
ITEM_SHORTABLE_EX	2147483648	Component with more than two pins may be shorted. The short will be created between Pins 1 and 2, all other pins will be ignored.
ITEM_SPEC_EX	131072	The component is a yield specification item.
ITEM_STRIP_LINE_EX	4	

Note
If none of the ITEM_INITIAL_ORIENTATION_<B> options are set, use the previously-selected orientation.

See also

See also *Connectivity Objects* (ael) and *DesignContext in AEL* (ael).

Where Used: (ael)

Schematic

create_parm()

Creates a parameter definition for a component. Returns a component parameter. The return value is used by *create_item()*.

Syntax

```
create_parm\(name, label, attrib, formSet, unitCode,
defaultValue\[, cbList\]\);
```

Where,

- *name* is a string; name of the parameter.
- *label* is a string; descriptive label for the parameter.
- *attrib* is an integer; an attribute code. Choices for parameters are:

Attribute Choices

Attribute	Numeric Equivalent	Description
PARM_COMPLEX	524288	For complex parameter numeric value, the prompt in the standard component dialog box displays *Complex, 1.2+j2.5*.
PARM_COMPLEX_ARRAY	134217728	For complex array parameter numeric value, the prompt in the standard component dialog box displays Array, (1.5,3.6)(2.4,4.7).
PARM_DISCRETE_VALUE	32	For discrete parameter numeric value. Refer to *Designing a Discrete Valued Parts Parametric Subnetwork* (ael).
PARM_DOE	4096	Allow doe entry page.
PARM_FIX	4194304	For fixed point parameter numeric value, the prompt in the standard component dialog box displays *Fixed Point*.
PARM_FIX_ARRAY	33554432	For fixed point array parameter numeric value, the prompt in the standard component dialog box displays Array, 1.25 3.5 ....
PARM_INT	131072	For integer parameter numeric value, the prompt in the standard component dialog box displays *Integer*.
PARM_INT_ARRAY	1677216	For integer array parameter numeric value, the prompt in the standard component dialog box displays Array, 1 2 3 ....
PARM_LAYOUT	256	Disable layout-related parameters if layout is not licensed.
PARM_NO_DISPLAY	512	Do not show parameter on schematic by default.
PARM_NOT_EDITED	1	The parameter value is not editable.
PARM_NOT_EVALUATED	536870912	To prevent any expression evaluation of the parameter value.
PARM_NOT_NETLISTED	64	The parameter should not be netlisted.
PARM_NOT_ON_SCREEN_EDITABLE	32768	The parameter value is not on-screen editable.
PARM_OPTIMIZABLE	1024	Allow optimization entry page.
PARM_PRECISION	8388608	For precision parameter numeric value, the prompt in the standard component dialog box displays Precision, 8.24.
PARM_REAL	65536	For real parameter numeric value, the prompt in the standard component dialog box displays *Real*.
PARM_REAL_ARRAY	67108864	For real array parameter numeric value, the prompt in the standard component dialog box displays Array, 1.5 3.6 ....
PARM_REPEATED	2	This is a repeated parameter; s[1]=100;

		s[2]=200;
PARM_RIGHT_HAND_ONLY	128	Do not generate left hand side when netlisting. Can be tested with netlist format %30. Also, when this attribute is set, the onscreen edit can partition the parameter displayed in the annotation into separate fields according to the annotation fmtstring, such as: start=%1s step=%2s.
PARM_STATISTICAL	2048	Allow statistical entry page.
PARM_STRING	262144	For string parameter numeric value, the prompt in the standard component dialog box displays *String*.
PARM_STRING_ARRAY	26843456	For string array parameter numeric value, the prompt in the standard component dialog box displays Array, (str1)(str2).

- *formSet* is the name of the form set used for this value. This is the same as name in the create_form_set() function.
- *unitCode* is the choice for units, where:
  - STRING_UNIT = -2
  - UNITLESS_UNIT = -1
  - FREQUENCY_UNIT = 0
  - RESISTANCE_UNIT = 1
  - CONDUCTANCE_UNIT = 2
  - INDUCTANCE_UNIT = 3
  - CAPACITANCE_UNIT = 4
  - LENGTH_UNIT = 5
  - TIME_UNIT = 6
  - ANGLE_UNIT = 7
  - POWER_UNIT = 8
  - VOLTAGE_UNIT = 9
  - CURRENT_UNIT = 10
  - DISTANCE_UNIT = 11
  - TEMPERATURE_UNIT = 12
  - DB_GAIN_UNIT = 13
- *defaultValue* is optional; a value returned from the *prm()* function. The *prm()* function generates an acceptable default value for parameters with different form sets.
- *cbList* is optional; the list of callbacks. For example,
- *list(dm_create_cb ("...", "..."), ...);*. Currently-supported callbacks are:
  - PARM_DEFAULT_VALUE_CB
  - PARM_MODIFIED_CB

Example

This example creates a parameter C representing capacitance, with attributes, using the StdFileFormSet form set, capacitance as the units and 1.0 pF as the default value. This might then be used by *create_item()* to create a capacitance with C as its parameter.

```
create_parm("C", "Capacitance", PARM_OPTIMIZABLE | PARM_STATISTICAL,
"StdFileFormSet", CAPACITANCE_UNIT,
prm( "StdForm", "1.0 pF"));
```

Download Example File

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Generate SnP Component

See also

Form Sets (ael)

Where Used (ael)

Schematic

create_text_form()

Creates a form whose set of possible values are a set of strings. These can be either a simple list of strings or a list of strings dynamically determined when an item referencing the form is placed. Returns: none.

See also: *Format Strings* (ael)

Syntax:

```
create_text_form(name, label[, dialogDataStr], attrib, netlistFormat,
displayFormat[, optionFunction, validateFunction, dataValue, dataFunction]);
```

where

name is the unique name of the form.

label is a string describing this form.

dialogDataStr is optional; any dialog data. For the standard edit component dialog box, this argument indicates the gui component to use for the breakdown of the parameter value input. For example, the standard Edit Component dialog supports these gui configurations:

- "StdForm"
- "StringAndReference"
- "SingleTextLine"
- "InstSelectionForm"
- "Node SetForm"

This argument can be omitted if dialogDataStr is the same as formName. For the standard component dialog, this field may be used as the name of the table entry field. For a complete listing, see *$HPEESOF_DIR/de/ael/pde_gemini.ael*.

attrib is an integer, and should be set to 0.

netlistFormat is a string describing the netlisting of this form's value. Refer to *Format Strings* (ael).

displayFormat is a string describing the display in the schematic of the values of this form.

optionFunction is optional; a function used to dynamically collect strings.

validateFunction is optional; a function to validate the strings collected by the option function.

dataValue is optional; a string; passed to the optionFunction, validateFunction, and dataFunction.

dataFunction is optional; a function to provide special control over netlist generation.

Example:

```
create_text_form("SingleTextLineInteger", "IntegerValue", "SingleTextLine", 0,
"%v","%v", NULL, stdforms_validate_integer, NULL);
```

Where Used: **(ael)**

Schematic

de_define_library_palette()

This function defines a palette group of ADS Components for a particular library that contains components from the same library or different libraries. The palette group is displayed in the palette side bar of the ADS Schematic or Layout windows. Bitmaps for the components are displayed in the active palette (if defined for the component, otherwise the component name appears). The palette group will be loaded into the Schematic or Layout window palette bars when the library owner is opened. The palette group will be unloaded from the Schematic and Layout window palette bars when the library owner is closed.

Returns

None

Syntax

```
de_define_library_palette( pal_libName, win_type, dsn_type, pal_name,
pal_label, insert_position [, palette_item1st = list(libName, cellName,
viewName, compLabel, compBitmap) [, ... ] [, palette_itemnth = list(...) ] ] );
```

Where,

- *pal_libName* is a string representing the name of the library owner of the palette. This can be empty is the palette is not owned by a library (e.g. a user palette)
- *winType* is the type of window: LAYOUT_WINDOW or SCHEMATIC_WINDOW

- *dsn_type* is the name of the design type for the palette group to be enabled within: "analogRF_net" or "sigproc_net"
- *pal_name* is a unique string name of the palette group, it is the name that will be displayed in the palette group selection list.
- *pal_label* is a descriptive label string for the palette group.
- *insert_position* is an integer value: -1 to alphabetically sort the list of palette groups after adding the new palette, -2 to append the new palette at the end, any other integer (0 or above) to insert the new palette in the specified position in the current list of palette groups.
- *palette_item1st* - *palette_itemnth* are optional palette item list arguments.
  Each optional palette item list argument contains a list with the following parameters and format:

  ```
  list(libName, cellName, viewName, compLabel, compBitmap)
  ```

  Where list parameters are,

  - *libName* is a string representing the name component's library.
    If the *libName* string is left empty "" or NULL, then the palette group's library name *pal_libName* will be used instead so it must be valid.
  - *cellName* is a string cell name of an ADS component.
  - *viewName* is a string view name for the ADS component to use during placement.
    If the *viewName* is not specified ("" or NULL), then an attempt will be made to find an appropriate default view for the component based on the palette group's window type *winType* setting. If no appropriate default view could be found then the palette item will generate an error for the component, since the view for the component was not found.
  - *compLabel* is a string component descriptive label.
  - *compBitmap* is a string bitmap name.

Example

```
de_define_library_palette("mylibrary", SCHEMATIC_WINDOW, "analogRF_net",
   "My Resistor Palette","My Favorite Resistor Components", 0,
   list("ads_rflib","R","symbol","ADS Resistor","R"),
   list("mylibrary","R","symbol","My Custom Resistor","My_R_Bitmap"),
   list("foolibrary","R","symbol","Foo Resistor","Foo_R"),
   list("","R_Model","","My Custom Resistor Model","My_R_Model_Bitmap"),
);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and later

See also

Where Used (ael)

Schematic, Layout

de_update_design_definition_ex()

Updates the item definitions for all the instances within the given design context.

Returns

None

Syntax

```
de_update_design_definition_ex(context [, updateHierarchy ]);
```

Where,

- *context* is the design context to update its instances item definitions.
- *updateHierarchy* is an optional argument that specifies to update the item definitions for all instances within the design hierarchy.
  This argument is FALSE by default if the argument is not specified, meaning it will only update the item definitions for the instances in the top level design. If the argument is specified as TRUE it will update all the instances' item definitions in the design's hierarchy.

Example

```
decl context = de_get_design_context(winInst);
de_update_design_definition_ex(context, TRUE);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Component Definition Functions (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

de_define_palette_group()

Defines a palette group. The palette group is displayed in the palette dialog. Bitmaps for the components are displayed in the active palette (if defined for the component, otherwise the component name appears). Returns: none.

Note
This function should be called at bootup for the palette to be displayed in the palette dialog.

Syntax:

```
de_define_palette_group(winType, dsnType, groupName, groupLabel, position,
item1name, item1label, item1bitmap name[, item2name, item2 label, item2 bitmap
name, ...] );
```

where

winType is the type of window, where:

- SCHEM_WIN = Schematic window
- LAYOUT_WIN = Layout window

dsnType is the name for each design type, such as *analogRF_net* or *sigproc_net*.

groupName is the name of the palette group.

groupLabel is the descriptive label for the group.

position is an integer: -1 to insert into an alphabetized list (only for DSP palettes, otherwise the result is random), -2 to append the new palette at the end, any other integer (0 or greater) to insert the new palette in the specified position in the list of palettes.

item1name is the first component name.

item1label is the first component label.

item1bitmap name is the first component bitmap file name.

item2name is optional; the second component name.

item2label is optional; the second component label.

item2bitmap name is optional; the second component bitmap file name.

Example:

```
de_define_palette_group(SCHEM_WIN, "analogRF_net", "mypal", "My palette", -2,
"R", "Resistor", "R", "C", "Capacitor", "C");
```

***Where Used:* (ael)**

Schematic

Note
The dk_define_palette_group()function was added in ADS2003A to keep track of the specific palettes that are installed by a design kit. If a design kit is disabled, the palettes for that design kit are disabled accordingly. The de_define_palette_group() function was used in previous versions of ADS; however, this function does not handle palette management. Therefore, the dk_define_palette_group() function is the recommended component palette group function for ADS design kits.

If you are not sure if your design kit will be used with an older version of ADS, the example code below shows one method of using dk_define_palette_group() while ensuring that your design kit will still be compatible with older versions of ADS.

library_group()

Defines a new library group composed of the listed components. Adds the group to the current component dictionary. Since groups are associated with the window by the type of design displayed in the window, the desired type of design must be indicated by a call to *set_design_type()* prior to calling this function. The library label appears in the library dialog, along with all the components in the group. Returns: none.

Syntax

```
library_group(name, label[,bToFront], item1, item2, ..., itemN);
```
Where,

- *name* is the name of the library group. Use an asterisk to make the item appear in the current workspace.
- *label* is a descriptive label for the library. Use an asterisk to make the item appear in the current workspace.
- *bToFront* is optional boolean flag. If TRUE (=1), the group will be added to the front of the library group list. If FALSE (=0 and the default), the group will be added to the end of the library list in the library browser.
- *itemN* is a defined component name; one or more can be given.

Example

```
library_group("ckt ac sim group", "AC Simulation", "AC", "Options", SweepPlan",
"ParamSweep", "NodeSet", "NodeSetByName");
```

Where Used (ael)

Schematic

prm()

Creates and returns a default parameter value. This is most frequently used in conjunction with *create_parm()* to assign a default value to an component parameter. Returns: A valid default parameter value for the given form.

Syntax:

```
prm(formName, paramValue1, paramValue2,..., paramValueN);
```

where

formName is a string; name of a form defined with *create_form()*. The form name must be one of the forms in the parameter's formset, specified in *create_parm()*.

ParamvalueN is one or more default values for the parameter, depending on the construction of the form:

- For text forms, the value is a single string, in quotes (see *create_text_form()* (ael)).
- For constant forms, no value is required (see *create_constant_form()* (ael)).
- For compound forms, the values of each field are described hierarchically, as with the default value of the *create_parm* ; that is, each field value requires a *prm()* function as is appropriate to the form set and desired form in the form set for the field (see *create_compound_form()* (ael)).

Example:

```
prm("stdForm", "500");
```

Typically used as in:

```
create_parm("depVar","DependentVariable", PARM_NO_DISPLAY,
"StdFileFormSet", UNITLESS_UNIT, prm("StdForm", ""));
create_parm("Variable Value", "Variable equation",
PARM_OPTIMIZABLE|PARM_RIGHT_HAND_ONLY|PARM_REPEATED|
PSTM_STATISTICAL, "VarEqnForms", UNITLESS_UNIT,
list(prm("VarFormEditCompPowerVar", prm("VarNameForm","X"),
prm("editcompPowerVar", "1.0"))));
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Generate SnP Component
PI Attenuator

***Where Used:* (ael)**

Schematic

set_design_choices()

Defines choices for characteristics allowed for parametric subnetwork components.
Returns: none.

Syntax:

```
set_design_choices(choice type, label, value ...);
```

where

choice type is the choice for parametric subnetwork components characteristics, where:

- 0 = artwork, such as "AEL macro"
- 1 = component attribute, such as "Layout Object"
- 2 = parameter attribute, such as "Not Edited"
- 3 = parameter form set, such as "SmtPADDataOnly (layout)"
- 4 = netlist model, such as "Subnetwork"
- 5 = symbol, such as "SYN_OPort"
- 6 = model parameter attribute, such as "Optimizable"
- 7 = parameter value type attribute, such as "Real"

label value pairs is one or more label strings paired with values, where the values are integers for types 0, 1, 2, 6, and 7 or strings for types 3, 4, or 5.

Example:

```
set_design_choices(3, "Optimizable", "StdFormSet","Integer", "StdFormSet",
"Real", "StdFormSet", "SmtPAD Data Only (layout),
push in `not netlisted' button", "StdFormSet");
```

Where Used: **(ael)**

Schematic

set_design_sub_choices()

Defines sub-choices for each characteristic choice for parametric subnetwork components.
Returns: none.

Syntax:

```
set_design_sub_choices(choice type, choice index, label, value ...);
```

where

choice type is a choice type, where:

- 0 = artwork
- 1 = component attribute
- 2 = parametric attribute
- 3 = parameter form set
- 4 = netlist model
- 5 = symbol

- 6 = model parameter attribute
- 7 = parameter value attribute

choice index is an Index indicating which label/value pair from *set_design_choices()* to attach sub-choices, where 0 is for the first pair. *label value* pairs is one or more label strings paired with values, where the values are integers or strings.

Example:

```
set_design_sub_choices(0, 2, "conn", 0, "cpad2", 1, "cpad3", 2);
//to set macro artwork choices. In this case the value in the label
//value pairs is unimportant.
```

Where Used: **(ael)**

Schematic

set_design_type()

Sets the current type of design for group definitions. Library groups are associated with design windows according to current type of design. The design type must be set using this function before the function *library_group()* will work. Returns: none.

See also: *set_simulator_type()* (ael).

Syntax:

```
set_design_type(type);
```

where

type is one of two valid string arguments: *analogRFnet* or *sigprocNet*. You may also see the integers *1* (for analogRFnet) and *11* (for sigprocNet) used, but you should use the strings for clarity. The type determines whether a library will be visible in the library browser when started from an analogRF schematic versus a signal processing schematic. Note that these string arguments are not the same as the analogous ones used in the *de_define_palette_group()* function.

Example:

```
set_design_type(analogRFnet);
```

Where Used: **(ael)**

Schematic

set_netlist_info()

Sets special netlisting characteristics for current type of design. Returns: none.

See also: *Format Strings* (ael).

Syntax:

```
set_netlist_info(prefix format, suffix format, priority, in format, out format,
input format);
```

where

prefix format is a special format string for beginning of a subnetwork netlist.

suffix format is a special format string for end of a subnetwork netlist.

priority is an integer priority code used in ordering design netlist in netlist output; usually set to 1.

in format is a string, not used.

out format is a string, not used.

input format is a string, not used.

Example:

```
/* set the format string for subnetwork netlist generation */
set_netlist_info("define %n (%@ %32?%M%:_net%m%; %e) n%37?parameters%:;
parameters%; %b %k=%p %e", "end %n", 1, "", "", "");
```

***Where Used:* (ael)**

Schematic

set_simulator_type()

Sets the current simulator for ADS 2003A and earlier. Used to associate commands and creation functions with a simulator type. It must be set before any *create_item()* or *create_*_form()* function is called to correctly associate a simulator type with a component, form, or formset definition. Returns: none.

Note
In the *create_*_form()* function, where * can be one of these: *create_compound_form()*, *create_text_form()*, or *create_constant_form()*.

Syntax:

```
set_simulator_type(simulator_type);
```

where

simulator_type is the choice for simulator type, where:

- 1 = Analog/RF
- 11 = ADS Ptolemy
- −1 = ADS Ptolemy or Analog/RF

Example:

```
set_simulator_type(1);
```

Where Used: **(ael)**

Schematic

Component Definition Query Functions

This section describes each Item Definition Query function in detail. The functions are listed in alphabetical order.

dm_find_form_definition() (ael)	*dm_get_item_definition_attribute()* (ael)
dm_find_item_definition() (ael)	*dm_get_parm_definition_attribute()* (ael)
dm_first_parm_definition() (ael)	*dm_get_simcode_from_designcode()* (ael)
dm_get_design_class_code() (ael)	*dm_index_parm_definition()* (ael)
dm_get_design_name() (ael)	*dm_next_parm_definition()* (ael)
dm_get_form_definition_attribute() (ael)	*dm_num_parm_definitions()* (ael)

dm_find_form_definition()

Returns a handle to a form definition given the form name and, optionally, a simulator ID. Then this handle can be used by the function *dm_get_form_definition_attribute()* to retrieve other attributes of the form.

Syntax:

```
dm_find_form_definition(name [, simulator]);
```

where

name is the name of the form.

simulator is the optional simulator ID code for the dictionary where the form is stored. The current simulator set by *set_simulator_type()* is assumed if this parameter is omitted.

Example:

```
decl formH, netFmt;
formH = dm_find_form_definition("stdForm", 1);
netFmt = dm_get_form_definition_attribute(formH, DM_FORM_NET_FORMAT);
```

***Where Used:* (ael)**

Schematic

dm_find_item_definition()

Returns an handle to an item definition given the item's name. The current design type simulator dictionary is used to find the item definition. Current design type can be set by *set_design_type()*.

Syntax:

```
dm_find_item_definition(name);
```

where

name is the name of the item.

Example:

```
decl item;
item = dm_find_item_definition("R");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Flatten Design
Netlist with Node Names

Where Used: **(ael)**

Schematic

dm_first_parm_definition()

Returns the handle to the first parameter definition of an item, given a parameter definition handle as returned from *dm_get_item_definition_attribute()*.

Syntax:

dm_first_parm_definition(parmHandle);

where

parmHandle is the handle to a parameter definition.

Example:

```
decl parmH, parmList, itemH;
itemH = dm_find_item_definition("R");
parmList = dm_get_item_definition_attribute(itemH, ITEM_PARMS);
parmH = dm_first_parm_definition(parmList);
```

Where Used: **(ael)**

Schematic

dm_get_design_class_code()

Returns a code representing the design class; where: 0 = network design.

Syntax:

```
dm_get_design_class_code(designHandleOrDesignCode);
```

where

designHandleOrDesignCode is a handle to a design, as returned from *db_get_design()* or a design type code as required by *set_design_type()*, such as *analogRF_net* or *sigproc_net*.

Examples:

```
decl code, designHandle;
code = dm_get_design_class_code(analogRF_net);
designHandle = db_get_design("myDesign");
code = dm_get_design_class_code(designHandle);
```

Where Used: **(ael)**

Schematic

dm_get_design_name()

Returns a string, the generic group name defined in AEL for a particular type of design.

Syntax:

```
dm_get_design_name(designHandleOrDesignCode);
```

where

designHandleOrDesignCode is a handle to a design, as returned from *db_get_design()* or a design type code as required by *set_design_type()*.

Example:

```
decl type;
type = dm_get_design_name(analogRF_net); // returns string analogRF_net
type = dm_get_design_name(sigproc_net); // returns string sigproc_net
```

Where Used: **(ael)**

Schematic

dm_get_form_definition_attribute()

Returns the value of a form definition attribute, given a handle to the form definition. These attributes are defined when the form is created with *create_compound_form()*, *create_text_form()*, or *create_constant_form()*.

Syntax:

```
dm_get_form_definition_attribute(form, attribute [,index]);
```

where

form is a form handle, as returned from *create*form()*.

Note: In the *create_*_form()* function, where * can be one of these: *create_compound_form()*, *create_text_form()*, or *create_constant_form()*.

attribute is the data for the form selected from this list:

- DM_FORM_NAME is the unique name of the form.
- DM_FORM_LABEL is the descriptive label for the form.
- DM_FORM_NET_FORMAT is the format string to netlist the parameter value.
- DM_FORM_DISP_FORMAT is the format string to display the parameter value in a schematic.
- DM_FORM_FIELDS are the labels for the form fields.
- DM_FORM_ATTR are the attribute codes for the form. See *create*form()* for more information.
- DM_FORM_NUMFIELDS is the number of fields in the form.
- DM_FORM_CLASS is the general class of the form.
- DM_FORM_PARMS is the list of parameters defining a form. Used in compound forms.

index is optional; the index specifying which form, in a complex form.

Example:

```
decl Label, formH;
formH = dm_find_form_definition("StdForm");
Label = dm_get_form_definition_attribute(formH, DM_FORM_LABEL);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.

Flatten Design
Netlist with Node Names

***Where Used:* (ael)**

Schematic

dm_get_item_definition_attribute()

Returns the value of an item definition attribute given a handle to the item.

Syntax:

dm_get_item_definition_attribute(item, attribute);

where

item is a handle to item definition, as returned by *dm_find_item_definition()*.

attribute is the type of data to retrieve from the item definition. These attributes are described in the item's AEL definition created with the *create_item()* function. The meaning of these fields is documented in existing AEL documentation under *create_item()*.

- ITEM_NAME is a unique name of the item.
- ITEM_LABEL is a descriptive label for the item.
- ITEM_ATTR is an integer, attribute code. See *create_item()* (ael) for more information.
- ITEM_PRIORITY is the item netlisting priority.
- ITEM_PREFIX is the instanceName prefix, such as the C in C12.
- ITEM_DIALOG_NAME is the string name of the dialog box used to edit item.
- ITEM_DIALOG_DATA is the descriptive string displayed in dialog.
- ITEM_NET_FORMAT is a netlist formatting string.
- ITEM_NET_DATA is the data used by the netlist format.
- ITEM_SCHEM_FORMAT Like a netlist formatting string describes the format of parameter display on schematic.
- ITEM_SCHEM_NAME is the name of a design containing the schematic symbol.
- ITEM_ART_TYPE is the type of artwork, where: 0 = no artwork; 1 = fixed artwork; 2 = AEL macro artwork.
- ITEM_ART_DATA is the name of a layout artwork design for fixed artwork and is an ael artwork generation function name for AEL macro artwork.
- ITEM_ICON is the name of the icon bitmap file for the palette. If NULL, use ITEM_NAME.
- ITEM_PARMS is the parameters created with *create_parm()*.
- ITEM_SIMTYPE is the simulator code (simcode). Simcode is a unique code for each simulator, such as circuit or system.
- ITEM_ATTR_EX is an integer; extra attribute code. See *create_item()* (ael) for more information.

Example:

```
decl attr, itemH;
itemH = dm_find_item_definition("MLIN");
attr= dm_get_item_definition_attribute(itemH, ITEM_ATTR);
```

Where Used: **(ael)**

Schematic

dm_get_parm_definition_attribute()

Returns a value of a parameter definition attribute, as defined with *create_parm()*. The meaning of these fields are documented under *create_parm()*.

Syntax:

```
dm_get_parm_definition_attribute(parmHandle, attribute);
```

where

parmHandle is the handle to a parameter definition.

attributes is the type of data to retrieve from the parameter definition, where:

- DM_PARM_NAME is the name of the parameter.
- DM_PARM_LABEL is a descriptive label for the parameter.
- DM_PARM_ATTR is an integer, attribute code. See *create_parm()* (ael) for more information.
- DM_PARM_UNIT is the unit for the parameter.
- DM_PARM_FORMSET is the name of the formset used for this parameter value.
- DM_PARM_DEFVALUE is the handle to a parameter which contains the default value. The default value of the parameter is determined by the *prm()* function. See *prm()* (ael) for more information.

Example:

```
decl itemH, parmList, parmH, parmName, parmDefPtr, parmDefVal;
itemH = dm_find_item_definition("R");
parmList = dm_get_item_definition_attribute(itemH, ITEM_PARMS);
parmH = dm_first_parm_definition(parmList);
parmName = dm_get_parm_definition_attribute(parmH, DM_PARM_NAME);
//An example of how to get the default string value of the parameter
parmDefPtr = dm_get_parm_definition_attribute(parmH, DM_PARM_DEFVALUE);
parmDefVal = db_get_parm_attribute(parmDefPt, PARM_VALUE_SVALUE);
```

Where Used: **(ael)**

Schematic

dm_get_simcode_from_designcode()

Returns a simcode, which is required by functions such as *dm_find_form_definition()*, given a design code, which is returned from functions such as *db_get_design_code()*.

Syntax:

```
dm_get_simcode_from_designcode( designCode);
```

where

designCode is the unique code for each type of design, such as *analogRF_net* or *sigproc_net*.

Example:

This example sets simCode to analogRF_net.

```
decl simCode;
simCode = dm_get_simcode_from_designcode(analogRF_net);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Netlist with Node Names

***Where Used:* (ael)**

Schematic

dm_index_parm_definition()

Returns a handle to the indexed parameter definition.

See also: *dm_first_parm_definition()* (ael), *dm_next_parm_definition()* (ael).

Syntax:

```
dm_index_parm_definition(parmHandle, indx);
```

where

parmHandle is the handle to a parameter definition.

indx is the indexed parameter to retrieve.

Examples:

```
decl itemH, parmList, parmH;
itemH = dm_find_item_definition("R");
parmList = dm_get_item_definition_attribute(itemH, ITEM_PARMS);
// retrieve the 2nd parameter of the item
parmH = dm_index_parm_definition(parmList, 2);
```

```
// retrieve the 3rd parameter of the item
parmH = dm_first_parm_definition(itemH);
parmH = dm_index_parm_definition(parmH, 3);
```

Where Used: **(ael)**

Schematic

dm_next_parm_definition()

Returns the next handle to the parameter definition of an item, given a parameter handle as returned from *dm_first_parm_definition()*.

Syntax:

```
dm_next_parm_definition(parmHandle);
```

where

parmHandle is the handle to a parameter definition.

Example:

```
decl itemH, parmH;
itemH = dm_find_item_definition("R");
parmH = dm_first_parm_definition(itemH);
parmH = dm_next_parm_definition(parmH);
```

Where Used: **(ael)**

Schematic

dm_num_parm_definitions()

Returns the number of parameter definitions in a parameter list, given a parameter definition handle.

Syntax:

```
dm_num_parm_definitions(parmHandle);
```

where

parmHandle is the handle to a parameter definition.

Example:

```
decl num, parmH, itemH;
```

```
itemH = dm_find_item_definition("MLIN");
parmH = dm_first_parm_definition(itemH);
num = dm_num_parm_definitions(parmH);
```

Where Used: **(ael)**

Schematic

Construction Line Functions

This section describes the following Construction Line Function in detail:

db_get_const_line_layerid() (ael)

db_get_const_line_layerid()

Returns the *LayerId* (ael) for the given construction line object.

Syntax

```
layerId = db_get_const_line_layerid(dbObject);
```

Where,

- *dbObject* is a construction line.

Example

```
decl layerId = db_get_const_line_layerid(dbConstLine);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Construction Line Functions (ael)
LayerId Overview (ael)

Where Used (ael)

Schematic, Layout

Database Query and Manipulation Functions

This section describes each Database Query and Manipulation functions in detail. The functions are listed in alphabetical order.

db_find_instance_ex()

Finds and returns an instance with a given instance name in the given *DesignContext* (ael). If no instance is found with that instance name in the given *DesignContext* (ael) then NULL is returned.

See also: *db_get_instance_component_name()* (ael), *db_get_instance_item_definition()* (ael), *db_get_instance_name()* (ael).

Syntax:

```
db_find_instance_ex(context, instName);
```

where

context is a DesignContext returned from a function such as *de_get_current_design_context()*.
instName is a string instance name.

Example:

```
decl context = de_get_current_design_context();
// Find an instance in the current context with instance name X1.
decl inst = db_find_instance_ex(context, "X1");
if (!inst)
   return; // No instance found with that instance name in the context.
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_get_component_name()

Returns a string component name of a given *DesignContext* (ael). The value returned from this function is the name of the component (mycomponent or C:/../my_wrk/networks/mycomponent or mylib:mycomponent). Note: The returned component name may be a full path.

See also: *db_get_design_name()* (ael), *db_get_design_type()* (ael).

Syntax:

```
decl compName = db_get_component_name(context);
```

where

context is a *DesignContext* (ael) returned from a function such as *de_get_current_design_context()* (ael).

Example:

```
decl context = de_get_current_design_context();
// Get the component name of the given DesignContext.
decl compName = db_get_component_name(context);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_get_design_name()

Returns a string design name of a given *DesignContext* (ael). The value returned from this function is the name of the design (mydesign or C:/../my_wrk/networks/mydesign) or could be referred to as the name of the component and view (mylib:mydesign:schematic). Note: The returned design name may be a full path.

See also: *db_get_component_name()* (ael), *db_get_design_type()* (ael).

Syntax:

```
decl designName = db_get_design_name(context);
```

where

context is a *DesignContext* (ael) returned from a function such as *de_get_current_design_context()* (ael).

Example:

```
decl context = de_get_current_design_context();
// Get the design name of the given DesignContext.
decl dsnName = db_get_design_name(context);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_get_design_type()

Returns an integer design type of a given *DesignContext* (ael). The value returned from this function is a design type integer code; i.e. analogRFnet or sigproc_net.

See also: *db_get_component_name()* (ael), *db_get_design_name()* (ael).

Syntax:

```
decl designType = db_get_design_type(context);
```

where

context is a *DesignContext* (ael) returned from a function such as *de_get_current_design_context()* (ael).

Example:

```
decl context = de_get_current_design_context();
// Get the design type of the given DesignContext.
decl dsnType = db_get_design_type(context);
if (dsnType == analogRFnet)
{
   // analog: analogRF design context.
}
else if(dsnType == sigproc_net)
{
  // digital: sigproc_net design context.
}
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_get_instance_component_name()

Returns a string component name of a given instance. The component name refers to the instance's type of component. The value returned from this function can be used to see if two instances are instances of the same component. This function can also be used to see if a component is of a particular predetermined type, like "R" or "eesoflib:R". Note: The return component name may be a full path.

Syntax:

```
db_get_instance_component_name(instance);
```

where

instance is a valid instance iterator, instance object, or instance handle.

Example:

```
decl context = de_get_current_design_context();
// Find an instance in the current context with instance name X1.
decl inst = db_find_instance_ex(context, "X1");
if (!inst)
   return; // No instance found with that instance name in the context.

// If X1 instance found, get its component name.
decl compName = db_get_instance_component_name(inst);
```

Version Introduced

ADS 2009 Update 1

See also

db_find_instance_ex() (ael)
db_get_instance_cell_name() (ael)
db_get_instance_design_name() (ael)
db_get_instance_library_name() (ael)
db_get_instance_name() (ael)

***Where Used:* (ael)**

Schematic, Layout

db_get_instance_item_definition()

Returns an item definition for a given instance.

See also: *db_find_instance_ex()* (ael).

Syntax:

```
db_get_instance_item_definition(instance);
```

where

instance is a valid instance iterator, instance object, or instance handle.

Example:

```
decl context = de_get_current_design_context();
decl instIter = db_create_inst_iter(context);
if (!db_inst_iter_is_valid(instIter))
   return;
// Get the instance's item definition.
decl itemDefP = db_get_instance_item_definition(instIter);
if (!itemDefP)
   return;
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_get_instance_name()

Returns a string instance name of a given instance.

Syntax:

```
db_get_instance_name(instance);
```

where

instance is a valid instance iterator, instance object, or instance handle.

Example:

```
decl context = de_get_current_design_context();
// Construct a list of instance names for the design context.
decl instNameList = list();
decl iter = db_create_inst_iter(context);
if (db_inst_iter_is_valid(iter))
{
   // Fill list with instance names of design context.
   for ( ; db_inst_iter_is_valid(iter); iter = db_inst_iter_get_next(iter))
   {
       decl instName = db_get_instance_name(iter);
       instNameList = append(instNameList, instName);
   }
}
```

Version Introduced

ADS 2009 Update 1

See also

db_find_instance_ex() (ael)
db_get_instance_cell_name() (ael)
db_get_instance_component_name() (ael)
db_get_instance_design_name() (ael)
db_get_instance_library_name() (ael)

***Where Used:* (ael)**

Schematic, Layout

db_get_item_definition()

Returns an item definition for a given *DesignContext* (ael).

See also: *db_get_design_name()* (ael), *db_get_component_name()* (ael).

Syntax:

```
db_get_item_definition(context);
```

where

context is a DesignContext returned from a function such as *de_get_current_design_context()*.

Example:

```
decl context = de_get_current_design_context();
// Get the DesignContext's item definition.
decl itemDefP = db_get_item_definition(context);
if (!itemDefP)
   return;
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_get_path_trace_bend_type()

Returns the corner (bend) type for a given path, trace, or wire shape. The return value is one of the following 3 corner type values:

- DB_MITERED_CORNER = mitered
- DB_SQUARE_CORNER = square
- DB_CURVED_CORNER = curved.

Note
Returns an AEL error if the given shape is not a path or trace.

Syntax

```
decl cornerType = db_get_path_trace_bend_type(shape);
```

Where,

- *shape* is the path/trace/wire to return the corner (bend) type for.

Example

```
decl curveRadius = NULL;
decl miterCutoff = NULL;
if (db_get_path_trace_bend_type(shape) == DB_CURVED_CORNER)
   curveRadius = db_get_path_trace_miter_radius(shape);
else if (db_get_path_trace_bend_type(shape) == DB_MITERED_CORNER)
   miterCutoff = db_get_path_trace_miter_radius(shape);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_set_path_corner() (ael)
db_get_path_trace_miter_radius() (ael)
db_set_curve_radius() (ael)
db_set_miter_cutoff() (ael)
db_get_path_trace_width() (ael)
db_set_path_width() (ael)
db_add_path() (ael)
db_end() (ael)

***Where Used* (ael)**

Schematic, Layout

db_get_path_trace_miter_radius()

Returns the miter cutoff percentage or curve radius for a given path, trace, or wire shape. Returns a cutoff percentage (0 – 100) for a mitered corner, or a curve radius in database units for a curved corner.

Note
Returns an AEL error if the given shape is not a path or trace.

Syntax

```
decl miterRadiusPercentage = db_get_path_trace_miter_radius(shape);
```

Where,

- *shape* is the path/trace/wire to return the curve radius or miter cutoff percentage for.

Example

```
decl curveRadius = NULL;
decl miterCutoff = NULL;
if (db_get_path_trace_bend_type(shape) == DB_CURVED_CORNER)
   curveRadius = db_get_path_trace_miter_radius(shape);
else if (db_get_path_trace_bend_type(shape) == DB_MITERED_CORNER)
   miterCutoff = db_get_path_trace_miter_radius(shape);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_path_trace_bend_type() (ael)
db_set_path_corner() (ael)
db_set_curve_radius() (ael)
db_set_miter_cutoff() (ael)
db_get_path_trace_width() (ael)
db_set_path_width() (ael)
db_add_path() (ael)
db_end() (ael)

***Where Used* (ael)**

Schematic, Layout

db_get_path_trace_width()

Returns the width in database units for a given path, trace, or wire shape.

Note
Returns an AEL error if the given shape is not a path or trace.

Syntax

```
decl width = db_get_path_trace_width (shape);
```

Where,

- *shape* is the path/trace/wire to return the width for.

Example

```
decl width= db_get_path_trace_width(shape);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_path_trace_bend_type() (ael)
db_set_path_corner() (ael)
db_get_path_trace_miter_radius() (ael)
db_set_curve_radius() (ael)
db_set_miter_cutoff() (ael)
db_set_path_width() (ael)
db_add_path() (ael)
db_end() (ael)

***Where Used* (ael)**

Schematic, Layout

db_mks_to_uu_factor()

Returns a real value conversion factor for converting circuit MKS units to database user units for a given DesignContext.

Note
This function is an alias to the preferred function *db_get_mks_to_uu_factor()* (ael).

Syntax:

```
db_mks_to_uu_factor(context);
```

where

context is a DesignContext returned from a function such as *de_get_current_design_context()*.

Example:

```
if (!de_current_context_is_valid())
   return;
decl context = de_get_current_design_context();
// Get the conversion factor from circuit units to database user units for this current context.
decl conversionFactor = db_mks_to_uu_factor(context);
decl userUnit = 20*conversionFactor;
```

Version Introduced

ADS 2009 Update 1

See also

db_get_context_unit_name() (ael)
db_get_dbu_to_uu_factor() (ael)
db_get_mks_to_uu_factor() (ael)
db_get_user_units_significant_digits() (ael)
db_get_uu_to_dbu_factor() (ael)
db_get_uu_to_mks_factor() (ael)

Where Used: **(ael)**

Schematic, Layout

db_transform_bbox_ex()

Transforms the given bounding box to be the smallest possible bounding box which completely contains the transformed input bounding box. Returns the transformed input bounding box.

Note
This is an important distinction for transforms with arbitrary rotation, because bounding boxes are always aligned with the axes.

Syntax:

```
decl newBbox = db_transform_bbox_ex(bbox, transform);
```

Where,

- *bbox* is a bounding box to transform.
- *transform* is the transformation to apply to the input bounding box.

Example:

```
decl newBbox = db_transform_bbox_ex\(bbox, transform\);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_map() (ael)
db_set_map() (ael)

***Where Used:* (ael)**

Schematic, Layout

db_clear_map()

Clears the transformation mapping established by *db_setup_transform()*, *db_set_map()*, or *db_setup_map()*. Returns: none.

Syntax:

```
db_clear_map();
```

Example:

```
db_clear_map();
```

***Where Used:* (ael)**

Schematic, Layout

db_first_parm()

Returns a handle to a parameter or instance identified by an instance handle or the first parameter of a list of parameters identified by a parameter handle.

Syntax:

```
db_first_parm(instOrParmHandle);
```

where

instOrParmHandle is a handle to a parameter or instance.

Example:

```
decl instH, parmH;
parmH = db_first_parm(instH);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.

Export a Modelcard to a MDIF File
Get MeasEqn String 2
Get Variable String 2

Where Used: **(ael)**

Schematic, Layout

db_get_bbox_x1()

Returns an integer, the x component of the lower-left corner of an object's bounding box. The bounding box is the maximum extent of any object, and is described as a rectangle.

Syntax:

```
db_get_bbox_x1(bboxHandle);
```

where

bboxHandle is the handle of an object's bounding box.

Example:

```
decl repBBox, x, repHandle;
repBBox = db_get_rep_bbox (repHandle);
x = db_get_bbox_x1(repBBox);
```

Where Used: **(ael)**

Schematic, Layout

db_get_bbox_x2()

Returns an integer, the x component of the upper-right corner of an object's bounding box. The bounding box is the maximum extent of any object, and is described as a rectangle.

Syntax:

```
db_get_bbox_x2(bboxHandle);
```

where

bboxHandle is the handle of an object's bounding box.

Example:

```
decl repBBox, x, repHandle;
repBBox = db_get_rep_bbox (repHandle);
x = db_get_bbox_x2(repBBox);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Export all Variables

Where Used: **(ael)**

Schematic, Layout

db_get_bbox_y1()

Returns an integer, the y component of the lower-left corner of an object's bounding box. The bounding box is the maximum extent of any object, and is described as a rectangle.

Syntax:

```
db_get_bbox_y1(bboxHandle);
```

where

bboxHandle is the handle of an object's bounding box.

Example:

```
decl repBBox, y, repHandle;
```

```
repBBox = db_get_rep_bbox (repHandle);
y = db_get_bbox_y1(repBBox);
```

***Where Used:* (ael)**

Schematic, Layout

db_get_bbox_y2()

Returns an integer, the y component of the upper-right corner of an object's bounding box. The bounding box is the maximum extent of any object, and is described as a rectangle.

Syntax:

```
db_get_bbox_y2(bboxHandle);
```

where

bboxHandle is the handle of an object's bounding box.

Example:

```
decl repBBox, y, repHandle;
repBBox = db_get_rep_bbox (repHandle);
y = db_get_bbox_y2(repBBox);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Export all Variables

***Where Used:* (ael)**

Schematic, Layout

db_get_instance_bbox()

Returns a handle to the bounding box of an instance. This is the smallest rectangle that completely encloses all data of an instance.

Syntax:

```
db_get_instance_bbox(instHandle, type);
```

where

InstHandle is the handle to an instance.

type indicates the type of bounding box, where:

- 0 = symbol bounding box
- 1 = annotation bounding box
- 2 = bounding box to include both symbol and annotation

Example:

```
decl bbox, x1;
bbox = db_get_instance_bbox(instHandle, 0);
x1 = db_get_bbox_x1(bbox);
```

***Where Used:* (ael)**

Schematic, Layout

db_get_instance_description()

Returns a string, the AEL component description string for an instance.

Syntax:

```
db_get_instance_description(instHandle);
```

where

InstHandle is the handle to an instance.

Example:

```
decl string, instH;
string = db_get_instance_description(instH);
```

***Where Used:* (ael)**

Schematic, Layout

db_get_instance_parm()

Returns a real number, the nominal value of an instance parameter given the parameter name or index (position in list of parameters).

Syntax:

```
db_get_instance_parm(designName, instHandle [, parameterReference]);
```

where

designName is the name of the design the instance is in.

instHandle is the instance value handle returned from a function such as *db_find_instance()*.

parameterReference is optional; the name of the parameter. If not supplied, 0.

Example:

```
decl inst, nomVal;
inst = db_find_instance(de_current_design_name(), 0, "TL1"); /* find TL1 in
current schematic design */
nomVal = db_get_instance_parm(de_current_design_name(), inst, "W"); /*
retrieve nominal value of W (width)*/
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.

Export a Modelcard to a MDIF File
Get Parameter Value

***Where Used:* (ael)**

Schematic, Layout

db_get_location_angle()

Returns an integer, the angle given a location handle in .001 of a degree. Location handles can be retrieved from pins using *db_get_pin_attribute()* with the PIN_LOCATION attribute.

Syntax:

```
db_get_location_angle(locationHandle);
```

where

locationHandle is the handle to a location.

Example:

```
decl ang, loc;
ang = db_get_location_angle(loc);
```

***Where Used:* (ael)**

Schematic, Layout

db_get_location_x()

Returns an integer, the x position given a location handle. Location handles can be retrieved from pins using *db_get_pin_attribute()* with the PIN_LOCATION attribute.

Syntax:

```
db_get_location_x(locationHandle);
```

where

locationHandle is the handle to a location.

Example:

```
decl x, loc;
x = db_get_location_x(loc);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
EEsof Part List - Modified
Replace all GROUND's with node name

***Where Used:* (ael)**

Schematic, Layout

db_get_location_y()

Returns an integer, the y position given a location handle. Location handles can be retrieved from pins using *db_get_pin_attribute()* with the PIN_LOCATION attribute.

Syntax:

```
db_get_location_y(locationHandle);
```

where

locationHandle is the handle to a location.

Example:

```
decl x, loc;
x = db_get_location_y(loc);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
EEsof Part List - Modified
Replace all GROUND's with node name

***Where Used:* (ael)**

Schematic, Layout

db_get_map()

Returns a handle to the current transformation mapping. Used in conjunction with *db_setup_map()* and *db_set_map()*.

Syntax:

```
db_get_map();
```

Example:

```
// get the transform of the instance and transform the points of the
// first shape and print the new coordinates to stderr
decl saveMap;
decl trans, points;
saveMap = db_get_map();
trans = db_get_instance_attribute(db_find_instance("mydsn",REP_LAY,"TL1"),
INST_TRANSFORM);
db_setup_transform(trans);
decl pointsCopy=points=db_first_point(db_first_dg(db_first_mask
(db_get_design_attribute(db_get_design("mydsn"),DESIGN_REP_LAY))));
pointsCopy = db_transform_points(pointsCopy);
while(points)
{
   decl coord= db_get_coord(points);
   decl x1=db_get_x(coord);
   decl y1=db_get_y(coord);
   fputs(stderr,fmt_tokens(list(x1,y1)));
   points=db_next_point(points);
}
db_set_map(saveMap); // restore the original mapping
```

***Where Used:* (ael)**

Schematic, Layout

db_get_map_attribute()

Returns an attribute of a transformation mapping, as retrieved by *db_get_map()* . Values are returned in database units, angle in 0.001 degrees, scale as a real number, and mirror flag as an integer 1 or 0.

Note that a mirror about the Y axis is stored in the transformation mapping as a combination of a mirror about the X axis followed by a 180 degree rotation, so query for the DB_MAP_MIR_Y attribute always returns 0. If a mirror about the Y axis was part of the transformation mapping, query for DB_MAP_MIR_X returns 1, and query for DB_MAP_ANG returns 180000 (or 180 degrees). If a rotation was involved as well as mirror, then query for the DB_MIR_ANG includes the rotation angle also.

Syntax:

```
db_get_map_attribute( mapAttribute [, map] );
```

where

mapAttribute is the attribute code for the map, where:

- DB_MAP_X = Horizontal translation value
- DB_MAP_Y = Vertical translation value
- DB_MAP_ANG = Rotation angle
- DB_MAP_SCL_X = Horizontal scale factor
- DB_MAP_SCL_Y = Vertical scale factor
- DB_MAP_MIR_X = Flag for mirror about X axis
- DB_MAP_MIR_Y = Flag for mirror about Y axis (always 0)

map is optional; the handle of transformation mapping. If omitted, assumes current mapping as set by *db_set_map()*, *db_setup_transformation()*, or *db_setup_map()*.

Example:

```
decl x;
x = db_get_map_attribute( DB_MAP_X );
```

Where Used: **(ael)**

Schematic, Layout

db_get_node_wires()

Returns a handle to the segment representing a wire for a given node.

Syntax:

```
db_get_node_wires(nodeHandle);
```

where

nodeHandle is the handle to a node returned from a function such as *db_first_node()* or *db_next_node()*.

Example:

```
decl segHandle, nodeHandle;
segHandle = db_get_node_wires(nodeHandle);
```

Where Used: **(ael)**

Schematic, Layout

db_get_parm_attribute()

Returns an attribute of a parameter.

Syntax:

```
db_get_parm_attribute(parmHandle, parmAttribute [ ,index]);
```

where

parmHandle is the handle to a parameter.

parmAttribute is the data for the parameter, where:

- PARM_VALUE_CODE = Type of parameter, where
  - 0 = no value
  - 1,2 = double
  - 3,4 = parameter list
  - 5,7 = string
  - 6 = no value, but form name is important
- PARM_NAME = Name of parameter
- PARM_FORM_NAME = Name of the parameter form
- PARM_NUM_DVALS = Number of double(real) values 0-3
- PARM_VALUE_DVALUE = Real value
- PARM_VALUE_SVALUE = String value (set if var reference)
- PARM_VALUE_LIST = List of parameters
- PARM_NO_PLOT = Flag indicating parameter visibility. Parameters are owned by instances. A parameter has a name and a value. The value can be a string, an array of 1-3 doubles or a further list of parameters. For example, in W=3, there is one dvalue (3); in W=gain, there is one svalue (gain). The value code indicates what the value type is. If the value is a double, the PARM_NUM_DVALS indicates how many make up the value; for example, constrained optimization has 3 dvalues (min, max and nominal). For complex values, it is often necessary to look up the form name and then the form definition to determine how the value is represented in the

data base. The AEL function format_instance_data() can convert complex parameter values into strings.

index is optional. If attribute is PARM_VALUE_DVALUE, the index can be 0, 1, or 2 to retrieve first, second, or third real value. If the index is not given, the first real value is returned.

Example:

```
decl pHandle = db_first_parm(instHandle);
decl fname = db_get_parm_attribute(pHandle, PARM_FORM_NAME);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.

Export a Modelcard to a MDIF File
Get MeasEqn String 2
Get Variable String 2

***Where Used:* (ael)**

Schematic, Layout

db_get_transform_angle()

Returns the angle of a transform object as returned from the function *db_get_instance_attribute()*. The return value is in thousandths of a degree, not in degrees. For example, a 90 degree angle is returned as 90000.

Syntax:

```
db_get_transform_angle(transformHandle);
```

where

transformHandle is the handle to an instance's transformation record.

Example:

```
decl angle;
decl instTrans;
instTrans = db_get_instance_attribute();
angle = db_get_transform_angle(instTrans);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
EEsof Part List - Modified
Get Instance Rotation

Where Used: **(ael)**

Schematic, Layout

db_get_transform_mirror_x()

Returns the x axis mirror flag of a transform object, as returned from the function *db_get_instance_attribute()*, where: 0 = not mirrored around the x axis, and 1 = mirrored around the x axis.

Syntax:

```
db_get_transform_mirror_x(transformHandle);
```

where

transformHandle is a handle to an instance's transformation record.

Example:

```
decl mirx;
decl instTrans;
instTrans = db_get_instance_attribute();
mirx = db_get_transform_mirror_x(instTrans);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
EEsof Part List - Modified
Get Instance Rotation

Where Used: **(ael)**

Schematic, Layout

db_get_transform_mirror_y()

Returns the y axis mirror flag of a transform object, as returned from the function

db_get_instance_attribute(), where: 0 = not mirrored around the y axis, and 1 = mirrored around the y axis.

Syntax:

```
db_get_transform_mirror_y(transformHandle);
```

where

transformHandle is a handle to an instance's transformation record.

Example:

```
decl miry;
decl instTrans;
instTrans = db_get_instance_attribute();
miry = db_get_transform_mirror_y(instTrans);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
EEsof Part List - Modified
Get Instance Rotation

Where Used: **(ael)**

Schematic, Layout

db_get_transform_x()

Returns the x coordinate of an instance's transformation record.

Syntax:

```
db_get_transform_x( transformHandle );
```

where

transformHandle is a handle to an instance's transformation record.

Example:

```
decl x;
decl instTrans;
instTrans = db_get_instance_attribute();
x = db_get_transform_x(instTrans);
```

***Where Used:* (ael)**

Schematic, Layout

db_get_transform_y()

Returns the y coordinate of an instance's transformation record.

Syntax:

```
db_get_transform_y( transformHandle );
```

where

transformHandle is a handle to an instance's transformation record.

Example:

```
decl y;
decl instTrans;
instTrans = db_get_instance_attribue();
y = db_get_transform_y(instTrans);
```

***Where Used:* (ael)**

Schematic, Layout

db_get_x()

Returns an integer, the x coordinate in data base units.

Syntax:

```
db_get_x(coordHandle);
```

where

coordHandle is a handle to a coordinate, as returned from *db_get_coord()*.

Example:

```
decl pointHandle, coordHandle, x;
coordHandle = db_get_coord(pointHandle);
x = db_get_x(coordHandle);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Find Close Vertexes
Polylines to Polygons

***Where Used:* (ael)**

Schematic, Layout

db_get_y()

Returns an integer; the y coordinate in data base units.

Syntax:

```
db_get_y(coordHandle);
```

where

coordHandle is a handle to a coordinate, as returned from *db_get_coord()*.

Example:

```
decl pointHandle, coordHandle, y;
coordHandle = db_get_coord(pointHandle);
y = db_get_y(coordHandle);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Find Close Vertexes
Polylines to Polygons

***Where Used:* (ael)**

Schematic, Layout

db_next_parm()

Returns a handle to the next parameter of the instance. Takes as an argument a previous handle to a parameter. NULL if end of the list.

Syntax:

```
db_next_parm(parmHandle);
```

where

parmHandle is a handle to a parameter as instance.

Example:

```
decl parmH;
db_next_parm(parmH);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.

Export a Modelcrad to a MDIF File
Get MeasEqn String 2
Get Variable String 2

Where Used: **(ael)**

Schematic, Layout

db_num_parms()

Returns the number of parameters in a parameter list.

Syntax:

```
db_num_parms(parmHandle);
```

where

parmHandle is a handle to a parameter list.

Example:

```
decl num, repH;
num = db_num_parms(db_first_instance(repH));
```

Where Used: **(ael)**

Schematic, Layout

db_set_map()

Sets the current transformation mapping. Returns: none.

Syntax:

```
db_set_map (mapHandle);
```

where

mapHandle is a handle to transformation mapping, as returned from *db_get_map()*.

Example:

```
decl saveMap, instTransH;
saveMap = db_get_map();
db_setup_transform(instTransH);
db_set_map(saveMap);
```

***Where Used:* (ael)**

Schematic, Layout

db_setup_map()

Modifies the current transformation mapping set by *db_set_map()*, *db_setup_transformation()*, or *db_setup_map()* by applying new mapping information with a cascading effect, with the effect of applying both maps in sequence. Returns: none.

Syntax:

```
db_setup_map( map );
```

db_setup_map([x, y, ang, xs, ys, mirX, mirY]);

where
In the first form:

- *map* is a map as retrieved by *db_get_map()* is the only argument.

In the second form:

- *x* is optional; the value of horizontal translation in database units.
- *y* is optional; the value of vertical translation in database units.
- *ang* is optional; the value of rotation angle in 0.001 degrees, rotation around x,y.
- *xs* is optional; the value of horizontal scale factor, a real number, scaled about x/y.
- *ys* is optional; the value of vertical scale factor, a real number, scaled about x/y.

- *mirX* is optional; the flag for mirror about horizontal line through x/y, after rotation.
- *mirY* is optional; the flag for mirror about vertical line through x/y, after rotation.

Example:

```
db_setup_map( 100, 0, 90000, 1.0, 1.0, 0, 0 );
```

Where Used: **(ael)**

Schematic, Layout

db_setup_transform()

Sets up the instance transformation mapping (rotation, translation and mirror). Returns: none.

Syntax:

```
db_setup_transform(instTransH);
```

where

instTransH is a handle to an instance transformation, as returned by *db_get_instance_attribute()*.

Example:

```
decl saveMap, instTransH;
saveMap = db_get_map();
db_setup_transform(instTransH);
db_set_map(saveMap);
```

Where Used: **(ael)**

Schematic, Layout

db_transform_angle()

Transforms an angle value by rotation and mirror values from the current transformation as set by *db_set_map()*, *db_setup_transformation()*, or *db_setup_map()*. Returns a modified value of angle.

See also: db_transform_points() , *db_transform_coord()* (ael).

Syntax:

```
db transform angle( angle );
```

where

angle is the value of angle in 0.001 degrees.

Example:

```
decl a;
a = db_transform_angle( 30 );
```

Where Used: **(ael)**

Schematic, Layout

db_transform_coord()

Transforms a single coordinate value using the current transformation as set by *db_set_map()*, *db_setup_transformation()*, or *db_setup_map()*. Returns a modified coordinate value.

See also: db_transform_points() , *db_transform_angle()* (ael).

Syntax:

```
db_transform_coord( coord );
```

where

coord is the coordinate value, in database units.

Example:

```
decl pnt, coord;
decl dgHandle;
pnt = db_first_point( dgHandle );
coord = db_get_coord( pnt );
coord = db_transform_coord( coord );
```

Where Used: **(ael)**

Schematic, Layout

de_get_gds_number()

This function returns the GDS number corrosponding to a layer number from a GDSII layer mapping file.

Syntax

```
decl gdsLayerNumber = de_get_gds_number( layerNumber, LayerMapFilePath);
```

where,

- `layerNumber` is the ADS layer number for which GDSII number is queried.
- `LayerMapFilePath` is the absolute path to the GDSII layer mapping file.

Return(s)

This function returns -1 if no GDSII number for the layer number is found.

Example

```
decl adsLayerNumber = 5;
decl LayerMapFilePath = "C:/users/LayerMap.map";
decl gdsNumber = de_get_gds_number(adsLayerNumber, LayerMapFilePath);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

Where Used (ael)

Layout

Design Context Functions

This section describes the following design context function:

- *db_clear_context()* (ael)
- *db_copy_context()* (ael)
- *db_get_cell_name()* (ael)
- *db_get_context_bbox()* (ael)
- *db_get_context_db_factor()* (ael)
- *db_get_context_unit_name()* (ael)
- *db_design_is_modified()* (ael)
- *db_get_dbu_to_uu_factor()* (ael)
- *db_get_library_name()* (ael)
- *db_get_mks_to_uu_factor()* (ael)
- *db_get_user_units_significant_digits()* (ael)
- *db_get_uu_to_dbu_factor()* (ael)
- *db_get_uu_to_mks_factor()* (ael)
- *db_get_view_name()* (ael)
- *db_is_same_context()* (ael)
- *db_save_design_without_prompting()* (ael)
- *de_create_new_layout_view()* (ael)
- *de_create_new_schematic_view()* (ael)
- *de_create_new_symbol_view()* (ael)
- *de_current_context_is_valid()* (ael)
- *de_find_design_context_from_name()* (ael)
- *de_get_current_design_context()* (ael)
- *de_get_design_context()* (ael)
- *de_get_design_context_from_name()* (ael)
- *de_get_windows_with_same_context()* (ael)
- *de_is_artwork_macro_context()* (ael)
- *de_is_layout_context()* (ael)
- *de_is_schematic_context()* (ael)
- *de_is_schematic_or_layout_context()* (ael)
- *de_is_symbol_context()* (ael)
- *de_show_context_in_new_window()* (ael)
- *de_window_has_valid_context()* (ael)

db_clear_context()

Clears everything in a given design context.
Returns NULL.

Syntax

```
db_clear_context(context);
```

Where,

- *context* is a valid *DesignContext* (ael).

Example

```
...
db_clear_context(designContext);
...
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

DesignContext in AEL (ael)
Design Context Functions (ael)

Where Used: (ael)

Schematic, Layout

db_copy_context()

Copies the shapes, pins, application objects, instances, and properties from a given source design context into a given destination design context.
Returns NULL.

Syntax

```
db_copy_context(sourceContext, destinationContext);
```

Where,

- *sourceContext* is the source design context to copy into the destination context.
- *destinationContext* is the destination design context to copy the source design context's contents into.

Example

```
db_copy_context( sourceContext, destContext);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_is_same_context() (ael)
DesignContext in AEL (ael)
Design Context Functions (ael)

Where Used (ael)

Schematic, Layout

db_design_is_modified()

Returns TRUE if the given design context has been modified, otherwise returns FALSE.

Syntax

```
decl isModified = db_design_is_modified(context);
```

Where,

- *context* is a valid *DesignContext* (ael).

Example

```
...
decl isModified = db_design_is_modified(designContext);
...
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

DesignContext in AEL (ael)
Design Context Functions (ael)

Where Used: (ael)

Schematic, Layout

db_get_cell_name()

Returns the cell name of a design context.

Syntax

```
decl cellName = db_get_cell_name(context);
```

Where,

- *context* is a *design context* (ael).

Example

```
decl context = de_get_design_context(winInst);
decl cellName = db_get_cell_name(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_library_name() (ael)

db_get_view_name() (ael)
db_get_component_name() (ael)
db_get_design_name() (ael)
de_parse_lib_cell_view_name() (ael)
de_format_lib_cell_view_name() (ael)
DesignContext in AEL (ael)
Design Context Functions (ael)

Where Used (ael)

Schematic, Layout

db_get_context_bbox()

Returns the bounding box of the given DesignContext.
The bounding box is the smallest rectangle that completely encloses all the design context's data.

Syntax

```
decl bboxH = db_get_context_bbox (context);
```

Where,

- *context* is the design context to get the bounding box information for.

Example

```
decl bbox = db_get_context_bbox(context);
decl x1 = db_get_bbox_x1(bbox);
decl y1 = db_get_bbox_y1(bbox);
decl x2 = db_get_bbox_x2(bbox);
decl y2 = db_get_bbox_y2(bbox);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_get_design_context() (ael)
DesignContext in AEL (ael)
Design Context Functions (ael)
db_get_bbox_x1() (ael)
db_get_bbox_y1() (ael)
db_get_bbox_x2() (ael)
db_get_bbox_y2() (ael)

Where Used (ael)

Schematic, Layout

db_get_context_db_factor()

Returns a real value conversion factor for converting user units to database units for a given DesignContext.

Note
This is an alias to *db_get_uu_to_dbu_factor()* (ael).

Syntax

```
decl uu2dbFactor = db_get_context_db_factor(context);
```

Where,

- *context* is a design context.

Example

```
decl uu2dbFactor = db_get_context_db_factor(de_get_design_context(winInstP));
dbY = uuY * uu2dbFactor;
uuX = db_get_x(coord)/ uu2dbFactor;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_context_unit_name() (ael)
db_get_dbu_to_uu_factor() (ael)
db_get_mks_to_uu_factor() (ael)
db_get_user_units_significant_digits() (ael)
db_get_uu_to_dbu_factor() (ael)
db_get_uu_to_mks_factor() (ael)

Where Used (ael)

Schematic, Layout

db_get_context_unit_name()

Returns the string unit name used by the given DesignContext.

Syntax

```
decl unitName = db_get_context_unit_name(context);
```

Where,

- *context* is the design context to get the string unit name for.

Example

```
decl unitName = db_get_context_unit_name(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_dbu_to_uu_factor() (ael)
db_get_mks_to_uu_factor() (ael)
db_get_user_units_significant_digits() (ael)
db_get_uu_to_dbu_factor() (ael)
db_get_uu_to_mks_factor() (ael)
DesignContext in AEL (ael)
Design Context Functions (ael)

Where Used (ael)

Schematic, Layout

db_get_dbu_to_uu_factor()

Returns the real value conversion factor for converting database units to user units for a given DesignContext.

Syntax

```
decl dbu2uu = db_get_dbu_to_uu_factor(context);
```

Where,

- *context* is the design context to get the real value units conversion factor from.

Example

```
decl uu_x1  =  dbu_x1 * db_get_dbu_to_uu_factor(context);
decl dbu_x2  = uu_x2  * db_get_uu_to_dbu_factor(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_context_unit_name() (ael)
db_get_mks_to_uu_factor() (ael)
db_get_user_units_significant_digits() (ael)
db_get_uu_to_dbu_factor() (ael)
db_get_uu_to_mks_factor() (ael)

Where Used (ael)

Schematic, Layout

db_get_library_name()

Returns the library name of a design context.

Syntax

```
decl libName = db_get_library_name(context);
```

Where,

- *context* is a *design context* (ael).

Example

```
decl context = de_get_design_context(winInst);
decl libName = db_get_library_name(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_cell_name() (ael)
db_get_view_name() (ael)
db_get_component_name() (ael)
db_get_design_name() (ael)
de_parse_lib_cell_view_name() (ael)
de_format_lib_cell_view_name() (ael)
DesignContext in AEL (ael)
Design Context Functions (ael)

Where Used (ael)

Schematic, Layout

db_get_mks_to_uu_factor()

Returns the real value conversion factor for converting circuit MKS units to database user units for a given DesignContext.

Syntax

```
decl uu2mks = db_get_mks_to_uu_factor(context);
```

Where,

- *context* is the design context to get the real value units conversion factor from.

Example

```
decl mks_x1 = uu_x1  * db_get_uu_to_mks_factor(context);
decl uu_x2  = mks_x2 * db_get_mks_to_uu_factor(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_context_unit_name() (ael)
db_get_dbu_to_uu_factor() (ael)
db_get_user_units_significant_digits() (ael)
db_get_uu_to_dbu_factor() (ael)
db_get_uu_to_mks_factor() (ael)

Where Used (ael)

Schematic, Layout

db_get_user_units_significant_digits()

Returns the integer number of user unit significant digits in the given DesignContext.
Gets the precision of the user units in the given design context.
This is the number of digits after the decimal point required when printing a user unit value to avoid precision loss.

Syntax

```
decl uuSigFigs = db_get_user_units_significant_digits(context);
```

Where,

- *context* is the design context to get the used integer user units precision from.

Example

```
decl uuSigFigs = db_get_user_units_significant_digits(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_context_unit_name() (ael)
db_get_dbu_to_uu_factor() (ael)
db_get_mks_to_uu_factor() (ael)
db_get_uu_to_dbu_factor() (ael)
db_get_uu_to_mks_factor() (ael)

Where Used (ael)

Schematic, Layout

db_get_uu_to_dbu_factor()

Returns the real value conversion factor for converting user units to database units for a given DesignContext.

Syntax

```
decl uu2dbFactor = db_get_uu_to_dbu_factor(context);
```

Where,

- *context* is a design context.

Example

```
decl uu2dbFactor = db_get_uu_to_dbu_factor(de_get_design_context(winInstP));
dbY = uuY * uu2dbFactor;
uuX = db_get_x(coord)/ uu2dbFactor;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011

See also

db_get_context_unit_name() (ael)
db_get_dbu_to_uu_factor() (ael)
db_get_mks_to_uu_factor() (ael)
db_get_user_units_significant_digits() (ael)
db_get_uu_to_mks_factor() (ael)

Where Used (ael)

Schematic, Layout

db_get_uu_to_mks_factor()

Returns the real value conversion factor for converting database user units to circuit MKS units for a given DesignContext.

Syntax

```
decl uu2mks = db_get_uu_to_mks_factor(context);
```

Where,

- *context* is the design context to get the real value units conversion factor from.

Example

```
decl mks_x1 = uu_x1  * db_get_uu_to_mks_factor(context);
decl uu_x2  = mks_x2 * db_get_mks_to_uu_factor(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_context_unit_name() (ael)
db_get_dbu_to_uu_factor() (ael)
db_get_mks_to_uu_factor() (ael)
db_get_user_units_significant_digits() (ael)
db_get_uu_to_dbu_factor() (ael)

Where Used (ael)

Schematic, Layout

db_get_view_name()

Returns the view name of a design context.

Syntax

```
viewName = db_get_view_name(context);
```

Where,

- *context* is a *design context* (ael).

Example

```
decl context = de_get_design_context(winInst);
decl viewName = db_get_view_name(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_library_name() (ael)
db_get_cell_name() (ael)
db_get_component_name() (ael)
db_get_design_name() (ael)
de_parse_lib_cell_view_name() (ael)
de_format_lib_cell_view_name() (ael)
DesignContext in AEL (ael)
Design Context Functions (ael)

Where Used (ael)

Schematic, Layout

db_is_same_context()

Returns TRUE if the two *design contexts* (ael) are equivalent. Returns FALSE if the two *design contexts* (ael) are not equivalent.

Syntax

```
decl isSame = db_is_same_context(designContext1, designContext2);
```

Where,

- *designContext1* is a *DesignContext* (ael).
- *designContext2* is a *DesignContext* (ael).

Example

```
// Get a particular symbol context
 decl symContext = de_get_design_context_from_name(symbolDesignName);
 decl context = de_get_current_design_context();
 if (db_is_same_context(symContext, context))
 {
     // Current design context is the same as the symbol context
     …
 }
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_get_design_context_from_name() (ael)
DesignContext in AEL (ael)
Design Context Functions (ael)

Where Used (ael)

Schematic, Layout

db_save_design_without_prompting()

Saves the given *design context* (ael) without notifying or prompting the user for permission.

Syntax

```
db_save_design_without_prompting(designContext);
```

Where,

- *designContext* is a valid *DesignContext* (ael).

Example

```
decl layerId = db_get_current_layerid(designContext);
db_add_rectangle(designContext, layerId, 150, 80, 180, 250);
...
db_save_design_without_prompting(designContext);
...
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

DesignContext in AEL (ael)
Design Context Functions (ael)

***Where Used* (ael)**

Schematic, Layout

de_create_new_layout_view()

Function to create a new layout view. Returns the *design context* (ael) to the created layout view.

Syntax

```
decl newContext = de_create_new_layout_view(libName,cellName,viewName);
```

Where,

- *libName* is the name of the existing library to create the cellview within.
- *cellName* is the name of the cell to create the view in.
- *viewName* is the name of the view to create.

Example

```
decl context = de_create_new_layout_view("myLib", "myCell", "layout");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Design Context Functions (ael)
de_create_new_schematic_view() (ael)
de_create_new_symbol_view() (ael)

***Where Used* (ael)**

Schematic, Layout

de_create_new_schematic_view()

Function to create a new schematic view. Returns the *design context* (ael) to the created schematic view.

Syntax

```
decl newContext = de_create_new_schematic_view(libName,cellName,viewName);
```

Where,

- *libName* is the name of the existing library to create the cellview within.
- *cellName* is the name of the cell to create the view in.
- *viewName* is the name of the view to create.

Example

```
decl context = de_create_new_schematic_view("myLib", "myCell", "schematic");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Design Context Functions (ael)
de_create_new_layout_view() (ael)
de_create_new_symbol_view() (ael)

Where Used (ael)

Schematic, Layout

de_create_new_symbol_view()

Function to create a new symbol view. Returns the *design context* (ael) to the created symbol view.

Syntax

```
decl newContext = de_create_new_symbol_view(libName,cellName,viewName);
```

Where,

- *libName* is the name of the existing library to create the cellview within.
- *cellName* is the name of the cell to create the view in.
- *viewName* is the name of the view to create.

Example

```
decl context = de_create_new_symbol_view("myLib", "myCell", "symbol");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Design Context Functions (ael)
de_create_new_layout_view() (ael)
de_create_new_schematic_view() (ael)

***Where Used* (ael)**

Schematic, Layout

de_current_context_is_valid()

Returns TRUE if there is a valid current *DesignContext* (ael). Returns FALSE otherwise.

See also: *de_get_current_design_context()* (ael), *de_get_design_context()* (ael), *de_window_has_valid_context()* (ael).

Syntax:

```
de_current_context_is_valid();
```

Example:

```
// Test if there is a valid current design context.
if (!de_current_context_is_valid())
   return;
// Get current design context.
decl context = de_get_current_design_context();
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

de_find_design_context_from_name()

This function is used to find the *DesignContext* (ael) from a design name. The function returns NULL if no *design context* (ael) could be found.

Note
This function does not throw an error if the design is not found, although it will give errors if there are problems loading a design or if the library is missing.

Syntax

```
decl context = de_find_design_context_from_name(dsnName);
```

Where,

- *dsnName* is the design name. The ADS design name is in the format

"libName:cellName:viewName"

Example

```
// Given design name find its design context.
decl context = de_find_design_context_from_name("myLibrary:mydesign:schematic";);
if (context == NULL)
  return; // DesignContext could not be found.
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_get_design_context_from_name() (ael)
db_get_design_name() (ael)
de_format_lib_cell_view_name() (ael)
DesignContext in AEL (ael)
Design Context Functions (ael)

***Where Used* (ael)**

Schematic, Layout

de_get_design_context_from_name()

This function is used to find the *DesignContext* (ael) from a design name. This function gives an AEL error if the design cannot be loaded or cannot be found.

Syntax

```
decl context = de_get_design_context_from_name(dsnName);
```

Where,

- *dsnName* is the design name. The ADS design name is in the format: "libName:cellName:viewName".

Example

```
// Given design name find its design context.
decl context = de_get_design_context_from_name("myLibrary:mydesign:schematic");
// If context couldn't be found an AEL Error message would have occurred.
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_find_design_context_from_name() (ael)

db_get_design_name() (ael)
de_format_lib_cell_view_name() (ael)
DesignContext in AEL (ael)
Design Context Functions (ael)

Where Used (ael)

Schematic, Layout

de_is_schematic_or_layout_context()

Returns TRUE if the given DesignContext or window instance's DesignContext is a schematic design context or is a layout design context; Returns FALSE otherwise.

Syntax

```
decl bIsSchemOrLayout = de_is_schematic_or_layout_context(context|winInst);
```

Where,

- *context* is the design context returned from a function such as *de_get_current_design_context()* (ael).
  or
- *winInst* is a window returned from a function such as *api_get_current_window()* (ael).

Example

```
decl bIsSchemOrLayout = de_is_schematic_or_layout_context(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_is_schematic_context() (ael)
de_is_layout_context() (ael)
de_is_symbol_context() (ael)
de_is_artwork_macro_context() (ael)

Where Used (ael)

Schematic, Layout

de_show_context_in_new_window()

This function opens the given design in a new window and returns the window.

Syntax

```
decl windowH = de_show_context_in_new_window( designContext );
```

Where,

- *designContext* is a valid *DesignContext* (ael).

Example

```
...
// Open the design inside a new window.
decl windowH = de_show_context_in_new_window(designContext);
...
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

DesignContext in AEL (ael)
Design Context Functions (ael)

***Where Used* (ael)**

Schematic, Layout

de_window_has_valid_context()

Returns TRUE if the given window has a valid *DesignContext* (ael). Returns FALSE otherwise.

See also: *de_get_current_design_context()* (ael), *de_current_context_is_valid()* (ael), *de_get_design_context()* (ael).

Syntax:

```
de_window_has_valid_context(winInst);
```

where

winInst is a window returned from a function such as *api_get_current_window()*.

Example:

```
decl winInst = api_get_current_window();
// Test if current window has a valid design context.
if (!de_window_has_valid_context(winInst))
   return;
// Get context from window.
decl context = de_get_design_context(winInst);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

de_get_current_design_context()

Returns the current *DesignContext* (ael). Gives an error if there is no current DesignContext available. If an artwork macro is currently running, the macro's DesignContext is returned. If a schematic window that is not in symbol view is current, a schematic DesignContext is returned. If a symbol window or schematic window that is in symbol view is current, a symbol DesignContext is returned. If a layout is current then a layout DesignContext is returned.

See also: *de_current_context_is_valid()* (ael), *de_get_design_context()* (ael), *de_window_has_valid_context()* (ael).

Syntax:

```
decl context = de_get_current_design_context();
```

Example:

```
decl context = de_get_current_design_context();
// Create an instance iterator for all instances in the current design context.
decl instIter = db_create_inst_iter(context);
if (db_inst_iter_is_valid(instIter))
{
   // Mark the first instance in the design context as selected.
   db_select(instIter, TRUE);
}
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

de_get_design_context()

Returns the *DesignContext* (ael) of the given window. Gives an error if there is no DesignContext for the given window. If a schematic window that is not in symbol view is given, a schematic DesignContext is returned. If a symbol window or schematic window that is in symbol view is given, a symbol DesignContext is returned. If a layout window is given then a layout DesignContext is returned.

See also: *de_current_context_is_valid()* (ael), *de_get_current_design_context()* (ael), *de_window_has_valid_context()* (ael).

Syntax:

```
de_get_design_context(winInst);
```

where
winInst is a window returned from a function such as *api_get_current_window*().

Example:

```
decl winInst = api_get_current_window();
decl context = de_get_design_context(winInst);
// Create an instance iterator for all instances in the current design context.
```

```
decl instIter = db_create_inst_iter(context);
if (db_inst_iter_is_valid(instIter))
{
   // Mark the first instance in the design context as selected.
   db_select(instIter, TRUE);
}
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

de_get_windows_with_same_context()

Returns a list of windows that are currently displaying the same design and view as the given window or given *DesignContext* (ael).

See also: *api_get_current_window()* (ael), *de_get_current_design_context()* (ael), *de_get_design_context()* (ael).

Syntax:

```
de_get_windows_with_same_context(context|winInst);
```

where

context is a DesignContext returned from a function such as *de_get_current_design_context()*.
or
winInst is a window returned from a function such as *api_get_current_window()*.

Example:

```
// Get all windows that have the same design and view as the current context.
decl context = de_get_current_design_context();
decl winList = de_get_windows_with_same_context(context);
// Get all windows that have the same design and view as the current window.
decl winInst = api_get_current_window();
decl winList = de_get_windows_with_same_context(winInst);
```

Version Introduced

ADS 2009 Update 1

See also

de_get_windows_with_same_library() (ael)

***Where Used:* (ael)**

Schematic, Layout

de_is_artwork_macro_context()

Returns TRUE if the given *DesignContext* (ael) or the given window's *DesignContext* (ael) is the DesignContext used for an artwork macro evaluation. Returns FALSE otherwise.

See also: *de_get_current_design_context()* (ael), *de_get_design_context()* (ael), *de_is_layout_context()* (ael), *de_is_schematic_or_layout_context()* (ael), *de_is_schematic_context()* (ael), *de_is_symbol_context()* (ael).

Syntax:

```
de_is_artwork_macro_context(context|winInst);
```

where

context is a DesignContext returned from a function such as *de_get_current_design_context()*.
or
winInst is a window returned from a function such as *api_get_current_window()*.

Example:

```
// Test if current context is an artwork macro context.
decl context = de_get_current_design_context();
if (!de_is_artwork_macro_context(context))
   return;
// Test if current window has an artwork macro context.
decl winInst = api_get_current_window();
if (!de_is_artwork_macro_context(winInst))
   return;
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

de_is_layout_context()

Returns TRUE if the given *DesignContext* (ael) or given window's *DesignContext* (ael) is a layout window not doing artwork macro evaluation. Returns FALSE otherwise.

See also: *de_get_current_design_context()* (ael), *de_get_design_context()* (ael), *de_is_artwork_macro_context()* (ael), *de_is_schematic_or_layout_context()* (ael), *de_is_schematic_context()* (ael), *de_is_symbol_context()* (ael).

Syntax:

```
de_is_layout_context(context|winInst);
```

where

context is a DesignContext returned from a function such as *de_get_current_design_context()*.
or
winInst is a window returned from a function such as *api_get_current_window()*.

Example:

```
// Test if current context is a layout window.
decl context = de_get_current_design_context();
```

```
if (!de_is_layout_context(context))
   return;
// Test if current window is a layout window.
decl winInst = api_get_current_window();
if (!de_is_layout_context(winInst))
   return;
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

de_is_schematic_context()

Returns TRUE if the given *DesignContext* (ael) or given window's *DesignContext* (ael) is a schematic window that is not in symbol view. Returns FALSE otherwise.

See also: *de_get_current_design_context()* (ael), *de_get_design_context()* (ael), *de_is_artwork_macro_context()* (ael), *de_is_layout_context()* (ael), *de_is_schematic_or_layout_context()* (ael), *de_is_symbol_context()* (ael).

Syntax:

```
de_is_schematic_context(context|winInst);
```

where

context is a DesignContext returned from a function such as *de_get_current_design_context()*.
or
winInst is a window returned from a function such as *api_get_current_window()*.

Example:

```
// Test if current context is a schematic window.
decl context = de_get_current_design_context();
if (!de_is_schematic_context(context))
   return;
// Test if current window is a schematic window.
decl winInst = api_get_current_window();
if (!de_is_schematic_context(winInst))
   return;
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

de_is_symbol_context()

Returns TRUE if the given *DesignContext* (ael) or given window's *DesignContext* (ael) is a symbol window or a schematic window in symbol view. Returns FALSE otherwise.

See also: *de_get_current_design_context()* (ael), *de_get_design_context()* (ael),

de_is_artwork_macro_context() (ael), *de_is_layout_context()* (ael), *de_is_schematic_or_layout_context()* (ael), *de_is_schematic_context()* (ael).

Syntax:

```
de_is_symbol_context(context|winInst);
```

where

context is a DesignContext returned from a function such as *de_get_current_design_context()*.
or
winInst is a window returned from a function such as *api_get_current_window()*.

Example:

```
// Test if current context is a symbol window.
decl context = de_get_current_design_context();
if (!de_is_symbol_context(context))
   return;
// Test if current window is a symbol window.
decl winInst = api_get_current_window();
if (!de_is_symbol_context(winInst))
   return;
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

Design Environment Query Functions

This section describes each Design Environment Query function in detail. The functions are listed in alphabetical order.

db_factor()

Returns a real value, a conversion factor from layout user units to meters.

Syntax:

```
db_factor();
```

Example:

```
decl conFact, uUnit;
conFact = db_factor();
uUnit = 20*conFact;
```

***Where Used:* (ael)**

Schematic, Layout

de_window_is_open()

Determines the status of a window.
Returns the status of window, where: **TRUE** = window is open, **FALSE** = window is not open.

Syntax:

```
decl bIsOpen = de_window_is_open(windowType);
```

Where,

- *windowType* is the type of window where:

MAIN_WIN = Main window
SCHEM_WIN = Schematic window
LAYOUT_WIN = Layout window

Example:

```
if (de_window_is_open(SCHEM_WIN))
   fputs(stderr,"SCHEMATIC is open");
else
   fputs(stderr,"no SCHEMATIC window is open");
```

Where Used: **(ael)**

Schematic, Layout

de_ang_factor()

Returns a real number, a conversion factor to convert a value from simulator units to degrees.

Syntax:

```
de_ang_factor();
```

Example:

```
decl cfact, degree_angle;
cfact = de_ang_factor();
degree_angle = cfact * 30;
```

Where Used: **(ael)**

Schematic, Layout

de_current_design_type()

Returns the current design type (the design opened in the active window). The active window set with *api_set_current_window()*. Design types are defined in the simulator ael definition files *stdcmds.ael* and *stddefs.ael*.

Syntax:

```
de_current_design_type();
```

Example:

```
decl b;
b = de_current_design_type();
```

Where Used: **(ael)**

Schematic, Layout

de_get_alternate_window()

Returns the alternate window instance handle and sets the current window to this alternate. If the alternate window instance does not yet exist, it will be created with the design and the new window instance handle is returned. If the current window type is SCHEMATIC, then the alternate window is type LAYOUT and vice versa.

Syntax:

```
de_get_alternate_window(winInst, designHandle[, forceCreateWin]);
```

where

winInst is the current window instance.

designHandle is the handle of a valid design.

forceCreateWin is optional. Default = TRUE. Specifies whether or not to use an alternate window instance with the same design where:

- TRUE = if no alternate window is opened, returns an alternate window instance with the same design
- FALSE = if no alternate window is opened, returns NULL

Example:

```
decl otherWinInst, winInst, designHandle;
winInst = api_get_current_window();
designHandle = de_get_design_in_window(winInst);
otherWinInst = de_get_alternate_window(winInst, designHandle);
```

***Where Used:* (ael)**

Schematic, Layout

de_get_design_instances()

Returns a list of instance names of a given element, belonging to the named design.

Syntax:

```
de_get_design_instances(designName[, elementName, attrib, stringList,
separator]);
```

where

designName is optional; the design name.

elementName is optional; the element name (e.g., CAP). NULL if not used.

attrib is optional; the element attribute to search for. Values are the attributes used in *create_item()* . See *create_item() Attribute Choices* (ael). NULL if not used.

stringList is optional; list of names to which new names are appended.

separator is optional; the separator inserted between the components. NULL if not used.

Example:

```
decl inst_list, inst_name;
inst_list = de_get_design_instances(de_current_design_name());
inst_name = car(inst_list);
// e.g. inst_list=list("R1", "R2")
```

***Where Used:* (ael)**

Schematic, Layout

de_get_file_names()

Returns a list of files with given file extension. Used in conjunction with *create_text_form()*.

Syntax:

```
de_get_file_names(path, fileExt);
```

where

path is a string containing a list of colon-separated or semi-colon separated directories where files may be located. In most cases this will be obtained from configuration environment, using *getenv()*.

fileExt is the file extension (without the dot ".").

Example:

```
decl filenames;
filenames = de_get_file_names(getenv("HPTOLEMY_TEMPLATE_PATH"), "tpl");
```

***Where Used:* (ael)**

Schematic, Layout

de_get_variable_names()

Returns a list of all variable names declared in a design.

Syntax:

```
de_get_variable_names(designName);
```

where

designName is the name of the design.

Example:

```
decl varNames;
varNames = de_get_variable_names("amp");
while(listlen(varNames) > 0)
{
     fputs(car(varNames));
     varNames=cdr(varNames);
}
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Calculate Equations

***Where Used:* (ael)**

Schematic

de_get_window()

Returns the currently active window, where: MAIN_WIN = Main window, SCHEM_WIN = Schematic window, and LAYOUT_WIN = Layout window.

Syntax:

```
de_get_window();
```

Example:

```
decl win;
win = de_get_window();
if (win==SCHEM_WIN)
  fputs(stderr, "It's in schematic");
```

Where Used: **(ael)**

Schematic, Layout

de_invoke_help()

Accepts two arguments and returns a command to the help server which then launches a web browser and displays the online documentation.

See also: *de_post_help()* (ael).

Syntax

```
de_invoke_help("bookName","topicString");
```

where

bookName is the name of the directory where your documentation files are stored. *bookName* is entered as a string.

topicString is the name of a component or topic that you want to display documentation for. *topicString* is entered as a string.

Example

```
de_invoke_help ( "mydkitcomponents", "mydkit_res");
```

Where Used: **(ael)**

Schematic, Layout

de_post_help()

Accepts one argument and returns a command to the help server which then launches a web browser and displays the online documentation.

See also: *de_invoke_help()* (ael).

Syntax

```
de_post_help("topicString");
```

where

topicString is the name of a component or topic that you want to display documentation for. *topicString* is entered as a string.

Example

```
de_post_help ( "Open Project" );
```

***Where Used:* (ael)**

Schematic, Layout

de_retrieve_version_info()

Returns a long string of product version information for the Design Environment. The first two lines relate to Schematic Capture version and license information.

See also: *de_version_number_int()* (ael).

Syntax:

```
de_retrieve_version_info()
```

Example:

```
decl versionInfo;
  versionInfo=de_retrieve_version_info();
  fputs(stderr, versionInfo);
  // will print out:
  // Design Environment (*) 210.100 May 13 2002
  // FEATURE ads_schematic agileesof 2.100 1-may-2002 0 4C7ADBC38E671FCAEEA3
"CYWBCUX RHSVSOUEYGOKE" b5688f44
```

***Where Used:* (ael)**

Schematic, Layout

de_short_design_name()

Strips off the fully qualified path to the design and any extension, returning just the name of the design. Returns the short design name without an extension.

Syntax:

```
de_short_design_name(designName);
```

where

designName is the name of design.

Example:

```
decl shortName=de_short_design_name(db_get_design_attribute(designH,
DESIGN_NAME));
```

***Where Used:* (ael)**

Schematic, Layout

de_version_number_int()

Returns the current version number as an integer. For example, version 1.9 is returned as 190.

See also: *de_retrieve_version_info()* (ael).

Syntax:

```
de_version_number_int()
```

Example:

One way to see the contents of each string is to execute the following function call in the Options > Command_Line window:

```
fputs(stderr, de_version_number_int());
```

which prints the returned string to the command window where ADS was started.

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Get Registry Value

***Where Used:* (ael)**

Schematic, Layout

get_eqn_list()

Returns a list of variables and equations for the current design. The list is sorted in alphabetical order.

Syntax:

```
get_eqn_list();
```

Example:

```
decl vareqnList;
vareqnList = get_eqn_list();
while (listlen(vareqnList) > 0)
{
  fputs(stderr, car(vareqnList));
  vareqnList=cdr(vareqnList);
}
```

Where Used: **(ael)**

Schematic

get_item_list()

Returns a string list of all components (instance names) matching a component name placed in the current design.

Syntax:

```
get_item_list(itemName);
```

where

itemName is the name of the component.

Example:

```
decl names;
names = get_item_list("R");
```

Where Used: **(ael)**

Schematic, Layout

get_measurement_list()

Returns a string list of all measurement components active for the current design.

Syntax:

```
get_measurement_list();
```

Example:

```
decl mNames;
mNames = get_measurement_list();
```

Where Used: (ael)

Schematic

Design Functions

This section describes the following functions:

- *db_get_appropriate_base_name()* (ael)
- *db_get_appropriate_base_name_from_design_name()* (ael)

db_get_appropriate_base_name_from_design_name()

Returns a suitable filename from the given design name or component name. For example, find a name to use for a dataset file.
The returned filename will be in the form of *libName__cellName* if the component name was provided or *libName__cellName__viewName* if the design name was provided.

Note
The returned filename is not guaranteed to be unique and therefore cannot be mapped backwards to a design context.

Syntax

```
decl filename =
db_get_appropriate_base_name_from_design_name(designName|componentName);
```

Where,

- *designName*|*componentName* is a string design name or component name.

Note
For ADS 2011 the given design name will be in the format "library:cell:viewname" and the component name will be in the format "library:cell".

Example

```
decl context = de_get_current_design_context();
decl designName = db_get_design_name(context);  // Get design name of the design context.
// Example of _designName_ value for a design:
//   designName == my_wkspace_lib:top_level:schematic

decl fileName1 = db_get_appropriate_base_name_from_design_name(designName);  // Get a file name
for the designname.
// Example of returned _fileName1_ value for a design:
//   fileName1 == my_wkspace_lib__top_level__schematic

…
decl componentName = db_get_component_name(context);  // Get component name of the design context.
// Example of _componentName_ value for a component.
//   componentName == my_wkspace_lib:top_level

decl fileName2 = db_get_appropriate_base_name_from_design_name(componentName); // Get a file name
for the given componentName.
// Example of returned _fileName2_ value for a component:
//   fileName2 == my_wkspace_lib__top_level
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_appropriate_base_name() (ael)
db_get_library_name() (ael),
db_get_cell_name() (ael),
db_get_view_name() (ael),
db_get_design_name() (ael),
de_format_lib_cell_view_name() (ael),
de_parse_lib_cell_view_name() (ael)

Where Used (ael)

Schematic, Layout

db_get_appropriate_base_name()

Returns a suitable filename for the given design context. For example, find a name to use for a dataset file.
The base filename returned may include the view name by using the optional 2nd argument *includeViewName*.
The returned filename will be in the form of *libName__cellName*, or in the form of *libName__cellName__viewName* if the argument *includeViewName* was set TRUE.

Note
The filename is not guaranteed to be unique and therefore cannot be mapped backwards to a design context.

Syntax

```
decl filename = db_get_appropriate_base_name(context [,includeViewName]);
```

Where,

- *context* is a DesignContext.
- *includeViewName* (optional) is a Boolean set to TRUE or FALSE.

  Note
  By default this argument is FALSE if the user does not provide an *includeViewName* argument value.

Example

```
decl context = de_get_current_design_context();
decl designName = db_get_design_name(context);  // Get design name of the design context.
// Example of a _designName_ value for a schematic design:
//   designName == my_wkspace_lib:top_level:schematic

decl fileName1 = db_get_appropriate_base_name(context);  // Get a file name for the design
context.
// Example of _fileName1_ value for a schematic design:
//   fileName1 == my_wkspace_lib__top_level

decl fileName2 = db_get_appropriate_base_name(context, TRUE); // Get a file name for the design
context.
// Example of _fileName2_ value for a schematic design:
//   fileName2 == my_wkspace_lib__top_level__schematic
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_appropriate_base_name_from_design_name() (ael)
db_get_library_name() (ael),
db_get_cell_name() (ael),
db_get_view_name() (ael),
db_get_design_name() (ael),
de_format_lib_cell_view_name() (ael),
de_parse_lib_cell_view_name() (ael)

***Where Used* (ael)**

Schematic, Layout

ExpressionContext Functions

- *ExpressionContext Overview* (ael)

This section describes the following ExpressionContext functions:

- *db_setup_expression_context()* (ael)
- *db_evaluate_param_expression()* (ael)
- *db_evaluate_param_value()* (ael)
- *db_evaluate_param_value_no_expr()* (ael)
- *db_evaluate_inst_param_value()* (ael)
- *db_evaluate_inst_param_value_no_expr()* (ael)

db_evaluate_inst_param_value_no_expr()

Evaluates an instance parameter value with the given parameter name without expression evaluation. Returns the parameter's nominal value converted to the appropriate units but without expression evaluation. If the returned parameter value is a number, it will be returned in MKS units.

Note
If the parameter value contains an expression then an AEL error will occur.

Syntax

```
decl paramValue = db_evaluate_inst_param_value_no_expr(inst, paramName);
```

Where,

- *inst* is the instance with the parameter to be evaluated.
- *paramName* is the name of the parameter on the instance to evaluate the parameter value from.

Example

```
decl context = de_get_current_design_context();
if (!db_is_schematic_context(context))
   return NULL;

decl instH = db_find_instance_ex(context, "MyXVar");
decl paramValue = db_evaluate_inst_param_value_no_expr(instH, "Freq");
return paramValue;  // Return the parameter's nominal value.
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_evaluate_inst_param_value() (ael), *ExpressionContext Overview* (ael)

Where Used (ael)

Schematic, Layout

db_evaluate_inst_param_value()

Returns the evaluated expression value for an instance's parameter with a given parameter name within the scope of the given *ExpressionContext* (ael). If the returned evaluated expression value is a number, it will be returned in MKS units.

Syntax

```
decl expressValue = db_evaluate_inst_param_value(expressContext, inst,
paramName);
```

Where,

- *expressContext* is an *ExpressionContext* (ael) that contains the scope of the parameter being evaluated.
  An *ExpressionContext* (ael) can be created with the function *db_setup_expression_context()* (ael).
- *inst* is the instance with the parameter to be evaluated within the *ExpressionContext* (ael)'s scope.
- *paramName* is the name of the parameter on the instance to evaluate the parameter value from.

Example

```
decl context = de_get_current_design_context();
if (!db_is_schematic_context(context))
   return NULL;

decl instH = db_find_instance_ex(context, "MyXVar");
decl expressionContext = db_setup_expression_context(context, FALSE);
decl evaluatedExpression = db_evaluate_inst_param_value(expressionContext, instH, "Freq");
return evaluatedExpression;  // Return the parameter's evaluated expression value.
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_evaluate_inst_param_value_no_expr() (ael), *db_setup_expression_context()* (ael), *ExpressionContext Overview* (ael)

Where Used (ael)

Schematic, Layout

db_evaluate_param_expression()

Evaluate an expression in a given expression context. The *Expression Context* (ael) argument defines the scope of the expression being evaluated. Returns an evaluated expression value. If the value is a number, it will be returned in MKS units

Syntax

```
decl expressValue = db_evaluate_param_expression(expressContext, expression);
```

Where,

- *expressContext* is an *Expression Context* (ael) returned from *db_setup_expression_context()* (ael).
- *expression* is string expression to be evaluated within the scope of the given *ExpressionContext* (ael).

Example

```
decl context = de_get_current_design_context();
if (!db_is_layout_context(context))
   return NULL;
decl expressionContext = db_setup_expression_context(context, TRUE);
/// evaluate the expression "X" in the expression context.
decl evaluatedExpression = db_evaluate_param_expression(expressionContext, "X * 50 Ohms");
return evaluatedExpression;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_setup_expression_context() (ael), *ExpressionContext Overview* (ael)

Where Used (ael)

Schematic, Layout

db_evaluate_param_value_no_expr()

Evaluates a parameter value without expression evaluation. Returns the parameter's nominal value converted to the appropriate units but without expression evaluation. If the returned parameter value is a number, it will be returned in MKS units.

Note
If the parameter value contains an expression then an AEL error will occur.

Syntax

```
decl paramValue = db_evaluate_param_value_no_expr(paramIter);
```

Where,

- *paramIter* is a parameter iterator that is referencing a parameter definition and parameter value that will be evaluated.

Example

```
decl context = de_get_current_design_context();
if (!db_is_schematic_context(context))
   return NULL;
```

```
decl instH = db_find_instance_ex(context, "MyXVar");
decl paramValue = NULL;
decl paramIter = db_create_param_iter(instH);
decl foundParamIter = db_param_iter_find_by_name(paramIter, "Freq");
if (db_param_iter_is_valid(foundParamIter))
{
   // Return the parameter's nominal value.
   paramValue = db_evaluate_param_value_no_expr(foundParamIter);
}
return paramValue;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_evaluate_param_value() (ael), *ExpressionContext Overview* (ael)

Where Used (ael)

Schematic, Layout

db_evaluate_param_value()

Returns the evaluated expression value from a given parameter within the scope of the given *ExpressionContext* (ael). If the returned evaluated expression value is a number, it will be returned in MKS units.

Syntax

```
decl expressValue = db_evaluate_param_value(expressContext, paramIter);
```

Where,

- *expressContext* is an *ExpressionContext* (ael) that contains the scope of the parameter being evaluated.
  An *ExpressionContext* (ael) can be created with the function *db_setup_expression_context()* (ael).
- *paramIter* is a parameter iterator that is referencing a parameter definition and parameter value that will be evaluated.

Example

```
decl context = de_get_current_design_context();
if (!db_is_schematic_context(context))
   return NULL;

decl instH = db_find_instance_ex(context, "MyXVar");
decl expressionContext = db_setup_expression_context(context, FALSE);
decl evaluatedExpression = NULL;
decl paramIter = db_create_param_iter(instH);
decl foundParamIter = db_param_iter_find_by_name(paramIter, "Freq");
if (db_param_iter_is_valid(foundParamIter))
{
   // Evaluate the parameter's value.
   evaluatedExpression = db_evaluate_param_value(expressionContext, foundParamIter);
```

```
}
return evaluatedExpression;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_evaluate_param_value_no_expr() (ael), *db_setup_expression_context()* (ael), *ExpressionContext Overview* (ael)

Where Used (ael)

Schematic, Layout

db_setup_expression_context()

Setup an *Expression Context* (ael) to use for expression evaluation. The returned *Expression Context* (ael) defines the scope for expressions to be evaluated. If argument useLayout is set TRUE, the expression hierarchy will use the "netlist from layout" policy, if set to FALSE, the expression evaluation will use the "netlist from schematic" policy.

Syntax

```
decl expressionContext = db_setup_expression_context(context, useLayout);
```

Where,

- *context* is a DesignContext or a HierarchyContext.
- *useLayout* is a TRUE or FALSE value, if TRUE will use the "netlist from layout" policy, if FALSE will use the "netlist from schematic" policy.

Example

```
decl context = de_get_current_design_context();
if (!db_is_layout_context(context))
   return NULL;
decl expressionContext = db_setup_expression_context(context, TRUE);
/// evaluate the expression "X" in the expression context.
decl evaluatedExpression = db_evaluate_param_expression(expressionContext, "X * 50 Ohms");
return evaluatedExpression;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

ExpressionContext Overview (ael), *db_evaluate_param_expression()* (ael), *db_evaluate_param_value()* (ael), *db_evaluate_inst_param_value()* (ael)

Where Used (ael)

Schematic, Layout

ExpressionContext - Overview

This section describes what is an ExpressionContext and how you can evaluate parameter expression values by using the AEL ExpressionContext based functions.

An ExpressionContext is an AEL object that holds information about a particular design hierarchy to use for evaluating expressions. The expression context is essentially a hierarchy context that is setup to evaluate expressions in a manner consistent with netlisting from layout or netlisting from schematic.

An ExpressionContext is created from a DesignContext or a HierarchyContext. The function *db_setup_expression_context()* (ael) can be used to create an ExpressionContext.

Example use of ExpressionContext AEL objects

Example 1: Evaluate an expression string

```
// Evaluate an expression string
decl context = de_get_current_design_context();
decl expressionContext = db_setup_expression_context(context, TRUE/*useLayout*/);

/// evaluate an expression
decl evaluatedExpression = db_evaluate_param_expression(expressionContext, "X * 50 Ohms");
```

Example 2: Evaluate a parameter's value.

```
// Evaluate a parameter's value.
decl context = de_get_current_design_context();
if (!db_is_schematic_context(context))
   return NULL;

decl instH = db_find_instance_ex(context, "MyXVar");
decl expressionContext = db_setup_expression_context(context, FALSE);
decl evaluatedExpression = NULL;
decl paramIter = db_create_param_iter(instH);
decl foundParamIter = db_param_iter_find_by_name(paramIter, “Freq”);
if (db_param_iter_is_valid(foundParamIter))
{
   // Evaluate the parameter's value.
   evaluatedExpression = db_evaluate_param_value(expressionContext, foundParamIter);
}
return evaluatedExpression;
```

Expected outputs example from using the ExpressionContext functions.

Example A: Evaluate value with units and no expression.

Testing using an expression and using an instance with a parameter with value such as Freq=33.33 kHz, produces the following return values for these function calls:

decl context = de_get_current_design_context();
decl expressionContext = db_setup_expression_context(context, FALSE);

1.) evaluatedExpression = db_evaluate_param_expression(expressionContext, "33.33 kHz");

evaluatedExpression == 3.3330000000000000e+04

2.) evaluatedExpression = db_evaluate_param_value(expressionContext, paramIter);
evaluatedExpression == 33330

3.) paramValue = db_evaluate_param_value_no_expr(paramIter);
paramValue == 33330

4.) evaluatedExpression = db_evaluate_inst_param_value(expressionContext, instH, "Freq");
evaluatedExpression == 33330

5.) paramValue = db_evaluate_inst_param_value_no_expr(instH, "Freq");
paramValue == 33330

Example B: Evaluate value with units and expression.

Second test case has an expression where a variable X is used. The variable X is = 7. Testing using an expression and an instance with a parameter with value such as Freq=33.33 kHz + X, produces the following return values for the function calls:

decl context = de_get_current_design_context();
decl expressionContext = db_setup_expression_context(context, FALSE);

1.) evaluatedExpression = db_evaluate_param_expression(expressionContext, "33.33 kHz + X");
evaluatedExpression == 3.3337000000000000e+04

2.) evaluatedExpression = db_evaluate_param_value(expressionContext, paramIter);
evaluatedExpression == 33337

3.) paramValue = db_evaluate_param_value_no_expr(paramIter);
(AEL ERROR occurs) == Parameter "Freq" contains an expression "33.33 kHz + X" that could not be evaluated.

4.) evaluatedExpression = db_evaluate_inst_param_value(expressionContext, instH, "Freq");
evaluatedExpression == 33337

5.) paramValue = db_evaluate_inst_param_value_no_expr(instH, "Freq");
(AEL ERROR occurs) == Parameter "Freq" contains an expression "33.33 kHz + X" that could not be evaluated.

ExpressionContext AEL functions

See *ExpressionContext Functions* (ael)

Version Introduced

ADS 2011

File Handling Functions

This section describes each File Handling function in detail. The functions are listed in alphabetical order.

ael_gfile_hasext() (ael)	*fprintf()* (ael)
chdir() (ael)	*fputs()* (ael)
fclose() (ael)	*freopen()* (ael)
fflush() (ael)	*remove()* (ael)
fgets() (ael)	*rename()* (ael)
fopen() (ael)	

ael_gfile_hasext()

Takes a string parameter and returns an integer that indicates where an extension exists in that string. An extension is found if there is a "." in the string. Returns: -1 if no extension is found. Otherwise it returns the position at which the extension occurs. If more than one "." is found, it assumes that the last "." found indicates the extension follows. Note that the position is 1-based.

Syntax:

```
ael_gfile_hasext(filename);
```

where

filename is a string; the name of the file.

Example:

```
fputs(stderr, ael_gfile_hasext("design.ael")); //7 is the output
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

chdir()

Changes the current directory to the specified directory. Changes the program's reference directory, making the "." directory be the new directory. Any configuration files are re-read. Returns: TRUE if a directory is created successfully; else FALSE.

Syntax:

```
chdir(directoryName);
```

where

directoryName is a string; name of the directory to change to.

Example:

```
decl a;
a = chdir("myDir");
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

fclose()

Closes a file opened with *fopen()* command. Returns: none.

Syntax:

```
fclose(fileID);
```

where

fileID is the file identifier returned by *fopen()*.

Example:

In this example "hello there" is written in a file as "hello".

```
decl fID;
fID = fopen("hello", "W");
   fputs(fID, "hello there");
fclose(fID);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.

File Exists?
Get File Size
Write Permission to File?

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic,

Layout, Simulation, GUI

fflush()

Flushes buffers to disk when writing a file opened with *fopen()* for write or append. Returns a value indicating whether buffers were successfully flushed, where:
0 = successful, and 1 = not successful

Syntax:

```
fflush(file);
```

where

fileID is the file value, created with *fopen()*.

Example:

```
decl fID;
fID = fopen("hello", "W");
   fflush(fID, "hello there");
fclose(fID);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

fgets()

Reads a line from a file. Returns a string; the current line is returned and the line advanced. Returns an empty string for blank lines. Returns NULL at end-of-file.

Syntax:

```
fgets(fileID);
```

where

fileID is the file value, created with *fopen()*.

Example:

```
decl fID, fLine;
```

```
fID = fopen("hello", "R");
fLine = fgets(fID);
fLine = "hello there"
fclose(fID);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Copy File
Insert Equations from File
Split SP2 MDIF File

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

fopen()

Opens a file and returns a file identifier. At some point the file must be closed using *fclose()*.

Syntax:

```
fopen(fileName, mode);
```

where

fileName is a string; name of file to open.

mode is a string; type of action to perform, where:

- "W" = write
- "R" = read
- "A" = append

Example:

```
decl fid;
fid = fopen("atest.txt", "W");
   fputs(fid, "hello");
fclose(fid);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and

Examples website. You will need a valid login ID to view the contents of this site and download an example file.
File Exists?
Get File Size
Write Permission to File?

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

fprintf()

Print formatted text to a file. This function works like the C language counterpart. An argument is required for each format code in the format string. Unlike the function fputs(), newline is not automatically appended to the printed text. Returns: none.

See also: *sprintf()* (ael), *fputs()* (ael).

Syntax:

```
fprintf(file, fmt, args...);
```

where

file is the open file handle for the output file.

fmt is the output format string.

args are the values to be included in the formatted output, as required by the format string. The format codes recognized are:

- %c is the single ASCII character
- %d is the signed decimal integer
- %e is the floating point value in scientific format
- %E is the same as %e but with E in exponent
- %f is the floating point value in fixed decimal format
- %g is the floating point value in flexible format, most compact of %e or %f
- %G is the same as %g but with E printed in exponent instead of e
- %i is the integer value
- %o is the octal integer
- %s is the string value
- %u is the unsigned integer value
- %x is the hexadecimal integer, using lower case abcdef
- %X is the hexadecimal integer, using upper case ABCDEF

Example:

Prints to the standard error stream: *The value of x is 5.5*.

```
x=5.5;
fprintf (stderr, "The value of %s is %f\n", "x", x);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Copy File
Layer/Process Manager

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

fputs()

Writes a value to the file or device specified. Adds a new line, \n, at the end of the text string. Then the file must be closed using *fclose()* (ael). Returns: none.

Syntax:

```
fputs(fileID, value);
```

where

fileID is the file identifier returned from *fopen()* (ael).

value is a string or a variable name. If a string is desired, it should be placed within quotes; if a variable name is desired, no quotes should be used.

Example:

```
decl fid;
fid = fopen("a test", "W");
   fputs(fid, "hello");
fclose(fid);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.

Copy File
Export a Modelcard to a MDIF File
Modify Circle Resolution

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

freopen()

Opens the named file for reading or writing, attaching an existing file handle to it. Returns a file value; NULL if the file cannot be opened (that is, file does not exist). Binary mode is not currently supported for files.

See also: *fopen()* (ael).

Syntax:

```
freopen(name, mode, handle);
```

where

name is a string; name of file to open.

mode is a string; type of action to perform, where:

- "W" = write
- "R" = read
- "A" = append

handle is a string; handle to be reopened.

Example:

```
decl file, fid;
file = freopen ("myfile.dat", "R", fid);
fclose(fid);
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

remove()

Deletes a given file. Returns: none.

Syntax:

```
remove(filename);
```

where

filename is the name of file to be deleted.

Example:

```
remove("myfile.txt");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Get Registry Value
Write Permission to Directory?

Note
Designs should be removed with de_delete_design() .

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

rename()

Renames a file. Fails if the new file already exists. Returns FALSE if successful, TRUE if error.

Syntax:

```
rename(name, new);
```

where

name is current name of file.

new is new name of file.

Example:

```
rename("myfile.dat", "newfile.dat");
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic,

Form Definition Functions

- ***Form Definition - Overview*** **(ael)**

This section describes the following functions:

- *dm_form_get_name()* (ael)
- *dm_form_get_label()* (ael)
- *dm_form_get_net_format()* (ael)
- *dm_form_get_disp_format()* (ael)
- *dm_form_get_class()* (ael)
- *dm_form_get_attr()* (ael)
- *dm_form_get_parms()* (ael)
- *dm_form_is_constant_form()* (ael)
- *dm_form_is_discrete()* (ael)
- *dm_get_string_form_class_selection()* (ael)

See also

- *Form Set - Overview* (ael)
- *Form Set Functions* (ael)
- *create_compound_form()* (ael)
- *create_constant_form()* (ael)
- *create_text_form()* (ael)

dm_form_get_attr()

Returns the *form* (ael) attribute code for the given *form* (ael).
The typical form attributes are: **FORM_TUNABLE** and **FORM_DISCRETE**.
For more information on the different *form* (ael) attributes, see *Form Attributes* (ael).

Syntax

```
decl formAttributeCode = dm_form_get_attr(formH);
```

Where,

- *formH* is a *form definition* (ael).

Example

```
decl formAttributeCode = dm_form_get_attr(formH);
// Test if this is a discrete valued parameter.
if ((formAttributeCode & FORM_TUNABLE) && (formAttributeCode & FORM_DISCRETE))
{
   // Found discrete value parameter.
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Form Definition - Overview (ael)
Form Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_form_get_class()

Returns the integer class code that represents the given *form* (ael)'s class type.
The typical class types are: **DM_NAME_CLASS**, **DM_NAMED_LIST_CLASS**, and **DM_STRING_CLASS**.
For more information on the different classes that define a *form* (ael), see *Form Class Codes* (ael).

Syntax

```
decl formClassCode = dm_form_get_class(formH);
```

Where,

- *formH* is a *form definition* (ael).

Example

```
decl formClassCode = dm_form_get_class(formH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Form Definition - Overview (ael)
Form Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_form_get_disp_format()

Returns the display format string of the given *form* (ael). The display format string defines how to display the *parameter value* (ael) as in the schematic or layout. Refer to *Format Strings* (ael) for more information on format string's syntax.

Syntax

```
decl dispFormatStr = dm_form_get_disp_format(formH);
```

Where,

- *formH* is a *form definition* (ael).

Example

```
decl dispFormatStr = dm_form_get_disp_format(formH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Form Definition - Overview (ael)
Form Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_form_get_label()

Returns a string descriptive label of the given *form* (ael).

Syntax

```
decl formLabel = dm_form_get_label(formH);
```

Where,

- *formH* is a *form definition* (ael).

Example

```
decl formLabel = dm_form_get_label(formH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Form Definition - Overview (ael)
Form Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_form_get_name()

Returns a string with the name of the given *form* (ael).

Syntax

```
decl formName = dm_form_get_name(formH);
```

Where,

- *formH* is a *form definition* (ael).

Example

```
decl formName = dm_form_get_name(formH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Form Definition - Overview (ael)
Form Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_form_get_net_format()

Returns the netlist format string of the given *form* (ael). The netlist format string defines how to netlist the *parameter value* (ael) as. Please refer to Format Strings for more information on format string's syntax.

Syntax

```
decl netlistFormatStr = dm_form_get_net_format(formH);
```

Where,

- *formH* is a *form definition* (ael).

Example

```
decl netlistFormatStr = dm_form_get_net_format(formH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Form Definition - Overview (ael)
Form Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_form_get_parms()

Used with compound forms. Returns the list of *parameter definitions* (ael) for the *form* (ael).

Syntax

```
decl formParmDefList = dm_form_get_parms(formH);
```

Where,

- *formH* is a *form definition* (ael).

Example

```
decl formParmDefList = dm_form_get_parms(formH);
if (formParmDefList != NULL)
{
   // Work with the compound form's Sub-Parameter Definitions.
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Form Definition - Overview (ael)
Form Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_form_is_constant_form()

Returns TRUE if the given *form* (ael) is a constant form.

Syntax

```
decl isConstantForm = dm_form_is_constant_form(formH);
```

Where,

- *formH* is a *form definition* (ael).

Example

```
decl isFormConstant = dm_form_is_constant_form(formH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Form Definition - Overview (ael)
Form Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_form_is_discrete()

Returns TRUE if the given *form* (ael) is discrete.

Syntax

```
decl isDiscrete = dm_form_is_discrete(formH);
```

Where,

- *formH* is a *form definition* (ael).

Example

```
decl isDiscrete = dm_form_is_discrete(formH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Form Definition - Overview (ael)
Form Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_get_string_form_class_selection()

Returns a list of strings pertinent for the given *form* (ael), if the given *form* (ael) is of class type **DM_STRING_CLASS**.
This is the list of strings to use and display in the Edit Component Parameter dialog box. Returns an empty list if the *form* (ael) is not of class type **DM_STRING_CLASS** or if there are no strings for the *form* (ael) to display.

Syntax

```
decl displayStringsList = dm_get_string_form_class_selection(formH);
```

Where,

- *formH* is a *form definition* (ael).

Example

```
decl displayStringsList = dm_get_string_form_class_selection(formH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Form Definition - Overview (ael)
Form Definition Functions (ael)
dm_form_get_class() (ael)

Where Used (ael)

Schematic, Layout

Form Definition - Overview

An ADS **Form Definition**, may also be just known as a **Form** within ADS. A Form manages the usage of the parameters that it holds.

The *parameter values* (ael) are defined in terms of the Forms the values can take. The characteristics of a Form govern its handling in a component's Edit Component Parameters Dialog box, on-screen editing in the schematic and layout, and netlisting. Some of the different types of Form include: string, constant forms, or more complex forms that are defined by AEL functions.

Each parameter specifies a *form set* (ael) that defines the allowable forms. Each parameter value specifies the form of that particular value. You can learn more about *form sets* (ael), here: *Form Set - Overview* (ael).

The functions *create_constant_form()* (ael), *create_compound_form()* (ael), and *create_text_form()* (ael) are used to create a form definition within ADS.

The types of information that a Form Definition AEL object stores about component parameter forms includes:

- A form name, a unique name for the form.
- A brief descriptive label for the form.
- A form attribute that can denote the type of form or type of value the parameter form holds.
- A netlist format string that defines how the *parameter's value* (ael) is netlisted.
- A display format string that defines how the *parameter's value* (ael) is displayed.
- A dialog data string.
- A form class code.
- A list of sub-parameters of the form.

The files `stdforms.ael`, `stditems.ael`, and `pde_gemini.ael` contain many of the standard form and component definitions used by the program. These files can be found within the ADS Installation `<$HPEESOF_DIR>/de/ael` directory.

Note
In ADS 2011 and newer releases, Forms and *Formsets* (ael) are defined in a library. The ADS standard Forms and *Formsets* (ael) are available for use in any library, but other Forms and *Formsets* (ael) will only be accessible from their own library.

Form information stored within a Form Definition

These are the different data fields stored within an AEL Form Definition Object.

Name

name is a string; a unique name of the form.

Label

label is a string; a short descriptive label for the form.

Attribute

attrib is an integer attribute code. To set multiple attributes, bit-wise OR their numeric equivalent values. For example, for discrete valued parts this value should be set to both FORM_DISCRETE and FORM_TUNABLE. So to set the attribute, bit-wise OR their numeric equivalent values and set the attribute code to the combined value: attrib = FORM_DISCRETE | FORM_TUNABLE.

Form Attribute Choices:

Attribute	Numeric Equivalent	Description
FORM_TUNABLE	4	
FORM_DISCRETE	64	

Netlist Format String

netlistFormat is format string to netlist the *parameter value* (ael) as. Refer to *Format Strings* (ael).

Display Format String

displayFormat is the format string to display in a schematic. Refer to *Format Strings* (ael).

Dialog Data String (Optional)

dialogDataStr is optional; is a string that can be any dialog data. For the standard Edit Component Parameters dialog box, this field can be used as the name of the table entry field. This argument indicates to the dialog the GUI component to use to breakdown the *parameter value* (ael) input. For example, the standard Edit Component dialog supports these GUI configurations:

- "StdForm"
- "StringAndReference"
- "SingleTextLine"
- "InstSelectionForm"
- "NodeSetForm"
- "FileBasedForm"
- "ReadFileForm"

Form Class Code

Form Class Code holds an integer value that represents the Form's class type. A new form can be defined using one of several classes. Each class type defines specific syntax requirements for the form. The following built in classes are characterized.

- `DM_NAME_CLASS` - **(Constant Class)** Used where the value is a constant and cannot be changed. Such as a Yes or a No value.
- `DM_NAMED_LIST_CLASS` - **(Compound Class)** Used to define a hierarchical parameter, where the *parameter value* (ael) is defined by several sub parameter values combined. For this class, sub-parameters are defined for the form using *create_parm()* (ael), each of which references a *form set* (ael). Since this form is indirectly referenced through another *parameter definition* (ael), it creates a recursive definition (only one level deep). The *parameter's value* (ael) is composed of the multiple fields set in the form's sub-parameter values.
- `DM_STRING_CLASS` - **(Text Class)** Used where a text value is required, but an arbitrary string is not appropriate. This occurs for parameters where a list of possible

values should be generated at run time and presented in a dialog box, or where additional verification must be performed on the typed response in the dialog box. Although not supported by the user interface anymore, most parameters in the program are using this class of form.

Sub-parameters

Repeated parameters and parameters using compound forms hold one or more sub-parameters, which are created with a call to *create_parm()* (ael).

Creating a new Form Definition

To create/define a new ADS form definition, the AEL functions *create_constant_form()* (ael), *create_text_form()* (ael), and *create_compound_form()* (ael) can be used.

create_constant_form()

- The forms created by *create_constant_form()* (ael) represent a fixed value, which is selected from a list of the possible values (normally from an option box) but does not edit textually.

An example of this might be the definition of the Yes form:

```
create_constant_form("Yes", "YES", 0, "yes", "yes");
```

See function: *create_constant_form()* (ael)

create_compound_form()

- The forms created by *create_compound_form()* (ael) represent values that contain one or more sub-parameters, some of which represent a value more complicated than a string. Each sub-parameter has its own set of forms for the values it can accept.

For example:

```
create_compound_form("LinearStart", "LinearStart", 0,
  "Start=%0s, Stop=%1s Step=%2s Lin=%3s",
  "Start=%0s, Stop=%1s Step=%2s Lin=%3s",
  create_parm("Start", "Start Value", 0 "FSTextForms",
    FREQUENCY_UNIT, prm("FSTextForm", "0")),
  create_parm("Stop", "Stop Value", 0 "FSTextForms",
    FREQUENCY_UNIT, prm("FSTextForm", "0")),
  create_parm("Step", "Step Value", 0 "FSTextForms",
    FREQUENCY_UNIT, prm("FSTextForm", "0")),
  create_parm("Lin", "Points in a Linear Sweep", 0 "FSTextForms",
    UNITLESS_UNIT, prm("FSTextForm", "0")));
```

See function: *create_compound_form()* (ael)

create_text_form()

- The forms created by *create_text_form()* (ael) represent values that accept a string, but not just any string will do. A dialog box to present options for the string and additional checking of the typed string can be specified. A data string can be provided for use in the option list generation and value verification.

An example might be the definition of the *SingleTextLineIntegerForm* form, used to represent a value that is an integer.

The value *stdforms_validate_integer* specifies a function that checks a value typed in by the user. If the value is an integer, the function returns 1 and the user-value is acceptable. If the value is not an integer, the function returns 0 and the user-value is not acceptable.

For example:

```
create_text_form("SingleTextLineInteger", "Integer Value",
  "SingleTextLine", 0, "%v", "%v", NULL, stdforms_validate_integer,
  NULL);
```

See function: *create_text_form()* (ael)

Accessing Form Definition Data

Getting a list of form definitions from a *Parameter Definition* (ael)

- See *dm_get_parm_definition_forms()* (ael):

```
decl formDefList = dm_get_parm_definition_forms(parmDef);
```

Getting a form definition from a *Parameter Iterator* (ael)

- See *db_param_iter_find_form()* (ael):

```
decl formDef = db_param_iter_find_form(paramIter);
```

Getting form name from a *Parameter Value* (ael)

- See *db_get_param_form_name()* (ael) to get a form name from a *parameter value* (ael).

```
decl strFormName = db_get_param_form_name (paramVal);
```

Getting form names from a *Form Set* (ael)

- See *dm_parm_get_formset_name()* (ael) to get a *formset* (ael) name from a *parameter definition* (ael).

```
decl parmFormSetName = dm_parm_get_formset_name(parmDefH);
```

- See *dm_get_form_set_form_names()* (ael) to get the form names from a *formset* (ael) with given *formset* (ael) name.

```
// Get the form names from a parameter's form set name.
 decl formNamesList = dm_get_form_set_form_names(parmFormSetName);
```

Querying Form Definition Data

A Form Definition object is useful when there is a need to query or retrieve component parameter form definition data, such as a netlist format string or display format string.

An Instance Iterator from a *DesignContext* (ael) can be used to traverse a design's instance data. The instance object and/or instance iterator can be used to retrieve a particular component instance's parameter information.

The AEL function *db_create_param_iter()* (ael) is used to retrieve the a *parameter iterator* (ael) for a particular instance within a *DesignContext* (ael). The *parameter iterator* (ael) can be used to retrieve different parameter information such as the *parameter's definition* (ael) or *parameter's value* (ael). For more information about *parameter iterators* (ael), see: *Parameter Iterator - Overview* (ael)

The AEL function *db_param_iter_get_parm_def()* (ael) can be used to get the *parameter definition* (ael) from a parameter iterator. For more information about *parameter definitions* (ael), see: *Parameter Definition - Overview* (ael).

From the *parameter definition* (ael) you can get a list of forms for a given parameter by using the function *dm_get_parm_definition_forms()* (ael).

See the list of *Form Definition Functions* (ael) for more information.

Example of Querying Form Definition Information

```
decl instIter = db_create_inst_iter(context);
for ( ; db_inst_iter_is_valid(instIter); instIter = db_inst_iter_get_next(instIter))
{
  decl paramIter = db_create_param_iter(instIter);
  for ( ; db_param_iter_is_valid(paramIter); paramIter = db_param_iter_get_next(paramIter))
  {
      decl parmDef = db_param_iter_get_parm_def(paramIter);
      decl parmFormList = dm_get_parm_definition_forms(parmDef);
      decl totalForms = listlen(parmFormList);
      decl idx;
      for (idx = 0; idx < totalForms; idx++)
      {
          decl formDefP = nth(idx, parmFormList);
          decl formName = dm_form_get_name(formDefP);
          decl formLabel = dm_form_get_label(formDefP);
          decl formNetFmtStr = dm_form_get_net_format(formDefP);
          decl formDispFmtStr = dm_form_get_disp_format(formDefP);
          decl formClassCode = dm_form_get_class(formDefP);
          …
      }
  }
}
```

List of Form Definition Functions

See the following: *Form Definition Functions* (ael)

Form Set Functions

- ***Form Set - Overview* (ael)**

This section describes the following functions:

- *dm_get_form_set_form_names()* (ael)

See also

- *Form Definition - Overview* (ael)
- *Form Definition Functions* (ael)
- *create_form_set()* (ael)

dm_get_form_set_form_names()

Returns the list of string *form* (ael) names in the given *form set* (ael).

Syntax

```
decl formNamesList = dm_get_form_set_form_names(strFormsetName);
```

Where,

- *strFormsetName* is a string that holds the unique name of an existing *form set* (ael).

Example

```
create_constant_form( "MY_YES", "YES", 0, "yes", "yes");
create_constant_form( "MY_NO", "NO", 0, "no", "no");
create_form_set( "MyYesNoFormSet", "MY_YES", "MY_NO");

decl formNamesList = dm_get_form_set_form_names("MyYesNoFormSet");
// formNamesList would have contents list("MY_YES", "MY_NO");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Form Set - Overview (ael)
Form Set Functions (ael)
Form Definition - Overview (ael)
Form Definition Functions (ael)

Where Used (ael)

Schematic, Layout

Form Set - Overview

An ADS **Form Set** may also be know as **FormSet** within ADS. A form set describes the types of values a parameter can take. When components are defined with their associated parameters, the allowable *forms* (ael) for each parameter are specified through a named form set.

A **FormSet** is composed of one or more *Forms* (ael), each representing a value option for a parameter. For more information about forms, see *Form Definition - Overview* (ael).

Several *forms* (ael) are associated together in sets for use with a parameter, where a *parameter value* (ael) can assume any of the *forms* (ael) in the set. To be valid, the *parameter definitions* (ael) must specify the name of a defined form set. In some cases, this can require the definition of a Form Set of one *Form* (ael).

For example, the Parameter Entry Mode menu in the Component Parameters dialog box is configured by the FormSet for the selected parameter. The name of the *Form* (ael) is used to configure the standard Component Parameters dialog box.

Forms (ael) and FormSets are all defined using these AEL functions:

- *create_form_set()* (ael)
- *create_text_form()* (ael)
- *create_constant_form()* (ael)
- *create_compound_form()* (ael)

Certain primitive *forms* (ael) are pre-defined within ADS pre-defined form sets and do not need to be created explicitly.

Some of the ADS pre-defined form sets are:

- StdFormSet
- StdFileFormSet
- ReadFileFormSet
- StringAndReferenceFormSet
- SingleTextLineFormSet
- SingleTextLineIntegerFormSet
- SingleTextLineDoubleFormSet

The files `stdforms.ael`, `stditems.ael`, and `pde_gemini.ael` contain many of the standard form and component definitions used by the program. These files can be found in the ADS installation `<$HPEESOF_DIR>/de/ael` directory.

Note
In ADS 2011 and newer versions, *Forms* (ael) and FormSets are defined in a library. The standard ADS *Forms* (ael) and FormSets that ship with ADS are available for use in any library, but other *Forms* (ael) and FormSets can be accessed only from their own library.

Creating a new FormSet

To create/define a new ADS FormSet, the AEL function *create_form_set()* (ael) can be used.
The following example defines a FormSet:

```
create_form_set("StdFileFormSet", "StdForm", "FileBasedForm");
```

See function: *create_form_set()* (ael).

List of Form Set Functions

See list here: *Form Set Functions* (ael)

HierarchyContext Functions

- *HierarchyContext Overview* (ael)

This section describes the following HierarchyContext functions:

- *db_create_hierarchy_context()* (ael)
- *db_is_primitive_instance_in_hierarchy()* (ael)
- *db_get_design_name_for_instance_in_hierarchy()* (ael)
- *db_get_hierarchy_context_for_instance()* (ael)
- *db_get_design_context_from_hierarchy()* (ael)
- *db_is_hierarchy_context_at_root()* (ael)
- *db_get_hierarchy_context_for_parent()* (ael)
- *db_get_parent_inst_name_in_hierarchy()* (ael)

db_create_hierarchy_context()

This function is used to create a *HierarchyContext* (ael) for a given design with a given descent policy. The given *DesignContext* (ael) is used as the root of the design hierarchy. The *HierarchyContext* (ael) is useful for traversing a given design's hierarchy.

Hierarchical Descent Policies

The hierarchical descent policy determines which sub-designs will be included in the hierarchy. The descent policy is used by the function *db_create_hierarchy_context()* (ael) to define the hierarchical descent policy for the created *HierarchyContext* (ael).

The valid hierarchical descent policies are:

- **DIRECT_DESCENDANTS** - Creating a hierarchy context with a **DIRECT_DESCENDANTS** hierarchical descent policy makes hierarchical traversal use the view that is displayed (i.e. the instance's master design) as the view that is considered the sub-design. Because there is always a view to display, there are no primitives. Note that for a schematic instance the sub design will be a symbol.
- **NETLIST_FROM_SCHEMATIC** - Creating a hierarchy context with a **NETLIST_FROM_SCHEMATIC** hierarchical descent policy makes hierarchy traversal prefer schematic views. This is the normal netlisting descent policy.
- **NETLIST_FROM_LAYOUT** - Creating a hierarchy context with a **NETLIST_FROM_LAYOUT** hierarchical descent policy makes hierarchy traversal prefer layout views.

Syntax

```
decl hierarchyContext = db_create_hierarchy_context(rootDesignContext,
descentPolicy);
```

Where,

- *rootDesignContext* is a valid *DesignContext* (ael) that represents the root of the hierarchy.
- *descentPolicy* is an integer hierarchical descent policy, such as: **DIRECT_DESCENDANTS**, **NETLIST_FROM_SCHEMATIC**, or **NETLIST_FROM_LAYOUT**.

Example

```
decl designContext = de_get_current_design_context();
// Get the hierarchy descent policy.
decl hierPolicy = NETLIST_FROM_SCHEMATIC;
if (de_is_layout_context(designContext))
    hierPolicy = NETLIST_FROM_LAYOUT;
decl hierarchyContext  = db_create_hierarchy_context(designContext, hierPolicy);
decl subDesignNameList = list();
decl instIter = db_create_inst_iter(hierarchyContext);
// Get list of instances sub-design names.
for( ; db_inst_iter_is_valid(instIter); instIter = db_inst_iter_get_next(instIter) )
{
   // Get instance's sub-design name if its a non-primitive.
   if (!db_is_primitive_instance_in_hierarchy(hierarchyContext, instIter))
   {
       decl subDesignName = db_get_design_name_for_instance_in_hierarchy(hierarchyContext,
instIter);
       subDesignNameList = append(subDesignNameList, list(subDesignName));
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

HierarchyContext Overview (ael)
HierarchyContext Functions (ael)
db_is_primitive_instance_in_hierarchy() (ael)
db_get_hierarchy_context_for_instance() (ael)
db_get_design_context_from_hierarchy() (ael)
db_get_parent_inst_name_in_hierarchy() (ael)
db_get_hierarchy_context_for_parent() (ael)
db_is_hierarchy_context_at_root() (ael)
db_get_design_name_for_instance_in_hierarchy() (ael)

Where Used (ael)

Schematic, Layout

db_get_design_context_from_hierarchy()

Returns the *DesignContext* (ael) from the given *HierarchyContext* (ael).

Syntax

```
decl designContext = db_get_design_context_from_hierarchy(hierarchyContext);
```

Where

- *hierarchyContext* is a *HierarchyContext* (ael).

Example

```
decl designContext = de_get_current_design_context();
```

```
decl hierarchyContext  = db_create_hierarchy_context (designContext, NETLIST_FROM_SCHEMATIC);
decl subDesignsList = list();
decl instIter = db_create_inst_iter(hierarchyContext);
// Get list of instances' sub-designs.
for( ; db_inst_iter_is_valid(instIter); instIter = db_inst_iter_get_next(instIter) )
{
   // Get instance's sub-design if instance is a non-primitive.
   if (!db_is_primitive_instance_in_hierarchy(hierarchyContext, instIter))
   {
       decl subDsnHierContext = db_get_hierarchy_context_for_instance (hierarchyContext,
instIter);
       decl subDesignContext = db_get_design_context_from_hierarchy(subDsnHierContext);
       subDesignsList = append(subDesignsList, list(subDesignContext));
   }
}
...
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

HierarchyContext Overview (ael)
HierarchyContext Functions (ael)
db_create_hierarchy_context() (ael)

Where Used (ael)

Schematic, Layout

db_get_design_name_for_instance_in_hierarchy()

Returns the sub-design name for a given instance in the given *HierarchyContext* (ael).

Syntax

```
decl dsnName = db_get_design_name_for_instance_in_hierarchy(hierarchyContext,
dbInst);
```

Where,

- *hierarchyContext* is a *HierarchyContext* (ael).
- *dbInst* is an instance object or instance iterator.

Example

```
decl designContext = de_get_current_design_context();
decl hierarchyContext  = db_create_hierarchy_context(designContext, NETLIST_FROM_LAYOUT);
decl subDesignNameList = list();
decl instIter = db_create_inst_iter(hierarchyContext);
// Get list of instances' sub-design names.
for( ; db_inst_iter_is_valid(instIter); instIter = db_inst_iter_get_next(instIter) )
{
   // Get instance's sub-design name if its a non-primitive.
   if (!db_is_primitive_instance_in_hierarchy(hierarchyContext, instIter))
   {
```

```
       decl subDesignName = db_get_design_name_for_instance_in_hierarchy(hierarchyContext,
instIter);
       subDesignNameList = append(subDesignNameList, list(subDesignName));
   }
}
...
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

HierarchyContext Overview (ael)
HierarchyContext Functions (ael)
db_create_hierarchy_context() (ael)
db_is_primitive_instance_in_hierarchy() (ael)
db_get_hierarchy_context_for_instance() (ael)

Where Used (ael)

Schematic, Layout

db_get_hierarchy_context_for_instance()

Returns the *HierarchyContext* (ael) for the given instance's sub-design in the given *HierarchyContext* (ael). You can use the instance's *HierarchyContext* (ael) to traverse the instances on the sub-design.

Syntax

```
decl subHierContext = db_get_hierarchy_context_for_instance(hierarchyContext,
dbInst);
```

Where,

- *hierarchyContext* is a *HierarchyContext* (ael).
- *dbInst* is an instance object or instance iterator.

Example

```
decl designContext = de_get_current_design_context();
decl hierarchyContext  = db_create_hierarchy_context(designContext, NETLIST_FROM_SCHEMATIC);
decl subDesignsList = list();
decl instIter = db_create_inst_iter(hierarchyContext);
// Get list of instances' sub-designs.
for( ; db_inst_iter_is_valid(instIter); instIter = db_inst_iter_get_next(instIter) )
{
   // Get instance's sub-design if instance is a non-primitive.
   if (!db_is_primitive_instance_in_hierarchy(hierarchyContext, instIter))
   {
       decl subDsnHierContext = db_get_hierarchy_context_for_instance(hierarchyContext, instIter);
       decl subDesignContext = db_get_design_context_from_hierarchy(subDsnHierContext);
       subDesignsList = append(subDesignsList, list(subDesignContext));
   }
}
...
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

HierarchyContext Overview (ael)
HierarchyContext Functions (ael)
db_create_hierarchy_context() (ael)
db_is_primitive_instance_in_hierarchy() (ael)

Where Used (ael)

Schematic, Layout

db_get_hierarchy_context_for_parent()

This function returns the *HierarchyContext* (ael) for the parent of the current design in the given *HierarchyContext* (ael).

Syntax

```
decl parentHierContext = db_get_hierarchy_context_for_parent(hierarchyContext);
```

Where,

- *hierarchyContext* is a *HierarchyContext* (ael).

Example

```
decl isAtRoot = db_is_hierarchy_context_at_root(hierarchyContext);
// Get design name of the parent design if HierarchyContext is not at root.
if (!isAtRoot)
{
   decl parentHierarchyContext = db_get_hierarchy_context_for_parent(hierarchyContext);
   decl parentDesignContext = db_get_design_context_from_hierarchy(parentHierarchyContext);
   decl parentDesignName = db_get_design_name(parentDesignContext);
}
...
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

HierarchyContext Overview (ael)
HierarchyContext Functions (ael)
db_create_hierarchy_context() (ael)
db_is_hierarchy_context_at_root() (ael)

Where Used (ael)

Schematic, Layout

db_get_parent_inst_name_in_hierarchy()

This function returns the instance name of the parent instance of the current design in the given *HierarchyContext* (ael).

Syntax

```
decl instName = db_get_parent_inst_name_in_hierarchy(hierarchyContext);
```

Where,

- *hierarchyContext* is a *HierarchyContext* (ael).

Example

```
decl hierarchyContext  = db_create_hierarchy_context(designContext, NETLIST_FROM_LAYOUT);

decl isAtRoot = db_is_hierarchy_context_at_root(hierarchyContext);
// Get name of parent instance name if HierarchyContext is not at root.
if (!isAtRoot)
{
  decl parentInstName = db_get_parent_inst_name_in_hierarchy(hierarchyContext);
}
...
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

HierarchyContext Overview (ael)
HierarchyContext Functions (ael)
db_create_hierarchy_context() (ael)
db_is_hierarchy_context_at_root() (ael)

Where Used (ael)

Schematic, Layout

db_is_hierarchy_context_at_root()

This function returns TRUE if the given *HierarchyContext* (ael) is at the root. Returns FALSE if the given *HierarchyContext* (ael) is not at the root.

Syntax

```
decl isAtRoot = db_is_hierarchy_context_at_root(hierarchyContext);
```

Where,

- *hierarchyContext* is a *HierarchyContext* (ael).

Example

```
decl hierarchyContext  = db_create_hierarchy_context(designContext, DIRECT_DESCENDANTS);
decl isAtRoot = db_is_hierarchy_context_at_root(hierarchyContext);
// Get name of parent instance name if HierarchyContext is not at root.
if (!isAtRoot)
{
   decl parentInstName = db_get_parent_inst_name_in_hierarchy(hierarchyContext);
}
...
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

HierarchyContext Overview (ael)
HierarchyContext Functions (ael)
db_create_hierarchy_context() (ael)

Where Used (ael)

Schematic, Layout

db_is_primitive_instance_in_hierarchy()

Returns TRUE if the instance is a primitive in the given *HierarchyContext* (ael). Returns FALSE if the instance is not a primitive in the given *HierarchyContext* (ael).

Syntax

```
decl isPrimitive = db_is_primitive_instance_in_hierarchy(hierarchyContext,
dbInst);
```

Where,

- *hierarchyContext* is a *HierarchyContext* (ael).
- *dbInst* is an instance object or instance iterator.

Example

```
decl designContext = de_get_current_design_context();
decl hierarchyContext  = db_create_hierarchy_context (designContext, NETLIST_FROM_SCHEMATIC);
decl subDesignNameList = list();
decl instIter = db_create_inst_iter(hierarchyContext);
// Get list of instances' sub-design names.
for( ; db_inst_iter_is_valid(instIter); instIter = db_inst_iter_get_next(instIter) )
{
   // Get instance's sub-design name if its a non-primitive.
   if (!db_is_primitive_instance_in_hierarchy(hierarchyContext, instIter))
   {
       decl subDesignName = db_get_design_name_for_instance_in_hierarchy(hierarchyContext,
instIter);
```

```
        subDesignNameList = append(subDesignNameList, list(subDesignName));
    }
}
...
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

HierarchyContext Overview (ael)
HierarchyContext Functions (ael)
db_create_hierarchy_context() (ael)

Where Used (ael)

Schematic, Layout

HierarchyContext - Overview

A HierarchyContext is an AEL object that holds information about a particular design's hierarchy. It allows you to write hierarchy traversal functions in a generic way, where the same API or set of HierarchyContext based functions can be used with many different HierarchyContexts. In this way, the same piece of code may work on a number of different hierarchical data, if the HierarchyContext is used as a parameter.

A HierarchyContext is created from a *DesignContext* (ael) and a Descent policy. The function *db_create_hierarchy_context()* (ael) is used to create a HierarchyContext.

Hierarchical Descent Policy

The hierarchical descent policy determines which sub-designs will be included in the hierarchy. The descent policy is used by the function *db_create_hierarchy_context()* (ael) to define the hierarchical descent policy for the created HierarchyContext.

The Available Descent Policies are:

- **AEL Constant: DIRECT_DESCENDANTS** - Creating a hierarchy context with a **DIRECT_DESCENDANTS** hierarchical descent policy makes hierarchical traversal use the view that is displayed (i.e. the instance's master design) as the view that is considered the sub-design. Because there is always a view to display, there are no primitives. Note that for a schematic instance the sub design will be a symbol.
- **AEL Constant: NETLIST_FROM_SCHEMATIC** - Creating a hierarchy context with a **NETLIST_FROM_SCHEMATIC** hierarchical descent policy makes hierarchy traversal prefer schematic views. This is the normal netlisting descent policy.
- **AEL Constant: NETLIST_FROM_LAYOUT** - Creating a hierarchy context with a **NETLIST_FROM_LAYOUT** hierarchical descent policy makes hierarchy traversal prefer layout views.

Database traversal using a HierarchyContext

To traverse a design hierarchy use a HierarchyContext. An instance iterator is used to traverse the instances of a design. See the function *db_create_inst_iter()* (ael) on how to create an instance iterator.

When traversing the instances of a design, you can test if an instance is a primitive or not with the function *db_is_primitive_instance_in_hierarchy()* (ael).

You can get the hierarchy context for an instance's sub-design by calling *db_get_hierarchy_context_for_instance()* (ael). You can then traverse the instances on the sub-design hierarchy.

You can test if a HierarchyContext is at the root of a hierarchy by calling *db_is_hierarchy_context_at_root()* (ael). If the HierarchyContext is not at the root, you can get the HierarchyContext for the parent design by calling *db_get_hierarchy_context_for_parent()* (ael).

Example use of HierarchyContext AEL objects

The following example AEL code will traverse a design's hierarchy.

```
// This example shows how to traverse the instances of a design's hierarchy.
// Get current design.
decl designContext = de_get_current_design_context();
// Get the hierarchy descent policy.
decl hierPolicy = NETLIST_FROM_SCHEMATIC;
if (de_is_layout_context(designContext))
   hierPolicy = NETLIST_FROM_LAYOUT;

// Get hierarchical context of the design.
decl topHierarchyContext = db_create_hierarchy_context(designContext, hierPolicy);

// Setup the design hierarchy queue list.
decl hierList = list(topHierarchyContext);

// Level order traversal of a design hierarchy.
while (listlen(hierList) > 0)
{
   // Take first design hierarchy off the queue list.
   decl hierContext = car(hierList);
   hierList = cdr(hierList);

   if (hierContext != NULL)
   {
      // Visit the design in the design hierarchy.
      fputs(stderr, strcat("Traversing design: ",
          identify_value(db_get_design_name(hierContext)), "\n"));
      // Traverse the instances in the design.
      decl instIter = db_create_inst_iter(hierContext);
      for ( ; db_inst_iter_is_valid(instIter);
          instIter = db_inst_iter_get_next(instIter))
      {
          // Visit the instance in the design.
          fputs(stderr, strcat("\t Visiting instance: ",
              identify_value(db_get_instance_name(instIter)), "\n"));

          decl isPrimitive = db_is_primitive_instance_in_hierarchy(hierContext, instIter);
          // If instance is primitive then no traversal of that
          // instance's hierarchy is required.
          if (isPrimitive)
              continue;

          // Queue that instance's hierarchy for traversal.
          decl subHierContext =
              db_get_hierarchy_context_for_instance(hierContext, instIter);
          if (subHierContext != NULL)
          {
              // Queue instance's hierarchy for traversal.
              hierList = append(hierList, list(subHierContext));
          }
      }
   }
}
...
```

HierarchyContext AEL Functions

See AEL *HierarchyContext Functions* (ael).

Version Introduced

ADS 2011

Highlight/Selection Functions

This section describes each highlight/selection function in detail. The functions are listed in alphabetical order.

db_highlight_net()

This function highlights or unhighlights a given *Net* (ael) within a given *design context* (ael).

Syntax

```
decl bOk = db_highlight_net(context, net [,highlight]);
```

Where,

- *context* is a *design context* (ael) that contains the *net* (ael) to highlight.
- *net* is a *Net* (ael) to highlight.
- *highlight* is an optional boolean argument, if TRUE highlights the net, if FALSE unhighlights the net.
  If the *highlight* argument is not provided, the default is TRUE.

Example

```
decl bOk = db_highlight_net(context, dbNet, TRUE);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Highlight Selection Functions (ael)
Net Overview (ael)
Net Functions (ael)
Net Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_highlight()

Function to highlight or unhighlight a given shape, instance, or pin.

See also: *db_is_highlighted()* (ael).

Syntax:

```
db_highlight(object[, highlight]);
```

where

object is an object or iterator of an instance, pin, or shape.
highlight is an optional boolean parameter where if TRUE will highlight the object, FALSE will unhighlight the object.
If the boolean highlight parameter is not given, then the object will be highlighted.

Example:

```
decl context = de_get_current_design_context();
decl instIter = db_create_inst_iter(context);
// Highlight any instances in current design context.
for ( ; db_inst_iter_is_valid(instIter); instIter = db_inst_iter_get_next(instIter))
{
        db_highlight(instIter);
}
// Unhighlight all shapes in the current design context.
decl shapeIter = db_create_shape_iter(context);
for ( ; db_shape_iter_is_valid(shapeIter); shapeIter = db_shape_iter_get_next(shapeIter))
   db_highlight(shapeIter, FALSE);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_is_highlighted()

Returns TRUE if the given instance, pin, or shape is highlighted. Otherwise it returns FALSE if the given instance, pin, or shape is not highlighted.

See also: *db_highlight()* (ael).

Syntax:

```
db_is_highlighted(object);
```

where

object is an iterator or an object of an instance, pin, or shape.

Example:

```
decl context = de_get_current_design_context();
decl instIter = db_create_inst_iter(context);
// Code to unhighlight all currently highlighted instances in current design context.
for (; db_inst_iter_is_valid(instIter); instIter = db_inst_iter_get_next(instIter))
{
    if (db_is_highlighted(instIter) == TRUE)
        db_highlight(instIter, FALSE);
}
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_is_selected()

Returns TRUE if the passed in instance, symbol port, or shape is selected. Otherwise it returns FALSE if the passed in instance, symbol port, or shape is not selected. Shapes are considered selected if either the entire shape is selected or if points on the shape are selected.

See also: *db_select()* (ael).

Syntax:

```
db_is_selected(object);
```

where

object is an iterator or an object of an instance, symbol port, or shape.

Example:

```
decl context = de_get_current_design_context();
decl instIter = db_create_inst_iter(context);
// Code to deselect all currently selected instances in current design context.
for (; db_inst_iter_is_valid(instIter); instIter = db_inst_iter_get_next(instIter))
{
    if (db_is_selected(instIter) == TRUE)
        db_select(instIter, FALSE);
}
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_select()

Function to select or deselect an instance, symbol port, or shape. If applicable, the design will automatically be redisplayed in all windows.

See also: *db_is_selected()* (ael).

Syntax:

```
db_select(object[, select]);
```

where

object is an object or iterator of an instance, symbol port, or shape.

select is an optional parameter where if TRUE will select the object, FALSE will deselect the object.
If the select parameter is not given, then the object is selected.

Example:

```
decl context = de_get_current_design_context();
decl instIter = db_create_inst_iter(context);
// Deselect any selected instances in current design context.
for (; db_inst_iter_is_valid(instIter); instIter = db_inst_iter_get_next(instIter))
{
    if (db_is_selected(instIter) == TRUE)
        db_select(instIter, FALSE);
}
// Deselect any selected shapes in the current design context.
decl shapeIter = db_create_shape_iter(context);
shapeIter = db_shape_iter_limit_selected(shapeIter);
for ( ; db_shape_iter_is_valid(shapeIter); shapeIter = db_shape_iter_get_next(shapeIter))
   db_select(shapeIter, FALSE);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

Instance Functions

This section describes the following Instance functions:

- *db_is_instance_deactivated()* (ael)
- *db_is_instance_deactivated_and_shorted()* (ael)
- *db_is_instance_ground()* (ael)
- *db_get_instance_cell_name()* (ael)
- *db_get_instance_library_name()* (ael)
- *db_get_instance_description()* (ael)
- *db_get_instance_design_name()* (ael)
- *db_get_instance_owner_design()* (ael)
- *db_get_instance_placement_transform()* (ael)
- *db_get_instance_special()* (ael)
- *db_get_instance_subcontext_transform()* (ael)

db_get_instance_cell_name()

Returns the string cell name of the given instance.

Syntax

```
decl cellName = db_get_instance_cell_name(dbInst);
```

Where

- *dbInst* is an instance or instance iterator.

Example

```
decl cellName;
decl dbInst = db_find_instance_ex(context, instName);
if (dbInst != NULL)
   cellName = db_get_instance_cell_name(dbInst);
if (cellName == “mycomp”)
{
    // If one of my “mycomp” cell instances, return its bounding box.
    return db_get_instance_bbox(dbInst, INST_SYMBOL_AND_ANNOT_BBOX );
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Instance Functions (ael)
db_get_instance_component_name() (ael)
db_get_instance_design_name() (ael)
db_get_instance_library_name() (ael)
db_get_instance_name() (ael)

Where Used (ael)

Schematic, Layout

db_get_instance_description()

Returns the item description string for the given instance. Returns NULL if no *item definition* (ael) exists for the given instance.

Syntax

```
decl description = db_get_instance_description(dbInst);
```

Where

- *dbInst* is an instance or instance iterator.

Example

```
decl desc;
decl dbInst = db_find_instance_ex(context, instName);
if (dbInst != NULL)
   desc = db_get_instance_description(dbInst);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions.

See also

Instance Functions (ael)
db_get_instance_component_name() (ael)
db_get_instance_name() (ael)
db_get_instance_special() (ael)

Where Used (ael)

Schematic, Layout

db_get_instance_design_name()

Returns the string design name of the instance's master design. The design name string value returned from this function is in the format "<library name>:<cell name>:<view name>".

Syntax

```
decl instMasterDesignName = db_get_instance_design_name(dbInst);
```

Where

- *dbInst* is an instance or instance iterator.

Example

```
decl instMasterDesignName = db_get_instance_design_name(dbInst);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions.

See also

Instance Functions (ael)
db_get_instance_cell_name() (ael)
db_get_instance_component_name() (ael)
db_get_instance_library_name() (ael)
db_get_instance_name() (ael)

Where Used (ael)

Schematic, Layout

db_get_instance_library_name()

Returns the string library name of the given instance.

Syntax

```
decl libName = db_get_instance_library_name(dbInst);
```

Where

- *dbInst* is an instance or instance iterator.

Example

```
decl dbInst = db_find_instance_ex(context, “X1”);
if (dbInst != NULL)
{
   decl instCellname = db_get_instance_cell_name(dbInst);
   decl instLibName = db_get_instance_library_name(dbInst);
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions.

See also

Instance Functions (ael)
db_get_instance_cell_name() (ael)
db_get_instance_component_name() (ael)
db_get_instance_design_name() (ael)
db_get_instance_name() (ael)

Where Used (ael)

Schematic, Layout

db_get_instance_owner_design()

Returns a design context to the instance's owner design. For a PSN, this is the master design.

Syntax

```
decl context = db_get_instance_owner_design(dbInst);
```

Where

- *dbInst* is an instance or instance iterator.

Example

```
decl ownerContext = db_get_instance_owner_design(dbInst);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions.

See also

Instance Functions (ael)
db_get_instance_cell_name() (ael)
db_get_instance_component_name() (ael)
db_get_instance_design_name() (ael)
db_get_instance_library_name() (ael)
db_get_instance_name() (ael)

Where Used (ael)

Schematic, Layout

db_get_instance_placement_transform()

Returns the transform of the given instance.

Syntax

```
decl transform = db_get_instance_placement_transform(dbInst);
```

Where

- *dbInst* is an instance or instance iterator.

Example

```
decl transform = db_get_instance_placement_transform( dbInst );
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions.

See also

Instance Functions (ael)
db_get_instance_subcontext_transform() (ael)

Where Used (ael)

Schematic, Layout

db_get_instance_special()

Returns the special flag code of an instance.

Syntax

```
decl special = db_get_instance_special(dbInst);
```

Where

- *dbInst* is an instance or instance iterator.

Example

```
// Iterate over instances and get to find the first VAR component instance.
decl instIter = db_create_inst_iter(context);
decl varInstH = NULL;
for (; db_inst_iter_is_valid(instIter);
     instIter = db_inst_iter_get_next(instIter) )
{
   decl instSpecial = db_get_instance_special(instIter);
   if ( !(instSpecial & INST_VARIABLE) )  // Continue if not VAR component.
       continue;

   varInstH = db_inst_iter_get_instance(instIter);
   break;
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Instance Functions (ael)
db_get_instance_component_name() (ael)
db_get_instance_description() (ael)
db_get_instance_name() (ael)
db_is_instance_ground() (ael)

Where Used (ael)

Schematic, Layout

db_get_instance_subcontext_transform()

Returns the subcontext transform of the given instance. This is the transform that can be used to change the shapes inside the instance to the parent design's coordinates. Most commonly used with DIRECT_DESCENDANTS hierarchy to draw, export, and/or examine shapes inside an instance.

Syntax

```
decl transform = db_get_instance_subcontext_transform(dbInst, row, col);
```

Where

- *dbInst* is an instance or instance iterator.
- *row* , *col* are the row and column in an arrayed instance.
  If you know you don't have an arrayed instance or want the "main" instance's transform,
  then pass 0 (zero), 0 (zero).

Example

```
decl transform = db_get_instance_subcontext_transform( dbInst, 0, 1);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions.

See also

Instance Functions (ael)
db_get_instance_placement_transform() (ael)
HierarchyContext Functions (ael)

Where Used (ael)

Schematic, Layout

db_is_instance_deactivated_and_shorted()

Returns TRUE if the given instance is deactivated and shorted.

Syntax

```
decl isShorted = db_is_instance_deactivated_and_shorted(dbInstance);
```

Where,

- *dbInstance* is an instance object or instance iterator.

Example

```
// Select only instances that are shorted.
decl context = de_get_current_design_context();
decl iter = db_create_inst_iter(context);
for (; db_inst_iter_is_valid(iter); iter = db_inst_iter_get_next(iter))
{
   // Select only the shorted instances.
   if (db_is_instance_deactivated_and_shorted(iter))
       db_select(iter);
}
...
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions.

See also

Instance Functions (ael)
db_is_instance_deactivated() (ael)

Where Used (ael)

Schematic, Layout

db_is_instance_deactivated()

Returns TRUE if instance is deactivated. Use to determine if an instance is deactivated.

Syntax

```
decl isDeactivated = db_is_instance_deactivated(dbInstance);
```

Where,

- *dbInstance* is an instance object or instance iterator.

Example

```
// Deselect only deactivated selected instances.
decl context = de_get_current_design_context();
decl iter = db_create_inst_iter(context);
iter = db_inst_iter_limit_selected(context);
for (; db_inst_iter_is_valid(iter); iter = db_inst_iter_get_next(iter))
{
   // Deselect only the deactivated instances.
   if (db_is_instance_deactivated(iter))
      db_select(iter, FALSE);
}
...
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions.

See also

Instance Functions (ael)
db_is_instance_deactivated_and_shorted() (ael)

Where Used (ael)

Schematic, Layout

db_is_instance_ground()

Returns TRUE if the given instance is a ground component. Returns FALSE if the given instance is not a ground component.

Syntax

```
decl isGround = db_is_instance_ground(dbInst);
```

Where

- *dbInst* is an instance or instance iterator.

Example

```
// Iterate over all the non ground instances
decl instIter = db_create_inst_iter(context);
for (; db_inst_iter_is_valid(instIter);
     instIter = db_inst_iter_get_next(instIter) )
{
  if (db_is_instance_ground(instIter))
      continue;
  …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions.

See also

Instance Functions (ael)
db_get_instance_special() (ael)

Where Used (ael)

Schematic, Layout

Instance Iterator Functions

This section describes the following instance iterator functions:

db_create_inst_iter() (ael)
db_inst_iter_enable_caching() (ael)
db_inst_iter_exclude_port_insts() (ael)
db_inst_iter_get_instance() (ael)
db_inst_iter_get_next() (ael)
db_inst_iter_is_valid() (ael)
db_inst_iter_limit_region() (ael)
db_inst_iter_limit_selected() (ael)

db_inst_iter_exclude_port_insts()

This function limits instance iteration to exclude port instances for a given instance iterator. If the iterator has been started (used to find an instance), then calling this function gives an AEL error.

Notes

- This function is a no-op in ADS 2011 and newer releases, because ports are represented only as pins not port instances in ADS 2011.
- In ADS 2009 Update 1 and earlier releases ports are represented by special types of instances. In ADS 2011 and newer releases, ports are represented as only pins.

Syntax

```
decl iter = db_inst_iter_exclude_port_insts(iter);
```

where,

- *iter* is an instance iterator returned from a function such as *db_create_inst_iter()* (ael).

Example

```
decl context = de_get_current_design_context();
// Highlight all the non-port instances in the design.
decl iter = db_create_inst_iter(context);
iter = db_inst_iter_exclude_port_insts(iter);
for( ; db_inst_iter_is_valid(iter); iter = db_inst_iter_get_next(iter))
   db_highlight(iter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions.

Note
This function has no use in ADS 2011 and newer versions.

See also

Instance Iterator Functions (ael)

***Where Used* (ael)**

Schematic, Layout

db_create_inst_iter()

Returns an iterator to the first instance belonging to a given *design context* (ael). If no instance is available, then the iterator returned will be invalid. The function *db_inst_iter_is_valid()* (ael) can be used to test if an iterator is valid.

See also: *db_inst_iter_enable_caching()* (ael), *db_inst_iter_get_instance()* (ael), *db_inst_iter_get_next()* (ael), *db_inst_iter_limit_region()* (ael), *db_inst_iter_limit_selected()* (ael), *db_inst_iter_is_valid()* (ael), *db_is_selected()* (ael), *db_select()* (ael).

Syntax:

```
db_create_inst_iter(context);
```

where
context is a DesignContext returned from a function such as *de_get_current_design_context()*.

Example:

```
decl context = de_get_current_design_context();
decl instIter = db_create_inst_iter(context);
// Select the first instance in the DesignContext.
if (db_inst_iter_is_valid(instIter))
   db_select(instIter);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_inst_iter_enable_caching()

Function to make the iterator cache its list.
This function is used when adding or removing instances during iteration. Use of the function causes a snapshot of the iterator's list to be taken when the iterator is first used to get an instance, thus preventing interference between the iterator and operations which modify the design.
If the iterator has been started (used to find an inst), then calling this function gives an AEL error.

See also: *db_create_inst_iter()* (ael).

Syntax:

```
db_inst_iter_enable_caching(iter);
```

where
iter is an instance iterator returned from a function such as

db_create_inst_iter().

Example:

```
//Example of how to delete instances using traversal with instance iterators.
decl context = de_get_current_design_context();
decl iter = db_create_inst_iter(context);
//Use instance iterators based off a current snapshot of instances
// in the DesignContext's design data.
iter = db_inst_iter_enable_caching(iter);
for ( ; db_inst_iter_is_valid(iter); iter = db_inst_iter_get_next(iter))
{
de_deselect_all();
db_select(iter);
de_delete(); // Delete selected instance.
}
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_inst_iter_get_instance()

Returns an instance object that is referenced by a given instance iterator. Returns NULL if no valid instance object is being referenced by the given instance iterator.
This function is not normally needed since an instance iterator can be passed directly to a function expecting an instance. The most common use is to build a list of instances that will be saved.

See also: *db_create_inst_iter()* (ael).

Syntax:

```
db_inst_iter_get_instance(iter);
```

where
iter is an instance iterator returned from a function such as *db_inst_iter_get_next()*.

Example:

```
decl context = de_get_current_design_context();
decl iter = db_create_inst_iter(context);
// Build a list of instances that will be used.
decl instList = list();
for ( ; db_inst_iter_is_valid(iter); iter = db_inst_iter_get_next(iter))
   instList = append(instList, list(db_inst_iter_get_instance(iter)));
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_inst_iter_get_next()

Returns an iterator to the next instance after the given instance iterator. If there is no next instance iterator available, an invalid instance iterator is returned. The function *db_inst_iter_is_valid()* (ael) can be used to test if an iterator is valid.

See also: *db_create_inst_iter()* (ael), *db_inst_iter_is_valid()* (ael).

Syntax:

```
db_inst_iter_get_next(iter);
```

where
iter is an instance iterator returned from a function such as *db_create_inst_iter()*.

Example:

```
decl context = de_get_current_design_context();
// Deselect any selected instances in the current design context.
decl instIter = db_create_inst_iter(context);
instIter = db_inst_iter_limit_selected(instIter);
for ( ; db_inst_iter_is_valid(instIter); instIter = db_inst_iter_get_next(instIter))
   db_select(instIter, FALSE);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_inst_iter_is_valid()

Returns TRUE if the instance iterator references a valid instance object.
Returns FALSE if the instance iterator references no valid instance object.

See also: *db_create_inst_iter()* (ael).

Syntax:

```
db_inst_iter_is_valid(iter);
```

where
iter is an instance iterator returned from a function such as *db_create_inst_iter()*.

Example:

```
decl context = de_get_current_design_context();
decl instIter = db_create_inst_iter(context);
// Select the first instance of the design context,
// if the design context returned a valid instance iterator.
if (db_inst_iter_is_valid(instIter) == TRUE)
   db_select(instIter);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_inst_iter_limit_region()

Function to limit the instance iteration to a region.
If the iterator has been started (used to find an inst), then calling this function gives an AEL error.

See also: *db_create_inst_iter()* (ael), *db_inst_iter_get_next()* (ael), *db_inst_iter_limit_selected()* (ael).

Syntax:

```
db_inst_iter_limit_region(instIter, x1, y1, x2, y2 [,allowIntersect]);
```

where
iter is an instance iterator returned from a function such as *db_create_inst_iter()*.
x1,*y1*,*x2*,*y2* are coordinates in database units to limit the iteration to.
allowIntersect is optional. By default is true, meaning that the region only has to touch the instance, not contain it.

Example:

```
//Example of how to deselect all instances in a particular drawing region.
decl context = de_get_current_design_context();
decl iter = db_create_inst_iter(context);
iter = db_inst_iter_limit_region(iter, 0, 0, 200, 200);
for ( ; db_inst_iter_is_valid(iter); iter = db_inst_iter_get_next(iter))
   db_select(iter, FALSE);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_inst_iter_limit_selected()

Function to limit the iteration to selected instances.
If the iterator has been started (used to find an inst), then calling this function gives an AEL error.

See also: *db_create_inst_iter()* (ael), *db_inst_iter_get_next()* (ael), *db_inst_iter_limit_region()* (ael).

Syntax:

```
db_inst_iter_limit_selected(iter);
```

where
iter is an instance iterator returned from a function such as

db_create_inst_iter().

Example:

```
//Example of how to deselect all instances:
decl context = de_get_current_design_context();
decl iter = db_create_inst_iter(context);
iter = db_inst_iter_limit_selected(iter);
for ( ; db_inst_iter_is_valid(iter); iter = db_inst_iter_get_next(iter))
   db_select(iter, FALSE);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

InstPin Functions

- *InstPin Overview* (ael)
- *Connectivity Objects Overview* (ael)

This section describes the following functions:

- *db_get_inst_pin_angle_normalized()* (ael)
- *db_get_inst_pin_bbox()* (ael)
- *db_get_inst_pin_instance()* (ael)
- *db_get_inst_pin_inst_term()* (ael)
- *db_get_inst_pin_net()* (ael)
- *db_get_inst_pin_snap_layerid()* (ael)
- *db_get_inst_pin_snap_point()* (ael)
- *db_get_inst_pin_term_number()* (ael)
- *db_get_inst_pin_term_type()* (ael)
- *db_is_inst_pin_connected_to_wire()* (ael)

See also

InstPin Iterator Functions (ael)

db_get_inst_pin_angle_normalized()

Returns the angle of a given *InstPin* (ael), normalized (zero degrees is right). The returned angle units are degrees*1000, so a 90 degree pin returns 90000.

Note
For normalized angles: Pins on the right side of a bbox ("on the right") are angle zero.

Syntax

```
decl angle = db_get_inst_pin_angle_normalized(dbInstPin);
```

Where,

- *dbPin* is an *InstPin* (ael) object or an *InstPin* (ael) iterator.

Example

```
decl instPinIter = db_create_inst_pin_iter(dbNet);
for ( ; db_inst_pin_iter_is_valid(instPinIter);
   instPinIter = db_inst_pin_iter_get_next(instPinIter))
{
   // Get the normalized angle of the instance pin.
   decl angle = db_get_inst_pin_angle_normalized(instPinIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstPin Overview (ael)
InstPin Functions (ael)
InstPin Iterator Functions (ael)
db_get_inst_pin_bbox() (ael)
db_get_inst_pin_snap_layerid() (ael)
db_get_inst_pin_snap_point() (ael)

Where Used (ael)

Schematic, Layout

db_get_inst_pin_bbox()

Returns the bounding box of the given *instance pin* (ael).

Syntax

```
decl bbox = db_get_inst_pin_bbox(dbInstPin);
```

Where,

- *dbInstPin* is an *InstPin* (ael) object or an *InstPin* (ael) iterator.

Example

```
decl instPinIter = db_create_inst_pin_iter(dbNet);
for (; db_inst_pin_iter_is_valid(instPinIter);
   instPinIter = db_inst_pin_iter_get_next(instPinIter))
{
   // Get the bounding box of the instance pin.
   decl bboxH = db_get_inst_pin_bbox(instPinIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstPin Overview (ael)
InstPin Functions (ael)
InstPin Iterator Functions (ael)
db_get_inst_pin_instance() (ael)
db_get_inst_pin_inst_term() (ael)
db_get_inst_pin_net() (ael)
db_get_inst_pin_snap_layerid() (ael)
db_get_inst_pin_snap_point() (ael)
db_get_inst_pin_term_number() (ael)
db_get_inst_pin_term_type() (ael)
db_is_inst_pin_connected_to_wire() (ael)

Where Used (ael)

Schematic, Layout

db_get_inst_pin_inst_term()

Returns the *InstTerm* (ael) (*instance terminal* (ael)) of the given *InstPin* (ael) (*instance pin* (ael)) object.

Syntax

```
decl dbInstTerm = db_get_inst_pin_inst_term(dbInstPin);
```

Where,

- *dbInstPin* is an *InstPin* (ael) object an *InstPin* (ael) iterator.

Example

```
decl instPinIter = db_create_inst_pin_iter(dbNet);
for (; db_inst_pin_iter_is_valid(instPinIter);
   instPinIter = db_inst_pin_iter_get_next(instPinIter))
{
   decl dbInstTerm = db_get_inst_pin_inst_term(instPinIter);
   decl termName = db_get_inst_term_name(dbInstTerm);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstPin Overview (ael)
InstPin Functions (ael)
InstPin Iterator Functions (ael)
db_get_inst_pin_bbox() (ael)
db_get_inst_pin_instance() (ael)
db_get_inst_pin_net() (ael)
db_get_inst_pin_snap_layerid() (ael)
db_get_inst_pin_snap_point() (ael)
db_get_inst_pin_term_number() (ael)
db_get_inst_pin_term_type() (ael)
db_is_inst_pin_connected_to_wire() (ael)
InstTerm Overview (ael)
InstTerm Functions (ael)

Where Used (ael)

Schematic, Layout

db_get_inst_pin_instance()

Returns the instance of the given *instance pin* (ael).

Syntax

```
decl dbInst = db_get_inst_pin_instance(dbInstPin);
```

Where,

- *dbInstPin* is an *InstPin* (ael) object or an *InstPin* (ael) iterator.

Example

```
decl instPinIter = db_create_inst_pin_iter(dbNet);
for (; db_inst_pin_iter_is_valid(instPinIter);
   instPinIter = db_inst_pin_iter_get_next(instPinIter))
{
   // Get the instance of the instance pin.
   decl instH = db_get_inst_pin_instance(instPinIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstPin Overview (ael)
InstPin Functions (ael)
InstPin Iterator Functions (ael)
db_get_inst_pin_bbox() (ael)
db_get_inst_pin_inst_term() (ael)
db_get_inst_pin_net() (ael)
db_get_inst_pin_snap_layerid() (ael)
db_get_inst_pin_snap_point() (ael)
db_get_inst_pin_term_number() (ael)
db_get_inst_pin_term_type() (ael)
db_is_inst_pin_connected_to_wire() (ael)

Where Used (ael)

Schematic, Layout

db_get_inst_pin_net()

Returns the *Net* (ael) connected to the given *instance pin* (ael). This function will return NULL if the *instance pin* (ael) is not connected to a *Net* (ael).

Syntax

```
decl dbNet = db_get_inst_pin_net(dbInstPin);
```

Where,

- *dbInstPin* is an *InstPin* (ael) object or an *InstPin* (ael) iterator.

Example

```
decl instPinIter = db_create_inst_pin_iter(dbNet);
for (; db_inst_pin_iter_is_valid(instPinIter);
   instPinIter = db_inst_pin_iter_get_next(instPinIter))
{
   // Get the Net of the instance pin.
   decl dbNet = db_get_inst_pin_net(instPinIter);
   // Use only instance pins that are connected to a Net.
   if (!dbNet)
       continue;
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstPin Overview (ael)
InstPin Functions (ael)
InstPin Iterator Functions (ael)
db_get_inst_pin_bbox() (ael)
db_get_inst_pin_instance() (ael)
db_get_inst_pin_inst_term() (ael)
db_get_inst_pin_snap_layerid() (ael)
db_get_inst_pin_snap_point() (ael)
db_get_inst_pin_term_number() (ael)
db_get_inst_pin_term_type() (ael)
db_is_inst_pin_connected_to_wire() (ael)
Net Overview (ael)
Net Functions (ael)

Where Used (ael)

Schematic, Layout

db_get_inst_pin_snap_layerid()

Returns the *LayerId* (ael) for the snap layer of an *instance pin* (ael). The snap layer of an *instance pin* (ael) is the layer of the primary shape (dot) if one exists, otherwise the layer of one of the shapes representing the *instance pin* (ael) is used.

Syntax

```
decl layerId = db_get_inst_pin_snap_layerid(dbInstPin);
```

Where,

- *dbInstPin* is an *InstPin* (ael) object or an *InstPin* (ael) iterator.

Example

```
decl context = de_get_current_design_context();
decl instPinIter = db_create_inst_pin_iter(context);
for (; db_inst_pin_iter_is_valid(instPinIter);
   instPinIter = db_inst_pin_iter_get_next(instPinIter))
{
   decl layerId = db_get_inst_pin_snap_layerid(instPinIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstPin Overview (ael)
InstPin Functions (ael)
InstPin Iterator Functions (ael)
db_get_inst_pin_bbox() (ael)
db_get_inst_pin_instance() (ael)
db_get_inst_pin_inst_term() (ael)
db_get_inst_pin_net() (ael)
db_get_inst_pin_snap_point() (ael)
db_get_inst_pin_term_number() (ael)
db_get_inst_pin_term_type() (ael)
db_is_inst_pin_connected_to_wire() (ael)
LayerId Overview (ael)
Layer and LayerId Functions (ael)

Where Used (ael)

Schematic, Layout

db_get_inst_pin_snap_point()

Returns the snap coordinate point of a given *instance pin* (ael) object.

Syntax

```
decl snapCoordH = db_get_inst_pin_snap_point(dbInstPin);
```

Where,

- *dbInstPin* is an *InstPin* (ael) object or an *InstPin* (ael) iterator.

Example

```
decl context = de_get_current_design_context();
decl instPinIter = db_create_inst_pin_iter(context);
for (; db_inst_pin_iter_is_valid(instPinIter);
   instPinIter = db_inst_pin_iter_get_next(instPinIter))
```

```
{
   decl snapCoordH = db_get_inst_pin_snap_point(instPinIter);
   decl snapX = db_get_x(snapCoordH);
   decl snapY = db_get_y(snapCoordH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstPin Overview (ael)
InstPin Functions (ael)
InstPin Iterator Functions (ael)
db_get_inst_pin_bbox() (ael)
db_get_inst_pin_instance() (ael)
db_get_inst_pin_inst_term() (ael)
db_get_inst_pin_net() (ael)
db_get_inst_pin_snap_layerid() (ael)
db_get_inst_pin_term_number() (ael)
db_get_inst_pin_term_type() (ael)
db_is_inst_pin_connected_to_wire() (ael)
LayerId Overview (ael)
Layer and LayerId Functions (ael)

Where Used (ael)

Schematic, Layout

db_get_inst_pin_term_number()

Returns the term number of a given *instance pin* (ael).

Syntax

```
decl termNumber = db_get_inst_pin_term_number(dbInstPin);
```

Where,

- *dbInstPin* is an *InstPin* (ael) object or an *InstPin* (ael) iterator.

Example

```
decl instPinIter = db_create_inst_pin_iter(context);
for (; db_inst_pin_iter_is_valid(instPinIter);
   instPinIter = db_inst_pin_iter_get_next(instPinIter))
{
   decl termNum = db_get_inst_pin_term_number(instPinIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstPin Overview (ael)
InstPin Functions (ael)
InstPin Iterator Functions (ael)
db_get_inst_pin_bbox() (ael)
db_get_inst_pin_instance() (ael)
db_get_inst_pin_inst_term() (ael)
db_get_inst_pin_net() (ael)
db_get_inst_pin_snap_layerid() (ael)
db_get_inst_pin_snap_point() (ael)
db_get_inst_pin_term_type() (ael)
db_is_inst_pin_connected_to_wire() (ael)

Where Used (ael)

Schematic, Layout

db_get_inst_pin_term_type()

Returns an integer terminal type of a given *instance pin* (ael) object. The terminal type describes the direction of the *instance pin* (ael).

The possible returned integer *terminal* (ael) types are :

- **INPUT_PIN** = 0, represents an input *instance terminal* (ael).
- **OUTPUT_PIN** = 1, represents an output *instance terminal* (ael).
- **IN_OUT_PIN** = 2, represents an input-output *instance terminal* (ael).

Syntax

```
decl termType = db_get_inst_pin_term_type(dbInstPin);
```

Where,

- *dbInstPin* is an *InstPin* (ael) object or a *InstPin* (ael) iterator.

Example

```
decl instPinIter = db_create_inst_pin_iter(dbInst);
for (; db_inst_pin_iter_is_valid(instPinIter);
   instPinIter = db_inst_pin_iter_get_next(instPinIter))
{
   decl termType = db_get_inst_pin_term_type(instPinIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstPin Overview (ael)
InstPin Functions (ael)
InstPin Iterator Functions (ael)
db_get_inst_pin_bbox() (ael)
db_get_inst_pin_instance() (ael)
db_get_inst_pin_inst_term() (ael)
db_get_inst_pin_net() (ael)
db_get_inst_pin_snap_layerid() (ael)
db_get_inst_pin_snap_point() (ael)
db_get_inst_pin_term_number() (ael)
db_is_inst_pin_connected_to_wire() (ael)

Where Used (ael)

Schematic, Layout

db_is_inst_pin_connected_to_wire()

Returns TRUE if the given *instance pin* (ael) is connected to a wire. Returns FALSE if the given *instance pin* (ael) is not connected to a wire.

Syntax

```
decl isConnectedToWire = db_is_inst_pin_connected_to_wire(dbInstPin);
```

Where,

- *dbInstPin* is an *InstPin* (ael) object or an *InstPin* (ael) iterator.

Example

```
decl context = de_get_current_design_context();
decl instPinIter = db_create_inst_pin_iter(context);
for (; db_inst_pin_iter_is_valid(instPinIter);
   instPinIter = db_inst_pin_iter_get_next(instPinIter))
{
   decl isConnectedToWire = db_is_inst_pin_connected_to_wire(instPinIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstPin Overview (ael)

InstPin Functions (ael)
InstPin Iterator Functions (ael)
db_get_inst_pin_bbox() (ael)
db_get_inst_pin_instance() (ael)
db_get_inst_pin_inst_term() (ael)
db_get_inst_pin_net() (ael)
db_get_inst_pin_snap_layerid() (ael)
db_get_inst_pin_snap_point() (ael)
db_get_inst_pin_term_number() (ael)
db_get_inst_pin_term_type() (ael)

Where Used (ael)

Schematic, Layout

InstPin Iterator Functions

- *InstPin Overview* (ael)
- *Connectivity Objects Overview* (ael)

This section describes the following functions:

- *db_create_inst_pin_iter()* (ael)
- *db_inst_pin_iter_is_valid()* (ael)
- *db_inst_pin_iter_get_next()* (ael)
- *db_inst_pin_iter_get_inst_pin()* (ael)

See also

InstPin Functions (ael)

db_create_inst_pin_iter()

Returns an *instance pin* (ael) iterator of the given instance or of *InstPins* (ael) connected to the given *Net* (ael). If no *instance pins* (ael) exist for the given instance or are connected to the given *Net* (ael), then the *instance pin* (ael) iterator that is returned will be invalid. The function *db_inst_pin_iter_is_valid()* (ael) can be used to test if an *InstPin* (ael) iterator is valid.

Syntax

```
decl instPinIter = db_create_inst_pin_iter(object);
```

Where,

- *object* is an instance, or is a *Net* (ael).

Example

```
decl instPinIter = db_create_inst_pin_iter(dbNet);
for (; db_inst_pin_iter_is_valid(instPinIter);
   instPinIter = db_inst_pin_iter_get_next(instPinIter))
{
   decl dbInstPinH = db_get_inst_pin_iter_get_inst_pin(instPinIter);
   decl bboxH = db_get_inst_pin_bbox(dbInstPinH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstPin Overview (ael)
InstPin Functions (ael)
InstPin Iterator Functions (ael)
db_inst_pin_iter_is_valid() (ael)

db_inst_pin_iter_get_next() (ael)
db_inst_pin_iter_get_inst_pin() (ael)

Where Used (ael)

Schematic, Layout

db_inst_pin_iter_get_inst_pin()

Returns the *instance pin* (ael) object referred to by the given *InstPin* (ael) iterator.

Syntax

```
decl dbInstPin = db_inst_pin_iter_get_inst_pin(instPinIter);
```

Where,

- *instPinIter* is a valid *InstPin* (ael) iterator.

Example

```
decl instPinIter = db_create_inst_pin_iter(dbNet);
for ( ; db_inst_pin_iter_is_valid(instPinIter);
   instPinIter = db_inst_pin_iter_get_next(instPinIter))
{
   decl dbInstPinH = db_get_inst_pin_iter_get_inst_pin(instPinIter);
   decl bboxH = db_get_inst_pin_bbox(dbInstPinH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstPin Overview (ael)
InstPin Functions (ael)
InstPin Iterator Functions (ael)
db_create_inst_pin_iter() (ael)
db_inst_pin_iter_is_valid() (ael)
db_inst_pin_iter_get_next() (ael)

Where Used (ael)

Schematic, Layout

db_inst_pin_iter_get_next()

Returns the next *InstPin* (ael) iterator from the given *InstPin* (ael) iterator.

Syntax

```
decl nextInstPinIter = db_inst_pin_iter_get_next(instPinIter);
```

Where,

- *instPinIter* is a valid *InstPin* (ael) iterator.

Example

```
decl instPinIter = db_create_inst_pin_iter(dbNet);
for ( ; db_inst_pin_iter_is_valid(instPinIter);
   instPinIter = db_inst_pin_iter_get_next(instPinIter))
{
   decl dbInstPinH = db_get_inst_pin_iter_get_inst_pin(instPinIter);
   decl bboxH = db_get_inst_pin_bbox(dbInstPinH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstPin Overview (ael)
InstPin Functions (ael)
InstPin Iterator Functions (ael)
db_create_inst_pin_iter() (ael)
db_inst_pin_iter_is_valid() (ael)
db_inst_pin_iter_get_inst_pin() (ael)

Where Used (ael)

Schematic, Layout

db_inst_pin_iter_is_valid()

Returns TRUE if the *InstPin* (ael) iterator is referencing a valid *instance pin* (ael).

Syntax

```
decl isValid = db_inst_pin_iter_is_valid(instPinIter);
```

Where,

- *instPinIter* is an *InstPin* (ael) iterator.

Example

```
decl instPinIter = db_create_inst_pin_iter(dbNet);
for ( ; db_inst_pin_iter_is_valid(instPinIter);
   instPinIter = db_inst_pin_iter_get_next(instPinIter))
{
   decl dbInstPinH = db_get_inst_pin_iter_get_inst_pin(instPinIter);
   decl bboxH = db_get_inst_pin_bbox(dbInstPinH);
   …
```

}

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstPin Overview (ael)
InstPin Functions (ael)
InstPin Iterator Functions (ael)
db_create_inst_pin_iter() (ael)
db_inst_pin_iter_get_next() (ael)
db_inst_pin_iter_get_inst_pin() (ael)

Where Used (ael)

Schematic, Layout

InstTerm Functions

- *InstTerm Overview* (ael)
- *Connectivity Objects Overview* (ael)

This section describes the following functions:

- *db_get_inst_term_name()* (ael)
- *db_get_inst_term_number()* (ael)
- *db_get_inst_term_net()* (ael)
- *db_get_inst_term_type()* (ael)
- *db_get_inst_term_instance()* (ael)
- *db_is_inst_term_grounded()* (ael)

See also

InstTerm Iterator Functions (ael)

db_get_inst_term_instance()

Returns the instance of a given *instance terminal* (ael).

Syntax

```
decl dbInst = db_get_inst_term_instance(dbInstTerm);
```

Where,

- *dbInstTerm* is an *InstTerm* (ael) object or *InstTerm* (ael) iterator

Example

```
decl instTermIter = db_create_inst_term_iter(dbNet);
for (; db_inst_term_iter_is_valid(instTermIter);
   instTermIter = db_inst_term_iter_get_next(instTermIter))
{
   //Get the instance name of the instance of the instance terminal.
   decl instH = db_get_inst_term_instance(instTermIter);
   decl instName = db_get_instance_name(instH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstTerm Overview (ael)
InstTerm Functions (ael)
InstTerm Iterator Functions (ael)
db_get_inst_term_name() (ael)
db_get_inst_term_number() (ael)
db_get_inst_term_net() (ael)

db_get_inst_term_type() (ael)
db_is_inst_term_grounded() (ael)

Where Used (ael)

Schematic, Layout

db_get_inst_term_name()

Returns the name of a given *instance terminal* (ael).

Syntax

```
decl instTermName = db_get_inst_term_name(dbInstTerm);
```

Where,

- *dbInstTerm* is an *InstTerm* (ael) object or *InstTerm* (ael) iterator.

Example

```
decl instTermIter = db_create_inst_term_iter(dbNet);
for (; db_inst_term_iter_is_valid(instTermIter);
   instTermIter = db_inst_term_iter_get_next(instTermIter))
{
   //Get the instance terminal name.
   decl instTermName = db_get_inst_term_name(instTermIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstTerm Overview (ael)
InstTerm Functions (ael)
InstTerm Iterator Functions (ael)
db_get_inst_term_number() (ael)
db_get_inst_term_net() (ael)
db_get_inst_term_type() (ael)
db_get_inst_term_instance() (ael)
db_is_inst_term_grounded() (ael)

Where Used (ael)

Schematic, Layout

db_get_inst_term_net()

Returns the *Net* (ael) of a given *instance terminal* (ael). This function will return NULL if the *InstTerm* (ael) is not connected to a *Net* (ael).

Syntax

```
decl dbNet = db_get_inst_term_net(dbInstTerm);
```

Where,

- *dbInstTerm* is an *InstTerm* (ael) object or *InstTerm* (ael) iterator.

Example

```
decl instTermIter = db_create_inst_term_iter(dbNet);
for (; db_inst_term_iter_is_valid(instTermIter);
   instTermIter = db_inst_term_iter_get_next(instTermIter))
{
   // If it is a connected InstTerm get its Net's name.
   decl netH = db_get_inst_term_net(instTermIter);
   if (netH != NULL)
   {
      //Get the net name of the net of the instance terminal.
      decl netName = db_get_net_name(netH);
      …
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstTerm Overview (ael)
InstTerm Functions (ael)
InstTerm Iterator Functions (ael)
db_get_inst_term_name() (ael)
db_get_inst_term_number() (ael)
db_get_inst_term_type() (ael)
db_get_inst_term_instance() (ael)
db_is_inst_term_grounded() (ael)
Net Overview (ael)
Net Functions (ael)

Where Used (ael)

Schematic, Layout

db_get_inst_term_number()

Returns the number of a given *instance terminal* (ael).

Syntax

```
decl number = db_get_inst_term_number(dbInstTerm);
```

Where,

- *dbInstTerm* is an *InstTerm* (ael) object or *InstTerm* (ael) iterator.

Example

```
decl instTermIter = db_create_inst_term_iter(dbNet);
for (; db_inst_term_iter_is_valid(instTermIter);
   instTermIter = db_inst_term_iter_get_next(instTermIter))
{
   //Get the instance terminal number.
   decl instTermNum = db_get_inst_term_number(instTermIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstTerm Overview (ael)
InstTerm Functions (ael)
InstTerm Iterator Functions (ael)
db_get_inst_term_name() (ael)
db_get_inst_term_net() (ael)
db_get_inst_term_type() (ael)
db_get_inst_term_instance() (ael)
db_is_inst_term_grounded() (ael)

Where Used (ael)

Schematic, Layout

db_get_inst_term_type()

Returns an integer terminal type of a given *InstTerm* (ael). The terminal type describes the direction of the *InstTerm* (ael).

The possible returned integer *terminal* (ael) types are :

- **INPUT_PIN** = 0, represents an input *instance terminal* (ael).
- **OUTPUT_PIN** = 1, represents an output *instance terminal* (ael).
- **IN_OUT_PIN** = 2, represents an input-output *instance terminal* (ael).

Syntax

```
decl termType = db_get_inst_term_type(dbInstTerm);
```

Where,

- *dbInstTerm* is an *InstTerm* (ael) object or a *InstTerm* (ael) iterator.

Example

```
decl instTermIter = db_create_inst_term_iter(dbInst);
for (; db_inst_term_iter_is_valid(instTermIter);
   instTermIter = db_inst_term_iter_get_next(instTermIter))
{
   decl termType = db_get_inst_term_type(instTermIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstTerm Overview (ael)
InstTerm Functions (ael)
InstTerm Iterator Functions (ael)
db_get_inst_term_name() (ael)
db_get_inst_term_number() (ael)
db_get_inst_term_net() (ael)
db_get_inst_term_instance() (ael)
db_is_inst_term_grounded() (ael)

Where Used (ael)

Schematic, Layout

db_is_inst_term_grounded()

Returns TRUE if the given *InstTerm* (ael) is grounded.

Syntax

```
decl isGroundedInstTerm = db_is_inst_term_grounded(dbInstTerm);
```

Where,

- *dbInstTerm* is an *InstTerm* (ael) object or *InstTerm* (ael) iterator.

Example

```
decl instTermIter = db_create_inst_term_iter(dbNet);
for (; db_inst_term_iter_is_valid(instTermIter);
   instTermIter = db_inst_term_iter_get_next(instTermIter))
{
   // Skip grounded instance terminals.
   if (db_is_inst_term_grounded(netIter))
       continue;
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstTerm Overview (ael)
InstTerm Functions (ael)
InstTerm Iterator Functions (ael)
db_get_inst_term_name() (ael)
db_get_inst_term_number() (ael)
db_get_inst_term_net() (ael)
db_get_inst_term_type() (ael)
db_get_inst_term_instance() (ael)

Where Used (ael)

Schematic, Layout

InstTerm Iterator Functions

- *InstTerm Overview* (ael)
- *Connectivity Objects Overview* (ael)

This section describes the following functions:

- *db_create_inst_term_iter()* (ael)
- *db_inst_term_iter_is_valid()* (ael)
- *db_inst_term_iter_get_next()* (ael)
- *db_inst_term_iter_get_inst_term()* (ael)

See also

InstTerm Functions (ael)

db_create_inst_term_iter()

Returns an *InstTerm* (ael) iterator from a given *Net* (ael) or Instance object. If no *instance terminals* (ael) exist for the given *Net* (ael) or Instance, then the *InstTerm* (ael) iterator that is returned will be invalid. The function *db_inst_term_iter_is_valid()* (ael) can be used to test if an *InstTerm* (ael) iterator is valid.

Syntax

```
decl instTermIter = db_create_inst_term_iter(object);
```

Where,

- *object* is a *Net* (ael) object or *Net* (ael) iterator, or is an instance object or instance iterator.

Example

```
decl instTermIter = db_create_inst_term_iter(dbNet);
for (; db_inst_term_iter_is_valid(instTermIter);
   instTermIter = db_inst_term_iter_get_next(instTermIter))
{
   decl dbInstTermH = db_get_inst_term_iter_get_inst_term(instTermIter);
   //Get the instance terminal number.
   decl instTermNum = db_get_inst_term_number(dbInstTermH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstTerm Overview (ael)
InstTerm Functions (ael)
InstTerm Iterator Functions (ael)
db_inst_term_iter_is_valid() (ael)

db_inst_term_iter_get_next() (ael)

Where Used (ael)

Schematic, Layout

db_inst_term_iter_get_inst_term()

Returns the *instance terminal* (ael) object that is referenced by the given *InstTerm* (ael) iterator.

Syntax

```
decl dbInstTerm = db_inst_term_iter_get_inst_term(instTermIter);
```

Where,

- *instTermIter* is a valid *InstTerm* (ael) iterator.

Example

```
decl instTermIter = db_create_inst_term_iter(dbNet);
for (; db_inst_term_iter_is_valid(instTermIter);
   instTermIter = db_inst_term_iter_get_next(instTermIter))
{
   decl dbInstTermH = db_get_inst_term_iter_get_inst_term(instTermIter);
   //Get the instance terminal number.
   decl instTermNum = db_get_inst_term_number(dbInstTermH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstTerm Overview (ael)
InstTerm Functions (ael)
InstTerm Iterator Functions (ael)
db_create_inst_term_iter() (ael)
db_inst_term_iter_is_valid() (ael)
db_inst_term_iter_get_next() (ael)

Where Used (ael)

Schematic, Layout

db_inst_term_iter_get_next()

Returns the next *InstTerm* (ael) iterator of the given *InstTerm* (ael) iterator.

Syntax

```
decl nextInstTermIter = db_inst_term_iter_get_next(instTermIter);
```

Where,

- *instTermIter* is a valid *InstTerm* (ael) iterator.

Example

```
decl instTermIter = db_create_inst_term_iter(dbNet);
for (; db_inst_term_iter_is_valid(instTermIter);
   instTermIter = db_inst_term_iter_get_next(instTermIter))
{
   decl dbInstTermH = db_get_inst_term_iter_get_inst_term(instTermIter);
   //Get the instance terminal number.
   decl instTermNum = db_get_inst_term_number(dbInstTermH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstTerm Overview (ael)
InstTerm Functions (ael)
InstTerm Iterator Functions (ael)
db_create_inst_term_iter() (ael)
db_inst_term_iter_is_valid() (ael)

Where Used (ael)

Schematic, Layout

db_inst_term_iter_is_valid()

Returns TRUE if the given *InstTerm* (ael) iterator is referencing a valid *InstTerm* (ael) object.

Syntax

```
decl isValid = db_inst_term_iter_is_valid(instTermIter);
```

Where,

- *instTermIter* is an *InstTerm* (ael) iterator.

Example

```
decl instTermIter = db_create_inst_term_iter(dbNet);
  for (; db_inst_term_iter_is_valid(instTermIter);
      instTermIter = db_inst_term_iter_get_next(instTermIter))
  {
      decl dbInstTermH = db_get_inst_term_iter_get_inst_term(instTermIter);
```

```
    //Get the instance terminal number.
    decl instTermNum = db_get_inst_term_number(dbInstTermH);
    …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
InstTerm Overview (ael)
InstTerm Functions (ael)
InstTerm Iterator Functions (ael)
db_create_inst_term_iter() (ael)
db_inst_term_iter_get_next() (ael)

Where Used (ael)

Schematic, Layout

Item Definition Functions

- ***Item Definition - Overview*** **(ael)**

This section describes the following functions:

- *dm_item_get_name()* (ael)
- *dm_item_get_label()* (ael)
- *dm_item_get_library_name()* (ael)
- *dm_item_get_attr()* (ael)
- *dm_item_get_ex_attr()* (ael)
- *dm_item_get_priority()* (ael)
- *dm_item_get_prefix()* (ael)
- *dm_item_get_netlist_format()* (ael)
- *dm_item_get_netlist_data()* (ael)
- *dm_item_get_symbol_format()* (ael)
- *dm_item_get_symbol_name()* (ael)
- *dm_item_get_icon_name()* (ael)
- *dm_item_get_dialog_name()* (ael)
- *dm_item_get_dialog_data()* (ael)
- *dm_item_get_artwork_type()* (ael)
- *dm_item_get_artwork_data()* (ael)
- *dm_item_get_parms()* (ael)
- *dm_item_get_parm_names()* (ael)
- *dm_item_is_bom_item()* (ael)
- *dm_item_is_netlist_from_layout()* (ael)
- *dm_item_is_variable()* (ael)
- *dm_item_is_unique()* (ael)
- *dm_item_is_sub_design()* (ael)
- *dm_item_is_port()* (ael)
- *dm_item_no_special_characters()* (ael)

See also

- *create_item()* (ael)
- *Component Definition Functions* (ael)
- *Parameter Definition - Overview* (ael)
- *Parameter Definition Functions* (ael)

dm_item_get_artwork_data()

This functon returns the string artwork data for an item definition. The artwork data string holds the name of the artwork function or design containing the artwork for the component.

For fixed artwork, the artwork data string should be a layout artwork design name within the * extension. For macro artwork, the artwork data string should be an AEL artwork generation function name.

Note
ADS 2011 and newer versions, ignores the artworkData field for fixed artwork artwork types. ADS only uses artworkData for ael generated artwork artwork types to retrieve the AEL artwork macro function name.

Syntax

```
decl itemArtworkDataStr = dm_item_get_artwork_data(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl itemArtworkDataStr = dm_item_get_artwork_data(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

***Where Used* (ael)**

Schematic, Layout

dm_item_get_artwork_type()

Returns the integer artwork type for an item definition. The artwork type indicates the type of artwork for the component.

- 0 = no artwork
- 1 = fixed artwork
- 2 = ael generated artwork
- 3 = synchronized

Syntax

```
decl itemArtworkType = dm_item_get_artwork_type(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl itemArtworkType = dm_item_get_artwork_type(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_get_attr()

Returns the item attribute code of an item definition. See *Item Attributes* (ael) for more information on what types of item attributes are available.

Syntax

```
decl attrCode = dm_item_get_attr(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl attrCode = dm_item_get_attr(itemH);
if (attrCode & ITEM_VARIABLE)
{
  // A VAR component was found.
  …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_get_dialog_data()

Returns the string dialog data for an item definition. The dialog data string is an optional string value to pass to the edit component parameter dialog’s create function.

Syntax

```
decl itemDialogDataStr = dm_item_get_dialog_data(itemH);
```

Where,

- *itemH* is an item definition

Example

```
decl itemDialogDataStr = dm_item_get_dialog_data(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_get_dialog_name()

Returns the string dialog name for an item definition. The dialog name string is the name of the edit component parameter dialog, usually “standard_dialog” that is used for editing the parameters and/or attributes of the component.

Syntax

```
decl itemDialogNameStr = dm_item_get_dialog_name(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl itemDialogNameStr = dm_item_get_dialog_name(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_get_ex_attr()

Returns the item extra attribute code of an item definition. Please see *Item Extra*

Attributes (ael) for more information on what types of extra item attributes are available.

Syntax

```
decl attrCode = dm_item_get_ex_attr(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl attrCode = dm_item_get_ex_attr(itemH);
if (attrCode & ITEM_INITIAL_ORIENTATION_LEFT_EX)
{
  // A left oriented component instance symbol attribute was found.
  …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_get_icon_name()

Returns the string icon name for an item definition. The icon name string stores the name of the bitmap file used for the component button in the palette.

Syntax

```
decl itemBitmapNameStr = dm_item_get_icon_name(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl itemBitmapNameStr = dm_item_get_icon_name(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_get_label()

Returns the string descriptive label of an item definition. This label is used as the balloon help for the component palette selection.

Syntax

```
decl labelStr = dm_item_get_label(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl labelStr = dm_item_get_label(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_get_library_name()

This function returns the string library name of an item definition. If the component does not have a library then an empty string is returned.

Note
This is the name of the library containing the cell with this item definition.

Syntax

```
decl libNameStr = dm_item_get_library_name(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl libNameStr = dm_item_get_library_name(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

***Where Used* (ael)**

Schematic, Layout

dm_item_get_name()

Returns the string name of an item definition.
The unique name of the component.

> **Note**
> The returned item name will also be the cell name.

Syntax

```
decl nameStr = dm_item_get_name(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl nameStr = dm_item_get_name(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

***Where Used* (ael)**

Schematic, Layout

dm_item_get_netlist_data()

Returns the netlist data string for an item definition. The netlist data is used in conjunction with the netlist format string option "%d". "%d" will netlist the string owned by the netlist data string. To learn more about netlist format strings, see *Format strings* (ael).

Syntax

```
decl itemNetlistDataStr = dm_item_get_netlist_data(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl itemNetlistDataStr = dm_item_get_netlist_data(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_get_netlist_format()

Returns the netlist format string for an item definition. The netlist format string controls the netlist formatting for the component. To learn more about netlist format strings, please see the following: *Format strings* (ael)

Syntax

```
decl itemNetlistFormatStr = dm_item_get_netlist_format(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl itemNetlistFormatStr = dm_item_get_netlist_format(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_get_parm_names()

Returns a list of string parameter names for an item definition. If the item definition contains no parameter definitions, then this function will return an empty list. An item definition can contain an optional parameter definition list which defines the parameters for that component. To learn more about parameter definitions, refer *Parameter Definition - Overview* (ael).

Syntax

```
decl itemParmNameList = dm_item_get_parm_names(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl context = de_get_current_design_context();

decl itemDefP = db_get_item_definition(context);
if (!itemDefP)
   return;

decl parmNameList = dm_item_get_parm_names(itemDefP)
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_get_parms()

Returns the parameter definition list for an item definition. If the item definition contains no parameter definitions, then this function will return NULL. An item definition can contain an optional parameter definition list which defines the parameters for that

component. To learn more about parameter definitions, please see : *Parameter Definition - Overview* (ael).

Syntax

```
decl itemParmDefList = dm_item_get_parms(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl context = de_get_current_design_context();

decl itemDefP = db_get_item_definition(context);
if (!itemDefP)
   return;

decl parmDefList = dm_item_get_parms(itemDefP);
decl parmDefP = dm_first_parm_definition(parmDefList);
for ( ; parmDefP != NULL; parmDefP = dm_next_parm_definition(parmDefP))
{
   // Get parameter definition information.
   decl defaultParmVal = dm_parm_get_defvalue(parmDefP);
   ...
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_get_prefix()

Returns the string instance prefix name for an item definition. An instance string name is typically something such as X1, X2, X3 .. Where “X” would be the instance prefix string name for that component.

Syntax

```
decl itemPrefixStr = dm_item_get_prefix(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl itemPrefixStr = dm_item_get_prefix(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

***Where Used* (ael)**

Schematic, Layout

dm_item_get_priority()

Returns the integer priority for an item definition. The priority indicates if special netlist handling is required or not. For all components except for netlist include components, the priority is NULL or -1. For netlist include components, the priority is typically set to zero to avoid illegal nested subcircuits.

Syntax

```
decl itemPriority = dm_item_get_priority(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl itemPriority = dm_item_get_priority(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

***Where Used* (ael)**

Schematic, Layout

dm_item_get_symbol_format()

Returns the symbol format string for an item definition. The symbol format string controls

the display of the component annotation in the schematic for the component. This format string also dictates how the on-screen editing works.

Syntax

```
decl itemDisplayFormatStr = dm_item_get_symbol_format(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl itemDisplayFormatStr = dm_item_get_symbol_format(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_get_symbol_name()

Returns the string symbol name for an item definition. The symbol name string stores the name of the schematic symbol for the component.

Note
ADS 2011 and newer versions, ignore the item definition's *symbolName* data field.

Syntax

```
decl itemSymbolNameStr = dm_item_get_symbol_name(itemH);
```

Where,

- *itemH* is an item definition

Example

```
decl itemSymbolNameStr = dm_item_get_symbol_name(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_is_bom_item()

Returns TRUE if an item has the isBom flag set, otherwise returns FALSE.

Syntax

```
decl itemIsBOM = dm_item_is_bom_item(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl itemIsBOM = dm_item_is_bom_item(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_is_netlist_from_layout()

Returns TRUE if an item is marked to be netlisted from layout, otherwise returns FALSE.

Syntax

```
decl itemIsNetlistedFromLayout = dm_item_is_netlist_from_layout(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl itemIsNetlistedFromLayout = dm_item_is_netlist_from_layout(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_is_port()

Returns TRUE if an item is a Port, otherwise it returns FALSE.

Note
In ADS 2011 and newer versions, there are no instance ports, instead ports are represented by pins.

Syntax

```
decl itemIsPort = dm_item_is_port(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl itemIsPort = dm_item_is_port(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_is_sub_design()

Returns TRUE if an item is a sub-circuit, otherwise it returns FALSE.

Syntax

```
decl itemIsSubCkt = dm_item_is_sub_design(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl itemIsSubCkt = dm_item_is_sub_design(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

***Where Used* (ael)**

Schematic, Layout

dm_item_is_unique()

Returns TRUE if an item is marked as a unique component, which limits there to be only one possible instance of that component per design, otherwise it returns FALSE.

Syntax

```
decl itemIsUnique = dm_item_is_unique(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl itemIsUnique = dm_item_is_unique(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

***Where Used* (ael)**

Schematic, Layout

dm_item_is_variable()

Returns TRUE if an item is marked as a variable (VAR) component, otherwise returns FALSE.

Syntax

```
decl itemIsVariable = dm_item_is_variable(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl itemIsVariable = dm_item_is_variable(itemH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_item_no_special_characters()

Returns TRUE if the item disallows special characters in the component's instance name, otherwise it returns FALSE.

Syntax

```
decl itemDisallowsSpecialChars = dm_item_no_special_characters(itemH);
```

Where,

- *itemH* is an item definition.

Example

```
decl itemAllowsSpecialChars = (dm_item_no_special_characters(itemH) ? FALSE : TRUE);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Item Definition - Overview (ael)
Item Definition Functions (ael)

Where Used (ael)

Schematic, Layout

Item Definition - Overview

An ADS Item Definition may also be known as a Component Definition or a Component Item Definition within ADS. An Item Definition defines and stores information about a particular ADS component/design. An Item Definition AEL object stores the component definition of an ADS component/design. The *create_item()* (ael) function is used to create a new component item definition within ADS.

The types of information that an Item Definition AEL object stores about each ADS component includes:

- the component's name, and brief description
- the default instance prefix name for instances of that component,
- the component's *parameter definitions* (ael),
- the component's netlist format rules and callbacks,
- the component's display format rules,
- the component's symbol name,
- the component's artwork type and artwork data,
- the component's attributes that helps denote if the component has special conditions or functionality.

Component information stored within an Item Definition

These are the different data fields stored within an AEL Item Definition Object.

Name

- *name* is a string. It is a unique name of the component.

> **Note**
> - In ADS 2011 and newer versions, the item name will be the cell name.
> - In ADS 2009 Update 1 and older versions, the item name may be represented by a full file path name.

Label

- *label* is a string; a descriptive label for the component. This is also used as the balloon help for the component palette selection.

Prefix

- *prefix* is a string; the prefix for the instance name (e.g., "TL" for name "TL1") used when the component is placed.

Attribute

- *attrib* is an attribute code. To set multiple attributes, bit-wise OR their numeric equivalent values.
  For example, to set both ITEM_BEND and ITEM_BOM_ITEM, bit-wise OR their values and set *attrib* to the combined value. e.g. attrib = ITEM_BEND | ITEM_BOM_ITEM;

Attribute Choices

Attribute	Numeric Equivalent	Description
No attribute set	0	Only the instance line is netlisted; the simulation behavior is defined by a user-compiled model, model card, or sub-circuit. For the latter two options, a Netlist Include must be employed to reference the model card or sub-circuit.
ITEM_BEND	2048	Component is a transmission line bend.
ITEM_BOM_ITEM	262144	Component is included in a BOM.
ITEM_DESIGN_INST	16	Component refers to a sub-design. When set, the component is netlisted as a sub-circuit with both a sub-circuit definition and an instance of the component. When not set, the component netlists only as a instance of the component.
ITEM_DRAWSHEET	16384	Component is a drawing sheet or PCB outline. Placed in the fourth field on the *create_item()* (ael) statement.
ITEM_GLOBAL	512	Component has global scope, such as MSUB.
ITEM_GLOBAL_NODE	536870912	Component is a global node.
ITEM_GROUND	1	Component is a ground.
ITEM_NO_LAYOUT_ANNOT	131072	Component does not generate annotation.
ITEM_NOT_ALL_PARM	1073741824	Component does not have an "all parameters" selection in the standard dialog box.
ITEM_NOT_NETLISTED	268435456	Component is not netlisted.
ITEM_NOT_PLACABLE	134217728	Component is not placable.
ITEM_PORT	32	Component is a port.
ITEM_SCOPE_GLOBAL	16777216	Component has global scope.
ITEM_SCOPE_LOCAL	4194304	Component has local scope.
ITEM_SCOPE_NESTED	8388608	Component has nested scope.
ITEM_TEE	4096	Component is a transmission line connection tee.
ITEM_UNIQUE	8	Only one instance of this component can be placed in a design.
ITEM_VARIABLE	2	Component is an equation/variable definition.

Priority

- *priority* is an integer; indicates special netlist handling. Specified as NULL or -1 for all components except the netlist include components. For netlist include components, set the netlisting priority to zero, to avoid illegal nested sub-circuits.

Icon Name

- *iconName* is a string; name of the bitmap file used for component button in a palette.

Dialog Name

- *dialogName* is a string; name of the dialog, usually `standard_dialog`. The dialog is used to edit the parameters and/or some other attributes of the component.

Dialog Data

- *dialogData* is a string; passed to the dialog creating function. Usually "*". Not really used.

Netlist Format

- *netlistFormat* is a string; used to control the netlist format for the component. Usually specified with the variable `standard_netlist` or `ComponentNetlistFmt`.

Netlist Data

- *netlistData* is a string; used in conjunction with the netlist format string. %d.

Display Format

- *displayFormat* is a string; used to control the display of the component annotation in the schematic. Usually specified with the variable `standard_symbol`. Currently this format string also dictates how on-screen editing works.

Symbol Name

- *symbolName* is a string; the name of the schematic symbol for the component.

> **Note**
> ADS 2011 and newer versions no longer uses *symbolName*; it ignores the *symbolName* data field.

Artwork Type

- *artworkType* is an integer; indicates the type of artwork for the component, where:
  - 0 = no artwork
  - 1 = fixed artwork
  - 2 = ael generated artwork
  - 3 = synchronized

Artwork Data

- *artworkData* is a string; the name of the artwork function or design containing the artwork. For fixed artwork, it should be a layout artwork design name. For macro artwork, it should be an AEL artwork generation function. If artwork Type = 1, string is set to the name of the design file containing the artwork .

> **Note**
> - For ADS 2009 Update 1 and earlier versions, do not use the `.dsn` extension when entering design file name.
> - ADS 2011 and newer versions ignores the *artworkData* field for fixed artwork artwork types. It only uses *artworkData* for AEL generated artwork artwork types to retrieve the AEL artwork macro function name.

Extra Attribute (Optional)

- *extraAttribute* is optional; an extension to the attribute code.
  Extra Attribute Choices:

Attribute	Numeric Equivalent	Description
ITEM_ANALYSIS_EX	16384	The component is an analysis item. Placed in the fourteenth field of the create_item() statement.
ITEM_CKT_MODEL_EX	64	Identifies the model as Analog/RF type.
ITEM_CROSS_EX	8	
ITEM_CURVE_EX	1	
ITEM_FILE_EX	4096	Identifies file component for the Edit Component dialog.
ITEM_FREQUENCY_EX	1024	Identifies frequency component for the sparam and freqplan dialog.
ITEM_GOAL_EX	262144	The component is a goal item.
ITEM_INITIAL_ORIENTATION_DOWN_EX	524288	Place -90.
ITEM_INITIAL_ORIENTATION_LEFT_EX	2097152	Place 180.
ITEM_INITIAL_ORIENTATION_RIGHT_EX	4194304	Place 0.
ITEM_INITIAL_ORIENTATION_UP_EX	1048576	Place 90.
ITEM_MEASUREMENT_EX	32768	The component is a measurement item.
ITEM_MSTRIP_EX	2	
ITEM_NO_SPL_CHAR_EX	268435456	Only allow alpha-numeric or underscore characters in the component instance name.
ITEM_NOT_AUTO_REPEAT_PLACEMENT_EX	8388608	Place one component at a time.
ITEM_PCB_EX	16	
ITEM_PRIMITIVE_EX	32	Identifies the associated item as a compiled component contained in the simulator.
ITEM_SHORTABLE_EX	147483648	Component with more than two pins may be shorted. The short will be created between Pins 1 and 2, all other pins will be ignored.
ITEM_SPEC_EX	131072	The component is a yield specification item.
ITEM_STRIP_LINE_EX	4	

Note
If none of the ITEM_INITIAL_ORIENTATION_..._EX options are set, use the previously-selected orientation.

Callback List (Optional)

- *cbList* is optional; the list of callbacks. For example, list(dm_create_cb ("...", "..."), ...). Currently supported callbacks are: ITEM_NETLIST_CB

Parameter Definition List (Optional)

- *parameterN* is optional; the list of parameters. Return value from the create_parm() command. For more information on Parameter Definitions, see *Parameter Definition - Overview* (ael).

Creating a new Component Definition

To create/define a new ADS component item definition, the AEL function *create_item()* (ael) is used to define a new ADS component.

Example of Creating a new Item Definition

```
/* TLIN */
create_item("TLIN", // name
    "Ideal 2-Terminal Transmission Line", // label
    "TL", // prefix
    0, // attribute
    NULL, // priority
    "TLIN", // iconName
    standard_dialog, // dialogName
    "*", // dialogData
    ComponentNetlistFmt, // netlistFormat
    "TLIN", // netlistData
    ComponentAnnotFmt, // displayFormat
    "SYM_TLin", // symbolName
    no_artwork, // artworkType
    NULL, // artworkData
    ITEM_PRIMITIVE_EX, // extraAttrib
    create_parm ("Z", "Characteristic Impedance", PARM_OPTIMIZABLE |
        PARM_STATISTICAL, "StdFileFormSet",
        RESISTANCE_UNIT, prm("StdForm","50.0")),
    create_parm ("E", "Electrical Length", PARM_OPTIMIZABLE |
        PARM_STATISTICAL, "StdFileFormSet", ANGLE_UNIT,
        prm("StdForm","90")),
    create_parm ("F", "Reference Frequency for Electrical Length",
        PARM_OPTIMIZABLE | PARM_STATISTICAL, "StdFileFormSet",
        FREQUENCY_UNIT, prm("StdForm","1 GHz")));
```

Querying Item Definition Data

An ItemDefinition object is useful when there is a need to query or retrieve component definition data.

To retrieve a component's item definition from a *design context* (ael), the AEL function *db_get_item_definition()* (ael) can be used.

An Instance Iterator from a *DesignContext* (ael) can be used to traverse a design's instance data. The instance object and/or instance iterator can be used to retrieve a particular component instance's item definition information. The AEL function *db_get_instance_item_definition()* (ael) is used to retrieve the item definition for a particular instance within a *DesignContext* (ael).

See the list of *Item Definition Functions* (ael) for more information.

Example of Querying Item Definition Information

```
decl context = de_get_current_design_context();

decl instIter = db_create_inst_iter(context);
for( ; db_inst_iter_is_valid(instIter);
   instIter = db_inst_iter_get_next(instIter); )
{
   // Get the instance's item definition.
   decl itemDefP = db_get_instance_item_definition(instIter);

   // Get the component item defintion name and artwork type of the instance.
   decl compName = dm_item_get_name(itemDefP);
   decl artworkType = dm_item_get_artwork_type(itemDefP);
   ...
}
```

List of Item Definition Functions

See *Item Definition Functions* (ael).

Layer and LayerId Functions

Overview - LayerId (ael)

This section describes the following Layer and LayerId functions:

Get LayerId and LayerId Name Functions

db_get_layerid() (ael)
db_layerid() (ael)
db_get_layerid_name() (ael)
db_get_layerid_names() (ael)
db_get_layerid_index() (ael)
db_get_layout_layerid_name() (ael)
db_get_layout_layerid_names() (ael)

Create Layer Function

db_create_layer() (ael)

Find LayerId Functions

db_find_layerid_by_name() (ael)
db_find_layout_layerid_by_name() (ael)

Find Layer Functions

db_find_layer_name_by_number() (ael)
db_find_layer_number_by_name() (ael)

Layer, LayerId Attribute Functions

db_get_layer_binding() (ael)
db_get_layer_number() (ael)
db_get_purpose_number() (ael)
db_get_layer_process_role() (ael)
db_set_layer_process_role() (ael)
db_is_layerid_invisible() (ael)
db_is_layerid_protected() (ael)
db_set_layerid_invisible() (ael)
db_set_layerid_protected() (ael)
db_get_layerid_alpha() (ael)
db_get_layerid_fill_mode() (ael)
db_get_layerid_fill_pattern() (ael)
db_get_layerid_line_style() (ael)
db_get_layerid_rgb() (ael)
db_set_layerid_alpha() (ael)
db_set_layerid_fill_mode() (ael)
db_set_layerid_fill_pattern() (ael)
db_set_layerid_line_style() (ael)
db_set_layerid_rgb() (ael)

ADS Pre-defined LayerId Functions

db_get_layerid_for_comp_name() (ael)
db_get_layerid_for_inst_name() (ael)
db_get_layerid_for_parameters() (ael)
db_get_layerid_for_symbol_body() (ael)
db_get_layerid_for_symbol_text() (ael)
db_get_layerid_for_schematic_wires() (ael)

db_create_layer()

This function creates a new physical layer for the given *design context* (ael), given the layer's name and number.
Returns TRUE if the new layer was created successfully, FALSE otherwise.

Syntax

```
decl bLayerIsCreated = db_create_layer(context, layerName, layerNumber);
```

Where,

- *context* is a *design context* (ael).
- *layerName* is the layer name for the layer to create.
- *layerNumber* is the integer number for the layer to create.

Example

```
decl context = de_get_current_design_context();
decl bIsCreated = db_create_layer(context, "MetalAlum4", 22);
if (bIsCreated == FALSE)
   return FALSE;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Layer and LayerId Functions (ael)

Where Used (ael)

Schematic, Layout

db_find_layer_name_by_number()

Finds and returns the name of a layer in a *design context* (ael) with the given layer number. Returns NULL if a layer with the given layer number could not be found.

Syntax

```
decl layerName = db_find_layer_name_by_number(context, layerNumber);
```

Where,

- *context* is a *design context* (ael).
- *layerNumber* is the integer number for the layer.

Example

```
decl context = de_get_current_design_context();
decl layerName = db_find_layer_name_by_number(context, 0);
if (layerName == NULL)
   return FALSE;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Layer and LayerId Functions (ael)
db_find_layer_number_by_name() (ael)

***Where Used* (ael)**

Schematic, Layout

db_find_layer_number_by_name()

Finds and returns the number of a layer in a *design context* (ael) with the given layer name. Returns NULL if a layer with the given layer name could not be found.

Syntax

```
decl layerNumber = db_find_layer_number_by_name(context, layerName);
```

Where,

- *context* is a *design context* (ael).
- *layerName* is the layer name.

Example

```
decl context = de_get_current_design_context();
decl layerNumber = db_find_layer_number_by_name(context, 'wires");
if (layerNumber == NULL)
   return FALSE;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Layer and LayerId Functions (ael)

db_find_layer_name_by_number() (ael)

Where Used (ael)

Schematic, Layout

db_find_layerid_by_name()

This function finds and returns a *LayerId* (ael) of a given *design context* (ael) that has a given *LayerId* (ael) name.
The function *db_get_layerid_name()* (ael) can be used to get a particular *LayerId* (ael)'s name.
If a *LayerId* (ael) cannot be retrieved with that *LayerId* (ael) name within that design context then NULL is returned.

Syntax

```
decl layerId = db_find_layerid_by_name(context, layerIdName);
```

Where,

- *context* is a *design context* (ael).
- *layerIdName* is the name of the layerId.

Note
A *LayerId* (ael) name is in the following format: "<layerName>:<purposeName>".
The function *db_get_layerid_name()* (ael) can be used to get the *LayerId* (ael) name for a particular *LayerId* (ael).

Example

```
decl context = de_get_current_design_context();
decl layerList = db_get_layerid_names(context);
while ( layerList != NULL )
{
   decl layerIdName = car( layerList );

   decl layerId = db_find_layerid_by_name(context, layerIdName);

   decl layerNumber = db_get_layer_number(layerId);
   decl purposeNumber = db_get_purpose_number(layerId);
   ...
   layerList = cdr( layerList );
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_get_layerid_name() (ael)

Where Used (ael)

Schematic, Layout

db_find_layout_layerid_by_name()

This function finds and returns a layout *LayerId* (ael) of a given *design context* (ael) that has the given *LayerId* (ael) name.
The function *db_get_layerid_name()* (ael) can be used to get a particular *LayerId* (ael)'s name.
If a *LayerId* (ael) cannot be retrieved with that *LayerId* (ael) name within that design context then NULL is returned.

Note
This function returns the same value as *db_find_layerid_by_name()* (ael) since there is no difference between the set of symbol, schematic, and layout *LayerIds* (ael) in ADS.

Syntax

```
decl layerId = db_find_layout_layerid_by_name(context, layerIdname);
```

Where,

- *context* is a *design context* (ael).
- *layerIdname* is a name of a *LayerId* (ael), returned from a function such as *db_get_layerid_name()* (ael).

Note
A *LayerId* (ael) name is in the following format: "<layerName>:<purposeName>". The function *db_get_layerid_name()* (ael) can be used to get the *LayerId* (ael) name for a particular *LayerId* (ael).

Example

```
decl layerIdName = db_get_layerid_name(layerId);
decl layerId = db_find_layout_layerid_by_name(context, layerIdName);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_get_layerid_name() (ael)
db_get_layout_layerid_name() (ael)
db_find_layerid_by_name() (ael)

Where Used (ael)

Schematic, Layout

db_get_layer_binding()

Returns the layer binding string for a given layer number in the given *design context* (ael)'s library's technology.

The layer binding string is a space-delimited string of binding names. A layer binds to all other layers that have at least one binding name in common. Layers that bind to each other are considered electrically shorted together.

Syntax

```
decl bindingStr = db_get_layer_binding(context, layerNum);
```

Where,

- *context* is the *DesignContext* (ael) of a design from the library whose technology the layer resides within.
- *layerNum* is the number of the layer to retrieve the layer binding string for.

Example

```
decl layerNum = db_get_layer_number(layerId);
decl bindingStr = db_get_layer_binding(context, layerNum);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer releases

See also

Layer and LayerId Functions (ael)

Where Used (ael)

Schematic, Layout

db_get_layer_number()

Returns the layer number for a given *LayerId* (ael).

Syntax

```
decl layerNum = db_get_layer_number(layerId);
```

Where,

- *layerId* is a *LayerId* (ael).

Example

```
decl context = de_get_current_design_context();
decl layerList = db_get_layerid_names(context);
while ( layerList != NULL )
{
   decl layerIdName = car( layerList );

   decl layerId = db_find_layerid_by_name(context, layerIdName);

   decl layerNumber = db_get_layer_number(layerId);
```

```
   decl purposeNumber = db_get_purpose_number(layerId);
   ...
   layerList = cdr( layerList );
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_get_purpose_number() (ael)

Where Used (ael)

Schematic, Layout

db_get_layer_process_role()

Returns the layer process role for a given layer. The layer process role is used by common design verification tasks such as design rule checking. It is also used to determine connectivity.

The layer process role returned is one of the following ADS layer process role AEL constants:

- **PROCESS_ROLE_NONE** - No process role. A shape on this layer does not connect electrically with other shapes.
- **PROCESS_ROLE_CONDUCTOR** - Layer contains conducting shapes.
- **PROCESS_ROLE_SEMICONDUCTOR** - Layer contains semi-conducting shapes. A shape on this layer does not connect electrically with other shapes.
- **PROCESS_ROLE_DIELECTRIC** - Layer contains dielectric shapes. A shape on this layer does not connect electrically with other shapes.
- **PROCESS_ROLE_CONDUCTOR_VIA** - Layer contains conducting via shapes.
- **PROCESS_ROLE_SEMICONDUCTOR_VIA** - Layer contains semi-conducting via shapes. A shape on this layer does not connect electrically with other shapes.
- **PROCESS_ROLE_DIELECTRIC_VIA** - Layer contains dielectric via shapes. A shape on this layer does not connect electrically with other shapes.
- **PROCESS_ROLE_CONDUCTOR_SLOT** - Layer contains conducting slot shapes. Slot layers are ignored by layout connectivity.
- **PROCESS_ROLE_SEMICONDUCTOR_SLOT** - Layer contains semi-conducting slot shapes. Slot layers are ignored by layout connectivity.
- **PROCESS_ROLE_DIELECTRIC_SLOT** - Layer contains dielectric slot shapes. Slot layers are ignored by layout connectivity.
- **PROCESS_ROLE_DRC** – Layer contains shapes representing DRC errors output by running the DRC tool.

Syntax

```
decl processRole = db_get_layer_process_role(context, layerNumber);
```

Where,

- *context* is a *DesignContext* (ael) of a design from the library whose technology the layer comes from.
- *layerNumber* is the number of the Layer to retrieve the layer process role for.

Example

```
decl context = de_get_current_design_context();
decl Metal1 = db_get_layerid(context, "Metal1", "drawing");
decl layerNum = db_get_layer_number(Metal1);
decl layerProcess = db_get_layer_process_role(context, layerNum);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Layer and LayerId Functions (ael)
db_set_layer_process_role() (ael)

***Where Used* (ael)**

Schematic, Layout

db_get_layerid_alpha()

Returns the alpha integer value for a given *LayerId* (ael).
The returned alpha is an integer alpha value within the range 0 to 255.

Note
The *LayerId* (ael) attribute value returned is the value as defined in the library plus any changes for that value that are stored in the workspace.

Syntax

```
decl alpha = db_get_layerid_alpha(libraryName|context, layerId);
```

Where,

- *libraryName*|*context* is:
  libraryName is the name of the library whose technology the *LayerId* (ael) comes from.
  or
  context is a *DesignContext* (ael) of a design from the library whose technology the *LayerId* (ael) comes from.
- *layerId* is a *LayerId* (ael).

Example

```
decl context = de_get_current_design_context();
decl Metal1 = db_get_layerid(context, "Metal1", "drawing");
decl alpha = db_get_layerid_alpha(context, Metal1);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_is_layerid_invisible() (ael)
db_is_layerid_protected() (ael)
db_set_layerid_invisible() (ael)
db_set_layerid_protected() (ael)
db_get_layerid_fill_mode() (ael)
db_get_layerid_fill_pattern() (ael)
db_get_layerid_line_style() (ael)
db_get_layerid_rgb() (ael)
db_set_layerid_alpha() (ael)
db_set_layerid_fill_mode() (ael)
db_set_layerid_fill_pattern() (ael)
db_set_layerid_line_style() (ael)
db_set_layerid_rgb() (ael)

Where Used (ael)

Schematic, Layout

db_get_layerid_fill_mode()

Returns the fill mode integer identifier for a given *LayerId* (ael).

There are 3 different fill mode integer values that can be returned. The 3 possible returned integer values are:
0 = outlined
1 = filled
2 = both (outlined and filled)

Note
The *LayerId* (ael) attribute value returned is the value as defined in the library plus any changes for that value that are stored in the workspace.

Syntax

```
decl fillMode = db_get_layerid_fill_mode(libraryName|context, layerId);
```

Where,

- *libraryName*|*context*:
  libraryName is the name of the library whose technology the *LayerId* (ael) comes from.
  or
  context is a *DesignContext* (ael) of a design from the library whose technology the *LayerId* (ael) comes from.
- *layerId* is a *LayerId* (ael).

Example

```
decl context = de_get_current_design_context();
decl Metal1 = db_get_layerid(context, "Metal1", "drawing");
decl fillMode = db_get_layerid_fill_mode(context, Metal1);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_is_layerid_invisible() (ael)
db_is_layerid_protected() (ael)
db_set_layerid_invisible() (ael)
db_set_layerid_protected() (ael)
db_get_layerid_alpha() (ael)
db_get_layerid_fill_pattern() (ael)
db_get_layerid_line_style() (ael)
db_get_layerid_rgb() (ael)
db_set_layerid_alpha() (ael)
db_set_layerid_fill_mode() (ael)
db_set_layerid_fill_pattern() (ael)
db_set_layerid_line_style() (ael)
db_set_layerid_rgb() (ael)

***Where Used* (ael)**

Schematic, Layout

db_get_layerid_fill_pattern()

Returns the fill pattern integer identifier for a given *LayerId* (ael).

Note
The *LayerId* (ael) attribute value returned is the value as defined in the library plus any changes for that value that are stored in the workspace.

Syntax

```
decl pattern = db_get_layerid_fill_pattern(libraryName|context, layerId);
```

Where,

- *libraryName*|*context* is:
  libraryName is the name of the library whose technology the *LayerId* (ael) comes from.
  or
  context is a *DesignContext* (ael) of a design from the library whose technology the *LayerId* (ael) comes from.
- *layerId* is a *LayerId* (ael).

Example

```
decl context = de_get_current_design_context();
decl Metal1 = db_get_layerid(context, "Metal1", "drawing");
decl fillPattern = db_get_layerid_fill_pattern(context, Metal1);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_is_layerid_invisible() (ael)
db_is_layerid_protected() (ael)
db_set_layerid_invisible() (ael)
db_set_layerid_protected() (ael)
db_get_layerid_alpha() (ael)
db_get_layerid_fill_mode() (ael)
db_get_layerid_line_style() (ael)
db_get_layerid_rgb() (ael)
db_set_layerid_alpha() (ael)
db_set_layerid_fill_mode() (ael)
db_set_layerid_fill_pattern() (ael)
db_set_layerid_line_style() (ael)
db_set_layerid_rgb() (ael)

***Where Used* (ael)**

Schematic, Layout

db_get_layerid_for_comp_name()

Returns the *LayerId* (ael) for the component name annotation used for component instances from a given *design context* (ael).

Syntax

```
decl layerId = db_get_layerid_for_comp_name(context);
```

Where,

- *context* is a *design context* (ael).

Example

```
decl context = de_get_current_design_context();
decl layerId = db_get_layerid_for_comp_name(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_get_layerid_for_inst_name() (ael)
db_get_layerid_for_parameters() (ael)
db_get_layerid_for_symbol_body() (ael)
db_get_layerid_for_symbol_text() (ael)
db_get_layerid_for_schematic_wires() (ael)

Where Used (ael)

Schematic, Layout

db_get_layerid_for_inst_name()

Returns the *LayerId* (ael) for the instance name annotation used for component instances from a given *design context* (ael).

Syntax

```
decl layerId = db_get_layerid_for_inst_name(context);
```

Where,

- *context* is a *design context* (ael).

Example

```
decl context = de_get_current_design_context();
decl layerId = db_get_layerid_for_inst_name(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_get_layerid_for_comp_name() (ael)
db_get_layerid_for_parameters() (ael)
db_get_layerid_for_symbol_body() (ael)
db_get_layerid_for_symbol_text() (ael)
db_get_layerid_for_schematic_wires() (ael)

Where Used (ael)

Schematic, Layout

db_get_layerid_for_parameters()

Returns the *LayerId* (ael) for the parameter annotation used for component instances from a given *design context* (ael).

Syntax

```
decl layerId = db_get_layerid_for_parameters(context);
```

Where,

- *context* is a *design context* (ael).

Example

```
decl context = de_get_current_design_context();
decl layerId = db_get_layerid_for_parameters(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_get_layerid_for_comp_name() (ael)
db_get_layerid_for_inst_name() (ael)
db_get_layerid_for_symbol_body() (ael)
db_get_layerid_for_symbol_text() (ael)
db_get_layerid_for_schematic_wires() (ael)

Where Used (ael)

Schematic, Layout

db_get_layerid_for_schematic_wires()

Returns the *LayerId* (ael) for the schematic wires from a given *design context* (ael).

Syntax

```
decl layerId = db_get_layerid_for_schematic_wires(context);
```

Where,

- *context* is a *design context* (ael).

Example

```
decl context = de_get_current_design_context();
decl layerId = db_get_layerid_for_schematic_wires(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_get_layerid_for_comp_name() (ael)
db_get_layerid_for_inst_name() (ael)
db_get_layerid_for_parameters() (ael)
db_get_layerid_for_symbol_body() (ael)
db_get_layerid_for_symbol_text() (ael)

Where Used (ael)

Schematic, Layout

db_get_layerid_for_symbol_body()

Returns the *LayerId* (ael) for the symbol body from a given *design context* (ael).

Syntax

```
decl layerId = db_get_layerid_for_symbol_body(context);
```

Where,

- *context* is a *design context* (ael).

Example

```
decl context = de_get_current_design_context();
decl layerId = db_get_layerid_for_symbol_body(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_get_layerid_for_comp_name() (ael)
db_get_layerid_for_inst_name() (ael)
db_get_layerid_for_parameters() (ael)
db_get_layerid_for_symbol_text() (ael)
db_get_layerid_for_schematic_wires() (ael)

Where Used (ael)

Schematic, Layout

db_get_layerid_for_symbol_text()

Returns the *LayerId* (ael) for the symbol text from a given *design context* (ael).

Syntax

```
decl layerId = db_get_layerid_for_symbol_text(context);
```

Where,

- *context* is a *design context* (ael).

Example

```
decl context = de_get_current_design_context();
decl layerId = db_get_layerid_for_symbol_text(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_get_layerid_for_comp_name() (ael)
db_get_layerid_for_inst_name() (ael)
db_get_layerid_for_parameters() (ael)
db_get_layerid_for_symbol_body() (ael)
db_get_layerid_for_schematic_wires() (ael)

Where Used (ael)

Schematic, Layout

db_get_layerid_index()

This function returns the *LayerId* (ael) integer index into the *design context* (ael)'s layer list given a *LayerId* (ael). The index value returned, is the same index for the given *LayerId* (ael) into the *LayerId* (ael) name list created from *db_get_layerid_names()* (ael).

Syntax

```
decl layerIndexNumber = db_get_layerid_index(context, layerId);
```

Where,

- *context* is a *design context* (ael).
- *layerId* is a *LayerId* (ael).

Example

```
decl context = de_get_current_design_context();
decl Metal1 = db_get_layerid(context, "Metal1", "drawing");
decl layerIndexNum = db_get_layerid_index(context, Metal1);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_get_layerid_names() (ael)

Where Used (ael)

Schematic, Layout

db_get_layerid_line_style()

Returns the line style integer identifier for a given *LayerId* (ael).

Here is the list of seven different line style integer values that can be returned:
0 = solid line
1 = dotted line
2 = double dotted line
3 = short dash line
4 = short dot dash line
5 = long dash line
6 = long dot dash line

Note
The *LayerId* (ael) attribute value returned is the value as defined in the library plus any changes for that value that are stored in the workspace.

Syntax

```
decl lineStyle = db_get_layerid_line_style(libraryName|context, layerId);
```

Where,

- *libraryName*|*context* is:
  libraryName is the name of the library whose technology the *LayerId* (ael) comes from.
  or
  context is a *DesignContext* (ael) of a design from the library whose technology the *LayerId* (ael) comes from.
- *layerId* is a *LayerId* (ael).

Example

```
decl context = de_get_current_design_context();
decl Metal1 = db_get_layerid(context, "Metal1", "drawing");
decl lineStyle = db_get_layerid_line_style(context, Metal1);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_is_layerid_invisible() (ael)
db_is_layerid_protected() (ael)
db_set_layerid_invisible() (ael)
db_set_layerid_protected() (ael)
db_get_layerid_alpha() (ael)
db_get_layerid_fill_mode() (ael)
db_get_layerid_fill_pattern() (ael)
db_get_layerid_rgb() (ael)
db_set_layerid_alpha() (ael)
db_set_layerid_fill_mode() (ael)
db_set_layerid_fill_pattern() (ael)
db_set_layerid_line_style() (ael)
db_set_layerid_rgb() (ael)

Where Used (ael)

Schematic, Layout

db_get_layerid_name()

Returns a *LayerId* (ael) name for a specified *LayerId* (ael) within a given *design context* (ael).

Note
A *LayerId* (ael) name is in the following format: "<layerName>:<purposeName>".

Syntax

```
decl name = db_get_layerid_name(context, layerId);
```

Where,

- *context* refers to the design context.
- *layerId* is the *LayerId* (ael).

Example

```
decl context = de_get_current_design_context();
decl Metal1 = db_get_layerid(context, "Metal1", "drawing");
decl layerIdName = db_get_layerid_name(context, Metal1);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_get_layerid_names() (ael)
db_find_layerid_by_name() (ael)

Where Used (ael)

Schematic, Layout

db_get_layerid_names()

This function returns an ordered list of *LayerId* (ael) names for a given *design context* (ael).

Notes
The order of the *LayerId* (ael) name list is determined by the technology's "Layer/Purpose Pair Drawing Order" setting.
A *LayerId* (ael) name is in the following format: "<layerName>:<purposeName>".

Syntax

```
decl namesList = db_get_layerid_names(context);
```

Where,

- *context* is a *design context* (ael).

Example

```
decl context = de_get_current_design_context();
decl layerList = db_get_layerid_names(context);
while ( layerList != NULL )
{
   decl layerIdName = car( layerList );
   decl layerId = db_find_layerid_by_name(context, layerIdName);
   decl layerNumber = db_get_layer_number(layerId);
   decl purposeNumber = db_get_purpose_number(layerId);
   ...
   layerList = cdr( layerList );
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_get_layerid_name() (ael)
db_get_layerid_index() (ael)

Where Used (ael)

Schematic, Layout

db_get_layerid_rgb()

Returns the RGB color integer values for a given *LayerId* (ael).

Note
The *LayerId* (ael) attribute value returned is the value as defined in the library plus any changes for that value that are stored in the workspace.

Syntax

```
decl rgb = db_get_layerid_rgb(libraryName|context, layerId, &red, &green,
&blue);
```

Where,

- *libraryName*|*context* is:
  libraryName is the name of the library whose technology the *LayerId* (ael) comes from.
  or
  context is a *DesignContext* (ael) of a design from the library whose technology the *LayerId* (ael) comes from.
- *layerId* is a *LayerId* (ael).
- *&red* is a return argument that will hold the returned red RGB integer value.
- *&green* is a return argument that will hold the returned green RGB integer value.
- *&blue* is a return argument that will hold the returned blue RGB integer value.

Example

```
decl context = de_get_current_design_context();
decl layerId = db_get_layerid(context, "Metal1", "drawing");
decl red, green, blue;
db_get_layerid_rgb(context, layerId, &ed, &reen, &lue);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_is_layerid_invisible() (ael)
db_is_layerid_protected() (ael)
db_set_layerid_invisible() (ael)
db_set_layerid_protected() (ael)
db_get_layerid_alpha() (ael)
db_get_layerid_fill_mode() (ael)
db_get_layerid_fill_pattern() (ael)
db_get_layerid_line_style() (ael)
db_set_layerid_alpha() (ael)
db_set_layerid_fill_mode() (ael)
db_set_layerid_fill_pattern() (ael)
db_set_layerid_line_style() (ael)

db_set_layerid_rgb() (ael)

Where Used (ael)

Schematic, Layout

db_get_layerid()

Returns a new *LayerId* (ael) object of a given layer/purpose pair, specified by their layer name and purpose name, of a given *design context* (ael). If the purpose name is not specified, the default "drawing" purpose, with name "drawing" and purpose number -1 will be used.

Syntax

```
decl layerid = db_get_layerid(context,layerName [, purposeName]);
```

Where,

- *context* is a *design context* (ael).
- *layerName* is the unique name of the layer.
- *purposeName* is the unique name of the purpose. It is optional and if not specified, the default "drawing" purpose, with name "drawing" and purpose number -1 is used.

Example

```
decl context = de_get_design_context(winInst);
decl Metal1 = db_get_layerid(context, "Metal1", "drawing");
decl Metal2 = db_get_layerid(context, "Metal2");  // Function uses default "drawing" purpose.

db_add_rectangle(context, Metal1, 0,0,10,10);
db_add_rectangle(context, Metal2, 10,10,20,20);
db_add_rectangle(context, Metal1, 2,20,30,30);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_layerid() (ael)

Where Used (ael)

Schematic, Layout

db_get_layout_layerid_name()

Returns the layout *LayerId* (ael) name for the given *design context* (ael) from the given *LayerId* (ael). The *LayerId* (ael) name returned is in the following format: "<layerName>:<purposeName>".

Notes
A *LayerId* (ael) name is in the following format: "<layerName>:<purposeName>".
This function returns the same value as *db_get_layerid_name()* (ael) since there is no difference between the set of symbol, schematic, and layout *LayerIds* (ael) in ADS.

Syntax

```
decl layerIdname = db_get_layout_layerid_name(context, layerId);
```

Where,

- *context* is a *design context* (ael).
- *layerId* is a *LayerId* (ael).

Example

```
decl context = de_get_current_design_context();
decl Metal1 = db_get_layerid(context, "Metal1", "drawing");
decl layerIdName = db_get_layout_layerid_name(context, Metal1);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_get_layerid_name() (ael)
db_get_layout_layerid_names() (ael)

Where Used (ael)

Schematic, Layout

db_get_layout_layerid_names()

Returns an ordered list of the layout LayerId names for the given *design context* (ael).

Notes
The order of the *LayerId* (ael) name list is determined by the technology's "Layer/Purpose Pair Drawing Order" setting.
This function returns the same value as *db_get_layerid_names()* (ael) since there is no difference between the set of symbol, schematic, and layout *LayerIds* (ael) in ADS.
The *LayerId* (ael) name is in the following format: "<layerName>:<purposeName>".

Syntax

```
decl layerIdnameList = db_get_layout_layerid_names(context);
```

Where,

- *context* is a *design context* (ael).

Example

```
decl context = de_get_current_design_context();
decl layerIdNameList = db_get_layout_layerid_names(context);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_get_layerid_names() (ael)
db_get_layout_layerid_name() (ael)

Where Used (ael)

Schematic, Layout

db_get_purpose_number()

Returns the purpose number for a given *LayerId* (ael).

Syntax

```
decl purposeNum = db_get_purpose_number(layerId);
```

Where,

- *layerId* is a *LayerId* (ael).

Example

```
decl context = de_get_current_design_context();
decl layerList = db_get_layerid_names(context);
while ( layerList != NULL )
{
   decl layerIdName = car( layerList );
   decl layerId = db_find_layerid_by_name(context, layerIdName);

   decl layerNumber = db_get_layer_number(layerId);
   decl purposeNumber = db_get_purpose_number(layerId);
   ...
   layerList = cdr( layerList );
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)

db_get_layer_number() (ael)

Where Used (ael)

Schematic, Layout

db_is_layerid_invisible()

Returns TRUE if the given *LayerId* (ael) is marked invisible, FALSE otherwise.

Note
The *LayerId* (ael) attribute value returned is the value as defined in the library plus any changes for that value that are stored in the workspace.

Syntax

```
decl invisible = db_is_layerid_invisible(libraryName|context, layerId);
```

Where,

- *libraryName*|*context* is:
  libraryName is the name of the library whose technology the *LayerId* (ael) comes from.
  or
  context is a *DesignContext* (ael) of a design from the library whose technology the *LayerId* (ael) comes from.
- *layerId* is a *LayerId* (ael).

Example

```
decl context = de_get_current_design_context();
decl layerId = db_get_layerid (context, "Metal2", "drawing");
decl isInVisible = db_is_layerid_invisible(context, layerId);
decl isProtected = db_is_layerid_protected(context, layerId);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_is_layerid_protected() (ael)
db_set_layerid_invisible() (ael)
db_set_layerid_protected() (ael)
db_get_layerid_alpha() (ael)
db_get_layerid_fill_mode() (ael)
db_get_layerid_fill_pattern() (ael)
db_get_layerid_line_style() (ael)
db_get_layerid_rgb() (ael)
db_set_layerid_alpha() (ael)
db_set_layerid_fill_mode() (ael)
db_set_layerid_fill_pattern() (ael)

db_set_layerid_line_style() (ael)
db_set_layerid_rgb() (ael)

Where Used (ael)

Schematic, Layout

db_is_layerid_protected()

Returns TRUE if the given *LayerId* (ael) is marked protected, FALSE otherwise.

Note
The *LayerId* (ael) attribute value returned is the value as defined in the library plus any changes for that value that are stored in the workspace.

Syntax

```
decl protect = db_is_layerid_protected(context, layerId);
```

Where,

- *context* is is a *DesignContext* (ael) of a design from the library whose technology the *LayerId* (ael) comes from.
- *layerId* is a *LayerId* (ael).

(Note that, unlike some similar functions, this function must take a context, not a library name. This is because artwork macro contexts override the protection status automatically, so without a context the wrong value would be returned.)

Example

```
decl context = de_get_current_design_context();
decl layerId = db_get_layerid (context, "Metal2", "drawing");
decl isInVisible = db_is_layerid_invisible(context, layerId);
decl isProtected = db_is_layerid_protected(context, layerId);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_is_layerid_invisible() (ael)
db_set_layerid_invisible() (ael)
db_set_layerid_protected() (ael)
db_get_layerid_alpha() (ael)
db_get_layerid_fill_mode() (ael)
db_get_layerid_fill_pattern() (ael)
db_get_layerid_line_style() (ael)
db_get_layerid_rgb() (ael)
db_set_layerid_alpha() (ael)
db_set_layerid_fill_mode() (ael)

db_set_layerid_fill_pattern() (ael)
db_set_layerid_line_style() (ael)
db_set_layerid_rgb() (ael)

Where Used (ael)

Schematic, Layout

db_layerid()

This function returns a new *LayerId* (ael) object of a given layer/purpose pair, specified by the layer number and optional purpose number.

Syntax

```
decl layerId = db_layerid(layerNum [, purposeNum]);
```

Where,

- *layerNum* is the unique integer number of the layer.
- *purposeNum* is the unique integer number of the purpose. If the purpose number is not specified, the default "drawing" purpose will be used that has purpose number -1.

Example

```
decl context = de_get_current_design_context();
decl layerId1 = db_layerid(layerNo1,purposeNo1);
decl layerId2 = db_layerid(layerNo2); // Function uses default "drawing" purpose number -1.
db_add_circle(context, layerId1, 10.0, 10.0, 25.5);
db_add_circle(context, layerId2, 50.0, 10.0, 25.5);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_layerid() (ael)
Layer and LayerId Functions (ael)
LayerId Overview (ael)

Where Used (ael)

Schematic, Layout

db_set_layer_process_role()

Sets the layer process role for a given layer. The layer process role is used by common design verification tasks such as design rule checking. It is also used to determine connectivity.

The layer process role to set must be one of the following ADS layer process role AEL constants:

- **PROCESS_ROLE_NONE** - No process role. A shape on this layer does not connect electrically with other shapes.
- **PROCESS_ROLE_CONDUCTOR** - Layer contains conducting shapes.
- **PROCESS_ROLE_SEMICONDUCTOR** - Layer contains semi-conducting shapes. A shape on this layer does not connect electrically with other shapes.
- **PROCESS_ROLE_DIELECTRIC** - Layer contains dielectric shapes. A shape on this layer does not connect electrically with other shapes.
- **PROCESS_ROLE_CONDUCTOR_VIA** - Layer contains conducting via shapes.
- **PROCESS_ROLE_SEMICONDUCTOR_VIA** - Layer contains semi-conducting via shapes. A shape on this layer does not connect electrically with other shapes.
- **PROCESS_ROLE_DIELECTRIC_VIA** - Layer contains dielectric via shapes. A shape on this layer does not connect electrically with other shapes.
- **PROCESS_ROLE_CONDUCTOR_SLOT** - Layer contains conducting slot shapes. Slot layers are ignored by layout connectivity.
- **PROCESS_ROLE_SEMICONDUCTOR_SLOT** - Layer contains semi-conducting slot shapes. Slot layers are ignored by layout connectivity.
- **PROCESS_ROLE_DIELECTRIC_SLOT** - Layer contains dielectric slot shapes. Slot layers are ignored by layout connectivity.
- **PROCESS_ROLE_DRC** – Layer contains shapes representing DRC errors output by running the DRC tool.

Syntax

```
db_set_layer_process_role(context, layerNumber, processRole);
```

Where,

- *context* is a *DesignContext* (ael) of a design from the library whose technology the layer comes from.
- *layerNumber* is the number of the Layer to set the layer process role for.
- *processRole* is the process role to set for the given layer.

Example

```
decl context = de_get_current_design_context();
decl Metal1 = db_get_layerid(context, "Metal1", "drawing");
decl layerNum = db_get_layer_number(Metal1);
db_set_layer_process_role(context, layerNum, PROCESS_ROLE_CONDUCTOR);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Layer and LayerId Functions (ael)
db_get_layer_process_role() (ael)

Where Used (ael)

Schematic, Layout

db_set_layerid_alpha()

Sets an alpha integer value (0-255) for a particular *LayerId* (ael) in a given library or a given *DesignContext* (ael)'s library's technology.
The acceptable alpha integer values are in range [0-255] positive.
Returns none.

Note
If a *LayerId* (ael) attribute value is different than the value stored in the library that defined the value, the difference is stored in the current workspace in a file such as <library name>.layerprf. The workspace *LayerId* (ael) attribute differences file is saved automatically after any value is changed.

Syntax

```
db_set_layerid_alpha(libraryName|context, layerId, alpha);
```

Where,

- *libraryName*|*context* is:
  libraryName is the name of the library whose technology the *LayerId* (ael) comes from.
  or
  context is a *DesignContext* (ael) of a design from the library whose technology the *LayerId* (ael) comes from.
- *layerId* is a *LayerId* (ael).
- *alpha* is an integer alpha value within the range 0 to 255.

Example

```
decl context = de_get_current_design_context();
decl libName = db_get_library_name(context);
decl Metal1 = db_get_layerid(context, "Metal1", "drawing");
db_set_layerid_alpha (libName, Metal1, 55);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_is_layerid_invisible() (ael)
db_is_layerid_protected() (ael)
db_set_layerid_invisible() (ael)
db_set_layerid_protected() (ael)
db_get_layerid_alpha() (ael)
db_get_layerid_fill_mode() (ael)
db_get_layerid_fill_pattern() (ael)
db_get_layerid_line_style() (ael)
db_get_layerid_rgb() (ael)
db_set_layerid_fill_mode() (ael)
db_set_layerid_fill_pattern() (ael)
db_set_layerid_line_style() (ael)
db_set_layerid_rgb() (ael)

***Where Used* (ael)**

Schematic, Layout

db_set_layerid_fill_mode()

Sets the fill mode for a particular *LayerId* (ael) in a given library or a given *DesignContext* (ael)'s library's technology.
Returns none.

There are 3 different fill mode integer values that a *LayerId* (ael) can have.
The 3 possible fill mode integer values are:
0 = outlined
1 = filled
2 = both (outlined and filled)

Note
If a *LayerId* (ael) attribute value is different than the value stored in the library that defined the value, the difference is stored in the current workspace in a file such as <library name>.layerprf. The workspace *LayerId* (ael) attribute differences file is saved automatically after any value is changed.

Syntax

```
db_set_layerid_fill_mode(libraryName|context, layerId, fillMode);
```

Where,

- *libraryName*|*context* is:
  libraryName is the name of the library whose technology the *LayerId* (ael) comes from.
  or
  context is a *DesignContext* (ael) of a design from the library whose technology the *LayerId* (ael) comes from.
- *layerId* is a *LayerId* (ael).
- *fillMode* is an integer fill mode type. The acceptable 3 fill mode integer choices are:
  0 = outlined
  1 = filled
  2 = both (outlined and filled)

Example

```
decl context = de_get_current_design_context();
decl libName = db_get_library_name(context);
decl Metal1 = db_get_layerid(context, "Metal1", "drawing");
db_set_layerid_fill_mode(libName, Metal1, 2);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)

db_is_layerid_invisible() (ael)
db_is_layerid_protected() (ael)
db_set_layerid_invisible() (ael)
db_set_layerid_protected() (ael)
db_get_layerid_alpha() (ael)
db_get_layerid_fill_mode() (ael)
db_get_layerid_fill_pattern() (ael)
db_get_layerid_line_style() (ael)
db_get_layerid_rgb() (ael)
db_set_layerid_alpha() (ael)
db_set_layerid_fill_pattern() (ael)
db_set_layerid_line_style() (ael)
db_set_layerid_rgb() (ael)

Where Used (ael)

Schematic, Layout

db_set_layerid_fill_pattern()

Sets an integer fill pattern for a particular *LayerId* (ael) in a given library or a given *DesignContext* (ael)'s library's technology.
Returns none.

Note
If a *LayerId* (ael) attribute value is different than the value stored in the library that defined the value, the difference is stored in the current workspace in a file such as <library name>.layerprf. The workspace *LayerId* (ael) attribute differences file is saved automatically after any value is changed.

Syntax

```
db_set_layerid_fill_pattern(libraryName|context, layerId, fillPattern);
```

Where,

- *libraryName*|*context* is:
  libraryName is the name of the library whose technology the *LayerId* (ael) comes from.
  or
  context is a *DesignContext* (ael) of a design from the library whose technology the *LayerId* (ael) comes from.
- *layerId* is a *LayerId* (ael).
- *fillPattern* is an integer fill pattern value.

Example

```
decl context = de_get_current_design_context();
decl libName = db_get_library_name(context);
decl Metal1 = db_get_layerid(context, "Metal1", "drawing");
db_set_layerid_fill_pattern(libName, Metal1, 3);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_is_layerid_invisible() (ael)
db_is_layerid_protected() (ael)
db_set_layerid_invisible() (ael)
db_set_layerid_protected() (ael)
db_get_layerid_alpha() (ael)
db_get_layerid_fill_mode() (ael)
db_get_layerid_fill_pattern() (ael)
db_get_layerid_line_style() (ael)
db_get_layerid_rgb() (ael)
db_set_layerid_alpha() (ael)
db_set_layerid_fill_mode() (ael)
db_set_layerid_line_style() (ael)
db_set_layerid_rgb() (ael)

***Where Used* (ael)**

Schematic, Layout

db_set_layerid_invisible()

Function to mark a *LayerId* (ael) to be invisible or visible.
Specify the invisible argument as TRUE to set the *LayerId* (ael) to be invisible, specify the invisible argument as FALSE to set the *LayerId* (ael) to be visible.

Note
If a *LayerId* (ael) attribute value is different than the value stored in the library that defined the value, the difference is stored in the current workspace in a file such as <library name>.layerprf. The workspace *LayerId* (ael) attribute differences file is saved automatically after any value is changed.

Syntax

```
db_set_layerid_invisible(libraryName|context, layerId, invisible);
```

Where,

- *libraryName*|*context* is:
  libraryName is the name of the library whose technology the *LayerId* (ael) comes from.
  or
  context is a *DesignContext* (ael) of a design from the library whose technology the *LayerId* (ael) comes from.
- *layerId* is a *LayerId* (ael).
- *invisible* is a boolean TRUE or FALSE value. You can specify it TRUE to set the *LayerId* (ael) to be invisible and FALSE to set the *LayerId* (ael) to be visible.

Example

```
decl context = de_get_current_design_context();
decl layerId = db_get_layerid (context, "Metal2", "drawing");
db_set_layerid_invisible(context, layerId, TRUE);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_is_layerid_invisible() (ael)
db_is_layerid_protected() (ael)
db_set_layerid_protected() (ael)
db_get_layerid_alpha() (ael)
db_get_layerid_fill_mode() (ael)
db_get_layerid_fill_pattern() (ael)
db_get_layerid_line_style() (ael)
db_get_layerid_rgb() (ael)
db_set_layerid_alpha() (ael)
db_set_layerid_fill_mode() (ael)
db_set_layerid_fill_pattern() (ael)
db_set_layerid_line_style() (ael)
db_set_layerid_rgb() (ael)

Where Used (ael)

Schematic, Layout

db_set_layerid_line_style()

Sets the line style for a particular *LayerId* (ael) in a given library or a given *DesignContext* (ael)'s library's technology.
Returns none.

There are 7 different line style integer values that a *LayerId* (ael) can have.
The 7 possible *LayerId* (ael) line style integer values are:

0 = solid line
1 = dotted line
2 = double dotted line
3 = short dash line
4 = short dot dash line
5 = long dash line
6 = long dot dash line

Note
If a *LayerId* (ael) attribute value is different than the value stored in the library that defined the value, the difference is stored in the current workspace in a file such as <library name>.layerprf. The workspace *LayerId* (ael) attribute differences file is saved automatically after any value is changed.

Syntax

```
db_set_layerid_line_style(libraryName|context, layerId, lineStyle);
```

Where,

- *libraryName*|*context* is:

libraryName is the name of the library whose technology the *LayerId* (ael) comes from.
or
context is a *DesignContext* (ael) of a design from the library whose technology the *LayerId* (ael) comes from.

- *layerId* is a *LayerId* (ael).
- *lineStyle* is an integer line style type. The acceptable 7 line style integer choices are:
  0 = solid line
  1 = dotted line
  2 = double dotted line
  3 = short dash line
  4 = short dot dash line
  5 = long dash line
  6 = long dot dash line

Example

```
decl context = de_get_current_design_context();
decl libName = db_get_library_name(context);
decl Metal1 = db_get_layerid(context, "Metal1", "drawing");
db_set_layerid_line_style(libName, Metal1, 0);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_is_layerid_invisible() (ael)
db_is_layerid_protected() (ael)
db_set_layerid_invisible() (ael)
db_set_layerid_protected() (ael)
db_get_layerid_alpha() (ael)
db_get_layerid_fill_mode() (ael)
db_get_layerid_fill_pattern() (ael)
db_get_layerid_line_style() (ael)
db_get_layerid_rgb() (ael)
db_set_layerid_alpha() (ael)
db_set_layerid_fill_mode() (ael)
db_set_layerid_fill_pattern() (ael)
db_set_layerid_rgb() (ael)

Where Used (ael)

Schematic, Layout

db_set_layerid_protected()

Function to mark a *LayerId* (ael) to be protected or unprotected.
Specify the protected argument as TRUE to set the *LayerId* (ael) to be protected, specify the protected argument as FALSE to set the *LayerId* (ael) to be unprotected.

Protected *LayerId* (ael) are non-editable.

Note
If a *LayerId* (ael) attribute value is different than the value stored in the library that defined the value, the difference is stored in the current workspace in a file such as <library name>.layerprf. The workspace *LayerId* (ael) attribute differences file is saved automatically after any value is changed.

Syntax

```
db_set_layerid_protected(libraryName|context, layerId, protected);
```

Where,

- *libraryName*|*context* is:
  libraryName is the name of the library whose technology the *LayerId* (ael) comes from.
  or
  context is a *DesignContext* (ael) of a design from the library whose technology the *LayerId* (ael) comes from.
- *layerId* is a *LayerId* (ael).
- *protected* is a boolean TRUE or FALSE value. Specify the protected argument as TRUE to set the *LayerId* (ael) to be protected and specify it as FALSE to set the *LayerId* (ael) to be unprotected.

Example

```
decl context = de_get_current_design_context();
decl layerId = db_get_layerid (context, "Metal2", "drawing");
if (db_is_layerid_protected(context, layerId))
   db_set_layerid_protected(context, layerId, FALSE);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_is_layerid_invisible() (ael)
db_is_layerid_protected() (ael)
db_set_layerid_invisible() (ael)
db_get_layerid_alpha() (ael)
db_get_layerid_fill_mode() (ael)
db_get_layerid_fill_pattern() (ael)
db_get_layerid_line_style() (ael)
db_get_layerid_rgb() (ael)
db_set_layerid_alpha() (ael)
db_set_layerid_fill_mode() (ael)
db_set_layerid_fill_pattern() (ael)
db_set_layerid_line_style() (ael)
db_set_layerid_rgb() (ael)

Where Used (ael)

Schematic, Layout

db_set_layerid_rgb()

Sets a RGB color for a particular *LayerId* (ael) in a given library or a given *DesignContext* (ael)'s library's technology.
The red, green, and blue arguments are RGB color integer values between 0 and 255.
Returns none.

Note
If a *LayerId* (ael) attribute value is different than the value stored in the library that defined the value, the difference is stored in the current workspace in a file such as <library name>.layerprf. The workspace *LayerId* (ael) attribute differences file is saved automatically after any value is changed.

Syntax

```
db_set_layerid_rgb(libraryName|context, layerId, red, green, blue);
```

Where,

- *libraryName*|*context* is:
  libraryName is the name of the library whose technology the *LayerId* (ael) comes from.
  or
  context is a *DesignContext* (ael) of a design from the library whose technology the *LayerId* (ael) comes from.
- *layerId* is a *LayerId* (ael).
- *red* is the red RGB integer value to set the *LayerId* (ael) RGB value with.
- *green* is the green RGB integer value to set the *LayerId* (ael) RGB value with.
- *blue* is the blue RGB integer value to set the *LayerId* (ael) RGB value with.

Example

```
decl context = de_get_current_design_context();
decl libName = db_get_library_name(context);
decl Metal1 = db_get_layerid(context, "Metal1", "drawing");
// Set the Metal1 layer color Purple.
db_set_layerid_rgb(libName, Metal1, 128, 0, 128);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

LayerId Overview (ael)
Layer and LayerId Functions (ael)
db_is_layerid_invisible() (ael)
db_is_layerid_protected() (ael)
db_set_layerid_invisible() (ael)
db_set_layerid_protected() (ael)
db_get_layerid_alpha() (ael)
db_get_layerid_fill_mode() (ael)
db_get_layerid_fill_pattern() (ael)

db_get_layerid_line_style() (ael)
db_get_layerid_rgb() (ael)
db_set_layerid_alpha() (ael)
db_set_layerid_fill_mode() (ael)
db_set_layerid_fill_pattern() (ael)
db_set_layerid_line_style() (ael)

Where Used (ael)

Schematic, Layout

Layer Identifier (LayerId) Overview

A Layer Identifier (LayerId) represents a layer and purpose pair. A layer/purpose pair is a combination of one layer and one purpose.

Layer

A layer has a name and a number. A layer normally represents a physical layer in a design process, and a layer's name and its number must be unique.

Purpose

A purpose has a name and a number. A purpose can be thought of as a subdivision of a layer. A purpose's name and its number must be unique. There are some predefined purposes in ADS. Normally most layout drawing will use a predefined purpose named "drawing".

You can create a new purposes for specific reasons. Some examples:

- Different shapes on the same layer require different DRC rules. So different rules can apply to the different purposes on that layer.
- You can distinguish between pads and other shapes on the same layer. This could be for DRC or just to show them in a different color or fill type.

Layer/Purpose Pair

A layer/purpose pair is a combination of one layer and one purpose.

Various attributes are associated with layer/purpose pairs.
The attributes are:

- color
- alpha
- fill pattern
- selectable
- visible
- shape display: (outline, filled, both)
- line style

LayerId Details

The AEL LayerId data structure type holds a layer and purpose number.

The types of information that a LayerId AEL object contains include:

- the layer number.
- the purpose number.

Layer/Purpose pair information accessible by a LayerId

These are the different data fields stored within an AEL LayerId Object.

Layer number

The layer number is a 32-bit integer.

Purpose number
The purpose number is a 32-bit integer.

Getting LayerIds

You can get a LayerId in AEL, with the *db_get_layerid()* (ael), *db_find_layerid_by_name()* (ael), and *db_layerid()* (ael) functions.

LayerId Functions

LayerId Functions (ael)

Examples

Example of Drawing Shapes on a particular LayerId

Example of drawing rectangles on particular layer/purpose pairs.

```
decl context = de_get_design_context(winInst);
decl Metal1 = db_get_layerid(context, "Metal1", "drawing");
decl Metal2 = db_get_layerid(context, "Metal2");

db_add_rectangle(context, Metal1, 0,0,10,10);
db_add_rectangle(context, Metal2, 10,10,20,20);
db_add_rectangle(context, Metal1, 2,20,30,30);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

List Management Functions

This section describes each List Management function in detail. The functions are listed in alphabetical order.

append()

Appends a new list to the end of an existing list. Lists are created with the *list()* (ael) function. Returns a new list with the appended items.

Syntax:

```
append(list, newList);
```

where

list is a list created with the *list()* (ael) function.

newList is a new list of items to append to the first list.

Example:

```
decl mylist;
mylist = list("one", "two");
mylist = append(mylist, list("three"));
//The result is mylist = ("one", "two", "three")
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Get Layers Protected Status
Reverse List
Unique List

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

car()

Returns the first item from a given list. Does not change the list, is passed as a parameter.

Syntax:

```
car(list);
```

where

list is a list created with the *list()* (ael) function.

Example:

The example returns the string item *a* (see comment).

```
decl c;
c = car(list("a", "b", "c"));      //the result is c="a"
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.

Get Layer Name
Get Layer Number
Set Visible Layer

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

cdr()

Returns the remainder of the list, after the first item is removed.

Syntax:

```
cdr(list);
```

where

list is a list created with the *list()* (ael) function.

Example:

The example returns the list containing one item: *there*.

```
decl shortList;
shortList = cdr(list("hello", "there"));
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Get Layer Name
Get Layer Number
Unique List

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

cons()

Constructs a list cell from the member value to be returned by *car()* (ael) and the next element value to be returned by *cdr()* (ael). The first argument can be any value, and the second argument is normally a list, or NULL. If the second argument is a list, then the value returned is a list with new member inserted at the beginning. If the second argument is NULL, then the value returned is a list with one member. If the second argument is neither a list nor NULL, then an improper list is returned. Returns a list cell; that is, returns a list with one element.

Syntax:

cons(carValue, cdrValue);

where

carValue is a member value.

cdrValue is the next element value.

Example:

```
x = cons( 5, NULL ); //the result is x=(5)
```

Also:

```
x = cons( 5, list(6,7) ); //the result is x=(5,6,7)
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

delete_nth()

Deletes an item in a list, given an integer index into the list and an existing list. Returns a new list with a deleted item. If index = list length, then NULL is returned.

Syntax:

```
delete_nth(index, list);
```

where

index is an integer index into list. If index = list length, then NULL is returned.

list is the list to delete an item from.

Example:

```
decl a = list(1,2,3,4,5);
decl b;
b = delete_nth(0,a);
fputs(stdout, identify_value(b));        // deletes the 1
b = delete_nth(1,a);
fputs(stdout, identify_value(b));        // deletes the 2
b = delete_nth(4,a);
fputs(stdout, identify_value(b));        // deletes the 5
b = delete_nth(5,a);
fputs(stdout, identify_value(b));        // returns NULL
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

insert()

Returns a new list with an inserted item, given an existing list and item to insert into the beginning of the list.

Syntax:

```
insert(list, listItem);
```

where

list is the list to insert the item into.

listItem is the item to insert.

Example:

```
decl list1;
list1 = list(1, 2, 3);
// The result is list1 = 1, 2, 3
list1 = insert(list1, 0);
// The result is list1 = 0, 1, 2, 3
========================================
To produce an empty list:
insert_nth(0, list(), 10); //produces list(10)
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

insert_nth()

Returns a new list with an inserted item, given an index into the list, an existing list and an item to insert into the list.

Syntax:

```
insert_nth(index, list, listItem);
```

where

index is an integer index into list. If index = list length, then NULL is returned.

list is the list to insert the item into.

listItem is the item to insert.

Example:

```
decl x = list(1,2,3,4,5);
decl y;
y = insert_nth(3,x,10);
fputs(stdout, identify_value(y)); // output: list(1,2,3,10,4,5)
```

```
fputs(stdout, identify_value(x)); // output: list(1,2,3,4,5)
y = insert_nth(0,x,10);
fputs(stdout, identify_value(y)); // output: list(10,1,2,3,4,5)
y = insert_nth(5,x,10);
fputs(stdout, identify_value(y)); // output: list(1,2,3,4,5,10)
y = insert_nth(7,x,10);
fputs(stdout, identify_value(y)); // output: NULL
y = insert_nth(-1,x,10);
fputs(stdout, identify_value(y)); // output: NULL
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

list()

Creates a list of items. The list can be composed of any type of item. Returns a list of given items.

Syntax:

```
list(item1, item2,..., itemN);
```

where

itemN is an item.

Example:

```
decl list1;
list1 = list(1, 34.5, 5, 7, "hello", 3+4i);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Reverse List
Set Parameter Value
Unique List

Where Used: **(ael)**

Measurement Expressions (Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name list() is used for more than one type of expression. If using *Advanced Design System* see **Simulator Expressions** > **Data Access Functions For Simulator Expressions** > **list() Expression** for comparison. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

listlen()

Returns an integer representing the length of a list; that is, the number of items in the list.

Syntax:

```
listlen(list);
```

where

list is a list of items created with the list function.

Example:

The example returns the integer 3:

```
decl len;
len = listlen(list(1,2,3));
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Get File Size
Number Open Windows
Reverse List

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

member()

Tests whether an item is a member of a list. Returns a list consisting of the original list from the member on or NULL.

Syntax:

```
member(m, list);
```

where

m is the item you are testing for.

list is a list of items.

Example:

In this example, member("def", L) returns the list ("def", "ghi").

```
decl L;
L=list("abc", "def", "ghi");
if(member("def", L))
   fputs(stderr, "of is");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Delete Layout Hierarchy
Set Layer File - Traverses Hierarchy
Unique List

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

nth()

Takes a list and an integer index into the list. Returns a copy of the corresponding *nth* list item unless that item is a list itself. If the *nth* item is a list, then a pointer to that list is returned-not a copy. Returns a NULL if the item number exceeds the list length.

Syntax:

```
nth(n, list);
```

where

n is an integer greater than or equal to 0; represents position of item in list to retrieve.

list is the list of items.

Example:

```
decl sixth;
sixth=nth(5, list(0, 1, 2, 3, 4, 5, 6, 7, 8));
// returns the number 5 (6th position)
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Get File Permissions
Get File Time Stamp
Get Registry Value

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

nthcdr()

Takes a list, and an integer index into the list. Returns a list consisting of the members of the original list from the *nth* item on.

Syntax:

```
nthcdr(n, list);
```

where

n is the ndex (integer ≥ 0) into list.

list is a list of items.

Example:

```
decl j, L;
L = list("a", "b", "c");
j=nthcdr(1, L);
// j ==list("b", "c")
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic,

Layout, Simulation, GUI

remov()

Returns a new list with all the items copied from the original list except for the item specified in the first parameter.

Syntax:

```
remov(item, alist);
```

where

item is a member of a list to be deleted from the new list.

alist is the original list that will be copied, except for item.

Example:

```
decl a,b;
a = list(1,5.8,4,10);
b = remov(4, a);        // returns the list (1,5.8,10)
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

repla()

Replaces an item in a list. Returns the original list, modified with the replaced item. If a list *a* is assigned to list *b* , it is assigned a pointer in a sense. However, if *a* is modified using repla(), then *a* and *b* continue to point to the same list. If *a* is modified using any function other than repla(), then a copy is made and *a* and *b* no longer point to the same list.

Syntax:

```
repla(list, item, index);
```

where

list is a list created with the list function.

item is the new list item.

index is the position in the list of the item to replace, numbered from 0 to n−1.

Example:

```
decl list1;
list1 = list(1, 2, 3);
repla(list1, 5, 2);
// The result is list1 = 1, 2, 5
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

sort_list()

Returns a new list with members of a given list in sorted order. If the compare function is not given and the list is homogeneous, a built-in function will be applied if:

- List has only integer values, or
- List has only real values, or
- List has only string values, or
- List has only list values where each list is comparable on a value by value basis (has the same structure of integer, real, string, and list members).

For other cases, a compare() function must be supplied. The compare() function is called with two member values as the argument and must return a value greater than, equal to, or less than zero, depending on whether the first member is greater than, equal to, or less than the second member. If the first argument is not a list, the function fails and reports an error.

Syntax:

sort_list(list, [, func]);

where

list is a list to be sorted.

func is optional. Address of function to perform member comparison.

Example:

This example redefines l to the list 1, 2, 3, 4, 9.

```
decl l = list(2, 4, 1, 3, 9);
defun my_compare(first, second)
{
   decl neg;
```

```
   if (first < second)
       neg = -1;
   else if (first > second)
       neg = 1;
   else
       neg = 0;
   return (neg);
}
l = sort_list(l, my_compare);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Knowledge center website. You will need to register at this site with a valid support contract to download an example file.
Netlist with Node Names

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Math Functions For AEL

This section describes each Math function in detail. The functions are listed in alphabetical order.

Note
If using *Advanced Design System*, for math functions used in simulations, refer to **Simulator Expressions**.

A	
abs() Function (ael) *acos() Function* (ael) *acosh() Function* (ael) *acot() Function* (ael) *acoth() Function* (ael)	*asin() Function* (ael) *asinh() Function* (ael) *atan() Function* (ael) *atan2() Function* (ael) *atanh() Function* (ael)
C,D,E	
ceil() Function (ael) *chr() Function* (ael) *cint() Function* (ael) *cmplx() Function* (ael) *conj() Function* (ael) *convBin() Function* (ael) *convHex() Function* (ael) *convOct() Function* (ael)	*cos() Function* (ael) *cosh() Function* (ael) *cot() Function* (ael) *coth() Function* (ael) *dB() Function* (ael) *dBm() Function* (ael) *deg() Function* (ael) *exp() Function* (ael)
F,I,L	
fix() Function (ael) *float() Function* (ael) *floor() Function* (ael) *im() Function* (ael) *imag() Function* (ael)	*int() Function* (ael) *ln() Function* (ael) *log() Function* (ael) *log10() Function* (ael)
M,N,P	
mag() Function (ael) *max2() Function* (ael) *min2() Function* (ael) *num() Function* (ael) *phase() Function* (ael)	*phasedeg() Function* (ael) *phaserad() Function* (ael) *polar() Function* (ael) *pow() Function* (ael)
R,S,T,X	
rad() Function (ael) *re() Function* (ael) *real() Function* (ael) *round() Function* (ael) *sgn() Function* (ael) *sin() Function* (ael) *sinc() Function* (ael)	*sinh() Function* (ael) *sqrt() Function* (ael) *step() Function* (ael) *tan() Function* (ael) *tanh() Function* (ael) *xor() Function* (ael)

abs()

Returns the absolute value of a real number or an integer. In the case of a complex number, the abs function:

- accepts one complex argument.
- returns a positive real number.
- returns the magnitude of its complex argument.

Syntax:

```
abs(num);
```

where

num is any number.

Example:

```
decl a;
a = abs(-12.3);        // a=12.3
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Find Close Vertexes

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name abs() is used for more than one type of expression. For comparison, see the Measurement Expression *abs() Measurement* (expmeas) if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **abs() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

acos()

Returns the inverse cosine, in radians, of a real number or complex number.

Syntax:

```
acos(realNum);
```

where

realNum is a real or complex number.

Example:

```
decl a;
a = acos(0.5);        // a=1.0471975511966
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name acos() is used for more than one type of expression. For comparison, see the Measurement Expression *acos() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **acos() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

acosh()

Returns the inverse hyperbolic cosine of an integer, real, or complex number.

Syntax:

```
acosh(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl a;
a = acosh(1.5);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name acosh() is used for more than one type of expression. For comparison, see the Measurement Expression *acosh() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **acosh() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

acot()

Returns the inverse cotangent of an integer, real, or complex number.

Syntax:

```
acot(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl a;
a = acot(1.5);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name acot() is used for more than one type of expression. For comparison, see the Measurement Expression *acot() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

acoth()

Returns the inverse hyperbolic cotangent of an integer, real, or complex number.

Syntax:

```
acoth(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl a;
a = acoth(1.5);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name acoth() is used for more than one type of expression. For comparison, see the Measurement Expression *acoth() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

asin()

Returns the inverse sine, in radians, of a real number, an integer, or complex number.

Syntax:

```
asin(realNum);
```

where

realNum is a real number, an integer, or complex number.

Example:

```
decl a;
a = asin(0.5);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name asin() is used for more than one type of expression. For comparison, see the Measurement Expression *asin() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **asin() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

asinh()

Returns the inverse hyperbolic sine of an integer, real, or complex number.

Syntax:

```
asinh(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl a;
a = asinh(.5);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

atan2()

Returns the arc tangent of the rectangular coordinates y and x.

Syntax:

```
atan2(y,x);
```

where

realNum is a real number, an integer, or complex number.

Example:

```
decl a;
a = atan2(10,15);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name atan2() is used for more than one type of expression. For comparison, see the Measurement Expression *atan2() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **atan2() Expression**. For more information on the different expression types and the contexts in which they are used, see [Duplicated Expression Names].

atan()

Returns the inverse tangent, in radians, of a real number, ±Π/2, an integer, or complex number.

Syntax:

```
atan(realNum);
```

where

realNum is a real number, an integer, or complex number.

Example:

```
decl b;
b = atan(0.5);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name atan() is used for more than one type of expression. For comparison, see the Measurement Expression *atan() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **atan() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

atanh()

Returns the inverse hyperbolic tangent of an integer, real, or complex number.

Syntax:

```
atanh(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl a;
a = atanh(.5);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name atanh() is used for more than one type of expression. For comparison, see the Measurement Expression *atanh() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **atanh() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

ceil()

Given a real number, returns the smallest integer not less than its argument; that is, its argument rounded to the next highest number.

Syntax:

```
ceil(realVal);
```

where

realVal is a real number.

Example:

```
a = ceil(5.27); // a=6
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name ceil() is used for more than one type of expression. For comparison, see the Measurement Expression *ceil() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **ceil() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

chr()

Returns the character representation of an integer.

Syntax:

```
chr(code);
```

where

code is a valid ASCII string representing a character.

Example:

The example returns the character @.

```
decl aChar;
aChar = chr\(64\);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name chr() is used for more than one type of expression. For comparison, see the Measurement Expression *chr() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

cint()

Given a noninteger real number, returns a rounded integer value.

Syntax:

```
cint(realVal);
```

where

realVal is a real number.

Example:

The example returns the integer *46* .

```
decl d;
d = cint(5.27);        // d=5
```

Also:

```
decl d;
d = cint(5.6);        // d=6
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Round Number

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name cint() is used for more than one type of expression. For comparison, see the Measurement Expression *cint() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

cmplx()

Given two real numbers representing the real and imaginary components of a complex number, returns a complex number.

Note
Use the *real* and *imag* functions to retrieve the real and imaginary components, respectively. The basic math functions operate on complex numbers.

Syntax:

```
cmplx(realPart, imagPart);
```

where

realPart is the real component of the number.

imagPart is the imaginary component.

Example:

```
decl cmpx;
cmpx = cmplx(10.0, 3);        // returns the complex number (10.0+3i)
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name cmplx() is used for more than one type of expression. For comparison, see the Measurement Expression *cmplx() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

conj()

Returns the conjugate of a complex number.

Syntax:

```
conj(num);
```

where

num is a complex number.

Example:

```
decl a;
a = conj(3 + 4i);        // returns the complex number a=3-4i
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name conj() is used for more than one type of expression. For comparison, see the Measurement Expression *conj() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **conj() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

convBin()

Returns a binary string of an integer with n-digits.

Syntax:

```
convBin(val, num);
```

where

val is any integer to be converted to binary string.

num is an integer to specify the number of digits in the binary string.

Example:

```
decl a;
a = convBin(1064, 8);        // returns a="00101000"
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name convBin() is used for more than one type of expression. For comparison, see the Measurement Expression *convBin() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

convHex()

Returns a hexidecimal string of an integer with n-digits.

Syntax:

```
convHex(val, num);
```

where

val is any integer to be converted to hexidecimal string.

num is an integer to specify the number of digits in the hexidecimal string.

Example:

```
decl a;
a = convHex(1064, 8);        // returns a="00000428"
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name convHex() is used for more than one type of expression. For comparison, see the Measurement Expression *convHex() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

convOct()

Returns an octal string of an integer with n-digits

Syntax:

```
convOct(val, num);
```

where

val is any integer to be converted to octal string.

num is an integer to specify the number of digits in the octal string.

Example:

```
decl a;
a = convOct(1064, 8);        // returns a="00002050"
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name convOct() is used for more than one type of expression. For comparison, see the Measurement Expression *convOct() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

cos()

Returns the cosine of a real number (in radians).

Syntax:

```
cos(realNum);
```

where

realNum is a real number, in radians.

Example:

```
decl c;
c = cos(PI/4);        // where PI is the AEL constant 3.1415926535898
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name cos() is used for more than one type of expression. For comparison, see the Measurement Expression *cos() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **cos() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

cosh()

Returns the hyperbolic cosine of an integer, real, or complex number.

Syntax:

```
cosh(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl a;
a = cosh\(1.5\);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name cosh() is used for more than one type of expression. For comparison, see the Measurement Expression *cosh() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **cosh() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

cot()

Returns the cotangent of an integer, real, or complex number.

Syntax:

```
cot(num);
```

where
num is any integer, real, or complex number.

Example:

```
decl a;
a = cot(1.5);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic,

Layout, Simulation, GUI

Note
The function name cot() is used for more than one type of expression. For comparison, see the Measurement Expression *cot() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **cot() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

coth()

Returns the hyperbolic cotangent of an integer, real, or complex number.

Syntax:

```
coth(num);
```

where
num is any integer, real, or complex number.

Example:

```
decl a;
a = coth(1.5);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name coth() is used for more than one type of expression. For comparison, see the Measurement Expression *coth() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **coth() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

dB()

Converts a voltage ratio to decibels. Returns a real number that represents a voltage ratio in decibels.

Syntax:

```
dB(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl a;
a = dB(1.5);        // a=3.5218251811136
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name dB() is used for more than one type of expression. For comparison, see the Measurement Expression *db() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **db() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

dBm()

Converts a voltage to decibels referenced to milliwatt. Returns a real number that represents a voltage ratio in milliwatt.

Syntax:

```
dBm(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl a;
a = dBm(1.5);        // a=13.521825181114
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name dbm() is used for more than one type of expression. For comparison, see the Measurement Expression *dbm() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **dbm() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

deg()

Converts an angle from radians to degrees. Returns a real number or NULL if an error occurs.

Syntax:

```
deg(num);
```

where

num is any integer or real number that represents an angle in radians.

Example:

```
decl a;
a = deg(1.5);         // a=85.943669269623
```

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name deg() is used for more than one type of expression. For comparison, see the Measurement Expression *deg() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **deg() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

exp()

Given an integer or real number as an exponent, returns e (~2.7183) raised to that exponent.

Syntax:

```
exp(realVal);
```

where

realVal is the exponent of *e* , where *e* is the AEL constant 2.718281828459.

Example:

```
decl b;
b = exp(10);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name exp() is used for more than one type of expression. For comparison, see the Measurement Expression *exp() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **exp() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

fix()

Takes a real number argument, truncates it, and returns an integer value.

Syntax:

```
fix(realVal);
```

where

realVal is a real number.

Example:

```
decl b;
b = fix(1.1);         // returns 1
b = fix(1.9);         // returns 1
b = fix(-1.1);         // returns -1
b = fix(-1.9);         // returns -1
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name fix() is used for more than one type of expression. For comparison, see the Measurement Expression *fix() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

float()

Converts an integer to a floating decimal number. Returns a real (floating-point) number.

Note
To convert a real number to an integer, use *int*.

Syntax:

```
float(intNum);
```

where

intNum is the integer to convert.

Example:

```
decl r;
r = float(10);        // r=10.0 or r is the floating point number 10
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
PI Attenuator

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name float() is used for more than one type of expression. For comparison, see the Measurement Expression *float() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

floor()

Returns the largest integer not more than its argument from a real number.

Syntax:

```
floor(realVal);
```

where

realVal is a real number to be converted to an integer.

Example:

```
decl d;
d = floor(1.1);         // d= 1
d = floor(1.9);         // d= 1
d = floor(-1.1);         // d= -2
d = floor(-1.9);         // d= -2
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name floor() is used for more than one type of expression. For comparison, see the Measurement Expression *floor() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **floor() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

im()

Returns a real value, the imaginary component of a complex number.

Syntax:

```
im(complexNum);
```

where

complexNum is a complex number.

Example:

```
decl rv;
rv = im(1-1);         //rv=0
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name im() is used for more than one type of expression. For comparison, see the Measurement Expression *im() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

imag()

Returns a real value, the imaginary component of a complex number.

Syntax:

```
imag(complexNum);
```

where

complexNum is a complex number.

Example:

```
decl rv;
rv = imag(1-1);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name imag() is used for more than one type of expression. For comparison, see the Measurement Expression *imag() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **imag() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

int()

Returns the largest integer not greater than its real-number argument. This function is the same as floor(). Therefore, the value retuned for negative numbers may not be what most people would expect. To avoid confusion, use *fix()* (ael) or *floor()* (ael) depending on which is appropriate.

Syntax:

```
int(realVal);
```

where

realVal is a real value.

Example:

```
decl i;
i = int(-334.235); // returns integer -335
i = int(334.235); // returns integer 334
i = int(-.45e3); // returns integer -450
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Force to Grid
Generate SnP Component
PI Attenuator

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name int() is used for more than one type of expression. For comparison, see the Measurement Expression *int() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **int() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

ln()

Formerly log() in Series IV. Returns the natural logarithm of an integer, real, or complex number.

Syntax:

```
ln(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl a;
a = ln(1.5);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name ln() is used for more than one type of expression. For comparison, see the Measurement Expression *ln() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **ln() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

log10()

Returns the base 10 logarithm of an integer or real number.

Note
log(x) performs the same operation.

Syntax:

```
log10(rvalue);
```

where

rvalue is an integer or real number.

Example:

```
decl lval;
lval = log10(10.4);        // 1val=1.0170333392988
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name log10() is used for more than one type of expression. For comparison, see the Measurement Expression *log10() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **log10() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

log()

Returns the base 10 logarithm of an integer or real number.

Note
log10(x) performs the same operation.

Syntax:

```
log(rvalue);
```

where

rvalue is an integer or real number.

Example:

```
decl lval;
lval = log(10.4);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name log() is used for more than one type of expression. For comparison, see the Measurement Expression *log() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **log() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

mag()

Returns the absolute/magnitude value of an integer, real, or complex number.

Syntax:

```
mag(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl a;
a = mag(-4-6i);         // a=7.211102550928
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name mag() is used for more than one type of expression. For comparison, see the Measurement Expression *mag() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **mag() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

Note
The function name mag() is used for more than one type of expression. For comparison, see the Measurement Expression *mag() Measurement* (expmeas).

max2()

Returns the larger value of two numeric values, or NULL if parameters are invalid.

Syntax:

```
max2(val1, val2);
```

where

val1 is any integer or real number.

val2 is any integer or real number.

Example:

```
decl a;
a = max2(1.5, -1.5);         // a=1.5
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name max2() is used for more than one type of expression. For comparison, see the Measurement Expression *max2() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

min2()

Returns the lesser value of two numeric values, or NULL if parameters are invalid.

Syntax:

```
min2(val1, val2);
```

where

val1 is any integer or real number.

val2 is any integer or real number.

Example:

```
decl a;
a = min2\(1.5, -1.5\);        // a=-1.5
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name min2() is used for more than one type of expression. For comparison, see the Measurement Expression *min2() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

num()

Returns an integer that represents an ASCII numeric value of the first character in the specified string.

Syntax:

```
num(str);
```

where

str is a string.

Example:

```
decl a;
a = num("/users/myhome/fullpath");        // a=47
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Strip Alpha

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name num() is used for more than one type of expression. For comparison, see the Measurement Expression *num() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

Note
The function name num() is used for more than one type of expression. For comparison, see the Measurement Expression *num() Measurement* (expmeas).

phase()

Returns a real number that represents the phase in degrees of the argument. NULL is returned in case of an error in the argument.

Note
phasedeg() performs the same operation.

Syntax:

```
phase(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl a;
a = phase(4+6i);        // a=56.30993247402
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name phase() is used for more than one type of expression. For comparison, see the Measurement Expression *phase() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **phase() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

phasedeg()

Returns a real number that represents the phase in degrees of the argument. NULL is returned in case of an error in the argument.

Note
phase() performs the same operation.

Syntax:

```
phasedeg(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl a;
a = phasedeg(4+6i);        // a=56.30993247402
```

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name phasedeg() is used for more than one type of expression. For comparison, see the Measurement Expression *phasedeg() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **phasedeg() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

phaserad()

Returns a real number that represents the phase in radians of the argument. NULL is returned in case of an error in the argument.

Syntax:

```
phaserad(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl a;
a = phaserad(1.5);        // a=0
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name phaserad() is used for more than one type of expression. For comparison, see the Measurement Expression *phaserad() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **phaserad() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

polar()

Converts polar magnitude and angle into a complex number. Returns: a complex number or NULL if an error occurs.

Syntax:

```
polar(mag, ang);
```

where

mag is any integer or real number that represents polar magnitude.

ang is any integer or real number that represents an angle.

Example:

```
decl a;
a = polar(1.5, 90);        // a=9.18485e-17+1.5i (a complex number)
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name polar() is used for more than one type of expression. For comparison, see the Measurement Expression *polar() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **polar() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

pow()

Returns an integer or real number raised to given power.

Syntax:

```
pow(num, exponent);
```

where

num is an integer or real number to raise.

exponent is an exponent to raise number by.

Example:

```
decl b;
b = pow(4, 10);         // b=1048576
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Polygons to Circles
Polylines to Circles
Round Number

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Notes

1. For very large parameter values, pow() converts integers to real numbers, and returns a real number.

Note
The function name pow() is used for more than one type of expression. For comparison, see the Measurement Expression *pow() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **pow() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

rad()

Converts angle from degrees to radians. Returns: A real number that represents an angle in radians.

Syntax:

```
rad(angle);
```

where

angle is any integer or real number that represents an angle in degrees.

Example:

```
decl a;
a = rad(90);        // a=1.5707963267949
```

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name rad() is used for more than one type of expression. For comparison, see the Measurement Expression *rad() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **rad() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

Note
The function name rad() is used for more than one type of expression. For comparison, see the Measurement Expression *rad() Measurement* (expmeas).

re()

Returns a real number, the real part of a complex value.

Note
real() performs the same operation.

Syntax:

```
re(complexNum);
```

where

complexNum is a complex number.

Example:

```
decl r;
r = re(1-1);        // returns 0
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name re() is used for more than one type of expression. For comparison, see the Measurement Expression *re() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

real()

Returns a real number, the real part of a complex value.

Note
re() performs the same operation.

Syntax:

```
real(complexNum);
```

where

complexNum is a complex number.

Example:

```
decl r;
r = real(1-1);        // returns 0
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic,

Layout, Simulation, GUI

Note
The function name real() is used for more than one type of expression. For comparison, see the Measurement Expression *real() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **real() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

round()

Returns an integer, a real number rounded to nearest integer value.

Syntax:

```
round(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl a;
a = round(1.5);        // returns 2
```

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name round() is used for more than one type of expression. For comparison, see the Measurement Expression *round() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

sgn()

Returns the integer sign of an integer or real number, as either 1 or -1.

Syntax:

```
sgn(number);
```

where

number is an integer or real number.

Example:

```
decl s;
s = sgn(-14.5);        // returns -1
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name sgn() is used for more than one type of expression. For comparison, see the Measurement Expression *sgn() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **sgn() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

sin()

Returns the sine of a real number (in radians).

Syntax:

```
sin(realNum);
```

where

realNum is a real number, in radians.

Example:

```
decl s;
s = sin(PI/4\);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name sin() is used for more than one type of expression. For comparison, see the Measurement Expression *sin() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **sin() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

sinc()

Returns a real number that represents sin(parm)/parm in radians.

Syntax:

```
sinc(parm);
```

where

parm is any integer, real, or complex number.

Example:

```
decl x;
x = sinc(3+4i);    //x ==  -3.86024 - 3.85862i
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name sinc() is used for more than one type of expression. For comparison, see the Measurement Expression *sinc() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **sinc() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

sinh()

Returns the hyperbolic sine of an integer, real, or complex number.

Syntax:

```
sinh(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl a;
a = sinh(1.5);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name sinh() is used for more than one type of expression. For comparison, see the Measurement Expression *sinh() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **sinh() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

sqrt()

Returns the square root of a positive integer or real number. If the parameter is a negative integer or negative real number, it is changed to a complex and then the square root is calculated.

Syntax:

```
sqrt(x);
```

where

x is a positive integer or real number, a square root of the number is calculated.

x is a negative integer or real number, the number is converted to a complex number by making *x* be the *REAL* part of the complex and setting the *IMAGINARY* part of the complex to 0.0. Then the square root of the new complex number is calculated.

x is a complex number, the square root of the complex number is calculated.

x is 0, then the answer is 0.0

Example:

```
decl x,y,z;
x = sqrt(25);      // 5
y = sqrt(-25);     // 0 + 5i
z = sqrt(-25+5i);  // .497543 + 5.02469i
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
PI Attenuator

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name sqrt() is used for more than one type of expression. For comparison, see the Measurement Expression *sqrt() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **sqrt() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

step()

A step function that returns 0, 0.5, or 1. Returns: 0 if the argument < 0, 0.5 if argument = 0, or 1 if the argument > 0.

Syntax:

```
step(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl increment;
increment = step(1.5);        // returns 1
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name step() is used for more than one type of expression. For comparison, see the Measurement Expression *step() Measurement* (expmeas) and if using *Advanced Design System* see **Simulator Expressions** > **Transient Source Functions** > **step() Expression**. Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

tan()

Returns the tangent of a real number (in radians).

Syntax:

```
tan(realNum);
```

where

realNum is a real number, in radians.

Example:

```
decl t;
t = tan(PI/4);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation and GUI.

Note
The function name tan() is used for more than one type of expression. For comparison, see the Measurement Expression *tan() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **tan() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

tanh()

Returns the hyperbolic tangent of an integer, real, or complex number.

Syntax:

```
tanh(num);
```

where

num is any integer, real, or complex number.

Example:

```
decl a;
a = tanh(1.5);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation and GUI.

Note
The function name tanh() is used for more than one type of expression. For comparison, see the Measurement Expression *tanh() Measurement* (expmeas), if using *Advanced Design System* see **Simulator Expressions** > **Math Functions for Simulator Expressions** > **tanh() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

xor()

Returns an integer that represents the exclusive OR between arguments.

Syntax:

```
xor(num1, num2);
```

where

num1 is any integer.

num2 is any integer.

Example:

```
decl a;
a = xor(16, 32);        // a=48
```

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name xor() is used for more than one type of expression. For comparison, see the Measurement Expression *xor() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

Net Functions

- *Net Overview* (ael)
- *Connectivity Objects Overview* (ael)

This section describes the following Net functions:

- *db_get_net_name()* (ael)
- *db_is_net_named_by_user()* (ael)
- *db_is_net_grounded()* (ael)

See also

Net Iterator Functions (ael)

db_get_net_name()

Returns the name of the given *Net* (ael).

Syntax

```
decl netName = db_get_net_name(dbNet);
```

Where,

- *dbNet* is a *Net* (ael) object or Net iterator.

Example

```
decl instTermIter = db_create_inst_term_iter(instHandle);
for (; db_inst_term_iter_is_valid(instTermIter);
   instTermIter = db_inst_term_iter_get_next(instTermIter))
{
   decl pinNo = db_get_inst_term_number(instTermIter);
   decl net = db_get_inst_term_net(instTermIter);
   if (net == NULL)
       continue;
   decl netName = db_get_net_name(net);
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Net Overview (ael)
Net Functions (ael)
Net Iterator Functions (ael)
db_is_net_named_by_user() (ael)

Where Used (ael)

Schematic, Layout

db_is_net_grounded()

Returns TRUE if the given *Net* (ael) is grounded.

Syntax

```
decl isNetGrounded = db_is_net_grounded(dbNet);
```

Where,

- *dbNet* is a *Net* (ael) object or Net iterator.

Example

```
decl netIter = db_create_net_iter(context);
for (; db_net_iter_is_valid(netIter);
   netIter = db_net_iter_get_next(netIter))
{
   // skip grounded nets/nodes
   if (db_is_net_grounded(netIter))
       continue;
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Net Overview (ael)
Net Functions (ael)
Net Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_is_net_named_by_user()

Returns TRUE if the given *Net* (ael) was named by the user.

Syntax

```
decl isNetUserNamed = db_is_net_named_by_user(dbNet);
```

Where,

- *dbNet* is a *Net* (ael) object or Net iterator.

Example

```
decl netIter = db_create_net_iter(context);
```

```
for (; db_net_iter_is_valid(netIter);
   netIter = db_net_iter_get_next(netIter))
{
   // Only work with user named nets.
   if (!db_is_net_named_by_user(netIter))
       continue;

   decl netName = db_get_net_name(netIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Net Overview (ael)
Net Functions (ael)
Net Iterator Functions (ael)
db_get_net_name() (ael)

Where Used (ael)

Schematic, Layout

Net Iterator Functions

- *Net Overview* (ael)
- *Connectivity Objects Overview* (ael)

This section describes the following Net Iterator functions:

- *db_create_net_iter()* (ael)
- *db_net_iter_is_valid()* (ael)
- *db_net_iter_get_next()* (ael)
- *db_net_iter_get_net()* (ael)

See also

Net Functions (ael)

db_create_net_iter()

Returns an iterator to the first *Net* (ael) belonging to a given *DesignContext* (ael). If no *Net* (ael) is available, then the iterator returned will be invalid. The function *db_net_iter_is_valid()* (ael) can be used to test if an iterator is valid.

Syntax

```
decl netIter = db_create_net_iter(context);
```

Where,

- *context* is a *DesignContext* (ael).

Example

```
decl netIter = db_create_net_iter(context);
for (; db_net_iter_is_valid(netIter);
   netIter = db_net_iter_get_next(netIter))
{
   decl dbNetH = db_get_net_iter_get_net(netIter);
   //Get the net name.
   decl netName = db_get_net_name(dbNetH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Net Overview (ael)
Net Functions (ael)
Net Iterator Functions (ael)
db_net_iter_is_valid() (ael)
db_net_iter_get_next() (ael)

Where Used (ael)

Schematic, Layout

db_net_iter_get_net()

Returns the *Net* (ael) object that is referenced by the given net iterator.

Syntax

```
decl dbNet = db_net_iter_get_net(netIter);
```

Where,

- *netIter* is a valid net iterator.

Example

```
decl netIter = db_create_net_iter(context);
for (; db_net_iter_is_valid(netIter);
   netIter = db_net_iter_get_next(netIter))
{
   decl dbNetH = db_get_net_iter_get_net(netIter);
   //Get the net name.
   decl netName = db_get_net_name(dbNetH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Net Overview (ael)
Net Functions (ael)
Net Iterator Functions (ael)
db_create_net_iter() (ael)
db_net_iter_is_valid() (ael)
db_net_iter_get_next() (ael)

Where Used (ael)

Schematic, Layout

db_net_iter_get_next()

This function returns the next net iterator of the given net iterator.

Syntax

```
decl nextNetIter = db_net_iter_get_next(netIter);
```

Where,

- *netIter* is a valid net iterator.

Example

```
decl netIter = db_create_net_iter(context);
for (; db_net_iter_is_valid(netIter);
   netIter = db_net_iter_get_next(netIter))
{
   decl dbNetH = db_get_net_iter_get_net(netIter);
   //Get the net name.
   decl netName = db_get_net_name(dbNetH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Net Overview (ael)
Net Functions (ael)
Net Iterator Functions (ael)
db_create_net_iter() (ael)
db_net_iter_is_valid() (ael)

Where Used (ael)

Schematic, Layout

db_net_iter_is_valid()

This function returns TRUE if the given net iterator references a valid *Net* (ael) object and returns FALSE if the net iterator references no valid *Net* (ael) object.

Syntax

```
decl isValid = db_net_iter_is_valid(netIter);
```

Where,

- *netIter* is a net iterator.

Example

```
decl netIter = db_create_net_iter(context);
for (; db_net_iter_is_valid(netIter);
   netIter = db_net_iter_get_next(netIter))
{
   decl dbNetH = db_get_net_iter_get_net(netIter);
   //Get the net name.
   decl netName = db_get_net_name(dbNetH);
   …
```

```
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Net Overview (ael)
Net Functions (ael)
Net Iterator Functions (ael)
db_create_net_iter() (ael)
db_net_iter_get_next() (ael)

Where Used (ael)

Schematic, Layout

Netlist Functions

This section describes the following netlist function:

de_netlist_interpret_format_string() (ael)
de_netlist_create() (ael)
de_netlist_get_text() (ael)
de_netlist_save() (ael)

de_netlist_create()

This functions generates a netlist for the given *DesignContext* (ael). It returns a netlist object with the full netlist for the indicated *DesignContext* (ael). The *DesignContext* (ael) argument is used only when de_netlist_create() is called to generate the netlist object. Once created, the netlist object exists independent of the *DesignContext* (ael) object used to create it.

Syntax

```
decl netlistH = de_netlist_create(context);
```

Where,

- *context* is a *DesignContext* (ael) returned from a function such as *de_get_current_design_context()* (ael).

Example

```
decl context = de_get_current_design_context();

// Generate the netlist for the given design context.
decl netlistH = de_netlist_create(context);

// Get the string text from the netlist object.
decl strText = de_netlist_get_text(netlistH);

// Save the netlist to a file.
de_netlist_save(netlistH, "netlist.txt");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions.

See also

Netlist Functions (ael)
de_netlist_get_text() (ael)
de_netlist_interpret_format_string() (ael)
de_netlist_save() (ael)

Where Used (ael)

Schematic, Layout

de_netlist_get_text()

Returns the full text of a netlist as an AEL string.

Syntax

```
decl strval = de_netlist_get_text(netlistH);
```

Where,

- *netlistH* is a netlist object, created from a call to *de_netlist_create()* (ael).

Example

```
decl context = de_get_current_design_context();
// Generate the netlist for the given design context.
decl netlistH = de_netlist_create(context);
// Get the full netlist text as a string.
decl netlistStr = de_netlist_get_text(netlistH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions.

See also

Netlist Functions (ael)
de_netlist_create() (ael)
de_netlist_interpret_format_string() (ael)
de_netlist_save() (ael)

Where Used (ael)

Schematic, Layout

de_netlist_save()

This function saves the given netlist object's text to an indicated file. Returns TRUE if the netlist text was written to and saved to the indicated file.

Syntax

```
decl ok = de_netlist_save(netlistH, filename);
```

Where,

- *netlistH* is a netlist object, created from a call to *de_netlist_create()* (ael).
- *filename* is the name of the file to save the netlist text to.

Example

```
decl context = de_get_current_design_context();

// Generate the netlist for the given design context.
decl netlistH = de_netlist_create(context);
```

```
// Save the netlist text to a file.
decl ok = de_netlist_save(netlistH, "netlist.txt");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions.

See also

Netlist Functions (ael)
de_netlist_create() (ael)
de_netlist_get_text() (ael)
de_netlist_interpret_format_string() (ael)

***Where Used* (ael)**

Schematic, Layout

de_netlist_interpret_format_string()

Given a netlist format string, this function will return a string representing an outline of the structure of the format string. The string outline is in a psuedo-code style to help explain how the format string is being interpreted.

To learn more about creating netlist format strings, please see *Format Strings* (ael).

See also: *Format Strings* (ael).

Syntax:

```
de_netlist_interpret_format_string(fmtStr);
```

where

fmtStr is a netlist format string.

Example:

```
decl myNetlistFmt = "%d:%t %b%29?%:%k=%p %;%e";
// Get a string representing the outline of the structure of the given format string.
decl fmtStrStructure = de_netlist_interpret_format_string(myNetlistFmt);
```

Example's "fmtStrStructure" string contents:

```
item data
":"
instance name
" "
loop over parameters, from beginning to end
  if the current parameter has no value
    <no contents>
  else
    parameter keyword
    "="
    parameter net format
    " "
```

Version Introduced

ADS 2009 Update 1

Where Used: **(ael)**

Schematic, Layout

Object Functions

This section describes each object function in detail. The functions are listed in alphabetical order.

db_is_pin()

Returns TRUE if the passed in object is a pin. Otherwise it returns FALSE if the passed in object is not a pin.

Syntax

```
decl isPin = db_is_pin(object);
```

Where,

- *object* is a Pin, Shape, Instance, … or other AEL object.

Example

```
decl context = de_get_current_design_context()

  // Get first pin in design context.
  decl pinIter = db_create_pin_iter(context);
  if (!db_pin_iter_is_valid(pinIter))
      return;

  // Test if object is a pin?
  decl isPin = db_is_pin(pinIter); // In this case, isPin will be TRUE.

  decl instIter = db_create_inst_iter(context);

  // Test if object is a pin?
  isPin = db_is_pin(instIter); // In this case, isPin will be FALSE.
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_is_instance() (ael),
db_is_shape() (ael),
Object Functions (ael)

Where Used (ael)

Schematic, Layout

db_is_instance()

Returns TRUE if the passed in object is an instance. Otherwise it returns FALSE if the

passed in object is not an instance.

See also: *db_is_pin()* (ael), *db_is_shape()* (ael), *Object Functions* (ael), *Instance Functions* (ael).

Syntax:

```
db_is_instance(object);
```

where

object is an instance iterator or an instance object.

Example:

```
decl context = de_get_current_design_context();
// Get first instance in current design context.
decl instIter = db_create_inst_iter(context);
if (db_inst_iter_is_valid(instIter) == FALSE)
   return;
decl isInst = db_is_instance(instIter);  // This case isInst should be TRUE.
// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
   return;
isInst = db_is_instance(shapeIter);  // This case isInst should be FALSE.
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_is_shape()

Returns TRUE if the passed in object is a shape. Otherwise it returns FALSE if the passed in object is not a shape.

See also: *db_is_instance()* (ael), *db_is_pin()* (ael), *db_shape_is_arc()* (ael), *db_shape_is_const_line()* (ael), *db_shape_is_dot()* (ael), *db_shape_is_ellipse()* (ael), *db_shape_is_path()* (ael), *db_shape_is_polygon()* (ael), *db_shape_is_polyline()* (ael), *db_shape_is_rectangle()* (ael), *db_shape_is_wire_label()* (ael).

Syntax:

```
db_is_shape(object);
```

where

object is a shape iterator or a shape object.

Example:

```
decl context = de_get_current_design_context();
// Get first shape in current design context.
```

```
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
   return;
decl isShape = db_is_shape(shapeIter);  // This case isShape should be TRUE.
// Get first instance in current design context.
decl instIter = db_create_inst_iter(context);
if (db_inst_iter_is_valid(instIter) == FALSE)
   return;
isShape = db_is_shape(instIter);  // This case isShape should be FALSE.
```

Version Introduced

ADS 2009 Update 1

Where Used: **(ael)**

Schematic, Layout

Parameter Definition Functions

- ***Parameter Definition - Overview*** **(ael)**

This section describes the following functions:

- *dm_parm_get_name()* (ael)
- *dm_parm_get_label()* (ael)
- *dm_parm_get_attr()* (ael)
- *dm_parm_get_unit()* (ael)
- *dm_parm_get_formset_name()* (ael)
- *dm_parm_get_defvalue()* (ael)
- *dm_get_parm_definition_forms()* (ael)

dm_get_parm_definition_forms()

Returns a list of *form definitions* (ael) for the given *parameter definition* (ael).

Syntax

```
formDefList = dm_get_parm_definition_forms(parmDefH);
```

Where,

- *parmDefH* is a *parameter definition* (ael).

Example

```
decl instIter = db_create_inst_iter(context);
for ( ; db_inst_iter_is_valid(instIter);
   instIter = db_inst_iter_get_next(instIter) )
{
   decl paramIter = db_create_param_iter(instIter);
   for ( ; db_param_iter_is_valid(paramIter);
       paramIter = db_param_iter_get_next(paramIter))
   {
       decl parmDef = db_param_iter_get_parm_def( paramIter);
       decl parmFormList = dm_get_parm_definition_forms(parmDef);
       decl totalForms = listlen(parmFormList);
       decl idx;
       for (idx = 0; idx < totalForms; idx++)
       {
           decl formDefP = nth(idx, parmFormList);
           decl formName = dm_form_get_name(formDefP);
           decl formLabel = dm_form_get_label(formDefP);
           decl formNetFmtStr = dm_form_get_net_format(formDefP);
           decl formDispFmtStr = dm_form_get_disp_format(formDefP);
           decl formClassCode = dm_form_get_class(formDefP);
           ...
       }
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Definition - Overview (ael)
Parameter Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_parm_get_attr()

Returns the attribute code of the given *parameter definition* (ael). The *parameter definition* (ael) attribute code defines which attributes the parameter holds. Typical attributes that parameters may hold are: PARM_REAL, PARM_REPEATED, PARM_NO_DISPLAY, PARM_OPTIMIZABLE, PARM_DOE, and PARM_STATISTICAL.

For more information on all of the attributes a parameter may hold, please see *Parameter Attributes* (ael).

Syntax

```
decl parmAttr = dm_parm_get_attr(parmDefH);
```

Where,

- *parmDefH* is a *parameter definition* (ael)
  .

Example

```
decl parmAttr = dm_parm_get_attr(parmDefH);
// Do some work on a parameter if it is optimizable and a real value?
if ((parmAttr & PARM_OPTIMIZABLE) && (parmAttr & PARM_REAL))
{
   // Work on real, optimizable parameter.
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Definition - Overview (ael)
Parameter Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_parm_get_defvalue()

Returns the default *parameter value* (ael) for the given *parameter definition* (ael).

Note
This function ignores any parameter's default value callback function.

Syntax

```
decl paramValH = dm_parm_get_defvalue(parmDefH);
```

Where,

- *parmDefH* is a *parameter definition* (ael).

Example

```
decl paramValH = dm_parm_get_defvalue(parmDefH);
…
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Definition - Overview (ael)
Parameter Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_parm_get_formset_name()

Returns the string name of the *form set* (ael) used for the given *parameter definition* (ael).
For more information on *form sets* (ael), please see *Form Set - Overview* (ael).

Syntax

```
decl parmFormSetName = dm_parm_get_formset_name(parmDefH);
```

Where,

- *parmDefH* is a *parameter definition* (ael).

Example

```
decl parmFormSetName = dm_parm_get_formset_name(parmDefH);
// Get the form names from the parameter's form set.
decl formNamesList = dm_get_form_set_form_names(parmFormSetName);
…
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Definition - Overview (ael)
Parameter Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_parm_get_label()

Returns the label of the given *parameter definition* (ael). The *parameter definition* (ael) label is a brief string description of the parameter. It is used in the ADS standard Edit Component Parameters dialog to provide descriptive information about the selected parameter within the dialog during editing.

Syntax

```
decl parmLabel = dm_parm_get_label(parmDefH);
```

Where,

- *parmDefH* is a *parameter definition* (ael).

Example

```
decl parmLabel = dm_parm_get_label(parmDefH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Definition - Overview (ael)
Parameter Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_parm_get_name()

Returns the name of the parameter.

Syntax

```
decl parmName = dm_parm_get_name(parmDefH);
```

Where,

- *parmDefH* is a *parameter definition* (ael).

Example

```
decl parmName = dm_parm_get_name(parmDefH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Definition - Overview (ael)
Parameter Definition Functions (ael)

Where Used (ael)

Schematic, Layout

dm_parm_get_unit()

Returns the unit code of the given *parameter definition* (ael). The *parameter definition* (ael) unit type code defines which unit type the parameter value holds.
Typical unit type codes that parameters may hold are: UNITLESS_UNIT, FREQUENCY_UNIT, LENGTH_UNIT, TIME_UNIT, CURRENT_UNIT or TEMPERATURE_UNIT.

For more information on all of the different unit type codes a parameter may hold, please see *Unit Type* (ael).

Syntax

```
decl parmUnit = dm_parm_get_unit(parmDefH)
```

Where,

- *parmDefH* is a *parameter definition.* (ael)

Example

```
decl parmUnit = dm_parm_get_unit(parmDefH);
// If the parameter holds a value using a temperature unit, do some work with it.
if (parmUnit == TEMPERATURE_UNIT)
{
  // Work with the parameter that hold a temperature value.
  …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Definition - Overview (ael)
Parameter Definition Functions (ael)

Where Used (ael)

Schematic, Layout

Parameter Definition - Overview

A Parameter Definition defines and stores information about a particular ADS component parameter. The Parameter definition describes the type of value the parameter can hold and defines the parameter's behavior. An ADS Parameter Definition is also referred to as a Component Parameter Definition or a Component Item Parameter Definition.

A Parameter Definition AEL object stores the parameter definition information. The *create_parm()* (ael) function is used to create a new parameter definition within ADS.

The types of information that a Parameter Definition AEL object stores about a component parameter includes:

- the parameter's name.
- a brief text description of the parameter.
- a parameter attribute that can denote the type of value the parameter holds, the way the parameter is displayed, the way the parameter is netlisted, the way the parameter is used, etc.
- the name of the parameter's *form set* (ael) which describes the types of values the parameter contains. The *form set* (ael) defines the type of parameter *forms* (ael) that the parameter accepts for use. Parameter *forms* (ael) are used to help define the options of the parameter name and value for display in the edit component parameter dialog box, and the display for on screen editing, and how the parameter name and value are written during netlisting.
- the parameter value's unit type (e.g. Time, Length, Inductance, Resistance, Temperature, ...).
- the default parameter value.
- list of parameter callback functions.

Parameter information stored within a Parameter Definition

These are the different data fields stored within an AEL Parameter Definition Object.

Name

name is a string; a unique name of the parameter.

Label

label is a string; a descriptive label for the parameter.
This is used in the ADS standard edit component parameters dialog to provide a description for the selected parameter in the dialog during editing.

Parameter Attribute

attrib is an attribute code.
To set multiple attributes, bit-wise OR their numeric equivalent values.
For example, to set both PARM_REAL and PARM_REPEATED, bit-wise OR their numeric equivalent values and set attrib to the combined value attrib = PARM_REAL | PARM_REPEATED.

Different Parameter Attribute options are:

Attribute	Numeric Equivalent	Description
PARM_COMPLEX	524288	For complex parameter numeric value, the prompt in the standard component dialog box displays Complex, 1.2+j2.5.
PARM_COMPLEXARRAY	134217728	For complex array parameter numeric value, the prompt in the standard component dialog box displays Array, (1.5,3.6)(2.4,4.7).
PARM_DISCRETE_VALUE	32	For discrete parameter numeric value. Refer to Designing a Discrete Valued Parts Parametric Subnetwork.
PARM_DOE	4096	Allow doe entry page.
PARM_FIX	4194304	For fixed point parameter numeric value, the prompt in the standard component dialog box displays Fixed Point.
PARM_FIXARRAY	33554432	For fixed point array parameter numeric value, the prompt in the standard component dialog box displays Array, 1.25 3.5 ....
PARM_INT	131072	For integer parameter numeric value, the prompt in the standard component dialog box displays Integer.
PARM_INTARRAY	1677216	For integer array parameter numeric value, the prompt in the standard component dialog box displays Array, 1 2 3 ....
PARM_LAYOUT	256	Disable layout-related parameters if layout is not licensed.
PARM_NO_DISPLAY	512	Do not show parameter on schematic by default
PARM_NOT_EDITED	1	The parameter value is not editable.
PARM_NOT_NETLISTED	64	The parameter should not be netlisted.
PARM_NOT_ON_SCREEN_EDITABLE	32768	The parameter value is not on-screen editable.
PARM_OPTIMIZABLE	1024	Allow optimization entry page.
PARM_PRECISION	8388608	For precision parameter numeric value, the prompt in the standard component dialog box displays Precision, 8.24.
PARM_REAL	65536	For real parameter numeric value, the prompt in the standard component dialog box displays Real.
PARM_REALARRAY	67108864	For real array parameter numeric value, the prompt in the standard component dialog box displays Array, 1.5 3.6 ....
PARM_REPEATED	2	This is a repeated parameter; s[1]=100; s[2]=200;
PARM_RIGHT_HAND_ONLY	128	Do not generate left hand side when netlisting. Can be tested with netlist format %30. Also, when this attribute is set, the onscreen edit can partition the parameter displayed in the annotation into separate fields according to the annotation fmtstring, such as: start=%1s step=%2s.
PARM_STATISTICAL	2048	Allow statistical entry page.
PARM_STRING	262144	For string parameter numeric value, the prompt in the standard component dialog box displays String.
PARM_STRINGARRAY	26843456	For string array parameter numeric value, the prompt in the standard component dialog box displays Array, (str1)(str2).

FormSet

formSet is the name of the *form set* (ael) used for this value. This is the same as name in the *create_form_set()* (ael) function.

Note
For ADS 2011 and newer versions, *forms* (ael) and *formsets* (ael) are defined in a library. A *parameter definition* (ael) will have access to only the *forms* (ael) and *formsets* (ael) within the same library, except for the ADS built-in standard *forms* (ael) and *formsets* (ael). The standard *forms* (ael) and *formsets* (ael) that ship with ADS are available for use in any library.

Unit Type

unitCode is a constant integer choice. It indicates the type of unit the *parameter value* (ael) holds.

Unit Type choices are:

- UNITLESS_UNIT = -1
- FREQUENCY_UNIT = 0
- RESISTANCE_UNIT = 1
- CONDUCTANCE_UNIT = 2
- INDUCTANCE_UNIT = 3
- CAPACITANCE_UNIT = 4
- LENGTH_UNIT = 5
- TIME_UNIT = 6
- ANGLE_UNIT = 7
- POWER_UNIT = 8
- VOLTAGE_UNIT = 9
- CURRENT_UNIT = 10
- DISTANCE_UNIT = 11
- TEMPERATURE_UNIT = 12
- DB_GAIN_UNIT = 13

Default Value (Optional)

defaultValue is optional; a *parameter value* (ael) returned from the *prm()* (ael) function. The *prm()* (ael) function generates an acceptable default value for parameters with different *form sets* (ael).

Callback List (Optional)

cbList is optional; the list of callbacks. For example, list(dm_create_cb ("...", "..."), ...). Currently supported callbacks are: PARM_DEFAULT_VALUE_CB and PARM_MODIFIED_CB

Examples

Creating a new Parameter Definition

To create/define a new ADS *parameter definition* (ael), the AEL function *create_parm()* (ael) is used to define a new ADS *parameter definition* (ael).
Typically the use of *parameter definitions* (ael) is during the creation of a component *item definition* (ael). See *Item Definition - Overview* (ael) for information on *item definitions* (ael).
During the creation of a component *item definition* (ael), typically a list of newly created *parameter definitions* (ael) are passed into the *create_item()* (ael) function that creates the component's *item definition* (ael).

Example of Creating Parameter Definitions

```
decl parmDefZ = create_parm ("Z", "Characteristic Impedance", PARM_OPTIMIZABLE |
   PARM_STATISTICAL, "StdFileFormSet",
   RESISTANCE_UNIT, prm("StdForm","50.0"));
decl parmDefE = create_parm ("E", "Electrical Length", PARM_OPTIMIZABLE |
   PARM_STATISTICAL, "StdFileFormSet", ANGLE_UNIT,
   prm("StdForm","90"));

decl parmDefF = create_parm ("F", "Reference Frequency for Electrical Length",
   PARM_OPTIMIZABLE | PARM_STATISTICAL, "StdFileFormSet",
   FREQUENCY_UNIT, prm("StdForm","1 GHz"));
...
```

This example creates a parameter C representing capacitance, with attributes, using the StdFileFormSet *form set* (ael), capacitance as the units and 1.0 pF as the default value. This might then be used by *create_item()* (ael) to create a capacitance with C as its parameter.

```
decl parmDefC = create_parm("C", "Capacitance",
   PARM_OPTIMIZABLE | PARM_STATISTICAL, "StdFileFormSet",
   CAPACITANCE_UNIT, prm( "StdForm", "1.0 pF"));
```

Accessing Parameter Definition Data

Get the list of parameter definitions from an item definition.

See *dm_item_get_parms()* (ael) to get a *parameter definition* (ael) list object from an *item definition* (ael). Note the functions *dm_first_parm_definition()* (ael) and *dm_next_parm_definition()* (ael) can be used to traverse the list of *parameter definitions* (ael).

```
decl parmDefList = dm_item_get_parms(itemDefP);
```

Get the parameter definition for a parameter referenced by a parameter iterator

See *db_param_iter_get_parm_def()* (ael) to get a *parameter definition* (ael) for the parameter referenced by the *parameter iterator* (ael).

```
// Get parameter definition for the parameter.
decl parmDefP = db_param_iter_get_parm_def(paramIter);
```

Get the list of parameter definitions for a compound form

See *dm_form_get_parms()* (ael) - To get a the list of *parameter definitions* (ael) from a compound *form* (ael).

```
decl formParmDefList = dm_form_get_parms(formH);
```

Querying Parameter Definition Data

A *Parameter Definition* (ael) object is useful when there is a need to query or retrieve component *parameter definition* (ael) data, such as default *parameter values* (ael).

Example of Traversing and Querying an Item Definition's Parameter Definitions

An *item definition* (ael) from a *DesignContext* (ael) can be used to get the list of the parameter definitions for traversal. The AEL function *dm_item_get_parms()* (ael) is used to retrieve the a parameter definition list for a particular component *item definition* (ael). See *Item Definition - Overview* (ael) for more information about *item definitions* (ael). The list of parameter definitions can then be traversed to retrieve each parameter's definition. The AEL functions *dm_first_parm_definition()* (ael) and *dm_next_parm_definition()* (ael) can be used to traverse the parameter definitions in the parameter definition list.

```
decl context = de_get_current_design_context();
decl itemDefP = db_get_item_definition(context);
if (!itemDefP)
  return;
decl parmDefList = dm_item_get_parms(itemDefP);
decl parmDefP = dm_first_parm_definition(parmDefList);
for ( ; parmDefP != NULL; parmDefP = dm_next_parm_definition(parmDefP))
{
   // Get parameter definition information.
   decl defaultParmVal = dm_parm_get_defvalue(parmDefP);
   ...
}
```

Example of Traversing and Querying an Instance's Parameter Definitions

An *Instance Iterator* (ael) from a *DesignContext* (ael) can be used to traverse a design's instance data. The instance object and/or instance iterator can be used to retrieve a particular component instance's parameter information. The AEL function *db_create_param_iter()* (ael) is used to retrieve the a *parameter iterator* (ael) for a particular instance within a *DesignContext* (ael). See *ParamIterator* (ael) for more information about *parameter iterators* (ael). The *parameter iterator* (ael) can be used to retrieve different parameter information such as the parameter's definition. The AEL function *db_param_iter_get_parm_def()* (ael) can be used to get the parameter definition from a *parameter iterator* (ael).

```
decl context = de_get_current_design_context();
decl instIter = db_create_inst_iter(context);
for ( ; db_inst_iter_is_valid(instIter);
      instIter = db_inst_iter_get_next(instIter) )
{
   // Loop through the instance's component parameters
   decl paramIter = db_create_param_iter(instIter);
   for ( ; db_param_iter_is_valid(paramIter);
       paramIter = db_param_iter_get_next(paramIter) )
   {
       // Get parameter definition for the parameter.
       decl parmDefP = db_param_iter_get_parm_def(paramIter);
       decl defaultParmVal = dm_parm_get_defvalue(parmDefP);
   }
   ...
}
```

List of Parameter Definition Functions

See list here: *Parameter Definition Functions* (ael).

Parameter Iterator Functions

- ***Parameter Iterator - Overview*** **(ael)**

This section describes the following parameter iterator functions:

- *db_create_param_iter()* (ael)
- *db_param_iter_is_valid()* (ael)
- *db_param_iter_get_next()* (ael)
- *db_param_iter_get_first()* (ael)
- *db_param_iter_flatten_repeats()* (ael)
- *db_param_iter_get_param()* (ael)
- *db_param_iter_get_parm_def()* (ael)
- *db_param_iter_is_repeatable()* (ael)
- *db_param_iter_is_repeated()* (ael)
- *db_param_iter_get_repeat_index()* (ael)
- *db_param_iter_get_sub_parameters()* (ael)
- *db_param_iter_find_form()* (ael)
- *db_param_iter_get_display_value()* (ael)
- *db_param_iter_get_netlist_value()* (ael)
- *db_param_iter_find_by_name()* (ael)
- *db_find_inst_param_by_name()* (ael)
- *db_param_iter_find_by_index()* (ael)
- *db_param_iter_find_by_name_and_repeat()* (ael)
- *db_param_iter_find_by_index_and_repeat()* (ael)

db_create_param_iter()

Returns an iterator to the first parameter in the collection. If no parameter is available, then the iterator returned will be invalid. The function *db_param_iter_is_valid()* (ael) can be used to test if a *parameter iterator* (ael) is valid.

Syntax

```
decl iter = db_create_param_iter(dbInst);
```

Where,

- *dbInst* is an instance or *instance iterator* (ael).

Example

```
decl designContext = de_get_current_design_context();
decl instIter = db_create_inst_iter(designContext);
for ( ; db_inst_iter_is_valid(instIter);
   instIter =  db_inst_iter_get_next(instIter) )
{
   decl paramIter = db_create_param_iter(instIter);

   for ( ; db_param_iter_is_valid(paramIter);
       paramIter = db_param_iter_get_next(paramIter) )
   {
       decl paramName = db_get_param_name(paramIter);
       …
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_find_inst_param_by_name()

Returns a new *parameter iterator* (ael) for the parameter with the given parameter name within the given *parameter iterator* (ael)'s list of parameters.

If there is no parameter found with that name, then the returned *parameter iterator* (ael) will be invalid. The function *db_param_iter_is_valid()* (ael) can be used to test if a *parameter iterator* (ael) is valid.

Syntax

```
decl foundParamIter = db_find_inst_param_by_name(paramIter, paramName)
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael)
- *paramName* is the string name of the parameter to find

Example

```
decl foundParamIter = db_find_inst_param_by_name(paramIter, "Freq");
if (db_param_iter_is_valid(foundParamIter))
{
   decl displayValueStr = db_param_iter_get_display_value(foundParamIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_param_iter_find_by_index_and_repeat()

Returns a new *parameter iterator* (ael) for the parameter with the given parameter index and the given parameter's repeat index within the given *parameter iterator* (ael)'s list of parameters. If there is no parameter found with that parameter index and repeat parameter index, then the returned *parameter iterator* (ael) will be invalid. The function *db_param_iter_is_valid()* (ael) can be used to test if a *parameter iterator* (ael) is valid.

Syntax

```
decl foundParamIter = db_param_iter_find_by_index_and_repeat(paramIter,
paramIndex, paramRepeatedIndex);
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael)
- *paramIndex* is the index of the top-level repeatable parameter to find
- *paramRepeatedIndex* is the index of the top-level parameter's repeated sub-parameter to find

Example

```
decl foundParamIter = db_param_iter_find_by_index_and_repeat(paramIter, 1, 4);
if (db_param_iter_is_valid(foundParamIter))
{
   decl displayValueStr = db_param_iter_get_display_value(foundParamIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_param_iter_find_by_index()

Returns a new *parameter iterator* (ael) for the parameter with the given parameter integer index within the given *parameter iterator* (ael)'s list of parameters. If there is no parameter found with that index, then the returned *parameter iterator* (ael) will be invalid. The function *db_param_iter_is_valid()* (ael) can be used to test if a *parameter iterator* (ael) is valid.

Syntax

```
decl foundParamIter = db_param_iter_find_by_index(paramIter, paramIndex);
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael)
- *paramIndex* is the index of the parameter to find

Example

```
decl foundParamIter = db_param_iter_find_by_index(paramIter, 2);
if (db_param_iter_is_valid(foundParamIter))
{
   decl displayValueStr = db_param_iter_get_display_value(foundParamIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_param_iter_find_by_name_and_repeat()

Returns a new *parameter iterator* (ael) for the parameter with the given parameter name and the given parameter repeat index within the given *parameter iterator* (ael)'s list of parameters. If there is no parameter found with that name and repeat index, then the returned *parameter iterator* (ael) will be invalid. The function *db_param_iter_is_valid()* (ael) can be used to test if a *parameter iterator* (ael) is valid.

Syntax

```
decl foundParamIter = db_param_iter_find_by_name_and_repeat(paramIter,
paramName, paramRepeatIndex);
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael)
- *paramName* is the string name of the top-level repeatable parameter to find
- *paramRepeatedIndex* is the index of the top-level parameter's repeated sub-parameter to find

Example

```
decl foundParamIter = db_param_iter_find_by_name_and_repeat(paramIter, "Freqs", 4);
```

```
if (db_param_iter_is_valid(foundParamIter))
{
   decl displayValueStr = db_param_iter_get_display_value(foundParamIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

***Where Used* (ael)**

Schematic, Layout

db_param_iter_find_by_name()

Returns a new *parameter iterator* (ael) for the parameter with the given parameter name within the given *parameter iterator* (ael)'s list of parameters.

If there is no parameter found with that name, then the returned *parameter iterator* (ael) will be invalid. The function *db_param_iter_is_valid()* (ael) can be used to test if a *parameter iterator* (ael) is valid.

Syntax

```
decl foundParamIter = db_param_iter_find_by_name(paramIter, paramName);
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael)
- *paramName* is the string name of the parameter to find

Example

```
decl foundParamIter = db_param_iter_find_by_name(paramIter, “Freq”);
if (db_param_iter_is_valid(foundParamIter))
{
   decl displayValueStr = db_param_iter_get_display_value(foundParamIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_param_iter_find_form()

Returns the *form definition* (ael) for the parameter referenced by the given *parameter iterator* (ael). If the *form definition* (ael) cannot be found, then this function returns NULL.

To learn more about *form definitions* (ael), please see: *Form Definition - Overview* (ael).

Syntax

```
decl formDef = db_param_iter_find_form(paramIter);
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael)

Example

```
decl parmFormDef = db_param_iter_find_form(paramIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_param_iter_flatten_repeats()

Function to flatten the repeated parameters while iterating. The *db_param_iter_get_next()* (ael) function will skip the top-level repeatable parameters and iterate through the repeated sub-parameters. If the iterator has been started (used to find a parameter), then calling this function gives an AEL error.

Syntax

```
paramIter = db_param_iter_flatten_repeats(paramIter);
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael).

Example

```
decl designContext = de_get_current_design_context();
decl instIter = db_create_inst_iter(designContext);
for ( ; db_inst_iter_is_valid(instIter);
   instIter =  db_inst_iter_get_next(instIter) )
{
   decl paramIter = db_create_param_iter(instIter);
   paramIter = db_param_iter_flatten_repeats(paramIter);
   for ( ; db_param_iter_is_valid(paramIter);
       paramIter = db_param_iter_get_next(paramIter) )
   {
       decl paramVal = db_param_iter_get_param(paramIter);
       …
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_param_iter_get_display_value()

Returns a string with the displayed value of the parameter referenced by the given *parameter iterator* (ael). If the parameter value cannot be formatted to a proper display value string, then an AEL error will occur.

Syntax

```
decl displayValueStr = db_param_iter_get_display_value(paramIter);
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael)

Example

```
decl parmDisplayValStr = db_param_iter_get_display_value(paramIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_param_iter_get_first()

Returns the first *parameter iterator* (ael) from the given *parameter iterator* (ael). This is useful if you want to reset iteration back to the beginning of the parameters.

Syntax

```
decl firstParamIter = db_param_iter_get_first(paramIter);
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael).

Example

```
decl designContext = de_get_current_design_context();
decl instIter = db_create_inst_iter(designContext);
for ( ; db_inst_iter_is_valid(instIter);
   instIter =  db_inst_iter_get_next(instIter) )
{
   decl paramIter = db_create_param_iter(instIter);
   for ( ; db_param_iter_is_valid(paramIter);
       paramIter = db_param_iter_get_next(paramIter) )
   {
       decl paramName = db_get_param_name(paramIter);
       …
   }

   // Move back to the beginning of the parameter collection
   paramIter = db_param_iter_get_first(paramIter);

   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_param_iter_get_netlist_value()

Returns a string with the netlist value of the parameter referenced by the given *parameter iterator* (ael).
If the parameter value cannot be formatted to a proper netlist value string, then an AEL error will occur.

Syntax

```
decl netlistValueStr = db_param_iter_get_netlist_value(paramIter);
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael)

Example

```
decl parmNetlistValStr = db_param_iter_get_netlist_value(paramIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_param_iter_get_next()

Returns the next *parameter iterator* (ael) from the given *parameter iterator* (ael).

Syntax

```
decl nextParamIter = db_param_iter_get_next(paramIter);
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael).

Example

```
decl designContext = de_get_current_design_context();
```

```
decl instIter = db_create_inst_iter(designContext);
for ( ; db_inst_iter_is_valid(instIter);
   instIter =  db_inst_iter_get_next(instIter) )
{
   decl paramIter = db_create_param_iter(instIter);
   for ( ; db_param_iter_is_valid(paramIter);
       paramIter = db_param_iter_get_next(paramIter) )
   {
       decl paramName = db_get_param_name(paramIter);
       …
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

***Where Used* (ael)**

Schematic, Layout

db_param_iter_get_param()

Returns the *parameter value* (ael) referred to by the given *parameter iterator* (ael).
To learn more about parameter values please see: *Parameter Value - Overview* (ael).

Syntax

```
decl paramValH = db_param_iter_get_param(paramIter);
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael).

Example

```
decl paramValH = db_param_iter_get_param(paramIter);
if (db_param_value_uses_string_form(paramValH))
{
   decl paramStrValue = db_get_param_string_value(paramValH);
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_param_iter_get_parm_def()

Returns the *parameter definition* (ael) referred to by the given *parameter iterator* (ael). To learn more about *parameter definitions* (ael) please see: *Parameter Definition - Overview* (ael).

Syntax

```
decl parmDef = db_param_iter_get_parm_def(paramIter);
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael).

Example

```
decl parmDef = db_param_iter_get_parm_def(paramIter);
decl parmUnit = dm_parm_get_unit(parmDef);
if (parmUnit == LENGTH_UNIT)
{
   ...
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_param_iter_get_repeat_index()

Returns the integer repeated index of the parameter referenced by the given *parameter iterator* (ael) if the parameter is a repeat of a top-level repeatable parameter. If the parameter is not a repeat of a top-level repeatable parameter, then this function will return -1.

Syntax

```
decl parmRepeatIndex = db_param_iter_get_repeat_index(paramIter);
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael).

Example

```
if (db_param_iter_is_repeated(paramIter))
{
  decl parmRepeatIndex = db_param_iter_get_repeat_index(paramIter);
  …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_param_iter_get_sub_parameters()

Returns a *parameter iterator* (ael) to the sub-parameter values of the parameter referenced by the given *parameter iterator* (ael), if that parameter is a repeatable parameter or a compound valued parameter.

If there are no valid sub-parameters for the parameter referenced by the given *parameter iterator* (ael), then the returned *sub-parameter iterator* (ael) will be invalid. The function *db_param_iter_is_valid()* (ael) can be used to test if a *parameter iterator* (ael) is valid.

Syntax

```
decl subParamIter = db_param_iter_get_sub_parameters(paramIter);
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael).

Example

```
decl designContext = de_get_current_design_context();
decl instIter = db_create_inst_iter(designContext);
for ( ; db_inst_iter_is_valid(instIter);
   instIter =  db_inst_iter_get_next(instIter) )
{
   decl paramIter = db_create_param_iter(instIter);
   for ( ; db_param_iter_is_valid(paramIter);
```

```
        paramIter = db_param_iter_get_next(paramIter) )
    {
        if (db_param_iter_is_repeatable(paramIter) ||
            db_param_uses_compound_form(paramIter))
        {
            // Iterate over the sub-parameters
            decl subParamIter = db_param_iter_get_sub_parameters(paramIter);
            for ( ; db_param_iter_is_valid(subParamIter);
               subParamIter = db_param_iter_get_next(subParamIter) )
            {
                decl paramVal = db_param_iter_get_param(subParamIter);
                …
            }
        }
    }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

***Where Used* (ael)**

Schematic, Layout

db_param_iter_is_repeatable()

Returns TRUE if the parameter referenced by the given *parameter iterator* (ael) is a top-level parameter that is repeatable. Returns FALSE otherwise.

Note
This function will not work when iteration has been flattened by using the *db_param_iter_flatten_repeats()* (ael) function.

Syntax

```
decl parmIsRepeatable = db_param_iter_is_repeatable(paramIter);
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael).

Example

```
decl parmIsRepeatable = db_param_iter_is_repeatable(paramIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_param_iter_is_repeated()

Returns TRUE if the parameter referenced by the given *parameter iterator* (ael) is a repeat of a top-level repeatable parameter. Returns FALSE otherwise.

Syntax

```
decl parmIsRepeated = db_param_iter_is_repeated(paramIter);
```

Where,

- *paramIter* is a valid *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael).

Example

```
decl parmIsRepeated = db_param_iter_is_repeated(paramIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

db_param_iter_is_valid()

Returns TRUE if the *parameter iterator* (ael) is referencing a valid parameter.

Syntax

```
decl isValid = db_param_iter_is_valid(paramIter);
```

Where,

- *paramIter* is a *parameter iterator* (ael) returned from a function such as *db_create_param_iter()* (ael)

Example

```
decl designContext = de_get_current_design_context();
decl instIter = db_create_inst_iter(designContext);
for ( ; db_inst_iter_is_valid(instIter);
   instIter =  db_inst_iter_get_next(instIter) )
{
   decl paramIter = db_create_param_iter(instIter);

   for ( ; db_param_iter_is_valid(paramIter);
       paramIter = db_param_iter_get_next(paramIter) )
   {
       decl paramName = db_get_param_name(paramIter);
       …
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Iterator - Overview (ael)
Parameter Iterator Functions (ael)

Where Used (ael)

Schematic, Layout

Parameter Iterator - Overview

An ADS Parameter Iterator references the parameter information for a particular instance of an ADS component. The Parameter Iterator object holds the collection of *parameter values* (ael) of the instance, the collection of *parameter definitions* (ael) for the instance, and a reference to the library that contains the *parameter definitions* (ael). The *db_create_param_iter()* (ael) function is used to create a new parameter iterator within ADS for a particular instance.

The types of information that a Parameter Iterator (ParamIter) AEL object can access about a component parameter includes:

- The *parameter value* (ael) of parameters of a given instance. See *Parameter Value - Overview* (ael) for more information about *parameter values* (ael).
- The *parameter definition* (ael) of parameters of a given instance. See *Parameter Definition - Overview* (ael) for more information about *parameter definitions* (ael).
- The repeated parameter information of a parameter of a given instance.

Creating a new Parameter Iterator

To create an ADS parameter iterator, the AEL function *db_create_param_iter()* (ael) is used to get a parameter iterator for a particular ADS component instance.

The parameter iterator can then be used to traverse all of the instance's *parameter values* (ael) and parameter definitions and can be used to retrieve information for an instance parameter such as its display value, its netlist value, any sub parameters, the repeated indices of a repeatable parameters, or the actual *parameter value* (ael) or *parameter definition* (ael) of a particular instance parameter.

Examples of Creating a new Parameter Iterator

```
decl designContext = de_get_current_design_context();
decl instIter = db_create_inst_iter(designContext);
for ( ; db_inst_iter_is_valid(instIter);
    instIter =  db_inst_iter_get_next(instIter) )
{
   decl paramIter = db_create_param_iter(instIter);

   for ( ; db_param_iter_is_valid(paramIter);
       paramIter = db_param_iter_get_next(paramIter) )
   {
     …
   }
}
```

Examples of Creating a Sub-Parameter Parameter Iterator

```
decl designContext = de_get_current_design_context();
decl instIter = db_create_inst_iter(designContext);
for ( ; db_inst_iter_is_valid(instIter);
    instIter =  db_inst_iter_get_next(instIter) )
{
   decl paramIter = db_create_param_iter(instIter);

   for ( ; db_param_iter_is_valid(paramIter);
       paramIter = db_param_iter_get_next(paramIter) )
   {
```

```
        // If the parameter referenced by the parameter iterator has sub-parameters
        //  Create a sub parameter iterator of that top-level parameter.
        decl subParamIter = db_param_iter_get_sub_parameters(paramIter);
        // Iterate over the sub-parameters if the sub-parameter iterator is valid.
        for ( ; db_param_iter_is_valid(subParamIter);
            subParamIter = db_param_iter_get_next(subParamIter) )
        {
            ...
        }
    }
}
```

Querying Parameter Information by using a ParamIter

A Parameter Iterator is useful when there is a need to query or retrieve component *parameter value* (ael) information or *parameter definition* (ael) information. An *Instance Iterator* (ael) from a *DesignContext* (ael) can be used to traverse a design's instance data. The instance object and/or instance iterator can be used to retrieve a particular component instance's parameter information. The AEL function *db_create_param_iter()* (ael) is used to retrieve the a parameter iterator for a particular instance within a *DesignContext* (ael).
See the list of *Parameter Iterator Functions* (ael) for more information.

Example of Querying Parameter Information by using a ParamIter

```
decl designContext = de_get_current_design_context();
decl instIter = db_create_inst_iter(designContext);
for ( ; db_inst_iter_is_valid(instIter);
    instIter =  db_inst_iter_get_next(instIter) )
{
   decl paramIter = db_create_param_iter(instIter);
   for ( ; db_param_iter_is_valid(paramIter);
      paramIter = db_param_iter_get_next(paramIter) )
   {
      decl parmName = db_get_param_name(paramIter);
      decl parmDefP = db_param_iter_get_parm_def(paramIter);
      decl defaultParmVal = dm_parm_get_defvalue(parmDefP);
      decl defaultParmValStr = "";
      if (db_param_value_uses_string_form(defaultParmVal))
      {
          // Get the default parameter value's string value
          defaultParmValStr = db_get_param_string_value (defaultParmVal);
      }

      decl parmVal = db_param_iter_get_param(paramIter);
      decl parmValStr = "";
      if (db_param_value_uses_string_form(parmVal))
      {
          // Get the parameter value's string value
          parmValStr = db_get_param_string_value(parmVal);
      }
      …
   }
}
```

List of Parameter Iterator Functions

See list here: *Parameter Iterator Functions* (ael).

Parameter Value Functions

- ***Parameter Value - Overview* (ael)**

This section describes the following functions:

- *db_get_param_name()* (ael)
- *db_get_param_value_code()* (ael)
- *db_is_param_repeatable()* (ael)
- *db_param_value_uses_string_form()* (ael)
- *db_param_value_uses_constant_form()* (ael)
- *db_param_value_uses_compound_form()* (ael)
- *db_get_param_form_name()* (ael)
- *db_set_param_form_name()* (ael)
- *db_get_param_string_value()* (ael)
- *db_set_param_string_value()* (ael)
- *db_is_param_displayed()* (ael)
- *db_set_param_displayed()* (ael)

db_get_param_form_name()

Returns the string *form* (ael) name of the given *parameter value* (ael)'s form. See *Form Definition - Overview* (ael) for more information on *forms* (ael).

Syntax

```
decl strFormName = db_get_param_form_name (paramVal);
```

Where,

- *paramval* is a *parameter value* (ael).

Example

```
decl formName = db_get_param_form_name(paramValH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Value - Overview (ael)
Parameter Value Functions (ael)

Where Used (ael)

Schematic, Layout

db_get_param_name()

Returns the string parameter name for the given *parameter value* (ael).

Syntax

```
decl param = db_get_param_name(paramVal);
```

Where,

- *paramVal* is a *parameter value* (ael).

Example

```
decl paramName = db_get_param_name(paramValH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Value - Overview (ael)
Parameter Value Functions (ael)

Where Used (ael)

Schematic, Layout

db_get_param_string_value()

Returns the string value of the given *parameter value* (ael).

Note:
The string value is only used if the *parameter value* (ael) is of string form type. You can use the *db_param_value_uses_string_form()* (ael) to check if a *parameter value* (ael) is using a string *form* (ael). If the *parameter value* (ael) is not of string *form* (ael) type, then the string value will be NULL.

Syntax

```
decl strParamVal = db_get_param_string_value(paramVal);
```

Where,

- *paramval* is a *parameter value* (ael).

Example

```
if (db_param_value_uses_string_form(paramValH))
{
   decl strParamVal = db_get_param_string_value(paramValH);
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Value - Overview (ael)
Parameter Value Functions (ael)

Where Used (ael)

Schematic, Layout

db_get_param_value_code()

Returns an integer *parameter value* (ael) code that represent's the *parameter value* (ael) *form* (ael)'s class type. See *Form Class Codes* (ael), for more information.

Syntax

```
decl paramValCode = db_get_param_value_code (paramVal);
```

Where,

- *paramVal* is a *parameter value* (ael).

Example

```
decl paramValCode = db_get_param_value_code (paramVal);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Value - Overview (ael)
Parameter Value Functions (ael)

Where Used (ael)

Schematic, Layout

db_is_param_displayed()

Returns TRUE if the *parameter value* (ael) is marked to be displayed in the current view. FALSE otherwise.

Syntax

```
decl isDisplayed = db_is_param_displayed(paramVal);
```

Where,

- *paramVal* is a *parameter value* (ael).

Example

```
decl isDisplayed = db_is_param_displayed(paramValH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Value - Overview (ael)
Parameter Value Functions (ael)

***Where Used* (ael)**

Schematic, Layout

db_is_param_repeatable()

Returns TRUE if the parameter is repeatable. FALSE otherwise.

Syntax

```
decl isRepeatable = db_is_param_repeatable (paramVal);
```

Where,

- *paramVal* is a *parameter value* (ael).

Example

```
decl isParamRepeatable = db_is_param_repeatable(paramValH);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Value - Overview (ael)
Parameter Value Functions (ael)

***Where Used* (ael)**

Schematic, Layout

db_param_value_uses_compound_form()

Returns TRUE if the parameter is compound valued. FALSE otherwise.

Syntax

```
decl isCompoundValue = db_param_value_uses_compound_form(paramVal);
```

Where,

- *paramVal* is a *parameter value* (ael).

Example

```
decl isCompoundValue = db_param_value_uses_compound_form(paramVal);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Value - Overview (ael)
Parameter Value Functions (ael)

***Where Used* (ael)**

Schematic, Layout

db_param_value_uses_constant_form()

Returns TRUE if the parameter is constant valued. FALSE otherwise.

Syntax

```
decl isConstValue = db_param_value_uses_constant_form (paramVal);
```

Where,

- *paramVal* is a *parameter value* (ael).

Example

```
decl isConstValue = db_param_value_uses_constant_form (paramVal);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Value - Overview (ael)
Parameter Value Functions (ael)

***Where Used* (ael)**

Schematic, Layout

db_param_value_uses_string_form ()

Returns TRUE if the parameter is string valued. FALSE otherwise.

Syntax

```
decl isStrValue = db_param_value_uses_string_form(paramVal);
```

Where,

- *paramVal* is a *parameter value* (ael).

Example

```
decl isStrValue = db_param_value_uses_string_form(paramVal);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Value - Overview (ael)
Parameter Value Functions (ael)

Where Used (ael)

Schematic, Layout

db_set_param_displayed()

Function to mark a *parameter value* (ael) to be displayed or not displayed in the current view.

Syntax

```
db_set_param_displayed(paramVal, toDisplay);
```

Where,

- *paramVal* is a *parameter value* (ael).
- *toDisplay* is a Boolean TRUE or FALSE.
  Where if set TRUE, the *parameter value* (ael) will be marked to be displayed in the current view; but if set FALSE, the *parameter value* (ael) will be marked to not be displayed in the current view.

Example

```
// If parameter value is not displayed in the current view, mark it as displayed.
if (!db_is_param_displayed(paramValH))
   db_set_param_displayed(paramValH, TRUE);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Value - Overview (ael)
Parameter Value Functions (ael)

Where Used (ael)

Schematic, Layout

db_set_param_form_name()

Sets the given *parameter value* (ael)'s *form* (ael) to be the *form* (ael) of the given string form name. See *Form Definition - Overview* (ael) for more information on *forms* (ael).

Syntax

```
db_set_param_form_name (paramVal, formName)
```

Where,

- *paramval* is a *parameter value* (ael).
- *formName* is a string *form* (ael) name.

Example

```
db_set_param_form_name(paramValH, “StdForm”);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Value - Overview (ael)
Parameter Value Functions (ael)

Where Used (ael)

Schematic, Layout

db_set_param_string_value()

Set the string value of the given *parameter value* (ael).

Note
The string value is only used if the *parameter value* (ael) is of string *form* (ael) type. You can use the *db_param_value_uses_string_form()* (ael) to check if a *parameter value* (ael) is using a string *form* (ael). If the *parameter value* (ael) is not of string *form* (ael) type, then the string value will be NULL.

Syntax

```
db_set_param_string_value(paramVal, strValue);
```

Where,

- *paramVal* is a *parameter value* (ael).
- *strValue* is a string representing the value, such as "12.33".

Example

```
if (db_param_value_uses_string_form(paramValH))
{
   db_set_param_string_value(paramValH, "12.33");
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Parameter Value - Overview (ael)
Parameter Value Functions (ael)

Where Used (ael)

Schematic, Layout

Parameter Value - Overview

All ADS parameter values are described by a *form* (ael) that defines how a parameter gets displayed and netlisted. The *form* (ael) for a parameter value must appear in the *formset* (ael) of the parameter value's *parameter definition* (ael).

ADS parameter values can come in four different classes or in other words, types. The four classes are string parameters, constant *form* (ael) parameters, compound *form* (ael) parameters and repeated parameters.
The parameter value's value code identifies which class/type is the parameter and its *form* (ael).

- A parameter value with value code class/type == **DM_STRING_CLASS** represents a string parameter. The parameter value is a string and the *form* (ael) controls how the parameter is displayed and netlisted.
- A parameter value with value code class/type == **DM_NAME_CLASS** represents a constant *form* (ael) parameter. The parameter value is like an enum and the *form* (ael) completely describes the parameter's value.
- A parameter value with value code class/type == **DM_NAMED_LIST_CLASS** represents a compound *form* (ael) parameter. The parameter value is a list of sub-parameters and the *form* (ael) describes how the parameter will be displayed and netlist. The *form* (ael) for each of the sub-parameters describes how the portion of the parameter will be displayed and netlisted.

  Note
  The sub-parameters of a compound *form* (ael) parameter can not be compound so there cannot be sub-parameters of sub-parameters.

- A parameter value with value code class/type == **DM_LIST_CLASS** represents a repeated parameter. A repeated parameter is not really a different *form* (ael) of a parameter, they are just parameters that can have more than one value on each instance of that parameter. Repeated parameters have only one parameter on the instance, but that parameter has a list of sub-parameters (the repeats). The list of sub-parameters should never be empty and the sub-parameters are unnamed. Repeated parameters are identified by name and index. The sub-parameters can't be repeated, but they can use compound *forms* (ael), so they can have sub-parameters.

The types of information that a Parameter Value AEL object stores includes:

- Name
- Value Code
- String Value
- Form Name: The *form* (ael) name of the parameter's value.
- Display Status: The parameter's value display status.
- Value Type: The parameter's value type, such as is it a constant, string or compound value.
- The parameter's value is repeated status.

Parameter information stored within a Parameter Value

These are the different data fields stored within an AEL Parameter Value Object.

Name

name is a string; the name of the parameter.

Value Code

value code is an integer; the value code represents the parameter value *form* (ael)'s class type. A *form* (ael) can be defined by several classes. Each class type defines specific syntax requirements for the *form* (ael). Some of the *form* (ael) classes are: **DM_NAME_CLASS**, **DM_NAMED_LIST_CLASS**, **DM_LIST_CLASS**, and **DM_STRING_CLASS**.

String Value

holds the string value of the parameter.

The string value is only used if the parameter value is of string *form* (ael) type. You can use the *db_param_value_uses_string_form()* (ael) to check if a parameter value is using a string *form* (ael). If the parameter value is not of string *form* (ael) type, then the string value will be NULL.

Form Name

Form name is the string name of the *form* (ael)used for this parameter value. Parameter *forms* (ael) are used to help define the options of the parameter name and value for display in the edit component parameter dialog box, display for on screen editing, and how the parameter name and value are written during netlisting.

Note
For ADS 2011 and newer ADS versions, *forms* (ael) and *formsets* (ael) are defined in a library. A *parameter definition* (ael) will have access to only the *forms* (ael) and *formsets* (ael) within the same library, except for the ADS built-in standard *forms* (ael) and *formsets* (ael). The standard *form* (ael) and *formsets* (ael) that ship with ADS will be available for use in any library.

Parameter's value type

denotes what is the type of the parameter value: a string value, a constant value, or a compound value. To determine what type a parameter value is holding, the functions: *db_param_value_uses_constant_form()* (ael), *db_param_value_uses_string_form()* (ael), and/or *db_param_value_uses_compound_form()* (ael) can be used.

Display Status

displayed status is a boolean variable and denotes if the parameter and its value will be displayed within the schematic or layout view.

Repeatable Status

repeatable status is a boolean; it denotes if the parameter value is part of a repeated parameter.

Example of Creating Default Parameter Values for Parameter Definitions

When defining the default value for a *parameter definition* (ael) within a component *item definition* (ael), the function *prm()* (ael) is used to create a new default parameter value for a *parameter definition* (ael) within ADS.

```
/* TLIN */
      create_item("TLIN",                          // name
          "Ideal 2-Terminal Transmission Line",    // label
          "TL",                                    // prefix
          0,                                       // attribute
          NULL,                                    // priority
          "TLIN",                                  // iconName
          standard_dialog,                         // dialogName
          "*",                                     // dialogData
          ComponentNetlistFmt,                     // netlistFormat
          "TLIN",                                  // netlistData
          ComponentAnnotFmt,                       // displayFormat
          "SYM_TLin",                              // symbolName
          no_artwork,                              // artworkType
          NULL,                                    // artworkData
          ITEM_PRIMITIVE_EX,                       // extraAttrib
          create_parm ("Z", "Characteristic Impedance", PARM_OPTIMIZABLE |
                       PARM_STATISTICAL, "StdFileFormSet",
                       RESISTANCE_UNIT, prm("StdForm","50.0")),
          create_parm ("E", "Electrical Length", PARM_OPTIMIZABLE |
                       PARM_STATISTICAL, "StdFileFormSet", ANGLE_UNIT,
                       prm("StdForm","90")),
          create_parm ("F", "Reference Frequency for Electrical Length",
                       PARM_OPTIMIZABLE | PARM_STATISTICAL, "StdFileFormSet",
                       FREQUENCY_UNIT, prm("StdForm","1 GHz")));
```

Note the calls above to the function *prm()* (ael) to create the default parameter value object for the different *parameter definitions* (ael).

Accessing Parameter Value Data

Get the parameter value for a parameter referenced by a *parameter iterator* (ael)

See *db_param_iter_get_param()* (ael) to get a parameter value for the parameter referenced by the *parameter iterator* (ael).

```
decl paramVal = db_param_iter_get_param(paramIter);
```

Get the default parameter value for a *parameter definition* (ael)

See *dm_parm_get_defvalue()* (ael) to get the default parameter value for a given *parameter definition* (ael).

```
decl paramValH = dm_parm_get_defvalue(parmDefH);
```

Querying Instance Parameter Value Data

A Parameter Value object is useful when there is a need to query or retrieve a component instance's parameter value data, such as what is the value of a parameter and is the parameter displayed.
An *Instance Iterator* (ael) from a *DesignContext* (ael) can be used to traverse a design's instance data. The instance object and/or instance iterator can be used to retrieve a particular component instance's parameter information. The AEL function *db_create_param_iter()* (ael) is used to retrieve the a *parameter iterator* (ael) for a particular instance within a *DesignContext* (ael). See *Parameter Iterator - Overview* (ael)

for more information about *parameter iterators* (ael). The *parameter iterator* (ael) can be used to retrieve different parameter information such as the parameter's value. The AEL function *db_param_iter_get_param()* (ael) can be used to get the parameter value from a *parameter iterator* (ael).

Example of Querying Instance Parameter Value Information

```
decl context = de_get_current_design_context();
decl instIter = db_create_inst_iter(context);
for ( ; db_inst_iter_is_valid(instIter);
      instIter = db_inst_iter_get_next(instIter) )
{
   // Loop through the instance's component parameters
   decl paramIter = db_create_param_iter(instIter);
   for ( ; db_param_iter_is_valid(paramIter);
       paramIter = db_param_iter_get_next(paramIter) )
   {
       // Mark to display only the user modified string parameters.
       decl paramVal = db_param_iter_get_param(paramIter);
       if (db_param_value_uses_string_form(paramVal))
       {
           // Get the string value of the string parameter.
           decl paramValueStr = db_get_param_string_value(paramVal);

           // Get the parameter definition's default parameter value for the parameter.
           decl parmDefP = db_param_iter_get_parm_def(paramIter);
           decl defaultParamVal = dm_parm_get_defvalue(parmDefP);
           decl defaultParamValueStr = db_get_param_string_value(defaultParamVal);

           // Mark to display only the string parameters and their values that
           //  are different than their default value.
           if (defaultParamValueStr != paramValueStr)
           {
               // The instance's parameter value is different
               //  than the component's default parameter value.
               // Mark to display these parameters only for the instance.
               db_set_param_displayed(paramVal, TRUE);
           }
           else
           {
               db_set_param_displayed(paramVal, FALSE);
           }
       }
   }
}
```

List of Parameter Value Functions

See list here: *Parameter Value Functions* (ael).

Pin Functions

- *Pin Overview* (ael)
- *Connectivity Objects Overview* (ael)

This section describes the following functions:

- *db_create_pin()* (ael)
- *db_edit_pin_attributes()* (ael)
- *db_get_pin_name()* (ael)
- *db_get_pin_term()* (ael)
- *db_get_pin_term_name()* (ael)
- *db_get_pin_term_number()* (ael)
- *db_get_pin_term_type()* (ael)
- *db_get_pin_bbox()* (ael)
- *db_get_pin_angle()* (ael)
- *db_get_pin_angle_normalized()* (ael)
- *db_get_pin_snap_point()* (ael)
- *db_get_pin_snap_layerid()* (ael)
- *db_is_pin_selected()* (ael)
- *db_select_pin()* (ael)

See also

Pin Iterator Functions (ael)

db_create_pin()

Creates a new *Pin* (ael) in a given *design context* (ael) and returns the new *Pin* (ael) object. Works for all modifiable *context* (ael) types.

Syntax

```
decl newPin = db_create_pin(context, x, y [, angle, layerId, termNum, termName,
termType] );
```

Where,

- *context* is a given *DesignContext* (ael) to create the *Pin* (ael) within.
- *x,y* are the x,y coordinates to generate the *Pin* (ael) at given in database units.

Optional pin arguments:

- *pinAngle* is a real value normalized *Pin* (ael) angle, where normalized *Pin* (ael) angles are angles where *Pins* (ael) on the right side of a bounding box are at angle zero. If angle is NULL or omitted, the *Pin* (ael) angle will be auto-calculated.
- *layerId* is the *LayerId* (ael) in the *design context* (ael) to create the *Pin* (ael) on. In layout and artwork macro context types: If *layerId* is NULL or omitted, then it will default to the context's current entry *LayerId* (ael).
  In symbol and schematic context types: the *layerId* argument is ignored, the *Pins* (ael) are created on a default schematic *Pin* (ael) layer.
- *termNum* is an integer *terminal* (ael) number to assign to the *Pin* (ael)'s associated *terminal* (ael).
  If termNum is zero, NULL or omitted, then the next available *Term* (ael) number will be found.

> **Note**
> More than one *Pin* (ael) object can physically represent the same *Term* (ael) object; therefore, more than one *Pin* (ael) object can have the exact same *terminal* (ael) number.

- *termName* is a string *terminal* (ael) name to give the associated *Pin* (ael)'s *terminal* (ael).
  If termName is empty, NULL, or omitted, then an appropriate name is automatically chosen.

  > **Note**
  > More than one *Pin* (ael) object can physically represent the same *Term* (ael) object; therefore, more than one *Pin* (ael) object can have the exact same *terminal* (ael) name.

- *termType* is an integer *terminal* (ael) type to assign the *Pin* (ael). The *terminal* (ael) type describes the direction of the *Pin* (ael).
  If termType is NULL or omitted, then the pin *terminal* (ael) type will be set to **IN_OUT_PIN** by default.
  The possible integer *terminal* (ael) types are : **(INPUT_PIN, OUTPUT_PIN, or IN_OUT_PIN)**
  Where:
  INPUT_PIN = 0, represents an input *terminal* (ael).
  OUTPUT_PIN = 1, represents an output *terminal* (ael).
  IN_OUT_PIN = 2 , represents an input-output *terminal* (ael).

Example

```
decl pin1 = db_create_pin(context, x1, y1, NULL, NULL, 1, "P1");
decl pin2 = db_create_pin(context, x2, y2, 180);
decl pin3 = db_create_pin(context, x3, y3);
decl pinZ = db_create_pin(context, x, y, 90, layerId, 4, "PZ", IN_OUT_PIN);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_edit_pin_attributes() (ael)
db_create_pin_iter() (ael)
db_pin_iter_is_valid() (ael)
db_pin_iter_get_next() (ael)

Where Used (ael)

Schematic, Layout

db_edit_pin_attributes()

Returns the modified *Pin* (ael) for a given *Pin* (ael) and the given modified *Pin* (ael) attributes. Useful for editing the *Pin* (ael)'s *Term* (ael) name, *Pin* (ael)'s *Term* (ael) number, *Pin* (ael)'s angle, and *Pin* (ael)'s *Term* (ael) type of a given *Pin* (ael).

Syntax

```
decl dbNewPin = db_edit_pin_attributes(dbPin, pinTermName, pinTermNumber,
pinAngle, pinTermType);
```

Where,

- *dbPin* is the given *Pin* (ael) to modify its *Pin* (ael) attributes.
- *pinTermName* is a string *terminal* (ael)name to give the associated *Pin* (ael)'s *terminal* (ael).

  Note
  More than one *Pin* (ael) object can physically represent the same *Term* (ael) object; therefore, more than one *Pin* (ael) object can have the exact same *terminal* (ael) name.

- *pinTermNumber* is an integer *Term* (ael) number to assign to the *Pin* (ael)'s associated *terminal* (ael).

  Note
  More than one *Pin* (ael) object can physically represent the same *Term* (ael) object; therefore, more than one *Pin* (ael) object can have the exact same *terminal* (ael) number.

- *pinAngle* is a real value normalized *Pin* (ael) angle, where normalized *Pin* (ael) angles are angles where *Pins* (ael) on the right side of a bounding box are at angle zero.
- *pinTermType* is an integer *terminal* (ael) type to assign the *Pin* (ael). The *terminal* (ael) type describes the direction of the *Pin* (ael).
  The possible integer *terminal* (ael) types are : **(INPUT_PIN, OUTPUT_PIN, or IN_OUT_PIN)**
  Where:
  INPUT_PIN = 0, represents an input *terminal* (ael).
  OUTPUT_PIN = 1, represents an output *terminal* (ael).
  IN_OUT_PIN = 2 , represents an input-output *terminal* (ael).

Example

```
dbPin = db_edit_pin_attributes(dbPin, "P1", 1, 135, IN_OUT_PIN);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_create_pin() (ael)
db_create_pin_iter() (ael)
db_pin_iter_is_valid() (ael)
db_pin_iter_get_next() (ael)

Where Used (ael)

Schematic, Layout

db_get_pin_angle_normalized()

Returns the angle of a given *Pin* (ael), normalized (zero degrees is right).
The returned angle units are degrees*1000, so a 90 degree pin returns 90000.

Note
For normalized angles: Pins on the right side of a bbox ("on the right") are angle zero.

Syntax

```
decl angle = db_get_pin_angle_normalized(dbPin);
```

Where,

- *dbPin* is a *Pin* (ael) object or a *Pin* (ael) iterator.

Example

```
decl pinIter = db_create_pin_iter(dbNet);
for ( ; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
   // Get the normalized angle of the pin.
   decl angle = db_get_pin_angle_normalized(pinIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_create_pin() (ael)
db_edit_pin_attributes() (ael)
db_get_pin_bbox() (ael)
db_get_pin_snap_layerid() (ael)
db_get_pin_snap_point() (ael)

Where Used (ael)

Schematic, Layout

db_get_pin_angle()

Returns the angle of a given *Pin* (ael).

Syntax

```
decl angle = db_get_pin_angle(dbPin);
```

Where,

- *dbPin* is a *Pin* (ael) object or a *Pin* (ael) iterator.

Example

```
decl pinIter = db_create_pin_iter(dbNet);
for (; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
   // Get the angle of the pin.
   decl angle = db_get_pin_angle(pinIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_create_pin() (ael)
db_edit_pin_attributes() (ael)
db_get_pin_bbox() (ael)
db_get_pin_snap_layerid() (ael)
db_get_pin_snap_point() (ael)

Where Used (ael)

Schematic, Layout

db_get_pin_bbox()

Returns the bounding box of a given *Pin* (ael).

Syntax

```
decl bbox = db_get_pin_bbox(dbPin);
```

Where,

- *dbPin* is a *Pin* (ael) object or a *Pin* (ael) iterator

Example

```
decl pinIter = db_create_pin_iter(dbNet);
for (; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
   // Get the bounding box of the pin.
   decl bboxH = db_get_pin_bbox(pinIter);
```

```
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_create_pin() (ael)
db_edit_pin_attributes() (ael)
db_get_pin_snap_layerid() (ael)
db_get_pin_snap_point() (ael)

Where Used (ael)

Schematic, Layout

db_get_pin_name()

Returns the name of a given *Pin* (ael) object. The *Pin* (ael) name is not very useful, it is recommended to use *terminal* (ael) names instead. The *Term* (ael) name is more useful, since it may indicate width and can be displayed as annotation, such as "In<0:7>". Please see the function *db_get_pin_term_name()* (ael) for how to get the *pin* (ael)'s *terminal* (ael) name.

Syntax

```
decl pinName = db_get_pin_name(dbPin);
```

Where,

- *dbPin* is an *Pin* (ael) object or a *Pin* (ael) iterator

Example

```
decl pinIter = db_create_pin_iter(dbNet);
for (; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
   decl pinName = db_get_pin_name(pinH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_get_pin_term_name() (ael)
db_create_pin() (ael)
db_edit_pin_attributes() (ael)

Where Used (ael)

Schematic, Layout

db_get_pin_snap_layerid()

Returns the *LayerId* (ael) for the snap layer of a *Pin* (ael). The snap layer of a *Pin* (ael) is the layer of the primary shape (dot) if one exists, otherwise the layer of one of the shapes representing the *Pin* (ael) is used.

Syntax

```
decl layerId = db_get_pin_snap_layerid(dbPin);
```

Where,

- dbPin is a *Pin* (ael) object.

Example

```
decl context = de_get_current_design_context();
decl pinIter = db_create_pin_iter(context);
for( ; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
   decl layerId = db_get_pin_snap_layerid(pinIter);
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_create_pin() (ael)
db_edit_pin_attributes() (ael)
db_get_pin_bbox() (ael)
db_get_pin_snap_point() (ael)

Where Used (ael)

Schematic, Layout

db_get_pin_snap_point()

Returns the snap coordinate point of a *Pin* (ael) object's.

Syntax

```
decl snapCoordH = db_get_pin_snap_point(dbPin);
```

Where,

- *dbPin* is an *Pin* (ael) object or a *Pin* (ael) iterator

Example

```
decl pinIter = db_create_pin_iter(context);
for (; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
   decl snapCoordH = db_get_pin_snap_point(pinIter);
   decl snapX = db_get_x(snapCoordH);
   decl snapY = db_get_y(snapCoordH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_create_pin() (ael)
db_edit_pin_attributes() (ael)
db_get_pin_bbox() (ael)
db_get_pin_snap_layerid() (ael)

Where Used (ael)

Schematic, Layout

db_get_pin_term_name()

Returns the *Term* (ael) name of a given *Pin* (ael) object's *Term* (ael). More than one *Pin* (ael) object can physically represent the same *Term* (ael) object; therefore, more than one *Pin* (ael) object can have the exact same *terminal* (ael) name.

Syntax

```
decl termName = db_get_pin_term_name(dbPin);
```

Where,

- *dbPin* is an *Pin* (ael) object or a *Pin* (ael) iterator

Example

```
decl pinIter = db_create_pin_iter(context);
for (; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
   decl termName = db_get_pin_term_name(pinIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_get_pin_term_number() (ael)
db_get_pin_term_type() (ael)
db_get_pin_term() (ael)
db_create_pin() (ael)
db_edit_pin_attributes() (ael)
Term Overview (ael)
Term Functions (ael)

Where Used (ael)

Schematic, Layout

db_get_pin_term_number()

Returns the *Term* (ael) number of a given *Pin* (ael) object's *Term* (ael). More than one *Pin* (ael) object can physically represent the same *Term* (ael) object; therefore, more than one *Pin* (ael) object can have the exact same *terminal* (ael) number.

Syntax

```
decl termNumber = db_get_pin_term_number(dbPin);
```

Where,

- *dbPin* is an *Pin* (ael) object or a *Pin* (ael) iterator

Example

```
decl pinIter = db_create_pin_iter(context);
for (; db_pin_iter_is_valid(pinIter);
```

```
   pinIter = db_pin_iter_get_next(pinIter))
{
   decl termNum = db_get_pin_term_number(pinIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_get_pin_term_name() (ael)
db_get_pin_term_type() (ael)
db_get_pin_term() (ael)
db_create_pin() (ael)
db_edit_pin_attributes() (ael)
Term Overview (ael)
Term Functions (ael)

Where Used (ael)

Schematic, Layout

db_get_pin_term_type()

Returns an integer *terminal* (ael) type of a given *Pin* (ael) object. The *terminal* (ael) type describes the direction of the *Pin* (ael).

The possible integer *terminal* (ael) types are :

- **INPUT_PIN** = 0, represents an input *terminal* (ael).
- **OUTPUT_PIN** = 1, represents an output *terminal* (ael).
- **IN_OUT_PIN** = 2, represents an input-output *terminal* (ael).

Syntax

```
decl termType = db_get_pin_term_type(dbPin);
```

Where,

- *dbPin* is an *Pin* (ael) object or a *Pin* (ael) iterator

Example

```
decl pinIter = db_create_pin_iter(context);
for (; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
   decl termType = db_get_pin_term_type(pinIter);
   …
```

```
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_get_pin_term_name() (ael)
db_get_pin_term_number() (ael)
db_get_pin_term() (ael)
db_create_pin() (ael)
db_edit_pin_attributes() (ael)
Term Overview (ael)
Term Functions (ael)

Where Used (ael)

Schematic, Layout

db_get_pin_term()

Returns the *Term* (ael) of a given *Pin* (ael) object. More than one *Pin* (ael) object can physically represent the same *Term* (ael) object.

Syntax

```
decl dbTerm = db_get_pin_term(dbPin);
```

Where,

- *dbPin* is an *Pin* (ael) object or a *Pin* (ael) iterator

Example

```
decl pinIter = db_create_pin_iter(dbNet);
for (; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
   decl dbTerm = db_get_pin_term(pinIter);
   decl termName = db_get_term_name(dbTerm);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_get_pin_term_name() (ael)
db_get_pin_term_number() (ael)
db_get_pin_term_type() (ael)
Term Overview (ael)
Term Functions (ael)

Where Used (ael)

Schematic, Layout

db_is_pin_selected()

Returns TRUE if the passed in *Pin* (ael) is selected. Otherwise it returns FALSE if the passed in *Pin* (ael) is not selected.

Note
A *Pin* (ael) is considered selected if any of its shapes are selected.

Syntax

```
decl isPinSelected = db_is_pin_selected(dbPin);
```

Where,

- *dbPin* is a *Pin* (ael) object or a *Pin* (ael) iterator

Example

```
decl context = de_get_current_design_context()
decl pinIter = db_create_pin_iter(context);
for (; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
   // Deselect all selected pins.
   if (db_is_pin_selected(pinIter) == TRUE)
       db_select_pin(pinIter, FALSE);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)

db_select_pin() (ael)
db_is_selected() (ael)
db_create_pin() (ael)
db_edit_pin_attributes() (ael)

Where Used (ael)

Schematic, Layout

db_select_pin()

Function to select or deselect a *Pin* (ael). If applicable, the design will automatically be redisplayed in all windows.

Syntax

```
db_select_pin(dbPin [, select]);
```

Where,

- *dbPin* is a *Pin* (ael) object or a *Pin* (ael) iterator
- *select* is an optional Boolean parameter where if TRUE will select the *Pin* (ael), FALSE will deselect the *Pin* (ael). If the select parameter is not given, then the *Pin* (ael) is selected.

Example

```
decl context = de_get_current_design_context()
decl pinIter = db_create_pin_iter(context);
pinIter = db_pin_iter_limit_selected(pinIter);  // Limit to selected pins only.
for (; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
   // Deselect all selected pins.
   db_select_pin(pinIter, FALSE);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_is_pin_selected() (ael)
db_select() (ael)
db_create_pin() (ael)
db_edit_pin_attributes() (ael)

Where Used (ael)

Schematic, Layout

Pin Iterator Functions

- *Pin Overview* (ael)
- *Connectivity Objects Overview* (ael)

This section describes the following functions:

- *db_create_pin_iter()* (ael)
- *db_pin_iter_is_valid()* (ael)
- *db_pin_iter_get_next()* (ael)
- *db_pin_iter_get_pin()* (ael)
- *db_pin_iter_limit_selected()* (ael)
- *db_pin_iter_limit_region()* (ael)

See also

Pin Functions (ael)

db_create_pin_iter()

Returns a *Pin* (ael) iterator from a given *Net* (ael) or *DesignContext* (ael). If no *pins* (ael) exist for the given *Net* (ael) or *DesignContext* (ael), then the *Pin* (ael) iterator that is returned will be invalid. The function *db_pin_iter_is_valid()* (ael) can be used to test if a *Pin* (ael) iterator is valid.

Syntax

```
decl pinIter = db_create_pin_iter(object);
```

Where,

- *object* is an *Net* (ael) object or *Net* (ael) iterator, or is a *DesignContext* (ael)

Example

```
decl pinIter = db_create_pin_iter(context);
for (; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
   // Get the pin's location.
   decl pinCoordH = db_get_pin_snap_point(pinIter);
   decl pinX = db_get_x(pinCoordH);
   decl pinY = db_get_y(pinCoordH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)

Pin Iterator Functions (ael)
db_pin_iter_is_valid() (ael)
db_pin_iter_get_next() (ael)

Where Used (ael)

Schematic, Layout

db_pin_iter_get_next()

Returns the next *Pin* (ael) iterator from the given *Pin* (ael) iterator.

Syntax

```
decl nextPinIter = db_pin_iter_get_next(pinIter);
```

Where,

- *pinIter* is a valid *Pin* (ael) iterator returned from a function such as *db_create_pin_iter()* (ael)

Example

```
decl pinIter = db_create_pin_iter(context);
for (; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
   // Select the pin.
   db_select_pin(pinIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_create_pin_iter() (ael)
db_pin_iter_is_valid() (ael)

Where Used (ael)

Schematic, Layout

db_pin_iter_get_pin()

Returns the *Pin* (ael) object referred to by the given *Pin* (ael) iterator.

Syntax

```
decl dbPin = db_pin_iter_get_pin(pinIter);
```

Where,

- *pinIter* is a valid *Pin* (ael) iterator returned from a function such as *db_create_pin_iter()* (ael)

Example

```
decl pinIter = db_create_pin_iter(context);
for (; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
   decl pinH = db_pin_iter_get_pin(pinIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_create_pin_iter() (ael)
db_pin_iter_is_valid() (ael)
db_pin_iter_get_next() (ael)

Where Used (ael)

Schematic, Layout

db_pin_iter_is_valid()

Returns TRUE if the *Pin* (ael) iterator is referencing a valid *Pin* (ael).

Syntax

```
decl isValid = db_pin_iter_is_valid(pinIter);
```

Where,

- *pinIter* is a *Pin* (ael) iterator returned from a function such as *db_create_pin_iter()* (ael)

Example

```
decl pinIter = db_create_pin_iter(context);
for (; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
```

```
   // Get the angle of the pin.
   decl angle = db_get_pin_angle(pinIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_create_pin_iter() (ael)
db_pin_iter_get_next() (ael)

Where Used (ael)

Schematic, Layout

db_pin_iter_limit_region()

Function to limit the *Pin* (ael) iteration to a region. If the iterator has been started (used to find a *Pin* (ael)), then calling this function gives an AEL error.

Syntax

```
decl pinIter = db_pin_iter_limit_region(pinIter, x1, y1, x2, y2);
```

Where,

- *pinIter* is a valid *Pin* (ael) iterator returned from a function such as *db_create_pin_iter()* (ael).
- *x1,y1,x2,y2* are coordinates in database units to limit the iteration to.

Example

```
decl context = de_get_current_design_context();

// Do some work on only the pins in a particular region.
decl pinIter = db_create_pin_iter(context);
pinIter = db_pin_iter_limit_region(pinIter, 0, 0, 200, 200);
for (; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
   decl pinH = db_pin_iter_get_pin(pinIter);

   // Select all of pins in a given region.
   db_select_pin(pinH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)
Pin Functions (ael)
Pin Iterator Functions (ael)
db_create_pin_iter() (ael)
db_pin_iter_is_valid() (ael)
db_pin_iter_get_next() (ael)
db_pin_iter_limit_selected() (ael)

Where Used (ael)

Schematic, Layout

db_pin_iter_limit_selected()

Function to limit the *Pin* (ael) iteration to selected *Pins* (ael). If the iterator has been started (used to find a *Pin* (ael)), then calling this function gives an AEL error.

Syntax

```
decl pinIter = db_pin_iter_limit_selected(pinIter);
```

Where,

- *pinIter* is a valid *Pin* (ael) iterator returned from a function such as *db_create_pin_iter()* (ael)

Example

```
// Do some work on only the selected pins.
decl pinIter = db_create_pin_iter(context);
pinIter = db_pin_iter_limit_selected(pinIter);
for (; db_pin_iter_is_valid(pinIter);
   pinIter = db_pin_iter_get_next(pinIter))
{
   decl pinH = db_pin_iter_get_pin(pinIter);

   // De-select all of the selected pins.
   db_select(pinH, FALSE);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Pin Overview (ael)

Pin Functions (ael)
Pin Iterator Functions (ael)
db_create_pin_iter() (ael)
db_pin_iter_is_valid() (ael)
db_pin_iter_get_next() (ael)
db_pin_iter_limit_region() (ael)

Where Used (ael)

Schematic, Layout

Preference Functions

This section describes each Preference function in detail. The functions are listed in alphabetical order.

db	
db_set_curve_radius() (ael) *db_set_miter_cutoff()* (ael)	*db_set_path_corner()* (ael) *db_set_path_width()* (ael)
de_a-r	
de_get_preference() (ael) *de_read_preferences()* (ael) *de_refresh_layers()* (ael)	
de_set_a-f	
de_set_annotation_font() (ael) *de_set_annotation_height()* (ael) *de_set_annotation_id_layer()* (ael) *de_set_annotation_name_layer()* (ael) *de_set_annotation_parameters_layer()* (ael) *de_set_annotation_precision()* (ael) *de_set_annotation_rows()* (ael) *de_set_arc_radius()* (ael) *de_set_background_color()* (ael) *de_set_backup_count()* (ael)	*de_set_coord_entry_popup()* (ael) *de_set_curve_radius()* (ael) *de_set_drag_move()* (ael) *de_set_drag_move_size()* (ael) *de_set_drag_move_units()* (ael) *de_set_dse_start()* (ael) *de_set_dual_placement()* (ael) *de_set_foreground_color()* (ael)
de_set_g-r	
de_set_global_db_factor() (ael) *de_set_grid_color()* (ael) *de_set_grid_display_type()* (ael) *de_set_grid_snap()* (ael) *de_set_grid_snap_mode()* (ael) *de_set_grid_snap_type()* (ael) *de_set_highlight_color()* (ael) *de_set_layer()* (ael) *de_set_major_grid_display()* (ael) *de_set_minor_grid_display()* (ael) *de_set_miter_cutoff()* (ael) *de_set_miter_length()* (ael) *de_set_oversize()* (ael) *de_set_path_corner()* (ael) *de_set_path_width()* (ael) *de_set_pin_color()* (ael) *de_set_pin_size()* (ael)	*de_set_pin_size_units()* (ael) *de_set_pin_snap()* (ael) *de_set_pin_snap_units()* (ael) *de_set_place_popup_mode()* (ael) *de_set_place_popup_on_zero_parm()* (ael) *de_set_plot_pin_names()* (ael) *de_set_plot_pin_numbers()* (ael) *de_set_plot_pins()* (ael) *de_set_plotting_depth()* (ael) *de_set_port_size()* (ael) *de_set_port_size_units()* (ael) *de_set_preference()* (ael) *de_set_reroute_wires()* (ael) *de_set_resolution_for_arc()* (ael) *de_set_rotation_increment()* (ael) *de_set_route_around_annot()* (ael)
de_set_s-u	
de_set_scale() (ael) *de_set_select_box_size()* (ael) *de_set_select_box_units()* (ael) *de_set_select_color()* (ael) *de_set_select_filter()* (ael) *de_set_select_inside_polygon()* (ael) *de_set_select_point_size()* (ael) *de_set_select_point_size_units()* (ael) *de_set_self_intersect()* (ael) *de_set_shape_entry_mode()* (ael) *de_set_step_and_repeat()* (ael) *de_set_tap_length()* (ael) *de_set_tee_color()* (ael)	*de_set_tee_size()* (ael) *de_set_tee_size_units()* (ael) *de_set_text_absolute()* (ael) *de_set_text_angle()* (ael) *de_set_text_font()* (ael) *de_set_text_height()* (ael) *de_set_text_justification()* (ael) *de_set_text_string()* (ael) *de_set_trace_sim_mode()* (ael) *de_set_trace_single_elem()* (ael) *de_set_trace_tech()* (ael) *de_set_trace_traverse()* (ael) *de_set_undo_edit_count()* (ael)
de_touch - set_port	
de_touch() (ael) *de_write_preferences()* (ael) *ly_find_layer_by_name()* (ael)	*ly_find_layer_name_by_num()* (ael) set_port_size_units()

db_set_curve_radius()

Sets the curve radius for paths and traces used when adding paths and traces with curved corners in the given design context.
Returns: none.

Syntax

```
db_set_curve_radius(context, radius);
```

Where,

- *context* is the design context to set the path/trace curve radius for.
- *radius* is the path/trace curve radius in degrees.

Example

```
db_set_curve_radius(context, 20.4);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_path_trace_bend_type() (ael)
db_set_path_corner() (ael)
db_get_path_trace_miter_radius() (ael)
db_set_miter_cutoff() (ael)
db_get_path_trace_width() (ael)
db_set_path_width() (ael)
db_add_path() (ael)
db_end() (ael)

Where Used (ael)

Schematic, Layout

db_set_miter_cutoff()

Sets the miter cutoff angle used when adding paths and traces with mitered corners in the given design context.
Angles less than the given number, result in the point being mitered.
Returns: none.

Syntax

```
db_set_miter_cutoff(context, miterAngle);
```

Where,

- *context* is the design context to set the path/trace miter cutoff angle for.
- *miterAngle* is the cutoff angle in degrees.

Example

```
db_set_miter_cutoff(context, 30.0);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_path_trace_bend_type() (ael)
db_set_path_corner() (ael)
db_get_path_trace_miter_radius() (ael)
db_set_curve_radius() (ael)
db_get_path_trace_width() (ael)
db_set_path_width() (ael)
db_add_path() (ael)
db_end() (ael)

Where Used (ael)

Schematic, Layout

db_set_path_corner()

Sets the corner type used when adding paths and traces in the given design context. Types can be square, mitered or curved.
Returns: none.

Syntax

```
db_set_path_corner(context, cornerType);
```

Where,

- *context* is the design context to set the path/trace corner type for.
- *cornerType* is set to one of the following 3 corner type values:
  - DB_MITERED_CORNER = mitered
  - DB_SQUARE_CORNER = square
  - DB_CURVED_CORNER = curved.

Example

```
db_set_path_corner(context, DB_CURVED_CORNER);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_path_trace_bend_type() (ael)
db_get_path_trace_miter_radius() (ael)
db_set_curve_radius() (ael)
db_set_miter_cutoff() (ael)
db_get_path_trace_width() (ael)

db_set_path_width() (ael)
db_add_path() (ael)
db_end() (ael)

Where Used (ael)

Schematic, Layout

db_set_path_width()

Sets the width for paths and traces used when adding paths and traces in the given design context.
Returns: none.

Syntax

```
db_set_path_width(context, width);
```

Where,

- *context* is the design context to set the path/trace width for.
- *width* is the path/trace width in user units (>0).

Example

```
db_set_path_width(context, 10.5);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_path_trace_width() (ael)
db_get_path_trace_bend_type() (ael)
db_set_path_corner() (ael)
db_get_path_trace_miter_radius() (ael)
db_set_curve_radius() (ael)
db_set_miter_cutoff() (ael)
db_add_path() (ael)
db_end() (ael)

Where Used (ael)

Schematic, Layout

de_get_preference()

Returns the value of a current preference, the preference setting.

Syntax

```
de_get_preference(preference, [repType | repHandle]);
```

Where,

- *preference* is an integer or predefined preference variable, as listed in Preference Variables for de_get_preference().
- *repType* is the type of representation, where:
  - REP_SCHEM = schematic representation
  - REP_LAY = layout representation
- *repHandle* is the handle of a design representation

Preference Variables for de_get_preference()

Path corner types:	
PREF_MITERED_PATH PREF_SQUARE_PATH	PREF_CURVED_PATH
Entry modes:	
PREF_ENTRY_SNAP_NONE PREF_ENTRY_90_SNAP	PREF_ENTRY_45_SNAP
Select filter (bits can be or'd together):	
PREF_NONE_SELECT_FILT PREF_ELEMENT_SELECT_FILT PREF_WIRE_SELECT_FILT PREF_POLYGON_SELECT_FILT PREF_POLYLINE_SELECT_FILT PREF_PATH_SELECT_FILT PREF_TEXT_SELECT_FILT	PREF_ARC_SELECT_FILT PREF_CIRCLE_SELECT_FILT PREF_PORT_SELECT_FILT PREF_FORMAT_SELECT_FILT PREF_POINT_SELECT_FILT PREF_ALL_SELECT_FILT
Grid snapping types:	
PREF_SNAP_TO_GRID PREF_SNAP_TO_PIN PREF_SNAP_TO_EDGE PREF_SNAP_TO_VERTEX	PREF_SNAP_TO_ARC_CENTER PREF_SNAP_TO_INTERSECT PREF_SNAP_TO_MIDPOINT
Grid display types:	
PREF_GRID_DOTS	PREF_GRID_LINES
Placement modes:	
PREF_DUAL_PLACE_OFF PREF_DUAL_PLACE_SINGLE	PREF_DUAL_PLACE_DSE
Selection mode types:	
PREF_SELECT_UNSEL_FIRST	PREF_SELECT_OUTSIDE_WIN
Selection mode for polygons:	
PREF_SELECT_MODE_ON	PREF_SELECT_MODE_INSIDE
Trace conversion simulation mode types:	
PREF_TRACE_SIM_TLIN PREF_TRACE_SIM_SINGLE	PREF_TRACE_SIM_NODAL
Trace conversion technology types:	
PREF_TRACE_TECH_MICROSTRIP PREF_TRACE_TECH_STRIPLINE	PREF_TRACE_TECH_PCB
Schematic annotation modes:	
PREF_INST_TEXT_FORMAT_NONE PREF_INST_TEXT_FORMAT_SHORT	PREF_INST_TEXT_FORMAT_FULL

Preference attribute types:

BACKUP_COUNT_P	OVERSIZE_P
BBOX_COLOR_P	PATH_BEND_P
BG_COLOR_P	PATH_MITER_PERCENT_P
CHECK_BINDING_P	PATH_RADIUS_P
CHECK_INTERSECTION_P	PATH_WIDTH_P
CHECK_NODAL_MISMATCH_P	PIN_COLOR_P
CHECK_PIN_VS_PORT_P	PIN_CURRENT_COLOR_P
CHECK_UNCONNECTED_PINS_ P	PIN_SIZE_P
CHECK_WIRES_IN_LAYOUT_P	PIN_SIZE_UNITS_P
COORD_ENTRY_POPUP_P	PIN_SNAP_SIZE_P
DISP_SUBNET_INST_NAMES_P	PIN_SNAP_UNITS_P
DRAG_MOVE_P	PLACE_POPUP_ON_ZERO_PARM_P
DRAG_MOVE_THRESHOLD_SIZE_P	PLACE_POPUP_P
DRAG_MOVE_THRESHOLD_UNITS_P	PLOT_PIN_NAMES_P
DSE_ART_X_DISTANCE_P	PLOT_PIN_NUMBERS_P
DSE_ART_Y_DISTANCE_P	PLOT_PINS_P
DSE_L2S_REPORT_P	PLOTTING_DEPTH_P
DSE_S2L_REPORT_P	PORT_COLOR_P
DSE_SYMB_X_DISTANCE_P	PORT_NAME_P
DSE_SYMB_Y_DISTANCE_P	PORT_NUMBER_P
DUAL_PLACEMENT_P	PORT_ORIENT_P
DVE_BIN_WIDTH_P	PORT_SIZE_P
DVE_EPSILON_P	PORT_SIZE_UNITS_P
DVE_FRINGE_P	PORT_TYPE_P
DVE_MAX_ERROR_P	REROUTE_WIRES_P
DVE_REAL_MEMORY_P	ROTATION_INC_P
DVE_STORAGE_PER_AREA_P	ROUTE_AROUND_INST_TEXT_P
ENTRY_MODE_P	ROUTE_DIST_SIZE_P
FG_COLOR_P	ROUTE_DIST_UNITS_P
FORCE_DELETE_P	SCALE_X_P
GLOBAL_ARC_RESOLUTION_P	SCALE_Y_P
GRID_COLOR_P	SCHEM_INCR_P
GRID_DISPLAY_MODE_P	SCHEM_PREC_P
GRID_DISPLAY_P	SCHEM_UNITS_P
GRID_DISPLAY_X_P	SELECT_BOX_SIZE_P
GRID_DISPLAY_Y_P	SELECT_BOX_UNITS_P
GRID_SNAP_MODE_P	SELECT_COLOR_P
GRID_SNAP_P	SELECT_FILTER_P
GRID_SNAP_X_P	SELECT_MODE_P
GRID_SNAP_Y_P	SELECT_POINT_SIZE_P
HIGHLIGHT_COLOR_P	SELECT_POINT_UNITS_P
INST_ID_LAYER_P	SHOW_CONNECTED_LAY_P
INST_NAME_LAYER_P	SHOW_CONNECTED_SCHEM_P
INST_PARAM1_LAYER_P	STEP_REPEAT_NUMCOLS_P
INST_TEXT_DOE_FORMAT_P	STEP_REPEAT_NUMROWS_P
INST_TEXT_FONT_P	STEP_REPEAT_XSPACE_P
INST_TEXT_HEIGHT_P	STEP_REPEAT_YSPACE_P
INST_TEXT_OPT_FORMAT_P	SWAP_KEEP_INST_NAME
INST_TEXT_PREC_P	TAP_LENGTH_P
INST_TEXT_ROWS_P	TEE_COLOR_P
INST_TEXT_STAT_FORMAT_P	TEE_SIZE_P
INST_TEXT_TUNE_FORMAT_P	TEE_SIZE_UNITS_P
INSTANCE_ANGLE_P	TEXT_ABSOLUTE_P
INSTANCE_AUTO_ROTATE_P	TEXT_ANGLE_P
INSTANCE_MIRROR_P	TEXT_FONT_P
INSTANCE_NAME_P	TEXT_HEIGHT_P
KEEP_NODE_NAMES_P	TEXT_JUST_P
LAYOUT_INCR_P	TO_ARC_RADIUS_P
LAYOUT_PREC_P	TRACE_MSUB_ID_P
LAYOUT_UNITS_P	TRACE_SIM_MODE_P
MAJOR_GRID_DISPLAY_P	TRACE_SINGLE_ELEM_P
MAJOR_GRID_DISPLAY_X_P	TUNE_HISTORY_SIZE_P
MAJOR_GRID_DISPLAY_Y_P	TUNE_MODE_P
MIN_VERTEX_DIST_P	TUNE_RANGE_P

MITER_ANGLE_P MITER_VERTEX_LENGTH_P NODE_NAME_COLOR_P NODE_VOLT_COLOR_P	TUNE_SCALE_P TUNE_STEP_SIZE_P UNDO_EDIT_COUNT_P VIEW_TYPE_P WINDOW_LL_X WINDOW_LL_Y WINDOW_UR_X WINDOW_UR_Y
Preference value types (units):	
UNITS_FREQ_P UNITS_RES_P UNITS_COND_P UNITS_IND_P UNITS_CAP_P UNITS_LNG_P	UNITS_TIME_P UNITS_ANG_P UNITS_POWER_P UNITS_VOLT_P UNITS_CUR_P UNITS_DIST_P
Preference value types (int, float, string):	
PREF_TYPE_FLOAT_VALUE PREF_TYPE_INT_VALUE	PREF_TYPE_STRING_VALUE

Example

```
decl cornerType;
cornerType = de_get_preference(PATH_BEND_P);
if (cornerType==PREF_MITERED_PATH)
fputs(stderr, "mitered");
else if (cornerType==PREF_SQUARE_PATH)
fputs(stderr, "square");
else
fputs(stderr, "curved");
```

Where Used: (ael)

Schematic, Layout

de_read_preferences()

Reads a preference file from disk, re-setting the preferences in the current window.
Returns: none.

Syntax:

```
de_read_preferences(preferenceFileName);
```

where

preferenceFileName is a string; name of the preference file to read.

Example:

```
de_read_preferences("myprefs.prf");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Layer/Process Manager

Where Used: **(ael)**

Schematic, Layout

de_refresh_layers()

Refreshes the layer information in the windows displaying the current design representation. Returns: None.

See also: de_read_layer() .

Syntax:

```
de_refresh_layers();
```

Example:

```
de_read_layer("mylayer.lay");
de_refresh_layers();
```

Where Used: **(ael)**

Schematic, Layout

de_set_annotation_font()

Sets the font for instance parameter annotation for the current window. Returns: none.

Syntax:

```
de_set_annotation_font(fontNumber);
```

where

fontNumber is an integer indicating text font number, where:

- 0 = Hershey Roman

- 1 = Hershey Roman Narrow

Example:

```
de_set_annotation_font(1);
```

Where Used: **(ael)**

Schematic

de_set_annotation_height()

Sets the text height used for component parameter annotation. Returns: none.

Syntax:

```
de_set_annotation_height(textHeight);
```

where

textHeight is the text height in user units.

Example:

```
de_set_annotation_height(0.1);
```

Where Used: **(ael)**

Schematic

de_set_annotation_id_layer()

Sets the layer number for instance name (ID) parameter annotation for the current window. The layer instance ID, design name and parameters are all separately specified. Returns: none.

Syntax:

```
de_set_annotation_id_layer(layerNum);
```

where

layerNum is the layer number (≥ 0).

Example:

```
de_set_annotation_id_layer(4);
```

Where Used: **(ael)**

Schematic

de_set_annotation_name_layer()

Sets the layer for instance name annotation for the current window. Returns: none.

Syntax:

```
de_set_annotation_name_layer(layerNum);
```

where

layerNum is the layer number (≥ 0).

Example:

```
de_set_annotation_name_layer(5);
```

Where Used: **(ael)**

Schematic

de_set_annotation_parameters_layer()

Sets the layer number for instance parameter annotation for the current window. Returns: none.

Syntax:

```
de_set_annotation_parameters_layer(layerNum);
```

where

layerNum is the layer number to place parameter annotation on (≥ 0).

Example:

```
de_set_annotation_parameters_layer(6);
```

Where Used: **(ael)**

Schematic

de_set_annotation_precision()

Sets the display precision of real numbers displayed in schematic annotation. This is the number of digits displayed to the right of the decimal point for real numbers. This does not affect precision transmitted to the simulator. Returns: none.

Syntax:

```
de_set_annotation_precision(precision);
```

where

precision is the number of digits to the right of the decimal point.

Example:

```
de_set_annotation_precision(3);
```

Where Used: **(ael)**

Schematic

de_set_annotation_rows()

Sets the number of rows for instance parameter annotation for the current window. Instances with more parameters than this number have parameters placed in columns set the left of the first column. Returns: none.

Syntax:

```
de_set_annotation_rows(numRows);
```

where

numRows is an integer (> 0), describing number of rows per column of instance parameter annotation.

Example:

```
de_set_annotation_rows(12);
```

Where Used: **(ael)**

Schematic

de_set_arc_radius()

Set the radius for any arcs created using the *de_vertex_to_arc()* command, which converts a vertex of a shape to an arc for the current window. Returns: none.

Syntax:

```
de_set_arc_radius(radius);
```

where

radius is the radius of arc in degrees.

Example:

```
de_set_arc_radius(20.0);
```

Where Used: **(ael)**

Schematic, Layout

de_set_background_color()

Sets the background color of the drawing area in the current window. Returns: TRUE if successful, FALSE if not successful.

Syntax:

```
de_set_background_color(colorNum);
```

where

colorNum is the number of defined color (integer > 0).

Example:

```
de_set_background_color(14);
```

Where Used: **(ael)**

Schematic, Layout

de_set_backup_count()

Sets the number of edits that must be performed before a backup file is made. Returns:

TRUE if successful, FALSE if not successful. When set to zero, auto_backup will be disabled.

Syntax:

```
de_set_backup_count(count);
```

where

count is an integer; the number of editing operations to perform before automatically writing a .bak backup file of the current design.

Example:

```
de_set_backup_count(10);
```
or
```
de_set_backup_count(0);
```

Where Used: **(ael)**

Schematic, Layout

de_set_coord_entry_popup()

Sets whether the Coordinate Entry dialog will be displayed for draw and place commands.
Returns: none.

Syntax:

```
de_set_coord_entry_popup(onOffFlag);
```

where

onOffFlag signals displaying Coordinate Entry dialog, where:

- TRUE = display
- FALSE = do not display

Example:

```
de_set_coord_entry_popup(TRUE);
```

Where Used: **(ael)**

Schematic, Layout

de_set_curve_radius()

Sets the curve radius in the current window used for paths and traces with rounded corners. Returns: none.

Syntax:

```
de_set_curve_radius(radius);
```

where

radius is the radius in degrees.

Example:

```
de_set_curve_radius(20.4);
```

Where Used: **(ael)**

Schematic, Layout

de_set_drag_move()

Sets the ability to drag and move objects with the left mouse button. Returns: none.

See also: *de_set_drag_move_size()* (ael), *de_set_drag_move_units()* (ael).

Syntax:

```
de_set_drag_move(mode);
```

where

mode is the type of ability to drag and move objects with left mouse button, where:

- 0 = off
- 1 = on

Example:

```
de_set_drag_move(1);
```

Where Used: **(ael)**

Schematic, Layout

de_set_drag_move_size()

Sets the minimum distance an object must be dragged before the object is moved.
Returns: none.

See also: *de_set_drag_move_units()* (ael).

Syntax:

```
de_set_drag_move_size(size);
```

where

size is the size for distance (in user units).

Example:

```
de_set_drag_move_size(12.4);
```

***Where Used:* (ael)**

Schematic, Layout

de_set_drag_move_units()

Sets the units of the drag move. Returns: none.

See also: *de_set_drag_move_size()* (ael).

Syntax:

```
de_set_drag_move_units(unit);
```

where

unit is the type of units, where:

- 0 = user units
- 1 = screen pixels

Example:

```
de_set_drag_move_units(1); //sets to screen pixels
```

***Where Used:* (ael)**

Schematic, Layout

de_set_dse_start()

Sets the starting instance for design synchronization in the current representation. Returns: none.

Syntax:

```
de_set_dse_start(x,y);
```

where

x,y is the point within select region of starting instance.

Example:

```
de_set_dse_start(10,20);
```

Where Used: **(ael)**

Schematic, Layout

de_set_dual_placement()

Sets the mode so that when enabled, causes any component placed in one representation to be automatically placed in the other representation. Returns: none.

Syntax:

```
de_set_dual_placement(dualPlaceMode);
```

where

dualPlaceMode is the placement choice, where:

- PREF_DUAL_PLACE_DSE = automatically perform a design synchronization after each part is placed
- PREF_DUAL_PLACE_SINGLE = automatically place connected equivalent component in other representation
- PREF_DUAL_PLACE_OFF = do not place automatically

Example:

```
de_set_dual_placement(PREF_DUAL_PLACE_SINGLE);
```

Where Used: **(ael)**

Schematic, Layout

de_set_foreground_color()

Sets the sketch color of the active window. Sets the color of the sketch mode for polygons, polylines, wires, traces, selection outlines, etc. Returns: TRUE if successful, FALSE if not successful.

Syntax:

```
de_set_foreground_color(color);
```

where

color is the color number, an integer ≥ 0.

Example:

```
de_set_foreground_color(2);
```

Where Used: **(ael)**

Schematic, Layout

de_set_global_db_factor()

Sets the simulator units to layout user unit conversion factor used during rendering of AEL artwork objects.

Syntax:

```
de_set_global_db_factor();
```

Example:

```
de_set_global_db_factor();
```

Where Used: **(ael)**

Layout

de_set_grid_color()

Sets the color of the visible grid in the current window. Returns: TRUE if successful, FALSE if not successful.

Syntax:

```
de_set_grid_color(color);
```

where

color is the color number, an integer ≥ 0.

Example:

```
de_set_grid_color(5);
```

Where Used: **(ael)**

Schematic, Layout

de_set_grid_display_type()

Sets the grid display to dots or dotted line for the current window. Returns: TRUE if successful, FALSE if not successful.

Syntax:

```
de_set_grid_display_type(type);
```

where

type is the type of grid display, where;

- PREF_GRID_DOTS = dotted
- PREF_GRID_LINES = lines

Example:

```
de_set_grid_display_type(PREF_GRID_DOTS);
```

Where Used: **(ael)**

Schematic, Layout

de_set_grid_snap()

Sets grid snap increments in X and Y direction and turns grid snapping for the current window on or off. Returns: none.

Syntax:

Syntax:

```
de_set_grid_snap(SnapX, SnapY, onOffFlag);
```

where

SnapX is the cursor snapping measurement in X direction.

SnapY is the cursor snapping measurement in Y direction.

onOffFlag signals grid snap setting, where:

- 0 = Off
- 1 = On

Example:

```
de_set_grid_snap(10,20,1);
```

Where Used: **(ael)**

Schematic, Layout

de_set_grid_snap_mode()

Turns grid snapping for the current window on or off. Returns: none.

Syntax:

```
de_set_grid_snap_mode(Mode);
```

where

Mode enables or disables grid snapping, where:

- TRUE = Enable
- FALSE = Disable

Example:

```
de_set_grid_snap_mode(TRUE);
```

Where Used: **(ael)**

Schematic, Layout

de_set_grid_snap_type()

Sets the snap type to one or more of grid, pin, or vertex for the current window. Returns: none.

Syntax:

```
de_set_grid_snap_type(type);
```

where

type is the type of snap, where:

- PREF_SNAP_TO_GRID = cursor will snap to the nearest grid point
- PREF_SNAP_TO_PIN = cursor will snap to the nearest pin
- PREF_SNAP_TO_VERTEX = cursor will snap to the nearest vertex on a shape
- PREF_SNAP_TO_EDGE = cursor will snap and slide along the nearest edge of a shape
- PREF_PREF_SNAP_TO_MIDPOINT = cursor will snap to the midpoint of a line if the midpoint is within the snap radius
- PREF_SNAP_TO_ARC_CENTER = cursor will snap to the center of an arc or circle
- SNAP_TO_INTERSECT = cursor will snap to the intersection of lines or edges of two different objects

Note
For details on grid snap mode behavior, refer to *Schematic Capture and Layout* (usrguide).

Example:

```
de_set_grid_snap_type(PREF_SNAP_TO_GRID|PREF_SNAP_TO_PIN);
```

Where Used: **(ael)**

Schematic, Layout

de_set_highlight_color()

Sets the highlight color for the current window (used for errors, and show commands). Returns: TRUE if successful, FALSE if not successful.

Syntax:

```
de_set_highlight_color(color);
```

where

color is the color for highlighting, an integer ≥ 0.

Example:

```
de_set_highlight_color(13);
```

Where Used: **(ael)**

Schematic, Layout

de_set_layer()

Sets the entry layer for the current layer for the current window. All subsequent shapes and text are added to this layer, until reset. Returns: none.

See also: de_add_layer() , de_remove_all_layers() .

Syntax:

```
de_set_layer(layerNum);
```

where

layerNum is the layer to set, an integer ≥ 0.

Example:

```
de_set_layer(15);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Polylines to Polygons
Polygons to Circles
Polylines to Circles

Where Used: **(ael)**

Schematic, Layout

de_set_major_grid_display()

Sets the major grid display factor. Returns: TRUE if successful, FALSE if not successful.

Syntax:

```
de_set_major_grid_display(xFactor, yFactor, onOff);
```

where

xFactor is the multiple of the snap grid in X.

yFactor is the multiple of the snap grid in Y.

onOff is the grid display options, where:

- TRUE = Display major grid
- FALSE = Do not display major grid

Example:

```
de_set_major_grid_display(8, 8, TRUE);
```

Where Used: **(ael)**

Schematic, Layout

de_set_minor_grid_display()

Sets the minor grid display factor to *xFactor* multiple of the snap grid in X and *yFactor* multiple of the snap grid in Y. Returns: TRUE if successful, FALSE if not successful.

Syntax:

```
de_set_minor_grid_display(xFactor, yFactor, onOff);
```

where

xFactor is a multiple of the snap grid in X.

yFactor is a multiple of the snap grid in Y.

onOff is the grid display options, where:

- TRUE = Display minor grid
- FALSE = Do not display minor grid

Example:

```
de_set_minor_grid_display(10, 10, TRUE);
```

Where Used: **(ael)**

Schematic, Layout

de_set_miter_cutoff()

Sets the miter cutoff angle for the path and trace command. Angles less than the given number, result in the point being mitered. Returns: none.

Syntax:

```
de_set_miter_cutoff(miterAngle);
```

where

miterAngle is the cutoff angle in degrees.

Example:

```
de_set_miter_cutoff(30.0);
```

Where Used: **(ael)**

Layout

de_set_miter_length()

Sets the miter length. It is often used in the *de_miter_vertex()* command, where both edges of a vertex need to be larger than the miter length for the command to be successful. Returns: none.

Syntax:

```
de_set_miter_length(length);
```

where

length is the length in user units (> 0).

Example:

```
de_set_miter_length(12.4);
```

Where Used: **(ael)**

Layout

de_set_oversize()

Sets the amount to oversize or undersize closed shapes for the *de_oversize()* command for the current window. Returns: none.

Syntax:

```
de_set_oversize(oversizeAmount, angle);
```

where

oversizeAmount is the amount to size shapes, where:

- positive numbers = oversize shapes
- negative numbers = undersize shapes

angle is the angle in degrees.

Example:

```
de_set_oversize(-5, 45);
```

Where Used: **(ael)**

Schematic, Layout

de_set_path_corner()

Sets the corner type used for paths and traces in the current window. Types can be square, mitered or curved. Returns: none.

Syntax:

```
de_set_path_corner(type);
```

where

type is the corner type where:

- PREF_MITERED_PATH = mitered
- PREF_SQUARE_PATH = square
- PREF_CURVED_PATH = curved

Example:

```
de_set_path_corner(PREF_MITERED_PATH);
```

Where Used: **(ael)**

Layout

de_set_path_width()

Sets the width for paths and traces used by the *de_add_path()* and *de_add_trace()* entry commands for the current window. Returns: none.

Syntax:

```
de_set_path_width(width);
```

where

width is the path width in user units (> 0).

Example:

```
de_set_path_width(10.8);
```

Where Used: **(ael)**

Layout

de_set_pin_color()

Sets the color used for instance pins for the current window. Returns: none.

Syntax:

```
de_set_pin_color(colorNum);
```

where

colorNum is the color number; an integer ≥ 0.

Example:

```
de_set_pin_color(12);
```

Where Used: **(ael)**

Schematic, Layout

de_set_pin_size()

Sets the size used to display instance pins of the current window. Instance pins are drawn as squares; this controls their size. Returns: none.

See also: *de_set_pin_size_units()* (ael).

Syntax:

```
de_set_pin_size(size);
```

where

size is the size to draw pins.

Example:

```
de_set_pin_size(12.4);
```

***Where Used:* (ael)**

Schematic, Layout

de_set_pin_size_units()

Sets the units for the pin size set by *de_set_pin_size()*. Returns: none.

See also: *de_set_pin_size()* (ael).

Syntax:

```
de_set_pin_size_units(unit);
```

where

unit is the type of units for pin size setting, where:

- 0 = user units
- 1 = screen pixels

Example:

```
de_set_pin_size_units(1); //sets to screen pixels
```

***Where Used:* (ael)**

Schematic, Layout

de_set_pin_snap()

Sets the maximum distance at which the cursor snaps to a pin when snap type includes PREF_SNAP_TO_PIN. Returns: none.

See also: *de_set_pin_snap_units()* (ael).

Syntax:

```
de_set_pin_snap(snapDist);
```

where

snapDist is the maximum distance at which cursor snaps to a pin.

Example:

```
de_set_pin_snap(10.0);
```

Where Used: **(ael)**

Schematic, Layout

de_set_pin_snap_units()

Sets the units for the pin snap distance set by *de_set_pin_snap()*. Returns: none.

See also: *de_set_pin_snap()* (ael).

Syntax:

de_set_pin_snap_units(unit);

where

unit is the type of units for snap distance.

- 0 = user units
- 1 = screen pixels

Example:

```
de_set_pin_snap_units(1); //sets to screen pixels
```

Where Used: **(ael)**

Schematic, Layout

de_set_place_popup_mode()

Sets mode for displaying the component Parameter dialog box for the current window. Returns: none.

Syntax:

```
de_set_place_popup_mode(onOffFlag);
```

where

onOffFlag signals dialog popup when instance is placed, where:

- 0 = do not popup dialog
- 1 = popup dialog

Example:

```
de_set_place_popup_mode(1);
```

Where Used: **(ael)**

Schematic, Layout

de_set_place_popup_on_zero_parm()

Sets mode for the current window for displaying the Component Parameter dialog box for components with no parameters. Returns: none.

See also: *de_get_preference()* (ael).

Syntax:

```
de_set_place_popup_on_zero_parm(onOffFlag);
```

where

onOffFlag signals dialog popup when a component with no parameters is chosen, where:

- TRUE = popup dialog
- FALSE = do not popup dialog

Example:

```
de_set_place_popup_on_zero_parm(TRUE);
```

***Where Used:* (ael)**

Schematic, Layout

de_set_plot_pin_names()

Turns the display of pin names on or off. Returns: none.

Syntax:

de_set_plot_pin_names(onOff);

where

onOff signals displaying of pins, where:

- FALSE = pin names do not display
- TRUE = pin names display

Example:

```
de_set_plot_pin_names( TRUE );
```

***Where Used:* (ael)**

Schematic, Layout

de_set_plot_pin_numbers()

Turns the plotting of pin numbers on and off for the current window. Returns: none.

Syntax:

```
de_set_plot_pin_numbers(onOffFlag);
```

where

onOffFlag signals plotting of pins, where:

- 0 = do not plot pin numbers
- 1 = plot them

Example:

```
de_set_plot_pin_numbers(0);
```

***Where Used:* (ael)**

Schematic, Layout

de_set_plot_pins()

Turns plotting of connected pins on or off for the current window. Returns: none.

Syntax:

de_set_plot_pins(onOffFlag);

where

onOffFlag signals plotting of pins, where:

- 0 = plot all pins
- 1 = plot only unconnected pins

Example:

```
de_set_plot_pins(1);
```

***Where Used:* (ael)**

Schematic, Layout

de_set_plotting_depth()

Sets the hierarchical level at which component instances in Layout are drawn in outline. Returns: none.

Syntax:

```
de_set_plotting_depth(depth);
```

where

depth is the depth is an integer greater than 1.

Example:

```
de_set_plotting_depth(4);
```

***Where Used:* (ael)**

Layout

de_set_port_size()

Sets the size of ports in the Layout representation. Returns: none.

See also: *de_set_port_size_units()* (ael).

Syntax:

```
de_set_port_size(size);
```

where

size is the size to draw ports.

Example:

```
de_set_port_size(12.4);
```

Where Used: **(ael)**

Schematic, Layout

de_set_port_size_units()

Sets the units for the port size set by *de_set_port_size()*. Returns: none.

See also: *de_set_port_size()* (ael).

Syntax:

```
de_set_port_size_units(unit);
```

where

unit is the type of units for port size setting, where:

- 0 = user units
- 1 = screen pixels

Example:

```
de_set_port_size_units(1); //sets to screen pixels
```

Where Used: **(ael)**

Schematic, Layout

de_set_preference()

Sets the value of any preference. Refer to *de_get_preference()* (ael) for preferences and values. Returns: none.

See also: *de_get_preference()* (ael), *de_set_resolution_for_arc()* (ael), *de_set_undo_edit_count()* (ael).

Syntax:

```
de_set_preference(preference, prefValue, [repType | repHandle]);
```

where

preference is an integer or predefined preference variable.

prefValue is an integer or predefined preference value variable.

repType is the type of representation, where:

- REP_SCHEM = schematic representation
- REP_LAY = layout representation

repHandle is the handle of a design representation

Example:

```
// Set the Trace conversion technology type to PCB
// In Schem/Lay window Options > Preferences Trace tab,
// under Element Set
de_set_preference(TRACE_TECH_P, PREF_TRACE_TECH_PCB);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Flatten Design
Generate SnP Component
Select All on Layer

Where Used: (ael)

Schematic, Layout

de_set_reroute_wires()

Sets wire or trace editing route mode for the current window. When a wire or trace is re-routed after an instance is moved, the wire or trace can be completely re-routed or rerouted only from its end point. Returns: none.

Syntax:

```
de_set_reroute_wires(routeMode);
```

where

routeMode signals routing of wires or traces, where:

- 0 = reroute entire wire
- 1 = route from end point only

Example:

```
de_set_reroute_wires(1);
```

Where Used: **(ael)**

Schematic, Layout

de_set_resolution_for_arc()

Sets the resolution that is used for approximating an arc when shapes with true arcs are converted to polygons with the *de_convert_to_polygon()* command for the current window. Returns: none.

Syntax:

```
de_set_resolution_for_arc(degree);
```

where

degree is the number for resolution, in degrees.

Example:

```
de_set_resolution_for_arc(5.0);
```

Where Used: **(ael)**

Schematic, Layout

de_set_rotation_increment()

Sets the step increment used by the *de_rotate()* command for the current window. Returns: none.

Syntax:

```
de_set_rotation_increment(angle);
```

where

angle is the angle in degrees for the step (≥ 0, < 360).

Example:

```
de_set_rotation_increment(45.0);
```

Where Used: **(ael)**

Schematic, Layout

de_set_route_around_annot()

When the Wire Route command is in progress, this mode determines if the wire should route around or through annotations. Returns: none.

Syntax:

```
de_set_route_around_annot(routeAroundMode);
```

where

routeAroundMode is the mode for routing wire, where:

- 0 = route through annotation
- 1 = route around annotation

Example:

```
de_set_route_around_annot(0);
```

Where Used: **(ael)**

Schematic

de_set_scale()

Set the X- and Y-scale factors used by the *de_scale()* command in the current window. Returns: none.

Syntax:

```
de_set_scale(sx, sy);
```

where

sx,sy is the X and Y scale factors (> 0).

Example:

```
de_set_scale(.5,.5);
```

Where Used: **(ael)**

Schematic, Layout

de_set_select_box_size()

Sets the select region size for the current window. Objects within a square of the size specified here are selected when any selection command is given. Returns: none.

See also: *de_set_select_box_units()* (ael).

Syntax:

```
de_set_select_box_size(size);
```

where

sized describes size of select region (side of select region square); size > 0.

Example:

```
de_set_select_box_size(4.0);
```

Where Used: **(ael)**

Schematic, Layout

de_set_select_box_units()

Sets the units for the select region size, set by *de_set_select_box_size()*, at the current window. Returns: none.

See also: *de_set_select_box_size()* (ael).

Syntax:

```
de_set_select_box_units(unit);
```

where

unit is the type of units for select region setting, where:

- 0 = user units
- 1 = screen pixels

Example:

```
de_set_select_box_units(1); //sets to screen pixels
```

Where Used: **(ael)**

Schematic, Layout

de_set_select_color()

Sets the color used to display selected objects for the current window. Returns: TRUE if successful; FALSE if not successful.

Syntax:

```
de_set_select_color(color);
```

where

color is the color to use for selected objects; an integer ≥ 0.

Example:

```
de_set_select_color(10);
```

Where Used: **(ael)**

Schematic, Layout

de_set_select_filter()

Sets the select filter mask for the current window. This controls what type of objects can be selected by the select commands. Returns: none.

Syntax:

```
de_set_select_filter(mask);
```

where

mask is one or more of the following:

- PREF_ARC_SELECT_FILT
- PREF_CIRCLE_SELECT_FILT
- PREF_ELEMENT_SELECT_FILT
- PREF_FORMAT_SELECT_FILT
- PREF_PATH_SELECT_FILT
- PREF_POINT_SELECT_FILT
- PREF_POLYGON_SELECT_FILT
- PREF_POLYLINE_SELECT_FILT
- PREF_PORT_SELECT_FILT
- PREF_TEXT_SELECT_FILT
- PREF_WIRE_SELECT_FILT
- PREF_NONE_SELECT_FILT
- PREF_ALL_SELECT_FILT

Example:

```
de_set_select_filter
   (PREF_POLYGON_SELECT_FILT|PREF_POLYLINE_SELECT_FILT);
// this means only polygons and polylines can be selected
```

Where Used: **(ael)**

Schematic, Layout

de_set_select_inside_polygon()

Sets the selection mode. Objects can be selected when the given point is enclosed by any closed shape (the default in Schematic window) or when any edge of an object is within the select region. Returns: none.

Syntax:

```
de_set_select_inside_polygon(onOffFlag);
```

where

onOffFlag signals selection of objects, where:

- PREF_SELECT_MODE_ON = 0
- PREF_SELECT_MODE_INSIDE = 1

Example:

```
de_set_select_inside_polygon(PREF_SELECT_MODE_INSIDE);
```

Where Used: **(ael)**

Schematic, Layout

de_set_select_point_size()

Sets the size of the selected vertices in the current window. Returns: none.

See also: *de_set_select_point_size_units()* (ael).

Syntax:

de_set_select_point_size(size);

where

size describes size of selected vertex (2 · size = side of select region square); size > 0.

Example:

```
de_set_select_point_size(4.0);
```

Where Used: **(ael)**

Schematic, Layout

de_set_select_point_size_units()

Sets the units for the selected vertices in the current window. Returns: none.

See also: *de_set_select_point_size()* (ael).

Syntax:

```
de_set_select_point_size_units(unit);
```

where

unit is the type of units for select point size setting, where:

- 0 = user units
- 1 = screen pixels

Example:

```
de_set_select_point_size_units(1); //sets to screen pixels
```

Where Used: **(ael)**

Schematic, Layout

de_set_self_intersect()

Sets the polygon self intersection checking for the current window. Returns: none.

Syntax:

```
de_set_self_intersect(onOffFlag);
```

where

onOffFlag is the setting for polygon self intersection checking, where:

- 0 = off
- 1 = on

Example:

```
de_set_self_intersect(1);
```

Where Used: **(ael)**

Schematic, Layout

de_set_shape_entry_mode()

Sets the entry mode in the current window for polygons and polylines to orthogonal or non-orthogonal. Returns: none.

Syntax:

```
de_set_shape_entry_mode(mode);
```

where

mode is the type of entry mode, where:

- 0 = non-orthogonal
- 1 = orthogonal

Example:

```
de_set_shape_entry_mode(1);
```

Where Used: **(ael)**

Schematic, Layout

de_set_step_and_repeat()

Sets variables for the Step and Repeat command. Returns: none.

NOTE
X and Y spacing is defined as the distance between the outer edges of the respective bounding boxes.

Syntax:

```
de_set_step_and_repeat(xSpacing, ySpacing, numRows, numCols[, connectFlag]);
```

where

xSpacing is the space between columns.

ySpacing is the space between rows.

numRows is the number of rows.

numCols is the number of columns.

connectFlag is an optional parameter, defaulted to FALSE. Set to TRUE if connection of overlapping pins and wire ends is desired. Note: in a large design, this increases the processing time.

Example:

```
de_step_and_repeat(0.125, 0.125, 2, 2, FALSE);
```

Where Used: **(ael)**

Schematic, Layout

de_set_tap_length()

Sets the length (w3 parameters) of the tee element produced when a transmission line element is tapped with the *de_tap_tlin()* command. Returns: none.

Syntax:

```
de_set_tap_length(length);
```

where

length is the length in simulator units (> 0). For valid ranges for the M3 parameter, refer to the data sheet for *mtee* or *stee* elements.

Example:

```
de_set_tap_length(12.4);
```

Where Used: **(ael)**

Layout

de_set_tee_color()

Sets the color of schematic tees (interconnect dots) for the current window. Returns: none.

Syntax:

```
de_set_tee_color(colorNum);
```

where

colorNum is the color to use for tees; an integer ≥ 0.

Example:

```
de_set_tee_color(12);
```

Where Used: **(ael)**

Schematic, Layout

de_set_tee_size()

Sets the size of tees (interconnect dots) used in schematics (for the current window). Returns: none.

See also: *de_set_tee_size_units()* (ael).

Syntax:

```
de_set_tee_size(size);
```

where

size is the size (tees are squares, this specifies length of a side) in user-defined units.

Example:

```
de_set_tee_size(50.0);
```

Where Used: **(ael)**

Schematic, Layout

de_set_tee_size_units()

Sets the type of units of the tee size. Returns: none.

See also: *de_set_tee_size()* (ael).

Syntax:

```
de_set_tee_size_units(unit);
```

where

unit is the type of units for tee size setting, where:

- 0 = user units
- 1 = screen pixels

Example:

```
de_set_tee_size_units(1); //sets to screen pixels
```

Where Used: **(ael)**

Schematic, Layout

de_set_text_absolute()

Sets absolute characteristics for text.

Syntax:

```
de_set_text_absolute(onOFF);
```

where

onOFFsignals absolute characteristics for text, where:

- 0 = text rotates relative to custom symbol
- 1 = text does not rotate relative to custom symbol

Example:

```
de_set_text_absolute(1);
//text does not rotate when custom symbol is rotated when placed in design
```

Where Used: **(ael)**

Schematic, Layout

de_set_text_angle()

Sets the angle in which text is placed for the current window. Returns: none.

Syntax:

```
de_set_text_angle(angle);
```

where

angle is the angle in degrees (≥ 0).

Example:

```
de_set_text_angle(12.7);
```

Where Used: **(ael)**

Schematic, Layout

de_set_text_font()

Sets the font type when placing text for the current window. Returns: none.

Syntax:

```
de_set_text_font(fontNum);
```

where

fontNum is the integer indicating text font number, where:

- 0 = Hershey Roman
- 1 = Hershey Roman Narrow

Example:

```
de_set_text_font(0);
```

Where Used: **(ael)**

Schematic, Layout

de_set_text_height()

Sets the height for placed text (in the current window). Returns: none.

Syntax:

```
de_set_text_height(height);
```

where

height is the height in user units (> 0).

Example:

```
de_set_text_height(12.4);
```

Where Used: **(ael)**

Schematic, Layout

de_set_text_justification()

Sets the default text justification. Used for all subsequent text placement. Returns: none.

Syntax:

```
de_set_text_justification(justification);
```

where

justification is an "or'd" combination of one of these:

- DB_BOT_JUST
- DB_MID_JUST
- DB_TOP_JUST

with one of these:

- DB_LEFT_JUST
- DB_CENTER_JUST
- DB_RIGHT_JUST

Example:

```
de_set_text_justification(DB_BOT_JUST | DB_LEFT_JUST);
```

Where Used: **(ael)**

Schematic, Layout

de_set_text_string()

Sets the text string used by the *de_add_text()* command for the current window. This is the string that is placed when the *de_add_text()* command is issued without a text string. Returns: none.

See also: *de_add_text()* (ael).

Syntax:

```
de_set_text_string(string);
```

where

string is the text string enclosed in quotes.

Example:

```
de_set_text_string("This is a text string");
```

Where Used: **(ael)**

Schematic, Layout

de_set_trace_sim_mode()

Sets the trace simulation mode for the current window. Traces can be simulated as a simple interconnection (short), as a single element or as a series of interconnected transmission line elements with discontinuity. Currently, this command has no effect. Traces are simulated as simple nodal interconnections (shorts). Returns: none.

Syntax:

```
de_set_trace_sim_mode(mode);
```

where

mode is the choice for trace simulation, where:

- 0 = full discontinuities
- 1 = single element
- 2 = short

Example:

```
de_set_trace_sim_mode(1);
```

Where Used: **(ael)**

Schematic, Layout

de_set_trace_single_elem()

Sets the name of the element used when trace simulation is set to a single element. Currently, this command has no effect. All traces are simulated as simple nodal interconnections (shorts). Returns: none.

Syntax:

```
de_set_trace_single_elem(itemName);
```

where

itemName is the name of the component to simulate trace as. This component can have a width (W) and length (L) parameter. If they exist, L is set to the total trace length and W is set to the total trace width. This applies to traces connected pin-to-pin only (without tees).

Example:

```
de_set_trace_single_elem("MLIN");
```

Where Used: **(ael)**

Schematic, Layout

de_set_trace_tech()

Sets the technology type for converting/simulating traces to microstrip, stripline, or PCB for the current window. Currently, this only affects the conversion of traces to transmission line elements. It is ignored for the simulation of traces. Returns: none.

Syntax:

```
de_set_trace_tech(technology);
```

where

technology is the choice for technology type, where:

- PREF_TRACE_TECH_MICROSTRIP = microstrip
- PREF_TRACE_TECH_STRIPLINE = stripline
- PREF_TRACE_TECH_PCB = printed circuit board

Example:

```
de_set_trace_tech(PREF_TRACE_TECH_STRIPLINE);
```

***Where Used:* (ael)**

Layout

de_set_trace_traverse()

Sets the trace traversal mode. When traces are converted to transmission line elements, complex traces (traces with tee junctions) can be fully converted, or the trace converted only up to the tee. Returns: none.

Syntax:

```
de_set_trace_traversal(mode);
```

where

mode is the choice for trace conversion, where:

- 0 = partial trace conversion
- 1 = full trace conversion

Example:

```
de_set_trace_traverse(0);
```

***Where Used:* (ael)**

Layout

de_set_undo_edit_count()

Sets the number of edit commands that can be undone. Returns: none.

Syntax:

```
de_set_undo_edit_count(count);
```

where

count is the number of edits that can be undone.

Example:

```
de_set_undo_edit_count(100);
```

***Where Used:* (ael)**

Schematic, Layout

de_touch()

Selects polygons on one layer (inLayer1) in relation to edges of polygons on another layer (inLayer2) if TouchCondition is true. Outputs data to opLayer polygon layer.

Syntax:

```
de_touch(inlayer1, inlayer2, opLayer, ACCEPT/REJECT, "TouchCondition")
```

where
inLayer1 - input layer 1
inLayer2 - input layer 2
opLayer - ouput layer
ACCEPT - selects the matching polygons
REJECTS - does not select the matching polygons
TouchCondition- Can be one of these - ">n" or "<n" or "==n" or "!=n" where n is a +ve integer

Example:

```
de_touch( enter correct variables );
```

Where Used: **(ael)**

Layout

de_write_preferences()

Writes the current preference settings of the active window to a file. Preferences include grid, object (not layer) colors, text attributes, select filters, modes. Returns: none.

Syntax:

```
de_write_preferences(fileName);
```

where

fileName is the name of the preference file.

Example:

```
de_write_preferences("myPreferences.prf");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.

Layer/Process Manager

***Where Used:* (ael)**

Schematic, Layout

ly_find_layer_by_name()

Retrieves a layer number given its name, for active layer set. Returns an integer, the layer number (-1 if the layer is not defined).

See also: *ly_find_layer_name_by_num()* (ael).

Syntax:

```
ly_find_layer_by_name(layerName);
```

where

layerName is the name of a layer; a string.

Example:

```
decl num;
num = ly_find_layer_by_name("conn1");
```

***Where Used:* (ael)**

Schematic, Layout

ly_find_layer_name_by_num()

Retrieves a layer name by its layer number, for the active layer set. Returns a string, the first layer defined with the given layer number (-1 if the layer is not defined).

Syntax:

```
ly_find_layer_name_by_num(layerNumber);
```

where

layerNumber is the layer number; an integer.

Example:

```
decl name;
name = ly_find_layer_name_by_num(12);
```

***Where Used:* (ael)**

Schematic, Layout

Primitive Polygon Functions

- ***Primitive Polygon Overview* (ael)**

This section describes the following Primitive Polygon functions:
db_create_primitive_polygon() (ael)
db_create_shape_from_primitive_polygon() (ael)
db_get_shape_primitive_polygon() (ael)
db_get_shape_control_polygon() (ael)
db_get_primitive_polygon_outer() (ael)
db_get_primitive_polygon_num_points() (ael)
db_get_primitive_polygon_point() (ael)
db_get_primitive_polygon_num_holes() (ael)
db_get_primitive_polygon_hole() (ael)
db_add_hole_to_primitive_polygon() (ael)
db_get_primitive_polygon_with_linked_holes() (ael)
db_primitive_polygon_has_arcs() (ael)
db_get_primitive_polygon_with_arcs_removed() (ael)
db_get_primitive_polygon_edge_is_arc() (ael)
db_get_edge_arc_angle() (ael)
db_get_edge_arc_bulge() (ael)
db_get_edge_arc_center() (ael)

db_add_hole_to_primitive_polygon()

Returns a new *primitive polygon* (ael) of a given *primitive polygon* (ael) with a given hole polygon added into it.

Note
Only the outer boundary of the given hole polygon will be used. If the hole polygon has holes, those holes will be ignored.

Syntax

```
decl newPolygon = db_add_hole_to_primitive_polygon( polygon , holePolygon);
```

Where,

- *polygon* is a primitive polygon to generate a new *primitive polygon* (ael) with an added hole inside of it.
- *holePolygon* is a *primitive polygon* (ael) that will be added as a hole inside of another polygon.

Example

```
decl isClosed;
decl polygon = db_get_shape_primitive_polygon( dbShape1, &sClosed );
decl holePolygon = db_get_shape_primitive_polygon( dbShape2, &sClosed );
decl polygonWithHole = db_add_hole_to_primitive_polygon(polygon, holePolygon);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_get_shape_primitive_polygon() (ael)

Where Used (ael)

Schematic, Layout

db_create_primitive_polygon()

Returns a new *primitive polygon* (ael) from a given list of outer boundary points, and a given optional list of point index to edge arc bulge factors.
If no bulge factors are provided, then all points on the boundaries will be non-arc segments and the polygon will have no arcs.

Note
The list of outer boundary coordinate points should be in DB units.
The optional given list of arc bulge factors should be in double DB units.
A negative arc bulge factor indicates a clockwise arc segment, and a positive ac bulge factor indicates a counter clockwise arc segment.
An arc bulge factor of zero will not be set on a point, the point will be a non-arc segment point.

Syntax

```
decl newPrimitivePolygon = db_create_primitve_polygon(listOfPoints[,
listOfBulges]);
```

Where,

- *listofPoints* is an AEL list of x,y coordinate pairs, such as:
  list(x1,y1,x2,y2,…,xN,yN)
  or is a list of lists of x,y coordinate pairs, such as:
  list(list(x1,y1),list(x2,y2),…,list(xN,yN))
- *listofBulges* is an optional AEL list of point index, double value arc bulge factors, such as:
  list(bulge1, bulge2, bulge3,…, bulgeN)
  or is a list of lists of point index, bulge factor pairs, such as:
  list(list(pointIndex1,bulge1),list(pointIndex2,bulge2),…,list(pointIndexN,bulgeN))

Example

```
// Create a new shape from a new primitive polygon
decl newPolygon = db_create_primitive_polygon(list(0,0,10,20,10,40,0,40,0,0));
decl newShape = db_create_shape_from_primitive_polygon(context, newPolygon, layerId);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_get_shape_primitive_polygon() (ael)

db_get_shape_control_polygon() (ael)
db_create_shape_from_primitive_polygon() (ael)

Where Used (ael)

Schematic, Layout

db_create_shape_from_primitive_polygon()

Creates and returns a new database shape in a given context on a given layerId from a given source *primitive polygon* (ael).
The returned new dbShape object has been added into the context.

Syntax

```
decl newShape = db_create_shape_from_primitive_polygon(context, polygon,
layerId);
```

Where,

- *context* is a design context to create the new shape object inside of.
- *polygon* is a *primitive polygon* (ael) to generate a new database shape from.
- *layerId* is the layerId to add the new shape object within for the given design context.

Example

```
// Create a new shape from a new primitive polygon
decl newPolygon = db_create_primitive_polygon(list(0,0,10,20,10,40,0,40,0,0));
decl newShape = db_create_shape_from_primitive_polygon(context, newPolygon, layerId);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_create_primitive_polygon() (ael)

Where Used (ael)

Schematic, Layout

db_get_edge_arc_angle()

Returns the arc sweep angle for a given *primitive polygon* (ael)'s given point index of an edge that is an arc.
The returned arc sweep angle is in 0.001 degree units.
The edge must be an arc, see *db_get_primitive_polygon_edge_is_arc()* (ael).

The specified point index must be in the value range from zero to number of points in the polygon minus one.
i.e. idx >= 0 and idx < db_get_primitive_polygon_num_points(polygon)

Returns an AEL Error if the given integer index "idx" is less than zero or greater than or equal to the number of points in the given *primitive polygon* (ael).
Returns an AEL Error if the given integer edge index "idx" is not a valid index to an edge that is an arc.

Note
For polygons with holes, this function only checks edges from the outer boundary.

Syntax

```
decl angle = db_get_edge_arc_angle(polygon, idx);
```

Where,

- *polygon* is a *primitive polygon* (ael) to get the edge, that is following the given point index, arc segment angle.
- *idx* is a positive integer point index that represents the polygon edge that is an arc segment within the polygon.
  The point index must be between zero and (db_get_primitive_polygon_num_points(polygon) – 1).

Example

```
decl polygon = db_get_shape_primitive_polygon(dbShape);
decl numPoints = db_get_primitive_polygon_num_points(polygon);
decl i;
for (i = 0; i < numPoints; ++i)
{
   if (db_get_primitive_polygon_edge_is_arc(polygon, i))
   {
       decl arcAngle = db_get_edge_arc_angle(polygon, i);
       ...
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_primitive_polygon_has_arcs() (ael)
db_get_primitive_polygon_edge_is_arc() (ael)
db_get_edge_arc_center() (ael)
db_get_shape_primitive_polygon() (ael)

Where Used (ael)

Schematic, Layout

db_get_edge_arc_bulge()

Returns the arc bulge factor for a given *primitive polygon* (ael) given point index of an

edge.
Returns the arc bulge factor in real double DB units.

The specified point index must be in the value range from zero to number of points in the polygon minus one.
i.e. idx >= 0 and idx < db_get_primitive_polygon_num_points(polygon)

Returns an AEL Error if the given integer index "idx" is less than zero or greater than or equal to the number of points in the given *primitive polygon* (ael).
Returns an AEL Error if the given integer edge index "idx" is not a valid index to an edge.

Note
A bulge factor of zero indicates the point is a linear, non-arc segment point.
A negative bulge factor value indicates a clockwise arc segment and a positive bulge factor value indicates a counter clockwise arc segment.

For polygons with holes, this function only checks edges from the outer boundary.

Syntax

```
decl bulgeFactor = db_get_edge_arc_bulge(polygon, idx);
```

Where,

- *polygon* is a *primitive polygon* (ael) to get the edge, that is following the given point index, arc bulge factor.
- *idx* is a positive integer point index that represents the polygon edge that is within the polygon.
  The point index must be between zero and (db_get_primitive_polygon_num_points(polygon) – 1).

Example

```
decl polygon = db_get_shape_primitive_polygon(dbShape);
decl numPoints = db_get_primitive_polygon_num_points(polygon);
decl i;
for (i = 0; i < numPoints; ++i)
{
   decl bulgeFactor = db_get_edge_arc_bulge(polygon, i);
   ...
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_primitive_polygon_has_arcs() (ael)
db_get_primitive_polygon_edge_is_arc() (ael)
db_get_edge_arc_center() (ael)
db_get_edge_arc_angle() (ael)
db_get_shape_primitive_polygon() (ael)

Where Used (ael)

Schematic, Layout

db_get_edge_arc_center()

Returns the center coordinate point for a given *primitive polygon* (ael)'s given point index of an edge that is an arc.
The returned center point coordinate is in DB units.
The edge must be an arc, see *db_get_primitive_polygon_edge_is_arc()* (ael).

The specified point index must be in the value range from zero to number of points in the polygon minus one.
i.e. idx >= 0 and idx < db_get_primitive_polygon_num_points(polygon)

Returns an AEL Error if the given integer index "idx" is less than zero or greater than or equal to the number of points in the given *primitive polygon* (ael).
Returns an AEL Error if the given integer edge index "idx" is not a valid index to an edge that is an arc.

Note
For polygons with holes, this function only checks edges from the outer boundary.

Use db_get_x(point) and db_get_y(point) to get the x and y values in db units.

Syntax

```
decl point = db_get_edge_arc_center(polygon, idx);
```

Where,

- *polygon* is a *primitive polygon* (ael) to get the edge, that is following the given point index, arc segment center point.
- *idx* is a positive integer point index that represents the polygon edge that is an arc segment within the polygon.
  The point index must be between zero and (db_get_primitive_polygon_num_points(polygon) – 1).

Example

```
decl polygon = db_get_shape_primitive_polygon(dbShape);
decl numPoints = db_get_primitive_polygon_num_points(polygon);
decl i;
for (i = 0; i < numPoints; ++i)
{
   if (db_get_primitive_polygon_edge_is_arc(polygon, i))
   {
       decl centerPoint = db_get_edge_arc_center(polygon, i);
       decl centerX = db_get_x (centerPoint);
       decl centerY = db_get_y (centerPoint);
       ...
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_primitive_polygon_has_arcs() (ael)
db_get_primitive_polygon_edge_is_arc() (ael)
db_get_edge_arc_angle() (ael)
db_get_shape_primitive_polygon() (ael)

***Where Used* (ael)**

Schematic, Layout

db_get_primitive_polygon_edge_is_arc()

Returns TRUE if the edge following the given point index in a given *primitive polygon* (ael) is an arc segment; returns FALSE otherwise.
The specified point index must be in the value range from zero to number of points in the polygon minus one.
i.e. idx >= 0 and idx < db_get_primitive_polygon_num_points(polygon)
Returns an AEL Error if the given integer index "idx" is less than zero or greater than or equal to the number of points in the given *primitive polygon* (ael).

Note
For polygons with holes, this function only checks edges from the outer boundary.
For non-closed shapes, checking the last edge is not meaningful and will always return FALSE.

Syntax

```
decl isArc = db_get_primitive_polygon_edge_is_arc(polygon, idx);
```

Where,

- *polygon* is a *primitive polygon* (ael) to get the edge following the given point index to check is an edge that is an arc segment.
- *idx* is a positive integer point index that represents the polygon edge to test is an arc segment within the polygon.
  The point index must be between zero and (db_get_primitive_polygon_num_points(polygon) – 1).

Example

```
decl polygon = db_get_shape_primitive_polygon(dbShape);
decl numPoints = db_get_primitive_polygon_num_points(polygon);
decl i;
for (i = 0; i < numPoints; ++i)
{
   if (db_get_primitive_polygon_edge_is_arc(polygon, i))
   {
       decl centerPoint = db_get_edge_arc_center(polygon, i);
       decl centerX = db_get_x (centerPoint);
       decl centerY = db_get_y (centerPoint);
       ...
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_primitive_polygon_has_arcs() (ael)
db_get_shape_primitive_polygon() (ael)

Where Used (ael)

Schematic, Layout

db_get_primitive_polygon_hole()

Returns a new *primitive polygon* (ael) equivalent to the specified hole at a specified hole index in the given *primitive polygon* (ael).
The specified hole index must be in the value range from zero to number of holes in the polygon minus one.
i.e. idx >= 0 and idx < db_get_primitive_polygon_num_holes(polygon)

Syntax

```
decl primitivePolygonHole = db_get_primitive_polygon_hole(polygon, idx);
```

Where,

- *polygon* is a *primitive polygon* (ael) to get one of its given hole's polygon from.
- *idx* is a positive integer point index that represents the hole polygon to retrieve from the polygon.
  The point index must be between zero and (db_get_primitive_polygon_num_holes(polygon) – 1).

Example

```
decl polygon = db_get_shape_primitive_polygon(dbShape);
decl numHoles = db_get_primitive_polygon_num_holes(polygon);
decl i;
for (i = 0; i < numHoles; ++i)
{
   decl primitivePolygonHole = db_get_primitive_polygon_hole(polygon, i);
   ...
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_get_shape_primitive_polygon() (ael)

db_get_primitive_polygon_num_holes() (ael)

***Where Used* (ael)**

Schematic, Layout

db_get_primitive_polygon_num_holes()

Returns the integer number of holes in the given *primitive polygon* (ael).
For most shapes, this will return zero.
Polygons with holes are most commonly generated from Boolean operations.

Note
If a primitive polygon has its holes linked, this function will return zero.

Syntax

```
decl numHoles = db_get_primitive_polygon_num_holes(polygon);
```

Where,

- *polygon* is a *primitive polygon* (ael) to retrieve the integer number of holes it contains.

Example

```
decl polygon = db_get_shape_primitive_polygon(dbShape);
decl numHoles = db_get_primitive_polygon_num_holes(polygon);
decl i;
for (i = 0; i < numHoles; ++i)
{
   decl primitivePolygonHole = db_get_primitive_polygon_hole(polygon, i);
   ...
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_get_shape_primitive_polygon() (ael)
db_get_primitive_polygon_hole() (ael)

***Where Used* (ael)**

Schematic, Layout

db_get_primitive_polygon_num_points()

Returns the number of points in the outer boundary of the *primitive polygon* (ael).
For most shapes, this will return zero.

Note
For a given polygon with holes, this function will return the number of points in the outer boundary. For a given polygon with its holes linked, this function will return zero.

Syntax

```
decl numPoints = db_get_primitive_polygon_num_points(polygon);
```

Where,

- *polygon* is a *primitive polygon* (ael) to get the number of outer boundary points from.

Example

```
decl polygon = db_get_shape_primitive_polygon(dbShape);
decl numPoints = db_get_primitive_polygon_num_points(polygon);
decl i;
for (i = 0; i < numPoints; ++i)
{
   decl coord = db_get_primitive_polygon_point(polygon, i);
   decl x = db_get_x(coord);
   decl y = db_get_y(coord);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_get_primitive_polygon_point() (ael)
db_get_shape_primitive_polygon() (ael)

Where Used (ael)

Schematic, Layout

db_get_primitive_polygon_outer()

Returns an outer *primitive polygon* (ael) of a given *primitive polygon* (ael).
The outer *primitive polygon* (ael) returned will have holes removed.

Note
If the given polygon has had its holes linked, this function will not unlink them and the entire polygon will be returned.

Syntax

```
decl outerPolygon = db_get_primitive_polygon_outer(polygon);
```

Where,

- *polygon* is a *primitive polygon* (ael) to get an outer polygon from.

Example

```
decl polygon = db_get_shape_primitive_polygon(dbShape);
decl outerPolygon = db_get_primitive_polygon_outer(polygon);
decl numPoints = db_get_primitive_polygon_num_points(outerPolygon);
decl i;
for (i = 0; i < numPoints; ++i)
{
   decl coord = db_get_primitive_polygon_point(outerPolygon, i);
   decl x = db_get_x(coord);
   decl y = db_get_y(coord);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_get_shape_primitive_polygon() (ael)

Where Used (ael)

Schematic, Layout

db_get_primitive_polygon_point()

Returns the coordinate point at a specified point index in the given *primitive polygon* (ael).

The specified point index must be in the value range from zero to number of points in the polygon minus one.
i.e. idx >= 0 and idx < db_get_primitive_polygon_num_points(polygon)

Note
For polygon with holes, this function only gets points from the outer boundary.
For closed shapes, there is an implicit connection between the last and first points.

For non-closed shapes, there is no connection between the last and first points.

Use db_get_x(coord) and db_get_y(coord) to get the point's x and y values in DB units.

Syntax

```
decl coord = db_get_primitive_polygon_point(polygon, idx);
```

Where,

- *polygon* is a *primitive polygon* (ael) to get the point coordinate from.
- *idx* is a positive integer point index that represents the point coordinate to retrieve from the polygon.
  The point index must be between zero and (db_get_primitive_polygon_num_points(polygon) – 1).

Example

```
decl polygon = db_get_shape_primitive_polygon(dbShape);
decl numPoints = db_get_primitive_polygon_num_points(polygon);
decl i;
for (i = 0; i < numPoints; ++i)
{
   decl coord = db_get_primitive_polygon_point(polygon, i);
   decl x = db_get_x(coord);
   decl y = db_get_y(coord);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_get_primitive_polygon_num_points() (ael)
db_get_shape_primitive_polygon() (ael)

Where Used (ael)

Schematic, Layout

db_get_primitive_polygon_with_arcs_removed()

Returns a new *primitive polygon* (ael) with arcs removed from a given *primitive polygon* (ael) and given arc resolution to use for arc removal.

Note
The arc resolution is in 0.001 degree units.

Syntax

```
decl newPolygon = db_get_primitive_polygon_with_arcs_removed(polygon[,
arcResolution|dbShape|context ]);
```

Where,

- *polygon* is a *primitive polygon* (ael) to generate a *primitive polygon* (ael) with arcs removed from.
- *arcResolution* is an optional arc resolution in 0.001 degree units to use for arc removal.
  The *arcResolution* optional argument can be provided as:
  - an integer arc resolution in 0.001 degree units
  - a real arc resolution in 0.001 degree units
  - a dbShape where the shape's specific arc resolution if set will be used, if the shape has no specific arc resolution set, then the default value of 5000 will be used.
  - a context where the design context's arc resolution setting will be used.
    If *arcResolution* is not specified then the default value of 5000 will be used.

Example

```
decl isClosed;
decl polygon = db_get_shape_primitive_polygon(dbShape, &sClosed);
if (db_primitive_polygon_has_arcs(polygon))
{
   decl polygonWithNoArcs = db_get_primitive_polygon_with_arcs_removed(polygon, dbShape);
   ...
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_primitive_polygon_has_arcs() (ael)
db_get_shape_primitive_polygon() (ael)

***Where Used* (ael)**

Schematic, Layout

db_get_primitive_polygon_with_linked_holes()

Returns a new *primitive polygon* (ael) with holes linked from a given *primitive polygon* (ael).
Converts the given *primitive polygon* (ael) into a simple polygon, i.e. a simple point list. Cutlines will link holes to outer boundaries and will also link holes to other holes.

Note
Calling this with anything other than a simple polygon is undefined, i.e. garbage in/garbage out.
Uses the current design context's arc resolution when context isn't specified, or uses 5000 if there is no current design context.

Syntax

```
decl newPolygon = db_get_primitive_polygon_with_linked_holes(polygon[,
arcResolution|dbShape|context ]);
```

Where,

- *polygon* is a *primitive polygon* (ael) to generate a *primitive polygon* (ael) with linked holes from.
- *arcResolution* is an optional arc resolution in 0.001 degree units to use for linking of the polygon's holes.
  The *arcResolution* optional argument can be provided as:
  - an integer arc resolution in 0.001 degree units
  - a real arc resolution in 0.001 degree units
  - a dbShape where the shape's specific arc resolution if set will be used, if the shape has no specific arc resolution set, then the default value of 5000 will be used.
  - a context where the design context's arc resolution setting will be used.
    If *arcResolution* is not specified then the default value of 5000 will be used.

Example

```
decl isClosed;
decl polygon = db_get_shape_primitive_polygon(dbShape, &sClosed);
decl polygonWithHolesLinked = db_get_primitive_polygon_with_linked_holes(polygon);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_get_shape_primitive_polygon() (ael)

Where Used (ael)

Schematic, Layout

db_get_shape_control_polygon()

Returns a *primitive polygon* (ael) containing the shape's control points. This polygon is guaranteed to not contain arcs or holes.
This function only works for wire, path and trace shapes. For other shapes, this function currently returns NULL without error.
The returned polygon's points are the vertices along the centerline.

Note
These vertices do not follow the curves if curved corners are turned on, they are control points.

Syntax

```
decl polygon = db_get_shape_control_polygon(dbShape);
```

Where,

- *dbShape* is a shape to get a control *primitive polygon* (ael) for.

Example

```
decl polygon = db_get_shape_control_polygon(dbShape);
if (polygon == NULL)
   return; //Not a wire, path, or trace, so return
decl numPoints = db_get_primitive_polygon_num_points(polygon);
decl i;
for (i = 0; i < numPoints; ++i)
{
    decl coord = db_get_primitive_polygon_point(polygon, i);
    decl x = db_get_x(coord);
    decl y = db_get_y(coord);
    …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_get_shape_primitive_polygon() (ael)

Where Used (ael)

Schematic, Layout

db_get_shape_primitive_polygon()

Returns a *primitive polygon* (ael) for a given shape. Works for all shapes.

For filled shapes, if the optional return argument isClosed is specified it will be returned as TRUE.
For non-filled shapes, the polygon returned represents the outline of the shape and if the optional return argument isClosed is specified, it will be returned as FALSE.

If the optional argument removeArcs is specified as TRUE, the returned *primitive polygon* (ael) will have all arcs removed.
If the optional argument removeArcs is specified as FALSE, then the returned *primitive polygon* (ael)'s arcs will not be removed.
If the optional argument removeArcs is specified, the parent design context of the given shape also needs to be provided. The arc resolution used for removing arcs will be determined from the shape object if the given shape has a specific arc resolution set. If there is no specific arc resolution set for the given shape, the parent design context of the shape will be used to retrieve the default arc resolution to use for arc removal.

Syntax

```
decl polygon = db_get_shape_primitive_polygon(dbShape[, &isClosed[, removeArcs,
context]]);
```

Where,

- *dbShape* is a shape to get a *primitive polygon* (ael) for.
- optional: *&isClosed* – return argument pointer that will be set to TRUE if the returned polygon is from a filled shape, will be set FALSE if the returned polygon is for a non-filled shape.
- optional: *removeArcs* – specified as TRUE or FALSE will denote if the return primitive polygon should have arcs removed. By default this optional argument is FALSE if not specified.
    - *context* – The parent design context of the given dbShape that will be used to determine arc resolution for removing arcs if the dbShape object does not have a specific arc resolution set for it.

Example

```
decl isClosed;
decl  polygon = db_get_shape_primitive_polygon(dbShape, &sClosed);
decl numPoints = db_get_primitive_polygon_num_points(polygon);
decl i;
```

```
for (i = 0; i < numPoints; ++i)
{
    decl coord = db_get_primitive_polygon_point(polygon, i);
    decl x = db_get_x(coord);
    decl y = db_get_y(coord);
    …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_get_shape_control_polygon() (ael)

Where Used (ael)

Schematic, Layout

db_primitive_polygon_has_arcs()

Returns TRUE if a given *primitive polygon* (ael) contains arcs; returns FALSE otherwise.

Syntax

```
decl bPolyHasArcs = db_primitive_polygon_has_arcs(polygon);
```

Where,

- *polygon* is a *primitive polygon* (ael) to test if it contains arcs.

Example

```
decl polygon = db_get_shape_primitive_polygon(dbShape);
decl bPolyHasArcs = db_primitive_polygon_has_arcs(polygon);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Primitive Polygon Overview (ael)
db_get_primitive_polygon_with_arcs_removed() (ael)
db_get_shape_primitive_polygon() (ael)

Where Used (ael)

Schematic, Layout

Primitive Polygon Overview

A primitive polygon AEL object represents a geometric shape. That shape has an outer boundary (the polygon's outline) and zero or more inner boundaries (holes in the polygon). These boundaries contain points which are connected by lines or arcs.

The purpose of a primitive polygon is to represent any arbitrary geometric shape.

Primitive Polygon applications:

A primitive polygon is useful to extract information about a particular shape object.
A primitive polygon can be created from a given shape object with the function *db_get_shape_primitive_polygon()* (ael) or *db_get_shape_control_polygon()* (ael).
A new database shape object can also be generated from a given given primitive polygon with the function *db_create_shape_from_primitive_polygon()* (ael).

A new primitive polygon can also be created from scratch by specifying the polygon's outer boundary’s point list and also a list of bulge factors for arc segments if desired with the function *db_create_primitive_polygon()* (ael) .

A primitive polygon is useful to retrieve different types of information about a geometric shape.
Such as:

- if a given shape is closed or not.
- if a given shape has outer boundary points.
  - a given shape's outer boundary coordinate point by point index.
- if a given shape's edge is an arc.
  - the edge's arc's bulge factor.
  - the edge's arc's center coordinate point.
  - the edge's arc's angle.
- if a given shape has holes.
  - the primitive polygon of a shape's hole by hole index

A primitive polygon can be used to retrieve different types of polygons for a given database shape,
Such as:

- a primitive control polygon of path, trace, wire shapes.
- a primitive polygon of a given shape with arcs removed.
- a primitive polygon of a given shape with holes removed.
- a primitive polygon of a given shape with holes linked.

A new database shape object can be created from a given primitive polygon.

- a new primitive polygon can be created from a specified point list of boundary points and bulge factors.
- a primitive polygon can have holes added to it.
- a new shape object can be generated from any given primitive polygon.

Example using Primitive Polygons

```
decl context=de_get_current_design_context();
```

```
decl it=db_create_shape_iter(context);
it = db_shape_iter_limit_selected(it);
decl isClosed;
decl polygon = db_get_shape_primitive_polygon(it,&sClosed);
de_info(isClosed);
de_info(polygon);
de_info(db_primitive_polygon_has_arcs(polygon));
de_info(db_get_primitive_polygon_num_holes(polygon));
decl numPoints = db_get_primitive_polygon_num_points(polygon);
de_info(numPoints);
decl i;
for (i = 0; i < numPoints; ++i)
{
   decl coord = db_get_primitive_polygon_point(polygon, i);
   de_info(strcat(db_get_x(coord),",",db_get_y(coord)));
   if (db_get_primitive_polygon_edge_is_arc(polygon, i))
   {
       de_info("arc");
       decl center = db_get_edge_arc_center(polygon, i);
       de_info(strcat("arcCenter: ", db_get_x(center),",",db_get_y(center)));
       decl angle = float(db_get_edge_arc_angle(polygon, i)) / 1000.0;
       de_info(strcat("arcAngle: ", angle) );
   }
}
polygon = db_get_shape_control_polygon(it);
if (polygon != NULL)
{
   numPoints = db_get_primitive_polygon_num_points(polygon);
   de_info(numPoints);
   for (i = 0; i < numPoints; ++i)
   {
       decl coord = db_get_primitive_polygon_point(polygon, i);
       de_info(strcat(db_get_x(coord),",",db_get_y(coord)));
   }
}
```

Property Functions

- ***Property Overview*** **(ael)**
- ***Property Iterator - Overview*** **(ael)**
- ***Property Iterator Functions*** **(ael)**

This section describes the following Property functions:

db_add_property() (ael)
db_remove_property() (ael)
db_remove_properties_with_prefix() (ael)
db_get_property() (ael)
db_get_property_as_string() (ael)

db_add_property()

Function to add a *property* (ael) to an ADS database object. ADS Database object types that properties can be owned by are : DesignContext, AppObject, Instance, Shape, Net, or Pin. You can replace an existing *property* (ael) value by passing TRUE for *doReplace* (the 4th argument).

Syntax

```
db_add_property( dbObject, propName, propValue, doReplace );
```

Where,

- *dbObject* is a DesignContext, Shape, Instance, AppObject, Pin, or Net.
- *propName* is a property name.
- *propValue* is a long, double, or string value.
- *doReplace* is a Boolean TRUE or FALSE, where if TRUE is specified then the *property* (ael) value will be replaced if it already exists.

Notes

- Adding properties to an object during property iteration of that same object will cause that property iterator to become corrupt, which can lead to a crash if that corrupt iterator is used.

Example

```
db_add_property(context, "MyPropertyDouble", 1.33, TRUE);
db_add_property(pinH, "MyPinPropertyLong", 25000, TRUE);
db_add_property(instH, "MyInstPropertyString", "MySpecialValue", TRUE);
db_add_property(shapeH, "MyShapeProperty", -33.231, TRUE);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Property Overview (ael)
db_get_property() (ael)
db_get_property_as_string() (ael)

db_remove_properties_with_prefix() (ael)
db_remove_property() (ael)

Where Used (ael)

Schematic, Layout

db_get_property_as_string()

This function is used to get the *property's* (ael) double, string, or long value returned as a string value from an ADS database object.
ADS Database object types that *properties* (ael) can be owned by are : DesignContext, AppObject, Instance, Shape, Net, or Pin.
The function will return NULL if the given database object has no *property* (ael) with the given name.

Syntax

```
decl strValue = db_get_property_as_string( dbObject, propName );
```

Where,

- *dbObject* is a DesignContext, Shape, Instance, AppObject, Pin, or Net.
- *propName* is a property name.

Example

```
decl propStrValue = db_get_property_as_string(instIter, "MyProperty");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Property Overview (ael)
db_add_property() (ael)
db_get_property() (ael)
db_remove_properties_with_prefix() (ael)
db_remove_property() (ael)

Where Used (ael)

Schematic, Layout

db_get_property()

Function to get the *property's* (ael) double, string, or long value from an ADS database object.
ADS Database object types that properties can be owned by are : DesignContext, AppObject, Instance, Shape, Net, or Pin.
The function will return NULL if the given database object has no *property* (ael) with the given name.

Syntax

```
decl value = db_get_property( dbObject, propName );
```

Where

- *dbObject* is a DesignContext, Shape, Instance, AppObject, Pin, or Net.
- *propName* is a property name.

Example

```
decl propValue = db_get_property(instIter, "MyProperty");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Property Overview (ael)
db_add_property() (ael)
db_get_property_as_string() (ael)
db_remove_properties_with_prefix() (ael)
db_remove_property() (ael)

Where Used (ael)

Schematic, Layout

db_remove_properties_with_prefix()

This function removes all *properties* (ael) from an ADS database object which begin with the given prefix.
ADS Database object types that *properties* (ael) can be owned by are : DesignContext, AppObject, Instance, Shape, Net, or Pin.

Notes

- Removing properties from an object during property iteration of that same object will cause that property iterator to become corrupt, which can lead to a crash if that corrupt iterator is used.

Syntax

```
db_remove_properties_with_prefix( dbObject, prefixStr );
```

Where,

- *dbObject* is a DesignContext, Shape, Instance, AppObject, Pin, or Net.
- *prefixStr* is a property name prefix.

Example

```
if (bRemoveEachIndividually == TRUE)
{
```

```
   db_remove_property(context, "MyProperty1");
   db_remove_property(context, "MyProperty2");
   db_remove_property(context, "MyProperty3");
   db_remove_property(context, "MyProperty4");
}
else
   db_remove_properties_with_prefix(context, "My");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Property Overview (ael)
db_add_property() (ael)
db_get_property() (ael)
db_get_property_as_string() (ael)
db_remove_property() (ael)

***Where Used* (ael)**

Schematic, Layout

db_remove_property()

Function to remove a *property* (ael) from an ADS database object.
The *properties* (ael) of ADS Database object types are : DesignContext, AppObject, Instance, Shape, Net, or Pin.

Note

- Removing properties from an object during property iteration of that same object will cause that property iterator to become corrupt, which can lead to a crash if that corrupt iterator is used.

Syntax

```
db_remove_property( dbObject, propName );
```

Where,

- *dbObject* is a DesignContext, Shape, Instance, AppObject, Pin, or Net.
- *propName* is a property name.

Example

```
db_remove_property(context, "MyPropertyDouble");
db_remove_property(pinH, "MyPinPropertyLong");
db_remove_property(instH, "MyInstPropertyString");
db_remove_property(shapeH, "MyShapeProperty");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Property Overview (ael)
db_add_property() (ael)
db_get_property() (ael)
db_get_property_as_string() (ael)
db_remove_properties_with_prefix() (ael)

Where Used (ael)

Schematic, Layout

Property Overview

A property consists of a user-defined name and a user-defined string, double or long value. In ADS a property can be added to any of the standard ADS database types: instances, shapes, application objects, pins, and nets. Properties can also be added to a design context. This will add a property to the design.

A *Property Iterator* (ael) can be used to traverse a database object's properties. The types of information that a Property AEL object stores includes:

- **Name**: *name* is a string; the unique name of the property for the object it is owned by.
- **Type**: Property *type* is determined by the value for the property. The possible types are long, double and string value types.
- **Value**: Property *value* holds the properties value; the types of values a property can hold are string, long, and double value types.

Creating Properties

You can add a property to an ADS database object in AEL, with the *db_add_property()* (ael) AEL function.

Syntax

```
db_add_property (dbObject, propName, propValue, replaceExisting);
```

Example of adding a property to different database objects

```
db_add_property(context, "MyPropertyDouble", 1.33, TRUE);
db_add_property(pinH, "MyPinPropertyLong", 25000, TRUE);
db_add_property(instH, "MyInstPropertyString", "MySpecialValue", TRUE);
db_add_property(shapeH, "MyShapeProperty", -33.231, TRUE);
```

Removing Properties

You can remove a property from an ADS database object in AEL, with the *db_remove_property()* (ael) AEL function.

Syntax

```
db_remove_property (dbObject, propName);
```

Example of removing a property to different database objects

```
db_remove_property(context, "MyPropertyDouble");
db_remove_property(pinH, "MyPinPropertyLong");
db_remove_property(instH, "MyInstPropertyString");
db_remove_property(shapeH, "MyShapeProperty");
```

Accessing Properties

You can access or get the value from a property owned by a database object with the AEL functions *db_get_property()* (ael) and *db_get_property_as_string()* (ael).

Syntax

```
value = db_get_property (dbObject, propName);
```

Where,

- *value* returned will be a string if the property is found. If property can't be found, then the value returned is NULL.

Example of Querying Property Value Information

```
decl context = de_get_current_design_context();
decl instIter = db_create_inst_iter(context);
decl propValue = NULL;
for ( ; db_inst_iter_is_valid(instIter);
      instIter = db_inst_iter_get_next(instIter) )
{
 // Get the 1st instance with my property and get its property value.
 propValue = db_get_property(instIter, "MyProperty");
 if (propValue != NULL)
 {
    break; // Found the 1st instance with my property value.
 }
}
if (propValue == NULL)
{
 // No instance had my property
 ...
}
else
{
 // Instance found with my property value, value is stored in propValue variable.
 ...
}
```

List of Property Functions

See the following link: *Property Functions* (ael)

Property Iterator Functions

- ***Property Iterator - Overview*** **(ael)**
- ***Property Overview*** **(ael)**
- ***Property Functions*** **(ael)**

This section describes the following Property Iterator functions:

db_create_prop_iter() (ael)
db_prop_iter_is_valid() (ael)
db_prop_iter_get_next() (ael)
db_prop_iter_get_name() (ael)
db_prop_iter_get_type() (ael)
db_prop_iter_get_value() (ael)
db_prop_iter_remove_prop() (ael)

db_create_prop_iter()

This function returns an iterator to the first *property* (ael) of the given database object. *Properties* (ael) can exist on DesignContexts, AppObjects, instances, shapes, nets, and pins.
If no *property* (ael) is available, then the iterator returned will be invalid.
The function *db_prop_iter_is_valid()* (ael) can be used to test if a *property iterator* (ael) is valid.

Syntax

```
decl iter = db_create_prop_iter( dbObject );
```

Where,

- *dbObject* is a DesignContext, AppObject, Shape, Instance, Pin, or Net.

Example

```
decl designContext = de_get_current_design_context();
decl instIter = db_create_inst_iter(designContext);
for ( ; db_inst_iter_is_valid(instIter);
    instIter =  db_inst_iter_get_next(instIter) )
{
   decl propIter = db_create_prop_iter(instIter);

   for ( ; db_prop_iter_is_valid(propIter);
       propIter = db_prop_iter_get_next(propIter) )
   {
       decl propType = db_prop_iter_get_type(propIter);
       decl propValue = db_prop_iter_get_value(propIter);
       …
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Property Overview (ael)
Property Iterator - Overview (ael)
db_prop_iter_is_valid() (ael)
db_prop_iter_get_next() (ael)
db_prop_iter_get_name() (ael)
db_prop_iter_get_type() (ael)
db_prop_iter_get_value() (ael)
db_prop_iter_remove_prop() (ael)

Where Used (ael)

Schematic, Layout

db_prop_iter_get_name()

Returns the name of the *property* (ael) referenced the given *property iterator* (ael).

Syntax

```
decl propName = db_prop_iter_get_name(propIter);
```

Where,

- *propIter* is a valid *property iterator* (ael) returned from a function such as *db_create_prop_iter()* (ael).

Example

```
decl designContext = de_get_current_design_context();
decl instIter = db_create_inst_iter(designContext);
for ( ; db_inst_iter_is_valid(instIter);
    instIter =  db_inst_iter_get_next(instIter) )
{
   decl propIter = db_create_prop_iter(instIter);

   for ( ; db_prop_iter_is_valid(propIter);
       propIter = db_prop_iter_get_next(propIter) )
   {
       decl propName = db_prop_iter_get_name(propIter);
       decl propType = db_prop_iter_get_type(propIter);
       decl propValue = db_prop_iter_get_value(propIter);
       …
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Property Overview (ael)
Property Iterator - Overview (ael)
db_create_prop_iter() (ael)

db_prop_iter_is_valid() (ael)
db_prop_iter_get_next() (ael)
db_prop_iter_get_type() (ael)
db_prop_iter_get_value() (ael)
db_prop_iter_remove_prop() (ael)

Where Used (ael)

Schematic, Layout

db_prop_iter_get_next()

Returns the next *property iterator* (ael) from the given *property iterator* (ael).

Syntax

```
decl nextPropIter = db_prop_iter_get_next(propIter);
```

Where,

- *propIter* is a valid *property iterator* (ael) returned from a function such as *db_create_prop_iter()* (ael).

Example

```
decl designContext = de_get_current_design_context();
decl instIter = db_create_inst_iter(designContext);
for ( ; db_inst_iter_is_valid(instIter);
    instIter =  db_inst_iter_get_next(instIter) )
{
   decl propIter = db_create_prop_iter(instIter);

   for ( ; db_prop_iter_is_valid(propIter);
       propIter = db_prop_iter_get_next(propIter) )
   {
       decl propType = db_prop_iter_get_type(propIter);
       decl propValue = db_prop_iter_get_value(propIter);
       …
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Property Overview (ael)
Property Iterator - Overview (ael)
db_create_prop_iter() (ael)
db_prop_iter_is_valid() (ael)
db_prop_iter_get_name() (ael)
db_prop_iter_get_type() (ael)
db_prop_iter_get_value() (ael)
db_prop_iter_remove_prop() (ael)

***Where Used* (ael)**

Schematic, Layout

db_prop_iter_get_type()

Returns the integer value type of the *property iterator's* (ael) current *property* (ael). The possible types of a *property* (ael) value are:

- PROPERTY_VALUE_LONG – describes a long *property* (ael) value
- PROPERTY_VALUE_STRING – describes a string *property* (ael) value
- PROPERTY_VALUE_DOUBLE – describes a double *property* (ael) value

Syntax

```
decl propType = db_prop_iter_get_type(propIter);
```

Where,

- *propIter* is a valid *property iterator* (ael) returned from a function such as *db_create_prop_iter()* (ael).

Example

```
decl designContext = de_get_current_design_context();
decl instIter = db_create_inst_iter(designContext);
for ( ; db_inst_iter_is_valid(instIter);
    instIter =  db_inst_iter_get_next(instIter) )
{
   decl propIter = db_create_prop_iter(instIter);

   for ( ; db_prop_iter_is_valid(propIter);
       propIter = db_prop_iter_get_next(propIter) )
   {
       decl propType = db_prop_iter_get_type(propIter);
       decl propValue = db_prop_iter_get_value(propIter);
       …
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Property Overview (ael)
Property Iterator - Overview (ael)
db_create_prop_iter() (ael)
db_prop_iter_is_valid() (ael)
db_prop_iter_get_next() (ael)
db_prop_iter_get_name() (ael)
db_prop_iter_get_value() (ael)
db_prop_iter_remove_prop() (ael)

***Where Used* (ael)**

Where Used (ael)

Schematic, Layout

db_prop_iter_get_value()

This function returns the string, long or real value, if the optional type argument is not specified.
If type argument is specified, it must be one of the following 3 types:

- PROPERTY_VALUE_LONG – describes a long *property* (ael) value
- PROPERTY_VALUE_STRING – describes a string *property* (ael) value
- PROPERTY_VALUE_DOUBLE – describes a double *property* (ael) value

If type argument is specified, then the function will return the *property* (ael) value as the type the argument specifies.

Syntax

```
decl propValue = db_prop_iter_get_value(propIter [, propType]);
```

Where,

- *propIter* is a valid *property iterator* (ael) returned from a function such as *db_create_prop_iter()* (ael).
- *propType* is an optional integer *property* (ael) type, which can be any one of the following 3 options: **PROPERTY_VALUE_LONG**, **PROPERTY_VALUE_STRING**, or **PROPERTY_VALUE_DOUBLE**

Example

```
decl designContext = de_get_current_design_context();
decl instIter = db_create_inst_iter(designContext);
for ( ; db_inst_iter_is_valid(instIter);
    instIter =  db_inst_iter_get_next(instIter) )
{
   decl propIter = db_create_prop_iter(instIter);

   for ( ; db_prop_iter_is_valid(propIter);
       propIter = db_prop_iter_get_next(propIter) )
   {
       decl propType = db_prop_iter_get_type(propIter);
       decl propValue = db_prop_iter_get_value(propIter);
       …
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Property Overview (ael)
Property Iterator - Overview (ael)

db_create_prop_iter() (ael)
db_prop_iter_is_valid() (ael)
db_prop_iter_get_next() (ael)
db_prop_iter_get_name() (ael)
db_prop_iter_get_type() (ael)
db_prop_iter_remove_prop() (ael)

Where Used (ael)

Schematic, Layout

db_prop_iter_is_valid()

Returns TRUE if the *property iterator* (ael) is referencing a valid *property* (ael).

Syntax

```
decl isValid = db_prop_iter_is_valid(propIter);
```

Where,

- *propIter* is a valid *property iterator* (ael) returned from a function such as *db_create_prop_iter()* (ael).

Example

```
decl designContext = de_get_current_design_context();
decl instIter = db_create_inst_iter(designContext);
for ( ; db_inst_iter_is_valid(instIter);
    instIter =  db_inst_iter_get_next(instIter) )
{
   decl propIter = db_create_prop_iter(instIter);

   for ( ; db_prop_iter_is_valid(propIter);
       propIter = db_prop_iter_get_next(propIter) )
   {
       decl propType = db_prop_iter_get_type(propIter);
       decl propValue = db_prop_iter_get_value(propIter);
       …
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Property Overview (ael)
Property Iterator - Overview (ael)
db_create_prop_iter() (ael)
db_prop_iter_get_next() (ael)
db_prop_iter_get_name() (ael)
db_prop_iter_get_type() (ael)
db_prop_iter_get_value() (ael)
db_prop_iter_remove_prop() (ael)

Where Used (ael)

Schematic, Layout

db_prop_iter_remove_prop()

Removes the given *property iterator's* (ael) current *property* (ael) and returns a *property iterator* (ael) to the next *property* (ael).

Note
The current iterator will become corrupt when its *property* (ael) is removed. A corrupt iterator can lead to a crash if used, so its recommended to set the given iterator to the next *property iterator* (ael) that is returned by this function.

Syntax

```
propIter = db_prop_iter_remove_prop(propIter);
```

Where,

- *propIter* is a valid *property iterator* (ael) returned from a function such as *db_create_prop_iter()* (ael).

Example

```
decl designContext = de_get_current_design_context();
decl instIter = db_create_inst_iter(designContext);
for ( ; db_inst_iter_is_valid(instIter);
    instIter =  db_inst_iter_get_next(instIter) )
{
   decl propIter = db_create_prop_iter(instIter);

   for ( ; db_prop_iter_is_valid(propIter);
       propIter = db_prop_iter_get_next(propIter) )
   {
       propIter= db_prop_iter_remove_prop(propIter);
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Property Overview (ael)
Property Iterator - Overview (ael)
db_create_prop_iter() (ael)
db_prop_iter_is_valid() (ael)
db_prop_iter_get_next() (ael)
db_prop_iter_get_name() (ael)
db_prop_iter_get_type() (ael)
db_prop_iter_get_value() (ael)

Where Used (ael)

Schematic, Layout

Property Iterator Overview

A property iterator can be used to iterate over a database object's *properties* (ael). A property iterator references a *property* (ael) on the owning database object. A *Property* (ael) consists of a user-defined name and a user-defined string, double or long value. In ADS a *property* (ael) can be added to any of the standard ADS database types: instances, shapes, application objects, pins, and nets. Properties can also be added to a design context, in which this will add a property to the design.

The types of information that a Property Iterator can be used to access includes:

- the list of properties a database object owns.
- the property's name.
- the property's type.
- the property's value.

For more information about ADS *properties* (ael), please see: *Property Overview* (ael).

Property information accessible by a Property Iterator

These are the different data fields stored within an AEL Property Iterator Object.

Database object's property list

- The property iterator can be used to access each of the *properties* (ael) on the database object and then access each of the *property's* (ael) datafields: property name and property value

Property Name

- name is a string; the unique name of the *property* (ael) for the object it is owned by.

Property Type

- the type is determined by the value for the *property* (ael). The possible types are long, double and string value types.

Property Value

- the value holds the *property's* (ael) value; the types of values a *property* (ael) can hold are string, long, and double value types.

Creating Property Iterators

You can create a property iterator to iterate over an ADS database object's *properties* (ael) in AEL, with the *db_create_prop_iter()* (ael) AEL function.

db_create_prop_iter()

Syntax

`decl propIter = db_create_prop_iter(dbObject);` - See documentation for *db_create_prop_iter()* (ael) for more details.

Where,

- *dbObject* is a design context, instance, shape, pin, or net database object.

Accessing Properties

You can access or get the value from a *property* (ael) owned by a database object via a Property Iterator with the AEL functions *db_prop_iter_get_name()* (ael), *db_prop_iter_get_type()* (ael), and *db_prop_iter_get_value()* (ael).

db_prop_iter_get_name()

Syntax

`decl name = db_prop_iter_get_name(propIter);` - See documentation for *db_prop_iter_get_name()* (ael) for more details.

Where,

- returned *name* will hold the property's string name.

db_prop_iter_get_type()

Syntax

`decl type = db_prop_iter_get_type(propIter);` - See documentation for db_prop_iter_gt_type() for more details.

Where,

- returned *type* can be returned as one of 3 types: **PROPERTY_VALUE_LONG**, **PROPERTY_VALUE_STRING**, or **PROPERTY_VALUE_DOUBLE**.

db_prop_iter_get_value()

Syntax

`decl value = db_prop_iter_get_value(propIter , type );` - See documentation for *db_prop_iter_get_value()* (ael) for more details.

Where,

- returned *value* can be a string, int or real, if the type argument is not specified.
  If type argument is specified, it must be one of the following 3 types: **PROPERTY_VALUE_LONG**, **PROPERTY_VALUE_STRING**, or **PROPERTY_VALUE_DOUBLE**.
  If type argument is specified, then it will return the property value as the type the argument specifies.

Querying Property Values with a property iterator

You can use a property iterator on any of the database objects: design contexts, instances, shapes, pins, and nets, to traverse and examine each of the *properties* (ael) they own.
See the list of *Property Iterator Functions* (ael) for more information.

Example of Querying Property Value Information with Property Iterators

```
decl context = de_get_current_design_context();
decl instIter = db_create_inst_iter(context);
for ( ; db_inst_iter_is_valid(instIter);
      instIter = db_inst_iter_get_next(instIter) )
{
 decl propIter = db_create_prop_iter(instIter);
 for ( ; db_prop_iter_is_valid(propIter);
       propIter = db_prop_iter_get_next(propIter) )
 {
   decl propType = db_prop_iter_get_type(propIter);
   decl propValue = db_prop_iter_get_value(propIter, proType);
   ...
 }
}
```

List of Property Iterator Functions

See the following link: *Property Iterator Functions* (ael)

Selection Functions

This section describes each selection function in detail. The functions are listed in alphabetical order.

db_get_shape_selected_points()

Returns a list of selected points' coordinates for a given shape. Will return an AEL list of coordinates. The list will be empty if there are no selected points on the given shape.

See also: *db_create_shape_iter()* (ael), *db_get_x()* (ael), *db_get_y()* (ael).

Syntax:

```
db_get_shape_selected_points(shape);
```

where

shape is a shape object, dg handle, or shape iterator.

Example:

```
decl context = de_get_current_design_context();
// For all selected rectangles in the current DesignContext get the
//  selected points coordinates and do something with the coordinates.
decl shapeIter = db_create_shape_iter(context);
shapeIter = db_shape_iter_limit_selected(shapeIter);
for( ; db_shape_iter_is_valid(shapeIter); shapeIter = db_shape_iter_get_next(shapeIter))
{
   // For all rectangles, process selected points
   if (db_shape_is_rectangle(shapeIter))
   {
       decl selCoordLst = db_get_shape_selected_points(shapeIter);

       // For any selected points on rectangles do something
       decl selIdx = 0;
       for (selIdx=0; selIdx < listlen(selCoordLst); selIdx++)
       {
            decl coordH = nth(selIdx, selCoordLst);
            decl x = db_get_x(coordH);
            decl y = db_get_y(coordH);
            ...
            // Do something here with each selected point.
            ...
       }
   }
}
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_select_all_on_layerid()

Function to select or deselect all shapes on a particular LayerId in the given design context.
Returns: None

Note
Protected layerIds are included in the selection. A LayerId can be tested if it is protected by the function *db_is_layerid_protected(context, layerId)* (ael).

Syntax

```
db_select_all_on_layerid(context, layerId [,bSelect]);
```

Where,

- *context* is the design context to select all shapes on a particular layerId within the design.
- *layerId* is the layerId in the design context to select all shapes on.
- *bSelect* is an optional argument, which is TRUE by default.
  If the argument is given and specified as TRUE it will select all shapes on the given layerId of the given design context.
  If the argument is specified FALSE, it will deselect all shapes on the given layerId of the given design context.

Example

```
decl context = de_get_design_context(winInst);
decl layerId1 = db_get_layerid("Metal1", "drawing");
decl layerId2 = db_get_layerid("Metal2");
db_select_all_on_layerid(context, layerId1);
db_select_all_on_layerid(context, layerId2, FALSE);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_select() (ael)
db_select_all() (ael)
db_deselect_all() (ael)

***Where Used* (ael)**

Schematic, Layout

db_cycle_select_item()

Select one item at a given location in the DesignContext. If multiple objects are at the same location, this function cycles through the available objects. After the function is called, there will be one item selected if any item is found at the location. Otherwise, no items will be selected. Returns the instance object, shape object, or port object of object item that was selected. Returns NULL if no object item was selected. User filters are honored.

See also: *db_pick_instance_at()* (ael), *db_pick_object_at()* (ael), *db_pick_shape_at()* (ael)
db_get_selection_tolerance() (ael).

Syntax:

```
db_cycle_select_item(context, x, y, tolerance);
```

where

context is a DesignContext returned from a function such as *de_get_current_design_context()*.
x , *y* is the x,y coordinate in database units.
tolerance is the user's selection tolerance.

Example:

```
decl winInst = api_get_current_window();
decl context = de_get_design_context(winInst);
decl tolerance = db_get_selection_tolerance(winInst);
decl x=10,y=20;
// Select object item at x, y location.
decl objhandle=db_cycle_select_item(context, x, y, tolerance);
if (!objhandle)
   return;  // No item selected.
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_deselect_all()

Deselect all items in a DesignContext. User filters are ignored.

See also: *db_is_selected()* (ael), *db_select()* (ael), *db_select_all()* (ael).

Syntax:

```
db_deselect_all(context);
```

where

context is a DesignContext returned from a function such as *de_get_current_design_context()*.

Example:

```
decl context = de_get_current_design_context();
// Deselect all objects for given DesignContext.
db_deselect_all(context);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_deselect_area()

Deselect all items in a given rectangular area of a DesignContext. User filters are ignored.

See also: *db_deselect_all()* (ael), *db_select()* (ael), *db_select_all()* (ael), *db_select_area()* (ael).

Syntax:

```
db_deselect_area(context, x1, y1, x2, y2);
```

where

context is a DesignContext returned from a function such as *de_get_current_design_context()*.
x1, *y1* is the x, y coordinate of the first corner of the rectangular area in database units.
x2, *y2* is the x,y coordinate of the second (opposite) corner of the rectangular area in database units.

Example:

```
decl winInst = api_get_current_window();
decl context = de_get_design_context(winInst);
// Rectangular area to deselect in database units.
decl x1=10,y1=20, x2=30, y2=40;
// Deselect all the items in the given rectangular area.
db_deselect_area(context, x1, y1, x2, y2);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_get_objects_selected_at()

Returns a list of shape objects, instance objects, and/or port objects that are selected at a particular location in a DesignContext. User filters are ignored.

See also: *db_get_selection_tolerance()* (ael).

Syntax:

```
db_get_objects_selected_at(context, x, y, tolerance, [, method]);
```

where

context is a DesignContext returned from a function such as *de_get_current_design_context()*.
x , *y* is the x,y coordinate in database units.
tolerance is the user's selection tolerance.
method is an optional selection method. If no *method* parameter is given, the METHOD_DEFAULT selection method will be used.
Available selection methods:

- METHOD_DEFAULT is a flag that is the same as not passing the method parameter.
- METHOD_ALLOW_INSIDE_OF_SHAPES is a flag indicating a point on the filled area of a shape will be considered.
- METHOD_DISALLOW_INSIDE_OF_SHAPES is a flag indicating a point must be on the boundary of a filled shape to be considered.
- METHOD_USE_PREFS_FOR_INSIDE is a flag indicating Allow/Disallow inside is determined by the preference value.
- METHOD_INST_NO_ANNOTATION is a flag indicating Annotation will not be considered. The point must be on the instance or shape's bounding box to be considered.
- METHOD_INST_ONLY_CHECK_BBOX is a flag indicating in layout, a click anywhere inside the instance bounding box will be considered. Without this flag, the point must be on an actual shape inside the instance. Note: The flag has no effect on a schematic, where clicks anywhere inside the bounding box are always considered.

Example:

```
decl winInst = api_get_current_window();
decl context = de_get_design_context(winInst);
decl tolerance = db_get_selection_tolerance(winInst);
decl x=10,y=20;
// Get all objects that are selected at a particular x, y location.
decl objList = db_get_objects_selected_at(context, x, y, tolerance);
// Highlight all those selected objects at a particular x,y location
decl idx=0;
for (idx=0; idx < listlen(objList); idx++)
{
   decl objH = nth(idx, objList);
   db_highlight(objH);
}
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_get_selection_tolerance()

Returns the user's selection tolerance for a given window. If the user's setting is in pixels,

then the tolerance is calculated based on the window's current zoom level.

See also: *db_cycle_select_item()* (ael), *db_get_objects_selected_at()* (ael), *db_pick_instance_at()* (ael), *db_pick_object_at()* (ael), *db_pick_shape_at()* (ael), *db_select()* (ael), *db_is_selected()* (ael).

Syntax:

```
db_get_selection_tolerance(winInst);
```

where

winInst is a window returned from a function such as *api_get_current_window()*.

Example:

```
decl winInst = api_get_current_window();
decl context = de_get_design_context(winInst);
// Get user selection tolerance for the current window.
decl tolerance = db_get_selection_tolerance(winInst);
decl x=10,y=20;
decl insthandle=db_pick_instance_at(context, x, y, tolerance);
if (!insthandle)
    return;
// Mark only the first instance at given x,y location as selected.
db_deselect_all(context);
db_select(insthandle, TRUE);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_pick_instance_at()

Returns the instance object which would be selected if the user clicked at the location. This function cycle picks through the instances at the given location. Each sequential call finds the next instance which would be selected if the user clicked at the location. User filters are ignored.

See also: *db_pick_object_at()* (ael), *db_pick_shape_at()* (ael), *db_get_selection_tolerance()* (ael).

Syntax:

```
db_pick_instance_at(context, x, y, tolerance, [, method]);
```

where

context is a DesignContext returned from a function such as *de_get_current_design_context()*.
x , *y* is the x,y coordinate in database units.
tolerance is the user's selection tolerance.
method is an optional selection method. If no *method* parameter is given, the METHOD_DEFAULT selection method will be used.
Available selection methods:

- METHOD_DEFAULT is a flag that is the same as not passing the method parameter.
- METHOD_ALLOW_INSIDE_OF_SHAPES is a flag indicating a point on the filled area of a shape will be considered.
- METHOD_DISALLOW_INSIDE_OF_SHAPES is a flag indicating a point must be on the boundary of a filled shape to be considered.
- METHOD_USE_PREFS_FOR_INSIDE is a flag indicating Allow/Disallow inside is determined by the preference value.
- METHOD_INST_NO_ANNOTATION is a flag indicating Annotation will not be considered. The point must be on the instance or shape's bounding box to be considered.
- METHOD_INST_ONLY_CHECK_BBOX is a flag indicating in layout, a click anywhere inside the instance bounding box will be considered. Without this flag, the point must be on an actual shape inside the instance. Note: The flag has no effect on a schematic, where clicks anywhere inside the bounding box are always considered.

Example:

```
decl winInst = api_get_current_window();
decl context = de_get_design_context(winInst);
// Get user selection tolerance for the current window.
decl tolerance = db_get_selection_tolerance(winInst);
decl x=10,y=20;
// Pick the instance at x, y location that would next be selected.
decl insthandle=db_pick_instance_at(context, x, y, tolerance);
if (!insthandle)
   return;
// Mark only that found instance at given x,y location as selected.
db_deselect_all(context);
db_select(insthandle, TRUE);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_pick_object_at()

Returns the shape object, instance object, or port object which would be selected if the user clicked at the location. This function cycle picks through the objects at the given location. Each sequential call finds the next object which would be selected if the user clicked at the location. User filters are ignored.

See also: *db_pick_instance_at()* (ael), *db_pick_shape_at()* (ael), *db_get_selection_tolerance()* (ael).

Syntax:

```
db_pick_object_at(context, x, y, tolerance, [, method]);
```

where

context is a DesignContext returned from a function such as

de_get_current_design_context().
x , *y* is the x,y coordinate in database units.
tolerance is the user's selection tolerance.
method is an optional selection method. If no *method* parameter is given, the METHOD_DEFAULT selection method will be used.
Available selection methods:

- METHOD_DEFAULT is a flag that is the same as not passing the method parameter.
- METHOD_ALLOW_INSIDE_OF_SHAPES is a flag indicating a point on the filled area of a shape will be considered.
- METHOD_DISALLOW_INSIDE_OF_SHAPES is a flag indicating a point must be on the boundary of a filled shape to be considered.
- METHOD_USE_PREFS_FOR_INSIDE is a flag indicating Allow/Disallow inside is determined by the preference value.
- METHOD_INST_NO_ANNOTATION is a flag indicating Annotation will not be considered. The point must be on the instance or shape's bounding box to be considered.
- METHOD_INST_ONLY_CHECK_BBOX is a flag indicating in layout, a click anywhere inside the instance bounding box will be considered. Without this flag, the point must be on an actual shape inside the instance. Note: The flag has no effect on a schematic, where clicks anywhere inside the bounding box are always considered.

Example:

```
decl winInst = api_get_current_window();
decl context = de_get_design_context(winInst);
// Get user selection tolerance for the current window.
decl tolerance = db_get_selection_tolerance(winInst);
decl x=10,y=20;
// Pick the object at x, y location that would next be selected.
decl objhandle=db_pick_object_at(context, x, y, tolerance);
if (!objhandle)
   return;
// Mark only the found object at given x,y location as selected.
db_deselect_all(context);
db_select(objhandle, TRUE);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_pick_shape_at()

Returns the shape object which would be selected if the user clicked at the location. This function cycle picks through the shapes at the given location. Each sequential call finds the next shape which would be selected if the user clicked at the location. User filters are ignored.

See also: *db_pick_instance_at()* (ael), *db_pick_object_at()* (ael), *db_get_selection_tolerance()* (ael).

Syntax:

```
db_pick_shape_at(context, x, y, tolerance, [, method]);
```

where

context is a DesignContext returned from a function such as *de_get_current_design_context()*.
x , *y* is the x,y coordinate in database units.
tolerance is the user's selection tolerance.
method is an optional selection method. If no *method* parameter is given, the METHOD_DEFAULT selection method will be used.
Available selection methods:

- METHOD_DEFAULT is a flag that is the same as not passing the method parameter.
- METHOD_ALLOW_INSIDE_OF_SHAPES is a flag indicating a point on the filled area of a shape will be considered.
- METHOD_DISALLOW_INSIDE_OF_SHAPES is a flag indicating a point must be on the boundary of a filled shape to be considered.
- METHOD_USE_PREFS_FOR_INSIDE is a flag indicating Allow/Disallow inside is determined by the preference value.
- METHOD_INST_NO_ANNOTATION is a flag indicating Annotation will not be considered. The point must be on the instance or shape's bounding box to be considered.
- METHOD_INST_ONLY_CHECK_BBOX is a flag indicating in layout, a click anywhere inside the instance bounding box will be considered. Without this flag, the point must be on an actual shape inside the instance. Note: The flag has no effect on a schematic, where clicks anywhere inside the bounding box are always considered.

Example:

```
decl winInst = api_get_current_window();
decl context = de_get_design_context(winInst);
// Get user selection tolerance for the current window.
decl tolerance = db_get_selection_tolerance(winInst);
decl x=10,y=20;
// Pick the shape object at x, y location that would next be selected.
decl shapehandle=db_pick_shape_at(context, x, y, tolerance);
if (!shapehandle)
   return;
// Mark only the found shape at given x,y location as selected.
db_deselect_all(context);
db_select(shapehandle, TRUE);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_select_all()

Select all items in a DesignContext. User filters are honored.

See also: *db_deselect_all()* (ael), *db_is_selected()* (ael), *db_select()* (ael).

Syntax:

```
db_select_all(context);
```

where

context is a DesignContext returned from a function such as *de_get_current_design_context()*.

Example:

```
decl context = de_get_current_design_context();
// Select all objects for given DesignContext.
db_select_all(context);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_select_area()

Select all items in a given rectangular area of a DesignContext. User filters are honored.

See also: *db_deselect_area()* (ael), *db_deselect_all()* (ael), *db_select()* (ael), *db_select_all()* (ael).

Syntax:

```
db_select_area(context, x1, y1, x2, y2);
```

where

context is a DesignContext returned from a function such as *de_get_current_design_context()*.
x1, *y1* is the x, y coordinate of the first corner of the rectangular area in database units.
x2, *y2* is the x,y coordinate of the second (opposite) corner of the rectangular area in database units.

Example:

```
decl winInst = api_get_current_window();
decl context = de_get_design_context(winInst);
// Rectangle area to select in database units.
decl x1=10,y1=20, x2=30, y2=40;
// Select all the items in the given rectangular area.
db_select_area(context, x1, y1, x2, y2);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

Shape Functions

This section describes the following shape function in detail:

db_get_arc_angle()

This functions returns the sweep angle of a given arc shape. The returned arc sweep angle is in 0.001 degree units.

Syntax

```
decl sweepAngle = db_get_arc_angle( dbShape );
```

Where,

- *dbShape* is a shape, dg, or shape iterator.

Example

```
if (db_shape_is_arc(dgH))
{
  decl arcAngle = db_get_arc_angle(dgH);
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_arc_center() (ael)
db_get_arc_start() (ael)
Shape Functions (ael)

Where Used: (ael)

Schematic, Layout

db_get_arc_center()

This functions returns the center coordinate of a given arc shape. The returned coordinate is in DB units.

Syntax

```
decl arcCenter = db_get_arc_center( dbShape );
```

Where,

- *dbShape* is a shape, dg, or shape iterator.

Example

```
if (db_shape_is_arc(dgH))
{
  decl arcCenter = db_get_arc_center(dgH);
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_arc_angle() (ael)
db_get_arc_start() (ael)
db_get_x() (ael)
db_get_y() (ael)
Shape Functions (ael)

Where Used: (ael)

Schematic, Layout

db_get_arc_start()

This functions returns the start coordinate of a given arc shape. The returned coordinate is in DB units.

Syntax

```
decl arcStart = db_get_arc_start( dbShape );
```

Where,

- *dbShape* is a shape, dg, or shape iterator.

Example

```
if (db_shape_is_arc(dgH))
{
  decl arcStartCoord = db_get_arc_start(dgH);
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_arc_angle() (ael)
db_get_arc_center() (ael)
db_get_x() (ael)
db_get_y() (ael)
Shape Functions (ael)

Where Used: (ael)

Schematic, Layout

db_get_ellipse_center()

This functions returns the center coordinate of a given ellipse shape. The returned coordinate is in DB units.

Syntax

```
decl ellipseCenter = db_get_ellipse_center( dbShape );
```

Where,

- *dbShape* is an ellipse shape, dg, or shape iterator.

Example

```
if (db_shape_is_ellipse(dgH))
{
  decl ellipseCenter = db_get_ellipse_center(dgH);
```

```
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_ellipse_x_radius() (ael)
db_get_ellipse_y_radius() (ael)
db_get_x() (ael)
db_get_y() (ael)
Shape Functions (ael)

***Where Used:* (ael)**

Schematic, Layout

db_get_ellipse_x_radius()

This functions returns the x radius of a given ellipse shape. The returned radius is in DB units.

Syntax

```
decl xRadius = db_get_ellipse_x_radius( dbShape );
```

Where,

- *dbShape* is an ellipse shape, dg, or shape iterator.

Example

```
if (db_shape_is_ellipse(dgH))
{
  decl xRadius = db_get_ellipse_x_radius(dgH);
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_ellipse_center() (ael)
db_get_ellipse_y_radius() (ael)
Shape Functions (ael)

***Where Used:* (ael)**

Schematic, Layout

db_get_ellipse_y_radius()

This functions returns the y radius of a given ellipse shape. The returned radius is in DB units.

Syntax

```
decl yRadius = db_get_ellipse_y_radius( dbShape );
```

Where,

- *dbShape* is an ellipse shape, dg, or shape iterator.

Example

```
if (db_shape_is_ellipse(dgH))
{
  decl yRadius = db_get_ellipse_y_radius(dgH);
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_ellipse_center() (ael)
db_get_ellipse_x_radius() (ael)
Shape Functions (ael)

Where Used: (ael)

Schematic, Layout

db_get_shape_layer()

Returns the layer number of a given shape.

See also: *db_get_shape_bbox()* (ael).

Syntax:

```
db_get_shape_layer(shape);
```

where
shape is a shape object or shape iterator returned from a function such as *db_create_shape_iter()*.

Example:

```
decl context = de_get_current_design_context();
// Get first selected shape in the current design context.
decl shapeIter = db_create_shape_iter(context);
shapeIter = db_shape_iter_limit_selected(shapeIter);
```

```
if (db_shape_iter_is_valid(shapeIter) == FALSE)
  return;
// Get layer number of first selected shape.
decl layerNo = db_get_shape_layer(shapeIter);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_get_shape_layerid()

This function returns the *LayerId* (ael) for a given shape.

Syntax

```
layerId = db_get_shape_layerid(dbShape);
```

Where,

- *dbShape* is a shape, dg, or shape iterator.

Example

```
decl context = de_get_current_design_context();
decl shapeIter = db_create_shape_iter(context);
for( ; db_shape_iter_is_valid(shapeIter);
   shapeIter = db_shape_iter_get_next(shapeIter))
{
   decl layerId = db_get_shape_layerid(shapeIter);
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

***Where Used:* (ael)**

Schematic, Layout

See also

Shape Functions (ael)
LayerId Overview (ael)

db_get_shape_net()

This functions returns the Net associated with the given shape. Returns NULL if there is no Net associated with the given shape.

Syntax

```
decl dbNet = db_get_shape_net( dbShape );
```

Where,

- *dbShape* is a shape, dg, or shape iterator.

Example

```
// Get the net of the first shape in the design.
decl dbNet = NULL;
decl dbShapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(dbShapeIter))
{
   dbNet = db_get_shape_net(dbshapeIter);
}
return dbNet;
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Net Functions (ael)
Shape Functions (ael)

Where Used: (ael)

Schematic, Layout

db_get_shape_text_string()

This functions returns a string containing the text within the given text shape or annotation shape. If the shape is not a text shape or an annotation shape, an AEL error will occur.

Syntax

```
decl textStr = db_get_shape_text_string( dbShape );
```

Where,
dbShape is a text or annotation shape.

Example

```
decl iter = db_create_shape_iter(context);
for( ;  db_shape_iter_is_valid(iter); iter = db_shape_iter_get_next(iter) )
{
   ...
   // Get the text from text shapes or annotation shapes.
   if( db_shape_is_text_or_annotation(iter) )
   {
       decl textStr = db_get_shape_text_string(iter);
       ...
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_shape_is_annotation() (ael),
db_shape_is_text() (ael),
db_shape_is_text_or_annotation() (ael),
Shape Functions (ael)

***Where Used:* (ael)**

Schematic, Layout

db_get_shape_type_description()

Returns a string that describes the type of the given shape, such as 'annotation", "arc", "construction line", "ellipse", "path", "polygon", "polyline", "rectangle", "text", "trace", "wire", or "wire label".

Note
The description of the various shapes is for display purposes and is subject to change. Do not use this function as a way to identify shapes in conditional statements, use the *db_shape_is_<arc,text,circle,rectangle...>()* functions instead.

Syntax

```
decl text = db_get_shape_type_description( dbShape );
```

Where,

- *dbShape* is a shape object.

Example

```
decl iter = db_create_shape_iter(context);
for ( ;  db_shape_iter_is_valid(iter); iter = db_shape_iter_get_next(iter) )
{
   ...
   // Get a text string description of the type of the shape.
   decl shapeTypeStr = db_get_shape_type_description(iter);
   ...
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Shape Functions (ael)

***Where Used* (ael)**

Schematic, Layout

db_get_text_absolute()

Function returns the boolean (TRUE,FALSE) absolute flag from a given text or annotation shape object. The text absolute flag indicates whether the text can rotate or not.

Syntax

```
decl isAbsolute = db_get_text_absolute(dbShape);
```

Where,

dbShape is a shape iterator or a shape object.

Example

```
decl context = de_get_current_design_context();

// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
    return;

decl isAbsolute = db_get_text_absolute(shapeIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Shape Functions (ael)
db_get_text_angle() (ael)
db_get_text_font_name() (ael)
db_get_text_height() (ael)
db_get_text_justification() (ael)
db_get_text_origin() (ael)
db_shape_is_annotation() (ael)
db_shape_is_text() (ael)

Where Used (ael)

Schematic, Layout

db_get_text_angle()

Returns the text angle in 0.001 degree units of the given text or annotation shape object.

Syntax

```
decl textAngle = db_get_text_angle(dbShape);
```

Where,

dbShape is a shape iterator or a shape object.

Example

```
decl context = de_get_current_design_context();

// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
    return;

decl textAngle = db_get_text_angle(shapeIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Shape Functions (ael)
db_get_text_absolute() (ael)
db_get_text_font_name() (ael)
db_get_text_height() (ael)
db_get_text_justification() (ael)
db_get_text_origin() (ael)
db_shape_is_annotation() (ael)
db_shape_is_text() (ael)

Where Used (ael)

Schematic, Layout

db_get_text_font_name()

Function returns the string font name for a given text or annotation shape object.

Syntax

```
decl textFontName = db_get_text_font_name(dbShape);
```

Where,

dbShape is a shape iterator or a shape object.

Example

```
decl context = de_get_current_design_context();

// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
    return;

decl textFontName = db_get_text_font_name(shapeIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Shape Functions (ael)
db_get_text_absolute() (ael)
db_get_text_angle() (ael)
db_get_text_height() (ael)
db_get_text_justification() (ael)
db_get_text_origin() (ael)
db_shape_is_annotation() (ael)
db_shape_is_text() (ael)

Where Used (ael)

Schematic, Layout

db_get_text_height()

Function returns the text height in database units for a given text or annotation shape object.

Syntax

```
decl textHeight = db_get_text_height(dbShape);
```

Where,

dbShape is a shape iterator or a shape object.

Example

```
decl context = de_get_current_design_context();

// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
    return;

decl textHeight = db_get_text_height(shapeIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Shape Functions (ael)
db_get_text_absolute() (ael)
db_get_text_angle() (ael)
db_get_text_font_name() (ael)

db_get_text_justification() (ael)
db_get_text_origin() (ael)
db_shape_is_annotation() (ael)
db_shape_is_text() (ael)

Where Used (ael)

Schematic, Layout

db_get_text_justification()

Function to return the justification attribute code from a text or annotation object. The returned value is a binary "OR'ed" integer representing the text justification as a combination of the following AEL constants: **DB_BOT_JUST**, **DB_MID_JUST**, **DB_TOP_JUST**, **DB_LEFT_JUST**, **DB_CENTER_JUST**, **DB_RIGHT_JUST**.

Syntax

```
decl justAttribs = db_get_text_justification(dbShape);
```

Where,

dbShape is a shape iterator or a shape object.

Example

```
decl context = de_get_current_design_context();

// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
    return;

decl justAttribs = db_get_text_justification(shapeIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Shape Functions (ael)
db_get_text_absolute() (ael)
db_get_text_angle() (ael)
db_get_text_font_name() (ael)
db_get_text_height() (ael)
db_get_text_origin() (ael)
db_shape_is_annotation() (ael)
db_shape_is_text() (ael)

Where Used (ael)

Schematic, Layout

db_get_text_origin()

Returns the origin's (anchor point) coordinate in database units of the given text or annotation shape object.

Syntax

```
decl coordH = db_get_text_origin(dbShape);
```

Where,

dbShape is a shape iterator or a shape object.

Example

```
decl context = de_get_current_design_context();

// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
    return;

decl coordH = db_get_text_origin(shapeIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Shape Functions (ael)
db_get_text_absolute() (ael)
db_get_text_angle() (ael)
db_get_text_font_name() (ael)
db_get_text_height() (ael)
db_get_text_justification() (ael)
db_get_x() (ael)
db_get_y() (ael)
db_shape_is_annotation() (ael)
db_shape_is_text() (ael)

Where Used (ael)

Schematic, Layout

db_is_cell_name_display()

Returns TRUE if the given annotation shape object is a cell name display.

Syntax

```
decl bIsCellName = db_is_cell_name_display(dbShape);
```

Where,

dbShape is a shape iterator or a shape object.

Example

```
decl context = de_get_current_design_context();

// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
    return;

decl isCellNameDisplay = db_is_cell_name_display(shapeIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Shape Functions (ael)
db_is_inst_name_display() (ael)
db_shape_is_annotation() (ael)
db_shape_is_wire_label() (ael)

Where Used (ael)

Schematic, Layout

db_is_inst_name_display()

Returns TRUE if the given annotation shape object is an instance name display.

Syntax

```
decl bIsInstanceName = db_is_inst_name_display(dbShape);
```

Where,

dbShape is a shape iterator or a shape object.

Example

```
decl context = de_get_current_design_context();

// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
    return;

decl isInstNameDisplay = db_is_inst_name_display(shapeIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Shape Functions (ael)
db_is_cell_name_display() (ael)
db_shape_is_annotation() (ael)
db_shape_is_wire_label() (ael)

Where Used (ael)

Schematic, Layout

db_shape_has_selected_points()

Returns TRUE if a given shape has any point(s) selected. If the shape has no points selected, then FALSE is returned.

Syntax

```
decl hasPointsSelected = db_shape_has_selected_points( dbShape );
```

Where,

- *dbShape* is a shape, dg, or shape iterator.

Example

```
...
decl hasPointsSelected = db_shape_has_selected_points( dbShape );
...
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Shape Functions (ael)
Selection Functions (ael)

Where Used: (ael)

Schematic, Layout

db_shape_is_annotation()

This returns TRUE if the shape is a text display which we consider to be an annotation – not directly selectable by the user.
That includes wire/pin/instance labels and instance annotation.

Syntax

```
decl bIsAnnotation = db_shape_is_annotation(object);
```

Where,

object is a shape iterator or a shape object.

Example

```
decl context = de_get_current_design_context();

// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
    return;

decl isShapeAnnot = db_shape_is_annotation(shapeIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Shape Functions (ael)
db_is_inst_name_display() (ael)
db_is_cell_name_display() (ael)
db_shape_is_wire_label() (ael)

***Where Used* (ael)**

Schematic, Layout

db_shape_is_circle()

Returns TRUE if the given shape is a circle. Returns FALSE if the given shape is not a circle. A circle is an ellipse.

See also: *db_is_shape()* (ael), *db_shape_is_arc()* (ael), *db_shape_is_const_line()* (ael), *db_shape_is_ellipse()* (ael), *db_shape_is_path()* (ael), *db_shape_is_polygon()* (ael), *db_shape_is_polyline()* (ael), *db_shape_is_wire_label()* (ael), *Shape Functions* (ael).

Syntax:

```
bIsCircle = db_shape_is_circle(dbShape);
```

where

dbShape is a shape, dg, or a shape iterator.

Example:

```
decl context = de_get_current_design_context();
// Return first circle shape in design.
decl shapeIter = db_create_shape_iter(context);
for( ; db_shape_iter_is_valid(shapeIter);
    shapeIter = db_shape_iter_get_next(shapeIter))
{
    if (db_shape_is_circle(shapeIter))
```

```
        return db_shape_iter_get_shape(shapeIter);
}
```

Version Introduced:

ADS 2011

Version Compatible

ADS 2011 and newer versions

***Where Used:* (ael)**

Schematic, Layout

db_shape_is_dot()

Returns TRUE if the given shape is a dot. Returns FALSE if the given shape is not a dot.

Note
A Dot shape represents a dot at a location point that can have a non-zero width and height.

See also: *db_is_shape()* (ael), *db_shape_is_arc()* (ael), *db_shape_is_const_line()* (ael), *db_shape_is_ellipse()* (ael), *db_shape_is_path()* (ael), *db_shape_is_polygon()* (ael), *db_shape_is_polyline()* (ael), *db_shape_is_wire_label()* (ael), *Shape Functions* (ael).

Syntax:

```
bIsShape = db_shape_is_dot(dbShape);
```

where

dbShape is a shape, dg, or a shape iterator.

Example:

```
decl context = de_get_current_design_context();
// Return first dot shape in design.
decl shapeIter = db_create_shape_iter(context);
for( ; db_shape_iter_is_valid(shapeIter);
    shapeIter = db_shape_iter_get_next(shapeIter))
{
    if (db_shape_is_dot(shapeIter))
        return db_shape_iter_get_shape(shapeIter);
}
```

Version Introduced:

ADS 2011

Version Compatible

ADS 2011 and newer versions

***Where Used:* (ael)**

Schematic, Layout

db_shape_is_text_or_annotation()

Returns TRUE if the given shape is a text, text display, or annotation object.

Syntax

```
decl bIsTextOrAnnot = db_shape_is_text_or_annotation(object);
```

Where,
object is a shape iterator or a shape object.

Example

```
decl context = de_get_current_design_context();

// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
    return;

decl isShapeTxtOrAnnot = db_shape_is_text_or_annotation(shapeIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Shape Functions (ael)

Where Used (ael)

Schematic, Layout

db_is_shape_text()

Returns TRUE if the given shape is a text object.

Syntax

```
decl bIsText = db_shape_is_text(object);
```

Where,
object is a shape iterator or a shape object.

Example

```
decl context = de_get_current_design_context();

// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
    return;

decl isShapeText = db_shape_is_text(shapeIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Shape Functions (ael)

Where Used (ael)

Schematic, Layout

db_shape_is_trace()

Returns TRUE if the given shape is a trace.

Syntax

```
decl isTrace = db_shape_is_trace(object);
```

Where,
object is a shape iterator or a shape object.

Example

```
decl context = de_get_current_design_context();
// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
   return;
decl isShapeTrace = db_shape_is_trace(shapeIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Shape Functions (ael)

Where Used (ael)

Schematic, Layout

db_shape_is_wire_or_trace()

Returns TRUE if the given shape is a wire or a trace object.

Syntax

```
decl bIsWireOrTrace = db_shape_is_wire_or_trace(object);
```

Where,
object is a shape iterator or a shape object.

Example

```
decl context = de_get_current_design_context();

// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
    return;

decl isShapeWireOrTrace = db_shape_is_wire_or_trace(shapeIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Shape Functions (ael)

Where Used (ael)

Schematic, Layout

db_shape_is_wire()

Returns TRUE if the given shape is a wire.

Syntax

decl isWire = db_shape_is_wire(object);

Where,
object is a shape iterator or a shape object.

Example

```
decl context = de_get_current_design_context();
// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
   return;
decl isShapeWire = db_shape_is_wire(shapeIter);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Shape Functions (ael)

Where Used (ael)

Schematic, Layout

db_get_shape_bbox()

Returns a handle to the bounding box of a shape. This is the smallest rectangle that completely encloses all data of a shape.

See also: *db_get_instance_bbox()* (ael), *db_get_shape_layer()* (ael).

Syntax:

```
db_get_shape_bbox(shape);
```

where
shape is a shape object or shape iterator returned from a function such as *db_create_shape_iter()*.

Example:

```
decl context = de_get_current_design_context();
// Get first selected shape in the current design context.
decl shapeIter = db_create_shape_iter(context);
shapeIter = db_shape_iter_limit_selected(shapeIter);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
  return;
// Get bounding box of first selected shape.
decl bboxH = db_get_shape_bbox(shapeIter);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_is_arc()

Returns TRUE if the given shape is an arc. Returns FALSE if the shape object is not an arc.

See also: *db_is_shape()* (ael), *db_shape_is_const_line()* (ael), *db_shape_is_dot()* (ael), *db_shape_is_ellipse()* (ael), *db_shape_is_path()* (ael), *db_shape_is_polygon()* (ael), *db_shape_is_polyline()* (ael), *db_shape_is_rectangle()* (ael), *db_shape_is_wire_label()* (ael).

Syntax:

```
db_shape_is_arc(object);
```

where

object is a shape iterator or a shape object.

Example:

```
decl context = de_get_current_design_context();
// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
```

```
if (db_shape_iter_is_valid(shapeIter) == FALSE)
   return;
decl isShapeArc = db_shape_is_arc(shapeIter);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_is_const_line()

Returns TRUE if the given shape is a construction line. Returns FALSE if the shape object is not a construction line.

See also: *db_is_shape()* (ael), *db_shape_is_arc()* (ael), *db_shape_is_dot()* (ael), *db_shape_is_ellipse()* (ael), *db_shape_is_path()* (ael), *db_shape_is_polygon()* (ael), *db_shape_is_polyline()* (ael), *db_shape_is_rectangle()* (ael), *db_shape_is_wire_label()* (ael).

Syntax:

```
db_shape_is_const_line(object);
```

where

object is a shape iterator or a shape object.

Example:

```
decl context = de_get_current_design_context();
// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
   return;
decl isShapeConstLine = db_shape_is_const_line(shapeIter);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_is_ellipse()

Returns TRUE if the given shape is an ellipse. Returns FALSE if the shape object is not an ellipse. Circles are ellipses.

See also: *db_is_shape()* (ael), *db_shape_is_arc()* (ael), *db_shape_is_const_line()* (ael), *db_shape_is_dot()* (ael), *db_shape_is_path()* (ael), *db_shape_is_polygon()* (ael), *db_shape_is_polyline()* (ael), *db_shape_is_rectangle()* (ael), *db_shape_is_wire_label()* (ael).

Syntax:

```
db_shape_is_ellipse(object);
```

where

object is a shape iterator or a shape object.

Example:

```
decl context = de_get_current_design_context();
// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
   return;
// Test if shape is an ellipse or circle.
decl isShapeEllipse = db_shape_is_ellipse(shapeIter);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_is_path()

Returns TRUE if the given shape is a path. Returns FALSE if the shape object is not a path.

See also: *db_is_shape()* (ael), *db_shape_is_arc()* (ael), *db_shape_is_const_line()* (ael), *db_shape_is_dot()* (ael), *db_shape_is_ellipse()* (ael), *db_shape_is_polygon()* (ael), *db_shape_is_polyline()* (ael), *db_shape_is_rectangle()* (ael), *db_shape_is_wire_label()* (ael).

Syntax:

```
db_shape_is_path(object);
```

where

object is a shape iterator or a shape object.

Example:

```
decl context = de_get_current_design_context();
// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
   return;
decl isShapePath = db_shape_is_path(shapeIter);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_is_polygon()

Returns TRUE if the given shape is a polygon. Returns FALSE if the shape object is not a polygon.

See also: *db_is_shape()* (ael), *db_shape_is_arc()* (ael), *db_shape_is_const_line()* (ael), *db_shape_is_dot()* (ael), *db_shape_is_ellipse()* (ael), *db_shape_is_path()* (ael), *db_shape_is_polyline()* (ael), *db_shape_is_rectangle()* (ael), *db_shape_is_wire_label()* (ael).

Syntax:

```
db_shape_is_polygon(object);
```

where

object is a shape iterator or a shape object.

Example:

```
decl context = de_get_current_design_context();
// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
   return;
decl isShapePolygon = db_shape_is_polygon(shapeIter);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_is_polyline()

Returns TRUE if the given shape is a polyline. Returns FALSE if the shape object is not a polyline.

See also: *db_is_shape()* (ael), *db_shape_is_arc()* (ael), *db_shape_is_const_line()* (ael), *db_shape_is_dot()* (ael), *db_shape_is_ellipse()* (ael), *db_shape_is_path()* (ael), *db_shape_is_polygon()* (ael), *db_shape_is_rectangle()* (ael), *db_shape_is_wire_label()* (ael).

Syntax:

```
db_shape_is_polyline(object);
```

where

object is a shape iterator or a shape object.

Example:

```
decl context = de_get_current_design_context();
// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
   return;
```

```
decl isShapePolyline = db_shape_is_polyline(shapeIter);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_is_rectangle()

Returns TRUE if the given shape is a rectangle. Returns FALSE if the shape object is not a rectangle.

See also: *db_is_shape()* (ael), *db_shape_is_arc()* (ael), *db_shape_is_const_line()* (ael), *db_shape_is_dot()* (ael), *db_shape_is_ellipse()* (ael), *db_shape_is_path()* (ael), *db_shape_is_polygon()* (ael), *db_shape_is_polyline()* (ael), *db_shape_is_wire_label()* (ael).

Syntax:

```
db_shape_is_rectangle(object);
```

where

object is a shape iterator or a shape object.

Example:

```
decl context = de_get_current_design_context();
// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
   return;
decl isShapeRectangle = db_shape_is_rectangle(shapeIter);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_is_wire_label()

Returns TRUE if the given shape is a wire label. Returns FALSE if the shape object is not a wire label.

See also: *db_is_shape()* (ael), *db_shape_is_arc()* (ael), *db_shape_is_const_line()* (ael), *db_shape_is_dot()* (ael), *db_shape_is_ellipse()* (ael), *db_shape_is_path()* (ael), *db_shape_is_polygon()* (ael), *db_shape_is_polyline()* (ael), *db_shape_is_rectangle()* (ael).

Syntax:

```
db_shape_is_wire_label(object);
```

where

object is a shape iterator or a shape object.

Example:

```
decl context = de_get_current_design_context();
// Get first shape in current design context.
decl shapeIter = db_create_shape_iter(context);
if (db_shape_iter_is_valid(shapeIter) == FALSE)
   return;
decl isShapeWireLabel = db_shape_is_wire_label(shapeIter);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

Shape Iterator Functions

This section describes each shape iterator function in detail. The functions are listed in alphabetical order.

db_shape_iter_exclude_invisible_layers()

Function to exclude invisible layers in the iteration. By default, a shape iterator will iterate over shapes on protected and invisible layers.
If the iterator has been started (used to find a shape), then calling this function gives an AEL error.
If the iterator was created from a Net, this function can not be used, an AEL error will occur.

This function does not work with shape iterators created from Nets.

See also: *db_create_shape_iter()* (ael), *db_shape_iter_get_next()* (ael), *db_shape_iter_exclude_protected_layers()* (ael).

Syntax:

```
db_shape_iter_include_exclude_layers(shapeIter);
```

where
shapeIter is a shape iterator returned from a function such as *db_create_shape_iter()*.

Example:

```
//Example of how to deselect all selected shapes on
// only the visible layers.
decl context = de_get_current_design_context();
decl iter = db_create_shape_iter(context);
//Iterate over all visible layers' shapes, excluding the invisible layers' shapes.
iter = db_shape_iter_exclude_invisible_layers(iter);
for ( ; db_shape_iter_is_valid(iter); iter = db_shape_iter_get_next(iter))
{
   //Deselect all visible layers' shapes.
   if (db_is_selected(iter))
       db_select(iter, FALSE);
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

***Where Used:* (ael)**

Schematic, Layout

db_shape_iter_exclude_invisible_layers()

Function to exclude invisible layers in the iteration. By default, a shape iterator will iterate over shapes on protected and invisible layers.
If the iterator has been started (used to find a shape), then calling this function gives an AEL error.
If the iterator was created from a Net, this function can not be used, an AEL error will occur.

This function does not work with shape iterators created from Nets.

See also: *db_create_shape_iter()* (ael), *db_shape_iter_get_next()* (ael), *db_shape_iter_exclude_protected_layers()* (ael).

Syntax:

```
db_shape_iter_include_exclude_layers(shapeIter);
```

where
shapeIter is a shape iterator returned from a function such as *db_create_shape_iter()*.

Example:

```
//Example of how to deselect all selected shapes on
// only the visible layers.
decl context = de_get_current_design_context();
decl iter = db_create_shape_iter(context);
//Iterate over all visible layers' shapes, excluding the invisible layers' shapes.
iter = db_shape_iter_exclude_invisible_layers(iter);
for ( ; db_shape_iter_is_valid(iter); iter = db_shape_iter_get_next(iter))
{
   //Deselect all visible layers' shapes.
   if (db_is_selected(iter))
       db_select(iter, FALSE);
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

***Where Used:* (ael)**

Schematic, Layout

db_shape_iter_exclude_protected_layers()

Function to exclude protected layers in the iteration. By default, a shape iterator will iterate over shapes on protected and invisible layers.
If the iterator has been started (used to find a shape), then calling this function gives an AEL error.
If the iterator was created from a Net, this function can not be used, an AEL error will occur.

This function does not work with shape iterators created from Nets.

See also: *db_create_shape_iter()* (ael), *db_shape_iter_get_next()* (ael), *db_shape_iter_exclude_invisible_layers()* (ael).

Syntax:

```
db_shape_iter_exclude_protected_layers(shapeIter);
```

where
shapeIter is a shape iterator returned from a function such as *db_create_shape_iter()*.

Example:

```
//Example of how to deselect all selected shapes that are not protected.
decl context = de_get_current_design_context();
decl iter = db_create_shape_iter(context);
//Iterate over all unprotected layers' shapes.
iter = db_shape_iter_exclude_protected_layers(iter);
for ( ; db_shape_iter_is_valid(iter); iter = db_shape_iter_get_next(iter))
{
   //Deselect all  unprotected layers' shapes.
   if (db_is_selected(iter))
       db_select(iter, FALSE);
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

***Where Used:* (ael)**

Schematic, Layout

db_shape_iter_limit_layerid()

Function to limit the shape iteration to a particular LayerId. Will use the layer even if it is protected or invisible.
If the iterator has been started (used to find a shape), then calling this function gives an AEL error.

See also

Syntax

```
shapeIter = db_shape_iter_limit_layerid(shapeIter, layerId);
```

Where,

- *shapeIter* is a shape iterator returned from a function such as *db_create_shape_iter()* (ael).
- *layerid* is a LayerId.

Example

```
decl context = de_get_current_design_context();
decl layerId = db_get_layerid(context, "Metal1", "drawing");
decl shapeIter = db_create_shape_iter(context);
shapeIter = db_shape_iter_limit_layerid(shapeIter, layerId);
for( ; db_shape_iter_is_valid(shapeIter);
  shapeIter = db_shape_iter_get_next(shapeIter))
{
    ...
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

Where Used: (ael)

Schematic, Layout

db_create_shape_iter()

Returns an iterator to the first shape belonging to a given *DesignContext* (ael) or a give Net.
If no shape is available in the given design context or given Net, then the iterator returned will be invalid. The function *db_shape_iter_is_valid()* (ael) can be used to test if an iterator is valid.

By default, a shape iterator will iterate over shapes on protected and invisible layers. For a design context's shape iteration, to disable iteration of shapes on protected and/or invisible layers, the functions *db_shape_iter_exclude_invisible_layers()* (ael), *db_shape_iter_exclude_protected_layers()* (ael) may be used. Please note annotation/text boxes are ignored by the shape iterators, so these objects are not included in an iteration of shapes.

Note
For a net's shape iteration, you can not exclude the protected and invisible layers during shape iteration. If the functions *db_shape_iter_exclude_invisible_layers()* (ael), *db_shape_iter_exclude_protected_layers()* (ael) are used with a shape iterator created from a net, an AEL error will occur.

Note
For a net's shape iteration, shapes on a pin are not included in the iteration of a net's shapes.

See also: *db_shape_iter_enable_caching()* (ael), *db_shape_iter_get_shape()* (ael), *db_shape_iter_get_next()* (ael), *db_shape_iter_exclude_invisible_layers()* (ael), *db_shape_iter_exclude_protected_layers()* (ael), *db_shape_iter_limit_layer()* (ael), *db_shape_iter_limit_purpose()* (ael), *db_shape_iter_limit_region()* (ael),

db_shape_iter_limit_selected() (ael), *db_shape_iter_is_valid()* (ael).

Syntax:

```
db_create_shape_iter(object);
```

where
object is a DesignContext or a Net.

Example:

```
decl context = de_get_current_design_context();
decl shapeIter = db_create_shape_iter(context);
shapeIter = db_shape_iter_exclude_invisible_layers(shapeIter);
shapeIter = db_shape_iter_exclude_protected_layers(shapeIter);
// Select the first visible and unprotected shape in the DesignContext.
if (db_shape_iter_is_valid(shapeIter))
   db_select(shapeIter);
```

Version Introduced

ADS 2009 Update 1
ADS 2011 (Added shape iteration for a net)

Version Compatible

ADS 2009 Update 1 and newer versions
ADS 2011 and newer versions (Shape iteration for a net is available)

Where Used: **(ael)**

Schematic, Layout

db_shape_iter_enable_caching()

Function to make the iterator cache its list.
This function is used when adding or removing shapes during iteration. Use of the function causes a snapshot of the iterator's list to be taken when the iterator is first used to get a shape, thus preventing interference between the iterator and operations which modify the design.
If the iterator has been started (used to find a shape), then calling this function gives an AEL error.

See also: *db_create_shape_iter()* (ael).

Syntax:

```
db_shape_iter_enable_caching(iter);
```

where
iter is a shape iterator returned from a function such as *db_create_shape_iter()*.

Example:

```
//Example of how to deselect all selected shapes.
decl context = de_get_current_design_context();
decl iter = db_create_shape_iter(context);
//Use shape iterators based off current snapshot of what shapes
//are selected in the current design context.
```

```
iter = db_shape_iter_enable_caching(iter);
for ( ; db_shape_iter_is_valid(iter); iter = db_shape_iter_get_next(iter))
{
   de_deselect_all();
   db_select(iter);
   de_delete(); // Delete shape.
}
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_iter_get_next()

Returns an iterator to the next shape after the given shape iterator. If there is no next shape iterator available, an invalid shape iterator is returned. The function *db_shape_iter_is_valid()* (ael) can be used to test if an iterator is valid.

See also: *db_create_shape_iter()* (ael), *db_shape_iter_is_valid()* (ael), *db_shape_iter_include_invisible_layers()* (ael), *db_shape_iter_include_protected_layers()* (ael).

Syntax:

```
db_shape_iter_get_next(iter);
```

where
iter is a shape iterator returned from a function such as *db_create_shape_iter()*.

Example:

```
decl context = de_get_current_design_context();
// Deselect any selected shapes in the current design context.
decl shapeIter = db_create_shape_iter(context);
shapeIter = db_shape_iter_limit_selected(shapeIter);
for ( ; db_shape_iter_is_valid(shapeIter); shapeIter = db_shape_iter_get_next(shapeIter))
   db_select_shape(shapeIter, FALSE);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_iter_get_shape()

Returns a shape object that is referenced by a given shape iterator. Returns NULL if no valid shape object is being referenced by the given shape iterator.
This function is not normally needed since an iterator can be passed directly to a function expecting a shape. The most common use is to build a list of shapes that will be saved.

See also: *db_create_shape_iter()* (ael).

Syntax:

```
db_shape_iter_get_shape(iter);
```

where

iter is a shape iterator returned from a function such as *db_create_shape_iter()*.

Example:

```
decl context = de_get_current_design_context();
decl iter = db_create_shape_iter(context);
// Build a list of shapes that will be used.
decl shapeList = list();
for ( ; db_shape_iter_is_valid(iter); iter = db_shape_iter_get_next(iter))
   shapeList = append(shapeList, list(db_shape_iter_get_shape(iter)));
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_iter_include_invisible_layers()

Function to include invisible layers in the iteration, which is no longer necessary since by default, a shape iterator will iterate over shapes on protected and invisible layers.
If the iterator has been started (used to find a shape), then calling this function gives an AEL error.

See also: *db_create_shape_iter()* (ael), *db_shape_iter_get_next()* (ael), *db_shape_iter_exclude_invisible_layers()* (ael), *db_shape_iter_exclude_protected_layers()* (ael), *db_shape_iter_include_protected_layers()* (ael).

Syntax:

```
db_shape_iter_include_invisible_layers(shapeIter);
```

where
shapeIter is a shape iterator returned from a function such as *db_create_shape_iter()*.

Example:

```
//Example of how to deselect all selected shapes on both
//invisible and visible layers.
decl context = de_get_current_design_context();
decl iter = db_create_shape_iter(context);
//Iterate over all invisible and visible layers' shapes.
iter = db_shape_iter_include_invisible_layers(iter);
for ( ; db_shape_iter_is_valid(iter); iter = db_shape_iter_get_next(iter))
{
   //Deselect all visible and invisible layers' shapes.
   if (db_is_selected(iter))
       db_select(iter, FALSE);
}
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_iter_include_protected_layers()

Function to include protected layers in the iteration, which is no longer necessary since by default, a shape iterator will iterate over shapes on protected and invisible layers.
If the iterator has been started (used to find a shape), then calling this function gives an AEL error.

See also: *db_create_shape_iter()* (ael), *db_shape_iter_get_next()* (ael), *db_shape_iter_exclude_invisible_layers()* (ael), *db_shape_iter_exclude_protected_layers()* (ael), *db_shape_iter_include_invisible_layers()* (ael).

Syntax:

```
db_shape_iter_include_protected_layers(shapeIter);
```

where
shapeIter is a shape iterator returned from a function such as *db_create_shape_iter()*.

Example:

```
//Example of how to deselect all selected shapes protected or not.
decl context = de_get_current_design_context();
decl iter = db_create_shape_iter(context);
//Iterate over all protected and unprotected layers' shapes.
iter = db_shape_iter_include_protected_layers(iter);
for ( ; db_shape_iter_is_valid(iter); iter = db_shape_iter_get_next(iter))
{
   //Deselect all protected and unprotected layers' shapes.
   if (db_is_selected(iter))
       db_select(iter, FALSE);
}
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_iter_is_valid()

This function returns TRUE if the shape iterator references a valid shape object and returns FALSE if the shape iterator references no valid shape object.

See also: *db_create_shape_iter()* (ael).

Syntax:

```
db_shape_iter_is_valid(iter);
```

where
iter is a shape iterator returned from a function such as *db_create_shape_iter()*.

Example:

```
decl context = de_get_current_design_context();
decl shapeIter = db_create_shape_iter(context);
// Select the first shape of the design context,
// if the design context returned a valid shape iterator.
if (db_shape_iter_is_valid(shapeIter) == TRUE)
   db_select(shapeIter);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_iter_limit_layer()

Function to limit the shape iteration to a layer.
If the iterator has been started (used to find a shape), then calling this function gives an AEL error.

See also: *db_create_shape_iter()* (ael), *db_shape_iter_get_next()* (ael), *db_shape_iter_include_invisible_layers()* (ael), *db_shape_iter_include_protected_layers()* (ael), *db_shape_iter_limit_purpose()* (ael), *db_shape_iter_limit_region()* (ael), *db_shape_iter_limit_selected()* (ael).

Syntax:

```
db_shape_iter_limit_layer(shapeIter, layer);
```

where
shapeIter is a shape iterator returned from a function such as *db_create_shape_iter()*.
layer is an integer representing the layer to limit the iteration to.

Example:

```
//Example of how to select all shapes on a particular layer.
decl context = de_get_current_design_context();
decl iter = db_create_shape_iter(context);
iter = db_shape_iter_limit_layer(iter, 8);
for ( ; db_shape_iter_is_valid(iter); iter = db_shape_iter_get_next(iter))
   db_select(iter, TRUE);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_iter_limit_purpose()

Function to limit the shape iteration to a purpose layer.
If the iterator has been started (used to find a shape), then calling this function gives an AEL error.

See also: *db_create_shape_iter()* (ael), *db_shape_iter_get_next()* (ael), *db_shape_iter_include_invisible_layers()* (ael), *db_shape_iter_include_protected_layers()* (ael), *db_shape_iter_limit_layer()* (ael), *db_shape_iter_limit_region()* (ael), *db_shape_iter_limit_selected()* (ael).

Syntax:

```
db_shape_iter_limit_purpose(shapeIter, purpose);
```

where
shapeIter is a shape iterator returned from a function such as *db_create_shape_iter()*.
purpose is an integer representing the purpose layer to limit the iteration to.

Example:

```
//Example of how to select all shapes on a particular purpose layer.
decl context = de_get_current_design_context();
decl iter = db_create_shape_iter(context);
iter = db_shape_iter_limit_purpose(iter, 3);
for ( ; db_shape_iter_is_valid(iter); iter = db_shape_iter_get_next(iter))
   db_select(iter, TRUE);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_iter_limit_region()

Function to limit the shape iteration to a region.
If the iterator has been started (used to find a shape), then calling this function gives an AEL error.

See also: *db_create_shape_iter()* (ael), *db_shape_iter_get_next()* (ael), *db_shape_iter_include_invisible_layers()* (ael), *db_shape_iter_include_protected_layers()* (ael), *db_shape_iter_limit_layer()* (ael), *db_shape_iter_limit_purpose()* (ael), *db_shape_iter_limit_selected()* (ael).

Syntax:

```
db_shape_iter_limit_region(shapeIter, x1, y1, x2, y2 [,allowIntersect]);
```

where
shapeIter is a shape iterator returned from a function such as *db_create_shape_iter()*.

x1,*y1*,*x2*,*y2* are coordinates in database units to limit the iteration to.
allowIntersect is optional. By default is true, meaning that the region only has to touch the shape, not contain it.

Example:

```
//Example of how to deselect all shapes in a particular region.
decl context = de_get_current_design_context();
decl iter = db_create_shape_iter(context);
iter = db_shape_iter_limit_region(iter, 0, 0, 200, 200);
for ( ; db_shape_iter_is_valid(iter); iter = db_shape_iter_get_next(iter))
   db_select(iter, FALSE);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_shape_iter_limit_selected()

Function to limit the iteration to selected shapes. Shapes are considered selected if either the entire shape is selected or if points on the shape are selected. Selected shapes on invisible or protected layers will be included in the iteration, even if db_shape_iter_include_protected_layers and db shape iter include protected layers are not called.
If the iterator has been started (used to find a shape), then calling this function gives an AEL error.

See also: *db_create_shape_iter()* (ael), *db_shape_iter_get_next()* (ael), *db_shape_iter_include_invisible_layers()* (ael), *db_shape_iter_include_protected_layers()* (ael),
db_shape_iter_limit_layer() (ael), *db_shape_iter_limit_purpose()* (ael), *db_shape_iter_limit_region()* (ael).

Syntax:

```
db_shape_iter_limit_selected(iter);
```

where
iter is a shape iterator returned from a function such as *db_create_shape_iter()*.

Example:

```
//Example of how to deselect all shapes:
decl context = de_get_current_design_context();
decl iter = db_create_shape_iter(context);
iter = db_shape_iter_limit_selected(iter);
for ( ; db_shape_iter_is_valid(iter); iter = db_shape_iter_get_next(iter))
   db_select(iter, FALSE);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

Simulation Functions

This section describes each Simulation function in detail.
The functions are listed in alphabetical order.

db_get_controller_design_name() (ael)
db_get_datadisp_name() (ael)
db_get_dataset_name() (ael)
db_get_hierarchy_policy_name() (ael)
db_get_opendds_option() (ael)
db_has_explicit_hierarchy_policy() (ael)
db_set_controller_design_name() (ael)
db_set_datadisp_name() (ael)
db_set_dataset_name() (ael)
db_set_hierarchy_policy_name() (ael)
db_set_opendds_option() (ael)
de_get_default_hierarchy_policy_name() (ael)
de_set_default_hierarchy_policy_name() (ael)

db_get_controller_design_name()

Returns the string design name for the controller design of the given design context. This function will return the design name of the given design context if no controller design has been specified for the given design context.
The design name string value returned from this function is in the format "<library name>:<cell name>:<view name>".

Syntax

```
controllerDesignName = db_get_controller_design_name(context);
```

Where

- *context* is the design context to get the controller design name for.

Example

```
decl context = de_get_design_context(winInst);
decl controllerDesignName = db_get_controller_design_name(context);
```

Return Value(s)

The string design name for the controller design of the given design context.

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_controller_design_name() (ael)
db_set_controller_design_name() (ael)

Where Used (ael)

Where Used (ael)

Schematic, Layout

db_get_datadisp_name()

Returns the string (DDS) Data Display file name for the given design context.
If no custom (DDS) Data Display name has been specified for the given design context, then ADS will use the design's cell name as the data display name.
If no custom data display name has been specified then this function will return the design's cell name.

Syntax

```
dataDisplayName = db_get_datadisp_name(context);
```

Where

- *context* is the design context to get the (DDS) datadisplay name for.

Example

```
decl context = de_get_design_context(winInst);
decl dataDisplayName = db_get_datadisp_name(context);
```

Return Value(s)

String name of the (DDS) Data Display file name for the given design context.

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_datadisp_name() (ael)
db_set_datadisp_name() (ael)

Where Used (ael)

Schematic, Layout

db_get_dataset_name()

Returns the string dataset name for the given design context.
If no custom dataset name has been specified for the given design context, then ADS will use the design's cell name as the data set name.
If no custom dataset name has been specified then this function will return the design's cell name.

Syntax

```
datasetName = db_get_dataset_name(context);
```

Where

- *context* is the design context to get the dataset name for.

Example

```
decl context = de_get_design_context(winInst);
decl dataSetName = db_get_dataset_name(context);
```

Return Value(s)

The string dataset name for the given design context.

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_dataset_name() (ael)
db_set_dataset_name() (ael)

Where Used (ael)

Schematic, Layout

db_get_hierarchy_policy_name()

Returns the string name of the HierarchyPolicy that will be used for simulation of the given design context.
Returns the name of the specified hierarchy policy set for the design, or the default policy name if there is no specific HierarchyPolicy set for the given design context.
To set a specific HierarchyPolicy for simulating a design, use the function *db_set_hierarchy_policy_name()* (ael).

Syntax

```
hierPolName = db_get_hierarchy_policy_name(context);
```

Where

- *context* is the design context to retrieve the active HierarchyPolicy name for.

Example

```
decl context = de_get_design_context(winInst);
decl hierPolname = db_get_hierarchy_policy_name(context);
```

Return Value(s)

The hierachy policy string name used for simulating the design.

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_hierarchy_policy_name() (ael)
db_has_explicit_hierarchy_policy() (ael)
db_set_hierarchy_policy_name() (ael)
de_get_default_hierarchy_policy_name() (ael)
de_set_default_hierarchy_policy_name() (ael)

Where Used (ael)

Schematic, Layout

db_get_opendds_option()

Returns TRUE if the given design context is specified to automatically open the Data Display after simulation. Returns FALSE otherwise.
This function will check for and return the value of the SIM_OPENDDS property on the given design context.
If there is no SIM_OPENDDS property found on the given design context, the function will next check for the setting from an OPEN_DDS_AFTER_SIM environment variable's value. If that environment variable's value is not set then this function will always return TRUE.

Syntax

```
openDDS = db_get_opendds_option(context);
```

Where

- *context* is the design context to get the "open data display automatically after simulation" setting from.

Example

```
decl context = de_get_design_context(winInst);
decl openDDS = db_get_opendds_option(context);
```

Return Value(s)

TRUE or FALSE

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_opendds_option() (ael)
db_set_opendds_option() (ael)

Where Used (ael)

Schematic, Layout

db_has_explicit_hierarchy_policy()

Returns TRUE if the given design context has a specific HierarchyPolicy name set that will be used for simulation of the design.
Returns FALSE if no hierarchy policy was specifically set for the design.
Use function *db_get_hierarchy_policy_name()* (ael) to get the name of the hierarchy policy used for simulation of a given design.
To set a specific HierarchyPolicy for simulating a design, use the function *db_set_hierarchy_policy_name()* (ael).

Syntax

```
isSpecificHierPolicySet = db_has_explicit_hierarchy_policy(context);
```

Where

- *context* is the design context to test if it has a specific HierarchyPolicy name set for it.

Example

```
decl context = de_get_design_context(winInst);
decl defaultPolicyName = de_get_default_hierarchy_policy_name();
decl hasSpecificPolicyName = db_has_explicit_hierarchy_policy (context);  // Is a specific policy
name for design set?
decl designSpecificPolicyName = “;
If (hasSpecificPolicyName)
{
   designSpecificPolicyname = db_get_hierarchy_policy_name(context);  // Get the specified policy
name for design.
}
```

Return Value(s)

TRUE or FALSE

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_hierarchy_policy_name() (ael)
db_has_explicit_hierarchy_policy() (ael)
db_set_hierarchy_policy_name() (ael)
de_get_default_hierarchy_policy_name() (ael)
de_set_default_hierarchy_policy_name() (ael)

Where Used (ael)

Schematic, Layout

db_set_controller_design_name()

Function to set the string design name of the controller design for the given design context.
The controller design name string value argument for this function should be in the format "<library name>:<cell name>:<view name>".

Syntax

```
db_set_controller_design_name(context, controllerDesignName);
```

Where

- *context* is the design context to set the controller design name for.
- *controllerDesignName* is a design name for the controller design.
  The design name string value argument should be in the format "<library name>:<cell name>:<view name>".

Example

```
decl context = de_get_design_context(winInst);
db_set_controller_design_name(context, controllerDesignName);
```

Return Value(s)

NULL

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_controller_design_name() (ael)
db_set_controller_design_name() (ael)

Where Used (ael)

Schematic, Layout

db_set_datadisp_name()

Function to specify the custom (DDS) Data Display name to use for the given design context.
If no custom Data Display name has been specified for the given design context, then ADS will use the design's cell name as the Data Display name.

Syntax

```
db_set_datadisp_name(context, dataDisplayName);
```

Where

- *context* is the design context to set the (DDS) datadisplay name for.

- *dataDisplayName* is the custom (DDS) Data display name to specify the given design context to use.

Example

```
decl context = de_get_design_context(winInst);
db_set_datadisp_name(context, dataDisplayName);
```

Return Value(s)

NULL

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_datadisp_name() (ael)
db_set_datadisp_name() (ael)

Where Used (ael)

Schematic, Layout

db_set_dataset_name()

Function to specify the custom dataset name to use for the given design context.
If no custom dataset name has been specified for the given design context, then ADS will use the design's cell name as the data set name.

Syntax

```
db_set_dataset_name(context, dataSetName);
```

Where

- *context* is the design context to set the dataset name for.
- *dataSetName* is the custom dataset name to specify the given design context to use.

Example

```
decl context = de_get_design_context(winInst);
db_set_dataset_name(context, datasetName);
```

Return Value(s)

NULL

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_dataset_name() (ael)
db_set_dataset_name() (ael)

Where Used (ael)

Schematic, Layout

db_set_hierarchy_policy_name()

Function to set the HierarchyPolicy name to use specifically for simulation of the given design context .

Syntax

```
db_set_hierarchy_policy_name(context, hierPolicyName);
```

Where

- *context* is the design context to set the HierarchyPolicy name for.
- *hierPolicyName* is the HierarchyPolicy string name to set the design to use.

Example

```
decl context = de_get_design_context(winInst);
db_set_hierarchy_policy_name(context, hierPolicyName);
```

Return Value(s)

NULL

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_hierarchy_policy_name() (ael)
db_has_explicit_hierarchy_policy() (ael)
db_set_hierarchy_policy_name() (ael)
de_get_default_hierarchy_policy_name() (ael)
de_set_default_hierarchy_policy_name() (ael)

Where Used (ael)

Schematic, Layout

db_set_opendds_option()

Function to specify for the given design if the Data Display window should open automatically after simulation of the design occurs.

Syntax

```
db_set_opendds_option(context, openDDS);
```

Where

- *context* is the design context to set the "open data display automatically after simulation" setting for.
- *openDDS* is boolean set as TRUE or FALSE.

Example

```
decl context = de_get_design_context(winInst);
// Do not invoke the Data Display automatically after simulation.
db_set_opendds_option(context, FALSE);
```

Return Value(s)

NULL

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_opendds_option() (ael)
db_set_opendds_option() (ael)

Where Used (ael)

Schematic, Layout

de_get_default_hierarchy_policy_name()

Returns the name of the default HierarchyPolicy that is used for simulating designs.
The default HierarchyPolicy is the policy used for simulation if no specific HierarchyPolicy has been specified for the top design.
To set a specific HierarchyPolicy for simulating a design, use the function *db_set_hierarchy_policy_name()* (ael).

Syntax

```
hierDefaultPolicyName = de_get_default_hierarchy_policy_name();
```

Example

```
decl hierDefaultPolicyName = de_get_default_hierarchy_policy_name();
```

Return Value(s)

The default hierachy policy string name used for simulating designs.

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_hierarchy_policy_name() (ael)
db_has_explicit_hierarchy_policy() (ael)
db_set_hierarchy_policy_name() (ael)
de_get_default_hierarchy_policy_name() (ael)
de_set_default_hierarchy_policy_name() (ael)

Where Used (ael)

Schematic, Layout

de_set_default_hierarchy_policy_name()

Sets the name of the default HierarchyPolicy that is used for simulating designs.
The default HierarchyPolicy is the policy used for simulation if no specific HierarchyPolicy has been specified for the top design.
To set a specific HierarchyPolicy for simulating a design, use the function *db_set_hierarchy_policy_name()* (ael).

Syntax

```
de_set_default_hierarchy_policy_name(hierDefaultPolicyName);
```

Where

- *hierDefaultPolicyName* is the default HierarchyPolicy string name to set designs to use.

Example

```
de_set_default_hierarchy_policy_name(hierPolicyName);
```

Return Value(s)

NULL

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

db_get_hierarchy_policy_name() (ael)
db_has_explicit_hierarchy_policy() (ael)
db_set_hierarchy_policy_name() (ael)
de_get_default_hierarchy_policy_name() (ael)
de_set_default_hierarchy_policy_name() (ael)

***Where Used* (ael)**

Schematic, Layout

Simulator Command Functions

This section describes each Simulator Command function in detail. The functions are listed in alphabetical order.

clear_server()

Clears the busy status state of the active server (simulator). This command helps to clear the server after a communications error. Returns: none.

Syntax:

```
clear_server(server_handle);
```

where

server_handle is a handle to simulator/server.

Example:

```
decl serverHandle;
clear_server(serverHandle);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

create_server()

Creates a handle to a server process that can be controlled through AEL. If the server process is not currently running, the server process is spawned automatically when it is sent a message. That is, you do not need to invoke the server process explicitly, but can simply start sending commands and data to it through AEL and you can be assured the process will be up and running. The server process receives commands and data and returns information through AEL.

Typically the server program is an EEsof program that communicates dynamically through a special message protocol, but it can be an ordinary program communicating using stdin

and stdout. For non-EEsof programs, every data message sent to the server is received by the program as a line to stdin and every line printed to stdout by the program is returned as a data message. Returns the handle of the server.

See also: *send_server_command()* (ael), *send_server_data()* (ael), *send_server_interrupt()* (ael), *send_server_kill()* (ael), *start_server()* (ael), *push_message_handler()* (ael), *pop_message_handler()* (ael).

Syntax:

```
create_server( name, label, hostMachine [, params, nonEmx, paramAelFunc,
createUnique, fReinstate]);
```

where

name is the name of the server program.

label is the description of program.

hostMachine is the name of host machine.

params is optional; the parameter string to be passed to program when spawned.

nonEmx is optional; the flag indicating whether server will be non-EEsof program, which cannot take advantage of special communications protocol, where:

- 0 = (default) for EEsof programs
- 1 = non-EEsof program

paramAelFunc is optional; the name of an AEL function which can be used to generate information to be passed to server when it is spawned. The name of the AEL function is provided as a string (enclosed in quotes). The function will be passed the original parameter string, and should return a new parameter string.

createUnique is optional; the flag indicating that more than 1 server process with the same name can be invoked: 0 = (default) for returning existing server with the same name, and 1 = create another server process even if one with the same name is already running
fReinstate is optional; the flag indicating whether to notify other processes that a new server process has been invoked, where:

- 0 = (default) Invoke a new server process and do not notify the other processes in the session
- 1 = Invoke a new server process and notify other processes in the session

If the server process was spawned previously, but has died since, the reinstate flag indicates whether or not other processes in the session get notified of the newly-spawned server process. Notification ensures that the newly-spawned server will be able to communicate with these processes.

Example:

```
decl surHandle=create_server("myserver", "MyServerProcess", "");
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

de_analyze()

Invokes the simulator and sends it a new, updated netlist of the current design. Options specified in the simulation setup dialog will be used for the simulation. Returns: none.

Syntax:

```
de_analyze();
```

Example:

```
de_analyze();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Batch Simulation
Design has Schematic?

Where Used: **(ael)**

Simulation

de_analyze_tune()

Invokes the simulator, sends the simulator a new, updated netlist of the current design, and prepares the simulator to receive de_tune() commands. Options specified in the simulation setup dialog are used for the simulation. Returns: none.

See also: *de_tune()* (ael), *de_tune_deinit()* (ael).

Syntax:

```
de_analyze_tune();
```

Example:

```
de_analyze_tune();
```

***Where Used:* (ael)**

Simulation

de_close_all_datadisplay()

Hides all data display windows that have been opened. Returns: none.

See also: *de_restore_all_datadisplay()* (ael).

Syntax:

```
de_close_all datadisplay();
```

Example:

```
de_close_all_datadisplay();
```

***Where Used:* (ael)**

Schematic

de_open_datadisplay()

Opens a saved data display file. Sends the *OPEN_FILE* command, with the name of the file to load, to the data display server. Returns: none.

See also: *de_new_datadisplay()* (ael).

Syntax:

```
de_open_datadisplay(fileName[, addcwd]);
```

where

fileName is the name of the saved graphics state file.

addcwd is the flag to indicate whether the current working directory should be prepended to the fileName.

Example:

```
de_open_datadisplay("mydata");
```

***Where Used:* (ael)**

Schematic

de_restore_all_datadisplay()

Opens all data display windows that have been hidden, using *de_close_all_datadisplay()*. Returns: none.

See also: *de_close_all_datadisplay()* (ael).

Syntax:

```
de_restore_all_datadisplay();
```

Example:

```
de_open_datadisplay("usr/test_wrk/plot1.dds");
de_open_datadisplay("usr/test_wrk/plot2.dds");
de_close_all_datadisplay();
de_restore_all_datadisplay();
```

Where Used: **(ael)**

Schematic

de_restore_status()

Opens the closed status server window. Returns: none.

Syntax:

```
de_restore_status();
```

Example:

```
de_restore_status();
```

Where Used: **(ael)**

Schematic

de_tune()

Activates a tune analysis after updating parameters to the values specified in the paramList. The function de_analyze_tune() must be called (once) to set up the conditions for tuning before the function de_tune() is called. Options specified in the simulation setup dialog are used for the simulation. Returns: none.

See also: *de_analyze_tune()* (ael), *de_tune_deinit()* (ael).

Syntax:

```
de_tune(paramList);
```

where

paramList is a list of parameter names and values. The parameter name and value pairs are separated by the bar character (|), and the list is terminated by a bar.

Example:

```
de_tune("C2.C 5.131 pF|SRC3.Vdc 1.168 V|");
```

Where Used: **(ael)**

Simulation

de_tune_deinit()

Terminates the tuning operation in the simulator. This function should be called after all tuning analyses have been performed. Returns: none.

See also: *de_analyze_tune()* (ael), *de_tune()* (ael).

Syntax:

```
de_tune_deinit();
```

Example:

```
de_tune_deinit();
```

Where Used: **(ael)**

Simulation

de_update_optimization_values()

Requests updated optimization/yield results from any optimization or yield analysis performed by the current server (simulator). All designs referenced by the DUT are updated. Returns: none.

Syntax:

```
de_update_optimization_values();
```

Example:

```
de_update_optimization_values();
```

***Where Used:* (ael)**

Schematic

invoke_process()

Function to invoke a child process.
Returns TRUE or FALSE depending on whether the process is started, and completes successfully.
For a process that blocks, a return value of TRUE indicates that the process started and completed successfully.
For a non-blocking process, a return value of TRUE indicates that the process started, but does not indicate anything about completion status.

Syntax:

```
invoke_process(process, argList, workingDir, block, showDialog, dialogMessage);
```

where

process is a string that specifies the process name.

argList is a list of string arguments to pass to the process.
Note: only list elements that are strings will be used as arguments, all other element types will be ignored.

workingDir is a string that specifies the working directory to use.

block is a boolean, TRUE means the current process should block and wait for the process to exit, FALSE means the

current process should not be blocked.

showDialog is a boolean, if block is TRUE, TRUE means that a cancel dialog should be shown so the process can be

killed, FALSE means no dialog should be shown (basically enabling non-graphical non-interactive processing).

dialogMessage is a string to display in the dialog if showDialog is TRUE.

Example:

```
// Start notepad with argument "my_file.txt", there will be a cancel dialog that blocks the ADS
application.
decl bOk= invoke_process("notepad.exe",list("my_file.txt"),"",TRUE,TRUE, "");
// Start notepad with argument "my_file.txt", the ADS application is blocked until notepad is
closed.
bOk= invoke_process("notepad.exe",list("my_file.txt"),"",TRUE,FALSE, "");
// Start notepad, the ADS application is not blocked.
bOk= invoke_process("notepad.exe",list(),"",FALSE,TRUE, "");
// Shows an error that the process can't be started.
bOk= invoke_process("noteblah.exe",list(),"",TRUE,TRUE, "");
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout, Symbol, Main

pop_message_handler()

Restores a saved message handler for a server to the state previous to the last call to *push_message_handler()*. See also *push_message_handler()*. Returns: none.

See also: *push_message_handler()* (ael), *create_server()* (ael).

Syntax:

```
pop_message_handler( server_handle );
```

where

server_handle is a handle of simulator/server returned by *create_server()*.

Example:

```
decl sh;
pop_message_handler( sh );
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

push_message_handler()

Installs a message handler function to receive messages from a server. When a message is received from the server, the handler function is called and passed the server handle, type code for the message, and the message text. The message type codes are:

0 = command
1 = data
2 = begin data block
3 = continue data block
4 = end data block
5 = done (server acknowledge of command)
6 = error
7 = server died

For non-EEsof server programs, only data messages may be exchanged. Returns: none.

See also: *pop_message_handler()* (ael), *create_server()* (ael).

Syntax:

```
push_message_handler( messageFn, server_handle);
```

where

messageFn is the name of an AEL function to receive messages from server.

server_handle is a handle of simulator/server returned by *create_server()*.

Example:

```
decl my_server=create_server("myserver", "MyServerProcess", "");
defun print_message( sh, type, msg )
{
if (type == 0)
 fputs( stderr, strcat( "Command:",msg ));
if (type == 1)
 fputs( stderr, strcat( "Data:",msg ));
if (type == 2)
 fputs( stderr, strcat( "Start data:",msg ));
if (type == 3)
 fputs( stderr, strcat( "Cont data:",msg ));
if (type == 4)
 fputs( stderr, strcat( "End data:",msg ));
if (type == 5)
 fputs( stderr, strcat( "Done:",msg ));
if (type == 6)
 fputs( stderr, strcat( "Error:",msg ));
if (type == 7)
 fputs( stderr, strcat( "Server died:",msg ));
}
push_message_handler( print_message, my_server );
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

send_server_command()

Sends a command to the server identified by server_handle. Prefix with the CMD command protocol statement. Returns: none.

See also: *create_server()* (ael).

Syntax:

```
send_server_command(command, server_handle, [wait, invokeDir]);
```

where

command is the command string to send to simulator.

server_handle is a handle to simulator/server, returned by *create_server()*.

wait is optional; the flag indicating whether the function should return immediately or wait for the "DONE" message from the server, where:

- 0 = (default) for immediate return
- 1 = wait for "DONE" message

invokeDir is optional; the directory from which to invoke the server if necessary, where:

- NULL = (default) for using current working directory
- DirName = for using the given directory

Example:

```
decl sh = create_server("myServer", "myServerProcess", "");
send_server_command("MAP_WIN", sh, 1);
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

send_server_data()

Sends data statement to the server identified by server_handle. Prefix data with DATA statement. Returns: none.

See also: *create_server()* (ael).

Syntax:

```
send_server_data(dataStatement, server_handle [, invokeDir]);
```

where

dataStatement is the data statement to send to the simulator.

server_handle is a handle to simulator/server, returned by *create_server()*.

invokeDir is optional; the directory in which to invoke the server, if necessary, where:

- NULL = (default) for using current working directory
- DirName = for using the given directory

Example:

```
decl sh = create_server("myServer", "myServerProcess", "");
send_server_data("new data", sh);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

send_server_interrupt()

Sends an interrupt to the current server identified by server_handle. Returns: none.

See also: *create_server()* (ael).

Syntax:

```
send_server_interrupt(server_handle[, message]);
```

where

server_handle is a handle to simulator/server, returned by *create_server()*.

invokeDir is optional; an interrupt message.

Example:

```
decl sh = create_server("myServer", "myServerProcess", "");
send_server_interrupt(sh);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

send_server_kill()

Sends the special kill command ("KILL") to the server identified by server_handle. Returns: none.

See also: *create_server()* (ael).

Syntax:

send_server_kill(server_handle);

where

server_handle is a handle to simulator/server returned by *create_server()*.

Example:

```
decl sh = create_server("myServer", "myServerProcess", "");
send_server_kill(sh);
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

server_running()

Returns the status of the server identified by serverHandle, where: 1 (True) = the server identified by serverHandle is running, and 0 (False) = the server identified by serverHandle is not running.

See also: *create_server()* (ael).

Syntax:

```
server_running(serverHandle);
```

where

serverHandle is the pointer for specific server to be evaluated, returned by *create_server()*.

Example:

```
decl running, SHandle=create_server("myserver", "MyServerProcess", "");
running = server_running(SHandle);
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

start_server()

Causes the server identified by serverHandle to be spawned. Returns: none.

See also: *create_server()* (ael).

Syntax:

```
start_server( server_handle [, mapStderr, invokeDir]);
```

where

server_handle is a handle of simulator/server, returned by *create_server()*.

mapStderr is optional; sets stderr remap, where:

- 0 = (default) Do not remap stderr.
- 1 = Remap stderr to a pipe. A line written to stderr is returned as a DATA message.

invokeDir is optional; the directory in which to invoke the server, if necessary:

- NULL = (default) for using current working directory
- DirName = for using the given directory

Example:

```
decl hpeesofsim = create_server("simulator", "HPEESOF simulator", "");
start_server( hpeesofsim);
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

String Functions

This section describes each String function in detail. The functions are listed in alphabetical order.

fmt()

Converts a float to a string. Returns a string in which the given value is converted to an ASCII string according to the width and precision specified.

Syntax:

```
fmt(realNum [, width, precision]);
```

where

realNum is the real number to convert.

width is optional; default = 10. The string width (number of characters in converted string). Must be equal to or larger than the total size of the resulting string plus a string termination character.

precision is optional; default = 6. The number of characters to the right of the decimal point.

Example:

The example returns the string "10.134599".

```
decl str;
str = fmt(10.134599);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

fmt_tokens()

Creates a string from a list of tokens. Concatenates the list items into a single string. This

is a convenient method for printing out the contents of a list. Returns: A string, the concatenation of the list items.

Syntax:

```
fmt_tokens(list);
```

where

list is a list of items to format into a string.

Example:

```
warning(errclass, 20, fmt_tokens(list(elem, "ANG not valid", ang)));
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
EEsof Part List - Modified
Netlist with Node Names

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

index()

Returns the position of the first occurrence of a string or character in a string; such as, returns the position of the first occurrence of str2 in str1 (where, n is an integer >= 0, where 0 is the first character and −1 = is not found).

Syntax:

```
index(str1, str2);
```

where

str1 is a string.

str2 is a string to search for.

Example:

```
decl pos;
pos = index("hello", "e");      //returns 1
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Calculate Equations
Insert Equations from File
Netlist with Node Names

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name index() is used for more than one type of expression. For comparison, if using *Advanced Design System* see **Simulator Expressions** > **Data Access Functions For Simulator Expressions** > **index() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

leftstr()

Returns the left portion of a string, truncating or padding on the right with blanks if necessary, until the length specified is obtained.

Syntax:

```
leftstr(string, length);
```

where

str is the string.

length is an integer, length to truncate or pad.

Example:

The example creates the string "hel".

```
decl nstr;
nstr = leftstr("hello", 3);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Get File Size
Is This UNIX?
Replace String

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

midstr()

Returns a piece of a string (the dissected string), starting and ending at the given indices. An end index of −1 indicates the end of the string. If the start index is also negative, it indicates a count from the end of the string.

Syntax:

```
midstr(str, startIndex, endIndex);
```

where

str is the string to operate on.

startIndex is the starting position.

endIndex is the ending position.

Example:

The example returns the string "ll".

```
decl mstr;
mstr = midstr("hello", 2,3);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Strip Alpha
Substring Character Position
Short File Name

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

parse()

Parses a string into tokens, where each token is delimited by a blank, tab, or operator. The following interpretations are made:

- Alphabetic characters include underscore (_) and dollar sign ($) as well as uppercase A through Z, lowercase a through z, and numbers 0 through 9.
- Numbers are interpreted as expected without separation into tokens because of sign or exponent notation.
- The characters period and E (or e) are interpreted as expected in the context of a number.
- A number followed by a string is interpreted as two tokens: the first numeric and the second a string.
- A string followed by a number (without an intervening operator or delimiting punctuation) is interpreted as a single string token.
- Operators are interpreted as string tokens and adjacent operators are merged into a single token. The other normal delimiters are blank and tab, which are treated as white space and are ignored except to separate tokens.

Special handling is provided for strings enclosed in double quotes, where the entire string (including the quotes) is interpreted as a single token. However, there is no provision for handling strings with embedded quotes.

If the optional delimiter and operator strings are used, stricter parsing rules are enabled. Only the listed characters will cause separation of the text into tokens, and these will always cause separation into tokens, regardless of the context. Delimiters will be discarded but will cause separation of tokens, while operators will be returned as single character strings, causing the special handling for real numbers, operators and strings to be unavailable when the delimiter or operator strings are specified. Returns: A list constructed from the tokens. A token may be returned as an integer, real, or string member in the list.

See also: *list()* (ael).

Syntax:

```
parse(text[, delim][, ops][, str]));
```

where

text is the string of text to be parsed.

delim is optional. String of delimiter characters. Recognizes normal arithmetic operators unless alternate characters are given in delimiter or special character strings. Default delimiters are space and tab.

ops is optional. String of special operator characters, each to be interpreted as

an individual token. Default special operators are:
+ - * / = , ~ & ^ < > ! # % ' () : ; ? @ [] \ ` {} '

str is optional. Specifies to only return strings, where:

- FALSE = (default) each element is appropriately converted to int, real, or strings
- TRUE = returns a list of strings, no conversion takes place

Example:

The example returns a list containing "hello", 1, 7.8, "*", and 32.6.

```
decl mylist;
mylist = parse("hello 1 7.8 * 32.6");
```

ALSO

```
parse("123.456", ".", NULL); // returns list(123,456)
parse("123.456", ".", NULL, TRUE); // returns list("123","456")
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Get File Permissions
Get File Size
Get File Time Stamp

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

parse_blank()

Parses a string of tokens separated by spaces or tabs into a list of strings. Returns a list of tokens.

Syntax:

```
parse_blank(str);
```

where

str a character string to be parsed.

Example:

```
decl a;
a = parse_blank("1.5  2.3  4.1");
```
This example creates the same list:
```
a = list("1.5", "2.3", "4.1");
```

***Where Used:* (ael)**

Schematic

rightstr()

Returns the right portion of a string value, truncating or padding on the left with blanks if necessary until the length specified is obtained.

Syntax:

```
rightstr(str, length);
```

where

str is the string to truncate or pad.

length is the new string length.

Example:

```
decl mystr;
mystr = rightstr("hello", 2);      //creates the string "lo"
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Add Extension to File Name
Change Component Scope to GLOBAL
Replace String

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

sprintf()

Format text values into a string. This function works like *fprintf()* (ael) except that the output is formatted into a string rather than a file. The same format specifications are valid for the format string. Returns a formatted string.

See also: *fprintf()* (ael).

Syntax:

```
sprintf(fmt, args...);
```

where

fmt is output format string.

args is the values to be included in the formatted output, as required by the format string.

Example:

Creates and stores the following string in the variable *s* : *The value of x is 5.5*.

```
decl x=5.5;
decl s;
s=sprintf ("The value of %s is %f", "x", x);
```

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name sprintf() is used for more than one type of expression. For comparison, if using *Advanced Design System* see **Simulator Expressions** > **Utility Functions for Simulator Expressions** > **index() Expression**. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

strcasecmp()

Compares two strings, ignoring differences in case between them. If the first and second strings are equivalent (except for case differences), this function returns the integer zero. If the first string is higher than the second in the ASCII collating sequence, this function will return an integer greater than zero. If the first string is lower than the second in the collating sequence, this function returns an integer less than zero.

See also: *strcmp()* (ael).

Syntax:

```
strcasecmp(str1, str2);
```

where

str1 is the first string to compare.

str2 is the second string to compare.

Example:

```
fputs(stderr, strcasecmp("A", "a"));
// will print 0
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

strcat()

Appends two or more strings to the end of the first, resulting in a single string. Returns a new string which is composed of the second string (and possibly additional strings) appended to the end of the first string. Maximum of 32 input arguments. If any argument is not a string or real number, the function fails and reports an error. At least one string entry is required. Real numbers do not require quotes. There is no limit on quantity of parameters for strings, integers or real numbers.

See also: *leftstr()* (ael), *midstr()* (ael), *rightstr()* (ael).

Syntax:

```
strcat(text1, text2 ...);
```

where

text1 is the first string to concatenate.

text2 is the second string, optionally followed by additional strings separated by commas.

Example:

```
decl str;
```

```
str = strcat("hello", "there"); // returns "hello there"
x = strcat("des",1); // returns "des1"
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Copy String
Design has Layout?
Get File Size

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Note
The function name strcat() is used for more than one type of expression. If using *Advanced Design System* see **Simulator Expressions** > **Utility Functions for Simulator Expressions** > **strcat() Expression** for comparison. For more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

strcmp()

Compares two strings. If the first and second strings are the same, this function returns the integer zero. If the first string is higher than the second in the ASCII collating sequence, this function returns an integer greater than zero. If the first string is lower than the second in the collating sequence, this function returns an integer less than zero. Returns an integer indicating the results.

See also: *strcasecmp()* (ael).

Syntax:

```
strcmp(str1, str2);
```

where

str1 is the first string to compare.

str2 is the second string to compare.

Example:

```
fputs(stderr, strcmp("a", "b"));
// will print -1
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

stripstr()

Returns a string with leading and trailing blanks and tabs removed.

Syntax:

```
stripstr(str);
```

where

str is the string to strip.

Example:

```
decl sstr;
sstr = stripstr("    hi there"); // returns "hi there"
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Get Meas Equation Variable String
Set Layer File - Traverses Hierarchy
Trim off Leading/Trailing Spaces

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

strlen()

Returns the length of a string, as an integer.

Syntax:

```
strlen(str);
```

where

str is a string.

Example:

```
decl len;
len = strlen("hi"); // returns 2
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Copy String
Replace String
Strip Alpha

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

tolower()

Converts a string to lowercase and return the converted string. If an integer having the value of an ASCII character is passed, then the integer is converted to represent the lowercase value for the ASCII character and returned. Returns a string (if a string is passed) or an integer (if an integer is passed).

See also: *toupper()* (ael), *strcasecmp()* (ael).

Syntax:

```
tolower(str);
```

where

str is a string to convert.

Example:

```
fputs(stderr, tolower("aBcD"));
// will print "abcd"
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Add Extension to File Name
Read Configuration File

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

toupper()

Converts a string to uppercase and return the converted string. If an integer having the value of an ASCII character is passed, then the integer is converted to represent the uppercase value for the ASCII character and returned. Returns a string (if a string is passed) or an integer (if an integer is passed).

See also: *tolower()* (ael), *strcasecmp()* (ael).

Syntax:

```
toupper(str);
```

where

str is a string to convert.

Example:

```
fputs(stderr, toupper("aBcD"));
// will print "ABCD"
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Is Sub String?
Read Configuration File
Substring Character Position

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic,

Layout, Simulation, GUI

val()

Returns a real number, the numeric value of an ASCII string.

Syntax:

val(string);

where

string is an ASCII string, representing a numeric value.

Example:

```
decl v;
v = 10 + val("2.5");
v = 12.5;
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Get File Size
Polylines to Polygons
Strip Alpha

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

Term Functions

- *Term Overview* (ael)
- *Connectivity Objects Overview* (ael)

This section describes the following Term Functions in detail:

- *db_get_term_name()* (ael)
- *db_get_term_net()* (ael)
- *db_get_term_number()* (ael)
- *db_get_term_type()* (ael)

See also

Term Iterator Functions (ael)

db_get_term_name()

Returns the name of a given *terminal* (ael).

Syntax

```
decl termName = db_get_term_name(dbTerm);
```

Where,

- *dbTerm* is a *Term* (ael) object or Term iterator.

Example

```
decl termIter = db_create_term_iter(context);
if (db_term_iter_is_valid(termIter))
{
   for (; db_term_iter_is_valid(termIter);
        termIter = db_term_iter_get_next(termIter))
   {
        // Get terminal name
        decl termName = db_get_term_name(termIter);
        …
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Term Overview (ael)
Term Functions (ael)
Term Iterator Functions (ael)
db_get_term_number() (ael)
db_get_term_type() (ael)

Where Used (ael)

Schematic, Layout

db_get_term_net()

Returns the *Net* (ael) of a given *terminal* (ael).

Syntax

```
decl dbNet = db_get_term_net(dbTerm);
```

Where,
dbTerm is a *Term* (ael) object or Term iterator.

Example

```
decl termIter = db_create_term_iter(context);
if (db_term_iter_is_valid(termIter))
{
   for (; db_term_iter_is_valid(termIter);
        termIter = db_term_iter_get_next(termIter))
   {
        //Get the net name and terminal name of the terminal.

        //Get net of the terminal
        decl dbNet = db_get_term_net(termIter);
        //Get the net name of the Net of the terminal
        decl netName = db_get_net_name(dbNet);
        //Get terminal name of the terminal
        decl termName = db_get_term_name(termIter);
        …
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Term Overview (ael)
Term Functions (ael)
Term Iterator Functions (ael)
db_get_term_name() (ael)
db_get_term_number() (ael)
db_get_term_type() (ael)

Where Used (ael)

Schematic, Layout

db_get_term_number()

Returns the number of a given *terminal* (ael).

Syntax

```
decl termNum = db_get_term_number(dbTerm);
```

Where,

- *dbTerm* is a *Term* (ael) object or Term iterator.

Example

```
decl termIter = db_create_term_iter(context);
if (db_term_iter_is_valid(termIter))
{
   for (; db_term_iter_is_valid(termIter);
        termIter = db_term_iter_get_next(termIter))
   {
        // Get terminal number
        decl termNum = db_get_term_number(termIter);
        …
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Term Overview (ael)
Term Functions (ael)
Term Iterator Functions (ael)
db_get_term_name() (ael)
db_get_term_type() (ael)

Where Used (ael)

Schematic, Layout

db_get_term_type()

Returns an integer terminal type of a given *Term* (ael) object. The terminal type describes the direction of the *Term* (ael).

The possible returned integer *terminal* (ael) types are :

- **INPUT_PIN** = 0, represents an input *terminal* (ael).
- **OUTPUT_PIN** = 1, represents an output *terminal* (ael).
- **IN_OUT_PIN** = 2, represents an input-output *terminal* (ael).

Syntax

```
decl termType = db_get_term_type(dbTerm)
```

Where,

- *dbTerm* is an *Term* (ael) object or a term iterator

Example

```
decl termIter = db_create_term_iter(context);
for(; db_term_iter_is_valid(termIter);
   termIter = db_term_iter_get_next(termIter))
{
   decl termType = db_get_term_type(termIter);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Term Overview (ael)
Term Functions (ael)
Term Iterator Functions (ael)
db_get_term_name() (ael)
db_get_term_number() (ael)

Where Used (ael)

Schematic, Layout

Term Iterator Functions

- *Term Overview* (ael)
- *Connectivity Objects Overview* (ael)

This section describes the following Term Iterator Functions in detail:

- *db_create_term_iter()* (ael)
- *db_term_iter_is_valid()* (ael)
- *db_term_iter_get_next()* (ael)
- *db_term_iter_get_term()* (ael)

See also

Term Functions (ael)

db_create_term_iter()

This function returns an iterator to the first *Term* (ael) belonging to a given *DesignContext* (ael) or given *Net* (ael). If no *terminals* (ael) exist for the given *Net* (ael) or *DesignContext* (ael), then the *Term* (ael) iterator that is returned will be invalid. The function *db_term_iter_is_valid()* (ael) can be used to test if a *Term* (ael) iterator is valid.

Syntax

```
decl termIter = db_create_term_iter(object);
```

Where,

- *object* is an *Net* (ael) object or net iterator, or is a *DesignContext* (ael).

Example

```
decl termIter = db_create_term_iter(dbNet);
for (; db_term_iter_is_valid(termIter);
   termIter = db_term_iter_get_next(termIter))
{
   decl dbTermH = db_get_term_iter_get_term(termIter);
   //Get the terminal number.
   decl termNum = db_get_term_number(dbTermH);
   ...
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Term Overview (ael)
Term Functions (ael)
Term Iterator Functions (ael)
db_term_iter_is_valid() (ael)
db_term_iter_get_next() (ael)

Where Used (ael)

Schematic, Layout

db_term_iter_get_next()

This function returns the next *Term* (ael) iterator of the given *Term* (ael) iterator.

Syntax

```
decl nextTermIter = db_term_iter_get_next(termIter);
```

Where,

- *termIter* is a valid *Term* (ael) iterator.

Example

```
decl termIter = db_create_term_iter(dbNet);
for (; db_term_iter_is_valid(termIter);
   termIter = db_term_iter_get_next(termIter))
{
   decl dbTermH = db_get_term_iter_get_term(termIter);
   //Get the terminal number.
   decl termNum = db_get_term_number(dbTermH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Term Overview (ael)
Term Functions (ael)
Term Iterator Functions (ael)
db_create_term_iter() (ael)
db_term_iter_is_valid() (ael)

Where Used (ael)

Schematic, Layout

db_term_iter_get_term()

This function returns the *terminal* (ael) object that is referenced by the given *Term* (ael) iterator

Syntax

```
decl dbTerm = db_term_iter_get_term(termIter);
```

Where,

- *termIter* is a valid *Term* (ael) iterator

Example

```
decl termIter = db_create_term_iter(dbNet);
for (; db_term_iter_is_valid(termIter);
   termIter = db_term_iter_get_next(termIter))
{
   decl dbTermH = db_get_term_iter_get_term(termIter);
   //Get the terminal number.
   decl termNum = db_get_term_number(dbTermH);
   …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Term Overview (ael)
Term Functions (ael)
Term Iterator Functions (ael)
db_create_term_iter() (ael)
db_term_iter_is_valid() (ael)
db_term_iter_get_next() (ael)

Where Used (ael)

Schematic, Layout

db_term_iter_is_valid()

This function returns TRUE if the given *Term* (ael) iterator is referencing a valid *terminal* (ael) object.

Syntax

```
decl isValid = db_term_iter_is_valid(termIter);
```

Where,

- *termIter* is a *Term* (ael) iterator.

Example

```
decl termIter = db_create_term_iter(dbNet);
for (; db_term_iter_is_valid(termIter);
   termIter = db_term_iter_get_next(termIter))
{
   decl dbTermH = db_get_term_iter_get_term(termIter);
   //Get the terminal number.
```

```
    decl termNum = db_get_term_number(dbTermH);
    …
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

Connectivity Objects Overview (ael)
Term Overview (ael)
Term Functions (ael)
Term Iterator Functions (ael)
db_create_term_iter() (ael)
db_term_iter_get_next() (ael)

Where Used (ael)

Schematic, Layout

Transaction Functions

This section describes each transaction function in detail. The functions are listed in alphabetical order.

db_commit_transaction()

Commit the given transaction.

See also: *db_create_transaction()* (ael), *db_rollback_transaction()* (ael), *db_transaction_is_empty()* (ael).

Syntax:

```
db_commit_transaction(transaction);
```

where
transaction is a transaction returned from a function such as *db_create_transaction()*.

Example:

```
decl context = de_get_current_design_context();
// Start a transaction.
decl transH = db_create_transaction(context, "Add Construction Lines");
// Add a horizontal construction line.
de_add_construction_line(0, 0, 100, 0);
// Add a diagonal construction line.
de_add_construction_line(0, 0, 100, 100);
// End a transaction. The transaction is now on the undo/redo list.
db_commit_transaction(transH);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_create_transaction()

Create a new transaction object for a given DesignContext. The transaction starts at the time of creation. All operations incurring inside the transaction, meaning starting after the transaction is started and before the transaction is commited, will be undoable and/or redoable in a single step. Transactions are useful for adding a number of commands and/or changes to the undo/redo stack that can be undone/redone in a single step at a later moment. The transaction merely groups the operations into one easy step that can be cancelled by rolling the transaction back.

See also: *db_commit_transaction()* (ael), *db_rollback_transaction()* (ael), *db_transaction_is_empty()* (ael).

Syntax:

```
db_create_transaction(context, descriptionStr);
```

where
context is a DesignContext returned from a function such as *de_get_current_design_context()*.
descriptionStr is a string that describes the transaction.

Example:

```
decl context = de_get_current_design_context();
// Start a transaction.
decl transH = db_create_transaction(context, "Add Construction Lines");
// Add a horizontal construction line.
de_add_construction_line(0, 0, 100, 0);
// Add a diagonal construction line.
de_add_construction_line(0, 0, 100, 100);
// End a transaction. The transaction is now on the undo/redo list.
db_commit_transaction(transH);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_rollback_transaction()

Rollback the given transaction. Undo all changes since the start of the transaction.

See also: *db_create_transaction()* (ael), *db_commit_transaction()* (ael), *db_transaction_is_empty()* (ael).

Syntax:

```
db_rollback_transaction(transaction);
```

where
transaction is a transaction returned from a function such as *db_create_transaction()*.

Example:

```
decl context = de_get_current_design_context();
decl transH = db_create_transaction(context, "Add Construction Lines");
// Add a horizontal construction line.
de_add_construction_line(0, 0, 100, 0);
// Add a diagonal construction line.
de_add_construction_line(0, 0, 100, 100);
// Undo the transaction.
db_rollback_transaction(transH);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

db_transaction_is_empty()

Returns TRUE if no changes have occured since the start of the transaction. Returns FALSE if changes have occured since the start of the transaction.

See also: *db_create_transaction()* (ael), *db_commit_transaction()* (ael), *db_rollback_transaction()* (ael).

Syntax:

```
db_transaction_is_empty(transaction);
```

where
transaction is a transaction returned from a function such as *db_create_transaction()*.

Example:

```
decl context = de_get_current_design_context();
// Start a transaction
decl transH = db_create_transaction(context, "Add Construction Lines");
// isEmpty is TRUE.
decl isEmpty = db_transaction_is_empty(transH);
// Add a horizontal construction line.
de_add_construction_line(0, 0, 100, 0);
// Add a diagonal construction line.
de_add_construction_line(0, 0, 100, 100);
// isEmpty is FALSE.
isEmpty = db_transaction_is_empty(transH);
// End a transaction.
db_commit_transaction(transH);
```

Version Introduced

ADS 2009 Update 1

***Where Used:* (ael)**

Schematic, Layout

User Interface Functions

This section describes each User Interface function in detail. The functions are listed in alphabetical order.

add_menu() (ael)	*de_data_dialog()* (ael)
add_separator() (ael)	*de_info()* (ael)
api_dlg_unmanage() (ael)	de_open_check_rep_dialog()
api_get_current_window() (ael)	*de_open_hierarchy_dialog()* (ael)
api_get_graph_color_index_by_name() (ael)	*de_open_info_dialog()* (ael)
api_get_graph_color_name_by_index() (ael)	*de_print_info()* (ael)
api_get_graph_pattern_index_by_name() (ael)	*de_prompt()* (ael)
api_get_graph_pattern_name_by_index() (ael)	*de_question()* (ael)
api_get_window_graph() (ael)	*set_user_menu_label()* (ael)
api_set_current_window() (ael)	
api_set_current_window_by_seq_num() (ael)	
check_user_menu() (ael)	
clear_user_menu() (ael)	

add_menu()

Adds an item, menuItemName, to the user-defined menu and associates the AEL function aelFuncName with the item. When you select the item, the function will be called. The user-defined menu appears in the menu bar of the current window and the menu name is set with *set_user_menu_label()*. Returns: none.

See also: *add_separator()* (ael), *set_user_menu_label()* (ael), *check_user_menu()* (ael), *clear_user_menu()* (ael).

Syntax:

```
add_menu(menuItemName, aelFuncName, menuName);
```

where

menuItemName is the name to display as a selectable item in the menu.

aelFuncName is a string. AEL function to call when item is selected (should have no arguments).

menuName is one of these pre-defined internal names. "User", "User2", "User3", "User4", "User5". It is also the name displayed on the top of the menu if it hasn't been changed by a call to set_user_menu_label().

Example:

This example only adds a menu to the first user menu and does not check first to see whether a menu slot has been used. For a complete example, which checks first for an empty slot, refer to *Creating a Custom Menu* (custom).

```
defun my_func_ael()
{
```

```
   fputs(stderr, "my menu");
}
add_menu("myfunc", "my_func_ael", "User");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Batch Simulation

Where Used: **(ael)**

Graphical user interface

add_separator()

Adds a separator in the menu. Returns: none.

See also: *add_menu()* (ael), *set_user_menu_label()* (ael), *check_user_menu()* (ael), *clear_user_menu()* (ael).

Syntax:

```
add_separator(menuName);
```

where

menuName is optional. It is a predefined internal name: *User*, *User2*, *User3*, *User4*, *User5*. It is also the name displayed on the top of the menu providing it wasn't changed by a call to set_user_menu_label(). Default is *User*.

Example:

```
add_menu("Command 1", "default_cb", NULL);
add_menu("Command 2", "default_cb", NULL);
add_separator();
add_menu("Command 3", "default_cb", NULL);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Batch Simulation

Where Used: **(ael)**

Graphical user interface

add_menu()

Adds an item, menuItemName, to the user-defined menu and associates the AEL function aelFuncName with the item. When you select the item, the function will be called. The user-defined menu appears in the menu bar of the current window and the menu name is set with *set_user_menu_label()*. Returns: none.

See also: *add_separator()* (ael), *set_user_menu_label()* (ael), *check_user_menu()* (ael), *clear_user_menu()* (ael).

Syntax:

```
add_menu(menuItemName, aelFuncName, menuName);
```

where

menuItemName is the name to display as a selectable item in the menu.

aelFuncName is a string. AEL function to call when item is selected (should have no arguments).

menuName is one of these pre-defined internal names. "User", "User2", "User3", "User4", "User5". It is also the name displayed on the top of the menu if it hasn't been changed by a call to set_user_menu_label().

Example:

This example only adds a menu to the first user menu and does not check first to see whether a menu slot has been used. For a complete example, which checks first for an empty slot, refer to *Creating a Custom Menu* (custom).

```
defun my_func_ael()
{
   fputs(stderr, "my menu");
}
add_menu("myfunc", "my_func_ael", "User");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Batch Simulation

Where Used: (ael)

Graphical user interface

add_separator()

Adds a separator in the menu. Returns: none.

See also: *add_menu()* (ael), *set_user_menu_label()* (ael), *check_user_menu()* (ael), *clear_user_menu()* (ael).

Syntax:

```
add_separator(menuName);
```

where

menuName is optional. It is a predefined internal name: *User*, *User2*, *User3*, *User4*, *User5*. It is also the name displayed on the top of the menu providing it wasn't changed by a call to set_user_menu_label(). Default is *User*.

Example:

```
add_menu("Command 1", "default_cb", NULL);
add_menu("Command 2", "default_cb", NULL);
add_separator();
add_menu("Command 3", "default_cb", NULL);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Batch Simulation

***Where Used:* (ael)**

Graphical user interface

api_dlg_unmanage()

Unmanages the specified dialog.

Syntax:

```
api_dlg_unmanage(dlgH);
```

where

dlgH is a dialog object that is obtained when the dialog is created via api_dlg_create_dialog().

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Netlist with Node Names

Where Used: **(ael)**

Graphical user interface

api_get_current_window()

Returns the handle to the currently active window.

Syntax:

```
api_get_current_window();
```

Example:

```
decl currentWindowHandle;
currentWindowHandle = api_get_current_window();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Export all Variables
Get Meas Equation Variable String
Get MeasEqn String 2

Where Used: **(ael)**

Graphical user interface

api_get_graph_color_index_by_name()

Returns the index of the color for the given color name; null if not found.

See also: *api_get_graph_color_name_by_index()* (ael), *api_get_current_window()* (ael), *api_get_window_graph()* (ael).

Syntax:

```
api_get_graph_color_index_by_name(graphH, colorName);
```

where

graphH is a handle to the window graphics area, as returned from api_get_window_graph().

colorName is the color name; a string. A list of available strings can be found in the $HPEESOF_DIR / config directory in the hpeecolor.cfg file.

Example:

```
decl winInst, graphH, colorIndex;
winInst = api_get_current_window();
graphH = api_get_window_graph (winInst);
colorIndex = api_get_graph_color_index_by_name (graphH, "black");
```

Where Used: **(ael)**

Graphical user interface

api_get_graph_color_name_by_index()

Returns the name of the color for the given color index; null if not found.

See also: *api_get_graph_color_index_by_name()* (ael), *api_get_current_window()* (ael), *api_get_window_graph()* (ael).

Syntax:

```
api_get_graph_color_name_by_index(graphH, colorIndex);
```

where

graphH is a handle to the window graphics area, as returned from api_get_window_graph().

colorIndex is the color index (integer >= 0).

Example:

```
decl winInst, graphH, colorName;
winInst = api_get_current_window();
graphH = api_get_window_graph (winInst);
colorName = api_get_graph_color_name_by_index (graphH, 1);
```

Where Used: **(ael)**

Graphical user interface

api_get_graph_pattern_index_by_name()

Returns the index of the pattern for the given pattern name; null if not found.

See also: *api_get_graph_pattern_name_by_index()* (ael), *api_get_current_window()* (ael), *api_get_window_graph()* (ael).

Syntax:

```
api_get_graph_pattern_index_by_name(graphH, ptnName);
```

where

graphH is a handle to the window graphics area, as returned from api_get_window_graph().

ptnName is the pattern name; a string. A list of available strings can be found in the $HPEESOF_DIR / config directory in the hpeefill.cfg file.

Example:

```
decl winInst, graphH, ptnIndex;
winInst = api_get_current_window();
graphH = api_get_window_graph (winInst);
ptnIndex = api_get_graph_pattern_index_by_name (graphH, "zigzag_1");
```

***Where Used* (ael):**

Graphical user interface

api_get_graph_pattern_name_by_index()

Returns the name of the pattern for the given pattern index; null if not found.

See also: *api_get_graph_pattern_index_by_name()* (ael), *api_get_current_window()* (ael), *api_get_window_graph()* (ael).

Syntax:

```
api_get_graph_pattern_name_by_index(graphH, ptnIndex);
```

where

graphH is a handle to the window graphics area, as returned from api_get_window_graph().

ptnIndex is the pattern index (integer >= 0).

Example:

```
decl winInst, graphH, ptnName;
winInst = api_get_current_window();
graphH = api_get_window_graph (winInst);
ptnName = api_get_graph_pattern_name_by_index (graphH, 1);
```

Where Used: **(ael)**

Graphical user interface

api_get_window_graph()

Returns a handle to the graphics area of the given window.

Syntax:

```
api_get_window_graph(windowHandle);
```

where

windowHandle is a handle of a window to retrieve the graphics area from.

Example:

```
decl windowHandle, graphHandle;
windowHandle = api_get_current_window();
graphHandle = api_get_window_graph(windowHandle);
```

Where Used: **(ael)**

Graphical user interface

api_set_current_window()

Sets a window to be currently active. Returns: the handle of the window replaced by the window specified by this function.

See also: *api_get_current_window()* (ael), *api_set_current_window_by_seq_num()* (ael).

Syntax:

```
api_set_current_window([windowType | windowHandle]);
```

where

windowType is the type of window, where:

- MAIN_WIN = Main window
- SCHEM_WIN = Schematic window

- LAYOUT_WIN = Layout window

OR

windowHandle is a handle of a window to retrieve the graphics area from.

Example:

```
decl oldWinHandle;
oldWinHandle = api_set_current_window(SCHEM_WIN);
```

OR

```
oldWinHandle = api_set_current_window(windowHandle);
```

***Where Used:* (ael)**

Graphical user interface

api_set_current_window_by_seq_num()

Makes the window instance having the given sequence number as the current window. This function can be used to select the exact window instance of windows that can have multiple instances. Returns: The handle to the previous window instance.

See also: *de_create_window()* (ael), *api_set_current_window_by_seq_num()* (ael).

Syntax:

```
api_set_current_window_by_seq_num(seqNo);
```

where

seqNo is the window sequence number as the order of when the window was opened.

Example:

```
// Closes the current window instance
api_set_current_window_by_seq_num (1);
de_close_window();
```

***Where Used:* (ael)**

Graphical user interface

check_user_menu()

Five user menus are available in each of the three ADS windows: Main, Schematic, and Layout. This function can be used to determine if a user menu is being used by another application. It returns the current label of the specified user menu. If the label has not be reset by a user or application, the default name will be returned. If a different label is returned, the label has been reset, and the menu is probably in use by another application.

This function takes an integer 1-5 and returns the default names of "User", "User2", "User3", "User4", or "User5". These predefined AEL constants may be used instead: deUserMenuName, deUser2MenuName, deUser3MenuName, etc. For a complete example, which uses this function to find the first available user menu, refer to *Creating a Custom Menu* (custom) in *Customization and Configuration* (custom).

See also: *add_menu()* (ael), *add_separator()* (ael), *set_user_menu_label()* (ael), *clear_user_menu()* (ael).

Syntax:

```
check_user_menu( menuIndex );
```

where

menuIndex is an integer 1-5

Example:

```
fputs(stderr, check_user_menu( 1 ));
```

***Where Used:* (ael)**

Graphical user interface

clear_user_menu()

Removes and clears all entries in the user-defined menu. Returns: none.

See also: *add_menu()* (ael), *add_separator()* (ael), *set_user_menu_label()* (ael), *check_user_menu()* (ael).

Syntax:

```
clear_user_menu();
```

Example:

The following command clears the second user menu, named "User2".

```
clear_user_menu(2);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Batch Simulation

***Where Used:* (ael)**

Graphical user interface

de_install_get_xy_pair()

Installs a user-defined event handler for prompting in the current window. Returns: none.

Syntax:

```
de_install_get_xy_pair(aelFuncName[, autoRepeatable, prm1, prm2,
trackingStyle]);
```

Where,

- *aelFuncName* is a string. AEL function to call when two points are entered through left mouse clickings. The AEL function receives X1, Y1, X2, Y2 as its parameters.
- *autoRepeatable* is optional; sets repeat capability, where:
  0 = one-shot (default). The routine is not re-installed after the (x,y) location is entered.
  1 =auto-repeatable. The event handler will be invoked every time the mouse is clicked until either the End Command button is pressed or another command is invoked.
- *prm1* is optional; the parameter to override the default event prompts. This is the prompt, if defined, for the 1st (x,y).
- *prm2* is optional; the parameter to override the default event prompts. This is the prompt, if defined, for the 2nd (x,y).
- *trackingStyle* is optional; sets the tracking style, where:
  TRACKING_RECT = rectangle rubberbanding lines between 1st and 2nd points.
  TRACKING_LINE = (default). straight rubberbanding line between 1st and 2nd points.

  TRACKING_NONE = no rubberbanding line between 1st and 2nd points.

Example:

```
defun my_print_2_points(x1,y1,x2,y2)
{
   fputs(stderr, fmt_tokens(list (x1,y1,x2,y2)));
}
de_install_get_xy_pair("my_print_2_points", 1, "enter the first point",
   "enter the second point", TRACKING_RECT);
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_install_get_xy() (ael)
de_list_select() (ael)

Where Used (ael)

Graphical user interface

de_install_get_xy()

Installs a user-defined event handler for prompting in the current window. Returns: none.

Syntax:

```
de_install_get_xy(aelFuncName[, autoRepeatable, prmStr]);
```

Where,

- *aelFuncName* is a string. AEL function to call when (x,y) location is entered by clicking the left mouse button. The (x,y) coordinate of the pointer location is passed in to the AEL function.
- *autoRepeatable* is optional; sets repeat capability, where:
  0 = one-shot (default). The routine is not re-installed after the (x,y) location is entered.
  1 = auto-repeatable. The event handler will be invoked every time the mouse is clicked until either the End Command button is pressed or another command is invoked.
- *prmStr* is optional; the parameter to override the default event prompt.

Example:

```
// The Example: prints out point each time mouse is clicked.
defun my_print_xy(x,y)
{
   fputs(stderr, fmt_tokens(list (x,y)));
}
de_install_get_xy("my_print_xy", 1, "enter the (x,y) location");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_install_get_xy_pair() (ael)
de_list_select() (ael)

Where Used (ael)

Graphical user interface

de_list_select()

Pops up a list selection dialog box. Returns a string. The "ok" button returns the selected string. The "cancel" button returns a NULL string which passes the is_string() function test.

Syntax:

```
de_list_select(title, selectionList, aelFuncName);
```

Where,

- *title* is the dialog box title.
- *selectionList* is the list of selections to display in the dialog box.
- *aelFuncName* is a string. AEL function to call when component selected.

Example:

```
//The example prints the string for the selected component after dialog box is displayed and OK is
pressed.
defun item_selected(item)
{
   fputs(stderr, strcat(item,"was selected"));
}
de_list_select("Items",list("one","two","three"),"item_selected");
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

de_install_get_xy() (ael)
de_install_get_xy_pair() (ael)

Where Used (ael)

Graphical user interface

de_data_dialog()

Displays a scrollable multiline text editor. Returns: none.

Syntax:

```
de_data_dialog (title, dataStr, editable, okCB, [userData]);
```

where

title is the dialog box title.

dataStr is the initial text to display in the dialog box.

editable indicates whether text is editable, where:

- 0 = text is not editable
- 1 = text is editable

okCB is the AEL function to execute when OK is pressed. If this is Null, then userData needs to specify a function which expects 1 string argument (*See Example 2*)

userData is optional; the AEL function executed when OK is selected. This function expects 1 string argument, dataStr. It will unmanage the data dialog.

Example 1: unmanage the dlg in the specified callback

```
// prints the text in the data dialog to stderr
defun print_data_dialog_text_stderr(okH, dlgH, winInst)
{
    decl infoH, text;
    infoH = api_dlg_find_item(dlgH, DE_TEXT_EDIT_PRINT_ITEM);
    api_dlg_get_resources(infoH, API_RN_VALUE, &ext);
    fputs(stderr, text);
    api_dlg_unmanage(dlgH);  // this is required when a callback func
                             // is specified as the 4th arg
                             // in 'de_data_dialog'
}
de_data_dialog("Date/Time Editor", "Date:12/12/94\n Time: 12:00PM\n",
1, "print_data_dialog_text_stderr");
```

Example 2: using NULL as the 4th arg in de_data_dialog

```
// prints the text in the data dialog to stderr
// dataStr is the text from the dialog box (the second parameter to
// 'de_data_dialog')
defun print_data_dialog_text_stderr(dataStr)
{
    fputs(stderr, dataStr);
    // api_dlg_unmanage is not required when a func is
    // specified as the 5th arg in 'de_data_dialog' and the 4th arg is
    // NULL
}
de_data_dialog("Date/Time Editor", "Date:12/12/94\n Time: 12:00PM\n",
1, NULL, "print_data_dialog_text_stderr");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Get Path Info
Netlist with Node Names

***Where Used:* (ael)**

Graphical user interface

de_info()

Displays an information dialog box. Returns: none.

Syntax:

```
de_info (infoMsg, timeoutFlag);
```

where

infoMsg is the information to display in the dialog box.

timeoutFlag is the setting to dismiss the dialog box after a timeout interval. The timeout interval is set via DIALOG_TIME_OUT environment variable, where:

- 0 = do not dismiss automatically
- 1 = dismiss automatically

Example:

```
de_info("Save completed", 1);
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.

Copy File
Scan Directory for Dirs
Write Permission to File?

***Where Used:* (ael)**

Graphical user interface

de_open_hierarchy_dialog()

Opens dialog box to display hierarchy of current design. Returns: none.

Syntax:

```
de_open_hierarchy_dialog();
```

Example:

```
de_open_hierarchy_dialog();
```

Where Used: **(ael)**

Graphical user interface

de_open_info_dialog()

Opens dialog box for displaying information. Returns: none.

Syntax:

```
de_open_info_dialog();
```

Example:

```
de_open_info_dialog();
```

Where Used: **(ael)**

Graphical user interface

de_print_info()

Prints the contents of the following dialog boxes: data, new features, info, hierarchy, editor, DSE schematic status, DSE layout status, and check rep report. Returns: none.

Syntax:

```
de_print_info(dialogType);
```

where

dialogType is one of the these dialog types:

- DE_DATA DIALOG
- DE_NEW_FEATURES_DIALOG
- DE_INFO_DIALOG
- DE_HIERARCHY_DIALOG
- DE_EDITOR2_DIALOG
- DE_DSE_SCHEM_STATUS_DIALOG
- DE_DSE_LAYOUT_STATUS_DIALOG
- DE_REP_REPORT_DIALOG

Example:

```
de_print_info(DE_DATA_DIALOG);
```

***Where Used:* (ael)**

Graphical user interface

de_prompt()

Invokes the prompt dialog box in the current window with user-supplied prompt string and sets a call back AEL function (*aelFuncName*) to be called when the OK button is selected. Dialog box has a single text field in which to type a response to the prompt string.
Returns: none.

Syntax:

```
de_prompt(title, promptLabel, defaultAnswer, okCB, aelFuncName[, numColumns[,
userCbData]]);
```

where

title is a dialog box title.

promptLabel is a string; label.

defaultAnswer is the initial text displayed in the text field. This string can be set to NULL.

okCB is the OK callback which will override the default OK callback. NULL if not used. This callback function should take three arguments. For example, my_prompt_ok_cb (okH, dlgH, winInst), where:

- okH is the handle to the OK button
- dlgH is the handle to the prompt dialog
- winInst is the current window instance

aelFuncName is a string. AEL function to call when OK is pressed. This function takes 1 string argument which is the response text in the text field. The response text is passed to the AEL function when OK is pressed. An optional user callback data can be passed to this function for its 2nd argument if specified in the *userCbData* optional argument.

numColumns is optional; is an integer to manually specify the display size in how many columns for the single text field.

userCbData is optional; is user specified callback data to pass into the aelFuncName function as that function's 2nd argument.

Example:

```
defun my_prompt_cb(input_str, userCbData)
{
```

```
  fputs(stderr, strcat(userCbData, input_str));
}
de_prompt("my prompt", "please enter answer", "yes", NULL, "my_prompt_cb", 20, "Retrieved value:
");
```

Where Used: **(ael)**

Graphical user interface

de_question()

Invokes question dialog box in the current window. Returns: none.

Syntax:

```
de_question (title, questionStr, yesAelFuncName, noAelFuncName);
```

where

title is the dialog box title.

questionStr is the question to display in the dialog box.

yesAelFuncName is a string. AEL function to call when yes is selected.

noAelFuncName is a string. AEL function to call when no is selected.

Example:

```
de_question("Save As", "Save File", "yes_cb", "no_cb");
defun yes_cb(selH, dlgH, winInst)
{
         fputs(stderr,"Answer yes");
         api_dlg_unmanage(dlgH);
}
defun no_cb(selH, dlgH, winInst)
{
         fputs(stderr,"Answer no");
         api_dlg_umanage(dlgH);
}
```

Where Used: **(ael)**

Graphical user interface

set_user_menu_label()

Sets the label for the user menu. Returns: none.

See also: *add_menu()* (ael), *add_separator()* (ael), *check_user_menu()* (ael), *clear_user_menu()* (ael).

Syntax:

```
set_user_menu_label(menuLabel, menuName);
```

where

menuLabel is the name to display on the user menu.

menuName is one of the following internal labels for the user menus: "User", "User2", "User3", "User4", or "User5".

Example:

For a complete example, which checks first for an empty slot, refer to *Creating a Custom Menu* (custom) in *Customization and Configuration* (custom).

```
// changes the label on the 2nd user menu
set_user_menu_label("My Menu", "User2");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Batch Simulation

***Where Used:* (ael)**

Graphical user interface

Utility Functions for AEL

This section describes each Utility function in detail. The functions are listed in alphabetical order.

A,C	
ael_file_exists() (ael) *ael_is_file_writable()* (ael) *arg()* (ael) *arg_list()* (ael) *array_lowerBound()* (ael) *array_size()* (ael) *array_type()* (ael) *array_upperBound()* (ael)	*call()* (ael) *call_depth()* (ael) *check_syntax()* (ael) *chmod()* (ael) *convert_array()* (ael)
D,E	
date_time() (ael) *de_error_bell()* (ael) *de_exit()* (ael) *de_get_env()* (ael) *de_set_error_bell()* (ael) *de_simple_dialog_cancel_cb()* (ael) *de_valid_name()* (ael)	*de_set_warning_bell()* (ael) *de_warning_bell()* (ael) *delete_word()* (ael) *error()* (ael) *evaluate()* (ael) *execute()* (ael) *expandenv()* (ael)
F,G	
file_loaded() (ael) *filedate()* (ael) *find_word()* (ael) *find_word_voc()* (ael) *fix_path()* (ael) *format_date_time()* (ael)	*generic_netlist_cb()* (ael) *get_dir_files()* (ael) *getcwd()* (ael) *getenv()* (ael) *geterror()* (ael) *getppid()* (ael) *getsysenv()* (ael)
I,L	
identify_value() (ael) *is_complex()* (ael) *is_dir()* (ael) *is_file()* (ael) *is_function()* (ael) *is_function_defined()* (ael) *is_integer()* (ael)	*is_list()* (ael) *is_real()* (ael) *is_string()* (ael) *is_type()* (ael) *is_voc()* (ael) *list_undefined()* (ael) *load()* (ael)
M,N,O,P	
mkdir() (ael) *num_args()* (ael) *offset_array()* (ael) *on_error()* (ael) *pcb_get_form_value()* (ael) *pcb_get_mks()* (ael)	*pcb_get_parm_type()* (ael) *pcb_get_string()* (ael) *pcb_set_form_value()* (ael) *pcb_set_mks()* (ael) *pcb_set_string()* (ael)
R,S,T,V,W	
rename_word() (ael) *resize_array()* (ael) *setenv()* (ael) *sleep()* (ael) *start_timer()* (ael) *system()* (ael) *tmpnam()* (ael)	*total_elapsed_time()* (ael) *validate_name()* (ael) *warning()* (ael) *what_col()* (ael) *what_file()* (ael) *what_function()* (ael) *what_line()* (ael)

ael_file_exists()

Returns TRUE if the given file path string or directory path string exists. Returns FALSE if the given file path string or directory path string does not exist.

Syntax

```
decl doesExist = ael_file_exists(pathStr);
```

Where,

- *pathStr* is a string of a full directory path or full file path.

Example

```
decl dirPath = "C:/my_test/directory/";
decl filePath = strcat(dirPath, "my_file.txt"); // Points to file:
C:/my_test/directory/my_file.txt
if ( ael_file_exists(dirPath ) && ael_is_file_writable(dirPath) )
{
   if (ael_file_exists(filePath) && ael_is_file_writable(filePath) )
   {
       decl fid = fopen(filePath, "A");
       if (fid == NULL)
           return errorVal; // TODO handle error

       fputs(fid, "# Added to the end of the File this message. \n");
       fclose(fid);
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

ael_is_file_writable() (ael)

Where Used (ael)

Schematic, Layout

ael_is_file_writable()

Returns TRUE if the given directory path string or file path string is writable. Returns FALSE if the given directory path string or file path string does not have write permissions.

Syntax

```
decl isWritable = ael_is_file_writable(pathStr);
```

Where,

- *pathStr* is a string of a full directory path or full file path.

Example

```
decl dirPath = "C:/my_test/directory/";
decl filePath = strcat(dirPath, "my_file.txt"); // Points to file:
C:/my_test/directory/my_file.txt
```

```
if ( ael_file_exists(dirPath ) && ael_is_file_writable(dirPath) )
{
   if (ael_file_exists(filePath) && ael_is_file_writable(filePath) )
   {
       decl fid = fopen(filePath, "A");
       if (fid == NULL)
           return errorVal; // TODO handle error

       fputs(fid, "# Added to the end of the File this message. \n");
       fclose(fid);
   }
}
```

Version Introduced

ADS 2011

Version Compatible

ADS 2011 and newer versions

See also

ael_file_exists() (ael)

Where Used (ael)

Schematic, Layout

arg()

Returns the nth argument passed into a function. Arguments start at 0.

Syntax:

```
arg(n);
```

where

n is a position. Integer number of the argument to retrieve.

Example:

```
defun test(a, b, c)
{
   fputs(stderr, arg(2)); // prints value of c
}
```

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

arg_list()

Returns the arguments passed to a function as a list.

Syntax:

```
arg_list();
```

Example:

```
defun test(a, b, c)
{
  decl l;
  l = arg_list();
      fputs(stderr, identify_value(arg_list()));
                  // prints value of a, b, c as a list
}
```

In the above example, where test(1, 2, 3), prints list(1, 2, 3)

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

array_lowerBound()

Returns the lower bound of the specified dimension in an AEL array. Since this function does not work on data from datasets, ensure that the parameters passed are arrays.

See also: *offset_array()* (ael), *resize_array()* (ael), *array_size()* (ael), *array_type()* (ael), *array_upperBound()* (ael), *convert_array()* (ael).

Syntax:

```
lb = array_lowerBound(<arr>,<dim>);
```

where

arr is a valid AEL array.

lb is an integer or NULL_VALUE if unsuccessful.

Example:

```
decl a = {{1,2,3},{4,5,6}};
fputs(stderr, array_lowerBound(a,1));  // outputs 0
```

```
fputs(stderr, array_lowerBound(a,2));  // outputs 0
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

array_size()

Returns the size of each dimension in an AEL array. Since this function does not work on data from datasets, ensure that the parameters passed are arrays.

See also: *offset_array()* (ael), *resize_array()* (ael), *array_type()* (ael), *array_lowerBound()* (ael), *array_upperBound()* (ael), *convert_array()* (ael).

Syntax:

```
size = array_size(<arr>)
```

where

arr is a valid AEL array.

size is defined by: if <arr> is one-dimensional array, an integer is returned; if <arr> is multi-dimensional array, an array is returned.

Example:

```
decl a = {{1,2,3},{4,5,6}};
decl b = {1,2,3};
fputs(stderr, identify_value(array_size(a))); // outputs {2,3}
fputs(stderr, identify_value(array_size(b))); // outputs 3
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

array_type()

Returns the type of elements in an AEL array. Since this function does not work on data from datasets, ensure that the parameters passed are arrays.

See also: *offset_array()* (ael), *resize_array()* (ael), *array_size()* (ael), *array_lowerBound()* (ael), *array_upperBound()* (ael), *convert_array()* (ael).

Syntax:

```
type = array_type(<arr>)
```

where

arr is a valid AEL array.

type is one of the following AEL strings: "integer", "real", "complex".

Example:

```
decl a = {{1,2,3},{4,5,6}};
decl b = convert_array(a,"complex");
fputs(stderr, array_type(a)); // outputs "integer"
fputs(stderr, array_type(b)); // outputs "complex"
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

array_upperBound()

Returns the upper bound of the specified dimension in an AEL array. Ensure that the parameters passed are arrays.

See also: *offset_array()* (ael), *resize_array()* (ael), *array_size()* (ael), *array_type()* (ael), *array_lowerBound()* (ael), *convert_array()* (ael).

Syntax:

```
ub = array_upperBound(<arr>,<dim>);
```

where

arr is a valid AEL array.

ub is an integer or NULL_VALUE if unsuccessful.

Example:

```
decl a = {{1,2,3},{4,5,6}};
fputs(stderr, array_upperBound(a,1));  // outputs 1
fputs(stderr, array_upperBound(a,2));  // outputs 2
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

call()

Invokes or calls a function, gives a function value and an argument list (as returned from *arglist()* or *list()*). Returns: none.

Syntax:

```
call(fnc, args);
```

where

fnc is the name of AEL function to execute.

args is the argument list to pass into the function.

Example:

```
defun my_fnc(s1,s2) {return strcat(s1,s2);}
call (my_fnc, list("a", "b"));
```

is equivalent to:

```
my_fnc("a", "b");
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

call_depth()

Returns the depth of the active call stack. The call depth can be used in conjunction with what_file and what_line to produce a call trace back. Returns the depth of the active call stack, where: 1 = depth if the function is called directly from the command line or menu, 2 = depth if the function is called from within another function.

Syntax:

```
call_depth();
```

Example:

```
defun test()
{
     fputs(stderr, call_depth());
}
```

If this example is invoked from the command line, outputs 0; when test() is invoked, outputs 1.

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

check_syntax()

Interprets a string as an AEL command and returns whether or not the syntax is valid.

Note The function stores the resulting commands in a temporary ATF file and then removes that ATF file. Also, the string is checked for SYNTAX only. It does not check that all identifiers are defined! See *list_undefined()* (ael) for this type of check. Returns True or False.

Syntax:

```
check_syntax(string);
```

where

num is a string argument that contains the text to be checked as AEL syntax.

Example:

```
decl str = "cos(3+);";
if ( !check_syntax(str) )
 fputs(stdout, "Invalid syntax");
else
 execute(str);        // returns False
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

chmod()

Changes the mode of the specified file. Returns NULL if an error occurs, TRUE if the chmod

occurs.

Syntax:

```
chmod(filename, int);
```

where

filename is a string that contains the name of the file to change.

int is a four-digit integer that specifies the mode to change to. The first digit is a 0 and the next three digits are the same integers that define the mode in the UNIX chmod command.

The second digit = permissions for the user who owns the file: read (4), write (2), and execute (1)
The third digit = permissions for other users in the file's group: read (4), write (2), and execute (1)
The fourth digit = permissions for other users NOT in the file's group: read (4), write (2), and execute (1)

The octal (0-7) value is calculated by adding up the values for each digit
User (rwx) = 4+2+1 = **7**
Group(rx) = 4+1 = **5**
World(x) = 1 = **1**

NOTE: This command currently does not work on Windows.

Example:

```
chmod("myfile.txt", 0751); // grants read,write,and execute permission
for user for myfile.txt
                           // grants read, execute permission for
group for myfile.txt
                           // grants execute permission for all for
myfile.txt

chmod("myfile.txt", 0777); // grants read, write, and execute permission
to everyone for myfile.txt
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

convert_array()

Returns a new AEL array created from an existing array, but with the elements of the existing array converted to the specified type. Since this function does not work on data from datasets, ensure that the parameters passed are arrays.

See also: *offset_array()* (ael), *resize_array()* (ael), *array_size()* (ael), *array_type()* (ael), *array_lowerBound()* (ael), *array_upperBound()* (ael).

Syntax:

```
newArr = convert_array(<arr>, <str>);
```

where

arr is a valid AEL array

str is a valid AEL string. Possible values:

- "integer", "int", "long" will convert to integer
- "real", "double" will convert to real
- "complex" will convert to complex

Example:

```
decl a = {{1,2,3},{4,5,6}};
decl b = convert_array(a,"complex");
fputs(stderr, identify_value(a)); // outputs {1,2,3},{4,5,6}}
fputs(stderr, identify_value(b)); // outputs
{{1+0i,2+01,3+0i},{4+0i,5+0i,6+0i}}
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

date_time()

Returns the date and time as a formatted string. This string returned by this function is identical to the C language's ctime function. Adds a new line, \n, at the end of the string.

Syntax:

```
date_time();
```

Example:

```
date_time(); //returns Sun Apr 6 15:24:13 2002
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

de_error_bell()

Sounds the error bell. Returns: none.

Syntax:

```
de_error_bell();
```

Example:

```
de_error_bell();                    // emits beep on error
```

Where Used: **(ael)**

Schematic

de_exit()

Exits Advanced Design System. Returns: none.

Syntax:

```
de_exit();
```

Example:

```
de_exit();
```

Where Used: **(ael)**

Schematic

de_get_env()

Retrieves environment file name.

See also: *getenv()* (ael), *setenv()* (ael).

Syntax:

```
de_get_env();
```

Example:

```
decl envFileName;
envFileName = de_get_env();
```

Where Used: **(ael)**

Schematic

delete_word()

Deletes a word from the current vocabulary, or from a specified vocabulary. Returns True if the name is found and deleted; False if cannot delete or if an invalid vocName was specified.

Syntax:

```
delete_word(name, vocName <opt>);
```

where

name is a string that contains the name of the word to delete.

vocName is a string that contains the name of the vocabulary from which to delete name.

Example:

```
decl a, b;
a = delete_word("b");
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

de_set_error_bell()

Enables or disables the error bell (on or off). Returns: none.

Syntax:

```
de_set_error_bell(bEnabled);
```

where

bEnable is 1 to turn on; 0 to turn off the error bell.

Example:

```
de_set_error_bell(1);        // enables the error bell
```

Where Used: **(ael)**

Schematic

de_set_warning_bell()

Enables or disables the warning bell (on or off). Returns: none.

Syntax:

```
de_set_warning_bell(bEnabled);
```

where

bEnabled is 1 to turn on; 0 to turn off the warning bell.

Example:

```
de_set_warning_bell(1);        // enables the warning bell
```

Where Used: **(ael)**

Schematic

de_simple_dialog_cancel_cb()

A general purpose cancel callback. Closes (but does not destroy) the dialog. The state of the dialog is preserved. Returns: none.

Syntax:

```
de_simple_dialog_cancel_cb(cancelH, dialogH, winInst);
```

where

cancelH is a handle to the cancel button.

dialogH is a handle to the dialog to close.

winInst is the handle of the window.

Example:

```
decl cancelH, dialogH;
if (dialogH == NULL)
{
dialogH = api_dlg_create_dialog("myDialog", winInst,
cancelH = api_dlg_create_item("cancelButton", API_PUSH_BUTTON_ITEM,
API_RN_CAPTION, "CANCEL");
); //end api_dlg_create_dialog
api_dlg_add_callback( cancelH, "de_simple_dialog_cancel_cb",
API_ACTIVATE_CALLBACK, dialogH);
}
```

***Where Used:* (ael)**

Schematic

de_valid_name()

Returns TRUE or FALSE

Syntax:

```
de_valid_name (name, bNoext, bCheckgs1, errmsg);
```

where

name is leaf name to be checked

bNoext is optional, if TRUE, dot (".") is not allowed in name

bCheckgs1 is optional, if TRUE, check validity of the valid characters in name. Valid characters are alpha numeric and _ ' @ # $ % & | + - ! ^

errmsg is the error message that will be output

Example:

```
de_valid_name("myName");
```
which is the same as
```
de_valid_name("myName", FALSE, TRUE, NULL);
```
because default bNoext is False, default bCheckgs1 is TRUE
and default errmsg is NULL

```
de_valid_name("myName", TRUE, TRUE, "Invalid name");
```

Where Used: **(ael)**

Schematic

de_warning_bell()

Sounds the warning bell. Returns: none.

Syntax:

```
de_warning_bell();
```

Example:

```
de_warning_bell(); // beeps the bell as warning
```

Where Used: **(ael)**

Schematic

error()

Reports an error. Pops up the error dialog, displaying given error. Returns: none.

Syntax:

```
error(errorFileName, errorNum, defaultErrString, infoString);
```

where

errorFileName is the name of error message file (for internationalized messages). The file does not have to exist if default error string is given.

errorNum is the line number in the message file for this error string.

defaultErrString is a string used f no error found or error file not found.

infoString is the non-internationalized varying part of error string.

Example:

```
error("aelcmd", 2, "item error", "hello");
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

evaluate()

Evaluates an AEL expression and returns the result. The string passed must be a valid AEL expression. The value of the expression is returned as well as stored in the global variable EvaluationResult.

AEL is always interpreted in the context of a vocabulary of definitions which are used to resolve identifier references. Whenever a variable or function name is used in the AEL program, the context vocabulary is searched for an existing definition. New variables and function definitions are added to the context vocabulary as well. The optional second argument allows the context to be overridden for this function by providing the name of the desired vocabulary.

In the design environment, two contexts are important: SimCmd and CmdOp. Generally SimCmd is the context for any AEL files loaded during program initialization and when opening a workspace. The context is CmdOp when interpreting menu commands and in the Command Line dialog box. SimCmd is a superset of CmdOp, so not all definitions in SimCmd are available from the Command Line dialog without specifying the context SimCmd specifically. Returns: Value of the expression.

By default, the scope of the variables accessed is global within the context. In other words, only global variables can be accessed. If local scope is desired, then the third parameter, which is optional, should be set to TRUE. This would most likely be the desired effect if this function is called from within another function so that the local variables can be accessed as well as the global variables.

See also: *load()* (ael) , *execute()* (ael).

Syntax

```
evaluate( string, [context], [local scope] );
```
Where,

- *string* is the text of the AEL expression.
- *context* is the name of vocabulary context for evaluation of the expression.
- *local scope* is a boolean TRUE or FALSE. TRUE means use local scope when accessing any variables found in <string>. FALSE, which is the default, uses global scope when accessing any variables found in <string>.

Example

```
decl x;
x = evaluate( "3 * 8" );   // sets x = integer value 24
                           // also sets EvaluationResult to 24
```

Where Used (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic,

Layout, Simulation, GUI

execute()

Interprets text as an AEL program. The text string must contain valid AEL statements.

AEL is always interpreted in the context of a vocabulary of definitions which are used to resolve identifier references. Whenever a variable or function name is used in the AEL program, the context vocabulary is searched for an existing definition. New variables and function definitions are added to the context vocabulary as well. The optional second argument allows the context to be overridden for this function by providing the name of the desired vocabulary.

In the design environment, two contexts are important: SimCmd and CmdOp. Generally SimCmd is the context for any AEL files loaded during program initialization and when opening a workspace. The context is CmdOp when interpreting menu commands and in the Command Line dialog box. SimCmd is a superset of CmdOp, so not all definitions in SimCmd are available from the Command Line dialog without specifying the context SimCmd specifically. Returns: None.

By default, the scope of the variables accessed is global within the context. In other words, only global variables can be accessed. If local scope is desired, then the 3rd parameter, which is optional, should be set to TRUE. This would most likely be the desired effect if this function is called from within another function so that the local variables can be accessed as well as the global variables.

Note
To use backslash '\' in combination with different characters you must type backslash twice (\\) instead of a single backslash. For more details, see *Introduction to AEL* (ael).

See also

load() (ael)
evaluate() (ael)

Syntax

```
execute( string, [context], [local scope] );
```

Where

- *string* is the text of the AEL program.
- *context* is the name of vocabulary context for evaluation of the expression.
- *local scope* is a boolean TRUE or FALSE. TRUE means use local scope when accessing any variables found in <string>. FALSE, which is the default, uses global scope when accessing any variables found in <string>.

Example

```
decl x;
execute( "x=3*8;" ); //sets x = integer value 24
```

Download Example File

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Set Meas Equation Variable String
Set Variable Value 2

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

expandenv()

Returns a string with the environment variables and EEsof configuration variables fully expanded into their values, given a string that contains environment variable references.

Syntax:

```
expandenv(str);
```

where

str is a string that contains environment variable references.

Example:

```
decl path;
path = expandenv("$PATH");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Create Directory
Get Registry Value
Read Configuration File

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

filedate()

Returns an integer that represents a time stamp of a file.

Syntax:

```
filedate(filepath);
```

where

filepath is a string that represents the file to check or a file pointer that points to a file obtained via *fopen()*.

Example:

```
decl a;
a = filedate("myAELfile.ael");
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

file_loaded()

Tests to see whether an AEL file has been read. Returns TRUE if the file exist and has been read, otherwise FALSE.

Syntax:

```
file_loaded(name [, voc, searchpath]);
```

where

name is a string that contains the name of the file to test.

voc is optional. A string that contains the name of the vocabulary (not really used for anything).

searchpath is optional. A string that contains the search path to search for the specified file.

Example:

```
decl a;
a = file_loaded("myAELfile.ael");
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

find_word()

Returns the address of an AEL word if it exists, otherwise NULL is returned.

Syntax:

```
find_word(name, voc);
```

where

name is a string that specifies the name of the AEL word to find.

voc is optional. A string that specifies what vocabulary to search for name. If it is not specified, the current vocabulary is searched. Note that parent vocabularies are searched if the word is not found at that level.

Example:

```
decl a;
a = find_word("cos");
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

find_word_voc()

Returns the address of a vocabulary in which a specified AEL word is found if it exists, otherwise NULL is returned.

Syntax:

```
find_word_voc(name, voc);
```

where

name is a string that specifies the name of the AEL word to find.

voc is optional. A string that specifies what vocabulary to search for name. If it is not specified, the current vocabulary is searched. Note that parent vocabularies are searched if the word is not found at that level.

Example:

```
decl a;
a = find_word_voc("cos", "AEL");
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

fix_path()

Adds the appropriate backslashes needed in a string to handle control characters correctly. This should be used when dealing with strings that represent paths. Returns a string that represents a path with backslashes added appropriately to handle control characters.

Syntax:

```
fix_path(path);
```

where

path is a string.

Example:

```
decl newPath;
newPath = fix_path("/users/myhome/project/template.ael");
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

format_date_time()

Returns the date and time as a formatted string, does not add a new line, \n, at the end of the string.The formatting can be specified using a format string. This string and

formatting rules are identical to the C language's *strftime()* function. Note, this function is sensitive to the LANG language environment variable and returns internationalized date and time strings based on this variable.

Syntax:

```
format_date_time(format_string);
```

where

format_string is identical to the format string accepted by the C library function *strftime()*. Each *%c* is replaced using values appropriate for the local environment). No more than *smax* characters are placed into *s*. The *strftime()* returns the number of characters, excluding the *\0*, or zero if more than *smax* characters were produced.

where
%a is the abbreviated weekday name
%A is the full weekday name
%b is the abbreviated month name
%B is the full month name
%c is the local date and time representation
%d is the day of the month (01-31)
%H is the hour (24-hour clock) (00-23)
%I is the hour (12-hour clock) (01-12)
%j is the day of the year (001-366)
%m is the month (01-12)
%M is the minute (00-59)
%p is the local equivalent of AM or PM
%S is the second (00-61)
%U is the week number of the year (Sunday as first day of week) (00-53)
%w is the weekday (0-6, Sunday is 0)
%W is the week number of the year (Monday as first day of week) (00-53)
%x is the local date representation
%X is the local time representation
%y is the year without century (00-99)
%Y is the year with century

Example:

```
decl datetime;
datetime = format_date_time("%a %B"); //prints "Weds July"
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

generic_netlist_cb()

This function allows the creation of a list of lists. It is not meant to be called directly.

Instead, use it to set up a netlist callback in an item definition. If using a user-defined netlist callback, call this function within the user-defined netlist callback, which will then have appropriate arguments for the non-data arguments.

The function returns a string value that represents the appropriate ADS formatting for an instance being netlisted.

Syntax:

```
generic_netlist_cb(cbP,
list(list(component, function, library),
list(list(param 1, netlist param name, parsing function), ...),
list(node param_1, node param_2,...),
list(default connection pin 1, list(), default connection pin 2, list(<node value 2>), ...),
list(pin name, pin name 2, ...),
list(, list(), <power pin 2>, list( 2),...)),
list(,list(),<substrate pin 2>, list(substrate node 2),...)),
instH, iterationInstName, iterationPinList)
```

Where

cbP is a handle to the callback function
cbData is a list of lists that represents callback data. This list of lists should have the following elements:

component is the value of parameter Model for the component name.
function is the function called if the component is referencing a parameter.
library is the specified component library.
param 1 is the parameter name.
netlist param name is the netlist name to which the parameter is mapped.
parsing function allows manipulation of the values for the current parameter. Each parameter in the list will be netlisted only if its value is not Null. If the parameter name is mapped. For example `list("r", "R", NULL)`, the value of r is output as R=<value>. You must optimize a parameter like this using variables.
node param_1 is an obsolete field, and is included for backward compatibility.
default connection pin 1 is a list of pins that will be connected to a default node value if the pin is unconnected in the schematic.
<node value> can be a fixed string, or "@<param name>" can be used to specify that a parameter's value should be used.
pin name is a list of pin names that defines the node order for the instance line.
<power pin> and *<power node>* are are concatenated together internally, and will function the same. The substrate pin/power pin is a non-physical pin that is declared in the terminal order field. When the pin name is encountered in the terminal order, it will not be found in the physical pin list. The power pin/substrate pin list will be consulted - if the name is found, the node value will be output. The node value can be a fixed string, or "@<param name>" can be used to specify that a parameter's value should be used.

instH is a handle to the instance being netlisted.
iterationInstName the instance name to use (eg. "inst_0_") for an iterated instance.
iterationPinList is the expanded net names for the expanded instance for an iterated instance. For example, for inst<0:1>, when inst_0_ is passed as iterationInstName, iterationPinList is passed in as list("*net1_0*", "*net2_0*", "*net3_0*").

Example:

```
generic_netlist_cb(cbP,
list(list("R", NULL, NULL), // Component is netlisted as R
list(list("r", "R", "cadenceParse"),list("temp","Temp", NULL)), // Parameters r, temp are
netlisted, r is renamed to R, temp is renamed to Temp,
                                                                  // parameter value r is formatted
by calling the function cadenceParse
        NULL,                                                    // (OBSOLETE) no parameters are
output as nodes
list("BULK", list("@sub")),                                       // If node BULK exists, and is
unconnected, the value
                                                                  // of parameter sub is output as
its node name
                list("PLUS", "MINUS", "BULK"),                    // Terminal order - this is
independent of pin number
                NULL,                                             // There are no power nodes
            NULL)), instP, NULL, NULL)                            // There are no substrate nodes
```

Where Used: **(ael)**

This function can be used within netlist callback functions. It should not be called directly, except by another netlist callback function, which should have appropriate arguments set for all of the parameters.

getcwd()

Returns a string representing the path name of the current working directory. This is almost always the path name of the workspace.

Syntax:

```
getcwd();
```

Example:

```
decl prjName;
prjName = getcwd();
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Batch Simulation
Layer/Process Manager
Split SP2 MDIF File

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

get_dir_files()

Returns an AEL list of files found in a specified directory. The suffix of the files to list may be specified.

Syntax:

```
get_dir_files(dirName [, suffix]);
```

where

dirName is a string that contains the directory name to list its files.

suffix is optional. A string that contains the suffix of the files to list.

Example:

```
decl ls;
ls = get_dir_files("/users/myhome/dbproject", "atf");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Layer/Process Manager
Scan Directory for Dirs

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

getenv()

Returns a string representing the configuration variable value. If the name of the configuration file is given, then the configuration variable values are obtained from the named environment file. If the name of the configuration variable is not specified, an application-specific default is used.

See also: *expandenv()* (ael), *setenv()* (ael).

Syntax:

```
getenv(name[, conf]);
```

where

name is the name of configuration variable.

conf is optional. Name of configuration file to use. In an attempt to locate the file, the program searches the following directories, in this order: workspace directory, home directory, custom directory, installation directory. The file name should be specified without a trailing *.cfg*. Entering a path for this file is not allowed.

Example:

```
decl val;
val = getenv("LAYERS_PATH");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Read Configuration File

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

geterror()

Returns a string that represents the last error message that occurred.

Syntax:

```
geterror();
```

Example:

```
decl a;
a = geterror();
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

getppid()

Returns an integer that represents the parent process ID of the current process.

Syntax:

```
getppid();
```

Example:

```
decl a;
a = getppid();
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

getsysenv()

Returns a string representing a system environment variable value.

Syntax:

```
getsysenv(name);
```

where

name is the name of a environment variable.

Example:

```
getsysenv("PRINTER");        //Return PRINTER value
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

identify_value()

Returns a single string representing the value(s) of any valid AEL object. The object can be of any type.

See also: *Database Overview* (ael).

Syntax:

```
identify_value(object);
```

where

object is any AEL object. The object can be of any type (such as list, handle, or variable), including complex lists (that is, lists containing other lists). List objects are enclosed in brackets [ ]. For non-printable objects, prints the name (such as handles).

Example:

```
// Shows 2 in an information dialog box
de_info(identify_value(2));
// Shows list ("a", "b", list(1,2.22,"c")) in an information dialog box
decl myList = list ("a", "b", list(1,2.22,"c"));
de_info(identify_value(myList));
// Sets myString to "12.3"
decl myString = identify_value(12.3);
de_info(identify_value(myString));
// Shows  in an information dialog box
// Window instances are not printable,
// so this just shows that it's not NULL.
de_info(api_get_current_window());
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Knowledge center website. You will need to register at this site with a valid support contract to download an example file.

Count Nr of Vertices on Selected Polygon

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

is_complex()

Identifies whether a given value is a complex number. Returns TRUE if a given value is a complex number, NULL if given value is not a complex number.

Syntax:

```
is_complex(value);
```

where

value is any value.

Example:

```
is_complex(10.0); // returns NULL
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

is_dir()

Determines if a given path is a directory. Returns TRUE if path is a directory, otherwise FALSE.

Syntax

```
is_dir(path);
```

where

path is a string that represents a path.

Example

```
decl a;
a = is_dir("/users/myhome/project");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Create Directory
Scan Directory for Dirs
Write Permission to Directory?

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

is_file()

Identifies whether a given value is a file identifier. Returns TRUE if a given value is a file identifier, NULL if given value is not a file identifier.

Syntax:

```
is_file(value);
```

where

value is any value.

Example:

```
decl f;
f = fopen("atest", "R");
is_file(f); // returns TRUE
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Get File Size
Get File Time Stamp
Write Permission to File?

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

is_function()

Identifies whether a given value is a function. Returns TRUE if a given value is a function, NULL if given value is not a function.

Syntax:

```
is_function(value);
```

where

value is any value.

Example:

```
is_function("abc"); // returns NULL
```

***Where Used* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

is_function_defined()

Identifies whether a given value is a defined function. Returns TRUE if a given value is a defined function, NULL if given value is not a defined function.

Syntax:

```
is_function_defined(value);
```

where

value is any string parameter.

Example:

```
if (is_function_defined("cos"))
 a = cos(1);
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

is_integer()

Identifies whether a given value is an integer number. Returns TRUE if a given value is an integer number, NULL if given value is not an integer.

Syntax:

```
is_integer(value);
```

where

value is any value.

Example:

```
is_integer(2.5); // returns NULL
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Copy String

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

is_list()

Identifies whether a given value is a list. Returns TRUE if a given value is a list, NULL if given value is not a list.

Syntax:

```
is_list(value);
```

where

value is any value.

Example:

```
decl list1;
list1 = list(1,2,3);
is_list(list1); // returns TRUE
is_list(4); // returns NULL
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Export all Variables
Reverse List
Set Variable Value 2

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

is_real()

Identifies whether a given value is a real number. Returns TRUE if a given value is a real number, NULL if given value is not a real number.

Syntax:

```
is_real(value);
```

where

value is any value.

Example:

```
is_real(12.4); // returns TRUE
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

is_string()

Identifies whether a given value is a string. Returns TRUE if a given value is a string, NULL if given value is not a string.

Syntax:

```
is_string(value);
```

where

value is any value.

Example:

```
is_string("abc"); // returns TRUE
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Copy String
Short File Name
Trim off Leading/Trailing Spaces

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

is_type()

Identifies whether a given value is of a given type. Returns TRUE if a given value is of a given type, NULL if given value does not match the type.

Syntax:

```
is_type(typeStr, value);
```

where

typeStr is the type to check the value against. The valid typeStr are: "int", "double", "complex", "list", "string", "file", "function", "vocabulary", "word".

value is any value.

Example:

```
is_type("int", 10); // returns TRUE
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

is_voc()

Identifies whether a given value is a vocabulary reference. Returns TRUE if a given value is a vocabulary reference, NULL if given value is not a vocabulary reference.

Syntax:

```
is_voc(value);
```

where

value is any value.

Example:

```
is_voc("not a voc"); // returns NULL
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

list_undefined()

Interprets a string as an AEL command and returns an AEL list of undefined identifiers in that command.

Syntax:

```
list_undefined(str);
```

where

str is a string that represents an AEL command(s).

Example:

```
decl a;
a = list_undefined("a = b + c + d;");
        // where c is not defined, returns list(c)
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

load()

Loads an AEL file and interprets its statements. If you load an AEL file from within another using *load()* , functions and variables defined in the loaded file cannot be accessed from the other file without a forward declaration preceding the call to *load()* . Use *decl name();* for forward declarations of variables and *defun name();* for forward declarations of functions.

AEL is always interpreted in the context of a vocabulary of definitions which are used to resolve identifier references. Whenever a variable or function name is used in the AEL program, the context vocabulary is searched for an existing definition. New variables and function definitions are added to the context vocabulary as well. The optional second argument allows the context to be overridden for this function by providing the name of the desired vocabulary.

In the design environment, two contexts are important: SimCmd and CmdOp. Generally SimCmd is the context for any AEL files loaded during program initialization and when opening a workspace. The context is CmdOp when interpreting menu commands and in the Command Line dialog box. SimCmd is a superset of CmdOp, so not all definitions in SimCmd are available from the Command Line dialog without specifying the context SimCmd specifically. Returns: none.

Syntax

```
load(aelFileName, [context]);
```
Where,

- *aelFileName* is the he name of an AEL file (without the extension).
- *context* is the name of the vocabulary context for interpreting the file.

Example

This example prints "I am a".

```
File a.ael:
defun func_a()
{
    fputs(stderr,"I am a");
}
Loading a.ael:
load("a");
func_a();
```

Download Example File

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Change Component Scope to GLOBAL
Count nr of Vertices on Selected Polygon

Split SP2 MDIF File

Where Used (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

mkdir()

Creates a directory. Returns TRUE if a directory is created; FALSE if a directory is not created.

Syntax:

```
mkdir(str);
```

where

str is a string that contains the name of the directory.

Example:

```
decl a;
a = mkdir("myDir");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Create Directory

Where Used: (ael)

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

num_args()

Returns the number of arguments passed into a function.

Syntax:

```
num_args();
```

Example:

```
defun test(a, b, c)
{
   fputs(stderr, num_args());
}
test(10);        // should print out 1
test(10, 15);    // should print out 2
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

offset_array()

Creates a new array by copying an existing AEL array except that the lower bound of a specified dimension is modified to a given value. Since this function does not work on data from datasets, ensure that the parameters passed are arrays.

See also: *resize_array()* (ael), *array_size()* (ael), *array_type()* (ael), *array_lowerBound()* (ael), *array_upperBound()* (ael), *convert_array()* (ael).

Syntax:

```
new = offset_array(<arr>, <dim>, <min>)
```

where

arr is a valid AEL array.

dim is an integer that specifies which dimension of <arr> will have its lower bound modified in the new array.

min is an integer that specifies what the lower bound of <dim> will be in the new array.

new is a new valid AEL array.

Example:

```
decl i,j;
       decl a = {{1,2,3},{4,5,6}};
       fputs(stderr, identify_value(a)); // outputs {{1,2,3},{4,5,6}}
       for (i=0;i<2;i++) // NOTE: access begins at 0
       for (j=0; j<3; j++)
```

```
        fputs(stderr,a[i,j]); // loop output is: 1 2 3 4 5 6
        decl b = resize_array(a,2,-1,3);
        fputs(stderr, identify_value(b));
        //outputs{{<int>,1,2,3,<int>},{<int>,4,5,6,<int>}}
        // where <int> is uninitialized integer.
        for (i=0;i<2;i++) // NOTE: access of 2nd dim begins at -1
        for (j=-1; j<4; j++)
        fputs(stderr,b[i,j]);
//loop output is:<int> 1 2 3 <int><int> 4 5 6 <int>
        decl b = offset_array(b,2,0);
        for (i=0;i<2;i++) // NOTE: access of 2nd dim now begins at 0
        for (j=0; j<5; j++)
        fputs(stderr,b[i,j]);
//loop output is:<int> 1 2 3 <int><int> 4 5 6 <int>
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

on_error()

Sets the AEL function to be called in case of an error. If the argument is omitted or is NULL, then the current error handler is removed. The address of the old error handler function is returned. If the argument is not an AEL function address, the function fails and reports an error.

When an error occurs, the error handler function is called and passes the following:

- Error code, an integer indicating line in message file. Common run time error codes for the class "AEL" are listed in *Common Run Time Error Codes*.
- Error class, a string indicating name of message file
- Operation code, an integer indicating the type of operation occurring. The AEL operation codes are listed in *AEL Operation Codes*.
- File name, a string identifying the file where error occurred.
- Line, an integer indicating the line number where error occurred (starting at 1).
- Column, an integer indicating the column number where error occurred (starting at 1).

The error handling function should return a value of NULL to ignore the error (after taking appropriate corrective action) or a non-zero integer value to allow AEL to continue error processing. Returns a function address.

See also: *error()* (ael).

Syntax:

```
on_error( [func] );
```

where

func is the address of error handling function.

Example:

```
defun report_error( code, class, op, line, col)
{
 fputs( stderr, strcat("Error occurred:", class, code, " at ", line,
"/", col));
 return TRUE;
}
on_error( report_error);
```

Common Run Time Error Codes

Code	Definition
300	Integer parameter value was required
301	Real parameter value was required
302	Two numeric parameters are required
303	String parameter value was required
304	Incorrect type of parameter value
305	Parameters cannot be matched
306	Stack reference to value not on stack
307	Wrong number of parameters passed
308	Numeric parameter expected
309	Function address missing for call
310	Function address not executable
311	Word reference parameter expected
312	Bad frame reference */
313	Integer divide by zero
314	Real divide by zero
315	Complex divide by zero
316	Illegal return value from compare function
317	Values not comparable
318	Log of negative number
319	Log of zero
320	Square root of negative number

AEL Operation Codes

Code	Definition	Code	Definition
0	discard return value	16	assign
1	logical OR	17	dereference word
2	logical AND	18	call function
3	logical NOT	19	call function, discard return value
4	logical test ==	20	return from function
5	logical test !=	21	branch on TRUE
6	logical test >=	25	bitwise AND
7	logical test <=	26	bitwise exclusive OR
8	logical test >	27	bitwise OR
9	logical test <	28	bitwise ones compliment
10	add	29	bitwise left shift
11	subtract	30	bitwise right shift
12	multiply	31	pre-increment
13	modulus divide	32	pre-decrement
14	divide	33	post-increment
15	negate	34	post-decrement

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

pcb_get_form_value()

Retrieves the value of a parameter that is a constant form. This function must be called within a PARM_MODIFIED_CB callback function created by *dm_create_cb()*.

See also: *pcb_get_mks()* (ael), *pcb_set_mks()* (ael), *pcb_set_form_value()* (ael), *pcb_get_string()* (ael), *pcb_set_string()* (ael).

Syntax:

```
pcb_get_form_value(callData, paramName)
```

where

callData is the third argument of the PARM_MODIFIED_CB callback function.

paramName is the name of the parameter to get the value from.

Example:

```
create_constant_form("first","first",0,"0","first");
create_constant_form("second","second",0,"1","second");
create_form_set("MyFormSet","first","second");
create_item(...
```

```
create_parm("A",
            "A parameter value",
            NULL,
            "MyFormSet",
            UNITLESS_UNIT,
            prm("first", ""),
            list(dm_create_cb(PARM_MODIFIED_CB, "a_modified_cb",
NULL, TRUE))),
create_parm("B",
            "B parameter value",
            NULL,
            "MyFormSet",
            UNITLESS_UNIT,
            prm("second", "")));
defun a_modified_cb(cbP, clientData, callData)
{
  decl a_formValue=pcb_get_form_value(callData, "A");
  decl b_formValue;
  if (strcmp(a_formValue, "first")==0)
      b_formValue="second";
  else
      b_formValue="first";
  return(pcb_set_form_value(NULL, "B", b_formValue));
}
```

Where Used: **(ael)**

Schematic

pcb_get_mks()

Retrieves the value of a parameter in MKS (unscaled) units. This function must be called within a PARM_MODIFIED_CB callback function created by *dm_create_cb()*.

See also: *pcb_set_mks()* (ael), *pcb_get_form_value()* (ael), *pcb_set_form_value()* (ael), *pcb_get_string()* (ael), *pcb_set_string()* (ael), *dm_create_cb()*.

Syntax:

```
pcb_get_mks(callData, paramName)
```

where

callData is the third argument of the PARM_MODIFIED_CB callback function.

paramName is the name of the parameter to get the value from.

Example:

```
create_item(...
  create_parm("A",
              "A parameter value",
              PARM_REAL | PARM_STATISTICAL | PARM_OPTIMIZABLE,
              "StdFormSet",
```

```
              0,
              prm("StdForm", "10.0"),
              list(dm_create_cb(PARM_MODIFIED_CB, "a_modified_cb",
NULL, TRUE))),
  create_parm("B",
              "B parameter value",
              PARM_REAL | PARM_STATISTICAL | PARM_OPTIMIZABLE,
              "StdFormSet",
              0,
              prm("StdForm", "20.0")));
defun a_modified_cb(cbP, clientData, callData)
{
   decl paramData=NULL;
   decl a_mks=pcb_get_mks(callData, "A");
   return(pcb_set_mks(paramData, "B", a_mks*2));
}
```

Where Used: **(ael)**

Schematic

pcb_get_parm_type()

Returns parameter type. This function must be called within a PARM_MODIFIED_CB callback function created by *dm_create_cb()*. Returns one of the following strings:

"NUMBER"
"EXPRESSION"
"CONSTANT_FORM"
"VARIABLE"
"UNKNOWN_FORM"
"UNKNOWN"
"STRING"
"DEFAULT_VALUE"
"DATA_FILE"
"ERROR"

See also: *pcb_get_mks()* (ael), *pcb_set_mks()* (ael), *pcb_get_form_value()* (ael), *pcb_set_form_value()* (ael), *pcb_get_string()* (ael), *pcb_set_string()* (ael).

Syntax:

```
pcb_get_parm_type(callData, paramName);
```

where

callData is the third argument of the PARM_MODIFIED_CB callback function.

paramName is the name of the parameter to get the type from.

Example:

```
defun a_modified_cb(cbP, clientData, callData)
{
     decl paramData=NULL;
     decl a_mks=pcb_get_mks(callData, "A");
     paramData=pcb_set_mks(paramData, "B", a_mks*2);
     return(pcb_set_mks(paramData, "C", a_mks*3));
}
```

***Where Used:* (ael)**

Schematic

pcb_get_string()

Retrieves the value of a parameter that is a constant form. This function must be called within a PARM_MODIFIED_CB callback function created by *dm_create_cb()*.

See also: *pcb_get_mks()* (ael), *pcb_set_mks()* (ael), *pcb_get_form_value()* (ael), *pcb_set_form_value()* (ael), *pcb_set_string()* (ael).

Syntax:

```
pcb_get_string(callData, paramName)
```

where

callData is the third argument of the PARM_MODIFIED_CB callback function.

paramName is the name of the parameter to get the value from.

Note
STRING_UNIT must be specified in create_parm for this function to work. See the following example:

Example:

```
create_item(...
 create_parm("A",
             "A parameter value",
             PARM_STRING,
             "StringAndReferenceFormSet",
             STRING_UNIT,
             prm("StringAndReference", "myAString"),
             list(dm_create_cb(PARM_MODIFIED_CB, "a_modified_cb",
NULL, TRUE))),
 create_parm("B",
             "B parameter value",
             PARM_STRING,
             "StringAndReferenceFormSet",
             STRING_UNIT,
             prm("StringAndReference", "myBString")));
defun a_modified_cb(cbP, clientData, callData)
{
   decl a_stringValue=pcb_get_string(callData, "A");
```

```
   return(pcb_set_string(NULL, "B", a_stringValue));
}
```

***Where Used:* (ael)**

Schematic

pcb_set_form_value()

Sets the value of a parameter that is a constant form. This function must be called within a PARM_MODIFIED_CB callback function created by *dm_create_cb()*. Returns data which must be returned by the PARM_MODIFIED_CB callback function.

See also: *pcb_get_mks()* (ael), *pcb_set_mks()* (ael), *pcb_get_form_value()* (ael), *pcb_get_string()* (ael), *pcb_set_string()* (ael).

Syntax:

```
pcb_set_form_value(paramData, paramName, value)
```

where

paramData is parameter data. NULL the first time. In addition to this variable being a parameter to this function, the value returned by this function must also be assigned to it.

paramName is the name of the parameter to set the value of.

value is the new value in MKS units.

Example:

```
create_constant_form("first","first",0,"0","first");
create_constant_form("second","second",0,"1","second");
create_form_set("MyFormSet","first","second");
create_item(...
 create_parm("A",
             "A parameter value",
             NULL,
             "MyFormSet",
             UNITLESS_UNIT,
             prm("first", ""),
             list(dm_create_cb(PARM_MODIFIED_CB, "a_modified_cb",
NULL, TRUE))),
 create_parm("B",
             "B parameter value",
             NULL,
             "MyFormSet",
             UNITLESS_UNIT,
             prm("second", "")));
defun a_modified_cb(cbP, clientData, callData)
{
   decl a_formValue=pcb_get_form_value(callData, "A");
```

```
   decl b_formValue;
   if (strcmp(a_formValue, "first")==0)
       b_formValue="second";
   else
       b_formValue="first";
   return(pcb_set_form_value(NULL, "B", b_formValue));
}
```

***Where Used:* (ael)**

Schematic

pcb_set_mks()

Sets the value of a parameter. The value must be in MKS (unscaled) units. This function must be called within a PARM_MODIFIED_CB callback function created by dm_create_cb(). Returns data which must be returned by the PARM_MODIFIED_CB callback function.

See also: *pcb_get_mks()* (ael), *pcb_get_form_value()* (ael), *pcb_set_form_value()* (ael), *pcb_get_string()* (ael), *pcb_set_string()* (ael).

Syntax:

```
pcb_set_mks(paramData, paramName, value)
```

where

paramData is parameter data. NULL the first time. In addition to this variable being a parameter to this function, the value returned by this function must also be assigned to it.

paramName is the name of the parameter to set the value of.

value is the new value in MKS units.

Example:

```
create_item(...
 create_parm("A",
             "A parameter value",
             PARM_REAL | PARM_STATISTICAL | PARM_OPTIMIZABLE,
             "StdFormSet",
             0,
             prm("StdForm", "10.0"),
             list(dm_create_cb(PARM_MODIFIED_CB, "a_modified_cb",
NULL, TRUE))),
 create_parm("B",
             "B parameter value",
             PARM_REAL | PARM_STATISTICAL | PARM_OPTIMIZABLE,
             "StdFormSet",
             0,
```

```
            prm("StdForm", "20.0")),
 create_parm("C",
             "C parameter value",
             PARM_REAL | PARM_STATISTICAL | PARM_OPTIMIZABLE,
             "StdFormSet",
             0,
             prm("StdForm", "30.0")));
defun a_modified_cb(cbP, clientData, callData)
{
   decl paramData=NULL;
   decl a_mks=pcb_get_mks(callData, "A");
   paramData=pcb_set_mks(paramData, "B", a_mks*2);
   return(pcb_set_mks(paramData, "C", a_mks*3));
}
```

***Where Used:* (ael)**

Schematic

pcb_set_string()

Sets the value of a parameter that is a constant form.This function must be called within a PARM_MODIFIED_CB callback function created by *dm_create_cb()*. Returns data which must be returned by the PARM_MODIFIED_CB callback function.

See also: *pcb_get_mks()* (ael), *pcb_set_mks()* (ael), *pcb_get_form_value()* (ael), *pcb_set_form_value()* (ael), pcb_set_string().

Syntax:

```
pcb_set_string(paramData, paramName, value)
```

where

paramData is parameter data. NULL the first time. In addition to this variable being a parameter to this function, the value returned by this function must also be assigned to it.

paramName is the name of the parameter to set the value of
value is the new value in string units

Example:

```
create_item(...
 create_parm("A",
             "A parameter value",
             PARM_STRING,
             "StringAndReferenceFormSet",
             STRING_UNIT,
             prm("StringAndReference", "myAString"),
             list(dm_create_cb(PARM_MODIFIED_CB, "a_modified_cb",
NULL, TRUE))),
 create_parm("B",
```

```
            "B parameter value",
            PARM_STRING,
            "StringAndReferenceFormSet",
            STRING_UNIT,
            prm("StringAndReference", "myBString")));
defun a_modified_cb(cbP, clientData, callData)
{
   decl a_stringValue=pcb_get_string(callData, "A");
   return(pcb_set_string(NULL, "B", a_stringValue));
}
```

Where Used: **(ael)**

Schematic

rename_word()

Finds an AEL word and renames it. Returns TRUE if rename took place, FALSE if an error occurred.

Syntax:

```
rename_word(orig, new [, voc]);
```

where

orig is a string that represents the original name of the word to be changed.

new is a string that represents the new name to be given to orig.

voc is optional. A string that represents what vocabulary to search for orig. If voc is not passed, the current vocabulary is searched. Note that the parent tree of the vocabulary is searched until orig is found.

Example:

```
decl a=10;
rename_word("a", "b");
            // a will no longer access a variable
            // the variable can now only be accessed with "b"
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

resize_array()

Creates a new array by copying an existing AEL array with resizing of the specified dimension. Note that if the dimension resize is bigger then the new elements of that dimension will be set to 0 or "". Since this function does not work on data from datasets, ensure that the parameters passed are arrays.

See also: *offset_array()* (ael), *array_size()* (ael), *array_type()* (ael), *array_lowerBound()* (ael), *array_upperBound()* (ael), *convert_array()* (ael).

Syntax:

```
new = resize_array(<arr>,<dim>,<min>,<max>)
```

where

arr is a valid AEL array.

dim is an integer that specifies which dimension of <arr> to resize in new array (ranges from 1 to inf).

min is an integer that specifies the lower bound of <dim> of new array.

max is an integer that specifies the upper bound of <dim> of new array.

new is a new valid AEL array.

Example:

```
decl a = {{1,2,3},{4,5,6}};
decl b = resize_array(a,1,0,5);
decl c = resize_array(a,2,0,1);
decl d = resize_array(a,2,-1,1);
decl e = resize_array(a,2,-1,3);
   fputs(stderr, identify_value(a)); // outputs {{1,2,3},{4,5,6}}
   fputs(stderr, identify_value(b));
   // outputs {{1,2,3},{4,5,6},{<int>,<int>,<int>},{<int>,<int>,<int>},
{<int>,<int>,<int>}}
   // where <int> is uninitialized integer
   fputs(stderr, identify_value(c)); // outputs {{1,2},{4,5}}
   fputs(stderr, identify_value(d)); // outputs {{<int>,1,2},{<int>,4,5}}
   fputs(stderr, identify_value(e)); // outputs {{<int>,1,2,3,<int>},
{<int>,4,5,6,<int>}}
```

Note
As demonstrated, you can add new elements to either the front or back of an existing array. In order to reset the offset back to 0 to max, see *offset_array()* (ael).

Where Used: **(ael)**

Schematic, Layout, Simulation, GUI

resize_array and the other functions that take an array as an input parameter cannot be used in Measurement Expressions (Data Display equations and Schematic MeasEqns)

setenv()

Sets the value of a named configuration variable. If the name of the configuration file is given, then the configuration variable values are obtained from the named environment file. If the name of the configuration variable is not specified, an application-specific default is used.
Returns: none.

See also: *expandenv()* (ael).

Syntax:

```
setenv(name, value[, conf, dir, savetodisk]);
```

where

name is the name of configuration variable.

value is a new value of variable.

conf is optional. Name of configuration file to use.

dir is optional. Integer value that specifies the directory where setenv preferences are saved, where

- ASTR_ENV_SAVE_CWD = current working directory
- ASTR_ENV_SAVE_HOME = home directory
- ASTR_ENV_SAVE_SYS = installation directory

savetodisk is optional. Integer value that controls whether the configuration is written to disk or not, where

- 0 = Do not write to disk
- 1 = Write to disk

Example:

```
setenv("LAYERS_PATH", "my_layers");
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Read Configuration File

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

sleep()

Sets an idle time for the program. Returns: None.

Syntax:

```
sleep(time);
```

where

time is the time to idle between next command executed, for seconds.

Example:

```
sleep(10); //sleep 10 seconds
```

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Remove Directory - rmdir()

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

start_timer()

Records the current timestamp internally. The recording is used when *total_elapsed_time()* is called. Returns: NULL.

Syntax:

```
start_timer();
```

Example:

```
start_timer();
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

system()

Executes a system command and returns the command exit status. The argument passes directly to the command interpreter. This function works on both UNIX and PC systems, but the execution and results of the command may be different.

Note
When you specify a path and command on the PC, the path can not contain spaces or the asterisk (*) or tilde (~) path completion characters. To workaround this limitation, add the path to your *Path* system variable.

To set your Path variable, select *Start* > *Settings* (Windows 2000 only) > *Control Panel* > *System* > *Advanced* > *Environment Variables* .

Syntax:

```
system(command[,mapStdErr,invokeDir,retValue]);
```

where

command is a system command. It can be a quoted string.

mapStdErr is an optional Boolean parameter (TRUE or FALSE). Specifies mapping stderr.

invokeDir is an optional string parameter. Specifies in which directory to invoke the command.

retValue is an optional Boolean parameter (TRUE or FALSE). Specifies whether to return the value of, and to specify to wait for, the command to finish before continuing AEL. If this is not passed or is FALSE, the return value is the program handle used by AEL, not useful to users.

Note
You cannot use redirect or pipes in command. Using a redirection or pipe causes an error.

Example:

```
system("mv abc def");    // invokes the mv system command
                                 // with the parameters abc and def
```

You can obtain a return value from a command executed in system() and AEL will not continue until the command has been completed.

The value returned is sent to stdout. If you are running a shell script and have an exit code, that exit code is not what is returned. Therefore, you must output to stdout any information you want returned to AEL.

Example:

This example sets "a" to a string representing a "ls" of the current directory.

```
decl a = system("ls",FALSE,"",TRUE);
```

Example:

This example sets "a" to a string representing a "ls" of /users/kla/pde directory.

```
decl a = system("ls",FALSE,"/users/kla/pde",TRUE);
```

Running Shell Scripts

Suppose xxx.ksh:

```
#! /bin/ksh
ls > tmp
cat tmp
exit 1
```

In Unix:

```
decl a = system("xxx.ksh", FALSE, "", TRUE);
decl a = system("ksh xxx.ksh", FALSE, "", TRUE);
```

Both lines return "ls" of current directory (because of cat command sending the contents of tmp to stdout).

In PC:

```
decl a = system("ksh xxx.ksh", FALSE, "", TRUE);
```

Same results. However, not that "ksh" must be placed in front of the shell script name.

Download Example File:

The following link(s) lead to the Agilent EEsof EDA Technical Support Documents and Examples website. You will need a valid login ID to view the contents of this site and download an example file.
Get File Permissions
Get File Size
Get File Time Stamp

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

tmpnam()

Generates the name of a unique temporary file. Returns a string that represents a unique temporary file.

Syntax:

```
tmpnam();
```

Example:

```
decl a;
a = tmpnam();    // a is a string that represents a unique temporary file
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

total_elapsed_time()

Returns the time elapsed since the last *start_timer()* was called as an AEL list of length 2. The first item of the list is an integer that represents the seconds elapsed. The second item of the list is an integer that represents the microseconds elapsed. If an error occurs (like *start_timer()* was not previously called), then NULL is returned.

Syntax:

```
total_elapsed_time();
```

Example:

```
decl time, sec, usec;
start_timer();
.....
time = total_elapsed_time();
sec = nth(0,time);
usec = nth(1,time);
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

validate_name()

Verifies that a string is a valid program file name; that is, that the string has no illegal punctuation characters or embedded spaces. Returns an integer indicating validity of file name, where: 0 = invalid, 1 = valid.

Syntax:

```
validate_name(string);
```

where

string is a string representing a file name.

Example:

Most often, this function is used in conjunction with *create_text_form()* to ensure that the file name entered is valid. In this example, the validate_name function will be called automatically to validate the value for the parameter that uses a text form, when the value is entered.

```
create_text_form("meas","Measurements", 0, "%v", "%v", get_measurement_list,
validate_name);
```

Where Used: **(ael)**

Schematic

warning()

Issues a warning message which is then printed in the warning dialog. Returns: none.

Syntax:

```
warning(errorFileName, lineNum, defaultMsg, string);
```

where

errorFileName is the name of an error file containing error messages.

lineNum is an integer, line number in the error file for this warning message.

defaultMsg is the default message string to use if error file or message not found.

string is the string to print out at end of message.

Example:

```
decl elem;
warning("art", 180, fmt_tokens(list(elem, "W1, W2, W3 or W4< W/2")), NULL);
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

what_col()

Returns the column number of the caller of the current function as an integer.

Syntax:

```
what_col([call_depth]);
```

where

call_depth is optional. An integer that instructs the function to return the column number of the function at the specified call_depth. If not supplied, call_depth is defaulted to 1.

Example:

File x.ael:

```
defun funct()
{    // refers to funct() being called
     fputs(stdout,what_col());            // prints 1
     // refers to what_col() being called
     fputs(stdout,what_col(0));           // 14
     // refers to funct() being called
     fputs(stdout,what_col(1));           // prints 1
     // refers to x.ael being called (or loaded)
     fputs(stdout,what_col(2));           // prints 1
}
funct();
     // refers to x.ael being called (or loaded)
fputs(stdout,what_col());           // prints 1
```

Where Used: **(ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

what_file()

Returns the file name of the caller of the current function as a string when the optional

argument, *call_depth()*, is supplied.

Syntax:

```
what_file([call_depth]);
```

where

call_depth is optional; default = 1. Instructs the function to return the file name of the file containing the calling function.

Example:

File x.ael:

```
defun funct()
{    // refers to funct() being called
     fputs(stdout,what_file());            // prints ./x.ael
     // refers to what_file() being called
     fputs(stdout,what_file(0));           // prints ./x.ael
     // refers to funct() being called
     fputs(stdout,what_file(1));           // prints ./x.ael
     // refers to x.ael being called (or loaded)
     fputs(stdout,what_file(2));           // prints ""
}
funct();
     // refers to x.ael being called (or loaded)
fputs(stdout,what_file());           // prints ""
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

what_function()

Takes an optional integer parameter to indicate what level of the function stack to extract.

Syntax:

```
what_function([level]);
```

where

level is optional; default = 1. Instructs the function which level of the function stack to extract.

Example:

```
defun x()
{
     fputs(stdout,what_function(0));           // prints out "what_function"
     fputs(stdout,what_function(1));           // prints out "x"
     fputs(stdout,what_function(2));           // prints "", indicating main
     fputs(stdout,what_function(3));           // ERROR
}
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

what_line()

Returns the line number of the caller of the current function as an integer.

Syntax:

```
what_line([call_depth]);
```

where

call_depth is optional; default = 1. Instructs the function to return the line number of the caller of the function at that call_depth.

Example:

File x,ael:

```
defun funct()
{    // refers to funct() being called
     fputs(stdout,what_line());            // prints 20
     // refers to what_line() being called
     fputs(stdout,what_line(0));           // 9
     // refers to funct() being called
     fputs(stdout,what_line(1));           // prints 20
     // refers to x.ael being called (or loaded)
     fputs(stdout,what_line(2));           // prints 1
}
funct();
     // refers to x.ael being called (or loaded)
fputs(stdout,what_line());           // prints 1
```

***Where Used:* (ael)**

Measurement Expressions (Data Display equations and Schematic MeasEqns), Schematic, Layout, Simulation, GUI

AEL Functions (alphabetical)

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

(For category-wise list of Functions, *click here* (ael))

A

Name	Type and Description
abs() Function (ael)	**Math Functions**: Returns the absolute value of a real number or an integer.
acos() Function (ael)	**Math Functions**: Returns the inverse cosine, in radians, of a real number or complex number.
acosh() Function (ael)	**Math Functions**: Returns the inverse hyperbolic cosine of an integer, real, or complex number.
acot() Function (ael)	**Math Functions**: Returns the inverse hyperbolic cosine of an integer, real, or complex number.
acoth() Function (ael)	**Math Functions**: Returns the inverse hyperbolic cotangent of an integer, real, or complex number.
add_menu() (ael)	**User Interface Function**: Adds an item, menuItemName, to the user-defined menu and associates the AEL function aelFuncName with the item.
add_separator() (ael)	**User Interface Function**: Adds a separator in the menu.
ael_file_exists() (ael)	**Utility Functions**: Returns TRUE if the given file path string or directory path string exists. Returns FALSE if the given file path string or directory path string does not exist.
ael_gfile_hasext() (ael)	**File Handling Functions**: Takes a string parameter and returns an integer that indicates where an extension exists in that string.
ael_is_file_writable() (ael)	**Utility Functions**: Returns TRUE if the given directory path string or file path string is writable. Returns FALSE if the given directory path string or file path string does not have write permissions.
api_dlg_unmanage() (ael)	**User Interface Function**: Unmanages the specified dialog.
api_get_current_window() (ael)	**User Interface Function**: Returns the handle to the currently active window.
api_get_graph_color_index_by_name() (ael)	**User Interface Function**: Returns the index of the color for the given color name
api_get_graph_color_name_by_index() (ael)	**User Interface Function**: Returns the name of the color for the given color index
api_get_graph_pattern_index_by_name() (ael)	**User Interface Function**: Returns the index of the pattern for the given pattern name
api_get_graph_pattern_name_by_index() (ael)	**User Interface Function**: Returns the name of the pattern for the given pattern index
api_get_window_graph() (ael)	**User Interface Function**: Returns a handle to the graphics area of the given window.
api_set_current_window() (ael)	**User Interface Function**: Sets a window to be currently active.
api_set_current_window_by_seq_num() (ael)	**User Interface Function**: Makes the window instance having the given sequence number as the current window.
append() (ael)	**List Management Functions**: Appends a new list to the end of an existing list.
arg() (ael)	**Utility Functions**: Returns the nth argument passed into a function.

arg_list() (ael)	**Utility Functions**: Returns the arguments passed to a function as a list.
array_lowerBound() (ael)	**Utility Functions**: Returns the lower bound of the specified dimension in an AEL array.
array_size() (ael)	**Utility Functions**: Returns the size of each dimension in an AEL array.
array_type() (ael)	**Utility Functions**: Returns the type of elements in an AEL array.
array_upperBound() (ael)	**Utility Functions**: Returns the upper bound of the specified dimension in an AEL array.
asin() Function (ael)	**Math Functions**: Returns the inverse sine, in radians, of a real number, an integer, or complex number.
asinh() Function (ael)	**Math Functions**: Returns the inverse hyperbolic sine of an integer, real, or complex number.
atan() Function (ael)	**Math Functions**: Returns the inverse tangent, in radians, of a real number, ±?/2, an integer, or complex number.
atan2() Function (ael)	**Math Functions**: Returns the arc tangent of the rectangular coordinates y and x.
atanh() Function (ael)	**Math Functions**: Returns the inverse hyperbolic tangent of an integer, real, or complex number.

C

Name	Type and Description
call() (ael)	**Utility Functions**: Invokes or calls a function, gives a function value and an argument list (as returned from arglist() or list()).
call_depth() (ael)	**Utility Functions**: Returns the depth of the active call stack.
car() (ael)	**List Management Functions**: Returns the first item from a given list.
cdr() (ael)	**List Management Functions**: Returns the remainder of the list, after the first item is removed.
ceil() Function (ael)	**Math Functions**: Given a real number, returns the smallest integer not less than its argument
chdir() (ael)	**File Handling Functions**: Changes the current directory to the specified directory.
check_syntax() (ael)	**Utility Functions**: Interprets a string as an AEL command and returns whether or not the syntax is valid.
check_user_menu() (ael)	**User Interface Function**: This function can be used to determine if a user menu is being used by another application.
chmod() (ael)	**Utility Functions**: Changes the mode of the specified file.
chr() Function (ael)	**Math Functions**: Returns the character representation of an integer.
cint() Function (ael)	**Math Functions**: Given a noninteger real number, returns a rounded integer value.
clear_server() (ael)	**Simulator Command Functions**: Clears the busy status state of the active server (simulator).
clear_user_menu() (ael)	**User Interface Function**: Removes and clears all entries in the user-defined menu.
cmplx() Function (ael)	**Math Functions**: Given two real numbers representing the real and imaginary components of a complex number, returns a complex number.
conj() Function (ael)	**Math Functions**: Returns the conjugate of a complex number.
cons() (ael)	**List Management Functions**: Constructs a list cell from the member value to be returned by car() and the next element value to be returned by cdr().
convBin() Function (ael)	**Math Functions**: Returns a binary string of an integer with n-digits.
convert_array() (ael)	**Utility Functions**: Returns a new AEL array created from an existing array, but with the elements of the existing array converted to the specified type.
convHex() Function (ael)	**Math Functions**: Returns a hexidecimal string of an integer with n-digits.
convOct() Function (ael)	**Math Functions**: Returns an octal string of an integer with n-digits
cos() Function (ael)	**Math Functions**: Returns the cosine of a real number (in radians).
cosh() Function (ael)	**Math Functions**: Returns the hyperbolic cosine of an integer, real, or complex number.
cot() Function (ael)	**Math Functions**: Returns the cotangent of an integer, real, or complex number.
coth() Function (ael)	**Math Functions**: Returns the hyperbolic cotangent of an integer, real, or complex number.
create_compound_form() (ael)	**Component Definition Functions**: Creates a new compound form and stores the form in the dictionary for the current simulator type.
create_constant_form() (ael)	**Component Definition Functions**: Creates a new constant form and stores the form in the dictionary for the current simulator type.
create_form_set() (ael)	**Component Definition Functions**: Creates a set of forms for use by the create_parm() function.
create_item() (ael)	**Component Definition Functions**: Creates a new component definition and stores it with the current dictionary.
create_parm() (ael)	**Component Definition Functions**: Creates a parameter definition for a component.
create_server() (ael)	**Simulator Command Functions**: Creates a handle to a server process that can be controlled through AEL.
create_text_form() (ael)	**Component Definition Functions**: Creates a form whose set of possible values are a set of strings.

D

Name	Type and Description
dB() Function (ael)	**Math Functions**: Converts a voltage ratio to decibels.
db_add_arc() (ael)	**Command Functions**: Adds a clockwise or counterclockwise circular arc to a polygon or polyline under construction in the given design context.
db_add_arc1() (ael)	**Command Functions**: Creates a standalone arc in the given design context.
db_add_arc2() (ael)	**Command Functions**: Creates a clockwise or counterclockwise standalone arc in the given design context.
db_add_arc3() (ael)	**Command Functions**: Creates a clockwise or counterclockwise standalone arc in the given design context.
db_add_arc4() (ael)	**Command Functions**: Creates a clockwise or counterclockwise standalone arc in the given design context, that is created with a user defined starting point coordinate, center point coordinate, and a chord length for an arc angle expressed in terms of circumference.
db_add_circle() (ael)	**Command Functions**: Adds a circle in the given design context in user-defined units.
db_add_construction_line() (ael)	**Command Functions**: Adds a construction line to the given design context using user-defined units.
db_add_hole_to_primitive_polygon() (ael)	**Primitive Polygon Functions**: Returns a new primitive polygon of a given primitive polygon with a given hole polygon added into it.
db_add_path() (ael)	**Command Functions**: Starts a path command sequence. Adds a path to the given design context.
db_add_point() (ael)	**Command Functions**: Adds a construction line to the given design context using user-defined units.
db_add_polygon() (ael)	**Command Functions**: Adds a point to a polygon, polyline, or path using user units
db_add_polyline() (ael)	**Command Functions**: Starts a polygon command sequence.
db_add_rectangle() (ael)	**Command Functions**: Adds a rectangle to the given design context.
db_add_property() (ael)	**Property Functions**: Adds a property to an ADS database object.
db_clear_context() (ael)	**Design Context Functions**: Clears everything in a given design context.
db_clear_map() (ael)	**Database Query and Manipulation Functions**: Clears the transformation mapping established by db_setup_transform(), db_set_map(), or db_setup_map().
db_commit_transaction() (ael)	**Transaction Functions**: Commit the given transaction.
db_copy_context() (ael)	**Design Context Functions**: Copies the shapes, pins, application objects, instances, and properties from a given source design context into a given destination design context.
db_create_hierarchy_context() (ael)	**HierarchyContext Functions**: create a HierarchyContext for a given design with a given descent policy.
db_create_inst_iter() (ael)	**Instance Iterator Functions**: Returns an iterator to

	the first instance belonging to a given design context.
db_create_inst_pin_iter() (ael)	**InstPin Iterator Functions**: Returns an instance pin iterator of the given instance or of InstPins connected to the given Net.
db_create_inst_term_iter() (ael)	**InstTerm Iterator Functions**: Returns an instance terminal iterator from a given Net or Instance object.
db_create_layer() (ael)	**Layer and LayerId Functions**: Creates a new physical layer for the given design context, given the layer's name and number.
db_create_net_iter() (ael)	**Net Iterator Functions**: Returns an iterator to the first Net belonging to a given design context.
db_create_param_iter() (ael)	**Parameter Iterator Functions**: Returns an iterator to the first parameter in the collection.
db_create_pin() (ael)	**Pin Functions**: Creates a new pin in a given context and returns the new pin object.
db_create_pin_iter() (ael)	**Pin Iterator Functions**: Returns a pin iterator from a given Net or DesignContext.
db_create_primitive_polygon() (ael)	**Primitive Polygon Functions**: Returns a new primitive polygon from a given list of outer boundary points, and a given optional list of point index to edge arc bulge factors.
db_create_prop_iter() (ael)	**Property Iterator Functions**: Returns an iterator to the first property of the given database object.
db_create_shape_from_primitive_polygon() (ael)	**Primitive Polygon Functions**: Creates and returns a new database shape in a given context on a given layerId from a given source primitive polygon.
db_create_shape_iter() (ael)	**Shape Iterator Functions**: Returns an iterator to the first shape belonging to a given design context.
db_create_transaction() (ael)	**Transaction Functions**: Create a new transaction object for a given DesignContext.
db_cycle_select_item() (ael)	**Selection Functions**: Select one item at a given location in the DesignContext.
db_deselect_all() (ael)	**Selection Functions**: Deselect all items in a DesignContext.
db_deselect_area() (ael)	**Selection Functions**: Deselect all items in a given rectangular area of a DesignContext.
db_design_is_modified() (ael)	**DesignContext Functions**: Returns TRUE if the given design context has been modified, otherwise returns FALSE.
db_edit_pin_attributes() (ael)	**Pin Functions**: Returns the modified pin for a given pin and the given modified pin attributes.
db_end() (ael)	**Command Functions**: Completes a polygon, polyline, wire or trace command sequence.
db_evaluate_inst_param_value() (ael)	**ExpressionContext Functions**: Returns the evaluated expression value for an instance's parameter with a given parameter name within the scope of the given ExpressionContext.
db_evaluate_inst_param_value_no_expr() (ael)	**ExpressionContext Functions**: Evaluates an instance parameter value with the given parameter name without expression evaluation.
db_evaluate_param_expression() (ael)	**ExpressionContext Functions**: Evaluate an expression in a given expression context.
db_evaluate_param_value() (ael)	**ExpressionContext Functions**: Returns the evaluated expression value from a given parameter within the scope of the given ExpressionContext.
db_evaluate_param_value_no_expr() (ael)	**ExpressionContext Functions**: Evaluates an instance parameter value with the given parameter

	name without expression evaluation.
db_factor() (ael)	**Design Environment Query Functions**: Returns a real value, a conversion factor from layout user units to meters.
db_find_instance_ex() (ael)	**Database Query and Manipulation Functions**: Finds and returns an instance with a given instance name in the given DesignContext.
db_find_inst_param_by_name() (ael)	**Parameter Iterator Functions**: Returns a new parameter iterator for the parameter with the given parameter name within the given parameter iterator's list of parameters.
db_find_layerid_by_name() (ael)	**Layer and LayerId Functions**: Returns a LayerId of a given design context that has a given LayerId name.
db_find_layer_name_by_number() (ael)	**Layer and LayerId Functions**: Finds and returns the name of a layer in a design context with the given layer number.
db_find_layer_number_by_name() (ael)	**Layer and LayerId Functions**: Finds and returns the number of a layer in a design context with the given layer name.
db_find_layout_layerid_by_name() (ael)	**Layer and LayerId Functions**: Finds and returns the layout LayerId for the given design context from the given LayerId name.
db_first_parm() (ael)	**Database Query and Manipulation Functions**: Returns a handle to a parameter or instance identified by an instance handle or the first parameter of a list of parameters identified by a parameter handle.
db_get_appropriate_base_name() (ael)	**Design Functions**: Returns a suitable filename for the given design context.
db_get_appropriate_base_name_from_design_name() (ael)	**Design Functions**: Returns a suitable filename from the given design name or component name.
db_get_arc_angle() (ael)	**Shape Functions**: Returns the sweep angle of a given arc shape.
db_get_arc_center() (ael)	**Shape Functions**: Returns the center coordinate of a given arc shape.
db_get_arc_start() (ael)	**Shape Functions**: Returns the start coordinate of a given arc shape.
db_get_bbox_x1() (ael)	**Database Query and Manipulation Functions**: Returns an integer, the x component of the lower-left corner of an object's bounding box.
db_get_bbox_x2() (ael)	**Database Query and Manipulation Functions**: Returns an integer, the x component of the upper-right corner of an object's bounding box.
db_get_bbox_y1() (ael)	**Database Query and Manipulation Functions**: Returns an integer, the y component of the lower-left corner of an object's bounding box.
db_get_bbox_y2() (ael)	**Database Query and Manipulation Functions**: Returns an integer, the y component of the upper-right corner of an object's bounding box.
db_get_cell_name() (ael)	**Design Context Functions**: Returns the string cell name of a design context.
db_get_component_name() (ael)	**Database Query and Manipulation Functions**: Returns a string component name of a given DesignContext.
db_get_const_line_layerid() (ael)	**Construction Line Functions**: Returns the LayerId for the given construction line object.
db_get_context_bbox() (ael)	**Design Context Functions**: Returns the bounding

	box of the given DesignContext.
db_get_context_db_factor() (ael)	**Design Context Functions**: Returns a real value conversion factor for converting user units to database units for a given DesignContext.
db_get_context_unit_name() (ael)	**Design Context Functions**: Returns the string unit name used by the given DesignContext.
db_get_controller_design_name() (ael)	**Simulation Functions**: Returns the string design name for the controller design of the given design context.
db_get_datadisp_name() (ael)	**Simulation Functions**: Returns the string (DDS) Data Display file name for the given design context.
db_get_dataset_name() (ael)	**Simulation Functions**: Returns the string dataset name for the given design context.
db_get_dbu_to_uu_factor() (ael)	**Design Context Functions**: Returns the real value conversion factor for converting database units to user units for a given DesignContext.
db_get_design_context_from_hierarchy() (ael)	**HierarchyContext Functions**: Returns the DesignContext from the given HierarchyContext.
db_get_design_name_for_instance_in_hierarchy() (ael)	**HierarchyContext Functions**: Returns the sub-design name for a given instance in the given HierarchyContext.
db_get_design_name() (ael)	**Database Query and Manipulation Functions**: Returns a string design name of a given DesignContext.
db_get_design_type() (ael)	**Database Query and Manipulation Functions**: Returns an integer design type of a given DesignContext.
db_get_edge_arc_angle() (ael)	**Primitive Polygon Functions**: Returns the arc sweep angle for a given primitive polygon's given point index of an edge that is an arc.
db_get_edge_arc_bulge() (ael)	**Primitive Polygon Functions**: Returns the arc bulge factor for a given primitive polygon given point index of an edge.
db_get_edge_arc_center() (ael)	**Primitive Polygon Functions**: Returns the center coordinate point for a given primitive polygon's given point index of an edge that is an arc.
db_get_ellipse_center() (ael)	**Shape Functions**: Returns the center coordinate of a given arc shape.
db_get_ellipse_x_radius() (ael)	**Shape Functions**: Returns the x radius of a given ellipse shape.
db_get_ellipse_y_radius() (ael)	**Shape Functions**: Returns the y radius of a given ellipse shape.
db_get_hierarchy_context_for_instance() (ael)	**HierarchyContext Functions**: Returns the HierarchyContext for the given instance’s sub-design in the given HierarchyContext.
db_get_hierarchy_context_for_parent() (ael)	**HierarchyContext Functions**: Returns the HierarchyContext for the parent of the current design in the given HierarchyContext.
db_get_hierarchy_policy_name() (ael)	**Simulation Functions**: Returns the string name of the HierarchyPolicy that will be used for simulation of the given design context.
db_get_instance_bbox() (ael)	**Database Query and Manipulation Functions**: Returns a handle to the bounding box of an instance.
db_get_instance_cell_name() (ael)	**Instance Functions**: Returns the string cell name of the given instance.
db_get_instance_component_name() (ael)	**Database Query and Manipulation Functions**: Returns a string component name of a given instance.

db_get_instance_description() (ael)	**Database Query and Manipulation Functions**: Returns a string, the AEL component description string for an instance.
db_get_instance_design_name() (ael)	**Instance Functions**: Returns the string design name of the instance's master design. The design name string value returned from this function is in the format "<library name>:<cell name>:<view name>".
db_get_instance_item_definition() (ael)	**Database Query and Manipulation Functions**: Returns an item definition for a given instance.
db_get_instance_library_name() (ael)	**Instance Functions**: Returns the string library name of the given instance.
db_get_instance_name() (ael)	**Database Query and Manipulation Functions**: Returns a string instance name of a given instance.
db_get_instance_owner_design() (ael)	**Instance Functions**: Returns a design context to the instance's owner design. For a PSN, this is the Master design.
db_get_instance_parm() (ael)	**Database Query and Manipulation Functions**: Returns a real number, the nominal value of an instance parameter given the parameter name or index (position in list of parameters).
db_get_instance_placement_transform() (ael)	**Instance Functions**: Returns the transform of the given instance.
db_get_instance_special() (ael)	**Instance Functions**: Returns the special flag code of an instance.
db_get_instance_subcontext_transform() (ael)	**Instance Functions**: Returns the subcontext transform of the given instance.
db_get_inst_pin_angle_normalized() (ael)	**InstPin Functions**: Returns the angle of a given InstPin, normalized (zero degrees is right).
db_get_inst_pin_bbox() (ael)	**InstPin Functions**: Returns the bounding box of the given instance pin.
db_get_inst_pin_instance() (ael)	**InstPin Functions**: Returns the instance of the given instance pin.
db_get_inst_pin_inst_term() (ael)	**InstPin Functions**: Returns the InstTerm (instance terminal) of the given InstPin (instance pin) object.
db_get_inst_pin_net() (ael)	**InstPin Functions**: Returns the Net connected to the given instance pin.
db_get_inst_pin_snap_layerid() (ael)	**InstPin Functions**: Returns the LayerId for the snap layer of an instance pin.
db_get_inst_pin_snap_point() (ael)	**InstPin Functions**: Returns the snap coordinate point of a given instance pin object.
db_get_inst_pin_term_number() (ael)	**InstPin Functions**: Returns the term number of a given Instance Pin.
db_get_inst_pin_term_type() (ael)	**InstPin Functions**: Returns an integer terminal type of a given instance pin object.
db_get_inst_term_instance() (ael)	**InstTerm Functions**: Returns the instance of a given instance terminal.
db_get_inst_term_name() (ael)	**InstTerm Functions**: Returns the name of a given instance terminal.
db_get_inst_term_net() (ael)	**InstTerm Functions**: Returns the Net of a given instance terminal.
db_get_inst_term_number() (ael)	**InstTerm Functions**: Returns the number of a given instance terminal.
db_get_inst_term_type() (ael)	**InstTerm Functions**: Returns an integer terminal type of a given instance term object.
db_get_item_definition() (ael)	**Database Query and Manipulation Functions**: Returns an item definition for a given DesignContext.

db_get_layerid() (ael)	**Layer and LayerId Functions**: Returns a new LayerId object of a given layer/purpose pair, specified by their layer name and purpose name, of a given design context.
db_get_layerid_alpha() (ael)	**Layer and LayerId Functions**: Returns the alpha integer value for a given LayerId.
db_get_layerid_fill_mode() (ael)	**Layer and LayerId Functions**: Returns the fill mode integer identifier for a given LayerId.
db_get_layerid_fill_pattern() (ael)	**Layer and LayerId Functions**: Returns the fill Patern integer identifier for a given LayerId.
db_get_layerid_for_comp_name() (ael)	**Layer and LayerId Functions**: Returns the LayerId for the component name annotation used for component instances from a given design context.
db_get_layerid_for_inst_name() (ael)	**Layer and LayerId Functions**: Returns the LayerId for the instance name annotation used for component instances from a given design context.
db_get_layerid_for_parameters() (ael)	**Layer and LayerId Functions**: Returns the LayerId for the parameter annotation used for component instances from a given design context.
db_get_layerid_for_schematic_wires() (ael)	**Layer and LayerId Functions**: Returns the LayerId for the schematic wires from a given schematic design context.
db_get_layerid_for_symbol_body() (ael)	**Layer and LayerId Functions**: Returns the LayerId for the symbol body from a given design context.
db_get_layerid_for_symbol_text() (ael)	**Layer and LayerId Functions**: Returns the LayerId for the symbol text from a given design context.
db_get_layerid_index() (ael)	**Layer and LayerId Functions**: Returns the LayerId integer index into the design context's layer list given a LayerId.
db_get_layerid_line_style() (ael)	**Layer and LayerId Functions**: Returns the line style integer identifier for a given LayerId.
db_get_layerid_name() (ael)	**Layer and LayerId Functions**: Returns a LayerId name for a specified LayerId within a given design context.
db_get_layerid_names() (ael)	**Layer and LayerId Functions**: Returns an ordered list of LayerId names for a given design context.
db_get_layerid_rgb() (ael)	**Layer and LayerId Functions**: Returns the RGB color integer value for a given LayerId.
db_get_layer_binding() (ael)	**Layer and LayerId Functions**: Returns the layer binding string for a given layer number in the given design context's library's technology.
db_get_layer_number() (ael)	**Layer and LayerId Functions**: Returns the layer number for a given LayerId.
db_get_layer_process_role() (ael)	**Layer and LayerId Functions**: Returns the layer process role for a given layer.
db_get_layout_layerid_name() (ael)	**Layer and LayerId Functions**: Returns the layout LayerId name for the given design context from the given LayerId.
db_get_layout_layerid_names() (ael)	**Layer and LayerId Functions**: Returns an ordered list of the layout LayerId names for the given design context.
db_get_library_name() (ael)	**Design Context Functions**: Returns the library name of a design context.
db_get_location_angle() (ael)	**Database Query and Manipulation Functions**: Returns an integer, the angle given a location handle in .001 of a degree.
db_get_location_x() (ael)	**Database Query and Manipulation Functions**:

	Returns an integer, the x position given a location handle.
db_get_location_y() (ael)	**Database Query and Manipulation Functions**: Returns an integer, the y position given a location handle.
db_get_map() (ael)	**Database Query and Manipulation Functions**: Returns a handle to the current transformation mapping.
db_get_map_attribute() (ael)	**Database Query and Manipulation Functions**: Returns an attribute of a transformation mapping, as retrieved by db_get_map() .
db_get_net_name() (ael)	**Net Functions**: Returns the name of the given Net.
db_get_node_wires() (ael)	**Database Query and Manipulation Functions**: Returns a handle to the segment representing a wire for a given node.
db_get_objects_selected_at() (ael)	**Selection Functions**: Returns a list of shape objects, instance objects, and/or port objects that are selected at a particular location in a DesignContext.
db_get_opendds_option() (ael)	**Simulation Functions**: Returns TRUE if the given design context is specified to automatically open the Data Display after simulation. Returns FALSE otherwise.
db_get_param_form_name() (ael)	**Parameter Value Functions**: Returns the string form name of the given parameter value's form.
db_get_param_name() (ael)	**Parameter Value Functions**: Returns the string parameter name for the given parameter value.
db_get_param_string_value() (ael)	**Parameter Value Functions**: Returns the string value of the given parameter value.
db_get_param_value_code() (ael)	**Parameter Value Functions**: Returns an integer parameter value code that represent's the parameter value form's class type.
db_get_parent_inst_name_in_hierarchy() (ael)	**HierarchyContext Functions**: Returns the instance name of the parent instance of the current design in the given HierarchyContext.
db_get_parm_attribute() (ael)	**Database Query and Manipulation Functions**: Returns an attribute of a parameter.
db_get_path_trace_bend_type() (ael)	**Database Query and Manipulation Functions**: Returns the corner (bend) type for a given path, trace, or wire shape.
db_get_path_trace_miter_radius() (ael)	**Database Query and Manipulation Functions**: Returns the miter cutoff percentage or curve radius for a given path, trace, or wire shape.
db_get_path_trace_width() (ael)	**Database Query and Manipulation Functions**: Returns the width in database units for a given path, trace, or wire shape.
db_get_pin_angle() (ael)	**Pin Functions**: Returns the angle of a given pin.
db_get_pin_angle_normalized() (ael)	**Pin Functions**: Returns the angle of a given Pin, normalized (zero degrees is right).
db_get_pin_bbox() (ael)	**Pin Functions**: Returns the bounding box of a given pin.
db_get_pin_name() (ael)	**Pin Functions**: Returns the name of a given Pin object.
db_get_pin_term() (ael)	**Pin Functions**: Returns the term of a given Pin object.
db_get_pin_term_name() (ael)	**Pin Functions**: Returns the term name of a given Pin object's term.
db_get_pin_term_number() (ael)	**Pin Functions**: Returns the term number of a given

	Pin object's term.
db_get_pin_term_type() (ael)	**Pin Functions**: Returns an integer terminal type of a given Pin object.
db_get_pin_snap_layerid() (ael)	**Pin Functions**: Returns the LayerId for the snap layer of a pin.
db_get_pin_snap_point() (ael)	**Pin Functions**: Returns the snap coordinate point of a Pin object's.
db_get_primitive_polygon_edge_is_arc() (ael)	**Primitive Polygon Functions**: Returns TRUE if the edge following the given point index in a given primitive polygon is an arc segment; returns FALSE otherwise.
db_get_primitive_polygon_hole() (ael)	**Primitive Polygon Functions**: Returns a new primitive polygon equivalent to the specified hole at a specified hole index in the given primitive polygon.
db_get_primitive_polygon_num_holes() (ael)	**Primitive Polygon Functions**: Returns the integer number of holes in the given primitive polygon.
db_get_primitive_polygon_num_points() (ael)	**Primitive Polygon Functions**: Returns the number of points in the outer boundary of the polygon.
db_get_primitive_polygon_outer() (ael)	**Primitive Polygon Functions**: Returns an outer primitive polygon of a given primitive polygon.
db_get_primitive_polygon_point() (ael)	**Primitive Polygon Functions**: Returns the coordinate point at a specified point index in the given primitive polygon.
db_get_primitive_polygon_with_arcs_removed() (ael)	**Primitive Polygon Functions**: Returns a new primitive polygon with arcs removed from a given primitive polygon and given arc resolution to use for arc removal.
db_get_primitive_polygon_with_linked_holes() (ael)	**Primitive Polygon Functions**: Returns a new primitive polygon with holes linked from a given primitive polygon.
db_get_property_as_string() (ael)	**Property Functions**: Get the property's double, string, or long value returned as a string value from an ADS database object.
db_get_property() (ael)	**Property Functions**: Get the property's double, string, or long value from an ADS database object.
db_get_purpose_number() (ael)	**Layer and LayerId Functions**: Returns the purpose number for a given LayerId.
db_get_selection_tolerance() (ael)	**Selection Functions**: Returns the user's selection tolerance for a given window.
db_get_shape_bbox() (ael)	**Shape Functions**: Returns a handle to the bounding box of a shape.
db_get_shape_control_polygon() (ael)	**Primitive Polygon Functions**: Returns a primitive polygon containing the shape's control points. This polygon is guaranteed to not contain arcs or holes.
db_get_shape_layer() (ael)	**Shape Functions**: Returns the layer number of a given shape.
db_get_shape_layerid() (ael)	**Shape Functions**: This function returns the LayerId for a given shape.
db_get_shape_net() (ael)	**Shape Functions**: This functions returns the Net associated with the given shape. Returns NULL if there is no Net associated with the given shape.
db_get_shape_primitive_polygon() (ael)	**Primitive Polygon Functions**: Returns a primitive polygon for a given shape. Works for all shapes.
db_get_shape_selected_points() (ael)	**Selection Functions**: Returns a list of the selected points' coordinates for a given shape.
db_get_shape_text_string() (ael)	**Shape Functions**: Returns a string containing the text within the given text shape or annotation shape.

db_get_shape_type_description() (ael)	**Shape Functions**: Returns a string that describes the type of the given shape.
db_get_text_absolute() (ael)	**Shape Functions**: Function returns the boolean (TRUE,FALSE) absolute flag from a given text or annotation shape object.
db_get_text_angle() (ael)	**Shape Functions**: Returns the text angle in 0.001 degree units of the given text or annotation shape object.
db_get_text_font_name() (ael)	**Shape Functions**: Function returns the string font name for a given text or annotation shape object.
db_get_text_height() (ael)	**Shape Functions**: Function returns the text height in database units for a given text or annotation shape object.
db_get_text_justification() (ael)	**Shape Functions**: Function to return the justification attribute code from a text or annotation object.
db_get_text_origin() (ael)	**Shape Functions**: Returns the origin's (anchor point) coordinate in database units of the given text or annotation shape object.
db_get_transform_angle() (ael)	**Database Query and Manipulation Functions**: Returns the angle of a transform object as returned from the function db_get_instance_attribute().
db_get_transform_mirror_x() (ael)	**Database Query and Manipulation Functions**: Returns the x axis mirror flag of a transform object, as returned from the function db_get_instance_attribute(),
db_get_transform_mirror_y() (ael)	**Database Query and Manipulation Functions**: Returns the y axis mirror flag of a transform object, as returned from the function db_get_instance_attribute()
db_get_transform_x() (ael)	**Database Query and Manipulation Functions**: Returns the x coordinate of an instance's transformation record.
db_get_transform_y() (ael)	**Database Query and Manipulation Functions**: Returns the y coordinate of an instance's transformation record.
db_get_user_units_significant_digits() (ael)	**Design Context Functions**: Returns the integer number of user unit significant digits in the given DesignContext.
db_get_uu_to_dbu_factor() (ael)	**Design Context Functions**: Returns the real value conversion factor for converting user units to database units for a given DesignContext.
db_get_uu_to_mks_factor() (ael)	**Design Context Functions**: Returns the real value conversion factor for converting database user units to circuit MKS units for a given DesignContext
db_get_view_name() (ael)	**Design Context Functions**: Returns the view name of a design context.
db_get_x() (ael)	**Database Query and Manipulation Functions**: Returns an integer, the x coordinate in data base units.
db_get_y() (ael)	**Database Query and Manipulation Functions**: Returns an integer; the y coordinate in data base units.
db_has_explicit_hierarchy_policy() (ael)	**Simulation Functions**: Returns TRUE if the given design context has a specific HierarchyPolicy name set that will be used for simulation of the design.
db_highlight() (ael)	**Highlight Selection Functions**: Function to highlight or unhighlight a given shape, instance, or symbol port.

db_highlight_instance_ex() (ael)	**Command Functions**: Highlights an instance using the highlight color.
db_highlight_net() (ael)	**Highlight Selection Functions**: This function highlights or unhighlights a given Net within a given design context.
db_inst_iter_enable_caching() (ael)	**Instance Iterator Functions**: Function to make the iterator cache its list.
db_inst_iter_exclude_port_insts() (ael)	**Instance Iterator Functions**: Limits instance iteration to exclude port instances for a given instance iterator.
db_inst_iter_get_instance() (ael)	**Instance Iterator Functions**: Returns an instance object that is referenced by a given instance iterator
db_inst_iter_get_next() (ael)	**Instance Iterator Functions**: Returns an iterator to the next instance after the given instance iterator.
db_inst_iter_is_valid() (ael)	**Instance Iterator Functions**: Returns TRUE if the instance iterator references a valid instance object.
db_inst_iter_limit_region() (ael)	**Instance Iterator Functions**: Function to limit the instance iteration to a region.
db_inst_iter_limit_selected() (ael)	**Instance Iterator Functions**: Function to limit the iteration to selected instances.
db_inst_term_iter_get_inst_term() (ael)	**InstTerm Iterator Functions**: Returns the instance terminal object that is referenced by the given instance terminal iterator.
db_inst_pin_iter_get_inst_pin() (ael)	**InstPin Iterator Functions**: Returns the instance pin object referred to by the given instance pin iterator.
db_inst_pin_iter_get_next() (ael)	**InstPin Iterator Functions**: Returns the next instance pin iterator from the given instance pin iterator.
db_inst_pin_iter_is_valid() (ael)	**InstPin Iterator Functions**: Returns TRUE if the instance pin iterator is referencing a valid instance pin.
db_inst_term_iter_get_next() (ael)	**InstTerm Iterator Functions**: Returns the next instance terminal iterator of the given instance terminal iterator.
db_inst_term_iter_is_valid() (ael)	**InstTerm Iterator Functions**: Returns TRUE if the given instance terminal iterator is referencing a valid instance terminal object.
db_is_cell_name_display() (ael)	**Shape Functions**: Returns TRUE if the given annotation shape object is a cell name display.
db_is_hierarchy_context_at_root() (ael)	**HierarchyContext Functions**: Returns TRUE if the given HierarchyContext is at the root.
db_is_highlighted() (ael)	**Highlight Selection Functions**: Returns TRUE if the given instance, symbol port, or shape is highlighted.
db_is_instance() (ael)	**Object Functions**: Returns TRUE if the passed in object is an instance.
db_is_instance_deactivated() (ael)	**Instance Functions**: Returns TRUE if instance is deactivated.
db_is_instance_deactivated_and_shorted() (ael)	**Instance Functions**: Returns TRUE if the given instance is shorted.
db_is_instance_ground() (ael)	**Instance Functions**: Returns TRUE if the given instance is a ground component. Returns FALSE if the given instance is not a ground component.
db_is_inst_name_display() (ael)	**Shape Functions**: Returns TRUE if the given annotation shape object is an instance name display.
db_is_inst_pin_connected_to_wire() (ael)	**InstPin Functions**: Returns TRUE if the given instance pin is connected to a wire. Returns FALSE if

	the given instance pin is not connected to a wire.
db_is_inst_term_grounded() (ael)	**InstTerm Functions**: Returns TRUE if the given instance terminal is grounded.
db_is_layerid_invisible() (ael)	**Layer and LayerId Functions**: Returns TRUE if the given LayerId is marked invisible, FALSE otherwise.
db_is_layerid_protected() (ael)	**Layer and LayerId Functions**: Returns TRUE if the given LayerId is marked protected, FALSE otherwise.
db_is_net_grounded() (ael)	**Net Functions**: Returns TRUE if the given Net is grounded.
db_is_net_named_by_user() (ael)	**Net Functions**: Returns TRUE if the given Net was named by the user.
db_is_param_displayed() (ael)	**Parameter Value Functions**: Returns TRUE if the parameter value is marked to be displayed in the current view.
db_is_param_repeatable() (ael)	**Parameter Value Functions**: Returns TRUE if the parameter is repeatable.
db_is_pin() (ael)	**Object Functions**: Returns TRUE if the passed in object is a pin.
db_is_pin_selected() (ael)	**Pin Functions**: Returns TRUE if the passed in pin is selected.
db_is_primitive_instance_in_hierarchy() (ael)	**HierarchyContext Functions**: Returns TRUE if the instance is a primitive in the given HierarchyContext.
db_is_same_context() (ael)	**DesignContext Functions**: Returns TRUE if the two contexts are equivalent.
db_is_selected() (ael)	**Highlight Selection Functions**: Returns TRUE if the passed in instance, symbol port, or shape is selected.
db_is_shape() (ael)	**Object Functions**: Returns TRUE if the passed in object is a shape.
db_layerid() (ael)	**Layer and LayerId Functions**: Returns a new LayerId object of a given layer/purpose pair, specified by the layer number and optional purpose number.
db_mks_to_uu_factor() (ael)	**Database Query and Manipulation Functions**: Returns a real value conversion factor for converting circuit units to database user units for a given DesignContext.
db_net_iter_get_net() (ael)	**Net Iterator Functions**: Returns the net object that is referenced by the given net iterator.
db_net_iter_get_next() (ael)	**Net Iterator Functions**: Returns the next net iterator of the given net iterator.
db_net_iter_is_valid() (ael)	**Net Iterator Functions**: Returns TRUE if the given net iterator references a valid net object
db_next_parm() (ael)	**Database Query and Manipulation Functions**: Returns a handle to the next parameter of the instance.
db_num_parms() (ael)	**Database Query and Manipulation Functions**: Returns the number of parameters in a parameter list.
db_param_iter_find_by_index() (ael)	**Parameter Iterator Functions**: Returns a new parameter iterator for the parameter with the given parameter integer index within the given parameter iterator's list of parameters.
db_param_iter_find_by_index_and_repeat() (ael)	**Parameter Iterator Functions**: Returns a new parameter iterator for the parameter with the given parameter index and the given parameter's repeat index within the given parameter iterator's list of parameters.
db_param_iter_find_by_name_and_repeat() (ael)	**Parameter Iterator Functions**: Returns a new parameter iterator for the parameter with the given

	parameter name and the given parameter repeat index within the given parameter iterator's list of parameters.
db_param_iter_find_by_name() (ael)	**Parameter Iterator Functions**: Returns a new parameter iterator for the parameter with the given parameter name within the given parameter iterator's list of parameters.
db_param_iter_flatten_repeats() (ael)	**Parameter Iterator Functions**: Function to flatten the repeated parameters while iterating.
db_param_iter_get_display_value() (ael)	**Parameter Iterator Functions**: Returns a string with the displayed value of the parameter referenced by the given parameter iterator.
db_param_iter_get_first() (ael)	**Parameter Iterator Functions**: Returns the first parameter iterator from the given parameter iterator.
db_param_iter_get_netlist_value() (ael)	**Parameter Iterator Functions**: Returns a string with the netlist value of the parameter referenced by the given parameter iterator.
db_param_iter_get_next() (ael)	**Parameter Iterator Functions**: Returns the next parameter iterator from the given parameter iterator.
db_param_iter_get_param() (ael)	**Parameter Iterator Functions**: Returns the parameter value referred to by the given parameter iterator.
db_param_iter_get_parm_def() (ael)	**Parameter Iterator Functions**: Returns the parameter definition referred to by the given parameter iterator.
db_param_iter_get_repeat_index() (ael)	**Parameter Iterator Functions**: Returns the integer repeated index of the parameter referenced by the given parameter iterator
db_param_iter_get_sub_parameters() (ael)	**Parameter Iterator Functions**: Returns a parameter iterator to the sub-parameter values of the parameter referenced by the given parameter iterator
db_param_iter_is_repeatable() (ael)	**Parameter Iterator Functions**: Returns TRUE if the parameter referenced by the given parameter iterator is a top-level parameter that is repeatable.
db_param_iter_is_repeated() (ael)	**Parameter Iterator Functions**: Returns TRUE if the parameter referenced by the given parameter iterator is a repeat of a top-level repeatable parameter.
db_param_iter_is_valid() (ael)	**Parameter Iterator Functions**: Returns TRUE if the parameter iterator is referencing a valid parameter.
db_pick_instance_at() (ael)	**Selection Functions**: Returns the instance object which would be selected if the user clicked at the location.
db_param_value_uses_compound_form() (ael)	**Parameter Value Functions**: Returns TRUE if the parameter is compound valued.
db_param_value_uses_constant_form() (ael)	**Parameter Value Functions**: Returns TRUE if the parameter is constant valued.
db_param_value_uses_string_form() (ael)	**Parameter Value Functions**: Returns TRUE if the parameter is string valued.
db_pick_object_at() (ael)	**Selection Functions**: Returns the shape object, instance object, or port object which would be selected if the user clicked at the location.
db_pick_shape_at() (ael)	**Selection Functions**: Returns the shape object which would be selected if the user clicked at the location.
db_pin_iter_get_next() (ael)	**Pin Iterator Functions**: Returns the next pin iterator from the given pin iterator.
db_pin_iter_get_pin() (ael)	**Pin Iterator Functions**: Returns the pin object

	referred to by the given pin iterator.
db_pin_iter_is_valid() (ael)	**Pin Iterator Functions**: Returns TRUE if the pin iterator is referencing a valid pin.
db_pin_iter_limit_selected() (ael)	**Pin Iterator Functions**: Function to limit the pin iteration to selected pins.
db_pin_iter_limit_region() (ael)	**Pin Iterator Functions**: Function to limit the pin iteration to a region.
db_primitive_polygon_has_arcs() (ael)	**Primitive Polygon Functions**: Returns TRUE if a given primitive polygon contains arcs; returns FALSE otherwise.
db_prop_iter_get_name() (ael)	**Property Iterator Functions**: Returns the name of the property referenced the given property iterator.
db_prop_iter_get_next() (ael)	**Property Iterator Functions**: Returns the next property iterator from the given property iterator.
db_prop_iter_get_type() (ael)	**Property Iterator Functions**: Returns the integer value type of the iterator's current property.
db_prop_iter_get_value() (ael)	**Property Iterator Functions**: Returns the string, long or real value, if the optional type argument is not specified.
db_prop_iter_is_valid() (ael)	**Property Iterator Functions**: Returns TRUE if the property iterator is referencing a valid property.
db_prop_iter_remove_prop() (ael)	**Property Iterator Functions**: Removes the given iterator's current property and returns an iterator to the next property.
db_remove_properties_with_prefix() (ael)	**Property Functions**: Removes all properties from an ADS database object.
db_remove_property() (ael)	**Property Functions**: Removes a property from an ADS database object.
db_rollback_transaction() (ael)	**Transaction Functions**: Rollback the given transaction.
db_save_design_without_prompting() (ael)	**DesignContext Functions**: Saves the given design without notifying or prompting the user for permission.
db_select() (ael)	**Highlight Selection Functions**: Function to select or deselect an instance, symbol port, or shape.
db_select_all() (ael)	**Selection Functions**: Select all items in a DesignContext.
db_select_all_on_layerid() (ael)	**Selection Functions**: Function to select or deselect all shapes on a particular LayerId in the given design context.
db_select_area() (ael)	**Selection Functions**: Select all items in a given rectangular area of a DesignContext.
db_select_pin() (ael)	**Pin Functions**: Function to select or deselect a pin.
db_set_controller_design_name() (ael)	**Simulation Functions**: Function to set the string design name of the controller design for the given design context.
db_set_curve_radius() (ael)	**Preference Functions**: Sets the curve radius for paths and traces used when adding paths and traces with curved corners in the given design context.
db_set_datadisp_name() (ael)	**Simulation Functions**: Function to specify the custom (DDS) Data Display name to use for the given design context.
db_set_dataset_name() (ael)	**Simulation Functions**: Function to specify the custom dataset name to use for the given design context.
db_set_hierarchy_policy_name() (ael)	**Simulation Functions**: Function to set the

	HierarchyPolicy name to use specifically for simulation of the given design context.
db_set_layer_process_role() (ael)	**Layer and LayerId Functions**: Sets the layer process role for a given layer.
db_set_layerid_alpha() (ael)	**Layer and LayerId Functions**: Sets an alpha integer value (0-255) for a particular LayerId in a given library or a given design context's library's technology. Returns none.
db_set_layerid_fill_mode() (ael)	**Layer and LayerId Functions**: Sets the fill mode for a particular LayerId in a given library or a given design context's library's technology. Returns none.
db_set_layerid_fill_pattern() (ael)	**Layer and LayerId Functions**: Sets an integer fill pattern for a particular LayerId in a given library or a given design context's library's technology. Returns none.
db_set_layerid_invisible() (ael)	**Layer and LayerId Functions**: Function to mark a LayerId to be invisible or visible.
db_set_layerid_line_style() (ael)	**Layer and LayerId Functions**: Sets the line style for a particular LayerId in a given library or a given design context's library's technology. Returns none.
db_set_layerid_protected() (ael)	**Layer and LayerId Functions**: Function to mark a LayerId to be protected or unprotected.
db_set_layerid_rgb() (ael)	**Layer and LayerId Functions**: Sets a RGB color for a particular LayerId in a given library or a given design context's library's technology. Returns none.
db_set_map() (ael)	**Database Query and Manipulation Functions**: Sets the current transformation mapping.
db_set_miter_cutoff() (ael)	**Preference Functions**: Sets the miter cutoff angle used when adding paths and traces with mitered corners in the given design context.
db_set_opendds_option() (ael)	**Simulation Functions**: Function to specify for the given design if the Data Display window should open automatically after simulation of the design occurs.
db_set_param_displayed() (ael)	**Parameter Value Functions**: Function to mark a parameter value to be displayed or not displayed in the current view.
db_set_param_form_name() (ael)	**Parameter Value Functions**: Sets the given parameter value's form to be the form of the given string form name.
db_set_param_string_value() (ael)	**Parameter Value Functions**: Set the string value of the given parameter value.
db_set_path_corner() (ael)	**Preference Functions**: Sets the corner type used when adding paths and traces in the given design context.
db_set_path_width() (ael)	**Preference Functions**: Sets the width for paths and traces used when adding paths and traces in the given design context.
db_setup_map() (ael)	**Database Query and Manipulation Functions**: Modifies the current transformation mapping set
db_setup_expression_context() (ael)	**ExpressionContext Functions**: Setup an expression context to use for expression evaluation. The returned expression context defines the scope for expressions to be evaluated.
db_setup_transform() (ael)	**Database Query and Manipulation Functions**: Sets up the instance transformation mapping (rotation, translation and mirror).
db_shape_has_selected_points() (ael)	**Shape Functions**: Returns TRUE if a given shape has any point(s) selected. If the shape has no points

	selected, then FALSE is returned.
db_shape_is_annotation() (ael)	**Shape Functions**: Returns TRUE if the given shape is an annotation object.
db_shape_is_arc() (ael)	**Shape Functions**: Returns TRUE if the given shape is an arc.
db_shape_is_circle() (ael)	**Shape Functions**: Returns TRUE if the given shape is a circle.
db_shape_is_const_line() (ael)	**Shape Functions**: Returns TRUE if the given shape is a construction line.
db_shape_is_dot() (ael)	**Shape Functions**: Returns TRUE if the given shape is a dot.
db_shape_is_ellipse() (ael)	**Shape Functions**: Returns TRUE if the given shape is an ellipse.
db_shape_is_path() (ael)	**Shape Functions**: Returns TRUE if the given shape is a path.
db_shape_is_polygon() (ael)	**Shape Functions**: Returns TRUE if the given shape is a polygon.
db_shape_is_polyline() (ael)	**Shape Functions**: Returns TRUE if the given shape is a polyline.
db_shape_is_rectangle() (ael)	**Shape Functions**: Returns TRUE if the given shape is a rectangle.
db_shape_is_text() (ael)	**Shape Functions**: Returns TRUE if the given shape is a text object.
db_shape_is_text_or_annotation() (ael)	**Shape Functions**: Returns TRUE if the given shape is a text or annotation object.
db_shape_is_trace() (ael)	**Shape Functions**: Returns TRUE if the given shape is a trace.
db_shape_is_wire() (ael)	**Shape Functions**: Returns TRUE if the given shape is a wire.
db_shape_is_wire_label() (ael)	**Shape Functions**: Returns TRUE if the given shape is a wire label.
db_shape_is_wire_or_trace() (ael)	**Shape Functions**: Returns TRUE if the given shape is a wire or trace.
db_shape_iter_enable_caching() (ael)	**Shape Iterator Functions**: Function to make the iterator cache its list.
db_shape_iter_exclude_invisible_layers() (ael)	**Shape Iterator Functions**: Function to exclude invisible layers in the shape iteration.
db_shape_iter_exclude_protected_layers() (ael)	**Shape Iterator Functions**: Function to exclude protected layers in the shape iteration.
db_shape_iter_get_next() (ael)	**Shape Iterator Functions**: Returns an iterator to the next shape after the given shape iterator.
db_shape_iter_get_shape() (ael)	**Shape Iterator Functions**: Returns a shape object that is referenced by a given shape iterator
db_shape_iter_include_invisible_layers() (ael)	**Shape Iterator Functions**: Function to include invisible layers in the iteration
db_shape_iter_include_protected_layers() (ael)	**Shape Iterator Functions**: Function to include protected layers in the iteration
db_shape_iter_is_valid() (ael)	**Shape Iterator Functions**: Returns TRUE if the shape iterator references a valid shape object.
db_shape_iter_limit_layer() (ael)	**Shape Iterator Functions**: Function to limit the shape iteration to a layer.
db_shape_iter_limit_layerid() (ael)	**Shape Iterator Functions**: Function to limit the shape iteration to a particular LayerId.
db_shape_iter_limit_purpose() (ael)	**Shape Iterator Functions**: Function to limit the shape iteration to a purpose layer.

db_shape_iter_limit_region() (ael)	**Shape Iterator Functions**: Function to limit the shape iteration to a region.
db_shape_iter_limit_selected() (ael)	**Shape Iterator Functions**: Function to limit the iteration to selected shapes.
db_transaction_is_empty() (ael)	**Transaction Functions**: Returns TRUE if no changes have occured since the start of the transaction.
db_transform_angle() (ael)	**Database Query and Manipulation Functions**: Transforms an angle value by rotation and mirror values from the current transformation as set by db_set_map(), db_setup_transformation(), or db_setup_map().
db_transform_bbox_ex() (ael)	**Database Query and Manipulation Functions**: Transforms the given bounding box to be the smallest possible bounding box which completely contains the transformed input bounding box.
db_transform_coord() (ael)	**Database Query and Manipulation Functions**: Transforms a single coordinate value using the current transformation as set by db_set_map(), db_setup_transformation(), or db_setup_map().
db_unhighlight_instances_ex() (ael)	**Command Functions**: Function to unhighlight all instances in a given design context.
dBm() Function (ael)	**Math Functions**: Converts a voltage to decibels referenced to milliwatt.
de_convert_path_to_trace() (ael)	**Command Functions**: Converts selected paths to traces.
de_dc_annotation() (ael)	**Command Functions**: Display DC node voltages and branch currents on the current schematic after a DC or Transient simulation.
de_deselect_window() (ael)	**Command Functions**: Deselects all objects in given window in current representation.
de_edit_annotation_attribute() (ael)	**Command Functions**: Edits attributes of selected instance's parameter annotation.
de_edit_text_string() (ael)	**Command Functions**: Edits text string in the current representation that was set with de_set_edit_text().
de_activate() (ael)	**Command Functions**: Activates an instance by setting the deactivate flag of an instance to false.
de_add_arc() (ael)	**Command Functions**: Adds a clockwise or counterclockwise, circular arc to a polygon or polyline in user units in the current representation.
de_add_arc1() (ael)	**Command Functions**: Adds an arc using user-defined units.
de_add_arc2() (ael)	**Command Functions**: Adds an arc using user-defined units.
de_add_arc3() (ael)	**Command Functions**: Adds an arc using user-defined units.
de_add_arc4() (ael)	**Command Functions**: Adds an arc using user-defined units.
de_add_circle() (ael)	**Command Functions**: Draws a circle in the current representation in user-defined units.
de_add_construction_line() (ael)	**Command Functions**: Adds a construction line using user-defined units.
de_add_path() (ael)	**Command Functions**: Starts a path command sequence.
de_add_point() (ael)	**Command Functions**: Adds a point to a polygon, polyline, or path using user-defined units.
de_add_polygon() (ael)	**Command Functions**: Adds a polygon to the current representation.

de_add_polyline() (ael)	**Command Functions**: Adds a polyline to the current representation.
de_add_property() (ael)	**Command Functions**: Adds property to data group, port, or instance.
de_add_rectangle() (ael)	**Command Functions**: Adds a rectangle to the current representation.
de_add_text() (ael)	**Command Functions**: Adds a text string to the current representation.
de_add_trace() (ael)	**Command Functions**: Starts trace command sequence for adding a trace to the current representation.
de_add_vertex() (ael)	**Command Functions**: Adds a vertex to an existing polygon, polyline, wire or trace in the current representation.
de_add_wire label() (ael)	**Command Functions**: Adds a label to a wire or pin at the x,y location.
de_add_wire() (ael)	**Command Functions**: Adds a wire vertex to a wire or trace connection in the current representation.
de_analyze() (ael)	**Simulator Command Functions**: Invokes the simulator and sends it a new, updated netlist of the current design.
de_analyze_tune() (ael)	**Simulator Command Functions**: Invokes the simulator, sends the simulator a new, updated netlist of the current design, and prepares the simulator to receive de_tune() commands.
de_ang_factor() (ael)	**Design Environment Query Functions**: Returns a real number, a conversion factor to convert a value from simulator units to degrees.
de_bom() (ael)	**Command Functions**: Generates a bill of material file for the current representation.
de_boolean_logical() (ael)	**Command Functions**: Performs boolean operations on shapes placed on different layers in a layout presentation.
de_break_connection() (ael)	**Command Functions**: Breaks connection to attached wire or trace from the selected instances in the current representation.
de_cell_exists() (ael)	**Command Functions**: Returns TRUE if a cell with the given cell name exists in the given open library.
de_cellview_exists() (ael)	**Command Functions**: Returns TRUE if a cellview with the given cell name and view name exists within a given open library.
de_change_annotation_layer() (ael)	**Command Functions**: Changes the layer of the nearest annotation (within the select region) to the current layer.
de_check_rep_options() (ael)	**Command Functions**: Generates a check representation report.
de_chop() (ael)	**Command Functions**: Removes the defined rectangular region from selected polygons, rectangles, circles, or paths.
de_clear_dc_annotation() (ael)	**Command Functions**: Removes the annotated DC node voltages and branch currents on the current schematic.
de_clear_highlighting() (ael)	**Command Functions**: Clears any highlighting from a design representation in a current window.
de_close_all() (ael)	**Command Functions**: Closes all designs from memory.
de_close_all_datadisplay() (ael)	**Simulator Command Functions**: Hides all data

	display windows that have been opened.
de_close_window() (ael)	**Command Functions**: Closes the current window instance.
de_close_workspace_without_prompting() (ael)	**Command Functions**: Closes the current open workspace.
de_connect() (ael)	**Command Functions**: Starts a wire or trace connection using autorouting.
de_convert_to_polygon() (ael)	**Command Functions**: Converts selected closed shapes to polygons.
de_convert_trace_to_path() (ael)	**Command Functions**: Converts selected traces to paths.
de_convert_traces_to_instances() (ael)	**Command Functions**: Converts selected traces to transmission line elements.
de_copy() (ael)	**Command Functions**: Copies the selected components in the current representation and places them at delta from the original position.
de_copy_cell() (ael)	**Command Functions**: Function to copy a cell. Returns TRUE if cell was successfully copied, FALSE if not.
de_copy_cellview() (ael)	**Command Functions**: Function to copy a cellview. Returns TRUE if cellview was successfully copied, FALSE if not.
de_copy_file() (ael)	**Command Functions**: Copies a file.
de_copy_to_buffer() (ael)	**Command Functions**: Copies the selected group in the current representation to the copy buffer.
de_copy_to_layer() (ael)	**Command Functions**: Copies the selected group in the current representation to the current active layer.
de_create_new_layout_view() (ael)	**Design Context Functions**: Function to create a new layout view. Returns the design context to the created layout view.
de_create_new_schematic_view() (ael)	**Design Context Functions**: Function to create a new schematic view. Returns the design context to the created schematic view.
de_create_new_symbol_view() (ael)	**Design Context Functions**: Function to create a new symbol view. Returns the design context to the created symbol view.
de_create_polygon() (ael)	**Command Functions**: Returns a Shape polygon object and adds the polygon to the given design context and refreshes the design context's view.
de_create_window() (ael)	**Command Functions**: Creates and opens a window.
de_crop() (ael)	**Command Functions**: Preserves the defined rectangular region from selected polygons, rectangles, circles, or paths while removing all areas outside of the region.
de_current_context_is_valid() (ael)	**Design Context Functions**: Returns TRUE if there is a valid current DesignContext.
de_current_design_type() (ael)	**Design Environment Query Functions**: Returns the current design type (the design opened in the active window).
de_data_dialog() (ael)	**User Interface Function**: Displays a scrollable multiline text editor.
de_deactivate() (ael)	**Command Functions**: Sets instance's deactivate flag to true (makes the instance deactivated).
de_define_edge_area_port() (ael)	**Command Functions**: Modifies a circle, path, polygon, rectangle to become a area port or modifies a polyline or arc to become a edge port.

de_define_library_palette() (ael)	**Component Definition Functions**: Defines a palette group of ADS Components for a particular library that contains components from the same library or different libraries.
de_define_npport() (ael)	**Command Functions**: Places a non-preferred port/pin in an artwork instance.
de_define_palette_group() (ael)	**Component Definition Functions**: Defines a palette group.
de_define_port() (ael)	**Command Functions**: Places a single port, with a unique port number, in an artwork instance using user units.
de_delete() (ael)	**Command Functions**: Deletes selected objects, or object near given point in current representation.
de_delete_all_orphaned_instances() (ael)	**Command Functions**: Deletes all the orphaned instances in the current representation.
de_delete_cell() (ael)	**Command Functions**: Function to delete a cell and all of its contents. Returns TRUE if cell was successfully deleted, FALSE if not.
de_delete_cellview() (ael)	**Command Functions**: Function to delete a cellview. Returns TRUE if cellview was successfully deleted, FALSE if not.
de_delete_view() (ael)	**Command Functions**: Deletes a stored view with the matching name of view.
de_delete_workspace() (ael)	**Command Functions**: Deletes a specified ADS workspace from the given workspace directory path.
de_deselect_all() (ael)	**Command Functions**: Deselects all objects in the current representation.
de_deselect_all_force() (ael)	**Command Functions**: Deselects all objects regardless of layer protection status.
de_deselect_by_name() (ael)	**Command Functions**: Deselects instances by instance name or ID.
de_difference() (ael)	**Command Functions**: Creates new polygons by subtracting the intersections of the selected shapes (e.g., polygon, rectangle, circle, or path) from the union of the selected shapes on the same layer.
de_draw_arc() (ael)	**Command Functions**: Adds a circular arc to a polygon or polyline in simulator units.
de_draw_arc1() (ael)	**Command Functions**: Adds an arc using simulator units.
de_draw_arc2() (ael)	**Command Functions**: Adds an arc using simulator units.
de_draw_arc3() (ael)	**Command Functions**: Adds an arc using simulator units.
de_draw_arc4() (ael)	**Command Functions**: Adds an arc using simulator units.
de_draw_circ() (ael)	**Command Functions**: Draws a circle in simulator units.
de_draw_point() (ael)	**Command Functions**: Adds a point to a polygon or polyline to the current representation using simulator units.
de_draw_port() (ael)	**Command Functions**: Adds a port to an art work instance.
de_draw_rect() (ael)	**Command Functions**: Draws a rectangle in simulator units.
de_draw_text() (ael)	**Command Functions**: Draws text at given location, with given font, height and angle, height and angle are in simulator units.

de_dse_l2s() (ael)	**Command Functions**: Synchronizes the schematic with the layout, using the layout as the reference representation.
de_dse_s2l() (ael)	**Command Functions**: Synchronizes the schematic with the layout, using the layout as the reference representation.
de_edit_item() (ael)	**Command Functions**: Allows a given item to be selected for editing.
de_edit_item_ex() (ael)	**Command Functions**: Allows a given item to be selected for editing. Returns an item info handle for the instance.
de_edit_path_trace() (ael)	**Command Functions**: Modifies attributes of a selected path or trace in the current representation.
de_edit_symbol_pin() (ael)	**Command Functions**: Sets up the editing of a symbol pin.
de_edit_text_attribute() (ael)	**Command Functions**: Edits the attributes of selected text components in the current representation.
de_empty() (ael)	**Command Functions**: Empties an enclosed, filled shape in the current representation, creating a hole.
de_end() (ael)	**Command Functions**: Completes a polygon, polyline, wire or trace command sequence. This command completes a shape.
de_end_command() (ael)	**Command Functions**: Terminates a repeating command sequence.
de_end_edit_item() (ael)	**Command Functions**: Commits and displays the editing changes of a given instance.
de_error_bell() (ael)	**Utility Functions**: Sounds the error bell.
de_exit() (ael)	**Utility Functions**: Exits Advanced Design System.
de_export_design() (ael)	**Command Functions**: Exports the current layout or schematic in given format.
de_fill() (ael)	**Command Functions**: Converts an empty shape (a hole) in the current representation into a filled shape (normal closed shape).
de_find_arc_center() (ael)	**Command Functions**: Moves the cursor to the center of the arc or circle selected within the select region and enters the coordinates to the event that was previously installed.
de_find_design_context_from_name() (ael)	**DesignContext Functions**: Function to find the DesignContext from a design name and a rep type.
de_find_line_center() (ael)	**Command Functions**: Moves the cursor to the pin closest to the center of the line selected within the select region and enters that point to the event that was previously installed.
de_find_pin() (ael)	**Command Functions**: Moves the cursor to the pin closest to the point selected within the select region and enters that point to the event that was previously installed.
de_find_vertex() (ael)	**Command Functions**: Moves the cursor to the pin closest to the vertex selected within the select region and enters that point to the event that was previously installed.
de_fix_instances() (ael)	**Command Functions**: Fixes the position of an instance in the current representation and prevents design synchronization from re-positioning it.
de_flatten() (ael)	**Command Functions**: Removes a single level of hierarchy.

de_format_lib_cell_view_name() (ael)	**Command Functions**: Returns a formatted string ADS design name constructed from a given libraryName, cellName, and viewName.
de_free_instances() (ael)	**Command Functions**: Frees the position of an instance in the current representation and allows design synchronization to re-position it.
de_free_item() (ael)	**Command Functions**: Frees the data structure created by de_init_item() when it is no longer needed.
de_generate_symbol_ex() (ael)	**Command Functions**: This function generates a symbol into the given symbol context based on the design within the given DesignContext.
de_get_alternate_window() (ael)	**Design Environment Query Functions**: Returns the alternate window instance handle and sets the current window to this alternate.
de_get_cells_in_library() (ael)	**Command Functions**: Returns an AEL list of string names of all cells in a given open library.
de_get_current_design_context() (ael)	**Design Context Functions**: Returns the current DesignContext.
de_get_data_parm() (ael)	**Command Functions**: Returns a string, the value of a given data reference or a named parameter belonging to a default component.
de_get_default_hierarchy_policy_name() (ael)	**Simulation Functions**: Returns the name of the default HierarchyPolicy that is used for simulating designs.
de_get_design_context() (ael)	**Design Context Functions**: Returns the DesignContext of the given window.
de_get_design_context_from_name() (ael)	**DesignContext Functions**: Function to find the DesignContext from a design name.
de_get_design_instances() (ael)	**Design Environment Query Functions**: Returns a list of instance names of a given element, belonging to the named design.
de_get_env() (ael)	**Utility Functions**: Retrieves environment file name.
de_get_file_names() (ael)	**Design Environment Query Functions**: Returns a list of files with given file extension.
de_get_open_libraries() (ael)	**Command Functions**: Returns an AEL list of the current open library names.
de_get_preference() (ael)	**Preference Functions**: Returns the value of a current preference, the preference setting.
de_get_views_in_library_cell() (ael)	**Command Functions**: Returns an AEL list of string names of all views in a given library, cell. With the optional windowType argument you can limit the returned list of view names to a certain window type "LAYOUT_WINDOW, SCHEMATIC_WINDOW, or SYMBOL_WINDOW".
de_get_window() (ael)	**Design Environment Query Functions**: Returns the currently active window
de_get_windows_with_same_context() (ael)	**Design Context Functions**: Returns a list of windows that are currently displaying the same design and view as the given window or DesignContext.
de_get_windows_with_same_library() (ael)	**Command Functions**: Returns an AEL list of window instances that are currently displaying designs stored in a given library.
de_group_edit_parameter_value() (ael)	**Command Functions**: Modifies selected instances with the parameter name to the value given.
de_hide_or_display_component_name() (ael)	**Command Functions**: Toggles the instance component name in annotation to hide or display.

de_import_design() (ael)	**Command Functions**: Imports a foreign design format.
de_info() (ael)	**User Interface Function**: Displays an information dialog box.
de_init_item() (ael)	**Command Functions**: Initializes the instance, readying it for placement.
de_insert_arrow() (ael)	**Command Functions**: Inserts an arrow draw as a polyline or polygon item.
de_insert_dimlin() (ael)	**Command Functions**: Inserts a dimension line.
de_install_get_xy() (ael)	**User Interface Functions**: Installs a user-defined event handler for prompting in the current window.
de_install_get_xy_pair() (ael)	**User Interface Functions**: Installs a user-defined event handler for prompting in the current window.
de_instantiate() (ael)	**Command Functions**: Creates a new design from the selected group.
de_intersection() (ael)	**Command Functions**: Creates new polygons formed by the intersections of selected shapes (e.g., polygon, rectangle, circle, or path) on the same layer.
de_invoke_help() (ael)	**Design Environment Query Functions**: Accepts two arguments and returns a command to the help server
de_is_artwork_macro_context() (ael)	**Design Context Functions**: Returns TRUE if the given window or DesignContext is an artwork macro evaluation.
de_is_cellview_open() (ael)	**Command Functions**: Function to test if a cellview is open. Returns TRUE if the cellview is open, returns FALSE if it is not.
de_is_layout_context() (ael)	**Design Context Functions**: Returns TRUE if the given window or DesignContext is a layout window.
de_is_library_open() (ael)	**Command Functions**: Returns TRUE if any library with the given library name is open; Returns FALSE otherwise.
de_is_project_or_workspace_open() (ael)	**Command Functions**: This function returns TRUE if any ADS Workspace (ADS2011 and later) or ADS Project(used in ADS2009 Update 1 and prior releases) is open, otherwise FALSE.
de_is_schematic_context() (ael)	**Design Context Functions**: Returns TRUE if the given window or DesignContext is a schematic window.
de_is_schematic_or_layout_context() (ael)	**Design Context Functions**: Returns TRUE if the given DesignContext or window instance's DesignContext is a schematic design context or is a layout design context; Returns FALSE otherwise.
de_is_symbol_context() (ael)	**Design Context Functions**: Returns TRUE if the given window or DesignContext is a symbol window.
de_last_view() (ael)	**Command Functions**: Recalls last view into the current window.
de_list_select() (ael)	**User Interface Functions**: Pops up a list selection dialog box.
de_mirror_x() (ael)	**Command Functions**: Mirrors selected objects around the X-axis, given a reference point.
de_mirror_y() (ael)	**Command Functions**: Mirrors selected objects around the Y-axis, given a reference point.
de_miter_vertex() (ael)	**Command Functions**: Creates a mitered edge on a polygon or polyline.
de_modify_arc_resolution() (ael)	**Command Functions**: Modifies the resolution of an arc.

de_modify_break() (ael)	**Command Functions**: Converts selected polygons into polylines; that is, breaks a closed shape into an open shape.
de_modify_circle_radius() (ael)	**Command Functions**: Modifies the radius of a circle.
de_modify_explode() (ael)	**Command Functions**: Converts selected polygons and polylines into two-point polyline segments.
de_modify_join() (ael)	**Command Functions**: Joins selected polylines with coincident end points into a single polyline or polygon.
de_move() (ael)	**Command Functions**: Moves selected items in the current representation by a given amount.
de_move_annotation() (ael)	**Command Functions**: Moves parameter annotation of a selected instance to a new location.
de_move_break() (ael)	**Command Functions**: Moves selected instances in the current representation, breaking any wire or trace connection.
de_move_to_layer() (ael)	**Command Functions**: Moves selected objects in the current representation to the current entry layer.
de_net() (ael)	**Command Functions**: Creates a netlist report file.
de_netlist_create() (ael)	**Netlist Functions**: Generates a netlist for the given DesignContext.
de_netlist_get_text() (ael)	**Netlist Functions**: Returns the full text of a netlist as an AEL string.
de_netlist_interpret_format_string() (ael)	**Netlist Functions**: Given a netlist format string, this function will return a string representing an outline of the structure of the format string.
de_netlist_save() (ael)	**Netlist Functions**: Saves the given netlist object's text to an indicated file.
de_new_datadisplay() (ael)	**Command Functions**: Sends the new window command to the data display server.
de_open_datadisplay() (ael)	**Simulator Command Functions**: Opens a saved data display file.
de_open_hierarchy_dialog() (ael)	**User Interface Function**: Opens dialog box to display hierarchy of current design.
de_open_info_dialog() (ael)	**User Interface Function**: Opens dialog box for displaying information.
de_open_workspace() (ael)	**Command Functions**: Opens a specified workspace directory path.
de_oversize() (ael)	**Command Functions**: Creates a new oversized or undersized shape from the selected closed shapes in the current representation.
de_pan_window() (ael)	**Command Functions**: Re-centers the viewing window to the given point.
de_parse_lib_cell_view_name() (ael)	**Command Functions**: Returns a parsed libraryName, cellName, and viewName from a given ADS design name.
de_parts() (ael)	**Command Functions**: Generates a parts list for the current representation and stores in the specified file.
de_parts_option_add_exclusion_items() (ael)	**Command Functions**: Adds an Exclusion List to the parts list options.
de_parts_option_add_inclusion_items() (ael)	**Command Functions**: Adds an Inclusion List to the parts list options.
de_parts_option_check_bom() (ael)	**Command Functions**: Checks the BOM Flag.
de_parts_option_include_header() (ael)	**Command Functions**: Sets the parts list option to Include Header.
de_parts_option_set_attribute_columns() (ael)	**Command Functions**: Sets the User Attribute

	Columns parts list options.
de_parts_option_set_center_placement() (ael)	**Command Functions**: Sets the Component Placement X,Y Coordinates parts list options.
de_parts_option_set_delimeter() (ael)	**Command Functions**: Sets the Delimeter Character parts list options.
de_parts_option_set_hierarchical() (ael)	**Command Functions**: Sets Hierarchical Reporting parts list option.
de_parts_option_set_package_offset() (ael)	**Command Functions**: Sets the Package Offsets parts list option.
de_parts_option_sort_by_component() (ael)	**Command Functions**: Sets the Sort by Component Name parts list option.
de_paste_from_buffer() (ael)	**Command Functions**: Copies the contents of a buffer to the current representation.
de_place_design_template() (ael)	**Command Functions**: Inserts the design template setup with de_set_design_template() into current design.
de_place_item() (ael)	**Command Functions**: Places the instance that was initialized by de_init_item().
de_place_port() (ael)	**Command Functions**: Places a symbol pin in the current representation's symbol view.
de_place_unplaced() (ael)	**Command Functions**: Places an instance that has not yet been placed in one representation.
de_playback_macro() (ael)	**Command Functions**: Executes an AEL or DEM file.
de_plot() (ael)	**Command Functions**: Creates a plot of the active design in the current window and sends it to the default printer or plotter.
de_plot_to_file() (ael)	**Command Functions**: Creates a plot of the active design and saves it to a file in a format that can be sent to a printer or plotter.
de_pop_outof_instance() (ael)	**Command Functions**: Returns to the previous design before a de_push_into_instance() command.
de_post_help() (ael)	**Design Environment Query Functions**: Accepts one argument and returns a command to the help server
de_print_info() (ael)	**User Interface Function**: Prints the contents of the following dialog boxes: data, new features, info, hierarchy, editor, DSE schematic status, DSE layout status, and check rep report.
de_prompt() (ael)	**User Interface Function**: Invokes the prompt dialog box in the current window with user-supplied prompt string
de_push_into_instance() (ael)	**Command Functions**: Pushes into hierarchical instance reference.
de_question() (ael)	**User Interface Function**: Invokes question dialog box in the current window.
de_read_preferences() (ael)	**Preference Functions**: Reads a preference file from disk, re-setting the preferences in the current window.
de_refresh_layers() (ael)	**Preference Functions**: Refreshes the layer information in the windows displaying the current design representation.
de_refresh_view() (ael)	**Command Functions**: Redraws the screen of the active window.
de_release_simulator() (ael)	**Command Functions**: Cancels the simulation and makes the simulator license available for other users on the network.

de_remove_properties() (ael)	**Command Functions**: Removes properties of design group, port, or instance.
de_rename_cell() (ael)	**Command Functions**: Function to rename a cell to a specified new name. Returns TRUE if cell was successfully renamed, FALSE if not.
de_rename_cellview() (ael)	**Command Functions**: Function to rename a cellview to a specified new name. Returns TRUE if cellview was successfully renamed, FALSE if not.
de_restore_all_datadisplay() (ael)	**Simulator Command Functions**: Opens all data display windows that have been hidden, using de_close_all_datadisplay().
de_restore_status() (ael)	**Simulator Command Functions**: Opens the closed status server window.
de_restore_view() (ael)	**Command Functions**: Recalls specified view into the current window.
de_retrieve_version_info() (ael)	**Design Environment Query Functions**: Returns a long string of product version information for the Design Environment.
de_rotate() (ael)	**Command Functions**: Rotates selected items in the current representation around a given point.
de_rotate_90() (ael)	**Command Functions**: Rotates an item being placed 90 degrees clockwise around pin 1.
de_rotate_center() (ael)	**Command Functions**: Rotates selected items in the current representation around the center of gravity if more than one object is selected.
de_rotate_image() (ael)	**Command Functions**: Rotates an instance (element) being placed in a specified direction around pin 1.
de_save_all_designs() (ael)	**Command Functions**: Saves all designs in memory.
de_save_design_template() (ael)	**Command Functions**: Saves the schematic representation of a design (or a design template) as a design template.
de_scale() (ael)	**Command Functions**: Scales items in the current representation by a scale factor.
de_search_and_replace() (ael)	**Command Functions**: Finds all references (depending on refType), given a variable or item reference, and highlights the item.
de_select_all() (ael)	**Command Functions**: Selects everything matching the select filter in the current window.
de_select_all_force() (ael)	**Command Functions**: Selects everything on a specified layer matching the select filter in the current window.
de_select_all_on_layer() (ael)	**Command Functions**: Selects everything on a specified layer matching the select filter in the current window.
de_select_by_name() (ael)	**Command Functions**: Selects instances by component name or instance name.
de_select_item() (ael)	**Command Functions**: Toggles a selection of an item within the select region in the current representation.
de_select_range() (ael)	**Command Functions**: Selects all items in the current representation enclosed by a given window range.
de_select_unplaced() (ael)	**Command Functions**: Selects an unplaced instance in the current representation to place in the other representation (from schematic to layout or layout to schematic).
de_select_window() (ael)	**Command Functions**: Selects all items (matching the select filter) in the current representation

	enclosed by given window.
de_set_annotation_font() (ael)	**Preference Functions**: Sets the font for instance parameter annotation for the current window.
de_set_annotation_height() (ael)	**Preference Functions**: Sets the text height used for component parameter annotation.
de_set_annotation_id_layer() (ael)	**Preference Functions**: Sets the layer number for instance name (ID) parameter annotation for the current window.
de_set_annotation_name_layer() (ael)	**Preference Functions**: Sets the layer number for instance parameter annotation for the current window.
de_set_annotation_parameters_layer() (ael)	**Preference Functions**: Sets the layer number for instance parameter annotation for the current window.
de_set_annotation_precision() (ael)	**Preference Functions**: Sets the display precision of real numbers displayed in schematic annotation.
de_set_annotation_rows() (ael)	**Preference Functions**: Sets the number of rows for instance parameter annotation for the current window.
de_set_arc_radius() (ael)	**Preference Functions**: Set the radius for any arcs created using the de_vertex_to_arc() command
de_set_background_color() (ael)	**Preference Functions**: Sets the background color of the drawing area in the current window.
de_set_backup_count() (ael)	**Preference Functions**: Sets the number of edits that must be performed before a backup file is made.
de_set_coord_entry_popup() (ael)	**Preference Functions**: Sets whether the Coordinate Entry dialog will be displayed for draw and place commands.
de_set_curve_radius() (ael)	**Preference Functions**: Sets the curve radius in the current window used for paths and traces with rounded corners.
de_set_default_hierarchy_policy_name() (ael)	**Simulation Functions**: Sets the name of the default HierarchyPolicy that is used for simulating designs.
de_set_design_template() (ael)	**Command Functions**: Sets up the design template for insertion into current design by de_place_design_template().
de_set_drag_move() (ael)	**Preference Functions**: Sets the ability to drag and move objects with the left mouse button.
de_set_drag_move_size() (ael)	**Preference Functions**: Sets the minimum distance an object must be dragged before the object is moved.
de_set_drag_move_units() (ael)	**Preference Functions**: Sets the units of the drag move.
de_set_dse_start() (ael)	**Preference Functions**: Sets the starting instance for design synchronization in the current representation.
de_set_dual_placement() (ael)	**Preference Functions**: Sets the mode so that when enabled, causes any component placed in one representation to be automatically placed in the other representation.
de_set_edit_property() (ael)	**Command Functions**: Sets the data group, port, or instance for property editing.
de_set_edit_symbol_pin() (ael)	**Command Functions**: Sets the symbol pin whose attributes are to be edited.
de_set_edit_text() (ael)	**Command Functions**: Sets the text string in the current representation to be edited by de_edit_text_string().
de_set_error_bell() (ael)	**Utility Functions**: Enables or disables the error bell

	(on or off).
de_set_foreground_color() (ael)	**Preference Functions**: Sets the sketch color of the active window.
de_set_global_db_factor() (ael)	**Preference Functions**: Sets the simulator units to layout user unit conversion factor used during rendering of AEL artwork objects.
de_set_grid_color() (ael)	**Preference Functions**: Sets the color of the visible grid in the current window.
de_set_grid_display_type() (ael)	**Preference Functions**: Sets the grid display to dots or dotted line for the current window.
de_set_grid_snap() (ael)	**Preference Functions**: Sets grid snap increments in X and Y direction and turns grid snapping for the current window on or off.
de_set_grid_snap_mode() (ael)	**Preference Functions**: Turns grid snapping for the current window on or off.
de_set_grid_snap_type() (ael)	**Preference Functions**: Sets the snap type to one or more of grid, pin, or vertex for the current window.
de_set_hierarchy_instance_path() (ael)	**Command Functions**: Function to set the instance path for the design within given design context.
de_set_hierarchy_root() (ael)	**Command Functions**: Function to set the hierarchy root for the current design within a given design context.
de_set_highlight_color() (ael)	**Preference Functions**: Sets the highlight color for the current window (used for errors, and show commands).
de_set_inst_pin_order_property() (ael)	**Command Functions**:
de_set_item_id() (ael)	**Command Functions**: Sets the ID of the item that is either being readied to be placed (from de_init_item()) or is being placed (from de_edit_item()).
de_set_item_parameters() (ael)	**Command Functions**: Sets the parameter values for a given item that has been selected by the de_init_item() or de_edit_item().
de_set_layer() (ael)	**Preference Functions**: Sets the entry layer for the current layer for the current window.
de_set_major_grid_display() (ael)	**Preference Functions**: Sets the major grid display factor.
de_set_minor_grid_display() (ael)	**Preference Functions**: Sets the minor grid display factor to xFactor multiple of the snap grid in X and yFactor multiple of the snap grid in Y.
de_set_miter_cutoff() (ael)	**Preference Functions**: Sets the miter cutoff angle for the path and trace command.
de_set_miter_length() (ael)	**Preference Functions**: Sets the miter length.
de_set_move_annotation() (ael)	**Command Functions**: Selects an instance to have its annotation moved.
de_set_origin() (ael)	**Command Functions**: Resets the origin of a representation (resets the 0,0 point).
de_set_oversize() (ael)	**Preference Functions**: Sets the amount to oversize or undersize closed shapes for the de_oversize() command for the current window.
de_set_path_corner() (ael)	**Preference Functions**: Sets the corner type used for paths and traces in the current window.
de_set_path_width() (ael)	**Preference Functions**: Sets the width for paths and traces used by the de_add_path() and de_add_trace() entry commands for the current window.
de_set_pin_color() (ael)	**Preference Functions**: Sets the color used for instance pins for the current window.

de_set_pin_size() (ael)	**Preference Functions**: Sets the size used to display instance pins of the current window.
de_set_pin_size_units() (ael)	**Preference Functions**: Sets the units for the pin size set by de_set_pin_size().
de_set_pin_snap() (ael)	**Preference Functions**: Sets the maximum distance at which the cursor snaps to a pin when snap type includes PREF_SNAP_TO_PIN.
de_set_pin_snap_units() (ael)	**Preference Functions**: Sets the units for the pin snap distance set by de_set_pin_snap().
de_set_place_popup_mode() (ael)	**Preference Functions**: Sets mode for displaying the component Parameter dialog box for the current window.
de_set_place_popup_on_zero_parm() (ael)	**Preference Functions**: Sets mode for the current window for displaying the Component Parameter dialog box for components with no parameters.
de_set_plot_pin_names() (ael)	**Preference Functions**: Turns the display of pin names on or off.
de_set_plot_pin_numbers() (ael)	**Preference Functions**: Turns the plotting of pin numbers on and off for the current window.
de_set_plot_pins() (ael)	**Preference Functions**: Turns plotting of connected pins on or off for the current window.
de_set_plotting_depth() (ael)	**Preference Functions**: Sets the hierarchical level at which component instances in Layout are drawn in outline.
de_set_port() (ael)	**Command Functions**: Sets attributes of a symbol port (pin) before it is created for the current window.
de_set_port_size() (ael)	**Preference Functions**: Sets the size of ports in the Layout representation.
de_set_port_size_units() (ael)	**Preference Functions**: Sets the units for the port size set by de_set_port_size().
de_set_preference() (ael)	**Preference Functions**: Sets the value of any preference.
de_set_reroute_wires() (ael)	**Preference Functions**: Sets wire or trace editing route mode for the current window.
de_set_resolution_for_arc() (ael)	**Preference Functions**: Sets the resolution that is used for approximating an arc
de_set_rotation_increment() (ael)	**Preference Functions**: Sets the step increment used by the de_rotate() command for the current window.
de_set_route_around_annot() (ael)	**Preference Functions**: When the Wire Route command is in progress, this mode determines if the wire should route around or through annotations.
de_set_scale() (ael)	**Preference Functions**: Set the X- and Y-scale factors used by the de_scale() command in the current window.
de_set_select_box_size() (ael)	**Preference Functions**: Sets the select region size for the current window.
de_set_select_box_units() (ael)	**Preference Functions**: Sets the units for the select region size, set by de_set_select_box_size(), at the current window.
de_set_select_color() (ael)	**Preference Functions**: Sets the color used to display selected objects for the current window.
de_set_select_filter() (ael)	**Preference Functions**: Sets the select filter mask for the current window.
de_set_select_inside_polygon() (ael)	**Preference Functions**: Sets the selection mode.
de_set_select_point_size() (ael)	**Preference Functions**: Sets the size of the selected vertices in the current window.

de_set_select_point_size_units() (ael)	**Preference Functions**: Sets the units for the selected vertices in the current window.
de_set_self_intersect() (ael)	**Preference Functions**: Sets the polygon self intersection checking for the current window.
de_set_shape_entry_mode() (ael)	**Preference Functions**: Sets the entry mode in the current window for polygons and polylines to orthogonal or non-orthogonal.
de_set_simulation_dataset() (ael)	**Command Functions**: Sets the name of the dataset used for simulation.
de_set_simulation_host() (ael)	**Command Functions**: Sets the name of the host computer to be used for simulation.
de_set_step_and_repeat() (ael)	**Preference Functions**: Sets variables for the Step and Repeat command.
de_set_swap_template_instance() (ael)	**Command Functions**: Sets the name of an item to swap within the function de_swap_instance().
de_set_tap_length() (ael)	**Preference Functions**: Sets the length (w3 parameters) of the tee element produced when a transmission line element is tapped with the de_tap_tlin() command.
de_set_tee_color() (ael)	**Preference Functions**: Sets the color of schematic tees (interconnect dots) for the current window.
de_set_tee_size() (ael)	**Preference Functions**: Sets the size of tees (interconnect dots) used in schematics (for the current window).
de_set_tee_size_units() (ael)	**Preference Functions**: Sets the type of units of the tee size.
de_set_text_absolute() (ael)	**Preference Functions**: Sets absolute characteristics for text.
de_set_text_angle() (ael)	**Preference Functions**: Sets the angle in which text is placed for the current window.
de_set_text_font() (ael)	**Preference Functions**: Sets the font type when placing text for the current window.
de_set_text_height() (ael)	**Preference Functions**: Sets the height for placed text (in the current window).
de_set_text_justification() (ael)	**Preference Functions**: Sets the default text justification.
de_set_text_string() (ael)	**Preference Functions**: Sets the text string used by the de_add_text() command for the current window.
de_set_trace_sim_mode() (ael)	**Preference Functions**: Sets the trace simulation mode for the current window.
de_set_trace_single_elem() (ael)	**Preference Functions**: Sets the name of the element used when trace simulation is set to a single element.
de_set_trace_tech() (ael)	**Preference Functions**: Sets the technology type for converting/simulating traces to microstrip, stripline, or PCB for the current window.
de_set_trace_traverse() (ael)	**Preference Functions**: Sets the trace traversal mode.
de_set_undo_edit_count() (ael)	**Preference Functions**: Sets the number of edit commands that can be undone.
de_set_warning_bell() (ael)	**Utility Functions**: Enables or disables the warning bell (on or off).
de_short_design_name() (ael)	**Design Environment Query Functions**: Strips off the fully qualified path to the design and any extension, returning just the name of the design.
de_shove() (ael)	**Command Functions**: Shove by a distance, vertices

	or whole polygons, whole text, and whole instances that are selected and lie on vector side of dividing line formed perpendicular to the vector.
de_show_context_in_new_window() (ael)	**DesignContext Functions**: Opens the given design in a new window and returns the window.
de_show_equiv_inst() (ael)	**Command Functions**: Highlights the equivalent instance in another representation to a given instance.
de_simple_dialog_cancel_cb() (ael)	**Utility Functions**: A general purpose cancel callback.
de_snap() (ael)	**Command Functions**: Force the vertices of selected shapes to the nearest snap grid; forces pin 1 of selected instances to nearest grid point.
de_split() (ael)	**Command Functions**: Divides selected polygons, rectangles, circles, or paths into multiple shapes using a defined rectangular region.
de_split_tlin() (ael)	**Command Functions**: (For Layout only.) Splits a transmission line element into two of the same elements at a given point, adjusting the new elements length parameters.
de_step_and_repeat() (ael)	**Command Functions**: Copies and places selected items multiple times in rows and columns.
de_store_current_view() (ael)	**Command Functions**: Assigns a name to a view and stores view window coordinates.
de_stretch() (ael)	**Command Functions**: Stretches or moves an edge of a given shape in the current representation.
de_stretch_dimlin() (ael)	**Command Functions**: Stretches a dimension line.
de_stretch_tlin() (ael)	**Command Functions**: (For Layout only) Stretches (increases or decreases its length) a given transmission line element in the current representation.
de_swap_instances() (ael)	**Command Functions**: Replaces all selected instances in the current window with the instance set with the function de_set_swap_template_instance().
de_tap_tlin() (ael)	**Command Functions**: (For Layout only) Taps a transmission line element (MLIN or SLIN) in the current representation, and inserts a tee element (MTEE or STEE).
de_touch() (ael)	**Preference Functions**: Selects or rejects polygons on one layer (inLayer1) in relation to edges of polygons on another layer (inLayer2) if TouchCondition is true.
de_tune() (ael)	**Simulator Command Functions**: Activates a tune analysis after updating parameters to the values specified in the paramList.
de_tune_deinit() (ael)	**Simulator Command Functions**: Terminates the tuning operation in the simulator.
de_undo() (ael)	**Command Functions**: Undoes the effect of the last draw or edit command in the current window.
de_undo_vertex() (ael)	**Command Functions**: Undoes or removes the last vertex entered with the de_add_point() command.
de_union() (ael)	**Command Functions**: Creates new polygons formed by the union of selected shapes (e.g., polygon, rectangle, circle, or path) on the same layer.
de_update_design_definition_ex() (ael)	**Component Definition Functions**: Updates the item definitions for all the instances within the given design context.
de_update_optimization_values() (ael)	**Simulator Command Functions**: Requests updated optimization/yield results from any optimization or

	yield analysis performed by the current server (simulator).
db_update_parameters_ex() (ael)	**Command Functions**: Modifies parameter values.
de_update_tune_parameters() (ael)	**Command Functions**: Modifies tuned parameter values.
de_valid_name() (ael)	**Utility Functions**: Returns TRUE or FALSE
de_version_number_int() (ael)	**Design Environment Query Functions**: Returns the current version number as an integer.
de_vertex_to_arc() (ael)	**Command Functions**: Converts a vertex on a shape to an arc.
de_view_all() (ael)	**Command Functions**: Expands the view window to view all data in the current window.
de_warning_bell() (ael)	**Utility Functions**: Sounds the warning bell.
de_window_has_valid_context() (ael)	**Design Context Functions**: Returns TRUE if the given window has a valid DesignContext.
de_window_is_open() (ael)	**Design Environment Query Functions**: Determines the status of a window.
de_write_preferences() (ael)	**Preference Functions**: Writes the current preference settings of the active window to a file.
de_zoom_in_point() (ael)	**Command Functions**: Zooms in (double magnification) at the location specified on the current window.
de_zoom_in_scale() (ael)	**Command Functions**: Zooms in by factor specified on the current window.
de_zoom_out_point() (ael)	**Command Functions**: Zooms out (double magnification) at the location specified on the current window.
de_zoom_out_scale() (ael)	**Command Functions**: Zooms out by factor specified on the current window.
de_zoom_window() (ael)	**Command Functions**: Describes a region to display in the current window.
deg() Function (ael)	**Math Functions**: Converts an angle from radians to degrees.
delete_nth() (ael)	**List Management Functions**: Deletes an item in a list, given an integer index into the list and an existing list.
delete_word() (ael)	**Utility Functions**: Deletes a word from the current vocabulary, or from a specified vocabulary.
dm_create_cb()	**Component Definition Functions**: Defines a callback function.
dm_find_form_definition() (ael)	**Component Definition Query Functions**: Returns a handle to a form definition given the form name and, optionally, a simulator ID.
dm_find_item_definition() (ael)	**Component Definition Query Functions**: Returns an handle to an item definition given the item's name.
dm_first_parm_definition() (ael)	**Component Definition Query Functions**: Returns the handle to the first parameter definition of an item, given a parameter definition handle as returned from dm_get_item_definition_attribute().
dm_form_get_attr() (ael)	**Form Definitions**: Returns the form attribute code for the given form.
dm_form_get_class() (ael)	**Form Definitions**: Returns the integer class code that represents the given form's class type.
dm_form_get_disp_format() (ael)	**Form Definitions**: Returns the display format string of the given form.

dm_form_get_label() (ael)	**Form Definitions**: Returns a string descriptive label of the given form.
dm_form_get_name() (ael)	**Form Definitions**: Returns a string with the name of the given form.
dm_form_get_net_format() (ael)	**Form Definitions**: Returns the netlist format string of the given form.
dm_form_get_parms() (ael)	**Form Definitions**: Returns the list of parameter definitions for the form.
dm_form_is_constant_form() (ael)	**Form Definitions**: Returns TRUE if the given form is a constant form.
dm_form_is_discrete() (ael)	**Form Definitions**: Returns TRUE if the given form is discrete.
dm_get_design_class_code() (ael)	**Component Definition Query Functions**: Returns a code representing the design class
dm_get_design_name() (ael)	**Component Definition Query Functions**: Returns a string, the generic group name defined in AEL for a particular type of design.
dm_get_form_definition_attribute() (ael)	**Component Definition Query Functions**: Returns the value of a form definition attribute, given a handle to the form definition.
dm_get_form_set_form_names() (ael)	**Form Set Functions**: Returns the list of string form names in the given form set.
dm_get_item_definition_attribute() (ael)	**Component Definition Query Functions**: Returns the value of an item definition attribute given a handle to the item.
dm_get_parm_definition_attribute() (ael)	**Component Definition Query Functions**: Returns a value of a parameter definition attribute, as defined with create_parm().
dm_get_parm_definition_forms() (ael)	**Parameter Definition Functions**: Returns a list of form definitions for the given parameter definition.
dm_get_simcode_from_designcode() (ael)	**Component Definition Query Functions**: Returns a simcode, which is required by functions such as dm_find_form_definition(),
dm_get_string_form_class_selection() (ael)	**Form Definitions**: Returns the list of string form names in the given form
dm_index_parm_definition() (ael)	**Component Definition Query Functions**: Returns a handle to the indexed parameter definition.
dm_item_get_artwork_data() (ael)	**Item Definition Functions**: Returns the string artwork data for an item definition.
dm_item_get_artwork_type() (ael)	**Item Definition Functions**: Returns the integer artwork type for an item definition.
dm_item_get_attr() (ael)	**Item Definition Functions**: Returns the item attribute code of an item definition.
dm_item_get_dialog_data() (ael)	**Item Definition Functions**: Returns the string dialog data for an item definition.
dm_item_get_dialog_name() (ael)	**Item Definition Functions**: Returns the string dialog name for an item definition.
dm_item_get_ex_attr() (ael)	**Item Definition Functions**: Returns the item extra attribute code of an item definition.
dm_item_get_icon_name() (ael)	**Item Definition Functions**: Returns the string icon name for an item definition.
dm_item_get_label() (ael)	**Item Definition Functions**: Returns the string descriptive label of an item definition.
dm_item_get_library_name() (ael)	**Item Definition Functions**: Returns the string library name of an item definition.
dm_item_get_name() (ael)	**Item Definition Functions**: Returns the string name

	of an item definition.
dm_item_get_netlist_data() (ael)	**Item Definition Functions**: Returns the netlist data string for an item definition.
dm_item_get_netlist_format() (ael)	**Item Definition Functions**: Returns the netlist format string for an item definition.
dm_item_get_parm_names() (ael)	**Item Definition Functions**: Returns a list of string parameter names for an item definition.
dm_item_get_parms() (ael)	**Item Definition Functions**: Returns the parameter definition list for an item definition.
dm_item_get_prefix() (ael)	**Item Definition Functions**: Returns the string instance prefix name for an item definition.
dm_item_get_priority() (ael)	**Item Definition Functions**: Returns the integer priority for an item definition.
dm_item_get_symbol_format() (ael)	**Item Definition Functions**: Returns the symbol format string for an item definition.
dm_item_get_symbol_name() (ael)	**Item Definition Functions**: Returns the string symbol name for an item definition.
dm_item_is_bom_item() (ael)	**Item Definition Functions**: Returns TRUE if an item has the isBom flag set.
dm_item_is_netlist_from_layout() (ael)	**Item Definition Functions**: Returns TRUE if an item is marked to be netlisted from layout
dm_item_is_port() (ael)	**Item Definition Functions**: Returns TRUE if an item is a Port
dm_item_is_sub_design() (ael)	**Item Definition Functions**: Returns TRUE if an item is a sub-circuit
dm_item_is_unique() (ael)	**Item Definition Functions**: Returns TRUE if an item is marked as a unique component
dm_item_is_variable() (ael)	**Item Definition Functions**: Returns TRUE if an item is marked as a variable (VAR) component
dm_item_no_special_characters() (ael)	**Item Definition Functions**: Returns TRUE if the item disallows special characters in the component's instance name
dm_next_parm_definition() (ael)	**Component Definition Query Functions**: Returns the next handle to the parameter definition of an item.
dm_num_parm_definitions() (ael)	**Component Definition Query Functions**: Returns the number of parameter definitions in a parameter list, given a parameter definition handle.
dm_parm_get_attr() (ael)	**Parameter Definition Functions**: Returns the attribute code of the given parameter definition.
dm_parm_get_defvalue() (ael)	**Parameter Definition Functions**: Returns the default parameter value for the given parameter definition.
dm_parm_get_formset_name() (ael)	**Parameter Definition Functions**: Returns the string name of the form set used for the given parameter definition.
dm_parm_get_label() (ael)	**Parameter Definition Functions**: Returns the label of the given parameter definition.
dm_parm_get_name() (ael)	**Parameter Definition Functions**: Returns the name of the parameter.
dm_parm_get_unit() (ael)	**Parameter Definition Functions**: Returns the unit code of the given parameter definition.

E

Name	Type and Description
error() (ael)	**Utility Functions**: Reports an error.
evaluate() (ael)	**Utility Functions**: Evaluates an AEL expression and returns the result.
execute() (ael)	**Utility Functions**: Interprets text as an AEL program.
exp() Function (ael)	**Math Functions**: Given an integer or real number as an exponent, returns e (~2.7183) raised to that exponent.
expandenv() (ael)	**Utility Functions**: Returns a string with the environment variables and EEsof configuration variables

F

Name	Type and Description
fclose() (ael)	**File Handling Functions**: Closes a file opened with fopen() command.
fflush() (ael)	**File Handling Functions**: Flushes buffers to disk when writing a file opened with fopen() for write or append.
fgets() (ael)	**File Handling Functions**: Reads a line from a file.
file_loaded() (ael)	**Utility Functions**: Tests to see whether an AEL file has been read.
filedate() (ael)	**Utility Functions**: Returns an integer that represents a time stamp of a file.
find_word() (ael)	**Utility Functions**: Returns the address of an AEL word if it exists
find_word_voc() (ael)	**Utility Functions**: Returns the address of a vocabulary in which a specified AEL word is found if it exists
fix() Function (ael)	**Math Functions**: Takes a real number argument, truncates it, and returns an integer value.
fix_path() (ael)	**Utility Functions**: Adds the appropriate backslashes needed in a string to handle control characters correctly.
float() Function (ael)	**Math Functions**: Converts an integer to a floating decimal number. Returns a real (floating-point) number.
floor() Function (ael)	**Math Functions**: Returns the largest integer not more than its argument from a real number.
fmt() (ael)	**String Functions**: Converts a float to a string.
fmt_tokens() (ael)	**String Functions**: Creates a string from a list of tokens.
fopen() (ael)	**File Handling Functions**: Opens a file and returns a file identifier.
format_date_time() (ael)	**Utility Functions**: Returns the date and time as a formatted string
fprintf() (ael)	**File Handling Functions**: Print formatted text to a file.
fputs() (ael)	**File Handling Functions**: Writes a value to the file or device specified.
freopen() (ael)	**File Handling Functions**: Opens the named file for reading or writing, attaching an existing file handle to it.

G

Name	Type and Description
generic_netlist_cb() (ael)	**Utility Functions**: This function allows the creation of a list of lists.
get_dir_files() (ael)	**Utility Functions**: Returns an AEL list of files found in a specified directory.
get_eqn_list() (ael)	**Design Environment Query Functions**: Returns a list of variables and equations for the current design.
get_item_list() (ael)	**Design Environment Query Functions**: Returns a string list of all components (instance names) matching a component name placed in the current design.
get_measurement_list() (ael)	**Design Environment Query Functions**: Returns a string list of all measurement components active for the current design.
getcwd() (ael)	**Utility Functions**: Returns a string representing the path name of the current working directory.
getenv() (ael)	**Utility Functions**: Returns a string representing the configuration variable value.
geterror() (ael)	**Utility Functions**: Returns a string that represents the last error message that occurred.
getppid() (ael)	**Utility Functions**: Returns an integer that represents the parent process ID of the current process.
getsysenv() (ael)	**Utility Functions**: Returns a string representing a system environment variable value.

I

Name	Type and Description
identify_value() (ael)	**Utility Functions**: Returns a single string representing the value(s) of any valid AEL object.
im() Function (ael)	**Math Functions**: Returns a real value, the imaginary component of a complex number.
imag() Function (ael)	**Math Functions**: Returns a real value, the imaginary component of a complex number.
index() Function (ael)	**String Functions**: Returns the position of the first occurrence of a string or character in a string
insert() (ael)	**List Management Functions**: Returns a new list with an inserted item, given an existing list and item to insert into the beginning of the list.
insert_nth() (ael)	**List Management Functions**: Returns a new list with an inserted item, given an index into the list, an existing list and an item to insert into the list.
int() Function (ael)	**Math Functions**: Returns the largest integer not greater than its real-number argument.
invoke_process() (ael)	**Simulator Command Functions**: Function to invoke a child process.
is_complex() (ael)	**Utility Functions**: Identifies whether a given value is a complex number.
is_dir() (ael)	**Utility Functions**: Determines if a given path is a directory.
is_file() (ael)	**Utility Functions**: Identifies whether a given value is a file identifier.
is_function() (ael)	**Utility Functions**: Identifies whether a given value is a function.
is_function_defined() (ael)	**Utility Functions**: Identifies whether a given value is a defined function.
is_integer() (ael)	**Utility Functions**: Identifies whether a given value is an integer number.
is_list() (ael)	**Utility Functions**: Identifies whether a given value is a list.
is_real() (ael)	**Utility Functions**: Identifies whether a given value is a real number.
is_string() (ael)	**Utility Functions**: Identifies whether a given value is a string.
is_type() (ael)	**Utility Functions**: Identifies whether a given value is of a given type.
is_voc() (ael)	**Utility Functions**: Identifies whether a given value is a vocabulary reference.

L

Name	Type and Description
leftstr() (ael)	**String Functions**: Returns the left portion of a string
library_group() (ael)	**Component Definition Functions**: Defines a new library group composed of the listed components.
list() Function (ael)	**List Management Functions**: Creates a list of items.
list_undefined() (ael)	**Utility Functions**: Interprets a string as an AEL command and returns an AEL list of undefined identifiers in that command.
listlen() (ael)	**List Management Functions**: Returns an integer representing the length of a list
ln() Function (ael)	**Math Functions**: Formerly log() in Series IV.
load() (ael)	**Utility Functions**: Loads an AEL file and interprets its statements.
log() Function (ael)	**Math Functions**: Returns the base 10 logarithm of an integer or real number.
log10() Function (ael)	**Math Functions**: Returns the base 10 logarithm of an integer or real number.
ly_find_layer_by_name() (ael)	**Preference Functions**: Retrieves a layer number given its name, for active layer set.
ly_find_layer_name_by_num() (ael)	**Preference Functions**: Retrieves a layer name by its layer number, for the active layer set.

M

Name	Type and Description
mag() Function (ael)	**Math Functions**: Returns the absolute/magnitude value of an integer, real, or complex number.
max2() Function (ael)	**Math Functions**: Returns the larger value of two numeric values, or NULL if parameters are invalid.
member() (ael)	**List Management Functions**: Tests whether an item is a member of a list.
midstr() (ael)	**String Functions**: Returns a piece of a string (the dissected string)
min2() Function (ael)	**Math Functions**: Returns the lesser value of two numeric values, or NULL if parameters are invalid.
mkdir() (ael)	**Utility Functions**: Creates a directory.

N

Name	Type and Description
nth() (ael)	**List Management Functions**: Takes a list and an integer index into the list.
nthcdr() (ael)	**List Management Functions**: Takes a list, and an integer index into the list.
num() Function (ael)	**Math Functions**: Returns an integer that represents an ASCII numeric value of the first character in the specified string.
num_args() (ael)	**Utility Functions**: Returns the number of arguments passed into a function.

O

Name	Type and Description
offset_array() (ael)	**Utility Functions**: Creates a new array by copying an existing AEL array
on_error() (ael)	**Utility Functions**: Sets the AEL function to be called in case of an error.

P

Name	Type and Description
parse() (ael)	**String Functions**: Parses a string into tokens, where each token is delimited by a blank, tab, or operator.
parse_blank() (ael)	**String Functions**: Parses a string of tokens separated by spaces or tabs into a list of strings.
pcb_get_form_value() (ael)	**Utility Functions**: Retrieves the value of a parameter that is a constant form.
pcb_get_mks() (ael)	**Utility Functions**: Retrieves the value of a parameter in MKS (unscaled) units.
pcb_get_parm_type() (ael)	**Utility Functions**: Returns parameter type.
pcb_get_string() (ael)	**Utility Functions**: Retrieves the value of a parameter that is a constant form.
pcb_set_form_value() (ael)	**Utility Functions**: Sets the value of a parameter that is a constant form.
pcb_set_mks() (ael)	**Utility Functions**: Sets the value of a parameter.
pcb_set_string() (ael)	**Utility Functions**: Sets the value of a parameter that is a constant form.
phase() Function (ael)	**Math Functions**: Returns a real number that represents the phase in degrees of the argument.
phasedeg() Function (ael)	**Math Functions**: Returns a real number that represents the phase in degrees of the argument.
phaserad() Function (ael)	**Math Functions**: Returns a real number that represents the phase in radians of the argument.
polar() Function (ael)	**Math Functions**: Converts polar magnitude and angle into a complex number.
pop_message_handler() (ael)	**Simulator Command Functions**: Restores a saved message handler for a server to the state previous to the last call to push_message_handler().
pow() Function (ael)	**Math Functions**: Returns an integer or real number raised to given power.
prm() (ael)	**Component Definition Functions**: Creates and returns a default parameter value.
push_message_handler() (ael)	**Simulator Command Functions**: Installs a message handler function to receive messages from a server.

R

Name	Type and Description
rad() Function (ael)	**Math Functions**: Converts angle from degrees to radians.
re() Function (ael)	**Math Functions**: Returns a real number, the real part of a complex value.
real() Function (ael)	**Math Functions**: Returns a real number, the real part of a complex value.
remov() (ael)	**List Management Functions**: Returns a new list with all the items copied from the original list except for the item specified in the first parameter.
remove() (ael)	**File Handling Functions**: Deletes a given file.
rename() (ael)	**File Handling Functions**: Renames a file.
rename_word() (ael)	**Utility Functions**: Finds an AEL word and renames it.
repla() (ael)	**List Management Functions**: Replaces an item in a list.
resize_array() (ael)	**Utility Functions**: Creates a new array by copying an existing AEL array with resizing of the specified dimension.
rightstr() (ael)	**String Functions**: Returns the right portion of a string value
round() Function (ael)	**Math Functions**: Returns an integer, a real number rounded to nearest integer value.

S

Name	Type and Description
send_server_command() (ael)	**Simulator Command Functions**: Sends a command to the server identified by server_handle.
send_server_data() (ael)	**Simulator Command Functions**: Sends data statement to the server identified by server_handle.
send_server_interrupt() (ael)	**Simulator Command Functions**: Sends an interrupt to the current server identified by server_handle.
send_server_kill() (ael)	**Simulator Command Functions**: Sends the special kill command ("KILL") to the server identified by server_handle.
server_running() (ael)	**Simulator Command Functions**: Returns the status of the server identified by serverHandle
set_design_choices() (ael)	**Component Definition Functions**: Defines choices for characteristics allowed for parametric subnetwork components.
set_design_sub_choices() (ael)	**Component Definition Functions**: Defines sub-choices for each characteristic choice for parametric subnetwork components.
set_design_type() (ael)	**Component Definition Functions**: Sets the current type of design for group definitions.
set_netlist_info() (ael)	**Component Definition Functions**: Sets special netlisting characteristics for current type of design.
set_simulator_type() (ael)	**Component Definition Functions**: Sets the current simulator for ADS 2003A and earlier.
set_user_menu_label() (ael)	**User Interface Function**: Sets the label for the user menu.
setenv() (ael)	**Utility Functions**: Sets the value of a named configuration variable.
sgn() Function (ael)	**Math Functions**: Returns the integer sign of an integer or real number, as either 1 or -1.
sin() Function (ael)	**Math Functions**: Returns the sine of a real number (in radians).
sinc() Function (ael)	**Math Functions**: Returns a real number that represents sin(parm)/parm in radians.
sinh() Function (ael)	**Math Functions**: Returns the hyperbolic sine of an integer, real, or complex number.
sleep() (ael)	**Utility Functions**: Sets an idle time for the program.
sort_list() (ael)	**List Management Functions**: Returns a new list with members of a given list in sorted order.
sprintf() Function (ael)	**String Functions**: Format text values into a string.
sqrt() Function (ael)	**Math Functions**: Returns the square root of a positive integer or real number.
start_server() (ael)	**Simulator Command Functions**: Causes the server identified by serverHandle to be spawned.
start_timer() (ael)	**Utility Functions**: Records the current timestamp internally.
step() Function (ael)	**Math Functions**: A step function that returns 0, 0.5, or 1.
strcasecmp() (ael)	**String Functions**: Compares two strings, ignoring differences in case between them.
strcat() Function (ael)	**String Functions**: Appends two or more strings to the end of the first, resulting in a single string.
strcmp() (ael)	**String Functions**: Compares two strings.
stripstr() (ael)	**String Functions**: Returns a string with leading and trailing blanks and tabs removed.
strlen() (ael)	**String Functions**: Returns the length of a string, as an integer.
system() (ael)	**Utility Functions**: Executes a system command and returns the command exit status.

T

Name	Type and Description
tan() Function (ael)	**Math Functions**: Returns the tangent of a real number (in radians).
tanh() Function (ael)	**Math Functions**: Returns the hyperbolic tangent of an integer, real, or complex number.
tmpnam() (ael)	**Utility Functions**: Generates the name of a unique temporary file.
tolower() (ael)	**String Functions**: Converts a string to lowercase and return the converted string.
total_elapsed_time() (ael)	**Utility Functions**: Returns the time elapsed since the last start_timer() was called as an AEL list of length 2.
toupper() (ael)	**String Functions**: Converts a string to uppercase and return the converted string.

V

Name	Type and Description
val() (ael)	**String Functions**: Returns a real number, the numeric value of an ASCII string.
validate_name() (ael)	**Utility Functions**: Verifies that a string is a valid program file name

W

Name	Type and Description
warning() (ael)	**Utility Functions**: Issues a warning message which is then printed in the warning dialog.
what_col() (ael)	**Utility Functions**: Returns the column number of the caller of the current function as an integer.
what_file() (ael)	**Utility Functions**: Returns the file name of the caller of the current function as a string
what_function() (ael)	**Utility Functions**: Takes an optional integer parameter to indicate what level of the function stack to extract.
what_line() (ael)	**Utility Functions**: Returns the line number of the caller of the current function as an integer.

X

Name	Type and Description
xor() Function (ael)	**Math Functions**: Returns an integer that represents the exclusive OR between arguments.

Using AEL Database Retrieval Functions

This section describes how you can extract design information from the Design Environment (DE) by using the AEL database retrieval (db) functions.

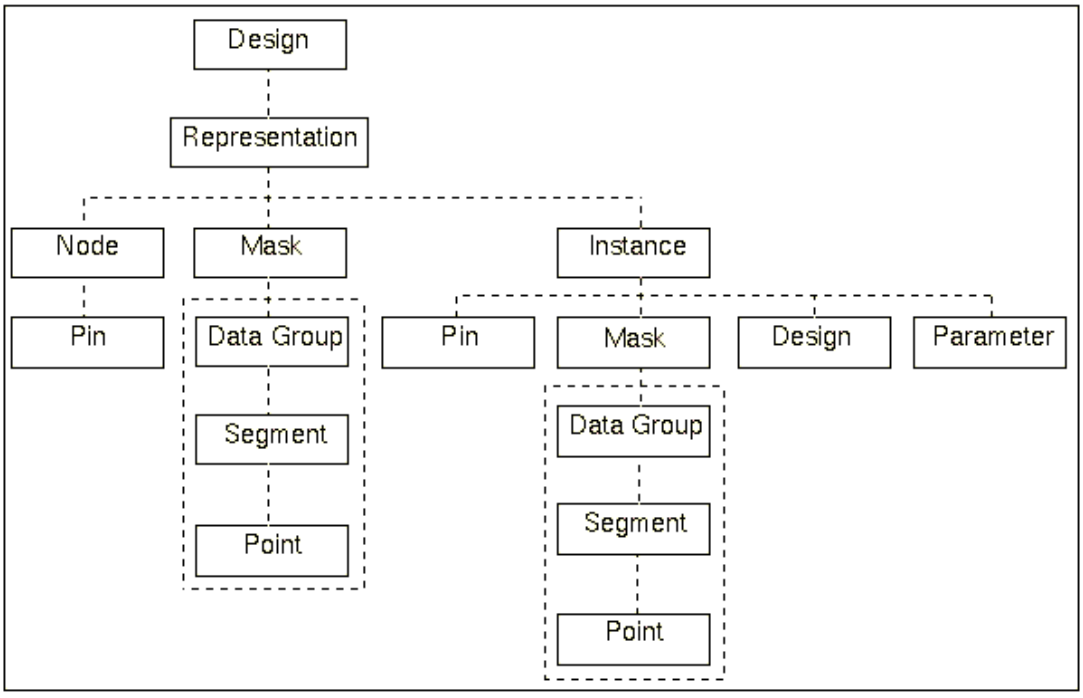
Database Overview

The database is structured around a hierarchy of objects, with the design object at the top. Every design contains two representations, a schematic and layout. In turn, these representations are composed of masks, nodes, pins, and instances. Masks contain shapes and text (data groups). Nodes, pins, and instances maintain connectivity. Instances can refer to another design and/or contain their own masks. Properties may be added to each of these database objects.

The following figure shows the design hierarchy and relationships among the design objects. For all objects, except representations, there can be a *list* of zero or more occurrences of the object. For example, a representation or instance may have a list of zero or more mask objects, a mask may have zero or more data groups. Each type of object is described briefly in the following sections.

Design

A *design* represents a complete Analog/RF or Digital Signal Processing design. Each design is stored in a separate file. Every design has two representations: a layout and a schematic. Either representation can be empty. Designs can reference other designs hierarchically through instances. There is no limit to the number of designs loaded in the program. When a hierarchical design is loaded, all the referenced designs are loaded as well.



Representation

Representations describe either the layout or the schematic of an Analog/RF or Digital Signal Processing design. The design synchronization engine (DSE) ensures or checks that the two are equivalent. Representations own masks (also called layers), instances, and nodes. Representations also own symbols (not shown on the chart). Symbols, used in schematics only, display a schematic in a hierarchy. A symbol owns pins and masks.

Mask

Masks (sometimes termed layers) group shapes and text. The only attribute of interest in a mask object is its number. The number is used as an index into a layer table, which in turn tells the program what color, name, fill pattern, etc. to use when displaying the objects grouped by the mask.

Node

A node, also known as *net*, represents an electrical connection between pins. Nodes are established by abutting pins of instances or creating a wire, naming a node, or trace that connects one or more pins. Nodes contain pointers to the pins they interconnect.

Instance

An instance represents either a primitive simulation element or a reference to another design. The term *instance* is also called *component* in the program and documentation. A resistor is represented as a primitive instance, while a subnetwork placed in another design is represented by a hierarchical instance. Regardless whether an instance is hierarchical or primitive, it can own a list of pins. Pins represent connection points for the instance. Interconnecting or abutting the pins of two instances forms an electrical connection (a node). Primitive instances can own masks. This is usually seen only in the layout representation. A microstrip transmission line instance owns the mask that describes its geometry. Instances can also own parameters. The type, name and number of parameters owned by an instance is dictated by an instance's component definition. Components are defined with the AEL *create_item()* and *create_param()* functions. When a new design is saved, a corresponding AEL definition is saved and used whenever an instance to the design is created.

Parameter

A parameter has a name and a value. The value of some parameters can be quite complex, so each value has a type. The simplest type is a single number, but some values

are composed of a pair of numbers, a triple, a list of numbers, a string, etc. For example, the default value of a resistor's parameter *R* for resistance is the string "50 ohm." Stored with each parameter is a form name that was specified in the *create_parm()* function. The form name indicates how to interpret the type of a parameter's value. A number of AEL functions are supplied that simplify the extraction of parameters and their values.

Pin

A pin represents an electrical connection point on an instance. Abutting or wiring pins together forms an electrical connection (called a *node*). In hierarchical layout representations, a pin is created for the instance for each port on the referenced design. In hierarchical schematic representations, the symbol for the referenced design should have the same number of pins as the referenced design. The two are matched by pin number. Pins maintain a handle to any connecting wire or trace.

Data Group

A data group represents a shape or text. Shapes have a type that indicates whether the shape is a wire or trace, polygon, polyline, rectangle, circle, arc, construction line, or text. Polygons and polylines are composed of one or more segments.

Segment

A segment can be a list of points or an arc. Segments have the attribute of being *filled* or *empty*. An empty segment must be enclosed by a filled segment, since it represents a hole. Segments are composed of a list of one or more points.

Point

A point is an X,Y coordinate. These coordinates are integers in the range of roughly plus or minus 2 billion. Coordinates stored in the database are stored in database units. The number of database units to each user unit is controlled by a representation's precision. If mil is specified as the layout unit, the default is to store 1 mil as the integer 100, 0.5 mil would be stored as the integer 50.

Property

A property is a pair of a name and value. A property value type can be a long, double or string. An unlimited number of properties may be added to the following database objects: designs, representations, symbols, masks, nodes, instances, pins, data groups and segments.

Traversing a Design

For every object there is a traversal function that returns a handle to the object. Handles are like C pointers, in that they provide a way to uniquely identify an object and retrieve information about it. Generally these functions come in pairs, a *get_first()* and a *get_next()*. The *get_first()* function takes a handle to the object's parent, while the *get_next()* function takes a handle to the previous object in the list. This example uses these functions to traverse the list of data groups belonging to a mask object and count them.

```
decl cnt, maskHandle, dgHandle;
cnt = 0;
dgHandle = db_first_dg( maskHandle );
while(  dgHandle )
{
  cnt = cnt + 1;
  dgHandle = db_next_dg( dgHandle );
}
```

Besides traversal functions, each object has an attribute retrieval function. Attributes are the actual data, such as location, select status, bounding box, mask layer, pin number, etc.

The AEL function reference for an object's attribute function supplies a list of attributes that can be retrieved from each kind of object. After you have a handle to an object, you can retrieve any or all of its attributes. An attribute that is shared by both instances and data groups is the selected status attribute. By querying this attribute you can determine if an object has been selected.

Some objects, such as instances, have a large number of attributes. The example code segment demonstrates extracting information from an instance using the *db_get_instance_attribute()* query function.

```
instHandle = db_first_instance(repHandle);
while (instHandle)
{
designName=db_get_instance_attribute(instHandle,INST_DESIGN_NAME);
  instName   = db_get_instance_attribute(instHandle, INST_NAME);
  fputs( stderr, fmt_tokens( list( designName, " ", instName) ) );
  instHandle = db_next_instance( instHandle );
```

Using the database query functions can be complex. For a guide, you can use one of the examples supplied, such as de_bom.ael, which traverses the instances of a design. This file can be found in the installation sub-directory $HPEESOF_DIR/de/ael.

Note
Be cautious when mixing query functions with database modification functions. As a design is modified, handles to the object in the database change and existing handles become invalid. After any modification, you should retraverse the database from the representation level on down to get new handles. Most modifications invalidate object handles.

Besides the list traversal functions, some functions retrieve design and instance information by name, rather than by handle. Examples are:

db_find_instance()
db_find_property()
db_get_instance_parm()

Details on these functions can be found in *Database Query and Manipulation Functions* (ael).

Where to Use AEL Functions

The following table shows where each function in *AEL* (ael) can be used.

- *Measurement Expressions (Data Display equations and Schematic MeasEqns)*—Post-process simulation data or write your own functions to extend the functionality beyond the built-in expressions. In addition to the functions marked *Measurement Expressions*, you can also use the functions in the *Measurement Expressions* (expmeas) documentation.
- *Schematic*—Manipulate the schematic, for example, create or edit component definitions, do database traversals, modify designs and more.
- *Layout*—Manipulate the layout, for example, create dynamic shapes by writing custom artwork macros and more. For more information, see *Layout Library* (layout).
- *Simulation*—Start and control simulations.
- *GUI*—Customize ADS's graphical user interface (GUI). For more information, see *Customization and Configuration* (custom).

Function	Measurement Expressions	Schematic	Layout	Simulation	GUI
abs() (ael)	X	X	X	X	X
acos() (ael)	X	X	X	X	X
acosh() (ael)	X	X	X	X	X
acot() (ael)	X	X	X	X	X
acoth() (ael)	X	X	X	X	X
add_menu() (ael)					X
add_separator() (ael)					X
ael_gfile_hasext() (ael)	X	X	X	X	X
api_dlg_unmanage() (ael)					X
api_get_current_window() (ael)					X
api_get_graph_color_index_by_name() (ael)					X
api_get_graph_color_name_by_index() (ael)					X
api_get_graph_pattern_index_by_name() (ael)					X
api_get_graph_pattern_name_by_index() (ael)					X
api_get_window_graph() (ael)					X
api_set_current_window() (ael)					X
api_set_current_window_by_seq_num() (ael)					X
append() (ael)	X	X	X	X	X
arg() (ael)	X	X	X	X	X
arg_list() (ael)	X	X	X	X	X
array_lowerBound() (ael)	X	X	X	X	X
array_size() (ael)	X	X	X	X	X
array_type() (ael)	X	X	X	X	X
array_upperBound() (ael)	X	X	X	X	X
asin() (ael)	X	X	X	X	X
asinh() (ael)	X	X	X	X	X
atan() (ael)	X	X	X	X	X
atan2() (ael)	X	X	X	X	X
atanh() (ael)	X	X	X	X	X
call() (ael)	X	X	X	X	X

call_depth() (ael)	X	X	X	X	X
car() (ael)	X	X	X	X	X
cdr() (ael)	X	X	X	X	X
ceil() (ael)	X	X	X	X	X
chdir() (ael)	X	X	X	X	X
check_syntax() (ael)	X	X	X	X	X
check_user_menu() (ael)					X
chmod() (ael)	X	X	X	X	X
chr() (ael)	X	X	X	X	X
cint() (ael)	X	X	X	X	X
clear_server() (ael)	X	X	X	X	X
clear_user_menu() (ael)					X
cmplx() (ael)	X	X	X	X	X
conj() (ael)	X	X	X	X	X
cons() (ael)	X	X	X	X	X
convBin() (ael)	X	X	X	X	X
convert_array() (ael)	X	X	X	X	X
convHex() (ael)	X	X	X	X	X
convOct() (ael)	X	X	X	X	X
cos() (ael)	X	X	X	X	X
cosh() (ael)	X	X	X	X	X
cot() (ael)	X	X	X	X	X
coth() (ael)	X	X	X	X	X
create_compound_form() (ael)		X			
create_constant_form() (ael)		X			
create_form_set() (ael)		X			
create_item() (ael)		X			
create_parm() (ael)		X			
create_server() (ael)	X	X	X	X	X
create_text_form() (ael)		X			
date_time() (ael)	X	X	X	X	X
dB() (ael)	X	X	X	X	X
db_clear_map() (ael)		X	X		
db_factor() (ael)		X	X		
db_first_parm() (ael)		X	X		
db_get_bbox_x1() (ael)		X	X		
db_get_bbox_x2() (ael)		X	X		
db_get_bbox_y1() (ael)		X	X		
db_get_bbox_y2() (ael)		X	X		
db_get_instance_bbox() (ael)		X	X		
db_get_instance_description() (ael)		X	X		
db_get_instance_parm() (ael)		X	X		
db_get_location_angle() (ael)		X	X		
db_get_location_x() (ael)		X	X		
db_get_location_y() (ael)		X	X		
db_get_map() (ael)		X	X		
db_get_map_attribute() (ael)		X	X		
db_get_node_wires() (ael)		X	X		

db_get_parm_attribute() (ael)		X	X		
db_get_pin_attribute() (ael)		X	X		
db_get_transform_angle() (ael)		X	X		
db_get_transform_mirror_x() (ael)		X	X		
db_get_transform_mirror_y() (ael)		X	X		
db_get_transform_x() (ael)		X	X		
db_get_transform_y() (ael)		X	X		
db_get_x() (ael)		X	X		
db_get_y() (ael)		X	X		
db_next_parm() (ael)		X	X		
db_num_parms() (ael)		X	X		
db_set_map() (ael)		X	X		
db_setup_map() (ael)		X	X		
db_setup_transform() (ael)		X	X		
db_transform_angle() (ael)		X	X		
db_transform_coord() (ael)		X	X		
dBm() (ael)	X	X	X	X	X
de_activate() (ael)		X	X		
de_add_arc() (ael)		X	X		
de_add_arc1() (ael)		X	X		
de_add_arc2() (ael)		X	X		
de_add_arc3() (ael)		X	X		
de_add_arc4() (ael)		X	X		
de_add_circle() (ael)		X	X		
de_add_construction_line() (ael)		X	X		
de_add_path() (ael)		X	X		
de_add_point() (ael)		X	X		
de_add_polygon() (ael)		X	X		
de_add_polyline() (ael)		X	X		
de_add_property() (ael)		X	X		
de_add_rectangle() (ael)		X	X		
de_add_text() (ael)		X	X		
de_add_trace() (ael)		X	X		
de_add_vertex() (ael)		X	X		
de_add_wire() (ael)		X	X		
de_add_wire_label() (ael)		X	X		
de_analyze() (ael)				X	
de_analyze_tune() (ael)				X	
de_ang_factor() (ael)		X	X		
de_bom() (ael)		X	X		
de_boolean_logical() (ael)			X		
de_break_connection() (ael)		X	X		
de_change_annotation_layer() (ael)		X			
de_check_rep_options() (ael)		X	X		
de_chop() (ael)			X		
de_clear_dc_annotation() (ael)		X			
de_clear_highlighting() (ael)		X	X		

de_close_all() (ael)		X	X		
de_close_all_datadisplay() (ael)		X			
de_close_window() (ael)		X	X		
de_connect() (ael)		X	X		
de_convert_path_to_trace() (ael)			X		
de_convert_to_polygon() (ael)		X	X		
de_convert_trace_to_path() (ael)			X		
de_convert_traces_to_instances() (ael)			X		
de_copy() (ael)		X	X		
de_copy_file() (ael)		X	X		
de_copy_to_buffer() (ael)		X	X		
de_copy_to_layer() (ael)		X	X		
de_create_window() (ael)		X	X		
de_crop() (ael)			X		
de_current_design_type() (ael)		X	X		
de_data_dialog() (ael)					X
de_dc_annotation() (ael)		X			
de_deactivate() (ael)		X	X		
de_define_edge_area_port() (ael)			X		
de_define_npport() (ael)			X		
de_define_palette_group() (ael)		X			
de_define_port() (ael)			X		
de_delete() (ael)		X	X		
de_delete_all_orphaned_instances() (ael)		X	X		
de_delete_view() (ael)		X	X		
de_deselect_all() (ael)		X	X		
de_deselect_all_force() (ael)		X	X		
de_deselect_by_name() (ael)		X	X		
de_deselect_window() (ael)		X	X		
de_difference() (ael)			X		
de_draw_arc() (ael)		X	X		
de_draw_arc1() (ael)		X	X		
de_draw_arc2() (ael)		X	X		
de_draw_arc3() (ael)		X	X		
de_draw_arc4() (ael)		X	X		
de_draw_circ() (ael)		X	X		
de_draw_point() (ael)		X	X		
de_draw_port() (ael)			X		
de_draw_rect() (ael)		X	X		
de_draw_text() (ael)		X	X		
de_dse_l2s() (ael)			X		
de_dse_s2l() (ael)		X			
de_edit_annotation_attribute() (ael)		X			
de_edit_item() (ael)		X	X		
de_edit_path_trace() (ael)			X		
de_edit_symbol_pin() (ael)		X	X		
de_edit_text_attribute() (ael)		X	X		
de_edit_text_string() (ael)		X	X		

de_empty() (ael)		X	X		
de_end() (ael)		X	X		
de_end_command() (ael)		X	X		
de_end_edit_item() (ael)		X	X		
de_error_bell() (ael)		X			
de_exit() (ael)		X			
de_export_design() (ael)		X	X		
de_fill() (ael)		X	X		
de_find_arc_center() (ael)		X	X		
de_find_line_center() (ael)		X	X		
de_find_pin() (ael)		X	X		
de_find_vertex() (ael)		X	X		
de_fix_instances() (ael)		X	X		
de_flatten() (ael)		X	X		
de_free_instances() (ael)		X	X		
de_free_item() (ael)		X	X		
de_get_alternate_window() (ael)		X	X		
de_get_data_parm() (ael)		X	X		
de_get_design_instances() (ael)		X	X		
de_get_env() (ael)		X			
de_get_file_names() (ael)		X	X		
de_get_preference() (ael)		X	X		
de_get_variable_names() (ael)		X			
de_get_window() (ael)		X	X		
de_group_edit_parameter_value() (ael)		X	X		
de_hide_or_display_component_name()) (ael)		X			
de_import_design() (ael)		X	X		
de_info() (ael)					X
de_init_item() (ael)		X	X		
de_insert_arrow() (ael)		X	X		
de_insert_dimlin() (ael)			X		
de_instantiate() (ael)		X	X		
de_intersection() (ael)			X		
de_invoke_help() (ael)		X	X		
de_last_view() (ael)		X	X		
de_mirror_x() (ael)		X	X		
de_mirror_y() (ael)		X	X		
de_miter_vertex() (ael)			X		
de_modify_arc_resolution() (ael)		X	X		
de_modify_break() (ael)		X	X		
de_modify_circle_radius() (ael)		X	X		
de_modify_explode() (ael)		X	X		
de_modify_join() (ael)		X	X		
de_move() (ael)		X	X		
de_move_annotation() (ael)		X			
de_move_break() (ael)		X	X		
de_move_to_layer() (ael)		X	X		

de_net() (ael)		X	X		
de_new_datadisplay() (ael)		X	X		
de_open_datadisplay() (ael)		X			
de_open_hierarchy_dialog() (ael)					X
de_open_info_dialog() (ael)					X
de_oversize() (ael)		X	X		
de_pan_window() (ael)		X	X		
de_parts() (ael)		X	X		
de_parts_option_add_exclusion_items() (ael)		X	X		
de_parts_option_add_inclusion_items() (ael)		X	X		
de_parts_option_check_bom() (ael)		X	X		
de_parts_option_include_header() (ael)		X	X		
de_parts_option_set_attribute_columns() (ael)		X	X		
de_parts_option_set_center_placement() (ael)		X	X		
de_parts_option_set_delimeter() (ael)		X	X		
de_parts_option_set_hierarchical() (ael)		X	X		
de_parts_option_set_package_offset() (ael)		X	X		
de_parts_option_sort_by_component() (ael)		X	X		
de_paste_from_buffer() (ael)		X	X		
de_place_design_template() (ael)		X			
de_place_item() (ael)		X	X		
de_place_port() (ael)		X	X		
de_place_unplaced() (ael)		X	X		
de_playback_macro() (ael)		X	X		
de_plot() (ael)		X	X		
de_plot_to_file() (ael)		X	X		
de_pop_outof_instance() (ael)		X	X		
de_post_help() (ael)		X	X		
de_print_info() (ael)					X
de_prompt() (ael)					X
de_push_into_instance() (ael)		X	X		
de_question() (ael)					X
de_read_preferences() (ael)		X	X		
de_refresh_layers() (ael)		X	X		
de_refresh_view() (ael)		X	X		
de_release_simulator() (ael)		X			
de_remove_properties() (ael)		X	X		
de_restore_all_datadisplay() (ael)		X			
de_restore_status() (ael)		X			
de_restore_view() (ael)		X	X		
de_retrieve_version_info() (ael)		X	X		
de_rotate() (ael)		X	X		
de_rotate_90() (ael)		X	X		
de_rotate_center() (ael)		X	X		
de_rotate_image() (ael)		X	X		
de_save_all_designs() (ael)		X	X		
de_save_design_template() (ael)		X			
de_scale() (ael)		X	X		

de_search_and_replace() (ael)		X	X		
de_select_all() (ael)		X	X		
de_select_all_force() (ael)		X	X		
de_select_all_on_layer() (ael)		X	X		
de_select_by_name() (ael)		X	X		
de_select_item() (ael)		X	X		
de_select_range() (ael)		X	X		
de_select_unplaced() (ael)		X	X		
de_select_window() (ael)		X	X		
de_set_annotation_font() (ael)		X			
de_set_annotation_height() (ael)		X			
de_set_annotation_id_layer() (ael)		X			
de_set_annotation_name_layer() (ael)		X			
de_set_annotation_parameters_layer() (ael)		X			
de_set_annotation_precision() (ael)		X			
de_set_annotation_rows() (ael)		X			
de_set_arc_radius() (ael)		X	X		
de_set_background_color() (ael)		X	X		
de_set_backup_count() (ael)		X	X		
de_set_coord_entry_popup() (ael)		X	X		
de_set_curve_radius() (ael)		X	X		
de_set_design_template() (ael)		X	X		
de_set_drag_move() (ael)		X	X		
de_set_drag_move_size() (ael)		X	X		
de_set_drag_move_units() (ael)		X	X		
de_set_dse_start() (ael)		X	X		
de_set_dual_placement() (ael)		X	X		
de_set_edit_property() (ael)		X	X		
de_set_edit_symbol_pin() (ael)		X	X		
de_set_edit_text() (ael)		X	X		
de_set_error_bell() (ael)		X			
de_set_foreground_color() (ael)		X	X		
de_set_global_db_factor() (ael)			X		
de_set_grid_color() (ael)		X	X		
de_set_grid_display_type() (ael)		X	X		
de_set_grid_snap() (ael)		X	X		
de_set_grid_snap_mode() (ael)		X	X		
de_set_grid_snap_type() (ael)		X	X		
de_set_highlight_color() (ael)		X	X		
de_set_inst_pin_order_property() (ael)		X	X		
de_set_item_id() (ael)		X	X		
de_set_item_parameters() (ael)		X	X		
de_set_layer() (ael)		X	X		
de_set_major_grid_display() (ael)		X	X		
de_set_minor_grid_display() (ael)		X	X		
de_set_miter_cutoff() (ael)			X		
de_set_miter_length() (ael)			X		

de_set_move_annotation() (ael)		X			
de_set_origin() (ael)		X	X		
de_set_oversize() (ael)		X	X		
de_set_path_corner() (ael)			X		
de_set_path_width() (ael)			X		
de_set_pin_color() (ael)		X	X		
de_set_pin_size() (ael)		X	X		
de_set_pin_size_units() (ael)		X	X		
de_set_pin_snap() (ael)		X	X		
de_set_pin_snap_units() (ael)		X	X		
de_set_place_popup_mode() (ael)		X	X		
de_set_place_popup_on_zero_parm() (ael)		X	X		
de_set_plot_pin_names() (ael)		X	X		
de_set_plot_pin_numbers() (ael)		X	X		
de_set_plot_pins() (ael)		X	X		
de_set_plotting_depth() (ael)			X		
de_set_port() (ael)		X	X		
de_set_port_size() (ael)		X	X		
de_set_port_size_units() (ael)		X	X		
de_set_preference() (ael)		X	X		
de_set_reroute_wires() (ael)		X	X		
de_set_resolution_for_arc() (ael)		X	X		
de_set_rotation_increment() (ael)		X	X		
de_set_route_around_annot() (ael)		X			
de_set_scale() (ael)		X	X		
de_set_select_box_size() (ael)		X	X		
de_set_select_box_units() (ael)		X	X		
de_set_select_color() (ael)		X	X		
de_set_select_filter() (ael)		X	X		
de_set_select_inside_polygon() (ael)		X	X		
de_set_select_point_size() (ael)		X	X		
de_set_select_point_size_units() (ael)		X	X		
de_set_self_intersect() (ael)		X	X		
de_set_shape_entry_mode() (ael)		X	X		
de_set_simulation_dataset() (ael)		X			
de_set_simulation_host() (ael)		X			
de_set_step_and_repeat() (ael)		X	X		
de_set_swap_template_instance() (ael)		X	X		
de_set_tap_length() (ael)			X		
de_set_tee_color() (ael)		X	X		
de_set_tee_size() (ael)		X	X		
de_set_tee_size_units() (ael)		X	X		
de_set_text_absolute() (ael)		X	X		
de_set_text_angle() (ael)		X	X		
de_set_text_font() (ael)		X	X		
de_set_text_height() (ael)		X	X		
de_set_text_justification() (ael)		X	X		
de_set_text_string() (ael)		X	X		

de_set_trace_sim_mode() (ael)		X	X		
de_set_trace_single_elem() (ael)		X	X		
de_set_trace_tech() (ael)			X		
de_set_trace_traverse() (ael)			X		
de_set_undo_edit_count() (ael)		X	X		
de_warning_bell() (ael)		X			
de_short_design_name() (ael)		X	X		
de_shove() (ael)		X	X		
de_show_equiv_inst() (ael)		X	X		
de_simple_dialog_cancel_cb() (ael)		X			
de_snap() (ael)		X	X		
de_split() (ael)			X		
de_split_tlin() (ael)			X		
de_step_and_repeat() (ael)		X	X		
de_store_current_view() (ael)		X	X		
de_stretch() (ael)		X	X		
de_stretch_dimlin() (ael)			X		
de_stretch_tlin() (ael)			X		
de_swap_instances() (ael)		X	X		
de_tap_tlin() (ael)			X		
de_tune() (ael)				X	
de_touch() (ael)			X		
de_tune_deinit() (ael)				X	
de_undo() (ael)		X	X		
de_undo_vertex() (ael)		X	X		
de_unhighlight_instances() (ael)		X	X		
de_union() (ael)			X		
de_update_optimization_values() (ael)		X			
de_update_parameters() (ael)		X	X		
de_update_tune_parameters() (ael)		X	X		
de_valid_name() (ael)		X			
de_version_number_int() (ael)		X	X		
de_vertex_to_arc() (ael)		X	X		
de_view_all() (ael)		X	X		
de_warning_bell() (ael)		X			
de_write_preferences() (ael)		X	X		
de_zoom_in_point() (ael)		X	X		
de_zoom_in_scale() (ael)		X	X		
de_zoom_out_point() (ael)		X	X		
de_zoom_out_scale() (ael)		X	X		
de_zoom_window() (ael)		X	X		
deg() (ael)	X	X	X	X	X
delete_nth() (ael)	X	X	X	X	X
delete_word() (ael)	X	X	X	X	X
dm_create_cb()		X			
dm_find_form_definition() (ael)		X			
dm_find_item_definition() (ael)		X			

dm_first_parm_definition() (ael)		X			
dm_get_design_class_code() (ael)		X			
dm_get_design_name() (ael)		X			
dm_get_form_definition_attribute() (ael)		X			
dm_get_item_definition_attribute() (ael)		X			
dm_get_parm_definition_attribute() (ael)		X			
dm_get_simcode_from_designcode() (ael)		X			
dm_index_parm_definition() (ael)		X			
dm_next_parm_definition() (ael)		X			
dm_num_parm_definitions() (ael)		X			
error() (ael)	X	X	X	X	X
evaluate() (ael)	X	X	X	X	X
execute() (ael)	X	X	X	X	X
exp() (ael)	X	X	X	X	X
expandenv() (ael)	X	X	X	X	X
fclose() (ael)	X	X	X	X	X
fflush() (ael)	X	X	X	X	X
fgets() (ael)	X	X	X	X	X
file_loaded() (ael)	X	X	X	X	X
filedate() (ael)	X	X	X	X	X
find_word() (ael)	X	X	X	X	X
find_word_voc() (ael)	X	X	X	X	X
fix() (ael)	X	X	X	X	X
fix_path() (ael)	X	X	X	X	X
float() (ael)	X	X	X	X	X
floor() (ael)	X	X	X	X	X
fmt() (ael)	X	X	X	X	X
fmt_tokens() (ael)	X	X	X	X	X
fopen() (ael)	X	X	X	X	X
format_date_time() (ael)	X	X	X	X	X
fprintf() (ael)	X	X	X	X	X
fputs() (ael)	X	X	X	X	X
freopen() (ael)	X	X	X	X	X
get_dir_files() (ael)	X	X	X	X	X
get_eqn_list() (ael)		X			
get_item_list() (ael)		X	X		
get_measurement_list() (ael)		X			
getcwd() (ael)	X	X	X	X	X
getenv() (ael)	X	X	X	X	X
geterror() (ael)	X	X	X	X	X
getppid() (ael)	X	X	X	X	X
getsysenv() (ael)	X	X	X	X	X
identify_value() (ael)	X	X	X	X	X
im() (ael)	X	X	X	X	X
imag() (ael)	X	X	X	X	X
index() (ael)	X	X	X	X	X
insert() (ael)	X	X	X	X	X
insert_nth() (ael)	X	X	X	X	X

int() (ael)	X	X	X	X	X
is_complex() (ael)	X	X	X	X	X
is_dir() (ael)	X	X	X	X	X
is_file() (ael)	X	X	X	X	X
is_function() (ael)	X	X	X	X	X
is_function_defined() (ael)	X	X	X	X	X
is_integer() (ael)	X	X	X	X	X
is_list() (ael)	X	X	X	X	X
is_real() (ael)	X	X	X	X	X
is_string() (ael)	X	X	X	X	X
is_type() (ael)	X	X	X	X	X
is_voc() (ael)	X	X	X	X	X
leftstr() (ael)	X	X	X	X	X
library_group() (ael)		X			
list() (ael)	X	X	X	X	X
list_undefined() (ael)	X	X	X	X	X
listlen() (ael)	X	X	X	X	X
ln() (ael)	X	X	X	X	X
load() (ael)	X	X	X	X	X
log() (ael)	X	X	X	X	X
log10() (ael)	X	X	X	X	X
ly_find_layer_by_name() (ael)		X	X		
ly_find_layer_name_by_num() (ael)		X	X		
mag() (ael)	X	X	X	X	X
max2() (ael)	X	X	X	X	X
member() (ael)	X	X	X	X	X
midstr() (ael)	X	X	X	X	X
min2() (ael)	X	X	X	X	X
mkdir() (ael)	X	X	X	X	X
nth() (ael)	X	X	X	X	X
nthcdr() (ael)	X	X	X	X	X
num() (ael)	X	X	X	X	X
num_args() (ael)	X	X	X	X	X
offset_array() (ael)	X	X	X	X	X
on_error() (ael)	X	X	X	X	X
parse() (ael)	X	X	X	X	X
parse_blank() (ael)		X			
pcb_get_form_value() (ael)		X			
pcb_get_mks() (ael)		X			
pcb_get_parm_type() (ael)		X			
pcb_get_string() (ael)		X			
pcb_set_form_value() (ael)		X			
pcb_set_mks() (ael)		X			
pcb_set_string() (ael)		X			
phase() (ael)	X	X	X	X	X
phasedeg() (ael)	X	X	X	X	X
phaserad() (ael)	X	X	X	X	X

polar() (ael)	X	X	X	X	X
pop_message_handler() (ael)	X	X	X	X	X
pow() (ael)	X	X	X	X	X
prm() (ael)		X			
push_message_handler() (ael)	X	X	X	X	X
rad() (ael)	X	X	X	X	X
re() (ael)	X	X	X	X	X
real() (ael)	X	X	X	X	X
remov() (ael)	X	X	X	X	X
remove() (ael)	X	X	X	X	X
rename() (ael)	X	X	X	X	X
rename_word() (ael)	X	X	X	X	X
repla() (ael)	X	X	X	X	X
resize_array() (ael)	X	X	X	X	X
rightstr() (ael)	X	X	X	X	X
round() (ael)	X	X	X	X	X
send_server_command() (ael)	X	X	X	X	X
send_server_data() (ael)	X	X	X	X	X
send_server_interrupt() (ael)	X	X	X	X	X
send_server_kill() (ael)	X	X	X	X	X
server_running() (ael)	X	X	X	X	X
set_design_choices() (ael)		X			
set_design_sub_choices() (ael)		X			
set_design_type() (ael)		X			
set_netlist_info() (ael)		X			
set_simulator_type() (ael)		X			
set_user_menu_label() (ael)					X
setenv() (ael)	X	X	X	X	X
sgn() (ael)	X	X	X	X	X
sin() (ael)	X	X	X	X	X
sinc() (ael)	X	X	X	X	X
sinh() (ael)	X	X	X	X	X
sleep() (ael)	X	X	X	X	X
sort_list() (ael)	X	X	X	X	X
sprintf() (ael)	X	X	X	X	X
sqrt() (ael)	X	X	X	X	X
start_server() (ael)	X	X	X	X	X
start_timer() (ael)	X	X	X	X	X
step() (ael)	X	X	X	X	X
strcasecmp() (ael)	X	X	X	X	X
strcat() (ael)	X	X	X	X	X
strcmp() (ael)	X	X	X	X	X
stripstr() (ael)	X	X	X	X	X
strlen() (ael)	X	X	X	X	X
system() (ael)	X	X	X	X	X
tan() (ael)	X	X	X	X	X
tanh() (ael)	X	X	X	X	X
tmpnam() (ael)	X	X	X	X	X

tolower() (ael)	X	X	X	X	X
total_elapsed_time() (ael)	X	X	X	X	X
toupper() (ael)	X	X	X	X	X
val() (ael)	X	X	X	X	X
validate_name() (ael)		X			
warning() (ael)	X	X	X	X	X
what_col() (ael)	X	X	X	X	X
what_file() (ael)	X	X	X	X	X
what_function() (ael)	X	X	X	X	X
what_line() (ael)	X	X	X	X	X
xor() (ael)	X	X	X	X	X

Obsolete Functions

Function	Attributes	Variables	Replacement/Availability
activate()			See *de_activate()* (ael).
add_layer()			See *db_create_layer()* (ael)
add_point()			See *de_add_point()* (ael).
add_vertex()			See *de_add_vertex()* (ael).
add_wire()			See *de_add_wire()* (ael).
analyze()			See *de_analyze()* (ael).
ang_factor()			See *de_ang_factor()* (ael).
arc()			See *de_add_arc()* (ael).
attach_project()			See *de_open_workspace()* (ael)
bell()			See *de_warning_bell()* (ael), *de_set_warning_bell()* (ael), *de_error_bell()* (ael), *de_set_error_bell()* (ael).
break_connection()			See *de_break_connection()* (ael).
change_annotation_layer()			See *de_change_annotation_layer()* (ael).
change_units()			Deleted
circle()			See *de_add_circle()* (ael).
clear_all()			See *de_close_all()* (ael)
clear_all_grids()			Deleted
clear_design()			Deleted.
clear_highlighting()			See *de_clear_highlighting()* (ael).
clear_rep()			See *db_clear_context()* (ael).
close_window()			See *de_close_window()* (ael).
config_window()			Deleted
connect()			See *de_connect()* (ael).
convert_to_polygon()			See *de_convert_to_polygon()* (ael).
convert_traces_to_instances()			See *de_convert_traces_to_instances()* (ael).
copy()			See *de_copy()* (ael).
copy_design()			See *de_copy_cellview()* (ael), *de_copy_cell()* (ael)
copy_project()			Deleted.
copy_to_buffer()			See *de_copy_to_buffer()* (ael).
copy_to_layer()			See *de_copy_to_layer()* (ael).
create_array_form()			Deleted
create_project()			Deleted.
create_window()			See *de_create_window()* (ael).
current_design_name()			Switch to a context; See *de_get_current_design_context()* (ael)
current_design_type()			See *de_current_design_type()* (ael).
data_dialog()			See *de_data_dialog()* (ael).
db_add_symbol_properties()			See *db_add_property()* (ael)

db_current_instance()			Deleted
db_find_instance()			See *db_find_instance_ex()* (ael)
db_find_property()			See *db_get_property()* (ael) or *db_get_property_as_string()* (ael)
db_first_dg()			See *db_create_shape_iter()* (ael) or *db_shape_iter_is_valid()* (ael) or *db_shape_iter_get_next()* (ael)
db_first_instance()			See *db_create_inst_iter()* (ael) or *db_inst_iter_is_valid()* (ael) or *db_inst_iter_get_next()* (ael)
db_first_mask()			See *db_create_shape_iter()* (ael) or *db_shape_iter_is_valid()* (ael) or *db_shape_iter_get_next()* (ael)
db_first_node()			See *db_create_net_iter()* (ael)
db_first_point()			See *db_get_shape_primitive_polygon()* (ael), *db_get_primitive_polygon_point()* (ael)
db_first_property()			See *db_create_prop_iter()* (ael), *db_prop_iter_is_valid()* (ael), *db_prop_iter_get_next()* (ael)
db_first_segment()			Deleted
db_free_points()			Deleted
db_get_arc_segment_attribute()			Deleted
	ARC_ANGLE		See *db_get_arc_angle()* (ael), *db_get_edge_arc_angle()* (ael)
	ARC_CENTER_POINT		See*db_get_arc_center()* (ael), *db_get_ellipse_center()* (ael), *db_get_edge_arc_center()* (ael)
	ARC_NUM_POINTS		Deleted
	ARC_START_POINT		See *db_get_arc_start()* (ael), *db_get_ellipse_x_radius()* (ael), *db_get_ellipse_y_radius()* (ael)
db_get_coord()			See *db_get_shape_primitive_polygon()* (ael), *db_get_primitive_polygon_point()* (ael)
db_get_current_design_rep()			See *de_get_current_design_context()* (ael)
db_get_design()			See *de_find_design_context_from_name()* (ael)
db_get_design_attribute()			Deleted
	DESIGN_COMMENT		Deleted
	DESIGN_MODIFIED		See *db_design_is_modified()* (ael)
	DESIGN_NAME		See *db_get_design_name()* (ael)
	DESIGN_PROPERTY		Deleted
	DESIGN_REP_LAY		Deleted
	DESIGN_REP_SCHEM		Deleted
	DESIGN_STAMP		Deleted
	DESIGN_TYPE		See *db_get_design_type()* (ael)
db_get_dg_attribute()			Deleted
	DG_BBOX		See *db_get_shape_bbox()* (ael)
	DG_DATA		Deleted
	DG_HIGHLIGHT		See *db_highlight()* (ael), *db_is_highlighted()* (ael)

	DG_NO_PLOT		Deleted
	DG_NUM_HOLES		Deleted
	DG_PNT_SELECTED		See *db_shape_has_selected_points()* (ael)
	DG_PROPERTY		See *db_create_prop_iter()* (ael)
	DG_SEG_SELECTED		Deleted
	DG_SELECT		See *db_is_selected()* (ael), *db_select()* (ael)
	DG_TYPE		Deleted
	ARC_DG_TYPE		See *db_shape_is_arc()* (ael)
	CIRCLE_DG_TYPE		See *db_shape_is_ellipse()* (ael)
	CONSTRUCTION_LINE_DG_TYPE		See *db_shape_is_const_line()* (ael)
	TEXT_DG_TYPE		See *db_shape_is_text_or_annotation()* (ael), *db_shape_is_text()* (ael), *db_shape_is_annotation()* (ael)
	WIRE_DG_TYPE		See *db_shape_is_wire_or_trace()* (ael), *db_shape_is_wire()* (ael), *db_shape_is_trace()* (ael)
	PATH_DG_TYPE		See *db_shape_is_path()* (ael)
	POLYGON_DG_TYPE		See *db_shape_is_polygon()* (ael)
	POLYLINE_DG_TYPE		See *db_shape_is_polyline()* (ael)
	RECTANGLE_DG_TYPE		See *db_shape_is_rectangle()* (ael)
		dlgH	(Global Variable) Deleted
db_get_instance_attribute ()			Deleted
	INST_ART_NAME		Deleted
	INST_ART_TYPE		Deleted
	INST_BBOX		See *db_get_instance_bbox()* (ael)
	INST_DEACTIVATE		See *db_is_instance_deactivated()* (ael)
	INST_DESIGN_NAME		See *db_get_instance_component_name()* (ael),*db_get_instance_design_name()* (ael)
	INST_DRC		Deleted
	INST_EQUIV_CREATED		Deleted
	INST_HIGHLIGHT		See *db_highlight()* (ael), *db_is_highlighted()* (ael)
	INST_ID		Deleted
	INST_MACRO_TYPE		Deleted
	INST_MASK_HEAD		Deleted
	INST_NAME		See *db_get_instance_name()* (ael)
	INST_NO_PLOT		Deleted
	INST_NULL_TYPE		Deleted
	INST_PARAM_HEAD		See *db_create_param_iter()* (ael)
	INST_PARAM_ROWS		Deleted
	INST_PIN_HEAD		See *db_create_inst_term_iter()* (ael), *db_create_inst_pin_iter()* (ael)
	INST_PROPERTY		Deleted
	INST_PSN_TYPE		Deleted
	INST_READIN		Deleted
	INST_REP_TYPE		Deleted

	INST_SELECT		See *db_is_selected()* (ael),*db_select()* (ael)
	INST_SPECIAL		See *db_get_instance_special()* (ael)
	INST_SYMB_TYPE		Deleted
	INST_SYMBOL_NAME		Deleted
	INST_TRANSFORM		See *db_get_instance_placement_transform()* (ael)
	INST_TRANSFORM_VALID		Obsolete, transforms are always valid in ADS 2011 and newer ADS versions.
	INST_TUNE		Deleted
	INST_TYPE		Deleted
	INST_VIEW		Deleted
	INST_VISITED		Deleted
db_get_mask_attribute()			Deleted
	MASK_DG		Deleted
	MASK_NUMBER		See *db_get_shape_layer()* (ael)
	MASK_PROPERTY		Deleted
db_get_node_attribute()			Deleted
	NODE_NAME		See *db_get_net_name()* (ael)
	NODE_NUMBER		See *db_get_net_name()* (ael)
	NODE_PIN_HEAD		See *db_create_term_iter()* (ael), *db_create_inst_term_iter()* (ael)
db_get_node_number()			See *db_get_net_name()* (ael)
db_get_path_attribute()			Deleted
	PATH_BEND		See *db_get_path_trace_bend_type()* (ael)
	PATH_MITERRADIUS		See *db_get_path_trace_miter_radius()* (ael)
	PATH_SEGMENT		See *db_get_shape_control_polygon()* (ael)
	PATH_WIDTH		See *db_get_path_trace_width()* (ael)
db_get_pin_attribute()			Deleted
	PIN_BINDING_LIST		Deleted
	PIN_DIRECTION		See *db_get_term_type()* (ael)
	PIN_INHERIT_BINDING		Deleted
	PIN_INST_PTR		See *db_get_inst_term_instance()* (ael)
	PIN_LOCATION		See *db_get_pin_snap_point()* (ael), *db_get_pin_angle_normalized()* (ael), *db_get_inst_pin_snap_point()* (ael), *db_get_inst_pin_angle_normalized()* (ael)
	PIN_MASK_NUMBER		See *db_get_pin_snap_layerid()* (ael)
	PIN_NAME		See *db_get_term_name()* (ael), *db_get_pin_term_name()* (ael)
	PIN_NODE_PTR		See *db_get_term_net()* (ael)
	PIN_NUMBER		See *db_get_term_number()* (ael), *db_get_inst_term_number()* (ael), *db_get_pin_term_number()* (ael), *db_get_inst_pin_term_number()* (ael)
	PIN_PROPERTY		See *db_create_prop_iter()* (ael)
	PIN_WIRE_PTR		Deleted

db_get_port_attribute()			Deleted
	PORT_LOCATION		See *db_get_pin_snap_point()* (ael), *db_get_pin_angle_normalized()* (ael)
	PORT_NAME		See *db_get_term_name()* (ael) or *db_get_pin_term_name()* (ael)
	PORT_NUMBER		See *db_get_term_number()* (ael), *db_get_inst_term_number()* (ael) or *db_get_pin_term_number()* (ael)
	PORT_POWER		Deleted
	PORT_PROPERTY		See *db_create_prop_iter()* (ael)
	PORT_SELECT		See *db_is_pin_selected()* (ael)
	PORT_SELECT_CHANGED		Deleted
db_get_port_number()			Deleted
db_get_property_attribute()			See *db_get_property()* (ael), *db_get_property_as_string()* (ael), *db_prop_iter_get_value()* (ael)
	PROPERTY_NAME		See *db_prop_iter_get_name()* (ael)
	PROPERTY_TYPE		See *db_prop_iter_get_name()* (ael)
db_get_rep()			Deleted
db_get_rep_attribute()			Deleted
	REP_BBOX		See *db_get_context_bbox()* (ael)
	REP_LAY_FILE		Deleted
	REP_OWNER		Deleted
	REP_PRF_FILE		Deleted
	REP_PROPERTY		See *db_create_prop_iter()* (ael)
	REP_SYMBOL		Deleted
	REP_TYPE		Deleted
db_get_rep_bbox()			See *db_get_context_bbox()* (ael)
db_get_rep_db_factor()			See *db_get_uu_to_dbu_factor()* (ael); alias: *db_get_context_db_factor()* (ael)
db_get_rep_unit_mks()			See *db_get_uu_to_mks_factor()* (ael)
db_get_rep_unit_name()			See *db_get_context_unit_name()* (ael)
db_get_rep_attribute()			Deleted
	REP_BBOX		See *db_get_context_bbox()* (ael)
	REP_LAY_FILE		Deleted
	REP_OWNER		Deleted
	REP_PRF_FILE		Deleted
	REP_PROPERTY		See *db_create_prop_iter()* (ael)
	REP_SYMBOL		Deleted
	REP_TYPE		Deleted
db_get_segment_attribute()			Deleted
	ARC_TYPE		See *db_get_primitive_polygon_edge_is_arc()* (ael)
	ORTHO_H_TYPE		See *db_get_primitive_polygon_edge_is_arc()* (ael)
	ORTHO_V_TYPE		See *db_get_primitive_polygon_edge_is_arc()* (ael)
	SEG_BBOX		See *db_get_shape_bbox()* (ael)

	SEG_INSIDE		See *db_get_shape_primitive_polygon()* (ael), *db_get_primitive_polygon_hole()* (ael)
	SEG_NUM_PNTS		See *db_get_shape_primitive_polygon()* (ael), *db_get_primitive_polygon_num_points()* (ael), *db_get_primitive_polygon_point()* (ael)
	SEG_PNT_LIST		See *db_get_shape_primitive_polygon()* (ael),*db_get_primitive_polygon_point()* (ael)
	SEG_PNT_SELECTED		See *db_shape_has_selected_points()* (ael)
	SEG_SELECT		Deleted
	SEG_TYPE		See *db_get_shape_primitive_polygon()* (ael), *db_get_primitive_polygon_edge_is_arc()* (ael)
db_get_symbol_attribute()			Deleted
	SYMB_BBOX		See *db_get_context_bbox()* (ael)
	SYMB_MASK_HEAD		See *db_create_shape_iter()* (ael)
	SYMB_PORT_HEAD		See *db_create_pin_iter()* (ael)
	SYMB_PROPERTY		See *db_create_prop_iter()* (ael)
db_get_text_attribute			Deleted
	ANNOT_DESIGN_NAME		See *db_is_cell_name_display()* (ael)
	ANNOT_INST_NAME		See *db_is_inst_name_display()* (ael)
	ANNOT_INST_PARAMETER		See *db_shape_is_annotation()* (ael), *db_is_cell_name_display()* (ael), *db_is_inst_name_display()* (ael)
	NORMAL_TEXT		See *db_shape_is_annotation()* (ael)
	TEXT_ABSOLUTE		See *db_get_text_absolute()* (ael)
	TEXT_FONT		See *db_get_text_font_name()* (ael)
	TEXT_FONT_NAME		See *db_get_text_font_name()* (ael)
	TEXT_HEIGHT		See *db_get_text_height()* (ael)
	TEXT_JUST		See *db_get_text_justification()* (ael)
	TEXT_LOCATION		See *db_get_text_angle()* (ael), *db_get_text_origin()* (ael)
	TEXT_PARAM_SEQ_NO		Deleted
	TEXT_STRING		See *db_get_shape_text_string()* (ael)
	TEXT_TYPE		See *db_shape_is_annotation()* (ael)
db_get_wire_attribute()			Deleted
	WIRE_BEND		See *db_get_path_trace_bend_type()* (ael)
	WIRE_MITERRADIUS		See *db_get_path_trace_miter_radius()* (ael)
	WIRE_SEGMENT		See *db_get_shape_control_polygon()* (ael)
	WIRE_WIDTH		See *db_get_path_trace_width()* (ael)
db_instance_next_pin			See *db_inst_term_iter_is_valid()* (ael), *db_inst_term_iter_get_next()* (ael)
db_next_dg			See *db_create_shape_iter()* (ael), *db_shape_iter_is_valid()* (ael), *db_shape_iter_get_next()* (ael)

db_next_instance			See *db_create_inst_iter()* (ael), *db_inst_iter_is_valid()* (ael), *db_inst_iter_get_next()* (ael)
db_next_mask			See *db_create_shape_iter()* (ael), *db_shape_iter_is_valid()* (ael), *db_shape_iter_get_next()* (ael)
db_next_node			See *db_net_iter_is_valid()* (ael), *db_net_iter_get_next()* (ael)
db_next_point			See*db_get_shape_primitive_polygon()* (ael),*db_get_primitive_polygon_point()* (ael)
db_next_port			See *db_pin_iter_is_valid()* (ael), *db_pin_iter_get_next()* (ael)
db_next_property			See *db_prop_iter_get_next()* (ael)
db_next_segment			Deleted
db_node_first_pin			See *db_create_term_iter()* (ael), *db_create_inst_term_iter()* (ael)
db_node_next_pin			See *db_term_iter_get_next()* (ael), *db_inst_term_iter_get_next()* (ael)
db_segment_to_points			Deleted
db_total_points			See *db_get_primitive_polygon_num_points()* (ael)
db_transform_bbox			See *db_transform_bbox_ex()* (ael)
db_transform_points			Deleted
de_add_layer			See *db_create_layer()* (ael)
de_archive_project			Deleted
de_attach_project()			Deleted
de_change_units			Deleted
de_clear_all_grids()			Deleted
de_clear_rep			See *db_clear_context()* (ael)
de_clear_show_connected			Deleted
de_close_design			Deleted
de_config_window			Deleted
de_copy_design			See *de_copy_cellview()* (ael), *de_copy_cell()* (ael)
de_copy_project			Deleted
de_create_project()			Deleted
de_current_design_name			See *de_get_current_design_context()* (ael)
de_delete_design			See *de_delete_cellview()* (ael)
de_delete_project			See *de_delete_workspace()* (ael)
de_delete_vertex()			See *de_delete()* (ael).
de_generate_symbol			See *de_generate_symbol_ex()* (ael)
de_get_current_design_rep_type			See *de_get_current_design_context()* (ael)
de_get_layer_attribute()			Deleted
	LAYER_BINDING_LIST		Deleted
	LAYER_COLOR		See *db_get_layerid_rgb()* (ael), *db_set_layerid_rgb()* (ael)
	LAYER_FILL		See *db_get_layerid_fill_pattern()* (ael), *db_set_layerid_fill_pattern()* (ael)

	LAYER_IGES_NUM		Deleted
	LAYER_LINE_TYPE		See *db_get_layerid_line_style()* (ael), *db_set_layerid_line_style()* (ael)
	LAYER_NAME		Deleted
	LAYER_NUMBER		See *db_get_layer_number()* (ael), *db_get_purpose_number()* (ael)
	LAYER_PLOT_MODE		See *db_get_layerid_fill_mode()* (ael), *db_set_layerid_fill_mode()* (ael)
	LAYER_PROTECTED		See *db_is_layerid_protected()* (ael), *db_set_layerid_protected()* (ael)
	LAYER_STREAM_NUM		Deleted
	LAYER_TYPE		See *db_get_layer_process_role()* (ael), *db_set_layer_process_role()* (ael)
	LAYER_VISIBLE		See *db_is_layerid_invisible()* (ael), *db_set_layerid_invisible()* (ael)
de_get_layer_names ()			See *db_get_layerid_names()* (ael)
de_update_parameters()			See *db_update_parameters_ex()* (ael)
de_get_variable_value()			See *db_evaluate_param_expression()* (ael)
de_highlight_instance			See *db_highlight_instance_ex()* (ael)
de_init_iteminfo()			See *de_init_item()* (ael).
de_init_tune()			See *de_tune()* (ael).
de_iteminfo_edit_instance()			See *de_edit_item()* (ael).
de_iteminfo_new_instance()			See *de_place_item()* (ael).
de_load_design()			See *de_get_design_context_from_name()* (ael)
de_load_grid()			See *de_open_datadisplay()* (ael).
de_load_item_artwork_image()			Deleted
de_merge_and()			See *de_intersection()* (ael)
de_merge_diff()			See *de_difference()* (ael)
de_merge_or()			See *de_union()* (ael)
de_move_vertex()			See *de_move()* (ael).
de_named_connection()			Deleted
de_netlist()			Deleted
de_new_design()			See *de_create_new_schematic_view()* (ael), *de_create_new_symbol_view()* (ael), *de_create_new_layout_view()* (ael).
de_new_project()			Deleted
de_new_text()			Deleted
de_open_check_rep_dialog()			Deleted
de_open_design()			See *de_get_design_context_from_name()* (ael)
de_open_grid()			See *de_new_datadisplay()* (ael).
de_open_project()			See *de_open_workspace()* (ael)
de_open_window()			See *de_show_context_in_new_window()* (ael)
de_optimization_continue()			Deleted
de_parameter_changed_callback()			Deleted
de_performance_optimization()			Deleted

de_place_orphan()			Deleted
de_query_iteminfo_attr()			Deleted
de_read_layer()			Deleted
de_redisplay()			Deleted
de_remove_prop()			See *de_remove_properties()* (ael).
de_remove_all_layers			Deleted
de_restore_all_grids			See *de_restore_all_datadisplay()* (ael).
de_save_design			See *db_save_design_without_prompting()* (ael)
de_save_grid()			Deleted
de_send_netlist()			Deleted
de_set_add_optional_parameters()			Deleted
de_set_coordinate_readout_mode			
de_set_instance_path_to_design()			Deleted
de_set_iteminfo_attr()			Deleted
de_set_named_connection()			Deleted
de_set_part_size()			See *de_set_port_size()* (ael).
de_set_route_dist()			Deleted
de_set_route_dist_units			Deleted
de_set_top_design_name()			Deleted
de_set_top_design_rep_type()			Deleted
de_set_trace_mcover_id()			Deleted
de_set_trace_msub_id()			Deleted
de_set_trace_mwall_id()			Deleted
de_set_trace_sub_id()			Deleted
de_set_trace_tand_id()			Deleted
de_set_trace_temp_id()			Deleted
de_set_window()			See *api_set_current_window()* (ael).
de_set_window_by_sequence()			See *api_set_current_window_by_seq_num()* (ael).
de_show_connected()			Deleted
de_show_design_in_window()			See *de_show_context_in_new_window()* (ael)
de_show_fixed()			Deleted
de_show_unplaced()			Deleted
de_sim_file_command()			Deleted
de_statistical_analysis()			Deleted
de_translate_design()			Deleted
de_tune_item()			See *de_tune()* (ael).
de_turn_off_trace_history()			Deleted
de_turn_on_trace_history()			Deleted
de_switch_view()			Deleted
de_unarchive_project()			Deleted
de_unhighlight_instances()			See *db_unhighlight_instances_ex()* (ael)
de_unmap_grids()			See *de_close_all_datadisplay()* (ael).
de_viewAll()			See *de_view_all()* (ael).
de_yield_optimization()			Deleted

de_variables()			See *de_set_hierarchy_root()* (ael), *de_set_hierarchy_instance_path()* (ael).
de_write_layer()			Deleted
deactivate()			See *de_deactivate()* (ael)
dedrc_run_drc()			See *dedrc_run_drc_ex()* (drc)
default_design_name()			Deleted
define_npport()			See *de_define_npport()* (ael).
define_port()			See *de_define_port()* (ael).
delete()			See *de_delete()* (ael).
delete_all_orphaned_instances()			Deleted
delete_design()			See *de_delete_cellview()* (ael)
delete_project()			See *de_delete_workspace()* (ael)
delete_vertex()			See *de_delete()* (ael).
demand_library_group()			Deleted
demand_palette_group()			Deleted
deselect_all()			See *de_deselect_all()* (ael).
deselect_all_force()			See *de_deselect_all()* (ael)
deselect_by_name()			See *de_deselect_by_name()* (ael).
deselect_item()			Deleted
deselect_window()			See *de_deselect_window()* (ael).
dm_num_parm_definition()			See *dm_num_parm_definitions()* (ael).
draw_arc()			See *de_draw_arc()* (ael).
draw_circ()			See *de_draw_circ()* (ael).
draw_point()			See *de_draw_point()* (ael).
draw_port()			See *de_draw_port()* (ael).
draw_rect()			See *de_draw_rect()* (ael).
draw_text()			See *de_draw_text()* (ael).
dse_l2s ()			See *de_dse_l2s()* (ael)
dse_place_orphan()			Deleted
dse_s2l()			See *de_dse_s2l()* (ael).
edit_annotation_attribute()			See *de_edit_annotation_attribute()* (ael).
edit_instance()			See *de_edit_item_ex()* (ael).
edit_item_parameters()			See *de_set_item_parameters()* (ael)
edit_path_trace()			See *de_edit_path_trace()* (ael).
edit_symbol_pin()			See *de_edit_symbol_pin()* (ael).
edit_text_attribute()			See *de_edit_text_attribute()* (ael).
edit_text_string()			See *de_edit_text_string()* (ael).
empty()			See *de_empty()* (ael).
end()			See *de_end()* (ael).
end_command()			See *de_end_command()* (ael).
fill()			See *de_fill()* (ael).
fix_instance()			See *de_fix_instances()* (ael).
flatten()			See *de_flatten()* (ael).
format_instance_data()			Deleted
free()			See *de_free_instances()* (ael)
get_data_parm()			See *de_get_data_parm()* (ael).
get_design_instances()			See *de_get_design_instances()* (ael).

get_design_list()			Deleted
get_file_names()			See *de_get_file_names()* (ael).
get_layer_attribute()			Deleted
get_parameter_names()			Deleted
get_push_history()			Deleted
get_preference()			See *de_get_preference()* (ael).
get_string_list()			Deleted
get_variable_names()			See *de_get_variable_names()* (ael).
get_window()			See *de_get_window()* (ael)
highlight_instance()			See *db_find_instance_ex()* (ael), *db_highlight()* (ael)
import_design()			See *de_import_design()* (ael).
info()			See *de_info()* (ael).
init_tune()			See *de_tune()* (ael)
instantiate()			See *de_instantiate()* (ael).
install_get_xy			See *de_install_get_xy()* (ael)
install_get_xy_pair			See *de_install_get_xy_pair()* (ael)
is_word()			Deleted
list_select			See *de_list_select()* (ael)
lock_project()			Deleted
ly_find_layer_by_gds_num			Deleted
mask_read()			Deleted
mask_write()			Deleted
merge_and()			See *de_intersection()* (ael)
merge_diff()			See *de_difference()* (ael)
merge_or()			See *de_union()* (ael)
mirror_x()			See *de_mirror_x()* (ael).
mirror_y()			See *de_mirror_y()* (ael).
miter_vertex()			See *de_miter_vertex()* (ael).
mks_factor()			Deleted
modify_break()			See *de_modify_break()* (ael).
modify_explode()			See *de_modify_explode()* (ael).
modify_join()			See *de_modify_join()* (ael).
move()			See *de_move()* (ael).
move_annotation()			See *de_move_annotation()* (ael).
move_break()			See *de_move_break()* (ael).
move_to_layer()			See *de_move_to_layer()* (ael).
move_vertex()			See *de_move()* (ael)
netlist()			Deleted
new_design()			See *de_create_new_schematic_view()* (ael), *de_create_new_symbol_view()* (ael), *de_create_new_layout_view()* (ael).
open_design()			See *de_get_design_context_from_name()* (ael)
open_grid()			See *de_new_datadisplay()* (ael).
open_window()			See *de_create_window()* (ael).
optimization_continue()			Deleted

oversize()			See *de_oversize()* (ael).
palette_group()			See *de_define_palette_group()* (ael).
pan_window()			See *de_pan_window()* (ael).
paste_from_buffer()			See *de_paste_from_buffer()* (ael).
path()			See *de_add_path()* (ael).
performance_optimization			Deleted
place_instance()			See *de_place_item()* (ael)
place_port()			See *de_place_port()* (ael).
place_unplaced()			See *de_place_unplaced()* (ael).
playback_macro()			See *de_playback_macro()* (ael).
plot()			See *de_plot()* (ael).
polygon()			See *de_add_polygon()* (ael).
polyline()			See *de_add_polyline()* (ael).
pop_outof_instance()			See *de_pop_outof_instance()* (ael).
pop_window()			Deleted
prompt()			See *de_prompt()* (ael).
push_into_instance()			See *de_push_into_instance()* (ael).
push_window()			Deleted
question()			See *de_question()* (ael).
read_layer()			Deleted
read_preference()			See *de_read_preferences()* (ael).
rectangle()			See *de_add_rectangle()* (ael).
redisplay()			See *de_refresh_view()* (ael).
reference_library_group()			Deleted
reference_palette_group()			Deleted
refresh_view()			See *de_refresh_view()* (ael).
remove_all_layers()			Deleted
restore_all_grids()			See *de_restore_all_datadisplay()* (ael).
restore_grids()			See *de_open_datadisplay()* (ael).
restore_status()			See *de_restore_status()* (ael).
rotate()			See *de_rotate()* (ael).
rotate_90()			See *de_rotate_90()* (ael).
rotate_image()			See *de_rotate_image()* (ael).
save_all()			See *de_save_all_designs()* (ael).
save_design()			See *db_save_design_without_prompting()* (ael)
save_grids()			Deleted
scale()			See *de_scale()* (ael).
screen_dump()			See *de_plot()* (ael)
search_and_replace()			See *de_search_and_replace()* (ael).
select_all()			See *de_select_all()* (ael).
select_all_force()			See *de_select_all_force()* (ael).
select_by_name()			See *de_select_by_name()* (ael).
select_item()			See *de_select_item()* (ael).
select_point()			Deleted
select_point_range()			See *de_select_range()* (ael).
select_range()			See *de_select_range()* (ael).

select_unplaced()			See *de_select_unplaced()* (ael).
select_window()			See *de_select_window()* (ael).
send_netlist()			Deleted
set_add_optional_parameters()			Deleted
set_annotation_font()			See *de_set_annotation_font()* (ael).
set_annotation_height()			See *de_set_annotation_height()* (ael).
set_annotation_id_layer()			See *de_set_annotation_id_layer()* (ael).
set_annotation_name_layer()			See *de_set_annotation_name_layer()* (ael).
set_annotation_parameters_layer()			See *de_set_annotation_parameters_layer()* (ael).
set_annotation_precision()			See *de_set_annotation_precision()* (ael).
set_annotation_rows()			See *de_set_annotation_rows()* (ael).
set_arc_radius()			See *de_set_arc_radius()* (ael).
set_auto_update_opt()			Deleted
set_background_color()			See *de_set_background_color()* (ael).
set_backup_count()			See *de_set_backup_count()* (ael).
set_check_self_intersection()			See *de_set_self_intersect()* (ael).
set_coordinate_readout_mode()			Deleted
set_curve_radius()			See *de_set_curve_radius()* (ael).
set_dse_start()			See *de_set_dse_start()* (ael).
set_dual_placement()			See *de_set_dual_placement()* (ael).
set_edit_text()			See *de_set_edit_text()* (ael).
set_entry_mode()			See *de_set_shape_entry_mode()* (ael).
set_foreground_color()			See *de_set_foreground_color()* (ael).
set_grid_color()			See *de_set_grid_color()* (ael).
set_grid_display_type()			See *de_set_grid_display_type()* (ael).
set_grid_snap()			See *de_set_grid_snap()* (ael).
set_grid_snap_type()			See *de_set_grid_snap_type()* (ael).
set_highlight_color()			See *de_set_highlight_color()* (ael).
set_instance()			See *de_init_item()* (ael).
set_instance_id()			See *de_set_item_id()* (ael)
set_instance_parameters()			See *de_set_item_parameters()* (ael)
set_layer()			See *de_set_layer()* (ael).
set_major_grid_display()			See *de_set_major_grid_display()* (ael).
set_minor_grid_display()			See *de_set_minor_grid_display()* (ael).
set_miter_cutoff()			See *de_set_miter_cutoff()* (ael).
set_num_pnts_for_arc()			Deleted
set_origin()			See *de_set_origin()* (ael).
set_oversize()			See de_set_oversize().
set_path_corner()			See *de_set_path_corner()* (ael).
set_path_width()			See *de_set_path_width()* (ael).
set_pin_color()			See *de_set_pin_color()* (ael).
set_pin_size()			See *de_set_pin_size()* (ael).
set_place_popup_mode()			See *de_set_place_popup_mode()* (ael).
set_plot_pin_mode()			See *de_set_plot_pins()* (ael).
set_plot_pin_names()			See *de_set_plot_pin_names()* (ael).

set_plot_pin_num_mode()			See *de_set_plot_pin_numbers()* (ael).
set_plotting_depth()			See *de_set_plotting_depth()* (ael).
set_port()			See *de_set_port()* (ael).
set_port_size_units()			See *de_set_port_size_units()* (ael)
set_reroute_wires()			See *de_set_reroute_wires()* (ael).
set_rotation_increment()			See *de_set_rotation_increment()* (ael).
set_scale()			See *de_set_scale()* (ael).
set_select_box_size()			See *de_set_select_box_size()* (ael).
set_select_color()			See *de_set_select_color()* (ael).
set_select_filter()			See *de_set_select_filter()* (ael).
set_select_inside_polygon()			See *de_set_select_inside_polygon()* (ael).
set_swap_template_instances()			See *de_set_swap_template_instance()* (ael).
set_tap_length()			See *de_set_tap_length()* (ael).
set_tee_color()			See *de_set_tee_color()* (ael).
set_tee_size()			See *de_set_tee_size()* (ael).
set_text_absolute()			See *de_set_text_absolute()* (ael).
set_text_angle()			See *de_set_text_angle()* (ael).
set_text_font()			See *de_set_text_font()* (ael).
set_text_height()			See *de_set_text_height()* (ael).
set_text_justification()			See *de_set_text_justification()* (ael).
set_text_string()			See *de_set_text_string()* (ael).
set_trace_mcover_id()			Deleted
set_trace_msub_id()			Deleted
set_trace_mwall_id()			Deleted
set_trace_sim_mode()			See *de_set_trace_sim_mode()* (ael).
set_trace_single_elem()			See *de_set_trace_single_elem()* (ael).
set_trace_tand_id()			Deleted
set_trace_tech()			See *de_set_trace_tech()* (ael).
set_trace_temp_id()			Deleted
set_trace_traverse()			See *de_set_trace_traverse()* (ael).
set_window()			See *api_set_current_window()* (ael).
shove()			See *de_shove()* (ael)
show_connected()			Deleted
show_equiv_inst()			See *de_show_equiv_inst()* (ael).
show_fixed()			Deleted
show_unmatched()			Deleted
show_unplaced()			Deleted
sim_file_command()			Deleted
snap()			See *de_snap()* (ael).
split_tlin()			See *de_split_tlin()* (ael).
statistical_analysis			Deleted
stretch()			See *de_stretch()* (ael).
stretch_tlin()			See *de_stretch_tlin()* (ael).
string_list()			Deleted
swap_instances()			See *de_swap_instances()* (ael).
switch_view()			Deleted

tap_tlin()			See *de_tap_tlin()* (ael).
text()			See *de_add_text()* (ael).
trace()			See *de_add_trace()* (ael).
translate_design()			Deleted
tune_item()			See *de_tune()* (ael).
undo()			See *de_undo()* (ael).
undo_vertex()			See *de_undo_vertex()* (ael).
unhighlight_instances()			See *db_unhighlight_instances_ex()* (ael)
unix_system()			Deleted
unmap_grids			See *de_close_all_datadisplay()* (ael).
update_optimization_values()			See *de_update_optimization_values()* (ael).
vertex_to_arc()			See *de_vertex_to_arc()* (ael).
view_all()			See *de_view_all()* (ael).
window_is_open()			See *de_window_is_open()* (ael).
		winInst	(Global) Deleted
		winInstP	(Global)Deleted
write_preference()			See *de_write_preferences()* (ael).
		x	(Global) Deleted
		y	(Global) Deleted
yield_optimization()			Deleted
zoom_in()			See *de_zoom_in_scale()* (ael).
zoom_out()			See *de_zoom_out_scale()* (ael).
zoom_window()			See *de_zoom_window()* (ael).

ADS Compatibility Examples

Design, Design Name, and Design Representation Examples

The use of design handles, design representation handles, and design names are being phased out for ADS 2011. ADS 2009 Update 1 introduces DesignContext's. These are the design data objects you should use going forward for AEL code in ADS 2011 and on. Design names are different in ADS 2011 then they were in ADS 2009 Update 1 or before, so switching design name dependent AEL code to instead use DesignContexts is a best practice going forward for ADS 2011 development. See Design Name Differences for more information.

Example 1 - Get design rep. from design name.

Note
Be careful with converting code that requires the use of a design name, since a design name in previous ADS releases compared to a design name in ADS 2011 are very different.

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
decl design_name = de_current_design_name();
decl designH = db_get_design(design_name);
decl design_rep = db_get_rep(designH,REP_SCHEM);
...
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...
decl context = de_get_current_design_context();
if (!de_is_schematic_context(context))
   return; // or error out if pre-condition is there to be a schematic design
...
```

Example 2 - Get design rep. from design name (Version 2).

Note: Be careful with converting code that requires the use of a design name, since a design name in earlier ADS version (ADS 2009 update 1 or earlier release) versus a design name in ADS 2011 are very different.

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
decl designH = db_get_design(design_name);
if (!designH)
    return;
decl design_rep = db_get_rep(designH,REP_SCHEM);
if (!design_rep)
    return;
...
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...
decl context1 = de_find_design_context_from_name(design_name);
if (context1 == NULL)
   return; // DesignContext could not be found.
if(!de_is_schematic_context(context1))
   return; // or error out if pre-condition is there to be a schematic design

// Also another Example:

decl context2 = de_get_design_context_from_name(design_name);
// Note: An AEL error will be returned if de_get_design_context() could not
//  retrieve a design context with the given design_name.
if (!de_is_schematic_context(context))
   return; // or error out if pre-condition is there to be a schematic design

...
```

Example 3 - Get design rep. from design in window.

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
decl designP = de_get_design_in_window(winInst);
if (designP == NULL)
   return(FALSE);
decl repType = de_query_window_rep_type(winInst);
decl repP = db_get_rep (designP, repType);
if (repP == NULL)
   return(FALSE);
...
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...
if (!de_window_has_valid_context(winInst))
  return FALSE;
decl context = de_get_design_context(winInst);
...
```

Example 4 - Get design name

Note
Be careful with converting code that requires the use of a design name, since a design name in ADS older version (ADS 2009 update 1 or earlier release) versus a design name in ADS 2011 are very different.

Be sure the AEL code really needs and requires to use a design name; in most cases the code can be modified to just use a design context in replace of a design name instead.

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
decl designName = current_design_name();
```

```
...
Converted code compatible with ADS 2011
// New, Converted, ADS 2011 Compatible Example
...
decl context = de_get_current_design_context();
decl designName = db_get_design_name(context);
...
```

Hierarchy Traversal Examples

Example 1 - Recursive Hierarchy traversal

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
defun my_traverse_design(repType, designName, instName)
{

  fputs( stderr, fmt_tokens( list( "Traversing", designName, instName ) ) );

  decl designHandle = db_get_design(designName);
  if (designHandle)
  {
      decl repHandle = db_get_rep(designHandle, repType);
      if (repHandle)
      {
          decl instHandle = db_first_instance(repHandle);
          while (instHandle)
          {
              designName = db_get_instance_attribute(instHandle, INST_DESIGN_NAME);
              instName = db_get_instance_attribute(instHandle, INST_NAME);

              if (designName)
                  my_traverse_design(repType, designName, instName);

              instHandle = db_next_instance(instHandle);
          }
      }
  }
}
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...
defun my_traverse_design(hierContext, compName, instName)
{
  fputs( stderr, fmt_tokens( list( "Traversing", compName, instName ) ) );

  if (hierContext != NULL)
  {
      decl iter = db_create_inst_iter(hierContext);
      for ( ; db_inst_iter_is_valid(iter); iter = db_inst_iter_get_next(iter))
      {
           decl isPrimitive =
               db_is_primitive_instance_in_hierarchy(hierContext, iter);
           decl subHierContext;
           if (!isPrimitive)
               subHierContext = db_get_hierarchy_context_for_instance(hierContext, iter);

           compName = db_get_instance_component_name(iter);
```

```
            instName = db_get_instance_name(iter);

            my_traverse_design(subHierContext, compName, instName);
        }
    }
}
```

Instance Examples

Example 1 - Traversing instances of a design to get first selected instance.

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
decl design_name=de_current_design_name();
decl design_rep=db_get_rep(db_get_design(design_name),REP_SCHEM);

// Get first selected instance.
decl insthandle=db_first_instance(design_rep);
while (insthandle&&(db_get_instance_attribute(insthandle,INST_SELECT )==0))
{
  insthandle=db_next_instance(insthandle);
}
if (!insthandle)
   return FALSE;  // No selected instances available.
...
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...
decl context = de_get_current_design_context();
if (!de_is_schematic_context(context))
   return FALSE;

decl instIter = db_create_inst_iter(context);
// Limit instance iterator to only selected instances.
instIter = db_inst_iter_limit_selected(instIter);
if (!db_inst_iter_is_valid(instIter))
   return FALSE;  // No selected instances available.

// Get first selected instance.
decl insthandle = db_inst_iter_get_instance(institer);
...
```

Example 2 - Traversing all instances of a design to get instance names.

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
decl designP = de_get_design_in_window(winInst);
if (designP == NULL)
  return NULL;
decl repType = de_query_window_rep_type(winInst);
```

```
decl design_rep = db_get_rep(designP, repType);

// Get list of instance names.
decl instNameList = list();
decl insthandle=db_first_instance(design_rep);
while (insthandle)
{
  decl instName = db_get_instance_attribute(insthandle,INST_NAME );
  instNameList = append(instNameList, list(instName));

  insthandle=db_next_instance(insthandle);
}
return instNameList;
...
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...
if (!de_window_has_valid_context(winInst))
    return NULL;
decl context = de_get_design_context(winInst);

// Get List of instance names.
decl instNameList = list();
decl instIter = db_create_inst_iter(context);
for( ; db_inst_iter_is_valid(instIter);
    instIter = db_inst_iter_get_next(instIter))
{
   // Add instance name to list.
   decl instName = db_get_instance_name(instIter);
   instNameList = append(instNameList, list(instNameList));
}
return instNameList;
...
```

Example 3 - Query/Modify attributes of an instance.

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
decl instName = db_get_instance_attribute(insthandle,INST_NAME);
decl isSelected = db_get_instance_attribute(insthandle,INST_SELECT);
decl isDeactivated = db_get_instance_attribute(insthandle,INST_DEACTIVATE);
decl isShorted = db_get_instance_attribute(insthandle,INST_SHORTED);
...
db_set_instance_attribute (insthandle, INST_SELECT, FALSE);
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...
decl instName = db_get_instance_name(insthandle);
decl isSelected = db_is_selected(insthandle);
decl isDeactivated = db_is_instance_deactivated(insthandle);
decl isShorted = db_is_instance_deactivated_and_shorted(insthandle);
...
db_select(insthandle, FALSE);
...
```

Pin and Port Examples

Example 1: Pin Traversal to get node number information

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)

  ...
  // Given an instance handle, return a list of sort node connection numbers

  decl pinHandle, pinNo, pinData, pinList;
  decl nodeHandle, nodeNo, nodeList;
  nodeList = NULL;
  pinData  = NULL;

  if( instHandle )
  {
      pinHandle = db_get_instance_attribute( instHandle, INST_PIN_HEAD );
      while( pinHandle )
      {
          pinNo = db_get_pin_attribute( pinHandle, PIN_NUMBER );
          nodeHandle = db_get_pin_attribute( pinHandle, PIN_NODE_PTR );
          if( nodeHandle )
          {
              nodeNo = db_get_node_number( nodeHandle );
              if( is_list( pinData ) )
                  pinData = append( pinData, list(list(pinNo, nodeNo) ) );
              else pinData = list( list( pinNo, nodeNo ) );
          }
          pinHandle = db_instance_next_pin( pinHandle );
      }

      /* sort pins by pin number */
      if( is_list( pinData ) )
          pinData = sort_list( pinData );

      /* pull off node numbers in sorted order */
      nodeList = NULL;
      while( pinData )
      {
          pinList = car( pinData );
          nodeNo = car( cdr( pinList ) );
          if( is_list( nodeList ) )
              nodeList = append( nodeList, list(nodeNo) );
          else
              nodeList = list(nodeNo);
          pinData = cdr( pinData );
      }
  }
  return( nodeList );
  ...
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example

  ...
  // Given an instance handle, return a list of sort node connection numbers

  if (instHandle == NULL)
       return NULL;
```

```
  decl pinData = NULL;
  decl instTermIter = db_create_inst_term_iter(instHandle);
  for (; db_inst_term_iter_is_valid(instTermIter);
       instTermIter = db_inst_term_iter_get_next(instTermIter))
  {
      decl pinNo = db_get_inst_term_number(instTermIter);
      decl net = db_get_inst_term_net(instTermIter);
      if (net == NULL)
          continue;   // bug?
      decl netName = db_get_net_name(net);
      if ( is_list( pinData ) )
          pinData = append( pinData, list(list(pinNo, netName)) );
      else
          pinData = list(list(pinNo, netName));
  }

  /* sort pins by pin number */
  if ( is_list( pinData ) )
      pinData = sort_list( pinData );

  /* pull off node numbers in sorted order */
  decl nodeList = NULL;
  for (; pinData; pinData = cdr(pinData))
  {
      decl pinList = car( pinData );
      decl netName = car( cdr( pinList ) );
      if ( is_list( nodeList ) )
          nodeList = append( nodeList, list(netName) );
      else
          nodeList = list(netName);
  }
  return( nodeList );
  ...
```

Example 2: Node Traversal to get node pin information for netlisting.

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)

  ...
  // Given a representation, traverse the rep's nodes.
  decl nodeHandle = db_first_node(repHandle);
  while(nodeHandle)
  {
      // Traverse the node's pins.
      decl pinHandle = db_get_node_attribute(nodeHandle, NODE_PIN_HEAD);
      while(pinHandle)
      {
          pinHandle = db_node_next_pin(pinHandle);
          ...
          // Do something with pin.
          ...
      }
      nodeHandle = db_next_node(nodeHandle);
  }
  ...
```

New, converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
  ...
  // First convert your code to use design context's instead of representations.
  //  See the examples of using DesignContexts in replace of reps. etc...
  ...

  // Given a DesignContext, traverse the context's Nets.
  decl netIter = db_create_net_iter(designContext);
  for (; db_net_iter_is_valid(netIter);
      netIter = db_net_iter_get_next(netIter))
  {
      // Traverse the Net's Terms.
      decl termIter = db_create_term_iter(netIter);
      for (; db_term_iter_is_valid(termIter);
          termIter = db_term_iter_get_next(termIter))
      {
          decl termHandle = db_term_iter_get_term(termIter);
          ...
          // Do something with terminal.
          ...
      }
  }
  ...
```

Example 3: Get the number of Symbol Pins...

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)

  ...
  // Return number of pins on the symbol
  decl designP, repP, symbolP, portP, pinCount = 0;

  if ((designP = db_get_design(dsnName)) == NULL)
      return pinCount;

  if ((repP = db_get_rep (designP, REP_SCHEM)) == NULL)
      return pinCount;

  if ((symbolP = db_get_rep_attribute (repP, REP_SYMBOL)) == NULL)
      return pinCount;

  if ((portP = db_get_symbol_attribute (symbolP, SYMB_PORT_HEAD))==NULL)
      return pinCount;

  for (; portP!=NULL; portP = db_next_port(portP), pinCount++) {}

  return pinCount;
```

New, Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
  ...
  // First convert your code to use design context's instead of representations.
  //  See the examples of using DesignContexts in replace of reps. etc...
  ...

  decl pinCount = 0;
  if (!de_is_symbol_context(context))
     return pinCount;

  decl pinIter = db_create_pin_iter(context); // Get pins on symbol context.
```

```
  for ( ; db_pin_iter_is_valid(pinIter); pinIter = db_pin_iter_get_next(pinIter))
      pinCount++;

  return pinCount;
```

Example 4: Get list of selected ports

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
// Get the selected ports in the current layout.
decl portList = NULL;

decl designH = de_get_current_design();
if ( designH != NULL )
{
  //--- get the layout representation -------------------------------
  decl repH = db_get_rep(designH, REP_LAY);
  if (repH != NULL)
  {
    //--- loop over all instances in layout -------------------------
    decl instH = db_first_instance(repH);
    while ( instH != NULL )
    {
      if (db_get_instance_attribute(instH, INST_SPECIAL) & INST_PORT)
      {
        //--- instance is port ---------------------------------------
        decl select = db_get_instance_attribute(instH, INST_SELECT);
        if ( select == TRUE )
        {
          //--- port is selected ---------------------------------------
          decl portName = db_get_instance_attribute(instH, INST_NAME);
          portList = append(portList, list(portName));
        }
      }
      instH = db_next_instance(instH);
    }
  }
}

//de_info(sprintf("selected portList = %s", identify_value(portList)));
return portList;
}
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...

decl portList = NULL;
if (!de_current_context_is_valid())
   return NULL;

decl context = de_get_current_design_context();
if (de_is_layout_context(context))
{
    decl pinIter = db_create_pin_iter(context);

    for ( ; db_pin_iter_is_valid(pinIter); pinIter = db_pin_iter_get_next(pinIter))
        if (db_is_pin_selected(pinIter))
        {
            decl pin = db_pin_iter_get_pin(pinIter);
```

```
            portList = append(portList, list(pin));
        }
}
return portList;
```

Example 5: Select ports in a given range.

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
// Select ports in a range in the current layout.

// Get range from input coordinates.
decl minX = (coordX1 < coordX2)? coordX1 : coordX2;
decl minY = (coordY1 < coordY2)? coordY1 : coordY2;
decl maxX = (coordX1 > coordX2)? coordX1 : coordX2;
decl maxY = (coordY1 > coordY2)? coordY1 : coordY2;

decl designH = de_get_current_design();
if ( designH != NULL )
{
  //--- get the layout representation ------------------------------
  decl repH = db_get_rep(designH, REP_LAY);
  if (repH != NULL)
  {
    //--- get the coordinate transform factors ----------------------
    decl uu2db = db_get_rep_db_factor(repH);
    decl db2uu = 1.0/uu2db;

    //--- loop over all instances in layout -------------------------
    decl instH = db_first_instance(repH);
    while ( instH != NULL )
    {
      if (db_get_instance_attribute(instH, INST_SPECIAL) & INST_PORT)
      {
         //--- instance is port -----------------------------------
         decl portName = db_get_instance_attribute(instIter, INST_NAME);

         //--- get port location ----------------------------------
         decl pinH = db_get_instance_attribute(instIter, INST_PIN_HEAD);
         decl locationH = db_get_pin_attribute(pinH, PIN_LOCATION);
         decl coordXp = db2uu * db_get_location_x(locationH);
         decl coordYp = db2uu * db_get_location_y(locationH);

         //--- check if port location is in range -------------------
         decl portInRange = ( (minX <= coordXp) && (coordXp <= maxX) &&
                              (minY <= coordYp) && (coordYp <= maxY) )? 1 : 0;
         if ( portInRange )
         {
            //--- select port ---------------------------------------
            db_set_instance_attribute(instH, INST_SELECT, TRUE);
         }

      }
      instH = db_next_instance(instH);
    }
    de_refresh_view();
  }
}
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
  ...
  // Select ports in a range in the current layout.

  if (!de_current_context_is_valid())
      return NULL;

  decl context = de_get_current_design_context();
  if (de_is_layout_context(context))
  {
      //--- get the coordinate transform factors ----------------------
      decl uu2db = db_get_context_db_factor(context);
      decl db2uu = 1.0/uu2db;

      // Get range from input coordinates.
      //--- get min and max of range ----------------------------------
      decl minX = (coordX1 < coordX2)? coordX1 : coordX2;
      decl minY = (coordY1 < coordY2)? coordY1 : coordY2;
      decl maxX = (coordX1 > coordX2)? coordX1 : coordX2;
      decl maxY = (coordY1 > coordY2)? coordY1 : coordY2;

      decl pinIter = db_create_pin_iter(context);
      for ( ; db_pin_iter_is_valid(pinIter); pinIter = db_pin_iter_get_next(pinIter))
      {
           decl coordH = db_get_pin_snap_point(pinIter);
           decl coordXp = db2uu * db_get_x(coordH);
           decl coordYp = db2uu * db_get_y(coordH);

           //--- check if port location is in range --------------------
           decl portInRange = ( (minX <= coordXp) && (coordXp <= maxX) &&
                                (minY <= coordYp) && (coordYp <= maxY) )? 1 : 0;
           if ( portInRange )
           {
               db_select_pin(pinIter);
           }
       }
       de_refresh_view();
  }
```

Example 6: Get Port Information from Design, such as Number, Name, etc.

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
decl portList=list();
decl design, rep, inst;
design = de_get_current_design();
rep = db_get_rep(design, REP_SCHEM);
for(inst = db_first_instance(rep);  inst; inst=db_next_instance(inst))
{
    if((db_get_instance_attribute(inst, INST_SPECIAL) & INST_PORT))
    {
        decl portName = db_get_instance_attribute(inst, INST_NAME);
        decl portNo = db_get_port_number(inst);
        decl portDir;

        decl portInfo = list(portName, portNo, portDir);
        portList = append( portList, list(portInfo));
   }
}
return(portList);
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...
decl portList=list();
decl designContext = de_get_current_design_context();
if (de_is_schematic_context(designContext))
{
    decl pinIter = db_create_pin_iter(designContext);
    for( ; db_pin_iter_is_valid(pinIter); pinIter = db_pin_iter_get_next(pinIter))
    {
        decl portName = db_get_pin_name(pinIter);
        decl portNo = db_get_pin_term_number(pinIter);
        decl portDir = db_get_pin_term_type(pinIter);

        decl portInfo=list(portName, portNo, portDir);
        portList=append( portList, list(portInfo));
    }
}
return(portList);
```

Example 7: Get which instances are connected to a particular port(pin)

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
decl design, rep, inst;
design = de_get_current_design();
rep = db_get_rep(design, REP_SCHEM);
decl portName = "";
for(inst = db_first_instance(rep); inst; inst=db_next_instance(inst))
{
  if((db_get_instance_attribute(inst, INST_SPECIAL) & INST_PORT))
  {
    decl portInst = inst;  // Port instance found.
    decl portName = db_get_instance_attribute(inst, INST_NAME);

    // From Port instance, get all other instances the port is connected to.
    decl instNameList = list();
    decl componentNameList = list();

    decl portInstPinH = db_get_instance_attribute(portInst, INST_PIN_HEAD);
    decl portInstPinNodePtr = db_get_pin_attribute(portInstPinH, PIN_NODE_PTR);
    decl portInstPinNodePinH = NULL;
    for (portInstPinNodePinH = db_get_node_attribute(portInstPinNodePtr,NODE_PIN_HEAD);
        portInstPinNodePinH;
        portInstPinNodePinH = db_get_pin_attribute(portInstPinNodePinH,PIN_NEXTNODEPIN))
    {
      decl checkInst = db_get_pin_attribute(portInstPinNodePinH, PIN_INST_PTR);
      // Get only non port instances connected to the ports.
      if ( !(db_get_instance_attribute(checkInst, INST_SPECIAL) & INST_PORT))
      {
        decl instName = db_get_instance_attribute(checkInst, INST_NAME);
        if (!member(instName, instNameList))
            instNameList = append(instNameList, list(instName));
        decl compName = db_get_instance_attribute(checkInst, INST_DESIGN_NAME);
        if (!member(compName, componentNameList))
            componentNameList = append(componentNameList, list(compName));
      }
    }
```

```
    // Instance names the port is connected to are in instNameList;
    decl instReportStr = strcat(" Port Name (", portName , ") has instances: ");
    decl idx=0;
    for (idx=0; idx < listlen(instNameList); idx++)
    {
       instReportStr = strcat(instReportStr, nth(idx, instNameList));
       if ((idx + 1) < listlen(instNameList))
         instReportStr = strcat(instReportStr, ", ");
    }
    instReportStr = strcat(instReportStr, "\n");
    fputs(stderr, instReportStr);

    // Component names the port is connected to are in componentNameList.
    decl compReportStr = strcat(" Port Name (", portName , ") has components: ");
    for (idx=0; idx < listlen(componentNameList); idx++)
    {
      compReportStr = strcat(compReportStr, nth(idx, componentNameList));
      if ((idx + 1) < listlen(componentNameList))
        compReportStr = strcat(compReportStr, ", ");
    }
    compReportStr = strcat(compReportStr, "\n");
    fputs(stderr, compReportStr);
  }
}
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...
decl designContext = de_get_current_design_context();

decl termIter = db_create_term_iter(designContext);
for ( ; db_term_iter_is_valid(termIter);
    termIter = db_term_iter_get_next(termIter))
{
    decl termName = db_get_term_name(termIter);
    decl termNet = db_get_term_net(termIter);
    decl instTermIter = db_create_inst_term_iter(termNet);
    decl instNameList = list();
    decl componentNameList = list();
    for ( ; db_inst_term_iter_is_valid(instTermIter);
        instTermIter = db_inst_term_iter_get_next(instTermIter))
    {
        decl inst = db_get_inst_term_instance(instTermIter);
        decl instName = db_get_instance_name(inst);
        if (!member(instName, instNameList))
            instNameList = append(instNameList, list(instName));
        decl compName = db_get_instance_component_name(inst);
        if (!member(compName, componentNameList))
            componentNameList = append(componentNameList, list(compName));
    }

    // Instance names the term is connected to are in instNameList;
    decl instReportStr = strcat(" Term Name (", termName , ") has instances: ");
    decl idx=0;
    for (idx=0; idx < listlen(instNameList); idx++)
    {
      instReportStr = strcat(instReportStr, nth(idx, instNameList));
      if ((idx + 1) < listlen(instNameList))
        instReportStr = strcat(instReportStr, ", ");
    }
    instReportStr = strcat(instReportStr, "\n");
    fputs(stderr, instReportStr);
```

```
    // Component names the term is connected to are in componentNameList.
    decl compReportStr = strcat(" Term Name (", termName , ") has components: ");
    for (idx=0; idx < listlen(componentNameList); idx++)
    {
      compReportStr = strcat(compReportStr, nth(idx, componentNameList));
      if ((idx + 1) < listlen(componentNameList))
        compReportStr = strcat(compReportStr, ", ");
    }
    compReportStr = strcat(compReportStr, "\n");
    fputs(stderr, compReportStr);
}
```

Shape Examples

Example 1: Traverse shapes and find the text objects.

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...

decl maskP = NULL;
decl dsn = de_get_design_in_window(winInst);
if (dsn != NULL)
{
    decl rep = db_get_rep(dsn, REP_SCHEM);
    if (rep != NULL)
        maskP = db_first_mask(rep);
}

decl textDgList = list();
while (maskP)
{
    decl dgP = db_first_dg(maskP);
    while (dgP)
    {
        decl dgtype = db_get_dg_attribute(dgP, DG_TYPE);

        if (dgtype == TEXT_DG_TYPE)
        {
            textDgList = append(textDgList, list(dgP));
        }
        dgP = db_next_dg(dgP);
    }
    maskP = db_next_mask(maskP);
}

return textDgList;
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...
if (!de_window_has_valid_context(context))
    return NULL;
decl textShapeList = list();
decl context = de_get_design_context(winInst);
decl shapeIter = db_create_shape_iter(context);
while (db_shape_iter_is_valid(shapeIter))
{
    decl shape = db_shape_iter_get_shape(shapeIter);
```

```
    if (db_shape_is_text(shape))
    {
        textShapeList = append(textShapeList, list(shape));
    }
    shapeIter = db_shape_iter_get_next(shapeIter);
}
return textShapeList;
```

Example 2: Get list of selected shapes

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
   ...
   decl selectedShapes = list();
   decl designH = de_get_design_in_window (winInst);
   decl repH = db_get_rep (designH, REP_LAY);
   if (repH)
   {
       decl maskListH = db_first_mask (repH);
       while (maskListH)
       {
           decl dgListH = db_get_mask_attribute (maskListH, MASK_DG);
           while (dgListH)
           {
               decl dgType = db_get_dg_attribute (dgListH, DG_TYPE);
               if (dgType == ARC_DG_TYPE       ||
                   dgType == CIRCLE_DG_TYPE    ||
                   dgType == PATH_DG_TYPE      ||
                   dgType == POLYLINE_DG_TYPE  ||
                   dgType == POLYGON_DG_TYPE   ||
                   dgType == RECTANGLE_DG_TYPE )
               {
                   decl dgSelect = db_get_dg_attribute (dgListH, DG_SELECT);
                   if (dgSelect)
                   {
                       selectedShapes = append (selectedShapes, list(dgListH));
                   }
               }
               dgListH = db_next_dg (dgListH);
           }
           maskListH = db_next_mask (maskListH);
       }
   }
   return selectedShapes;
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...
decl selectedShapes = list();
decl context = de_get_design_context(winInst);
decl iter = db_create_shape_iter(context);
iter = db_shape_iter_limit_selected(iter);
for ( ; db_shape_iter_is_valid(iter); iter = db_shape_iter_get_next(iter))
{
    if (db_shape_is_arc(iter)     ||
        db_shape_is_ellipse(iter) ||
        db_shape_is_path(iter)    ||
        db_shape_is_polyline(iter)||
        db_shape_is_polygon(iter) ||
        db_shape_is_rectangle(iter)  )
```

```
    {
        selectedShapes = append(selectedShapes, list(db_shape_iter_get_shape(iter)));
    }
}
return selectedShapes;
```

Form Examples

Example 1: Querying Form Definition Attributes

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
decl formName = db_get_parm_attribute(paramHandle, PARM_FORM_NAME);
decl formDefP = dm_find_form_definition(formName);
decl formParms = dm_get_form_definition_attribute(formDefP, DM_FORM_PARMS);
decl formLabel = dm_get_form_definition_attribute(formDefP, DM_FORM_LABEL);
decl formClass = dm_get_form_definition_attribute(formDefP, DM_FORM_CLASS);
decl formDspStr = dm_get_form_definition_attribute(formDefP, DM_FORM_DISP_FORMAT);
formName = dm_get_form_definition_attribute(formDefP, DM_FORM_NAME);

if (dm_get_form_definition_attribute(formDefP, DM_FORM_ATTR) & FORM_DISCRETE))
{
   // Discrete parameter value.
   ...
}
...
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...
decl formDefP = db_param_iter_find_form(paramIter);
if (!formDefP)
   return;
decl formName = dm_form_get_name(formDefP);
decl formParms = dm_form_get_parms(formDefP);
decl formLabel = dm_form_get_label(formDefP);
decl formClass = dm_form_get_class(formDefP);
decl formDspStr = dm_form_get_disp_format(formDefP);

if (dm_form_is_discrete(formDefP))
{
   // Discrete parameter value.
   ...
}
...
```

Item Definition Examples

Example 1: Querying Item Definition Attributes

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
decl attr = dm_get_item_definition_attribute(itemDefP, ITEM_ATTR);
```

```
decl attrEx = dm_get_item_definition_attribute (itemDefP, ITEM_ATTR_EX);
decl dlgName = dm_get_item_definition_attribute (itemDefP, ITEM_DIALOG_NAME);
decl label = dm_get_item_definition_attribute (itemDefP, ITEM_LABEL);
decl artType = dm_get_item_definition_attribute(itemDefP, ITEM_ART_TYPE);
decl artData = dm_get_item_definition_attribute(itemDefP, ITEM_ART_DATA);
decl itemName = dm_get_item_definition_attribute(itemDefP, ITEM_NAME);
decl itemParmDefList = dm_get_item_definition_attribute(itemDefP, ITEM_PARMS);
decl icon = dm_get_item_definition_attribute (itemDefP, ITEM_ICON);
...
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...
decl attr = dm_item_get_attr(itemDefP);
decl attrEx = dm_item_get_ex_attr(itemDefP);
decl dlgName = dm_item_get_dialog_name(itemDefP);
decl label = dm_item_get_label(itemDefP);
decl artType = dm_item_get_artwork_type(itemDefP);
decl artData = dm_item_get_artwork_data(itemDefP);
decl itemName = dm_item_get_name(itemDefP);
decl itemParmDefList = dm_item_get_parms(itemDefP);
decl icon = dm_item_get_icon_name(itemDefP);
...
```

Parameter Examples

Example 1: Parameter names from an instance's parameters

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...

decl paramH = db_get_instance_attribute(instH, INST_PARAM_HEAD);
while (paramH)
{
    decl parmName = db_get_parm_attribute(paramH, PARM_NAME);
    // Do some work with the parameter name.
    ...
    paramH = db_next_parm(paramH);
}
...
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...

decl paramIter = db_create_param_iter(instH);
for ( ; db_param_iter_is_valid(paramIter);
    paramIter = db_param_iter_get_next(paramIter) )
{
    decl parmName = db_get_param_name(paramIter);
    // Do some work with the parameter name.
    ...
}
...
```

Example 2: Querying Parameter Value Attributes

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
decl parmSval = db_get_parm_attribute( paramH, PARM_VALUE_SVALUE );
decl formName = db_get_parm_attribute( paramH, PARM_FORM_NAME );
decl parmName = db_get_parm_attribute( paramH, PARM_NAME );
decl parmType = db_get_parm_attribute( paramH, PARM_VALUE_CODE );
decl subparmValList = db_get_parm_attribute( paramH, PARM_VALUE_LIST );
while (subparmValList)
{
   // Visit each sub-parameter
   ...
   subparmValList = db_next_parm(subParmValList);
}

...
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...
decl paramH = db_param_iter_get_param( paramIter );
decl parmSval = db_get_param_string_value( paramH );
decl formName = db_get_param_form_name( paramH );
decl parmName = db_get_param_name( paramH );
decl parmType = db_get_param_value_code( paramH );
decl subParamIter = db_param_iter_get_sub_parameters( paramIter );
for ( ; db_param_iter_is_valid(subParamIter);
   subParamIter = db_param_iter_get_next(subParamIter);
{
   decl subParamVal = db_param_iter_get_param(subParamIter);
   // Visit each sub-parameter
   ...
}
...
```

Example 3: Querying Parameter Definition Attributes

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
decl parmLabel = dm_get_parm_definition_attribute(parmDefP, DM_PARM_LABEL);
decl parmName = dm_get_parm_definition_attribute(parmDefP, PARM_NAME);
decl unitType = dm_get_parm_definition_attribute(parmDefP, DM_PARM_UNIT);
decl parmAttr = dm_get_parm_definition_attribute (parmDefP, DM_PARM_ATTR);
decl defaultParamValP = dm_get_parm_definition_attribute(parmDefP, DM_PARM_DEFVALUE);
...
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...
decl parmLabel = dm_parm_get_label(parmDefP);
decl parmName = dm_parm_get_name(parmDefP);
```

```
decl unitType = dm_parm_get_unit(parmDefP);
decl parmAttr = dm_parm_get_attr(parmDefP);
decl defaultParamValP = dm_parm_get_defvalue(parmDefP);
...
```

Example 4: Find a parameter on an instance by parameter index and by parameter name

Old ADS Code (till ADS 2009 Update 1)

```
// Original ADS Example (ADS 2009 update 1 or earlier release)
...
decl inst = db_find_instance(current_design_name(), 0, "TL1");
if (!inst)
   return;
// Find instance parameter's nominal value by index
decl value1 = db_get_instance_parm(current_design_name(), inst, 3);
// Find instance parameter's nominal value by name
decl value2 = db_get_instance_parm(current_design_name(), inst, "W");
```

Converted code compatible with ADS 2011

```
// New, Converted, ADS 2011 Compatible Example
...
decl inst = db_find_instance_ex(de_get_current_design_context(), "TL1");
if (!inst)
   return;

decl paramIter = db_create_param_iter(inst);

// Find instance parameter's display value by index
decl foundParamIter1 = db_param_iter_find_by_index(paramIter, 3);
decl value1 = "";
if (db_param_iter_is_valid(foundParamIter1))
   value1 = db_param_iter_get_display_value(foundParamIter1);

// Find instance parameter's display value by name
decl foundParamIter2 = db_param_iter_find_by_name(paramIter, "W");
decl value2 = "";
if (db_param_iter_is_valid(foundParamIter2))
   value2 = db_param_iter_get_display_value(foundParamIter2);
```