ENTERPRISE ARCHITECT

User Guide Series

Scripting

Author: Sparx Systems

Date: 2022-10-03

Version: 16.0

CREATED WITH ENTERPRISE ARCHITECT

Table of Contents

Scripting 6
Scripting Window 9
Script Group Properties 11
JavaScript Math Library 13
Arithmetic and Algebraic 14
sqrt 15
lsqrt 16
cbrt 17
polevl, p1evl 18
chbevl 20
round 21
floor 22
ceil 23
frexp 24
ldexp 25
fabs 26
signbit 27
isnan 28
isfinite 29
poladd 30
polsub 31
polmul 32
poldiv 33
polsbt 34
poleva 35
polclr 36
polmov 37
Exponential and Trigonometric 38
acos 39
acosh 40
asinh 41
atanh 42
asin 43
atan 44
atan2 45
cos 46
cosdg 47
exp 48
exp2 49
exp10 50
cosh 51
sinh 52
tanh 53
log 54
log2 55
log10 56
pow 57

powi 58
sin 59
sindg 60
tan 61
tandg 62
Exponential integral 63
expn 64
shichi 65
sici 67
Gamma 69
beta 70
lbeta 71
fac 72
gamma 73
lgam 74
incbet 75
incbi 77
igam 78
igamc 79
igami 80
psi 81
rgamma 83
Error function 84
erf 85
erfc 86
dawsn 87
fresnl 88
Bessel 90
airy 91
j0 93
j1 94
jn 95
jv 96
y0 97
y1 98
yn 99
yv 100
i0 101
i0e 102
i1 103
i1e 104
iv 105
k0 106
k0e 107
k1 108
k1e 109
kn 110
Hypergeometric 111
hyperg 112
hyp2f1 113
hyp2f0 115

onef2 116
threef0 117
Elliptic 118
ellpe 119
ellie 120
ellpk 121
ellik 123
ellpj 124
Probability 125
bdtr 126
bdtrc 128
bdtri 130
chdtr 131
chdtrc 132
chdtri 133
fdtr 134
fdtrc 135
fdtri 137
gdtr 139
gdtrc 140
nbdtr 141
nbdtrc 142
ndtr 143
ndtri 144
pdtr 145
pdtrc 146
pdtri 147
stdtr 148
Miscellaneous 150
polylog 151
spence 153
zetac 154
zeta 155
struve 157
Matrix 158
fftr 159
simq 160
minv 161
mmmpy 163
mvmpy 164
mtransp 165
eigens 166
Numerical Integration 169
simpsn 170
Complex Arithmetic 171
cadd 172
csub 174
cmul 176
cdiv 178
cabs 180
csqrt 181

Complex Exponential and Trigonometric 183
cexp 184
clog 185
ccos 186
cacos 187
csin 188
casin 189
ctan 190
catan 191
ccot 192
errors 193
JavaScript Console 194
Console Window 197
Solvers Interface 198
Script Editor 199
Session Object 202
Workflows 203
Workflow Script Functions 204
Functions - Validate and Control User Input 206
Functions - Create a Search With User Tasks 208
Filled Workflow Data Structures 209
Workflow Data Structures You Fill 211
Functions You Call 212
Script Debugging 213

Scripting

Enterprise Architect's scripting environment is a flexible and easy to use facility that supports both JavaScript and the Microsoft scripting languages JScript and VBScript. When any script runs, it has access to a built-in 'Repository' object. Using this script object you can programmatically inspect and/or modify elements within your currently open model. Enterprise Architect also provides feature rich editors, and tools to run, debug and manage your scripts. Scripts are modular and can include other scripts by name using the *!include* directive. They can be used for a broad range of purposes, from documentation to validation and refactoring, and they can be of enormous help with automating time consuming tasks.

Script Engine Support

- Mozilla SpiderMonkey [version 1.8]
- Microsoft Scripting Engine

Script Languages

- JavaScript
- JScript
- VBScript

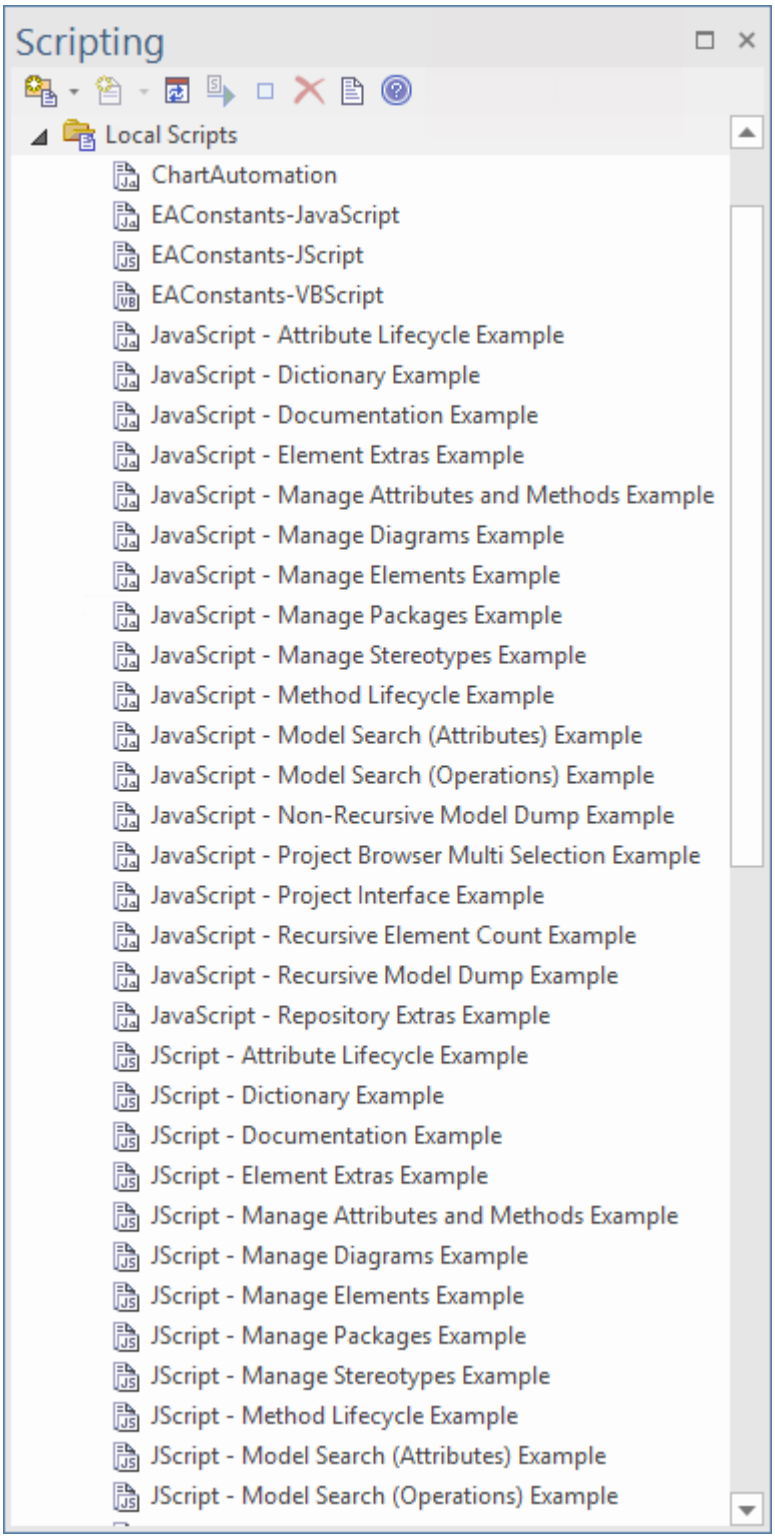
Benefits

- Inspecting and reporting on model and element composition
- Modifying and updating element properties
- Running queries to obtain extended model information
- Modifying diagram layouts
- Being called from report document templates to populate reports
- Creating and implementing process workflows
- Being included in MDG Technologies to augment domain specific languages
- Extensive UI access to scripts through context menus
- Automation Server role for in-process and out-of-process COM clients (Scripting is itself an example of an in-process client; Add-Ins are another)
- Element access governance through Workflow security
- Model Search integration

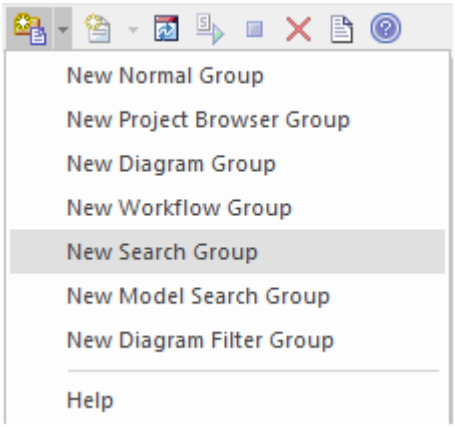
Using Scripts

The management interface for Scripting is the Scripting window, showing the Script Tree View, which you use to review, create and edit scripts.

Other than the Local Scripts, which are file based and installed with Enterprise Architect, all other scripts are stored as model assets and can be shared with all users of the model. Script debuggers can help you with script development and script editors can provide you with information on the automation interfaces available to you. You can analyze the execution, for example by recording a Sequence diagram of the script execution and halting execution to view local variables.



Script Groups



Scripts are managed and contained in groups. Each group has an attribute called 'Type'. This attribute is used to help Enterprise Architect decide how and where the script can be used and from which features it should be made available. The properties of a script group can be viewed from its shortcut menu.

Script Storage

Built-in scripts are file based and are installed with Enterprise Architect. They appear under the *Local Scripts* group.

You cannot edit or delete Local scripts, but you can copy the contents easily enough.

User-defined scripts are model based, and as such can be shared by a community. They are listed in the group to which they belong.

Using Solvers

Anywhere in Enterprise Architect that has JavaScript code, such as in Simulation, you can now use a JavaScript construct called 'Solver' (the Solver Class) to integrate with external tools and have direct use of the functionality within each tool to simply and intuitively perform complex maths and charting functions. The calls help you to easily interchange variables between the built in JavaScript engine and each environment. Two Math Libraries that are supported are MATLAB and Octave.

To use the Solver Class, you need to have a knowledge of the functions available in your preferred Math Library and the parameters they use, as described in the product documentation.

Being part of the JavaScript engine, Solver Classes are also immediately accessible to Add-In writers creating model based JavaScript Add-Ins.

Also see the *Octave Solver, MATLAB Solver* and *Solvers* Help topics.

Notes

- This facility is available in the Corporate, Unified and Ultimate Editions
- If you intend to use the Scripting facility under Crossover/WINE, you must also install Internet Explorer version 6.0 or above

Scripting Window

The Scripting window is composed of a toolbar and a view of all scripts according to group. The script groups and their scripts also have context menus that provide some or all of these options:

- Group Properties - to display or edit script group properties in the 'Script Group Properties' dialog
- Run Script - to execute the selected script (or press Ctrl while you double-click on the script name)
- Debug Script - to debug the selected script
- Edit Script - to update the selected script (or double-click on the script name to display the 'Script Editor', which usually displays a script template, determined by the user group type as assigned on creation or on the 'Script Group Properties' dialog)
- Rename Script - to change the name of the selected group or script
- New VBScript/JScript/JavaScript - add a new script to the selected user group
- Import Workflow Script - to display the 'Browser' dialog through which you locate and select a workflow script source (.vbs) file to import into the Workflow script folder
- Delete Group/Script - to delete the selected user group or script

You can also move or copy a script from one user scripts folder to another; to:

- Move a script, highlight it in the Scripting window and drag it into the user scripts folder it now belongs to
- Copy a script, highlight it in the Scripting window and press Ctrl while you drag it into the user scripts folder in which to duplicate it

Access

Ribbon	Specialize > Tools > Script Library

Script Toolbar

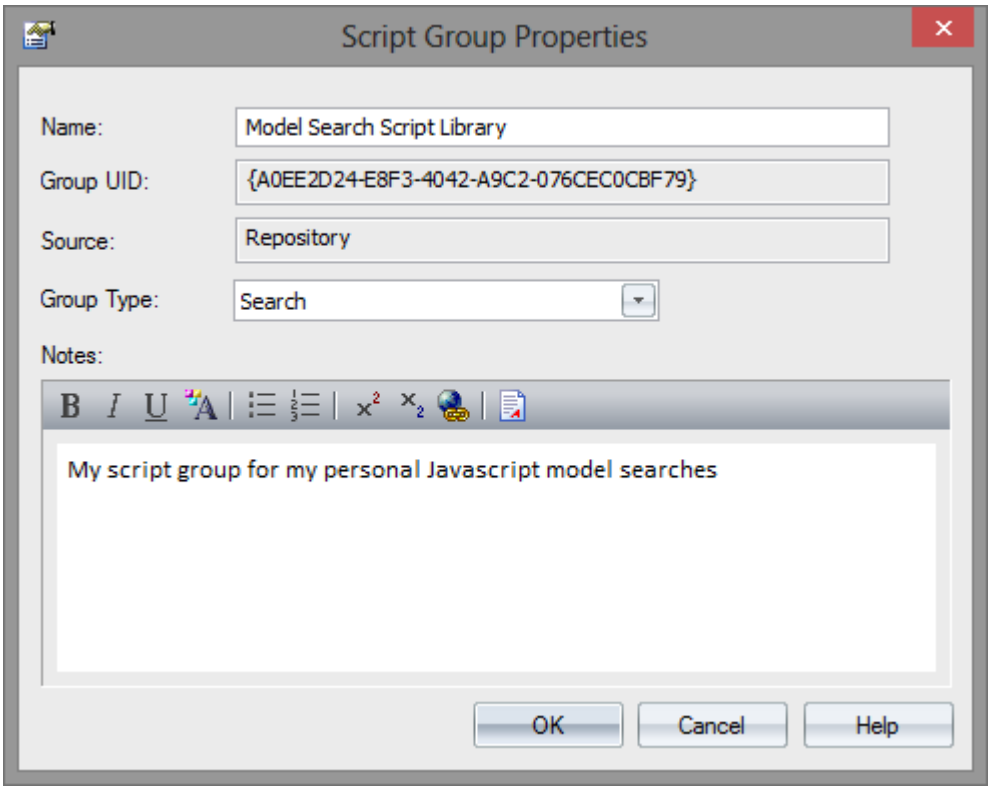
Icon	Action
	Create a new script group; this option displays a short menu of the types of script group you can create, namely: • Normal Group ( ) • Browser window Group ( ) • Diagram Group ( ) • Workflow Group ( ) • Search Group ( ) • Model Search Group The new group is added to the end of the list in the Scripting window, with the 'New group' text highlighted so that you can type in the group name.
	Create a new script file in the selected script group; this displays a short menu of the types of script you can create, namely:

	• VBScript ( ) • JScript ( ) • JavaScript ( ) The new script is added to the end of the list in the selected group, with the 'New script' text highlighted so that you can type in the script name.
	Refresh the script tree in the Scripting window; this icon also reloads any changes made to a workflow script.
	Compile and execute the selected script. The output from the script is written to the 'Script' tab of the System Output window, which you display using the View Script Output button.
	Stop an executing script; the icon is disabled if no script is executing.
	Delete a script from the model; you cannot use this icon to delete a script group (see the earlier 'Context Menu' item), scripts in the 'Local Scripts' group, or a script that is executing. The system prompts you to confirm the deletion only if the 'Confirm Deletes' checkbox is selected in the 'Project Browser' panel of the 'General' page of the 'Preferences' dialog; if this option is not selected, no prompt is displayed. Script deletion is permanent - scripts cannot be recovered.
	Display the System Output window with the results of the most recently executed script displayed in the 'Script' tab.

Notes

- This facility is available in the Corporate, Unified and Ultimate Editions
- If you add, delete or change a script, you might have to reload the model in order for the changes to take effect
- If you select to delete a script group that contains scripts, the system always prompts you to confirm the action regardless of any system settings for delete operations; be certain that you intend to delete the group and its scripts before confirming the deletion - deletion of script groups and scripts is permanent

Script Group Properties



When you create a script you develop it within a script group, the properties of which determine how that script is to be made available to the user - through the Browser window context menu to operate on objects of a specific type, or through a diagram context menu. You create a Script Group using the first icon on the Scripting window toolbar.

Access

Ribbon	Specialize > Tools > Script Library > Scripts > right-click on [Group name] > Group Properties

Define the Script Group Properties

Field/Button	Action
Name	Type in the name of the script group.
Group UID	(Read only) The automatically assigned GUID for the group.
Source	(Read only) The location of the template used to create the script.
Group Type	Click on the drop-down arrow and select the type of script contained in the group; this can be one of:

	• Normal - ( ) General model scripts • Browser window - ( ) Scripts that are listed in and can be executed from the Browser window 'Scripts' context menu option • Workflow - ( ) Scripts executed by Enterprise Architect's workflow engine; you can create only VB scripts of this type • Search - ( ) Scripts that can be executed as model searches; these scripts are listed in the 'Search' field of the Model Search window, in the last category in the list • Diagram - ( ) Scripts that can be executed from the 'Scripts' submenu of the diagram context menu • Find in Project - ( ) Scripts that can be executed from the 'Scripts' submenu of a context menu within the Model Search view, on the results of a successfully-executed SQL search that includes CLASSGUID and CLASSTYPE, or a Query-built search • Element - Scripts that can be executed from the 'Scripts' submenu of element context menus; accessible from the Browser window, Diagram, Model Search, Element List, Package Browser and Gantt views • Package - Scripts that can be executed from the 'Scripts' submenu of Package context menus; accessible from the Browser window • Diagram - Scripts that can be executed from the 'Scripts' context menu option for diagrams; accessible from the Browser window and diagrams • Link - Scripts that can be executed from the 'Scripts' context menu option for connectors; accessible from diagrams
Notes	Type in any comments you need regarding this script group.

JavaScript Math Library

The legendary Cephes Math Library is fully and tightly integrated with the JavaScript engine available within Enterprise Architect. This library is a collection of more than 400 high-quality mathematical routines for scientific and engineering applications, providing a huge range of mathematical potential for modelers wanting to take their engineering and systems models to the next level.

The function library implements the IEEE Std 754 double-precision standard.

- Arithmetic and Algebraic
- Exponential and Trigonometric
- Exponential integral
- Gamma
- Error function
- Bessel
- Hypergeometric
- Elliptic
- Probability
- Miscellaneous
- Matrix
- Numerical Integration
- Complex Arithmetic
- Complex Exponential and Trigonometric
- errors

Arithmetic and Algebraic

- sqrt - square root
- lsqrt - integer square root
- cbrt - cube root
- polevl, p1evl - evaluate polynomial
- chbevl - evaluate Chebyshev series
- round - round to nearest integer value
- ceil - truncate upward to integer
- floor - truncate downward to integer
- frexp - extract exponent
- ldexp - add integer to exponent
- fabs - absolute value
- signbit - return sign bit as int
- isnan - number test
- isfinite - finite test
- poladd - add polynomials
- polsub - subtract polynomials
- polmul - multiply polynomials
- poldiv - divide polynomials
- polsbt - substitute polynomial variable
- poleva - evaluate polynomial
- polclr - set all coefficients to zero
- polmov - copy coefficients

sqrt

Square root.

SYNOPSIS:

double x, y, sqrt();
y = sqrt(x);

DESCRIPTION:

Returns the square root of x.

Range reduction involves isolating the power of two of the argument and using a polynomial approximation to obtain a rough value for the square root. Then Heron's iteration is used three times to converge to an accurate value.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	0, 10	60000	2.1e-17	7.9e-18
IEEE	0,1.7e308	30000	1.7e-16	6.3e-17

ERROR MESSAGES:

message	condition	value returned
domain	x < 0	0.0

lsqrt

Integer square root.

SYNOPSIS:

long x, y;

long lsqrt();

y = lsqrt(x);

DESCRIPTION:

Returns a long integer square root of the long integer argument. The computation is by binary long division. The largest possible result is lsqrt(2,147,483,647) = 46341.

If x < 0, the square root of |x| is returned, and an error message is available.

ACCURACY:

An extra, roundoff, bit is computed; hence the result is the nearest integer to the actual square root.

cbrt

Cube root.

SYNOPSIS:

double x, y, cbrt();
y = cbrt(x);

DESCRIPTION:

Returns the cube root of the argument, which could be negative. Range reduction involves determining the power of 2 of the argument. A polynomial of degree 2 applied to the mantissa, and multiplication by the cube root of 1, 2, or 4 approximates the root to within about 0.1%. Then Newton's iteration is used three times to converge to an accurate result.

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
DEC	-10,10	200000	1.8e-17	6.2e-18
IEEE	0,1e308	30000	1.5e-16	5.0e-17

JavaScript:

```
//
//Plot of y = 3vx.
//

function plotYforX(x1, x2)
{
	for(var x = x1; x <= x2; x++)
	{
		var y = cephes.cbrt(x);
		Session.Output("plot of x for " + x + " gives y of " + y);
	}
}

function main()
{
	plotYforX(-1,6);
}
main();
```

polevl, p1evl

Evaluate polynomial.

SYNOPSIS:

```
int N;
double x, y, coef[N+1], polevl[];
y = polevl(x, coef, N);
```

DESCRIPTION:

Evaluates polynomial of degree N:

$$y = C_0 + C_1 x + C_2 x^2 + \ldots + C_N x^N$$

Coefficients are stored in reverse order:

$coef[0] = C_N$, ..., $coef[N] = C_0$.

The function p1evl() assumes that coef[N] = 1.0 and is omitted from the array. Its calling arguments are otherwise the same as polevl().

SPEED:

In the interest of speed, there are no checks for out of bounds arithmetic. This routine is used by most of the functions in the library. Depending on available equipment features, the user might want to rewrite the program in microcode or assembly language.

JavaScript:

Example:

```
function stirlingFormula(x)
{
    var STIR = [ 7.873113957930936 28397E-4, -2.29549961613378126380E-4,
            -2.68132617805781232825E-3, 3.47222221605458667310E-3,
             8.33333333333482257126E-2 ];
    var SQTPI = 2.50662827463100050242E0;
    var MAXSTIR = 143.01608;
    var w = 1.0 / x;
    var y = cephes.exp(x);
```

```
    var w = 1.0 + w * cephes.polevl(w, STIR, 4);
    if (x > MAXSTIR) {
        var v = cephes.pow(x, 0.5 * x - 0.25);
        y = v * (v / y);
    } else {
        y = cephes.pow(x, x - 0.5) / y;
    }
    y = SQTPI * y * w;
    return y;
}
```

chbevl

Evaluate Chebyshev series.

SYNOPSIS:

```
int N;
double x, y, coef[N], chebevl();

y = chbevl(x, coef, N);
```

DESCRIPTION:

Evaluates the series

$$y = \sum_{i=0}^{N-1} {}' \; coef[i] \; T_i(x/2)$$

of Chebyshev polynomials Ti at argument x/2.

Coefficients are stored in reverse order, i.e. the zero order term is last in the array. Note N is the number of coefficients, not the order.

If coefficients are for the interval a to b, x must have been transformed to x -> 2(2x - b - a)/(b-a) before entering the routine. This maps x from (a, b) to (-1, 1), over which the Chebyshev polynomials are defined.

If the coefficients are for the inverted interval, in which (a, b) is mapped to (1/b, 1/a), the transformation required is x -> 2(2ab/x - b - a)/(b-a). If b is infinity, this becomes x -> 4a/x - 1.

SPEED:

Taking advantage of the recurrence properties of the Chebyshev polynomials, the routine requires one more addition per loop than evaluating a nested polynomial of the same degree.

JavaScript:

```
var y = cephes.chbevl(x, coef, N);
```

round

Round double to nearest or even integer valued double

SYNOPSIS:

double x, y, round();

y = round(x);

DESCRIPTION:

Returns the nearest integer to x as a double precision floating point result. If x ends in 0.5 exactly, the nearest even integer is chosen.

ACCURACY:

If x is greater than 1/(2*MACHEP), its closest machine representation is already an integer, so rounding does not change it.

floor

SYNOPSIS:

```
double floor(x);
 double x,y;
y = floor(x);
```

DESCRIPTION:

floor() returns the largest integer less than or equal to x. It truncates toward minus infinity.

ceil

SYNOPSIS:

```
double ceil(x);
 double x, y;
y = ceil(x);
```

DESCRIPTION:

ceil() returns the smallest integer greater than or equal to x. It truncates toward plus infinity.

frexp

Extract exponent.

SYNOPSIS:

```
double frexp(x, expnt);
 double x;
 int expnt;
y = frexp(x, &expnt);
```

DESCRIPTION:

frexp() extracts the exponent from x. It returns an integer power of two to expnt and the significand between 0.5 and 1 to y. Thus x = y * 2**expn.

ldexp

SYNOPSIS:

```
double ldexp(x,n);
 double x;
 int n;
y = ldexp(x, n);
```

DESCRIPTION:

ldexp() multiplies x by 2**n.

fabs

Absolute value.

SYNOPSIS:

```
double x, y;
y = fabs(x);
```

DESCRIPTION:

Returns the absolute value of the argument.

signbit

SYNOPSIS:

```
int signbit(x);
double x;
int n;
n = signbit(x);
```

DESCRIPTION:

signbit(x) returns 1 if the sign bit of x is 1, else 0.

isnan

SYNOPSIS:

```
int isnan(x);
double x;
int n;

n = isnan(x);
```

DESCRIPTION:

Returns true if x is not a number.

isfinite

SYNOPSIS:

```
int isfinite();
double x;
int n;

n = isfinite(x);
```

DESCRIPTION:

Return true if x is not infinite and is not a NaN

poladd

Polynomial Addition

SYNOPSIS:

```
int maxpol, na, nb, nc;
double a[na], b[nb], c[nc];

nc = max(na, nb);
polini( nc );
poladd( a, na, b, nb, c );
```

DESCRIPTION:

poladd(a, na, b, nb, c); c = b + a, nc = max(na, nb)

In this description a, b, c are polynomials of degree na, nb, nc respectively.

The degree of a polynomial cannot exceed a run-time value MAXPOL.

An operation that attempts to use or generate a polynomial of higher degree might produce a result that suffers truncation at degree MAXPOL.

The value of MAXPOL is set by calling the function

polini(MAXPOL);

Each polynomial is represented by an array containing its coefficients, together with a separately declared integer equal to the degree of the polynomial.

The coefficients appear in ascending order; that is,

$$a(x) = a[0] + a[1] * x + a[2] * x^2 + \ldots + a[na] * x^{na} .$$

polsub

Polynomial Subtraction

SYNOPSIS:

```
int maxpol, na, nb, nc;
double a[], b[], c[];

nc = max(na, nb);
polini( nc );
polsub( a, na, b, nb, c );
```

DESCRIPTION:

polsub(a, na, b, nb, c); c = b - a, nc = max(na, nb)

a, b, c are polynomials of degree na, nb, nc respectively.

The degree of a polynomial cannot exceed a run-time value MAXPOL.

An operation that attempts to use or generate a polynomial of higher degree might produce a result that suffers truncation at degree MAXPOL.

The value of MAXPOL is set by calling the function

polini(MAXPOL);

Each polynomial is represented by an array containing its coefficients, together with a separately declared integer equal to the degree of the polynomial.

The coefficients appear in ascending order; that is:

$$a(x) = a[0] + a[1] * x + a[2] * x^2 + \ldots + a[na] * x^{na} .$$

polmul

Polynomial Multiplication

SYNOPSIS:

```
int maxpol, na, nb, nc;
double a[], b[], c[];

nc = na + nb;
polini( nc );
polmul( a, na, b, nb, c );
```

DESCRIPTION:

polmul(a, na, b, nb, c); c = b * a, nc = na + nb

a, b, c are polynomials of degree na, nb, nc respectively.

The degree of a polynomial cannot exceed a run-time value MAXPOL.

An operation that attempts to use or generate a polynomial of higher degree might produce a result that suffers truncation at degree MAXPOL.

The value of MAXPOL is set by calling the function

polini(MAXPOL);

Each polynomial is represented by an array containing its coefficients, together with a separately declared integer equal to the degree of the polynomial.

The coefficients appear in ascending order; that is,

$$a(x) = a[0] + a[1] * x + a[2] * x^2 + \ldots + a[na] * x^{na} .$$

poldiv

Polynomial Division

SYNOPSIS:

```
int maxpol, na, nb, nc;
double a[], b[], c[];

nc = na + nb
polini( MAXPOL );
i = poldiv( a, na, b, nb, c );
```

DESCRIPTION:

i = poldiv(a, na, b, nb, c); c = b / a, nc = MAXPOL

returns i = the degree of the first nonzero coefficient of a.

The computed quotient c must be divided by x^i.

An error message is printed if a is identically zero.

a, b, c are polynomials of degree na, nb, nc respectively.

The degree of a polynomial cannot exceed a run-time value MAXPOL.

An operation that attempts to use or generate a polynomial of higher degree might produce a result that suffers truncation at degree MAXPOL.

The value of MAXPOL is set by calling the function

polini(MAXPOL);

Each polynomial is represented by an array containing its coefficients, together with a separately declared integer equal to the degree of the polynomial.

The coefficients appear in ascending order; that is,

$$a(x) = a[0] + a[1] * x + a[2] * x^2 + \ldots + a[na] * x^{na} .$$

polsbt

Substitute Polynomial Variable

SYNOPSIS:

```
int a, b;
double a[na], b[nb], c[nc];
polsbt( a, na, b, nb, c );
```

DESCRIPTION:

If a and b are polynomials, and t = a(x), then

c(t) = b(a(x))

is a polynomial found by substituting a(x) for t.

The subroutine call for this is:

```
polsbt( a, na, b, nb, c );
```

a, b, c are polynomials of degree na, nb, nc respectively.

The degree of a polynomial cannot exceed a run-time value MAXPOL.

An operation that attempts to use or generate a polynomial of higher degree might produce a result that suffers truncation at degree MAXPOL.

The value of MAXPOL is set by calling the function

```
polini( MAXPOL );
```

Each polynomial is represented by an array containing its coefficients, together with a separately declared integer equal to the degree of the polynomial.

The coefficients appear in ascending order; that is,

$$a(x) = a[0] + a[1] * x + a[2] * x^2 + ... + a[na] * x^{na} .$$

poleva

Polynomial Evaluation

SYNOPSIS:

```
int na;
double sum, x;
double a[na];

sum = poleva( a, na, x );
```

DESCRIPTION:

Evaluate polynomial a(t) at t = x.

The polynomial is represented by an array containing its coefficients, together with a separately declared integer equal to the degree of the polynomial.

The coefficients appear in ascending order; that is,

$$a(x) = a[0] + a[1] * x + a[2] * x^2 + ... + a[na] * x^{na} .$$

polclr

Clear Polynomial

SYNOPSIS:

```
int na;
double a[na];
polclr( a, na );
```

DESCRIPTION:

Set all coefficients of polynomial a to zero, up to a[na].

The polynomial is represented by an array containing its coefficients, together with a separately declared integer equal to the degree of the polynomial.

The coefficients appear in ascending order; that is,

$$a(x) = a[0] + a[1] * x + a[2] * x^2 + \ldots + a[na] * x^{na} .$$

polmov

Move Polynomial

SYNOPSIS:

```
int na;
double a[na], b[na];
polmov( a, na, b );
```

DESCRIPTION:

Set b = a. Copies coefficients of polynomial a, to b.

The polynomial is represented by an array containing its coefficients, together with a separately declared integer equal to the degree of the polynomial.

The coefficients appear in ascending order; that is,

$$a(x) = a[0] + a[1] * x + a[2] * x^2 + ... + a[na] * x^{na} .$$

Exponential and Trigonometric

- acos - Arc cosine
- acosh - Arc hyperbolic cosine
- asinh - Arc hyperbolic sine
- atanh - Arc hyperbolic tangent
- asin - Arcsine
- atan - Arctangent
- atan2 - Quadrant correct arctangent
- cos - Cosine
- cosdg - Cosine of arg in degrees
- exp - Exponential, base e
- exp2 - Exponential, base 2
- exp10 - Exponential, base 10
- cosh - Hyperbolic cosine
- sinh - Hyperbolic sine
- tanh - Hyperbolic tangent
- log - Logarithm, base e
- log2 - Logarithm, base 2
- log10 - Logarithm, base 10
- pow - Power
- powi - Integer power
- sin - Sine
- sindg - Sine of arg in degrees
- tan - Tangent
- tandg - Tangent of arg in degrees

acos

Inverse circular cosine.

SYNOPSIS:

double x, y, acos();
y = acos(x);

DESCRIPTION:

Returns radian angle between 0 and pi whose cosine is x.

Analytically, acos(x) = pi/2 - asin(x). However if |x| is near 1, there is cancellation error in subtracting asin(x) from pi/2. Hence if x < -0.5, acos(x) = pi - 2.0 * asin(sqrt((1+x)/2)); or if x > +0.5, acos(x) = 2.0 * asin(sqrt((1-x)/2)).

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	-1, 1	50000	3.3e-17	8.2e-18
IEEE	-1, 1	10^6	2.2e-16	6.5e-17

ERROR MESSAGES:

message	condition	value returned
domain	|x| > 1	NAN

acosh

Inverse hyperbolic cosine.

SYNOPSIS:

```
double x, y, acosh();
y = acosh(x);
```

DESCRIPTION:

Returns the inverse hyperbolic cosine of an argument.

If 1 <= x < 1.5, a rational approximation:

sqrt(z) * P(z)/Q(z)

where z = x-1, is used. Otherwise:

acosh(x) = log(x + sqrt((x-1)(x+1)).

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	1,3	30000	4.2e-17	1.1e-17
IEEE	1,3	30000	4.6e-16	8.7e-17

ERROR MESSAGES:

message	condition	value returned
domain	|x| < 1	NAN

asinh

Inverse hyperbolic sine.

SYNOPSIS:

double x, y, asinh();
y = asinh(x);

DESCRIPTION:

Returns the inverse hyperbolic sine of an argument.

If |x| < 0.5, the function is approximated by a rational form x + x**3 P(x)/Q(x).
Otherwise, asinh(x) = log(x + sqrt(1 + x*x)).

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	-3,3	75000	4.6e-17	1.1e-17
IEEE	-1,1	30000	3.7e-16	7.8e-17
IEEE	1,3	30000	2.5e-16	6.7e-17

atanh

Inverse hyperbolic tangent.

SYNOPSIS:

double x, y, atanh();
y = atanh(x);

DESCRIPTION:

Returns the inverse hyperbolic tangent of an argument in the range MINLOG to MAXLOG.

If |x| < 0.5, the rational form x + x**3 P(x)/Q(x) is employed. Otherwise:
atanh(x) = 0.5 * log((1+x)/(1-x)).

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	-1,1	50000	2.4e-17	6.4e-18
IEEE	-1,1	30000	1.9e-16	5.2e-17

asin

Inverse circular sine.

SYNOPSIS:

double x, y, asin();
y = asin(x);

DESCRIPTION:

Returns the radian angle between -pi/2 and +pi/2 whose sine is x.

A rational function of the form x + x**3 P(x**2)/Q(x**2) is used for |x| in the interval [0, 0.5]. If |x| > 0.5 it is transformed by the identity:

asin(x) = pi/2 - 2 asin(sqrt((1-x)/2)).

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	-1, 1	40000	2.6e-17	7.1e-18
IEEE	-1, 1	10^6	1.9e-16	5.4e-17

ERROR MESSAGES:

message	condition	value returned
domain	|x| > 1	NAN

atan

Inverse circular tangent (arctangent).

SYNOPSIS:

double x, y, atan();
y = atan(x);

DESCRIPTION:

Returns the radian angle between -pi/2 and +pi/2 whose tangent is x.

Range reduction is from three intervals into the interval from zero to 0.66. The approximant uses a rational function of degree 4/5 of the form x + x**3 P(x)/Q(x).

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	-10, 10	50000	2.4e-17	8.3e-18
IEEE	-10, 10	10^6	1.8e-16	5.0e-17

atan2

Quadrant correct inverse circular tangent.

SYNOPSIS:

double x, y, z, atan2();
z = atan2(y, x);

DESCRIPTION:

Returns the radian angle whose tangent is y/x.
Define compile time symbol ANSIC = 1 for ANSI standard, range -PI < z <= +PI, args (y,x);
else ANSIC = 0 for range 0 to 2PI, args (x,y).

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
IEEE	-10, 10	10^6	2.5e-16	6.9e-17

cos

Circular cosine.

SYNOPSIS:

```
double x, y, cos();
y = cos(x);
```

DESCRIPTION:

Range reduction is into intervals of pi/4. The reduction error is nearly eliminated by contriving an extended precision modular arithmetic.

Two polynomial approximating functions are employed.

Between 0 and pi/4 the cosine is approximated by:

```
1  -  x**2 Q(x**2).
```

Between pi/4 and pi/2 the sine is represented as:

```
x  +  x**3 P(x**2).
```

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
IEEE	-1.07e9,+1.07e9	130000	2.1e-16	5.4e-17
DEC	0,+1.07e9	17000	3.0e-17	7.2e-18

cosdg

Circular cosine of angle in degrees.

SYNOPSIS:

double x, y, cosdg();
y = cosdg(x);

DESCRIPTION:

Range reduction is into intervals of 45 degrees. Two polynomial approximating functions are employed.
Between 0 and pi/4 the cosine is approximated by:
1 - x**2 P(x**2).
Between pi/4 and pi/2 the sine is represented as:
x + x**3 P(x**2).

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
DEC	+-1000	3400	3.5e-17	9.1e-18
IEEE	+-1000	30000	2.1e-16	5.7e-17

exp

Exponential function.

SYNOPSIS:

double x, y, exp();
y = exp(x);

DESCRIPTION:

Returns e (2.71828...) raised to the x power.

Range reduction is accomplished by separating the argument into an integer k and fraction f such that:

$$e^{x} = 2^{k} e^{f}$$

A Pade' form
1 + 2x P(x**2)/(Q(x**2) - P(x**2)) of degree 2/3 is used to approximate exp(f) in the basic interval [-0.5, 0.5].

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
DEC	+- 88	50000	2.8e-17	7.0e-18
IEEE	+- 708	40000	2.0e-16	5.6e-17

Error amplification in the exponential function can be a serious matter. The error propagation involves:
exp(X(1+delta)) = exp(X) (1 + X*delta + ...)
This shows that a 1 lsb error in representing X produces a relative error of X times 1 lsb in the function. While the routine gives an accurate result for arguments that are exactly represented by a double precision computer number, the result contains an amplified roundoff error for large arguments not exactly represented.

ERROR MESSAGES:

message	condition	value returned
underflow	x < MINLOG	0.0
overflow	x > MAXLOG	INFINITY

exp2

Base 2 exponential function.

SYNOPSIS:

double x, y, exp2();
y = exp2(x);

DESCRIPTION:

Returns 2 raised to the x power.

Range reduction is accomplished by separating the argument into an integer k and fraction f, such that:

$$2^x = 2^k \, 2^f$$

A Pade' form:

1 + 2x P(x**2) / (Q(x**2) - x P(x**2))

approximates 2**x in the basic range [-0.5, 0.5].

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
IEEE	-1022,+1024	30000	1.8e-16	5.4e-17

ERROR MESSAGES:

message	condition	value returned
underflow	x < -MAXL2	0.0
overflow	x > MAXL2	MAXNUM

For DEC arithmetic, MAXL2 = 127.
For IEEE arithmetic, MAXL2 = 1024.

exp10

Base 10 exponential function. (Common antilogarithm.)

SYNOPSIS:

```
double x, y, exp10();
y = exp10(x);
```

DESCRIPTION:

Returns 10 raised to the x power.

Range reduction is accomplished by expressing the argument as 10**x = 2**n 10**f, with |f| < 0.5 log10(2).
The Pade' form:

```
   1 + 2x P(x**2)/(Q(x**2) - P(x**2))
```

is used to approximate 10**f.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
IEEE	-307,+307	30000	2.2e-16	5.5e-17

Test result from an earlier version (2.1):

DEC	-38,+38	70000	3.1e-17	7.0e-18

ERROR MESSAGES:

message	condition	value returned
underflow	x < -MAXL10	0.0
overflow	x > MAXL10	MAXNUM

DEC arithmetic: MAXL10 = 38.230809449325611792.
IEEE arithmetic: MAXL10 = 308.2547155599167.

cosh

Hyperbolic cosine.

SYNOPSIS:

```
double x, y, cosh();
y = cosh(x);
```

DESCRIPTION:

Returns the hyperbolic cosine of an argument in the range MINLOG to MAXLOG.

cosh(x) = (exp(x) + exp(-x))/2.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	+- 88	50000	4.0e-17	7.7e-18
IEEE	+-MAXLOG	30000	2.6e-16	5.7e-17

ERROR MESSAGES:

message	condition	value returned
overflow	|x| > MAXLOG	MAXNUM

sinh

Hyperbolic sine.

SYNOPSIS:

double x, y, sinh();
y = sinh(x);

DESCRIPTION:

Returns the hyperbolic sine of an argument in the range MINLOG to MAXLOG.

The range is partitioned into two segments. If |x| <= 1, a rational function of the form x + x**3 P(x)/Q(x) is employed. Otherwise the calculation is sinh(x) = (exp(x) - exp(-x))/2.

ACCURACY:

			Relative error:	
arithmetic	domain	# trials	peak	rms
DEC	+- 88	50000	4.0e-17	7.7e-18
IEEE	+-MAXLOG	30000	2.6e-16	5.7e-17

tanh

Hyperbolic tangent.

SYNOPSIS:

double x, y, tanh();
y = tanh(x);

DESCRIPTION:

Returns the hyperbolic tangent of an argument in the range MINLOG to MAXLOG.

A rational function is used for |x| < 0.625. The form:
x + x**3 P(x)/Q(x) of Cody _& Waite
is employed.
Otherwise:
tanh(x) = sinh(x)/cosh(x) = 1 - 2/(exp(2x) + 1).

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	-2,2	50000	3.3e-17	6.4e-18
IEEE	-2,2	30000	2.5e-16	5.8e-17

log

Natural logarithm.

SYNOPSIS:

double x, y, log();
y = log(x);

DESCRIPTION:

Returns the base e (2.718...) logarithm of x.

The argument is separated into its exponent and fractional parts. If the exponent is between -1 and +1, the logarithm of the fraction is approximated by:

log(1+x) = x - 0.5 x**2 + x**3 P(x)/Q(x).

Otherwise, setting z = 2(x-1)/x+1),

log(x) = z + z**3 P(z)/Q(z).

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
IEEE	0.5, 2.0	150000	1.44e-16	5.06e-17
IEEE	+-MAXNUM	30000	1.20e-16	4.78e-17
DEC	0, 10	170000	1.8e-17	6.3e-18

In the tests over the interval [+-MAXNUM], the logarithms of the random arguments were uniformly distributed over [0,MAXLOG].

ERROR MESSAGES:

singularity: x = 0; returns -INFINITY
domain: x < 0; returns NAN

log2

Base 2 logarithm.

SYNOPSIS:

double x, y, log2();
y = log2(x);

DESCRIPTION:

Returns the base 2 logarithm of x.

The argument is separated into its exponent and fractional parts. If the exponent is between -1 and +1, the base e logarithm of the fraction is approximated by:

log(1+x) = x - 0.5 x**2 + x**3 P(x)/Q(x).

Otherwise, setting z = 2(x-1)/x+1),

log(x) = z + z**3 P(z)/Q(z).

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
IEEE	0.5, 2.0	30000	2.0e-16	5.5e-17
IEEE	exp(+-700)	40000	1.3e-16	4.6e-17

In the tests over the interval [exp(+-700)], the logarithms of the random arguments were uniformly distributed.

ERROR MESSAGES:

singularity: x = 0; returns -INFINITY
domain: x < 0; returns NAN

log10

Common logarithm.

SYNOPSIS:

double x, y, log10();
y = log10(x);

DESCRIPTION:

Returns logarithm to the base 10 of x.

The argument is separated into its exponent and fractional parts. The logarithm of the fraction is approximated by:

log(1+x) = x - 0.5 x**2 + x**3 P(x)/Q(x).

ACCURACY:

Relative error:

arithmetic	domain	# trials	peak	rms
IEEE	0.5, 2.0	30000	1.5e-16	5.0e-17
IEEE	0, MAXNUM	30000	1.4e-16	4.8e-17
DEC	1, MAXNUM	50000	2.5e-17	6.0e-18

In the tests over the interval [1, MAXNUM], the logarithms of the random arguments were uniformly distributed over [0, MAXLOG].

ERROR MESSAGES:

singularity: x = 0; returns -INFINITY
domain: x < 0; returns NAN

pow

Power function

SYNOPSIS:

double x, y, z, pow();
z = pow(x, y);

DESCRIPTION:

Computes x raised to the yth power. Analytically:

x**y = exp(y log(x)).

Following Cody and Waite, this program uses a lookup table of 2**-i/16 and pseudo extended precision arithmetic to obtain an extra three bits of accuracy in both the logarithm and the exponential.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
IEEE	-26,26	30000	4.2e-16	7.7e-17
DEC	-26,26	60000	4.8e-17	9.1e-18

1/26 < x < 26, with log(x) uniformly distributed.
-26 < y < 26, y uniformly distributed.

IEEE	0,8700	30000	1.5e-14	2.1e-15

0.99 < x < 1.01, 0 < y < 8700, uniformly distributed.

ERROR MESSAGES:

message	condition	value returned
overflow	x**y > MAXNUM	INFINITY
underflow	x**y < 1/MAXNUM	0.0
domain	x<0 and y noninteger	0.0

powi

Real raised to integer power.

SYNOPSIS:

```
double x, y, powi();
int n;

y = powi(x, n);
```

DESCRIPTION:

Returns an argument x raised to the nth power. The routine efficiently decomposes n as a sum of powers of two. The desired power is a product of two-to-the-kth powers of x. Thus to compute the 32767 power of x requires 28 multiplications instead of 32767 multiplications.

ACCURACY:

	Relative error:				
arithmetic	x domain	n domain	# trials	peak	rms
DEC	.04,26	-26,26	100000	2.7e-16	4.3e-17
IEEE	.04,26	-26,26	50000	2.0e-15	3.8e-16
IEEE	1,2	-1022,1023	50000	8.6e-14	1.6e-14

Returns MAXNUM on overflow, zero on underflow.

sin

Circular sine.

SYNOPSIS:

double x, y, sin();
y = sin(x);

DESCRIPTION:

Range reduction is into intervals of pi/4. The reduction error is nearly eliminated by contriving an extended precision modular arithmetic.

Two polynomial approximating functions are employed.
Between 0 and pi/4 the sine is approximated by:

x + x**3 P(x**2).

Between pi/4 and pi/2 the cosine is represented as:
1 - x**2 Q(x**2).

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	0, 10	150000	3.0e-17	7.8e-18
IEEE	-1.07e9,+1.07e9	130000	2.1e-16	5.4e-17

ERROR MESSAGES:

message	condition	value returned
total loss	x > 1.073741824e9	0.0

Partial loss of accuracy begins to occur at x = 2**30 = 1.074e9. The loss is not gradual, but jumps suddenly to about 1 part in 10e7. Results might be meaningless for x > 2**49 = 5.6e14. The routine as implemented flags a TLOSS error for x > 2**30 and returns 0.0.

sindg

Circular sine of an angle in degrees.

SYNOPSIS:

```
double x, y, sindg();
y = sindg(x);
```

DESCRIPTION:

Range reduction is into intervals of 45 degrees. Two polynomial approximating functions are employed.
Between 0 and pi/4 the sine is approximated by:

```
    x  +  x**3 P(x**2).
```

Between pi/4 and pi/2 the cosine is represented as:

```
    1  -  x**2 P(x**2).
```

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
DEC	+-1000	3100	3.3e-17	9.0e-18
IEEE	+-1000	30000	2.3e-16	5.6e-17

ERROR MESSAGES:

message	condition	value returned
total loss	x > 8.0e14 (DEC)	0.0
	x > 1.0e14 (IEEE)	

tan

Circular tangent.

SYNOPSIS:

double x, y, tan();
y = tan(x);

DESCRIPTION:

Returns the circular tangent of the radian argument x.
Range reduction is modulo pi/4.
A rational function:

x + x**3 P(x**2)/Q(x**2)

is employed in the basic interval [0, pi/4].

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
DEC	+-1.07e9	44000	4.1e-17	1.0e-17
IEEE	+-1.07e9	30000	2.9e-16	8.1e-17

ERROR MESSAGES:

message	condition	value returned
total loss	x > 1.073741824e9	0.0

tandg

Circular tangent of argument in degrees.

SYNOPSIS:

```
double x, y, tandg();
y = tandg(x);
```

DESCRIPTION:

Returns the circular tangent of the argument x in degrees.
Range reduction is modulo pi/4. A rational function:

```
x + x**3 P(x**2)/Q(x**2)
```

is employed in the basic interval [0, pi/4].

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	0,10	8000	3.4e-17	1.2e-17
IEEE	0,10	30000	3.2e-16	8.4e-17

ERROR MESSAGES:

message	condition	value returned
total loss	x > 8.0e14 (DEC) x > 1.0e14 (IEEE)	0.0
singularity	x = 180 k + 90	MAXNUM

Exponential integral

- expn - Exponential integral
- shichi - Hyperbolic sine and cosine integrals
- sici - Sine and cosine integrals

expn

Exponential integral En.

SYNOPSIS:

```
int n;
double x, y, expn();
y = expn(n, x);
```

DESCRIPTION:

Evaluates the exponential integral.

$$E_n(x) = \int_1^{\infty} \frac{e^{-xt}}{t^n}\, dt.$$

Both n and x must be nonnegative.

The routine employs either a power series, a continued fraction, or an asymptotic formula depending on the relative values of n and x.

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
DEC	0, 30	5000	2.0e-16	4.6e-17
IEEE	0, 30	10000	1.7e-15	3.6e-16

shichi

Hyperbolic sine and cosine integrals.

SYNOPSIS:

```
double x, Chi, Shi, shichi();
shichi(x, &Chi, &Shi);
```

DESCRIPTION:

Approximates the integrals

$$\mathrm{Chi}(x) = \mathrm{eul} + \ln x + \int_0^x \frac{\cosh t - 1}{t}\,dt,$$

$$\mathrm{Shi}(x) = \int_0^x \frac{\sinh t}{t}\,dt$$

where eul = 0.57721566490153286061 is Euler's constant. The integrals are evaluated by power series for x < 8 and by Chebyshev expansions for x between 8 and 88. For large x, both functions approach exp(x)/2x. Arguments greater than 88 in magnitude return MAXNUM.

ACCURACY:

Test interval 0 to 88.

			Relative error:	
arithmetic	function	# trials	peak	rms
DEC	Shi	3000	9.1e-17	
IEEE	Shi	30000	6.9e-16	1.6e-16

Absolute error, except relative when |Chi| > 1:

DEC	Chi	2500	9.3e-17	
IEEE	Chi	30000	8.4e-16	1.4e-16

sici

Sine and cosine integrals.

SYNOPSIS:

```
double x, Ci, Si, sici();
sici(x, &Si, &Ci);
```

DESCRIPTION:

Evaluates the integrals:

$$\mathrm{Ci}(x) = \mathrm{eul} + \ln x + \int_0^x \frac{\cos t - 1}{t}\,dt,$$

$$\mathrm{Si}(x) = \int_0^x \frac{\sin t}{t}\,dt$$

where eul = 0.57721566490153286061 is Euler's constant. The integrals are approximated by rational functions. For x > 8 auxiliary functions f(x) and g(x) are employed such that

$$\mathrm{Ci}(x) = f(x)\sin(x) - g(x)\cos(x)$$
$$\mathrm{Si}(x) = \pi/2 - f(x)\cos(x) - g(x)\sin(x)$$

ACCURACY:

Test interval = [0,50].
Absolute error, except relative when > 1:

arithmetic	function	# trials	peak	rms

IEEE	Si	30000	4.4e-16	7.3e-17
IEEE	Ci	30000	6.9e-16	5.1e-17
DEC	Si	5000	4.4e-17	9.0e-18
DEC	Ci	5300	7.9e-17	5.2e-18

Gamma

- beta - beta
- lbeta - natural log of beta
- fac - factorial
- gamma - gamma
- lgam - logarithm of gamma function
- incbet - incomplete beta integral
- incbi - inverse of incomplete beta integral
- igam - incomplete gamma integral
- igamc - complemented gamma integral
- igami - inverse gamma integral
- psi - Psi (digamma) function
- rgamma - reciprocal Gamma

beta

Beta function.

SYNOPSIS:

```
double a, b, y, beta();
y = beta(a, b);
```

DESCRIPTION:

```
                 -     -
                | (a) | (b)
beta(a, b)  =  -----------
                 -
                | (a+b)
```

For large arguments the logarithm of the function is evaluated using lgam(), then exponentiated.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	0,30	1700	7.7e-15	1.5e-15
IEEE	0,30	30000	8.1e-14	1.1e-14

ERROR MESSAGES:

message	condition	value returned
overflow	log(beta) > MAXLOG	0.0
	a or b <0 integer	0.0

lbeta

Natural log of |beta|.

Return the sign of beta in sgngam.

fac

Factorial function.

SYNOPSIS:

```
double y, fac();
int i;
y = fac(i);
```

DESCRIPTION:

Returns factorial of i = 1 * 2 * 3 * ... * i.
fac(0) = 1.0.

Due to machine arithmetic bounds the largest value of i accepted is 33 in DEC arithmetic or 170 in IEEE arithmetic. Greater values, or negative ones, produce an error message and return MAXNUM.

ACCURACY:

For i < 34 the values are simply tabulated, and have full machine accuracy. If i > 55, fac(i) = gamma(i+1);

Relative error:

arithmetic	domain	peak
IEEE	0, 170	1.4e-15
DEC	0, 33	1.4e-17

gamma

Gamma function.

SYNOPSIS:

double x, y, gamma();
y = gamma(x);

DESCRIPTION:

Returns the gamma function of the argument. The result is correctly signed, and the sign (+1 or -1) is also returned in a global (extern) variable named sgngam. This variable is also filled in by the logarithmic gamma function lgam().

Arguments |x| <= 34 are reduced by recurrence and the function approximated by a rational function of degree 6/7 in the interval (2,3). Large arguments are handled by Stirling's formula. Large negative arguments are made positive using a reflection formula.

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
DEC	-34, 34	10000	1.3e-16	2.5e-17
IEEE	-170,-33	20000	2.3e-15	3.3e-16
IEEE	-33, 33	20000	9.4e-16	2.2e-16
IEEE	33, 171.6	20000	2.3e-15	3.2e-16

Error for arguments outside the test range will be larger owing to error amplification by the exponential function.

lgam

Natural logarithm of gamma function.

SYNOPSIS:

double x, y, lgam();
y = lgam(x);

DESCRIPTION:

Returns the base e (2.718...) logarithm of the absolute value of the gamma function of the argument. The sign (+1 or -1) of the gamma function is returned in a global (extern) variable named sgngam.

For arguments greater than 13, the logarithm of the gamma function is approximated by the logarithmic version of Stirling's formula using a polynomial approximation of degree 4. Arguments between -33 and +33 are reduced by recurrence to the interval [2,3] of a rational approximation. The cosecant reflection formula is employed for arguments less than -33.

Arguments greater than MAXLGM return MAXNUM and an error message.

MAXLGM = 2.035093e36 for DEC arithmetic or 2.556348e305 for IEEE arithmetic.

ACCURACY:

arithmetic	domain	# trials	peak	rms
DEC	0, 3	7000	5.2e-17	1.3e-17
DEC	2.718, 2.035e36	5000	3.9e-17	9.9e-18
IEEE	0, 3	28000	5.4e-16	1.1e-16
IEEE	2.718, 2.556e305	40000	3.5e-16	8.3e-17

The error criterion was relative when the function magnitude was greater than one but absolute when it was less than one.

This test used the relative error criterion, though at certain points the relative error could be much higher than indicated.

IEEE	-200, -4	10000	4.8e-16	1.3e-16

incbet

Incomplete beta integral.

SYNOPSIS:

```
double a, b, x, y, incbet();
y = incbet(a, b, x);
```

DESCRIPTION:

Returns the incomplete beta integral of the arguments, evaluated from zero to x. The function is defined as:

```
                x
   -            -
  | (a+b)      | |  a-1     b-1
 -----------   |   t   (1-t)   dt.
  -    -     | |
 | (a) | (b)   -
               0
```

The domain of definition is 0 <= x <= 1. In this implementation a and b are restricted to positive values. The integral from x to 1 can be obtained by the symmetry relation:

1 - incbet(a, b, x) = incbet(b, a, 1-x).

The integral is evaluated by a continued fraction expansion or, when b*x is small, by a power series.

ACCURACY:

Tested at uniformly distributed random points (a,b,x) with a and b in "domain" and x between 0 and 1.

			Relative error	
arithmetic	domain	# trials	peak	rms
IEEE	0,5	10000	6.9e-15	4.5e-16
IEEE	0,85	250000	2.2e-13	1.7e-14
IEEE	0,1000	30000	5.3e-12	6.3e-13
IEEE	0,10000	250000	9.3e-11	7.1e-12
IEEE	0,100000	10000	8.7e-10	4.8e-11

Outputs smaller than the IEEE gradual underflow threshold were excluded from these statistics.

ERROR MESSAGES:

message	condition	value returned
domain	x<0, x>1	0.0

underflow	0.0

incbi

Inverse of incomplete beta integral.

SYNOPSIS:

```
double a, b, x, y, incbi();
x = incbi(a, b, y);
```

DESCRIPTION:

Given y, the function finds x such that:

incbet(a, b, x) = y .

The routine performs interval halving or Newton iterations to find the root of incbet(a,b,x) - y = 0.

ACCURACY:

				Relative error:	
	x	a,b			
arithmetic	domain	domain	# trials	peak	rms
IEEE	0,1	.5,10000	50000	5.8e-12	1.3e-13
IEEE	0,1	.25,100	100000	1.8e-13	3.9e-15
IEEE	0,1	0,5	50000	1.1e-12	5.5e-15
VAX	0,1	.5,100	25000	3.5e-14	1.1e-15
With a and b constrained to half-integer or integer values:					
IEEE	0,1	.5,10000	50000	5.8e-12	1.1e-13
IEEE	0,1	.5,100	100000	1.7e-14	7.9e-16
With a = .5, b constrained to half-integer or integer values:					
IEEE	0,1	.5,10000	10000	8.3e-11	1.0e-11

igam

Incomplete gamma integral.

SYNOPSIS:

double a, x, y, igam();
y = igam(a, x);

DESCRIPTION:

The function is defined by

$$\mathrm{igam}(a,x) = \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1}\, dt.$$

In this implementation both arguments must be positive. The integral is evaluated by either a power series or continued fraction expansion, depending on the relative values of a and x.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
IEEE	0,30	200000	3.6e-14	2.9e-15
IEEE	0,100	300000	9.9e-14	1.5e-14

igamc

Complemented incomplete gamma integral.

SYNOPSIS:

```
double a, x, y, igamc();
y = igamc(a, x);
```

DESCRIPTION:

The function is defined by

```
 igamc(a,x)   =   1 - igam(a,x)

                    inf.
                     -
              1      | |  -t  a-1
         =  -----    |   e   t    dt
             -      | |
            | (a)    -
                     x
```

In this implementation both arguments must be positive. The integral is evaluated by either a power series or continued fraction expansion, depending on the relative values of a and x.

ACCURACY:

Tested at random a, x.

	a	x	Relative error:		
arithmetic	domain	domain	# trials	peak	rms
IEEE	0.5,100	0,100	200000	1.9e-14	1.7e-15
IEEE	0.01,0.5	0,100	200000	1.4e-13	1.6e-15

igami

Inverse of complemented incomplete gamma integral.

SYNOPSIS:

double a, x, p, igami();
x = igami(a, p);

DESCRIPTION:

Given p, the function finds x such that

igamc(a, x) = p.

Starting with the approximate value

$$x = a\,t^3$$

where

$$t = 1 - d - \mathrm{ndtri}(p)\,\sqrt{d}$$

and

$$d = 1/9a,$$

the routine performs up to 10 Newton iterations to find the root of igamc(a,x) - p = 0.

ACCURACY:

Tested at random a, p in the intervals indicated.

	a	p		Relative error:	
arithmetic	domain	domain	# trials	peak	rms
IEEE	0.5,100	0,0.5	100000	1.0e-14	1.7e-15
IEEE	0.01,0.5	0,0.5	100000	9.0e-14	3.4e-15
IEEE	0.5,10000	0,0.5	20000	2.3e-13	3.8e-14

psi

Psi (digamma) function.

SYNOPSIS:

```
double x, y, psi();
y = psi(x);
```

DESCRIPTION:

$$\text{psi}(x) = \frac{d}{dx} \ln |\Gamma(x)$$

is the logarithmic derivative of the gamma function.
For integer x:

$$\text{psi}(n) = -\text{EUL} + \sum_{k=1}^{n-1} 1/k.$$

This formula is used for $0 < n <= 10$. If x is negative, it is transformed to a positive argument by the reflection formula $\text{psi}(1-x) = \text{psi}(x) + \pi \cot(\pi x)$. For general positive x, the argument is made greater than 10 using the recurrence $\text{psi}(x+1) = \text{psi}(x) + 1/x$. Then this asymptotic expansion is applied:

$$\text{psi}(x) = \log(x) - 1/2x - \sum_{k=1}^{\infty} \frac{B_{2k}}{2k\, x^{2k}}$$

where the B2k are Bernoulli numbers.

ACCURACY:

Relative error (except absolute when |psi| < 1):

arithmetic	domain	# trials	peak	rms
DEC	0,30	2500	1.7e-16	2.0e-17
IEEE	0,30	30000	1.3e-15	1.4e-16

IEEE -30,0 40000 1.5e-15 2.2e-16

ERROR MESSAGES:

message condition value returned
singularity x integer <=0 MAXNUM

rgamma

Reciprocal gamma function.

SYNOPSIS:

double x, y, rgamma();

y = rgamma(x);

DESCRIPTION:

Returns one divided by the gamma function of the argument.

The function is approximated by a Chebyshev expansion in the interval [0,1]. Range reduction is by recurrence for arguments between -34.034 and +34.84425627277176174. 1/MAXNUM is returned for positive arguments outside this range. For arguments less than -34.034 the cosecant reflection formula is applied; logarithms are employed to avoid unnecessary overflow.

The reciprocal gamma function has no singularities, but overflow and underflow could occur for large arguments. These conditions return either MAXNUM or 1/MAXNUM with the appropriate sign.

ACCURACY:

Relative error:

arithmetic	domain	# trials	peak	rms
DEC	-30,+30	4000	1.2e-16	1.8e-17
IEEE	-30,+30	30000	1.1e-15	2.0e-16

For arguments less than -34.034 the peak error is in the order of 5e-15 (DEC), excepting overflow or underflow.

Error function

- erf - Error function
- erfc - Complemented error function
- dawsn - Dawson's integral
- fresnl - Fresnel integral

erf

Error function.

SYNOPSIS:

double x, y, erf();
y = erf(x);

DESCRIPTION:

The integral is

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2)\, dt.$$

The magnitude of x is limited to 9.231948545 for DEC arithmetic; 1 or -1 is returned outside this range.
For 0 <= |x| < 1, erf(x) = x * P4(x**2)/Q5(x**2); otherwise erf(x) = 1 - erfc(x).

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	0,1	14000	4.7e-17	1.5e-17
IEEE	0,1	30000	3.7e-16	1.0e-16

erfc

Complementary error function.

SYNOPSIS:

```
double x, y, erfc();
y = erfc(x);
```

DESCRIPTION:

```
 1 - erf(x) =
                    inf.
                     -
             2       | |        2
  erfc(x)  =  --------    |   exp(- t) dt
           sqrt(pi)   | |
                    -
                     x
```

For small x, erfc(x) = 1 - erf(x); otherwise rational approximations are computed.
A special function expx2.c is used to suppress error amplification in computing exp(-x^2).

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
IEEE	0,26.6417	30000	1.3e-15	2.2e-16

ERROR MESSAGES:

message	condition	value returned
underflow	x > 9.231948545 (DEC)	0.0

dawsn

Dawson's Integral.

SYNOPSIS:

```
double x, y, dawsn();
y = dawsn(x);
```

DESCRIPTION:

Approximates the integral

$$\text{dawsn}(x) = \exp(-x^2) \int_0^x \exp(t^2)\, dt$$

Three different rational approximations are employed, for the intervals 0 to 3.25; 3.25 to 6.25; and 6.25 up.

ACCURACY:

			Relative error:	
arithmetic	domain	# trials	peak	rms
IEEE	0,10	10000	6.9e-16	1.0e-16
DEC	0,10	6000	7.4e-17	1.4e-17

fresnl

Fresnel integral.

SYNOPSIS:

```
double x, S, C;
void fresnl();
fresnl(x, _&S, _&C);
```

DESCRIPTION:

Evaluates the Fresnel integrals

```
          x
          -
         | |
C(x) =   |   cos(pi/2 t**2) dt,
       | |
       -
        0


          x
          -
         | |
S(x) =   |   sin(pi/2 t**2) dt.
       | |
       -
        0
```

The integrals are evaluated by a power series for x < 1. For x >= 1 auxiliary functions f(x) and g(x) are employed such that:

```
C(x) = 0.5 + f(x) sin(pi/2 x**2) - g(x) cos(pi/2 x**2)
S(x) = 0.5 - f(x) cos(pi/2 x**2) - g(x) sin(pi/2 x**2)
```

ACCURACY:

Relative error.

Arithmetic	function	domain	# trials	peak	rms
IEEE	S(x)	0, 10	10000	2.0e-15	3.2e-16
IEEE	C(x)	0, 10	10000	1.8e-15	3.3e-16
DEC	S(x)	0, 10	6000	2.2e-16	3.9e-17
DEC	C(x)	0, 10	5000	2.3e-16	3.9e-17

JavaScript:

```
var x=  2.5625;
var r = cephes.fresnl(x);
Session,Output(r.result);
Session,Output(r.ssa);
Session,Output(r.csa);
```

Return value: Object
Format: JSON

```
{
  "result" : int,
  "ssa" : double,
  "cca" : double
}
```

Bessel

- airy - Airy function
- j0 - Bessel, order 0
- j1 - Bessel, order 1
- jn - Bessel, order n
- jv - Bessel, noninteger order
- y0 - Bessel, second kind, order 0
- y1 - Bessel, second kind, order 1
- yn - Bessel, second kind, order n
- yv - Bessel, noninteger order
- i0 - modified Bessel, order 0
- i0e - exponentially scaled i0
- i1 - modified Bessel, order 1
- i1e - exponentially scaled i1
- iv - modified Bessel, nonint. order
- k0 - modified Bessel, 3rd kind, order 0
- k0e - exponentially scaled k0
- k1 - modified Bessel, 3rd kind, order 1
- k1e - exponentially scaled k1
- kn - modified Bessel, 3rd kind, order n

airy

Airy function.

SYNOPSIS:

double x, ai, aip, bi, bip;
int airy();
airy(x, _&ai, _&aip, _&bi, _&bip);

DESCRIPTION:

Solution of the differential equation:

y"(x) = xy.

The function returns the two independent solutions Ai, Bi and their first derivatives Ai'(x), Bi'(x).
Evaluation is by power series summation for small x, by rational minimax approximations for large x.

ACCURACY:

Error criterion is absolute when function <= 1, relative when function > 1, except * denotes relative error criterion.
For large negative x, the absolute error increases as x^1.5.
For large positive x, the relative error increases as x^1.5.

Arithmetic	domain	function	# trials	peak	rms
IEEE	-10, 0	Ai	10000	1.6e-15	2.7e-16
IEEE	0, 10	Ai	10000	2.3e-14*	1.8e-15*
IEEE	-10, 0	Ai'	10000	4.6e-15	7.6e-16
IEEE	0, 10	Ai'	10000	1.8e-14*	1.5e-15*
IEEE	-10, 10	Bi	30000	4.2e-15	5.3e-16
IEEE	-10, 10	Bi'	30000	4.9e-15	7.3e-16
DEC	-10, 0	Ai	5000	1.7e-16	2.8e-17
DEC	0, 10	Ai	5000	2.1e-15*	1.7e-16*
DEC	-10, 0	Ai'	5000	4.7e-16	7.8e-17
DEC	0, 10	Ai'	12000	1.8e-15*	1.5e-16*
DEC	-10, 10	Bi	10000	5.5e-16	6.8e-17
DEC	-10, 10	Bi'	7000	5.3e-16	8.7e-17

JavaScript:

```
var x = 9.50313909;
var a = cephes.airy(x);
```

Return value: Object

Format: JSON

```
{
  "result" : integer,
  "ai"  : double,
  "aip" : double.
  "bi"  : double,
  "bip" : double
}
```

j0

Bessel function of order zero.

SYNOPSIS:

```
double x, y, j0();
y = j0(x);
```

DESCRIPTION:

Returns a Bessel function of order zero of the argument. The domain is divided into the intervals [0, 5] and (5, infinity). In the first interval this rational approximation is used:

$$(w - r_1^2)(w - r_2^2)\, P(w) / Q(w)$$

where $w = x^2$ and each r is a zero of the function.

In the second interval, the Hankel asymptotic expansion is employed with two rational functions of degree 6/6 and 7/7.

ACCURACY:

Absolute error:

arithmetic	domain	# trials	peak	rms
DEC	0, 30	10000	4.4e-17	6.3e-18
IEEE	0, 30	60000	4.2e-16	1.1e-16

j1

Bessel function of order one.

SYNOPSIS:

double x, y, j1();
y = j1(x);

DESCRIPTION:

Returns a Bessel function of order one of the argument.

The domain is divided into the intervals [0, 8] and (8, infinity). In the first interval a 24 term Chebyshev expansion is used. In the second, the asymptotic trigonometric representation is employed, using two rational functions of degree 5/5.

ACCURACY:

		Absolute error:		
arithmetic	domain	# trials	peak	rms
DEC	0, 30	10000	4.0e-17	1.1e-17
IEEE	0, 30	30000	2.6e-16	1.1e-16

jn

Bessel function of integer order.

SYNOPSIS:

int n;
double x, y, jn();
y = jn(n, x);

DESCRIPTION:

Returns a Bessel function of order n, where n is a (possibly negative) integer.

The ratio of jn(x) to j0(x) is computed by backward recurrence. First the ratio jn/jn-1 is found by a continued fraction expansion. Then the recurrence relating successive orders is applied until j0 or j1 is reached.

If n = 0 or 1 the routine for j0 or j1 is called directly.

ACCURACY:

Absolute error:

arithmetic	range	# trials	peak	rms
DEC	0, 30	5500	6.9e-17	9.3e-18
IEEE	0, 30	5000	4.4e-16	7.9e-17

Not suitable for large n or x. Use jv() instead.

jv

Bessel function of non-integer order.

SYNOPSIS:

double v, x, y, jv();
y = jv(v, x);

DESCRIPTION:

Returns a Bessel function of order v of the argument, where v is real. Negative x is allowed if v is an integer.

Several expansions are included: the ascending power series, the Hankel expansion, and two transitional expansions for large v. If v is not too large, it is reduced by recurrence to a region of best accuracy. The transitional expansions give 12D accuracy for v > 500.

ACCURACY:

Results for integer v are indicated by *, where x and v both vary from -125 to +125. Otherwise, x ranges from 0 to 125, v ranges as indicated by "domain." Error criterion is absolute, except relative when |jv()| > 1.

arithmetic	v domain	x domain	# trials	peak	rms
IEEE	0,125	0,125	100000	4.6e-15	2.2e-16
IEEE	-125,0	0,125	40000	5.4e-11	3.7e-13
IEEE	0,500	0,500	20000	4.4e-15	4.0e-16
Integer v:					
IEEE	-125,125	-125,125	50000	3.5e-15*	1.9e-16*

y0

Bessel function of the second kind, order zero, of the argument.

SYNOPSIS:

```
double x, y, y0();
y = y0(x);
```

DESCRIPTION:

Returns a Bessel function of the second kind, of order zero, of the argument.

The domain is divided into the intervals [0, 5] and (5, infinity). In the first interval a rational approximation R(x) is employed to compute:

```
  y0(x)  = R(x)  +   2 * log(x) * j0(x) / PI.
```

Thus a call to j0() is required.

In the second interval, the Hankel asymptotic expansion is employed with two rational functions of degree 6/6 and 7/7.

ACCURACY:

Absolute error, when y0(x) < 1; else relative error:

arithmetic	domain	# trials	peak	rms
DEC	0, 30	9400	7.0e-17	7.9e-18
IEEE	0, 30	30000	1.3e-15	1.6e-16

y1

Bessel function of second kind of order one.

SYNOPSIS:

double x, y, y1();
y = y1(x);

DESCRIPTION:

Returns a Bessel function of the second kind of order one of the argument.

The domain is divided into the intervals [0, 8] and (8, infinity). In the first interval a 25 term Chebyshev expansion is used, and a call to j1() is required. In the second, the asymptotic trigonometric representation is employed using two rational functions of degree 5/5.

ACCURACY:

	Absolute error:			
arithmetic	domain	# trials	peak	rms
DEC	0, 30	10000	8.6e-17	1.3e-17
IEEE	0, 30	30000	1.0e-15	1.3e-16

(error criterion relative when |y1| > 1).

yn

Bessel function of second kind of integer order.

SYNOPSIS:

```
double x, y, yn();
int n;
y = yn(n, x);
```

DESCRIPTION:

Returns a Bessel function of order n, where n is a (possibly negative) integer.
The function is evaluated by forward recurrence on n, starting with values computed by the routines y0() and y1().
If n = 0 or 1 the routine for y0 or y1 is called directly.

ACCURACY:

		Absolute error, except relative when y > 1:		
arithmetic	domain	# trials	peak	rms
DEC	0, 30	2200	2.9e-16	5.3e-17
IEEE	0, 30	30000	3.4e-15	4.3e-16

ERROR MESSAGES:

message	condition	value returned
singularity	x = 0	MAXNUM
overflow		MAXNUM

Spot checked against tables for x, n between 0 and 100.

yv

Bessel function of the second kind, of non-integer order.

SYNOPSIS:

double v, x, y, yv();
y = yv(v, x);

DESCRIPTION:

Returns a Bessel function of the second kind, of order v of the argument, where v is a non-integer.

ACCURACY:

Not accurately characterized, but spot checked against tables.

i0

Modified Bessel function of order zero.

SYNOPSIS:

```
double x, y, i0();
y = i0(x);
```

DESCRIPTION:

Returns a modified Bessel function of order zero of the argument.

The function is defined as i0(x) = j0(ix).

The range is partitioned into the two intervals [0,8] and (8, infinity). Chebyshev polynomial expansions are employed in each interval.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	0,30	6000	8.2e-17	1.9e-17
IEEE	0,30	30000	5.8e-16	1.4e-16

i0e

Modified Bessel function of order zero, exponentially scaled.

SYNOPSIS:

```
double x, y, i0e();
y = i0e(x);
```

DESCRIPTION:

Returns exponentially scaled modified Bessel function of order zero of the argument.
The function is defined as i0e(x) = exp(-|x|) j0(ix).

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
IEEE	0,30	30000	5.4e-16	1.2e-16

i1

Modified Bessel function of order one.

SYNOPSIS:

double x, y, i1();
y = i1(x);

DESCRIPTION:

Returns the modified Bessel function of order one of the argument.

The function is defined as i1(x) = -i j1(ix).

The range is partitioned into the two intervals [0,8] and (8, infinity). Chebyshev polynomial expansions are employed in each interval.

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
DEC	0, 30	3400	1.2e-16	2.3e-17
IEEE	0, 30	30000	1.9e-15	2.1e-16

i1e

Modified Bessel function of order one, exponentially scaled.

SYNOPSIS:

```
double x, y, i1e();
y = i1e(x);
```

DESCRIPTION:

Returns the exponentially scaled modified Bessel function of order one of the argument.
The function is defined as i1(x) = -i exp(-|x|) j1(ix).

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
IEEE	0, 30	30000	2.0e-15	2.0e-16

iv

Modified Bessel function of noninteger order.

SYNOPSIS:

double v, x, y, iv();
y = iv(v, x);

DESCRIPTION:

Returns the modified Bessel function of order v of the argument. If x is negative, v must be integer-valued.

The function is defined as Iv(x) = Jv(ix). Here, it is computed in terms of the confluent hypergeometric function, according to the formula:

$$Iv(x) = (x/2)^v e^{-x} \, hyperg(v+0.5, 2v+1, 2x) / gamma(v+1)$$

If v is a negative integer, then v is replaced by -v.

ACCURACY:

Tested at random points (v, x), with v between 0 and 30, x between 0 and 28.

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	0,30	2000	3.1e-15	5.4e-16
IEEE	0,30	10000	1.7e-14	2.7e-15

Accuracy is diminished if v is near a negative integer.

k0

Modified Bessel function, third kind, order zero.

SYNOPSIS:

double x, y, k0();
y = k0(x);

DESCRIPTION:

Returns the modified Bessel function of the third kind of order zero of the argument.

The range is partitioned into the two intervals [0,8] and (8, infinity). Chebyshev polynomial expansions are employed in each interval.

ACCURACY:

Tested at 2000 random points between 0 and 8. Peak absolute error (relative when K0 > 1) was 1.46e-14; rms, 4.26e-15.

	Relative error:			
arithmetic	domain	# trials	peak	rms
DEC	0, 30	3100	1.3e-16	2.1e-17
IEEE	0, 30	30000	1.2e-15	1.6e-16

ERROR MESSAGES:

message	condition	value returned
domain	x <= 0	MAXNUM

k0e

Modified Bessel function, third kind, order zero, exponentially scaled.

SYNOPSIS:

double x, y, k0e();
y = k0e(x);

DESCRIPTION:

Returns the exponentially scaled, modified Bessel function of the third kind of order zero of the argument.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
IEEE	0, 30	30000	1.4e-15	1.4e-16

k1

Modified Bessel function, third kind, order one.

SYNOPSIS:

double x, y, k1();
y = k1(x);

DESCRIPTION:

Computes the modified Bessel function of the third kind, of order one of the argument.

The range is partitioned into the two intervals [0,2] and (2, infinity). Chebyshev polynomial expansions are employed in each interval.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	0, 30	3300	8.9e-17	2.2e-17
IEEE	0, 30	30000	1.2e-15	1.6e-16

ERROR MESSAGES:

message	condition	value returned
domain	x <= 0	MAXNUM

k1e

Modified Bessel function, third kind, order one, exponentially scaled.

SYNOPSIS:

```
double x, y, k1e();
y = k1e(x);
```

DESCRIPTION:

Returns the exponentially scaled, modified Bessel function of the third kind of order one of the argument:

```
k1e(x) = exp(x) * k1(x).
```

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
IEEE	0, 30	30000	7.8e-16	1.2e-16

kn

Modified Bessel function, third kind, integer order.

SYNOPSIS:

double x, y, kn();
int n;
y = kn(n, x);

DESCRIPTION:

Returns the modified Bessel function of the third kind, of order n of the argument.

The range is partitioned into the two intervals [0,9.55] and (9.55, infinity). An ascending power series is used in the low range, and an asymptotic expansion in the high range.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	0,30	3000	1.3e-9	5.8e-11
IEEE	0,30	90000	1.8e-8	3.0e-10

Error is high only near the crossover point x = 9.55 between the two expansions used.

Hypergeometric

- hyperg - confluent hypergeometric
- hyp2f1 - Gauss hypergeometric function
- hyp2f0 - 2F0
- onef2 - 1F2
- threef0 - 3F0

hyperg

Confluent hypergeometric function.

SYNOPSIS:

```
double a, b, x, y, hyperg();
y = hyperg(a, b, x);
```

DESCRIPTION:

Computes the confluent hypergeometric function

$$F(a,b;x) = 1 + \frac{a\,x}{b\,1!} + \frac{a(a+1)\,x^2}{b(b+1)\,2!} + \ldots$$

Many higher transcendental functions are special cases of this power series.

As is evident from the formula, b must not be a negative integer or zero unless a is an integer with 0 >= a > b.

The routine attempts both a direct summation of the series and an asymptotic expansion. In each case error due to roundoff, cancellation and nonconvergence is estimated. The result with smaller estimated error is returned.

ACCURACY:

Tested at random points (a, b, x), all three variables ranging from 0 to 30.

			Relative error:	
arithmetic	domain	# trials	peak	rms
DEC	0,30	2000	1.2e-15	1.3e-16

qtst1:

21800 max = 1.4200E-14 rms = 1.0841E-15 ave = -5.3640E-17

ltstd:

25500 max = 1.2759e-14 rms = 3.7155e-16 ave = 1.5384e-18

IEEE	0,30	30000	1.8e-14	1.1e-15

Larger errors can be observed when b is near a negative integer or zero. Certain combinations of arguments yield serious cancellation errors in the power series summation and also are not in the region of near convergence of the asymptotic series. An error message is printed if the self-estimated relative error is greater than 1.0e-12.

hyp2f1

Gauss hypergeometric function 2F1.

SYNOPSIS:

```
double a, b, c, x, y, hyp2f1();
y = hyp2f1(a, b, c, x);
```

DESCRIPTION:

```
 hyp2f1( a, b, c, x )  =   F ( a, b; c; x )
                         2 1


          inf.
           -   a(a+1)...(a+k) b(b+1)...(b+k)   k+1
  =  1 +   >   ----------------------------  x   .
           -         c(c+1)...(c+k) (k+1)!
         k = 0
```

Cases addressed are:

- Tests and escapes for negative integer a, b, or c
- Linear transformation if c - a or c - b negative integer
- Special case c = a or c = b
- Linear transformation for x near +1
- Transformation for x < -0.5
- Psi function expansion if x > 0.5 and c - a - b integer Conditionally, a recurrence on c to make c-a-b > 0

|x| > 1 is rejected.

The parameters a, b, c are considered to be integer valued if they are within 1.0e-14 of the nearest integer (1.0e-13 for IEEE arithmetic).

ACCURACY:

Relative error (-1 < x < 1):

arithmetic	domain	# trials	peak	rms
IEEE	-1,7	230000	1.2e-11	5.2e-14

Several special cases also tested with a, b, c in the range -7 to 7.

ERROR MESSAGES:

A "partial loss of precision" message is printed if the internally estimated relative error exceeds 1^-12.

A "singularity" message is printed on overflow or in cases not addressed (such as x < -1).

hyp2f0

See the hyperg Help topic.

onef2

See the struve Help topic.

threef0

See the struve Help topic.

Elliptic

- ellpe - complete elliptic integral (E)
- ellie - incomplete elliptic integral (E)
- ellpk - complete elliptic integral (K)
- ellik - incomplete elliptic integral (K)
- ellpj - Jacobian elliptic functions

ellpe

Complete elliptic integral of the second kind.

SYNOPSIS:

```
double m1, y, ellpe();
y = ellpe(m1);
```

DESCRIPTION:

Approximates the integral

```
           pi/2
            -
           | |              2
E(m)  =    |    sqrt( 1 - m sin t ) dt
         | |
         -
          0
```

Where m = 1 - m1, using the approximation:

P(x) - x log x Q(x).

Though there are no singularities, the argument m1 is used rather than m, for compatibility with ellpk().

E(1) = 1; E(0) = pi/2.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	0, 1	13000	3.1e-17	9.4e-18
IEEE	0, 1	10000	2.1e-16	7.3e-17

ERROR MESSAGES:

message	condition	value returned
domain	x<0, x>1	0.0

ellie

Incomplete elliptic integral of the second kind.

SYNOPSIS:

```
double phi, m, y, ellie();
y = ellie(phi, m);
```

DESCRIPTION:

Approximates the integral:

```
                phi
                 -
                | |
                |                2
E(phi_\m)  =    |    sqrt(1 - m sin t) dt
                |
              | |
              -
               0
```

of amplitude phi and modulus m, using the arithmetic - geometric mean algorithm.

ACCURACY:

Tested at random arguments with phi in [-10, 10] and m in [0, 1].

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	0,2	2000	1.9e-16	3.4e-17
IEEE	-10,10	150000	3.3e-15	1.4e-16

ellpk

Complete elliptic integral of the first kind.

SYNOPSIS:

```
double m1, y, ellpk();
y = ellpk(m1);
```

DESCRIPTION:

Approximates the integral:

```
          pi/2
           -
          | |
          |          dt
K(m)  =   |    ------------------
          |                2
        | |    sqrt(1 - m sin t)
          -
          0
```

where m = 1 - m1, using the approximation:

P(x) - log x Q(x).

The argument m1 is used rather than m, so that the logarithmic singularity at m = 1 will be shifted to the origin; this preserves maximum accuracy.

K(0) = pi/2.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	0,1	16000	3.5e-17	1.1e-17
IEEE	0,1	30000	2.5e-16	6.8e-17

ERROR MESSAGES:

message	condition	value returned

domain	x<0, x>1	0.0

ellik

Incomplete elliptic integral of the first kind.

SYNOPSIS:

```
double phi, m, y, ellik();
y = ellik(phi, m);
```

DESCRIPTION:

Approximates the integral:

```
                phi
                 -
                | |
                |          dt
F(phi_\m)  =    |    ------------------
                |                2
               | |    sqrt(1 - m sin t)
                -
                0
```

of amplitude phi and modulus m, using the arithmetic - geometric mean algorithm.

ACCURACY:

Tested at random points with m in [0, 1] and phi as indicated.

```
                    Relative error:
arithmetic   domain     # trials      peak         rms
```

ellpj

Jacobian Elliptic Functions.

SYNOPSIS:

double u, m, sn, cn, dn, phi;

int ellpj();

ellpj(u, m, _&sn, _&cn, _&dn, _&phi);

DESCRIPTION:

Evaluates the Jacobian elliptic functions sn(u|m), cn(u|m), and dn(u|m) of parameter m between 0 and 1, and real argument u.

These functions are periodic, with quarter-period on the real axis equal to the complete elliptic integral ellpk(1.0-m).

Relation to incomplete elliptic integral:

If u = ellik(phi,m), then sn(u|m) = sin(phi), and cn(u|m) = cos(phi).

Phi is called the amplitude of u.

Computation is by means of the arithmetic-geometric mean algorithm, except when m is within 1e-9 of 0 or 1.

In the latter case with m close to 1, the approximation applies only for phi < pi/2.

ACCURACY:

Tested at random points with u between 0 and 10, m between 0 and 1.

Absolute error (* = relative error):

arithmetic	function	# trials	peak	rms
DEC	sn	1800	4.5e-16	8.7e-17
IEEE	phi	10000	9.2e-16*	1.4e-16*
IEEE	sn	50000	4.1e-15	4.6e-16
IEEE	cn	40000	3.6e-15	4.4e-16
IEEE	dn	100000	3.9e-15	1.7e-16

Larger errors occur for m near 1.

Peak error observed in consistency check using addition theorem for sn(u+v) was 4e-16 (absolute). Also tested by the earlier relation to the incomplete elliptic integral. Accuracy deteriorates when u is large.

Probability

- bdtr - Binomial distribution
- bdtrc - Complemented binomial
- bdtri - Inverse binomial
- chdtr - Chi square distribution
- chdtrc - Complemented Chi square
- chdtri - Inverse Chi square
- fdtr - F distribution
- fdtrc - Complemented F
- fdtri - Inverse F distribution
- gdtr - Gamma distribution
- gdtrc - Complemented gamma
- nbdtr - Negative binomial distribution
- nbdtrc - Complemented negative binomial
- ndtr - Normal distribution
- ndtri - Inverse normal distribution
- pdtr - Poisson distribution
- pdtrc - Complemented Poisson
- pdtri - Inverse Poisson distribution
- stdtr - Student's t distribution

bdtr

Binomial distribution.

SYNOPSIS:

```
int k, n;
double p, y, bdtr();
y = bdtr(k, n, p);
```

DESCRIPTION:

Returns the sum of the terms 0 through k of the Binomial probability density:

```
  k
  --  ( n )   j      n-j
  >   (   )  p  (1-p)
  --  ( j )
 j=0
```

The terms are not summed directly; instead the incomplete beta integral is employed, according to the formula:

y = bdtr(k, n, p) = incbet(n-k, k+1, 1-p).

The arguments must be positive, with p ranging from 0 to 1.

ACCURACY:

Tested at random points (a,b,p), with p between 0 and 1.

```
             a,b                  Relative error:
arithmetic  domain     # trials      peak         rms
  For p between 0.001 and 1:
    IEEE     0,100       100000      4.3e-15     2.6e-16
```

ERROR MESSAGES:

```
  message      condition      value returned
  domain         k < 0            0.0
                 n < k
                 x < 0, x > 1
```

bdtrc

Complemented binomial distribution.

SYNOPSIS:

```
int k, n;
double p, y, bdtrc();
y = bdtrc(k, n, p);
```

DESCRIPTION:

Returns the sum of the terms k+1 through n of the Binomial
probability density:

$$\sum_{j=k+1}^{n} \binom{n}{j} p^{j} (1-p)^{n-j}$$

The terms are not summed directly; instead the incomplete beta integral is employed, according to the formula:

y = bdtrc(k, n, p) = incbet(k+1, n-k, p).

The arguments must be positive, with p ranging from 0 to 1.

ACCURACY:

Tested at random points (a,b,p).

	a,b		Relative error:	
arithmetic	domain	# trials	peak	rms
For p between 0.001 and 1:				
IEEE	0,100	100000	6.7e-15	8.2e-16
For p between 0 and .001:				
IEEE	0,100	100000	1.5e-13	2.7e-15

ERROR MESSAGES:

message	condition	value returned

domain	x<0, x>1, n<k	0.0

bdtri

Inverse binomial distribution.

SYNOPSIS:

int k, n;
double p, y, bdtri();
p = bdtr(k, n, y);

DESCRIPTION:

Finds the event probability p such that the sum of the terms 0 through k of the Binomial probability density is equal to the given cumulative probability y.
This is accomplished using the inverse beta integral function and the relation:

1 - p = incbi(n-k, k+1, y).

ACCURACY:

Tested at random points (a,b,p).

	a,b		Relative error:	
arithmetic	domain	# trials	peak	rms
For p between 0.001 and 1:				
IEEE	0,100	100000	2.3e-14	6.4e-16
IEEE	0,10000	100000	6.6e-12	1.2e-13
For p between 10^-6 and 0.001:				
IEEE	0,100	100000	2.0e-12	1.3e-14
IEEE	0,10000	100000	1.5e-12	3.2e-14

See the incbi Help topic.

ERROR MESSAGES:

message	condition	value returned
domain	k < 0, n <= k	0.0
	x < 0, x > 1	

chdtr

Chi-square distribution.

SYNOPSIS:

double df, x, y, chdtr();
y = chdtr(df, x);

DESCRIPTION:

Returns the area under the left hand tail (from 0 to x) of the Chi square probability density function, with v degrees of freedom.

$$P(x \mid v) = \frac{1}{2^{v/2}\,\Gamma(v/2)} \int_0^x t^{v/2-1}\, e^{-t/2}\, dt$$

where x is the Chi-square variable.

The incomplete gamma integral is used, according to the formula:

y = chdtr(v, x) = igam(v/2.0, x/2.0).

The arguments must both be positive.

ACCURACY:

See the igam() Help topic.

ERROR MESSAGES:

message	condition	value returned
domain	x < 0 or v < 1	0.0

chdtrc

Complemented Chi-square distribution.

SYNOPSIS:

double v, x, y, chdtrc();
y = chdtrc(v, x);

DESCRIPTION:

Returns the area under the right hand tail (from x to infinity) of the Chi square probability density function with v degrees of freedom:

$$P(x \mid v) = \frac{1}{2^{v/2}\,\Gamma(v/2)} \int_{x}^{\infty} t^{v/2-1}\, e^{-t/2}\, dt$$

where x is the Chi-square variable.

The incomplete gamma integral is used, according to the formula:

y = chdtr(v, x) = igamc(v/2.0, x/2.0).

The arguments must both be positive.

ACCURACY:

See the igamc() Help topic.

ERROR MESSAGES:

message	condition	value returned
domain	x < 0 or v < 1	0.0

chdtri

Inverse of complemented Chi-square distribution.

SYNOPSIS:

double df, x, y, chdtri();
x = chdtri(df, y);

DESCRIPTION:

Finds the Chi-square argument x, such that the integral from x to infinity of the Chi-square density is equal to the given cumulative probability y.

This is accomplished using the inverse gamma integral function and the relation:

x/2 = igami(df/2, y);

ACCURACY:

See the igami Help topic.

ERROR MESSAGES:

message	condition	value returned
domain	y < 0 or y > 1	0.0
	v < 1	

fdtr

F distribution.

SYNOPSIS:

int df1, df2;
double x, y, fdtr();
y = fdtr(df1, df2, x);

DESCRIPTION:

Returns the area from zero to x under the F density function (also known as Snedcor's density, or the variance ratio density).
This is the density of x = (u1/df1)/(u2/df2), where u1 and u2 are random variables having Chi square distributions with df1 and df2 degrees of freedom, respectively.

The incomplete beta integral is used, according to the formula

P(x) = incbet(df1/2, df2/2, (df1*x/(df2 + df1*x)).

The arguments a and b are greater than zero, and x is nonnegative.

ACCURACY:

Tested at random points (a,b,x).

	x	a,b		Relative error:	
arithmetic	domain	domain	# trials	peak	rms
IEEE	0,1	0,100	100000	9.8e-15	1.7e-15
IEEE	1,5	0,100	100000	6.5e-15	3.5e-16
IEEE	0,1	1,10000	100000	2.2e-11	3.3e-12
IEEE	1,5	1,10000	100000	1.1e-11	1.7e-13

See the incbet Help topic.

ERROR MESSAGES:

message	condition	value returned
domain	a<0, b<0, x<0	0.0

fdtrc

Complemented F distribution.

SYNOPSIS:

```
int df1, df2;
double x, y, fdtrc();
y = fdtrc(df1, df2, x);
```

DESCRIPTION:

Returns the area from x to infinity under the F density function (also known as Snedcor's density or the variance ratio density).

$$1-P(x) = \frac{1}{B(a,b)} \int_{x}^{\infty} t^{a-1} (1-t)^{b-1} \, dt$$

The incomplete beta integral is used, according to the formula

P(x) = incbet(df2/2, df1/2, (df2/(df2 + df1*x)).

ACCURACY:

Tested at random points (a,b,x) in the indicated intervals.

	x	a,b		Relative error:	
arithmetic	domain	domain	# trials	peak	rms
IEEE	0,1	1,100	100000	3.7e-14	5.9e-16
IEEE	1,5	1,100	100000	8.0e-15	1.6e-15
IEEE	0,1	1,10000	100000	1.8e-11	3.5e-13
IEEE	1,5	1,10000	100000	2.0e-11	3.0e-12

ERROR MESSAGES:

message	condition	value returned
domain	a<0, b<0, x<0	0.0

fdtri

Inverse of complemented F distribution.

SYNOPSIS:

int df1, df2;
double x, p, fdtri();
x = fdtri(df1, df2, p);

DESCRIPTION:

Finds the F density argument x, such that the integral from x to infinity of the F density is equal to the given probability p.
This is accomplished using the inverse beta integral function and the relations:

z = incbi(df2/2, df1/2, p)
x = df2 (1-z) / (df1 z).

Note: These relations hold for the inverse of the uncomplemented F distribution:

z = incbi(df1/2, df2/2, p)
x = df2 z / (df1 (1-z)).

ACCURACY:

Tested at random points (a,b,p).

a,b		Relative error:		
arithmetic	domain	# trials	peak	rms
For p between .001 and 1:				
IEEE	1,100	100000	8.3e-15	4.7e-16
IEEE	1,10000	100000	2.1e-11	1.4e-13
For p between 10^-6 and 10^-3:				
IEEE	1,100	50000	1.3e-12	8.4e-15
IEEE	1,10000	50000	3.0e-12	4.8e-14

See the fdtrc Help topic.

ERROR MESSAGES:

message	condition	value returned

domain	p <= 0 or p > 1 v < 1	0.0

gdtr

Gamma distribution function.

SYNOPSIS:

```
double a, b, x, y, gdtr();
y = gdtr(a, b, x);
```

DESCRIPTION:

Returns the integral from zero to x of the gamma probability density function:

```
                x
       b        -
      a         | |   b-1  -at
y =  -----      |     t    e    dt
      -     | |
     | (b)    -
             0
```

The incomplete gamma integral is used, according to the relation:

y = igam(b, ax).

ACCURACY:

See the igam() Help topic.

ERROR MESSAGES:

message	condition	value returned
domain	x < 0	0.0

gdtrc

Complemented gamma distribution function.

SYNOPSIS:

```
double a, b, x, y, gdtrc();
y = gdtrc(a, b, x);
```

DESCRIPTION:

Returns the integral from x to infinity of the gamma probability density function:

```
              inf.
       b       -
      a       | |   b-1  -at
y =  -----    |    t    e    dt
      -     | |
     | (b)   -
              x
```

The incomplete gamma integral is used, according to the relation:

y = igamc(b, ax).

ACCURACY:

See the igamc() Help topic.

ERROR MESSAGES:

message	condition	value returned
domain	x < 0	0.0

nbdtr

Negative binomial distribution.

SYNOPSIS:

```
int k, n;
double p, y, nbdtr();
y = nbdtr(k, n, p);
```

DESCRIPTION:

Returns the sum of the terms 0 through k of the negative binomial distribution:

```
  k
  --  (n+j-1)   n      j
  >   (     )  p  (1-p)
  --  (  j  )
 j=0
```

In a sequence of Bernoulli trials, this is the probability that k or fewer failures precede the nth success.

The terms are not computed individually; instead the incomplete beta integral is employed, according to the formula:

y = nbdtr(k, n, p) = incbet(n, k+1, p).

The arguments must be positive, with p ranging from 0 to 1.

ACCURACY:

Tested at random points (a,b,p), with p between 0 and 1.

	a,b		Relative error:	
arithmetic	domain	# trials	peak	rms
IEEE	0,100	100000	1.7e-13	8.8e-15

See the incbet Help topic.

nbdtrc

Complemented negative binomial distribution.

SYNOPSIS:

```
int k, n;
double p, y, nbdtrc();
y = nbdtrc(k, n, p);
```

DESCRIPTION:

Returns the sum of the terms k+1 to infinity of the negative binomial distribution:

$$\sum_{j=k+1}^{\infty} \binom{n+j-1}{j} p^{n} (1-p)^{j}$$

The terms are not computed individually; instead the incomplete beta integral is employed, according to the formula:

y = nbdtrc(k, n, p) = incbet(k+1, n, 1-p).

The arguments must be positive, with p ranging from 0 to 1.

ACCURACY:

Tested at random points (a,b,p), with p between 0 and 1.

	a,b		Relative error:	
arithmetic	domain	# trials	peak	rms
IEEE	0,100	100000	1.7e-13	8.8e-15

See the incbet Help topic.

ndtr

Normal distribution function.

SYNOPSIS:

```
double x, y, ndtr();
y = ndtr(x);
```

DESCRIPTION:

Returns the area under the Gaussian probability density function, integrated from minus infinity to x:

```
                    x
                    -
          1        | |        2
  ndtr(x)  = ---------    |    exp(- t /2) dt
        sqrt(2pi)  | |
                    -
                   -inf.

        =  (1 + erf(z)) / 2
        =  erfc(z) / 2
```

where z = x/sqrt(2).

Computation is via the functions erf and erfc, with care to avoid error amplification in computing exp(-x^2).

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
IEEE	-13,0	30000	1.3e-15	2.2e-16

ERROR MESSAGES:

message	condition	value returned
underflow	x > 37.519379347	0.0

ndtri

Inverse of Normal distribution function.

SYNOPSIS:

double x, y, ndtri();
x = ndtri(y);

DESCRIPTION:

Returns the argument, x, for which the area under the Gaussian probability density function (integrated from minus infinity to x) is equal to y.

For small arguments 0 < y < exp(-2), the program computes z = sqrt(-2.0 * log(y)); then the approximation is x = z - log(z)/z - (1/z) P(1/z) / Q(1/z).

There are two rational functions P/Q, one for 0 < y < exp(-32) and the other for y up to exp(-2).

For larger arguments, w = y - 0.5, and x/sqrt(2pi) = w + w**3 R(w**2)/S(w**2)).

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	0.125, 1	5500	9.5e-17	2.1e-17
DEC	6e-39, 0.135	3500	5.7e-17	1.3e-17
IEEE	0.125, 1	20000	7.2e-16	1.3e-16
IEEE	3e-308, 0.135	50000	4.6e-16	9.8e-17

ERROR MESSAGES:

message	condition	value returned
domain	x <= 0	-MAXNUM
domain	x >= 1	MAXNUM

pdtr

Poisson distribution.

SYNOPSIS:

```
int k;
double m, y, pdtr();
y = pdtr(k, m);
```

DESCRIPTION:

Returns the sum of the first k terms of the Poisson distribution:

$$\sum_{j=0}^{k} \frac{e^{-m} m^j}{j!}$$

The terms are not summed directly; instead the incomplete gamma integral is employed, according to the relation:

y = pdtr(k, m) = igamc(k+1, m).

The arguments must both be positive.

ACCURACY:
See the igamc Help topic.

pdtrc

Complemented poisson distribution.

SYNOPSIS:

```
int k;
double m, y, pdtrc();
y = pdtrc(k, m);
```

DESCRIPTION:

Returns the sum of the terms k+1 to infinity of the Poisson distribution:

$$\sum_{j=k+1}^{\infty} \frac{e^{-m} m^j}{j!}$$

The terms are not summed directly; instead the incomplete gamma integral is employed, according to the formula:

y = pdtrc(k, m) = igam(k+1, m).

The arguments must both be positive.

ACCURACY:
See the igam Help topic.

pdtri

Inverse Poisson distribution.

SYNOPSIS:

```
int k;
double m, y, pdtr();
m = pdtri(k, y);
```

DESCRIPTION:

Finds the Poisson variable x such that the integral from 0 to x of the Poisson density is equal to the given probability y. This is accomplished using the inverse gamma integral function and the relation

```
m = igami(k+1, y).
```

ACCURACY:

See the igami Help topic.

ERROR MESSAGES:

message	condition	value returned
domain	y < 0 or y >= 1	0.0
	k < 0	

stdtr

Student's t distribution.

SYNOPSIS:

```
double t, stdtr();
short k;
y = stdtr(k, t);
```

DESCRIPTION:

Computes the integral from minus infinity to t of the Student t distribution with integer k > 0 degrees of freedom:

$$\frac{\Gamma((k+1)/2)}{\sqrt{k\pi}\,\Gamma(k/2)} \int_{-\infty}^{t} \left(1 + \frac{x}{k}\right)^{-(k+1)/2} dx$$

Relation to incomplete beta integral:

$$1 - \mathrm{stdtr}(k,t) = 0.5 * \mathrm{incbet}(k/2, 1/2, z)$$

where

$$z = k/(k + t^{**}2).$$

For t < -2, this is the method of computation.

For higher t, a direct method is derived from integration by parts.

Since the function is symmetric about t=0, the area under the right tail of the density is found by calling the function with -t instead of t.

ACCURACY:

Tested at random 1 <= k <= 25. The 'domain' refers to t.

Relative error:

arithmetic	domain	# trials	peak	rms
IEEE	-100,-2	50000	5.9e-15	1.4e-15
IEEE	-2,100	500000	2.7e-15	4.9e-17

Miscellaneous

- polylog - Polylogarithms
- spence - Dilogarithm
- zetac - Riemann Zeta function
- zeta - Two-argument zeta function
- struve - Struve function

polylog

Polylogarithms.

SYNOPSIS:

```
double x, y, polylog();
int n;
y = polylog(n, x);
```

The polylogarithm of order n is defined by the series:

$$Li_n(x) = \sum_{k=1}^{\infty} \frac{x^k}{k^n}.$$

For x = 1,

$$Li_n(1) = \sum_{k=1}^{\infty} \frac{1}{k^n} = \text{Riemann zeta function (n)}.$$

When n = 2, the function is the dilogarithm, related to Spence's integral:

$$Li_2(x) = \int_0^x \frac{-\ln(1-t)}{t}\,dt = \int_1^{1-x} \frac{\ln t}{1-t}\,dt = \text{spence}(1-x).$$

References:

Lewin, L., *Polylogarithms and Associated Functions*,
North Holland, 1981.

Lewin, L., ed., *Structural Properties of Polylogarithms*,
American Mathematical Society, 1991.

ACCURACY:

Relative error:

arithmetic	domain	n	# trials	peak	rms
IEEE	0, 1	2	50000	6.2e-16	8.0e-17
IEEE	0, 1	3	100000	2.5e-16	6.6e-17
IEEE	0, 1	4	30000	1.7e-16	4.9e-17
IEEE	0, 1	5	30000	5.1e-16	7.8e-17

spence

Dilogarithm.

SYNOPSIS:

```
double x, y, spence();
y = spence(x);
```

DESCRIPTION:

Computes the integral:

```
                  x
                  -
                 | | log t
spence(x)  =  -  |   ----- dt
                | |   t - 1
                 -
                 1
```

for x >= 0. A rational approximation gives the integral in the interval (0.5, 1.5). Transformation formulas for 1/x and 1-x are employed outside the basic expansion range.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
IEEE	0,4	30000	3.9e-15	5.4e-16
DEC	0,4	3000	2.5e-16	4.5e-17

zetac

Riemann zeta function.

SYNOPSIS:
double x, y, zetac();
y = zetac(x);

DESCRIPTION:

$$\text{zetac}(x) = \sum_{k=2}^{\infty} k^{-x}, \quad x > 1,$$

Is related to the Riemann zeta function by:

Riemann zeta(x) = zetac(x) + 1.

Extension of the function definition for x < 1 is implemented.
Zero is returned for x > log2(MAXNUM).

An overflow error might occur for large negative x, due to the gamma function in the reflection formula.

ACCURACY:
Tabulated values have full machine accuracy.

		Relative error:		
arithmetic	domain	# trials	peak	rms
IEEE	1,50	10000	9.8e-16	1.3e-16
DEC	1,50	2000	1.1e-16	1.9e-17

zeta

Riemann zeta function of two arguments.

SYNOPSIS:

```
double x, q, y, zeta();
y = zeta(x, q);
```

DESCRIPTION:

$$\zeta(x,q) = \sum_{k=0}^{\infty} (k+q)^{-x}$$

where x > 1 and q is not a negative integer or zero.
The Euler-Maclaurin summation formula is used to obtain the expansion

$$\zeta(x,q) = \sum_{k=1}^{n} (k+q)^{-x}$$

$$+ \frac{(n+q)^{1-x}}{x-1} - \frac{1}{2(n+q)^{x}} + \sum_{j=1}^{\infty} \frac{B_{2j}\, x(x+1)\cdots(x+2j)}{(2j)!\,(n+q)^{x+2j+1}}$$

where the B2j are Bernoulli numbers.
Note that zeta(x,1) = zetac(x) + 1.
(see zetac)

ACCURACY:

REFERENCE:

Gradshteyn, I. S., and I. M. Ryzhik, *Tables of Integrals, Series, and Products*, p. 1073; Academic Press, 1980.

struve

Struve function.

SYNOPSIS:

double v, x, y, struve();
y = struve(v, x);

DESCRIPTION:

Computes the Struve function Hv(x) of order v, argument x. Negative x is rejected unless v is an integer.

This module also contains the hypergeometric functions 1F2 and 3F0, and a routine for the Bessel function Yv(x) with noninteger v.

ACCURACY:

Not accurately characterized, but spot checked against tables.

Matrix

- fftr - Fast Fourier transform
- simq - Simultaneous linear equations
- minv - Matrix inversion
- mmmpy - Matrix multiply
- mvmpy - Matrix times vector
- mtransp - Matrix transpose
- eigens - Eigenvectors (symmetric matrix)

fftr

FFT of Real Valued Sequence.

SYNOPSIS:

double x[], sine[];

int m;

fftr(x, m, sine);

DESCRIPTION:

Computes the (complex valued) discrete Fourier transform of the real valued sequence x[]. The input sequence x[] contains n = 2**m samples. The program fills array sine[k] with n/4 + 1 values of sin(2 PI k / n).

Data format for complex valued output is real part followed by imaginary part. The output is developed in the input array x[].

The algorithm takes advantage of the fact that the FFT of an n point real sequence can be obtained from an n/2 point complex FFT.

A radix 2 FFT algorithm is used.

Execution time on an LSI-11/23 with floating point chip is 1.0 sec for n = 256.

REFERENCE:

E. Oran Brigham, *The Fast Fourier Transform*; Prentice-Hall, Inc., 1974

simq

Solution of simultaneous linear equations AX = B by Gaussian elimination with partial pivoting.

SYNOPSIS:

double A[n*n], B[n], X[n];

int n, flag;

int IPS[];

int simq();

ercode = simq(A, B, X, n, flag, IPS);

DESCRIPTION:

B, X, IPS are vectors of length n.

A is an n x n matrix (i.e. a vector of length n*n), stored row-wise; that is, A(i,j) = A[ij], where ij = i*n + j, which is the transpose of the normal column-wise storage.

The contents of matrix A are destroyed.

Set flag=0 to solve.

Set flag=-1 to do a new back substitution for a different B vector using the same A matrix previously reduced when flag=0.

The routine returns nonzero on error; messages are printed.

ACCURACY:

Depends on the conditioning (range of eigenvalues) of matrix A.

REFERENCE:

Computer Solution of Linear Algebraic Systems

by George E. Forsythe and Cleve B. Moler; Prentice-Hall, 1967.

minv

Matrix inversion.

SYNOPSIS:

int n, errcod;
double A[n*n], X[n*n];
double B[n];
int IPS[n];
int minv();

errcod = minv(A, X, n, B, IPS);

DESCRIPTION:

Finds the inverse of the n by n matrix A. The result goes to X. B and IPS are scratch-pad arrays of length n. The contents of matrix A are destroyed.

The routine returns nonzero on error; error messages are printed by the subroutine simq().

JavaScript:

```
function test_minv()
{
/*
* Finds the inverse of the n by n matrix A.  The result goes
* to X.   B and IPS are scratch pad arrays of length n.
* The contents of matrix A are destroyed
*/
        Session.Output("calling cephes.minv( A,X,n,B,IPS) where:");
        var n = 10; // n x n matrix A (10x10)
        var A = [
                        [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],
                        [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],
                        [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],
                        [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],
                        [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],
                        [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],
                        [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],
                        [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],
                        [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],
```

```
                                [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
        ];

        var X = new Array(10);     // output
        var B = new Array(10);      // scratch pad
        var IPS = new Array(10); // scratch pad

        Session.Output("  n = " + n);
        Session.Output("  length of A is" + n*n);
        Session.Output("A is matrix of " + dimensionsOfArray(A));
        var ir = cephes.minv(A,X,n,B,IPS);

        var s = cephes.geterrormsg();
        if(s.length>0)
        {
                Session.Output("error output by minv: " + s);
        }
        else
        {
                Session.Output("minv returned " + ir);
                Session.Output("X is matrix of " + dimensionsOfArray(X));
                printMatrix("X",X,10,10);
        }
}
```

mmmpy

Matrix-Matrix multiply

SYNOPSIS

```
int r, c;
double A[r*c], B[c*r], Y[r*r];
mmmpy( r, c, A, B, Y );
```

DESCRIPTION

Multiply an r (rows) by c (columns) matrix A on the left by a c (rows) by r (columns) matrix B on the right to produce an r by r matrix Y.

mvmpy

Matrix-Vector multiply

SYNOPSIS

```
int r, c;
double A[r*c], V[c], Y[r];
mvmpy( r, c, A, V, Y );
```

DESCRIPTION

Multiply r (rows) by c (columns) matrix A on the left by column vector V of dimension c on the right to produce a (column) vector Y output of dimension r.

mtransp

Matrix Transpose

SYNOPSIS

```
int n;
double A[n*n], T[n*n];
mtransp( n, A, T )
```

DESCRIPTION

Transpose the n by n square matrix A and put the result in T.
T can occupy the same storage as A.

eigens

Eigenvalues and eigenvectors of a real symmetric matrix.

SYNOPSIS:

int n;
double A[n*(n+1)/2], EV[n*n], E[n];
void eigens(A, EV, E, n);

DESCRIPTION:

The algorithm is due to J. vonNeumann.

- -

A[] is a symmetric matrix stored in lower triangular form. That is, A[row, column] = A[(row*row+row)/2 + column] or the equivalent with row and column interchanged. The indices row and column run from 0 through n-1.

EV[] is the output matrix of eigenvectors stored columnwise. That is, the elements of each eigenvector appear in sequential memory order. The jth element of the ith eigenvector is EV[n*i+j] = EV[i][j].

E[] is the output matrix of eigenvalues. The ith element of E corresponds to the ith eigenvector (the ith row of EV).

On output, the matrix A will have been diagonalized and its original contents are destroyed.

ACCURACY:

The error is controlled by an internal parameter called RANGE which is set to 1e-10. After diagonalization, the off-diagonal elements of A will have been reduced by this factor.

ERROR MESSAGES:

None.

JavaScript:

```
function test_eigens()
{
        var A = [
                        [0.1,0.2,0.3,0.4],
                        [0.5,0.6,0.7,0.8],
                        [0.9,0.8,0.7,0.6],
                        [0.5,0.4,0.3,0.2]
```

```
        ];
        var EV = new Array();
        var E = new Array();
        var N = 4;
        Session.Output("calling cephes.eigens( A, EV, E, N) where:");
        Session.Output(" A is NxN input matrix and N = " + N);
        printMatrix("A",A,N,N);
        cephes.eigens(A, EV, E, N);
        Session.Output(" EV is matrix of " + dimensionsOfArray(EV));
        printMatrix("Y",EV,N,N);
        Session.Output(" ");
        Session.Output(" E is matrix of " + dimensionsOfArray(E));
        printMatrix("Y",E,N,N);
        Session.Output(" ");
}


function printMatrix(name, M, rows, cols)
{
        for(var r = 0; r < rows; r++)
        {
                for(var c = 0; c < cols; c++)
                {
                        Session.Output(name + "[" + r + "][" + c + "] = " +M[r][c]);
                }
        }
}

var str="";

function dimensionsOfArrayX(v)
{
        str += v.length;
        if(v.length)
        {
                var e = v[0];
                if(Array.isArray(e))
                {
                        str += " x ";
                        dimensionsOfArrayX(e);
                }
        }
```

```
}
function dimensionsOfArray(v)
{
        str = "";
        dimensionsOfArrayX(v);
        return str;
}
```

Numerical Integration

- simpsn - Simpson's rule

simpsn

Simpson Numerical Integration

SYNOPSIS

```
double simpsn( f, delta )
double f[];          /* tabulated function */
double delta;        /* spacing of arguments */
double simpsn( f, delta );
```

DESCRIPTION

Numerical integration of function tabulated at equally spaced arguments
Uses 8th order Cote integration formula.

Complex Arithmetic

- cadd - Complex addition
- csub - Subtraction
- cmul - Multiplication
- cdiv - Division
- cabs - Absolute value
- csqrt - Square root

cadd

Addition.

SYNOPSIS:

```
typedef struct {
    double r;    real part
    double i;    imaginary part
   }cmplx;

cmplx *a, *b, *c;

cadd(a, b, c);    c = b + a
```

DESCRIPTION:

```
  c.r  =  b.r + a.r
  c.i  =  b.i + a.i
```

ACCURACY:

In DEC arithmetic, the test (1/z) * z = 1 had peak relative error 3.1e-17, rms 1.2e-17. The test (y/z) * (z/y) = 1 had peak relative error 8.3e-17, rms 2.1e-17.

Tests in the rectangle {-10,+10}:

			Relative error:	
arithmetic	function	# trials	peak	rms
DEC	cadd	10000	1.4e-17	3.4e-18
IEEE	cadd	100000	1.1e-16	2.7e-17
DEC	csub	10000	1.4e-17	4.5e-18
IEEE	csub	100000	1.1e-16	3.4e-17
DEC	cmul	3000	2.3e-17	8.7e-18
IEEE	cmul	100000	2.1e-16	6.9e-17
DEC	cdiv	18000	4.9e-17	1.3e-17
IEEE	cdiv	100000	3.7e-16	1.1e-16

JavaScript:

```
var a = {"r":0.5,"i",0.5};
```

```
var b = {"r":0.5,"i",0.5};
var c = cephes.cadd(a,b);
Session.Output("c.r=" + c.r + ", c.i=" + c.i);
```

csub

Subtraction.

SYNOPSIS:

```
typedef struct {
    double r;    real part
    double i;    imaginary part
   }cmplx;

cmplx *a, *b, *c;
csub(a, b, c);    c = b - a
```

DESCRIPTION:

```
  c.r  =  b.r - a.r
  c.i  =  b.i - a.i
```

ACCURACY:

In DEC arithmetic, the test (1/z) * z = 1 had peak relative error 3.1e-17, rms 1.2e-17. The test (y/z) * (z/y) = 1 had peak relative error 8.3e-17, rms 2.1e-17.

Tests in the rectangle {-10,+10}:

			Relative error:	
arithmetic	function	# trials	peak	rms
DEC	cadd	10000	1.4e-17	3.4e-18
IEEE	cadd	100000	1.1e-16	2.7e-17
DEC	csub	10000	1.4e-17	4.5e-18
IEEE	csub	100000	1.1e-16	3.4e-17
DEC	cmul	3000	2.3e-17	8.7e-18
IEEE	cmul	100000	2.1e-16	6.9e-17
DEC	cdiv	18000	4.9e-17	1.3e-17
IEEE	cdiv	100000	3.7e-16	1.1e-16

JavaScript:

```
var a = {"r":0.5,"i",0.5};
var b = {"r":0.5,"i",0.5};
var c = cephes.csub(a,b);
```

```
Session.Output("c.r=" + c.r + ", c.i=" + c.i);
```

cmul

Multiplication.

SYNOPSIS:

```
typedef struct {
    double r;    real part
    double i;    imaginary part
   }cmplx;

cmplx *a, *b, *c;

cmul(a, b, c);    c = b * a
```

DESCRIPTION:

```
  c.r  =  b.r * a.r  -  b.i * a.i
  c.i  =  b.r * a.i  +  b.i * a.r
```

ACCURACY:

In DEC arithmetic, the test (1/z) * z = 1 had peak relative error 3.1e-17, rms 1.2e-17. The test (y/z) * (z/y) = 1 had peak relative error 8.3e-17, rms 2.1e-17.

Tests in the rectangle {-10,+10}:

			Relative error:	
arithmetic	function	# trials	peak	rms
DEC	cadd	10000	1.4e-17	3.4e-18
IEEE	cadd	100000	1.1e-16	2.7e-17
DEC	csub	10000	1.4e-17	4.5e-18
IEEE	csub	100000	1.1e-16	3.4e-17
DEC	cmul	3000	2.3e-17	8.7e-18
IEEE	cmul	100000	2.1e-16	6.9e-17
DEC	cdiv	18000	4.9e-17	1.3e-17
IEEE	cdiv	100000	3.7e-16	1.1e-16

JavaScript:

```
var a = {"r":0.5,"i",0.5};
var b = {"r":0.5,"i",0.5};
var c = cephes.cmul(a,b);
```

```
Session.Output("c.r=" + c.r + ", c.i=" + c.i);
```

cdiv

Division.

SYNOPSIS:

```
typedef struct {
     double r;    real part
     double i;    imaginary part
    }cmplx;

cmplx *a, *b, *c;

cdiv(a, b, c);     c = b / a
```

DESCRIPTION:

```
   d   = a.r * a.r  +  a.i * a.i
   c.r  = (b.r * a.r  + b.i * a.i)/d
   c.i  = (b.i * a.r  -  b.r * a.i)/d
```

ACCURACY:

In DEC arithmetic, the test (1/z) * z = 1 had peak relative error 3.1e-17, rms 1.2e-17. The test (y/z) * (z/y) = 1 had peak relative error 8.3e-17, rms 2.1e-17.

Tests in the rectangle {-10,+10}:

			Relative error:	
arithmetic	function	# trials	peak	rms
DEC	cadd	10000	1.4e-17	3.4e-18
IEEE	cadd	100000	1.1e-16	2.7e-17
DEC	csub	10000	1.4e-17	4.5e-18
IEEE	csub	100000	1.1e-16	3.4e-17
DEC	cmul	3000	2.3e-17	8.7e-18
IEEE	cmul	100000	2.1e-16	6.9e-17
DEC	cdiv	18000	4.9e-17	1.3e-17
IEEE	cdiv	100000	3.7e-16	1.1e-16

JavaScript:

```
var a = {"r":0.5,"i",0.5};
```

```
var b = {"r":0.5,"i",0.5};
var c = cephes.cdiv(a,b);
Session.Output("c.r=" + c.r + ", c.i=" + c.i);
```

cabs

Complex absolute value.

SYNOPSIS:

```
double cabs();
cmplx z;
double a;
a = cabs(&z);
```

DESCRIPTION:

If z = x + iy

then

a = sqrt(x**2 + y**2).

Overflow and underflow are avoided by testing the magnitudes of x and y before squaring. If either is outside half of the floating point full scale range, both are rescaled.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	-30,+30	30000	3.2e-17	9.2e-18
IEEE	-10,+10	100000	2.7e-16	6.9e-17

JavaScript:

```
var z = {"r":3.14,"i":3.14};
var a = cephes.cabs(z);
```

where a is an object of schema

```
{
  "r" : double,
  "i" : double
}
```

csqrt

Complex square root.

SYNOPSIS:

void csqrt();
cmplx z, w;

csqrt(&z, &w);

DESCRIPTION:

If z = x + iy, r = |z|, then

$$\text{Im } w = [(r - x)/2]^{1/2},$$

Re w = y / 2 Im w.

Note that -w is also a square root of z. The root chosen is always in the upper half plane.

Because of the potential for a cancellation error in r - x, the result is sharpened by doing a Heron iteration (see sqrt) in complex arithmetic.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	-10,+10	25000	3.2e-17	9.6e-18
IEEE	-10,+10	100000	3.2e-16	7.7e-17

JavaScript:

var x = {"r":4.5,"i":3.14} ;
var a = cephes.csqrt(x);

returns a, complex object of schema
{
"r" : double,

"i": double

}

Complex Exponential and Trigonometric

- cexp - Exponential
- clog - Logarithm
- ccos - Cosine
- cacos - Arc cosine
- csin - Sine
- casin - Arc sine
- ctan - Tangent
- catan - Arc tangent
- ccot - Cotangent

cexp

Complex exponential function.

SYNOPSIS:

```
void cexp();
cmplx z, w;

cexp(&z, &w);
```

DESCRIPTION:

Returns the exponential of the complex argument z into the complex result w.

If

$z = x + iy$,

$r = \exp(x)$,

then

$w = r \cos y + i\, r \sin y$.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	-10,+10	8700	3.7e-17	1.1e-17
IEEE	-10,+10	30000	3.0e-16	8.7e-17

clog

Complex natural logarithm.

SYNOPSIS:

```
void clog();
cmplx z, w;

clog(&z, &w);
```

DESCRIPTION:

Returns a complex logarithm to the base e (2.718...) of the complex argument x.

If z = x + iy, r = sqrt(x**2 + y**2),
then
    w = log(r) + i arctan(y/x).

The arctangent ranges from -PI to +PI.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	-10,+10	7000	8.5e-17	1.9e-17
IEEE	-10,+10	30000	5.0e-15	1.1e-16

Larger relative errors can be observed for z near 1 +i0. In IEEE arithmetic the peak absolute error is 5.2e-16, rms absolute error 1.0e-16.

CCOS

Complex circular cosine.

SYNOPSIS:

```
void ccos();
cmplx z, w;

ccos(&z, &w);
```

DESCRIPTION:

If

$z = x + iy,$

then

$w = \cos x \cosh y - i \sin x \sinh y.$

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
DEC	-10,+10	8400	4.5e-17	1.3e-17
IEEE	-10,+10	30000	3.8e-16	1.0e-16

cacos

Complex circular arc cosine.

SYNOPSIS:

```
void cacos();
cmplx z, w;

cacos(&z, &w);
```

DESCRIPTION:

w = arccos z = PI/2 - arcsin z.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	-10,+10	5200	1.6e-15	2.8e-16
IEEE	-10,+10	30000	1.8e-14	2.2e-15

csin

Complex circular sine.

SYNOPSIS:

```
void csin();
cmplx z, w;

csin(&z, &w);
```

DESCRIPTION:

If

$$z = x + iy,$$

then

$$w = \sin x \cosh y + i \cos x \sinh y.$$

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
DEC	-10,+10	8400	5.3e-17	1.3e-17
IEEE	-10,+10	30000	3.8e-16	1.0e-16

casin

Complex circular arc sine.

SYNOPSIS:

```
void casin();
cmplx z, w;

casin(&z, &w);
```

DESCRIPTION:

Inverse complex sine:

$$w = -i \log(iz + \sqrt{1 - z^2}).$$

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
DEC	-10,+10	10100	2.1e-15	3.4e-16
IEEE	-10,+10	30000	2.2e-14	2.7e-15

Larger relative error can be observed for z near zero. Also tested by csin(casin(z)) = z.

ctan

Complex circular tangent.

SYNOPSIS:

```
void ctan();
cmplx z, w;

ctan(&z, &w);
```

DESCRIPTION:

If

$$z = x + iy,$$

then

$$w = \frac{\sin 2x + i \sinh 2y}{\cos 2x + \cosh 2y}.$$

On the real axis the denominator is zero at odd multiples of PI/2. The denominator is evaluated by its Taylor series near these points.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	-10,+10	5200	7.1e-17	1.6e-17
IEEE	-10,+10	30000	7.2e-16	1.2e-16

Also tested by ctan * ccot = 1 and catan(ctan(z)) = z.

catan

Complex circular arc tangent.

SYNOPSIS:

```
void catan();
cmplx z, w;

catan(&z, &w);
```

DESCRIPTION:

If

$$z = x + iy,$$

then

$$\text{Re } w = -\arctan\left(\frac{2x}{(1 - x^2 - y^2)}\right) + k\,\text{PI}$$

$$\text{Im } w = -\frac{1}{4}\log\left(\frac{(x^2 + (y+1)^2)}{(x^2 + (y-1)^2)}\right)$$

Where k is an arbitrary integer.

ACCURACY:

		Relative error:		
arithmetic	domain	# trials	peak	rms
DEC	-10,+10	5900	1.3e-16	7.8e-18
IEEE	-10,+10	30000	2.3e-15	8.5e-17

The check catan(ctan(z)) = z, with |x| and |y| < PI/2, had peak relative error 1.5e-16, rms relative error 2.9e-17. See also clog().

ccot

Complex circular cotangent.

SYNOPSIS:

```
void ccot();
cmplx z, w;

ccot(&z, &w);
```

DESCRIPTION:

If

$$z = x + iy,$$

then

$$w = \frac{\sin 2x - i \sinh 2y}{\cosh 2y - \cos 2x}.$$

On the real axis, the denominator has zeros at even multiples of PI/2. Near these points it is evaluated by a Taylor series.

ACCURACY:

	Relative error:			
arithmetic	domain	# trials	peak	rms
DEC	-10,+10	3000	6.5e-17	1.6e-17
IEEE	-10,+10	30000	9.2e-16	1.2e-16

Also tested by ctan * ccot = 1 + i0.

errors

Printing an error message

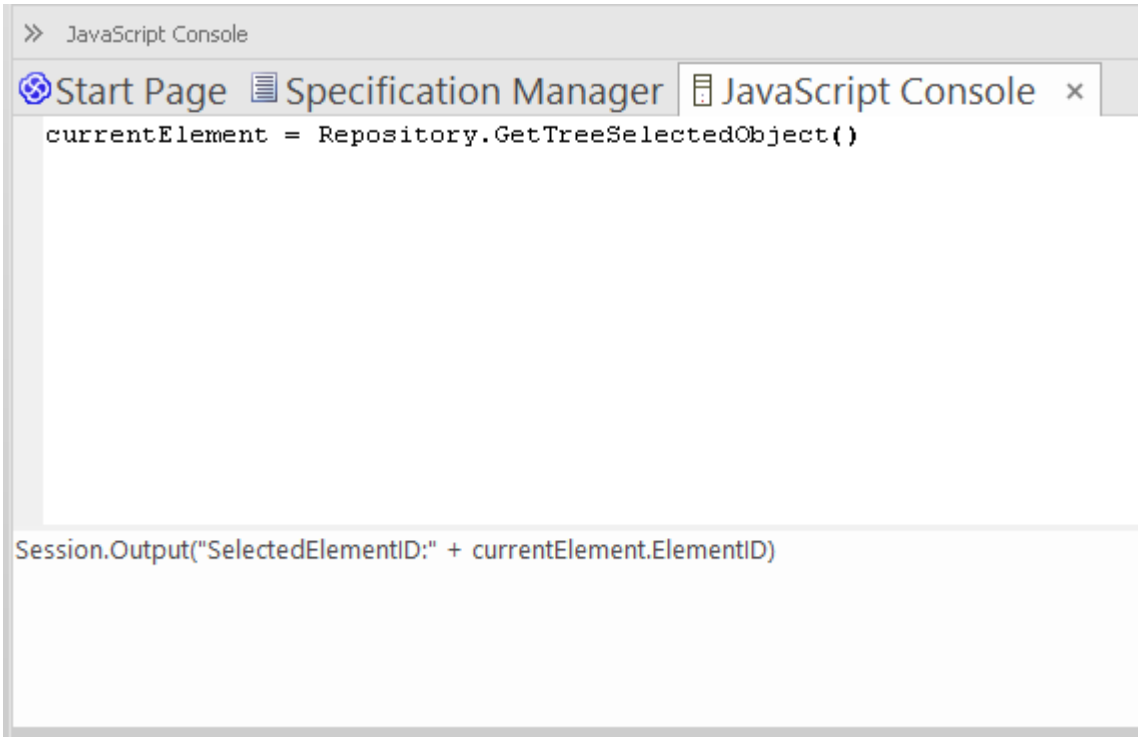
```
var cephes_errors = [
"'unknown'",      /* error code 0 */
"'domain'",       /* error code 1 */
"'singularity'",  /* et seq.      */
"'overflow'",
"'underflow'",
"'total loss of precision'",
"'partial loss of precision'" ];

function printError()
{
        var er = cephes.geterror();
        if(er>0)
        {
                Session.Output( "cephes error " + err + " " + cephes_errors[er]);
        }
}
```

Testing for error

```
if(cephes.inerror())
{
   printError();
}
```

JavaScript Console

The JavaScript console is a command line interpreter that accepts single line JavaScript commands that will be executed one at a time. You type the commands into the text entry panel at the bottom of the screen and, when you press the Enter key to execute the command, it is added to the upper, output window with any output from the command. Consider this example.



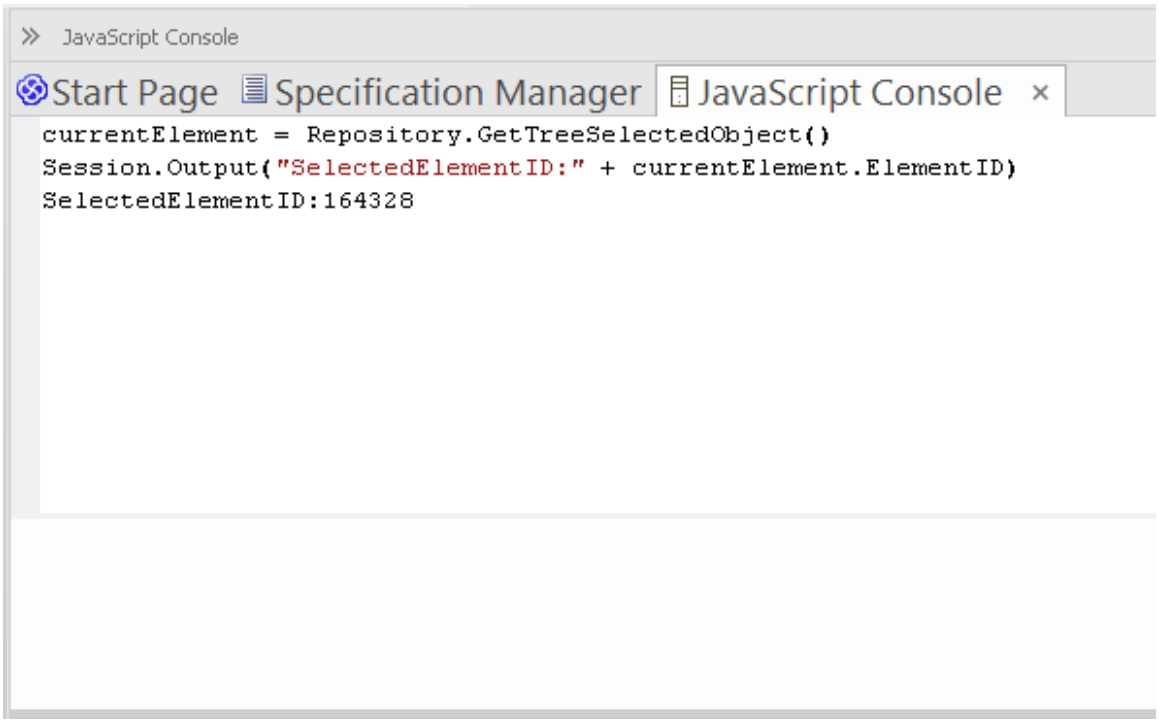
In the bottom panel, the user has typed:

```
currentElement = Repository.GetTreeSelectedObject()
```

They have pressed the Enter key, and the command has been displayed in the top panel. The user has then typed, in the lower panel:

```
Session.Output("Selected ElementID:"+currentElement.ElementID)
```

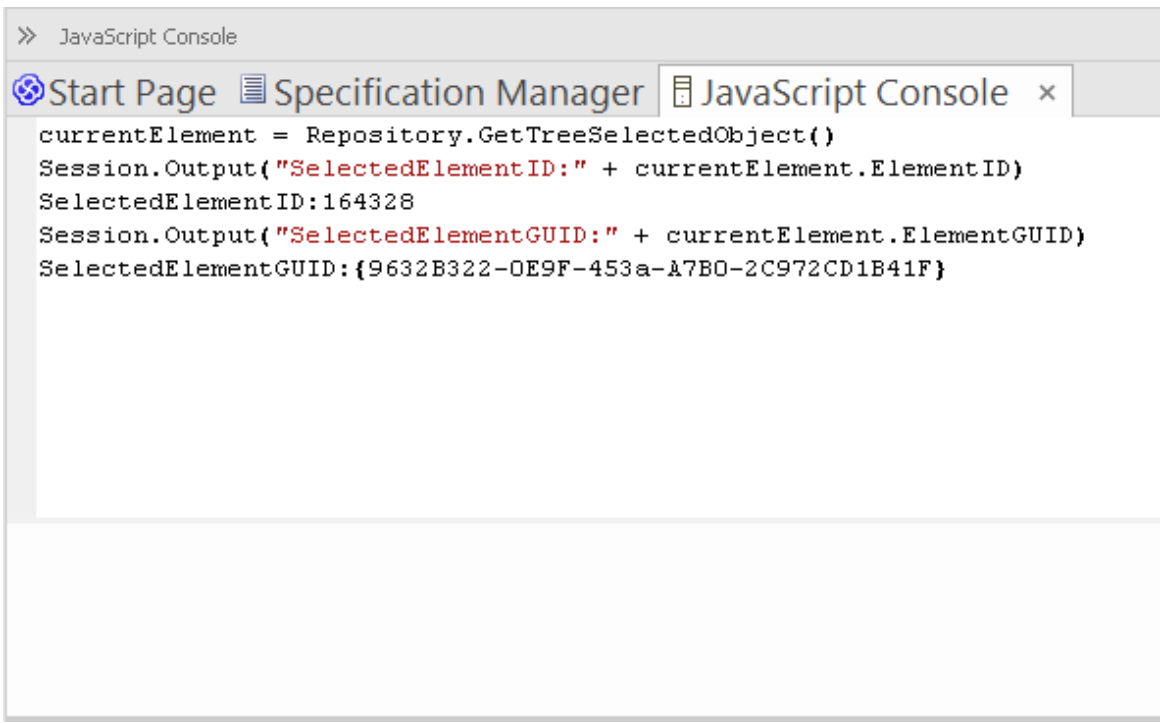
When they press the Enter key, the Console displays this command and the output from the command.



The user then enters a third command:

```
Session.Output("SelectedElementGUID:" + currentElement.ElementGUID)
```

This results in the output shown here, in the upper panel:



This feature is available in the Corporate, Unified and Ultimate editions of Enterprise Architect.

Access

Ribbon	Specialize > Tools > JavaScript

	Simulate > Console > JavaScript

Console Commands

Console commands are preceded by the ! character and instruct the console to perform an action.

The available console commands include:

- !clear - clears the console display
- !save - saves the console display to a file
- !help - prints a list of commands
- !close - closes the console
- !include <scriptname> - executes the named script item; scriptname is of the format GroupName.ScriptName (spaces are allowed in names)
- ? - lists commands (same as !help)
- ? <variable or function name> - outputs the value.

To list these commands on the 'Console' tab itself, type ? in the lower panel (without the preceding ! character) and press the Enter key.

If you intend to execute scripts, you might want to have the Script Library (Scripting window) open as well, so that you can see the scripts available to call. Select the 'Specialize > Tools > Script Library' ribbon option.

Console Window

The Console window is a command line interpreter through which you can quickly enable a script engine and enter commands to act on the script (JScript, JavaScript and VBScript).

You can open the Console window for JavaScript through the Simulate and Specialize ribbons. You can open the Console window for VBScript and JScript through the Specialize ribbon.

For the three scripting languages, you type the commands in the field at the bottom of the window; when you press the Enter key, the script console executes the commands and displays any output immediately. The Console commands are described in the *JavaScript Console* Help topic.

Access

Ribbon	Specialize > Tools > JavaScript Specialize > Tools > VBScript Specialize > Tools > JScript Simulate > Console > JavaScript

Notes

- This facility is available in the Corporate, Unified and Ultimate Editions

Solvers Interface

The Solvers Interface enables you to invoke a set of commands in JavaScript that define and enact a Solver Class to perform mathematical operations on data. The principle function of the Solver Class is to provide integration with external tools such as MATLAB and Octave during a simulation, and either expose the results in Octave or MATLAB, or bring them back into Enterprise Architect for representation there, perhaps in a Dynamic Chart. More generally, the Solvers interface can be used in model-based Add-Ins and custom scripts.

To call functions from Octave or MATLAB, you need to be familiar with the functions available in the appropriate product library, as described in the product documentation.

Solver Constructor

Constructor	Description
Solver(string solverName)	Creates a new Solver connected to a new instance of the specified helper application.

Solver Methods

Method	Description
get(string name)	Retrieves a named value from within the Solver environment.
set(string name, object value)	Assigns a new value to a named variable in the Solver environment.
exec(string name, string arguments, int returnValues)	Executes a named function. The actual functions will depend on the type of Solver being used.

Script Editor

Using the Script Editor you can perform a number of operations on an open script file, such as:

- Save changes to the current script
- Save the current script under a different name
- Run the script
- Debug the script
- Stop the executing script
- View the script output in the 'Scripts' tab of the System Output window

The editor is based on, and provides the facilities of, the common Code Editor in the application work area.

Access

Ribbon	Specialize > Tools > Script Library > expand script group and right-click on [script name] > Edit Script or Specialize > Tools > Script Library > expand script group and double-click on [script name]

Facilities

Facility	Detail
Scripting Objects	Enterprise Architect adds to the available functionality and features of the editor script language by providing inbuilt objects; these are either Type Libraries providing Intelli-sense for editing purposes, or Runtime objects providing access to objects of the types described in the Type Libraries. The available Intelli-sense scripting objects are: • EA • MathLib • System The runtime scripting objects are: • Repository (Type: IDualRepository, an instance of EA.Repository, the Enterprise Architect Automation Interface) • Maths (Type: IMath, an instance of MathLib; this exposes functions from the Cephes mathematical library for use in scripts) • Session (Type: ISession, an instance of System)
Script Editing Intelli-sense (Required Syntax)	Intelli-sense is available not only in the 'Script Editor', but also in the 'Script Console'; Intelli-sense at its most basic is presented for the inbuilt functionality of the script engine. For Intelli-sense on the additional Enterprise Architect scripting objects (as listed) you must declare variables according to syntax that specifies a type; it is not necessary to use this syntax to execute a script properly, it is only present so that the correct Intelli-sense can be displayed for an item.

	The syntax can be seen in, for example: Dim e as EA.Element Then when you type, in this case, e., the editor displays a list of member functions and properties of e's type. You select one of these to complete the line of script; you might, therefore, type: VBTrace(e. As you type the period, the editor presents the appropriate list and you might double-click on, for example, Abstract; this is inserted in the line, and you continue to type or select the rest of the statement, in this case adding the end space and parenthesis: VBTrace(e.Abstract)
Keystrokes	In the Script Editor or Console, Intelli-sense is presented on these keystrokes. • Press . (period) after an item to list any members for that item's type • Press Ctrl+Space on a word to list any Intelli-sense items with a name starting with the string at the point the keys were pressed • Press Ctrl+Space when not on a word to display any available top level Intelli-sense items - these are the Intelli-sense objects already described plus any built-in methods and properties of the current scripting language
Include Script Libraries	An *Include* statement (!INC) allows a script to reference constants, functions and variables defined by another script accessible within the Scripting Window. Include statements are typically used at the beginning of a script. To include a script library, use this syntax: !INC [Script Group Name].[Script Name] For example: !INC Local Scripts.EAConstants-VBScript
Using Inbuilt Math Functions	Various mathematical functions are available within the Script Editor, through the use of the inbuilt Maths object. You can access the Maths object within the Script Editor by typing 'Maths' followed by a period. The Intelli-sense feature displays a list of the available mathematical functions provided by the Cephes Mathematical Library. For example: Session.Output "The square root of 9 is " & Maths.sqrt(9) Session.Output "2^10 = " & Maths.pow(2,10) The Maths object is available in the Unified and Ultimate Editions of Enterprise Architect.
Using COM / ActiveX Objects	VBScript, JScript and JavaScript can each create and work with ActiveX / COM objects. This can help you to work with external libraries, or to interact with other applications external to Enterprise Architect. For example, the Scripting.FileSystemObject Class can be used to read and write files on the local machine. The syntax for creating a new object varies slightly for each language, as illustrated by these examples: VBScript: set fsObject = CreateObject("Scripting.FileSystemObject") JScript: fsObject = new ActiveXObject("Scripting.FileSystemObject"); JavaScript:

	fsObject = new COMObject("Scripting.FileSystemObject");
Using JavaScript with Out-of-process COM Servers	Users of JavaScript in Enterprise Architect can access out-of-process COM servers. The application must be registered on the machine as providing local server support. The syntax for creating or obtaining a reference to an out-of-process server is: var server = new COMObject(*progID*, true); where *progID* is the registered program ID for the COM component ('Excel.Application', for example).
System Script Library	When Enterprise Architect is installed on your system, it includes a default script library that provides a number of helpful scripting functions, varying from simple string functions to functions for defining your own CSV or XMI import and export. To use the script library you must enable it in the 'MDG Technologies' dialog ('Specialize > Technologies > Manage Technology' ribbon option). Scroll through the list of technologies, and select the 'Enabled' checkbox against 'EAScriptLib'.

Notes

- The Script Editor is available in the Corporate, Unified and Ultimate Editions
- Enterprise Architect scripting supports declaring variables to match the Enterprise Architect types; this enables the editor to present Intelli-sense, but is not necessary for executing the script

Session Object

The Session runtime object provides a common input/feedback mechanism across all script languages, giving access to objects of the types described in the System Type library. It is available through Scripting window to any script run within Enterprise Architect.

Properties

Properties	Detail
Attributes	• UserName - Returns the current windows username (read only) • Version - Returns the version of this object (read only)
Methods	• Input(string Prompt) - displays a dialog box prompting the user to input a value; returns the string value that was entered by the user • Output(string Output) - writes text to the current default output location; during: - Normal script execution, output is written to the 'Script' tab of the System Output window - Script Debugging, output is written to the Debug window - Use of the Script Console, output is written to the Console • Prompt(string Prompt, long PromptType) - displays a modal dialog containing the specified prompt text and button types; returns the 'PromptResult' value corresponding to the button that the user clicked
PromptType values	• promptOK = 1 • promptYESNO = 2 • promptYESNOCANCEL = 3 • promptOKCANCEL = 4
PromptResult values	• resultOK = 1 • resultCancel = 2 • resultYes = 3 • resultNo = 4
Session.Prompt Example	(VBScript) If (Session.Prompt("Continue?", promptYESNO) = resultYes) Then...

Workflows

Workflows validate user work and actions against the policy and procedures within your model, providing a robust approach to applying company policy and strengthening project development guidelines.

Project Administrators can write workflows to manage the way users interact with a model, such as managing security, staff compliance and model access, and monitoring changes made by users. Administrators can also use workflows to control a user's capacity to change a model element, taking into account factors such as access rights, group membership and even the value of a proposed change.

Application of Workflows

Consideration	Description
Project Governance	Good corporate governance relies on well written and transparent project development guidelines and company policy. A project might be compromised if the appropriate policies and procedures are poorly understood and not followed correctly - effective governance can be hampered by human error and the costs of recovering from the inadequate compliance of developers.
Policies, Procedures and Development	Company policy and procedures can be integrated with the development process to manage workflows, determine access rights, extend role based security permissions and respond to property change events. This approach reduces compliance costs, enhances collaborative development and gives you confidence that projects are being developed correctly the first time around. Development teams can adhere to best practice guidelines that govern model validation, change management, access controls and general development principles.

Workflow Options

There are two options for using Workflow scripts:

- VBScript
- JavaScript.

The VBScript is an older version and is limited to a series of commands. The JavaScript version is a more recent edition and allows full access to all the Automation functions.

The JavaScript is documented under *EA_Connect* and the *Workflow Add-In Events* Help Topics.

The VBScript is documented under *Workflow Script Functions* Help topic.

Notes

- Workflows are available in the Corporate, Unified and Ultimate Editions of Enterprise Architect

Workflow Script Functions

Note: From Release 15.0 of Enterprise Architect, VBScript workflow scripts are available for use but deprecated. You can now use the Enterprise Architect Add-In model event EA_Connect to respond to Workflow Add-In events, which have a wider range of features and are not reliant on Visual Basic.

Workflow scripts are created in the Scripting window, under the Workflow group type as VBScripts. They are executed by the Enterprise Architect workflow engine, to manage user input.

You can make use of a range of functions and data structures to develop your scripts.

Access

Open the Scripting window using one of the methods outlined here, then click on the New Group button to create a new Workflow script group, before clicking on the New Script button to create a new script.

Ribbon	Specialize > Tools > Scripting

Workflow functions and data structures

Function	Description
Script Implementation	When a model is launched, the Workflow Engine is initialized with the current user and group memberships; this information determines who can access and modify parts of a given model. When a selected event occurs, the script engine is initialized with values including the author's name and access rights, and the element name and version details. The workflow script implements rules governing change management, access control and model validation; if a user attempts to make changes in violation of company policy, the script denies the update. The user is notified why the validation failed and the activity is logged. These reminders help to reinforce company policy, reduce human error and provide management with valuable project feedback.
Functions for User Input	These are functions that Enterprise Architect calls to validate and control user input. For each of the functions that Enterprise Architect calls, a set of objects are filled.
Functions to create a Search	These are functions that Enterprise Architect calls to create a search with user tasks.
Workflow Data Structures Enterprise Architect fills	These are workflow data structure objects that Enterprise Architect fills.
Workflow Data Structures you fill	These are Workflow data structure objects that you can fill.
Functions you call	These are functions that Enterprise Architect provides for you to call.

Notes

- If you make changes to a workflow script listed in the Scripting window, click on the Refresh Scripts button in the Scripting window toolbar to reload the script with the changes
- Workflow Scripting is available in the Corporate, Unified and Ultimate Editions of Enterprise Architect
- Workflow Scripting requires User Security to be enabled in order to function
- You need 'Admin Workflow' permission to develop and manage Workflow Scripts

Functions - Validate and Control User Input

Enterprise Architect calls a number of functions to validate and control user input. For each function a set of objects is filled.

Validate/Control User Input

Function	Action
AllowPhaseUpdate(OldValue, NewValue)	Validate a change a user has made to a phase. Return Value: • True to allow this user to make this change • False to disallow the change and revert to the previous value
AllowStatusUpdate(OldValue, NewValue)	Validate a change a user has made to a status. Return Value: • True to allow this user to make this change • False to disallow the change and revert to the previous value
AllowTagUpdate(TagName, OldValue, NewValue)	Validate a change a user has made to a Tagged Value. Return Value: • True to allow this user to make this change • False to disallow the change and revert to the previous value
AllowVersionUpdate(OldValue, NewValue)	Validate a change a user has made to a version. Return Value: • True to allow this user to make this change • False to disallow the change and revert to the previous value
CanEditPhase()	Enable or disable the control for editing a phase Return Value: • True to allow this user to make changes by enabling the control • False to completely disable edit of this property by disabling the control
CanEditStatus()	Enable or disable the control for editing a status. Return Value: • True to allow this user to make changes by enabling the control • False to completely disable edit of this property by disabling the control
CanEditTag(TagName)	Enable or disable the control for editing a Tagged Value. Return Value: • True to allow this user to make changes by enabling the control • False to completely disable edit of this property by disabling the control
CanEditVersion()	Enable or disable the control for editing a version. Return Value:

	• True to allow this user to make changes by enabling the control • False to completely disable edit of this property by disabling the control
OnPreNewElement(ElementType, ElementStereotype)	Allow or disallow the creation of the specified element. Return Value: • True to allow this user to create the element/connector • False to prevent this user from creating the element
OnPreNewConnector(ConnectorType, ConnectorSubType, ConnectorStereotype)	Allow or disallow the creation of the specified connector. Return Value: • True to allow this user to create the element/connector • False to prevent this user from creating the element
PreAllowPhaseUpdate(OldValue, NewValue)	Determine what information is required to validate this change. Return Value: Semi-colon separated list of additional data required in order to validate this change. Supported Data Type: • Tests - fill the Tests array in the WorkflowContext object
PreAllowStatusUpdate(OldValue, NewValue)	Determine what information is required to validate this change. Return Value: Semi-colon separated list of additional data required in order to validate this change. Supported Data Type: Tests - fill the Tests array in the WorkflowContext object
PreAllowTagUpdate(TagName, OldValue, NewValue)	Determine what information is required to validate this change. Return Value: Semi-colon separated list of additional data required in order to validate this change. Supported Data Type: Tests - fill the Tests array in the WorkflowContext object
PreAllowVersionUpdate(OldValue, NewValue)	Determine what information is required to validate this change. Return Value: Semi-colon separated list of additional data required in order to validate this change. Supported Data Type: Tests - fill the Tests array in the WorkflowContext object

Functions - Create a Search With User Tasks

These are functions that Enterprise Architect calls to create a search with user tasks.

Functions

Function	Action
GetWorkflowTasks	Describe the searches that this user must run. Return Value: Ignored

Filled Workflow Data Structures

These are the workflow data structures (objects) that Enterprise Architect fills.

Data Structures

Workflow Data Structure	Description
WorkflowUser	This object provides information about the user currently logged in to the model. It is filled by Enterprise Architect before any function is called by Enterprise Architect; it has these properties: • Username - the username for login to the system (if using Windows Authentication, this matches the Windows username) • Firstname - as found in the 'Security Users' dialog • Surname - as found in the 'Security Users' dialog • Fullname - the combination <Firstname> <Surname> (the form Enterprise Architect uses for 'Author' fields and similar) • Department - the department in which the user works, as found in the 'Security Users' dialog Calls: This object calls the IsMemberOf(GroupName) function.
WorkflowContext	This object provides information about the object currently in context. It is filled by Enterprise Architect before any searches except GetWorkflowTasks are run; it has these properties: • MetaType - the type of the current object, either an Enterprise Architect core type or a profile-specified metatype • Name - as found in the object 'Properties' dialog • Status - as found in the object 'Properties' dialog • Phase - as found in the object 'Properties' dialog • Version - as found in the object 'Properties' dialog • Stereotypes - an array of strings for the stereotypes applied to this object • Tags - an array of Tagged Values, providing: - Name - the Tagged Value name - Value - the Tagged Value value • Tests - an array of tests; only filled during an Allow* call after the PreAllow* call has specified that tests are required; provides these details, as found in the Test Cases window: - Name - Status - RunBy - CheckedBy - TestClass - TestType Calls: This object calls the TagValue(TagName) function.

Functions

Function	Action
IsMemberOf(GroupName)	Check the group membership of the current user. Return Value: Returns the value True if the current user is a member of the group with the specified name.
TagValue(TagName)	Get the value from a named tag. Return Value: Returns the value of the first Tagged Value with that name, or an empty string if no Tagged Value with that name exists.

Workflow Data Structures You Fill

These are the workflow data structures (objects) that you can fill.

Data Structures

Workflow Data Structure	Description
WorkflowStatus	Use this data structure to provide information on the status of the object. • LogEntry - set to True or False to indicate whether or not a log item should be recorded • Reason - indicate what reason should be recorded in the log • Action - indicate how to display the log message; valid values are: MessageBox, StatusBar and Output (default)
WorkflowSearches	Use this data structure to provide an array of searches. Use Redim WorkflowSearches(x) to specify the number of searches being provided. Each search has these attributes: • Name - the name of this search • Group - the name of the group that this search should appear under in the 'Search' combo box • ID - the GUID for this search • Tasks - the array of tasks that this search looks for; an entry describes how to find all objects required to meet a particular task: - Name - the name of the task, as displayed in the Model Search view; workflow searches are grouped by this field by default - Conditions - an array of conditions, all of which must be matched for an object to be included in this task; a condition is a comparison of a single field to a value: - Column - the name of the field - Operator - operator types, either = (provide matching values only) or <> (provide non-matching values only) - Value - if this contains a comma, the string is treated as a comma separated list of values to compare against; otherwise the string is a single value to compare against

Functions You Call

undefined

Functions

Function	Action
NewSearch(name, group, guid, taskcount)	undefined
NewTask(name, conditioncount)	undefined
NewCondition(column, operator, value)	undefined
SetLastError(message, outputMethod)	undefined

Script Debugging

Script debugging aids in the development and maintenance of model scripts, and monitoring their activity at the time of execution. While debugging a script, you can:

- Control execution flow using the 'Debug', 'Step Over', 'Step Into', 'Step Out' and 'Stop Script' buttons on the Script Editor toolbar
- Set Breakpoints, Recording Markers and Tracepoint Markers
- Use the Debug window to view output generated by the script
- Use the Locals window to inspect values of variables, including objects from the Automation Interface
- Use the Record & Analyze window to record a Sequence diagram of the script execution

Access

Ribbon	Specialize > Tools > Script Library > right-click on [script name] > Debug Script
Other	Script Editor window toolbar : Click on the toolbar icon

Begin debugging a model script

Step	Action
1	Open a model script in the Script Editor.
2	Set any Breakpoints on the appropriate line(s) of code.
3	Click on the toolbar icon (Debug).

Notes

- Script debugging is supported for VBScript, JScript and JavaScript
- VBScript and JScript require the Microsoft Process Debug Manager to be installed on the local machine; this is available through various Microsoft products including the free 'Microsoft Script Debugger'
- Breakpoints are not saved for scripts and will not persist when the script is next opened
- While debugging, script output is redirected to the Debug window