ENTERPRISE ARCHITECT

User Guide Series

Add-ins & Scripting

Author: Sparx Systems

Date: 2022-10-03

Version: 16.0

CREATED WITH ENTERPRISE ARCHITECT

Table of Contents

Add-ins & Scripting 13
Scripting 15
Scripting Window 18
Script Group Properties 20
JavaScript Math Library 22
Arithmetic and Algebraic 23
sqrt 24
lsqrt 25
cbrt 26
polevl, p1evl 27
chbevl 29
round 30
floor 31
ceil 32
frexp 33
ldexp 34
fabs 35
signbit 36
isnan 37
isfinite 38
poladd 39
polsub 40
polmul 41
poldiv 42
polsbt 43
poleva 44
polclr 45
polmov 46
Exponential and Trigonometric 47
acos 48
acosh 49
asinh 50
atanh 51
asin 52
atan 53
atan2 54
cos 55
cosdg 56
exp 57
exp2 58
exp10 59
cosh 60
sinh 61
tanh 62
log 63
log2 64
log10 65

pow 66
powi 67
sin 68
sindg 69
tan 70
tandg 71
Exponential integral 72
expn 73
shichi 74
sici 76
Gamma 78
beta 79
lbeta 80
fac 81
gamma 82
lgam 83
incbet 84
incbi 86
igam 87
igamc 88
igami 89
psi 90
rgamma 92
Error function 93
erf 94
erfc 95
dawsn 96
fresnl 97
Bessel 99
airy 100
j0 102
j1 103
jn 104
jv 105
y0 106
y1 107
yn 108
yv 109
i0 110
i0e 111
i1 112
i1e 113
iv 114
k0 115
k0e 116
k1 117
k1e 118
kn 119
Hypergeometric 120
hyperg 121
hyp2f1 122

hyp2f0 124
onef2 125
threef0 126
Elliptic 127
ellpe 128
ellie 129
ellpk 130
ellik 132
ellpj 133
Probability 134
bdtr 135
bdtrc 137
bdtri 139
chdtr 140
chdtrc 141
chdtri 142
fdtr 143
fdtrc 144
fdtri 146
gdtr 148
gdtrc 149
nbdtr 150
nbdtrc 151
ndtr 152
ndtri 153
pdtr 154
pdtrc 155
pdtri 156
stdtr 157
Miscellaneous 159
polylog 160
spence 162
zetac 163
zeta 164
struve 166
Matrix 167
fftr 168
simq 169
minv 170
mmmpy 172
mvmpy 173
mtransp 174
eigens 175
Numerical Integration 178
simpsn 179
Complex Arithmetic 180
cadd 181
csub 183
cmul 185
cdiv 187
cabs 189

csqrt 190
Complex Exponential and Trigonometric 192
cexp 193
clog 194
ccos 195
cacos 196
csin 197
casin 198
ctan 199
catan 200
ccot 201
errors 202
JavaScript Console 203
Console Window 206
Solvers Interface 207
Script Editor 208
Session Object 211
Workflows 212
Workflow Script Functions 213
Functions - Validate and Control User Input 215
Functions - Create a Search With User Tasks 217
Filled Workflow Data Structures 218
Workflow Data Structures You Fill 220
Functions You Call 221
Script Debugging 222
Hybrid Scripting 223
C# Example 224
Java Example 226
Model Add-Ins 228
Create an Add-In 229
Responding to Events 231
Edit Add-In Code 232
Model Add-In Management 233
Signal Reference Library 234
Sample Add-Ins 235
Enterprise Architect Add-In Model 236
The Add-In Manager 237
Create and Deploy Add-Ins 238
Create Add-Ins 239
Define Menu Items 240
Deploy Add-Ins 242
Tips and Tricks 244
Add-In Search 246
EA_SampleSearch 247
XML Format (Search Data) 248
Add-In Events 250
EA_OnAddinPropertiesTabChanging 251
EA_Connect 252
EA_Disconnect 253
EA_GetMenuItems 254
EA_GetMenuState 256

EA_GetRibbonCategory 258
EA_MenuClick 259
EA_OnOutputItemClicked 261
EA_OnOutputItemDoubleClicked 262
EA_ShowHelp 263
Broadcast Events 264
Add-In License Management Events 266
EA_AddinLicenseValidate 267
EA_AddinLicenseGetDescription 268
EA_GetSharedAddinName 269
Custom Table Events 271
EA_OnCustomTableBeginEdit 272
EA_OnCustomTableEndEdit 273
EA_OnCustomTableSelectionChanged 274
EA_OnCustomTableCellUpdated 275
Schema Composer Events 276
EA_GenerateFromSchema 277
EA_GetProfileInfo 278
EA_IsSchemaExporter 279
Compartment Events 280
EA_QueryAvailableCompartments 281
EA_GetCompartmentData 283
Context Item Events 286
EA_OnContextItemChanged 287
EA_OnContextItemDoubleClicked 288
EA_OnNotifyContextItemModified 289
EA_FileClose 290
EA_FileNew 291
EA_FileOpen 292
EA_OnPostCloseDiagram 293
EA_OnPostInitialized 294
EA_OnPostOpenDiagram 295
EA_OnPostTransform 296
EA_OnPreExitInstance 297
EA_OnRetrieveModelTemplate 298
EA_OnTabChanged 300
EA_LoadWindowManager 301
Model Validation Events 302
EA_OnInitializeUserRules 303
EA_OnStartValidation 304
EA_OnEndValidation 305
EA_OnRunElementRule 306
EA_OnRunPackageRule 307
EA_OnRunDiagramRule 308
EA_OnRunConnectorRule 309
EA_OnRunAttributeRule 310
EA_OnRunMethodRule 311
EA_OnRunParameterRule 312
Model Validation Example 313
Post-New Events 319
EA_OnPostNewElement 320

EA_OnPostNewConnector 321
EA_OnPostNewDiagram 322
EA_OnPostNewDiagramObject 323
EA_OnPostNewAttribute 324
EA_OnPostNewMethod 325
EA_OnPostNewPackage 326
EA_OnPostNewGlossaryTerm 327
Pre-Deletion Events 328
EA_OnPreDeleteElement 329
EA_OnPreDeleteAttribute 330
EA_OnPreDeleteMethod 331
EA_OnPreDeleteConnector 332
EA_OnPreDeleteDiagram 333
EA_OnPreDeleteDiagramObject 334
EA_OnPreDeletePackage 335
EA_OnPreDeleteGlossaryTerm 336
Pre New-Object Events 337
EA_OnPreNewElement 338
EA_OnPreNewConnector 339
EA_OnPreNewDiagram 340
EA_OnPreNewDiagramObject 341
EA_OnPreDropFromTree 342
EA_OnPreNewAttribute 343
EA_OnPreNewMethod 344
EA_OnPreNewPackage 345
EA_OnPreNewGlossaryTerm 346
Tagged Value Events 347
EA_OnAttributeTagEdit 348
EA_OnConnectorTagEdit 349
EA_OnElementTagEdit 350
EA_OnMethodTagEdit 351
Technology Events 352
EA_OnInitializeTechnologies 353
EA_OnPreActivateTechnology 354
EA_OnPostActivateTechnology 355
EA_OnPreDeleteTechnology 356
EA_OnDeleteTechnology 358
EA_OnImportTechnology 359
Technology Rules 360
EARules_Initialize 361
Diagram Appearance Rule Events 363
EARules_ClosePartitionName 364
EARules_ElementDisplayName 365
EARules_GetCompartmentItem 366
EARules_GetCompartmentName 368
EARules_GetNameUnderline 369
EARules_GetPropertyString 370
EARules_GetShapeScript 371
EARules_ShowStereotype 372
EARules_StereotypeDisplayName 373
User Interface Rule Events 374

EARules_AllowNesting 375
EARules_AppendChildDiagrams 376
EARules_AppendChildElements 378
EARules_CanOverrideStereotype 380
EARules_CanProxy 381
EARules_CanReparent 382
EARules_CreateModel 383
EARules_EnableElementProperty 384
EARules_ForceLength 386
EARules_GetEquivalentDiagram 387
EARules_IsAdjustable 388
EARules_PropagateStereotype 389
EARules_ShowElementProperty 390
EARules_ShowFrame 392
EARules_ShowParentFrame 393
Custom Views 394
Create a Custom View 395
Custom Docked Window 396
MDG Add-Ins 398
MDG Events 399
MDG_BuildProject 400
MDG_Connect 401
MDG_Disconnect 402
MDG_GetConnectedPackages 403
MDG_GetProperty 404
MDG_Merge 405
MDG_NewClass 408
MDG_PostGenerate 409
MDG_PostMerge 410
MDG_PreGenerate 411
MDG_PreMerge 412
MDG_PreReverse 413
MDG_RunExe 414
MDG_View 415
Workflow Add-In Events 416
EA_AllowPropertyUpdate 417
EA_AllowTagUpdate 418
EA_CanEditProperty 419
EA_CanEditTag 420
Enterprise Architect Object Model 421
Using the Automation Interface 422
Connect to the Interface 423
Set References In Visual Basic 426
Examples and Tips 427
Call from Enterprise Architect 429
Available Resources 431
Reference 432
Interface Overview 433
App Object 435
Enumerations 436
ConstLayoutStyles 438

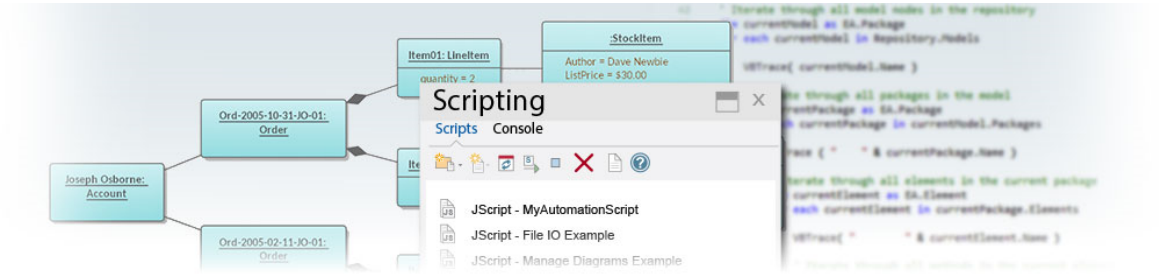
CreateBaselineFlag 439
CreateModelType 440
DocumentBreak 441
DocumentPageOrientation 442
DocumentType 443
EAEditionTypes 444
EnumRelationSetType 445
ExportPackageXMIFlag 446
MDGMenus 447
MessageFlag 448
ObjectType 449
PropType 451
ReloadType 452
ScenarioDiagramType 453
ScenarioStepType 454
ScenarioTestType 455
XMIType 456
Repository Package 457
Author Class 458
Client Class 459
Collection Class 461
The AddNew Function 463
Datatype Class 467
EventProperties Class 470
EventProperty Class 471
ModelWatcher Class 472
Package Class 473
ProjectIssues Class 483
ProjectResource Class 485
ProjectRole Class 487
PropertyType Class 489
Reference Class 491
Repository Class 493
SecurityUser Class 516
Stereotype Class 518
Task Class 520
Term Class 522
Properties Tab Package 524
PropertiesTab Class 525
Element Package 527
Constraint Class 529
Effort Class 531
Element Class 533
ElementGrid Class 548
File Class 549
Issue (Maintenance) Class 551
Metric Class 553
Requirement Class 555
Resource Class 557
Risk Class 559
Scenario Class 561

ScenarioExtension Class 563
ScenarioStep Class 564
TaggedValue Class 566
Test Class 568
Element Features Package 570
Attribute Class 571
AttributeConstraint Class 577
AttributeTag Class 579
CustomProperties Collection 581
EmbeddedElements Collection 582
Method Class 583
MethodConstraint Class 588
MethodTag Class 590
Parameter Class 592
ParamTag Class 595
Partitions Collection 597
Properties Class 598
TemplateParameter Class 600
Transitions Collection 602
Connector Package 603
Connector Class 604
ConnectorConstraint Class 612
ConnectorEnd Class 614
ConnectorTag Class 617
RoleTag Class 619
TemplateBinding Class 621
Diagram Package 623
Diagram Class 624
DiagramLink Class 632
DiagramObject Class 635
SwimlaneDef Class 641
Swimlanes Class 643
Swimlane Class 645
Project Interface Package 646
Project Class 647
Chart Package 665
Chart Enumerations 666
ChartAxisCrossType 667
ChartAxisIndex 668
ChartAxisLabelType 669
ChartAxisTickMarkType 670
ChartAxisType 671
ChartBarShape 672
ChartCategory 673
ChartColorMode 675
ChartCurveType 676
ChartDashStyle 677
ChartFrameStyle 678
ChartGradientType 679
ChartMarkerShape 680
ChartStockSeriesType 681

ChartType 682
ChartWallOptions 683
Chart Class 684
ChartAxisIndex Class 687
ChartDataValue Class 689
ChartDiagram3D Class 690
ChartFormatSeries Class 691
ChartSeries Class 692
Document Generator Interface Package 696
DocumentGenerator Class 697
Data Miner Package 703
DataMinerManager Class 704
DataMiner Class 706
DataSet Class 707
DMArray Class 708
DMAction Class 709
DMScript Class 710
DMConnection Class 711
TypeInfoProperties Package 712
TypeInfoProperties Class 713
TypeInfoProperty Class 714
Mail Interface Package 715
MailInterface Class 716
Search Window Package 719
EAContext Class 720
EASelection Class 722
SearchWindow Class 724
Simulation Package 726
Simulation Class 727
Schema Composer Package 729
SchemaProperty Class 730
SchemaProfile Class 732
SchemaComposer Class 733
ModelTypeEnum Class 735
ModelType Class 736
SchemaTypeEnum Class 738
SchemaType Class 739
SchemaPropEnum Class 740
SearchType Enumeration 741
SchemaNamespace Class 742
SchemaNamespaceEnum Class 743
Code Samples 744
Open the Repository 745
Iterate Through a .EAP File 746
Add and Manage Packages 747
Add and Manage Elements 749
Add a Connector 750
Add and Manage Diagrams 752
Add and Delete Features 753
Element Extras 754
Repository Extras 758

Stereotypes 761
Work With Attributes 762
Work With Methods 764

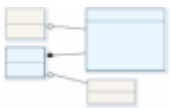
Add-ins & Scripting



Enterprise Architect has an incredible range of built-in features for working with models, but it also provides a range of environments for accessing and manipulating the contents of a repository programmatically. This is an extremely useful facility that gives you unlimited ability to query and manipulate models, add to the Enterprise Architect user interface, generate reports, and even create support for new modeling languages. The Automation Interface gives you access to the Object Model, which is an easy-to-use and well defined set of objects with properties and methods that can be used to query and manipulate the repository and its contents, shielding the programmer from having to know the underlying repository data structures.

The Automation Interface is available from a scripting framework built into the Enterprise Architect user interface, through external scripting environments, or through Add-Ins that can be built in a wide range of programming languages.

Facilities

Facility	Description
Scripting 	Learn about the flexible and easy-to-use scripting capability to programmatically inspect and/or modify elements within your currently open model.
Object Model 	Discover the Enterprise Architect Object Model. Write your own custom programs that access the information stored in Enterprise Architect.
Add-In Model 	The Enterprise Architect Add-In Model helps you build on the features provided by the Automation Interface to enable you to extend the Enterprise Architect user interface.
MDG Add-Ins 	MDG Add-Ins are specialized types of Add-In that have additional features and extra requirements. MDG Add-Ins are focused on generation, synchronization and general processes concerned with converting models to code and code to models.
Code Samples and Reference 	Access the wealth of knowledge and samples to help you complete your Add-In.

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

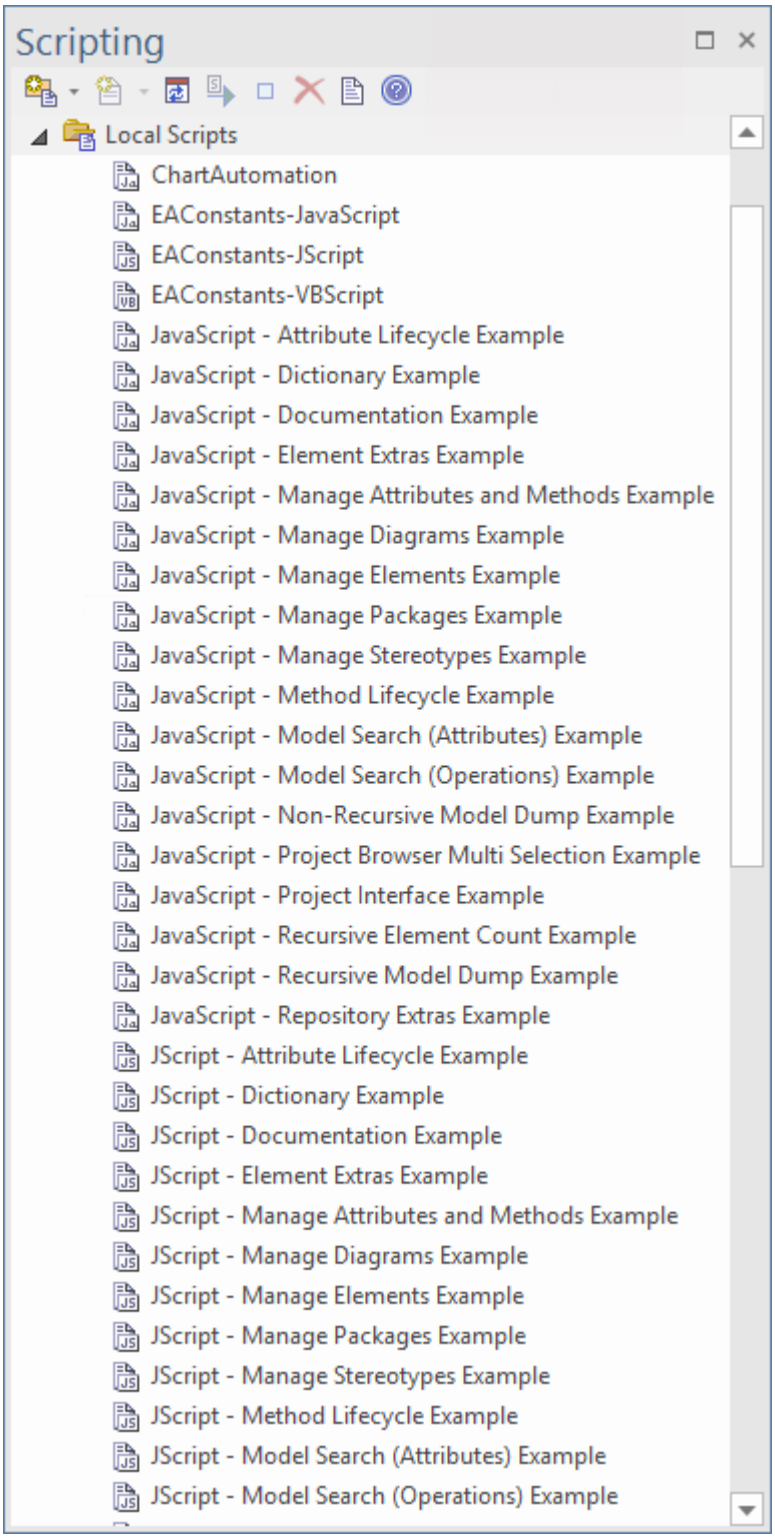
Benefits

- 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

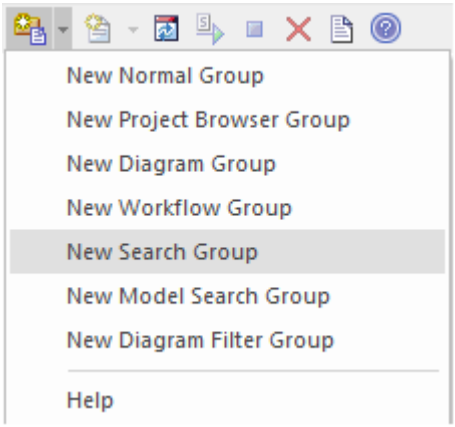
Using Scripts

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

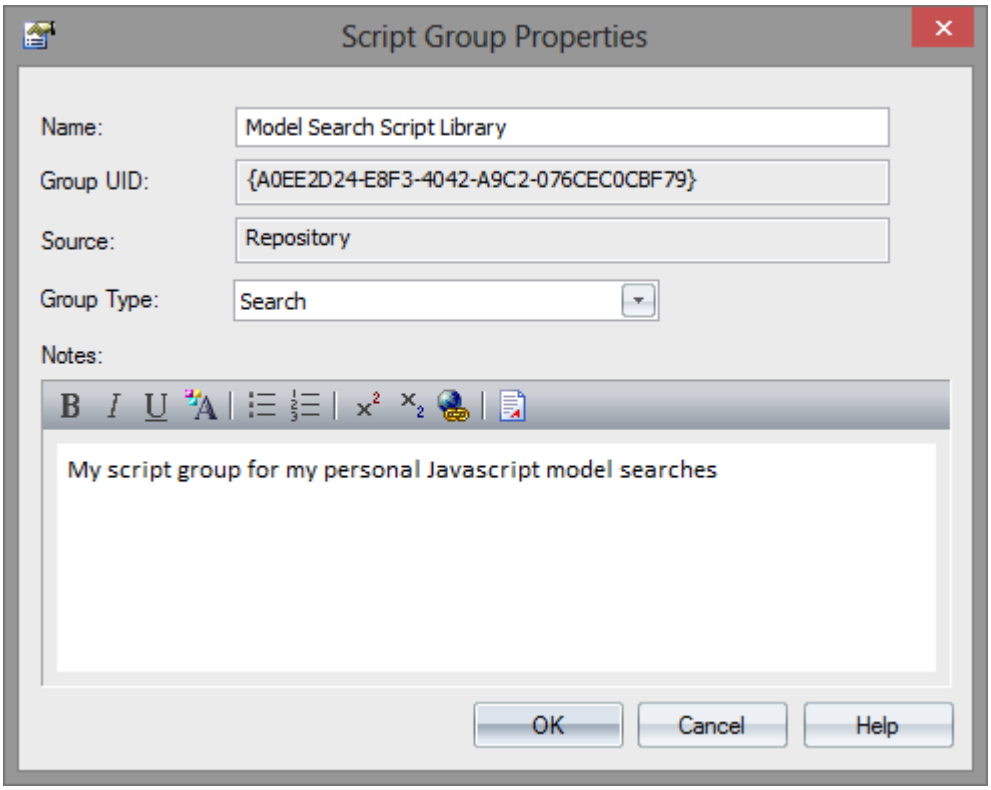
Icon	Action
	Create a new script group; this option displays a short menu of the types of script group you can create, namely: • Normal Group () • Browser window Group () • Diagram Group () • Workflow Group () • Search Group () • Model Search Group The new group is added to the end of the list in the 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:

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

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:

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

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:

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

$$w = x^2$$

where w = x 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:

```
              v  -x
Iv(x) = (x/2)  e   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:

$$\sum_{j=0}^{k} \binom{n+j-1}{j} p^n (1-p)^j$$

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:

$$\mathrm{ndtr}(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp(-t^2/2)\, dt$$

$$= (1 + \mathrm{erf}(z)) / 2$$
$$= \mathrm{erfc}(z) / 2$$

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:

$$\text{spence}(x) = -\int_1^x \frac{\log t}{t-1}\,dt$$

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

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

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

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

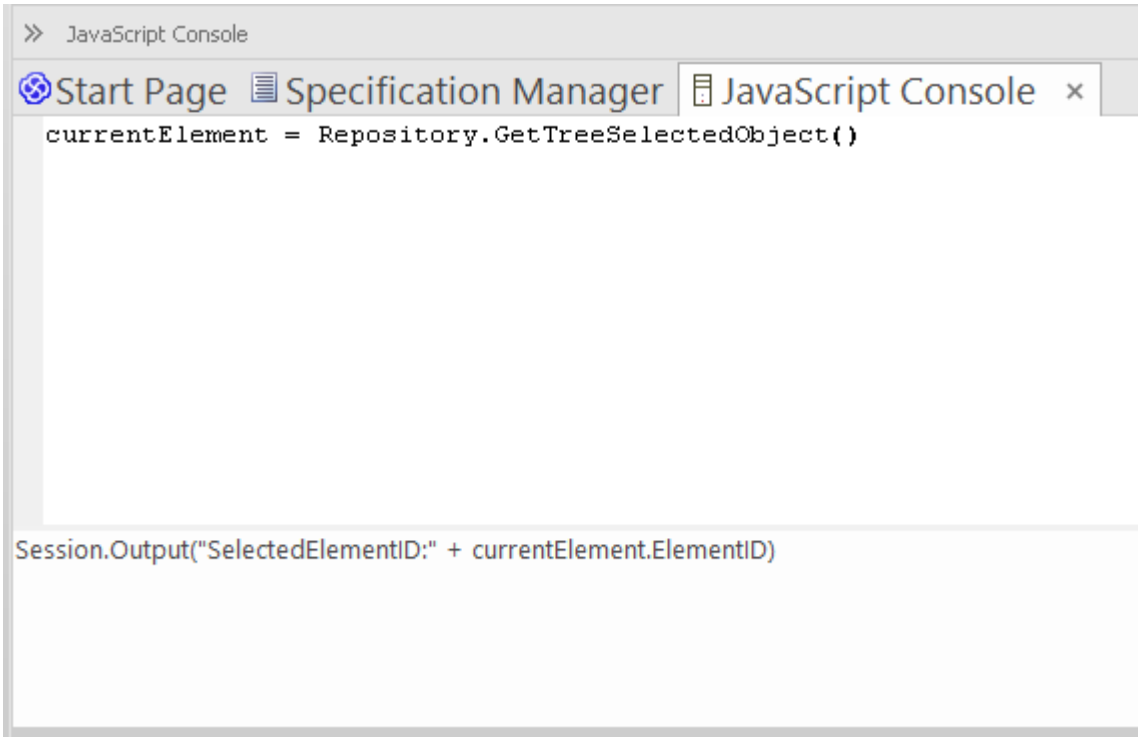
```
var cephes_errors = [
"'unknown'",      /* error code 0 */
"'domain'",       /* error code 1 */
"'singularity'",  /* et seq.      */
"'overflow'",
"'underflow'",
"'total loss of precision'",
"'partial loss of precision'" ];

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.



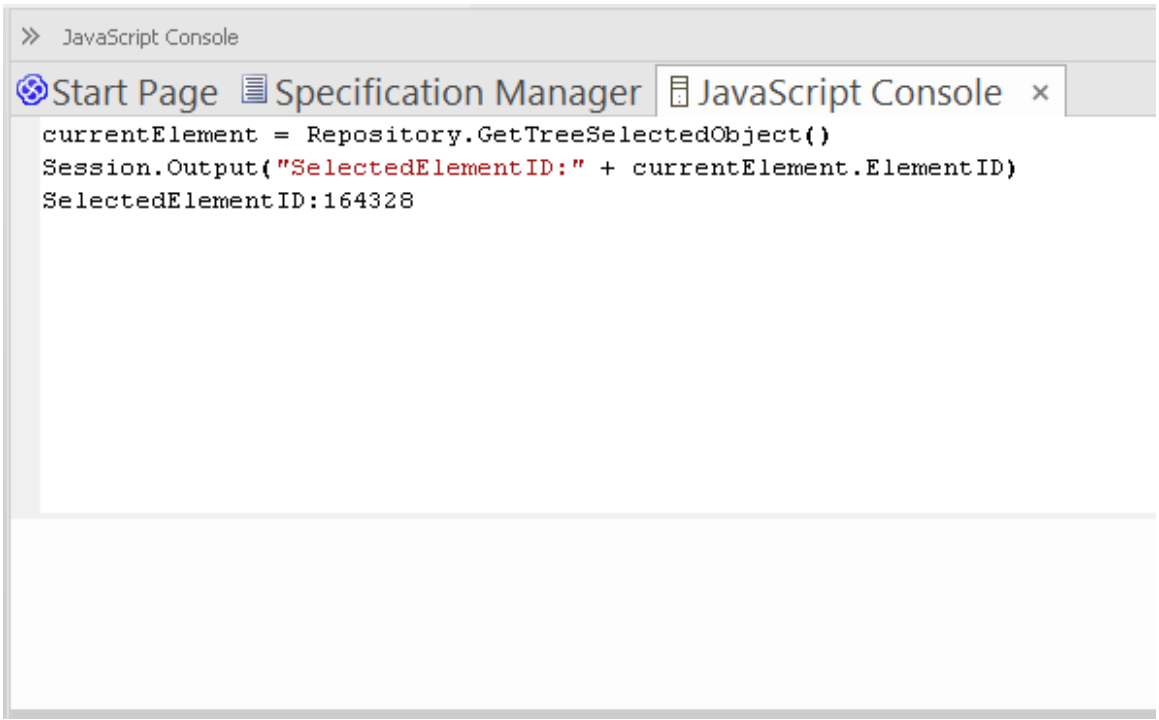
In the bottom panel, the user has typed:

```
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)
```

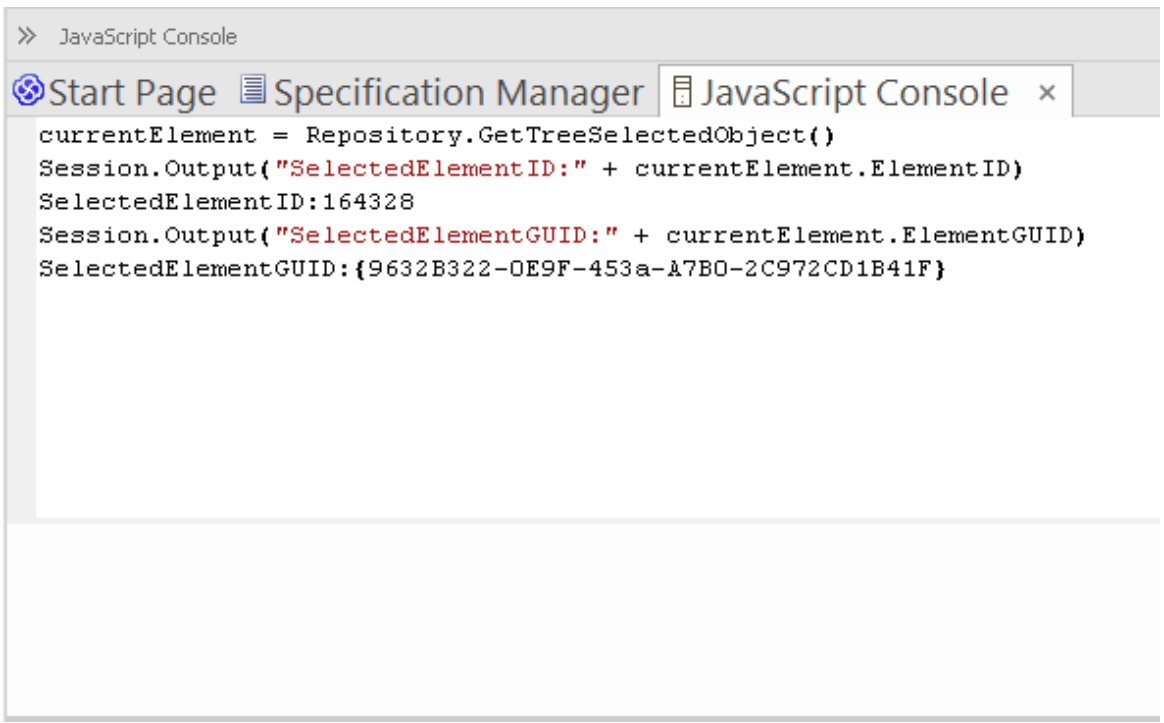
When they press the Enter key, the Console displays this command and the output from the command.



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

Hybrid Scripting

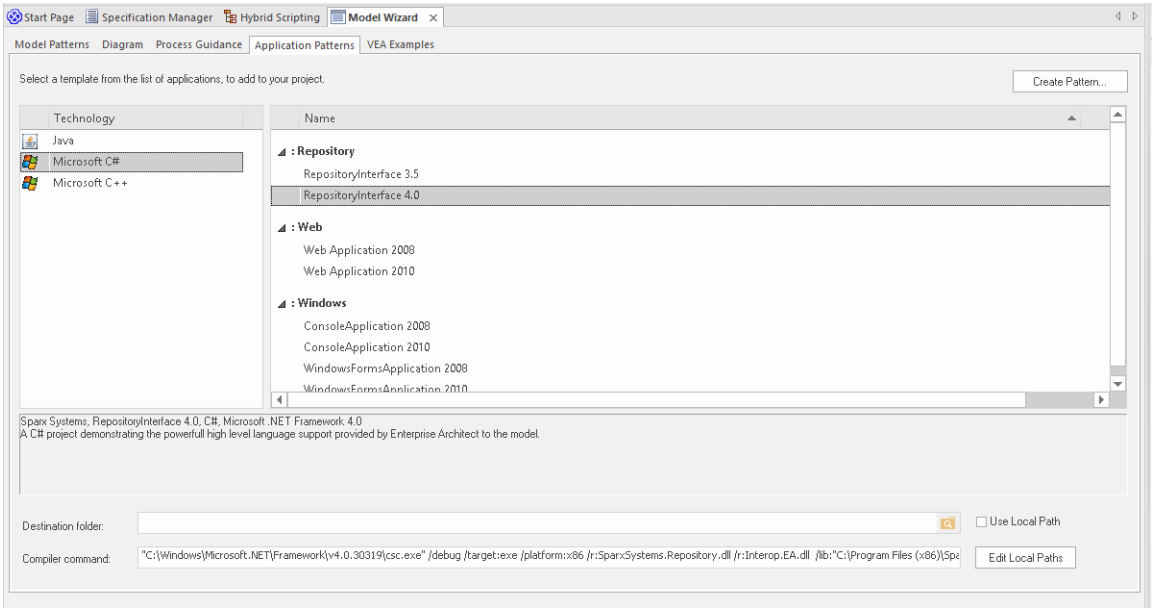
Hybrid scripting extends the capabilities of the standard scripting environment to high level languages such as Java and C#. Hybrid scripting provides a speed advantage over conventional scripting, and also allows script authors to leverage existing skills in popular programming languages.

Access

Ribbon	Develop > Source Code > Create From Pattern > Application Patterns

Features

- Superior execution speed
- Enhanced interoperability
- Full Visual Execution Analyzer support



C# Example

This sample program demonstrates how easy it is to navigate, query and report on the current model using any Microsoft .NET language. This example is written in C#.

When run, it will print the names of every Package in the model you are currently using.

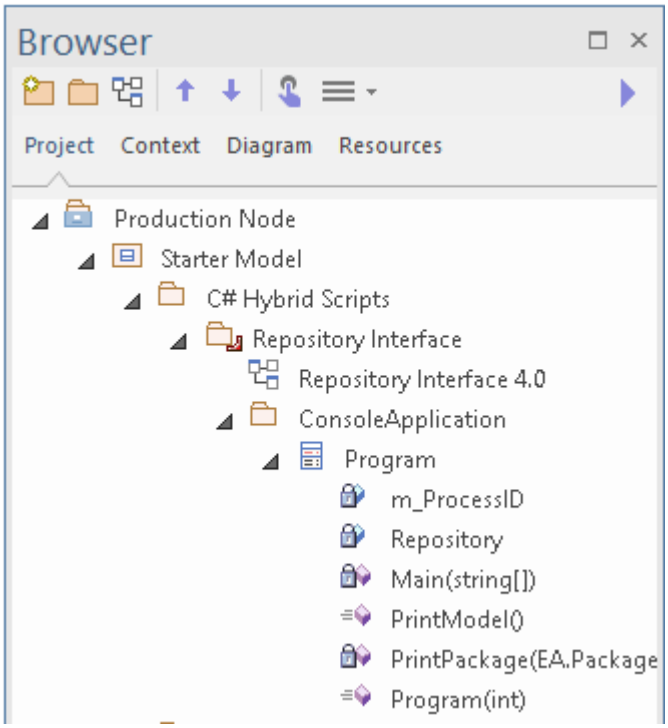
Create the Project

In the Browser window, select the Package in which to create the template, then use the 'Develop > Source Code > Create from Pattern' ribbon option to display the Patterns window; click on the 'Application Patterns' option.

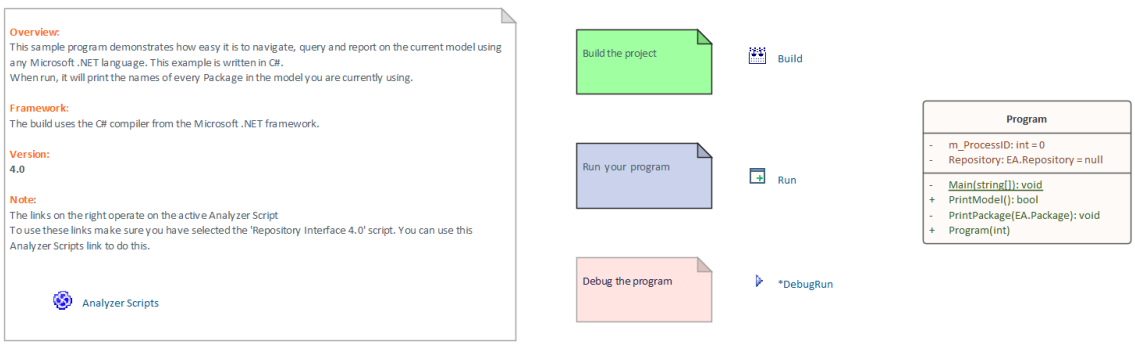
From the 'Application Patterns' page, select the *Microsoft C# > RepositoryInterface* template. (You can choose from either the 3.5 or the 4.0 framework versions.) Specify the destination folder on the file system where the project template will be created, and click on the OK button.

Open the Project

A Package structure similar to this will be created for you.



Expand the structure until you locate the *Repository Interface n.n* diagram and open it.



Build the Script

The commands on this diagram will operate on the active build configuration. Before executing them, double-click on the *Analyzer Scripts* link and select the checkbox next to the 'Repository Interface' build configuration.

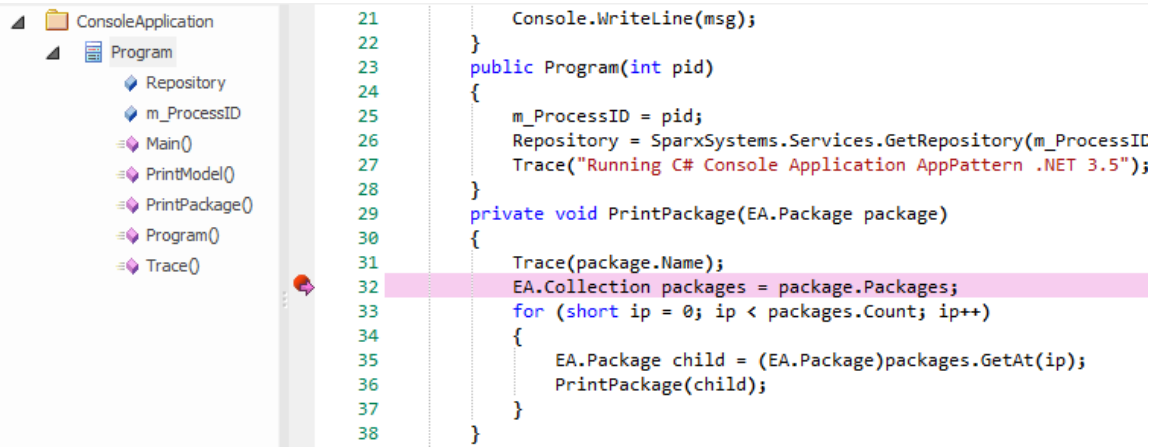
Run the Script

Double-click on the *Run* link to open the Console. The Console will pause after completion so you can read the output from the program; this output will also be sent to the 'Script' tab of the System Output window. You can alter this by changing the code.

Debug the Script

Select the 'Program' Class from the Browser window and press Ctrl+E to open the source code.

Place a Breakpoint in one of the functions and then double-click on the *DebugRun* link. When the Breakpoint is encountered, the line of code will become highlighted in the editor, as shown:



Java Example

This sample program demonstrates how easy it is to navigate, query and report on the current model using a high-level language such as Java.

When run, it will print the names of every Package in the currently-loaded model.

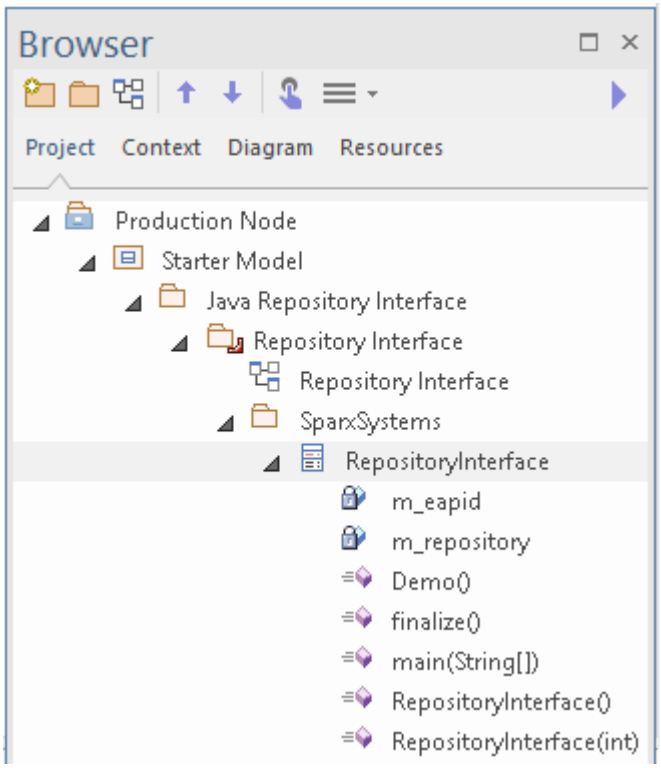
Create the Project

In the Browser window, select the Package in which to create the template, then use the 'Develop > Source Code > Create from Pattern' ribbon option to display the Patterns window; click on the 'Application Patterns' option.

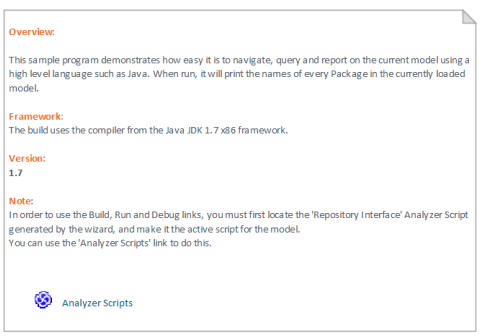
From the 'Application Patterns' page, select the *Java > RepositoryInterface* template. Specify the destination folder on the file system in which the project template will be created, and click on the OK button.

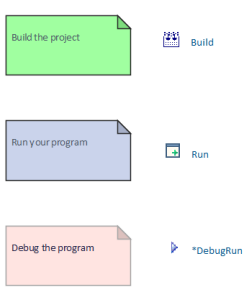
Open the Project

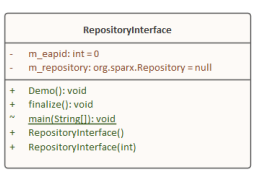
A Package structure similar to this will be created for you.



Expand the structure until you locate the 'Repository Interface' diagram and open it.







Build the Script

The commands on the diagram will operate on the active build configuration. Before executing them, double-click on the *Analyzer Scripts* link and select the checkbox next to the 'Repository Interface' build configuration.

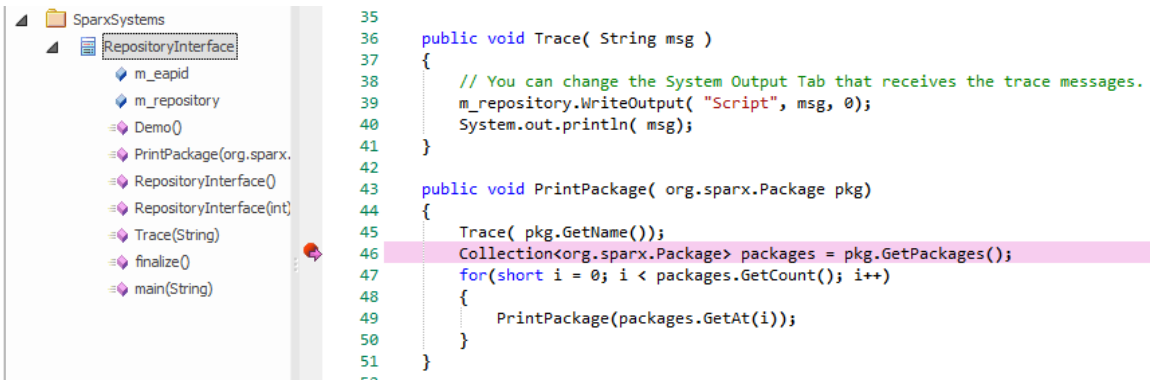
Run the Script

Double-click on the *Run* link; a Console will open. The Console will pause after completion so you can read the output. The output from the program will also be output to the 'Script' tab of the System Output window. You can alter this by changing the code.

Debug the Script

Select the 'Program' Class from the Browser window and press Ctrl+E to open the source code.

Place a breakpoint in one of the functions and then double-click on the *DebugRun* link. When the breakpoint is encountered the line of code will become highlighted in the editor, as shown.



Model Add-Ins

Enterprise Architect offers the function of developing and deploying Add-Ins completely within your model.

When to use a Model Add-In

Reason	Discussion
High Deployment Costs	In organizations where installing new or updated software is expensive, model Add-Ins can offer a workaround. New functionality can be added to Enterprise Architect without the need for new software to be installed on user machines.
Required for all users	When all users of a model need an Add-In to use the model as intended it can be difficult to ensure that the Add-In is installed and updated on all user machines. Model based Add-Ins are loaded by all required users automatically on model load. Alternative deployments allow users to opt-in to using an Add-In, with access controlled by security group.
Model Specific Behavior	For users regularly using multiple models, there will likely be some functions that are only required in some models but not others. By using model based Add-Ins, these functions can be added freely without requiring explicit coding based on the model.
Self Documenting	By modeling your Add-In directly, the documentation describing it is always accurate.

When not to use a Model Add-In

Reason	Discussion
Complex User Interface	The User Interface that model Add-Ins can create is currently not as expressive as Add-Ins written in a traditional IDE. If you need to show your users complex dialogs or forms, you might be better off using an alternative technology.
Use across many models	Add-In functionality that is required across multiple models might not be a good fit for model Add-Ins. In this situation you might need to consider the relative costs of a traditional Add-In vs deploying a model Add-In using XMI, controlled Packages or a re-usable asset service.

Notes

- This feature is available in the Corporate, Unified and Ultimate Editions of Enterprise Architect, from Release 15.0

Create an Add-In

Model Based Add-Ins are defined within the model, using Classes that are stereotyped as 'JavascriptAddin'. Using these stereotyped Classes, you can specify Receptions, Methods and Properties that together define the behaviors of the Add-In, and how it responds to the various events occurring within the system.

Receptions are defined for the Class, by specifying a signal that will be received. The Receptions allow you to specify JavaScript code that will be executed in response to receipt of the corresponding signal. Signals that are relevant to your Model Based Add-In should be included within the model in which you are defining or using Model Based Add-Ins. The Model Wizard (Start Page 'Create from Pattern' tab) offers a pattern that contains all of the signals relevant to Model Based Add-Ins, providing an easy means by which to include these signals in your model.

Functions defined as methods of the Class can be called by the Receptions code, while the Class attributes can be used to define global variables that are available to the executing code.

Create a JavaScript Add-In

Step	Action
1	Click on the  icon and select the 'Management > Model Add-Ins' Perspective.
2	Create or open a (Class) diagram on which to work, then open the 'Model Add-Ins' page of the Diagram Toolbox. (Use the Toolbox menu to select the 'Model Add-ins' page of the Toolbox.)
3	Create a JavascriptAddin by dropping the 'JavascriptAddin' icon from the toolbox onto a diagram. The name of your JavascriptAddin Class will be used in generated JavaScript code. It needs to be a valid JavaScript identifier.
4	Locate the Signal Library. Signals are used to define the entry points into your Add-In. If not already in your model, the Signal Library is available for import as a model pattern.
5	Open the receptions list. Add a reception for any Signal that you want to receive. A reasonable starting point would be to include: • EA_Connect • EA_GetMenuItems • EA_MenuClick

6	Open the Behavior window for your Class ('Develop > Source Code > Behavior').
	This shows all the available behavioral features that you can add code to, including the receptions created previously.
	Examples for the signals discussed earlier are:
	
	EA_Connect
	`return "";`
	
	EA_GetMenuItems
	`if(MenuName == "-Example Add-in")`
	`return ["Item 1", "Item 2", "-", "About"];`

	else return "-Example Add-in"; **EA_MenuClick** Session.Prompt("You clicked " + ItemName, 1);
7	Enable your Add-In using the 'Manage Add-Ins' dialog. If security is enabled in your model, this requires model administration rights.
8	You can now test and further develop your Add-In.

Responding to Events

In order for your model Add-In to respond to events, you must define Receptions on the Add-In Class, corresponding to the signals (or events) that you want to handle. You can then define handler code, using JavaScript, for each of the defined Receptions.

You can also define additional functions as Operations on the class, again using JavaScript. These functions can then be called from the Reception handler code.

Define Receptions

Step	Description
1	Select a JavaScriptAddin on a diagram.
2	From the ribbon, select the option "Design > Element > Behavior". The 'Behavior' code editor window is displayed.
3	Ensure that the Structure Tree is visible. Click on the  icon to toggle display of the Structure Tree.
4	Right-click on the class at the top of the Structure Tree. Choose the option 'Add Reception'. The 'Select Signal' dialog is displayed
5	Navigate to where you imported the Signal Reference Library - select the Signal for which you want to add a Reception. Click on the OK button.
6	In the right hand panel, enter JavaScript code to define the required behavior.
7	Repeat steps 4 through 6 for any other signals that you wish to handle.

Edit Add-In Code

The Class 'Behavior' view provides a convenient view for editing the code associated with the behavioral features of your Class.

Access

Ribbon	Develop > Source Code > Behavior

Syntax Highlighting

The Class 'Behavior' view highlights code using the language assigned to the Class. For Model Add-Ins, this should be JavaScript.

Retrieving return values in JavaScript

When handling a reception for an event with OUT/INOUT parameters, values must be read and assigned using the .val attribute of those parameters.

For example, to set the value of the TagValue parameter on the EA_OnElementTagEdit event:

```
TagValue.val = "Hello World!"
```

Adding Operations

Right click on the Class node at the top of the Structure Tree to add a new operation.

All operations should be given names that are valid for JavaScript functions.

All code written will be generated to a function on a JavaScript object. Therefore, to call any function you have written, you will need to prefix it with: *this*.

Model Add-In Management

Model Add-Ins are managed according to the security groups they are assigned to. Each Add-In defaults to 'not loaded' by anyone until it is assigned to a particular security group and either enabled for all users of that group or made optional, which means that each member of the group will need to explicitly enable that Add-In.

Access

Ribbon	Specialize > Add-Ins > Manage Addin

Available Add-Ins

The 'Manage Add-Ins' dialog lists the Add-Ins that are currently enabled for your model, and provides information on each one as explained in this table.

Column	Description
Groups	For projects in which security is enabled, you can select the list of security groups that will be able to access each Add-In. Only users with 'Configure Model Add-Ins' permission can change this column.
Status	This column allows you to select the behavior of each Add-In for users within included security groups. • **Disabled** means that the Add-In can not be used by any users • **Enabled** means that the Add-In is loaded and run for all users in the selected security groups • **Optional** means that each user can choose to enable the Add-In themselves; by default any Add-Ins will be disabled until users enable them Only users with 'Configure Model Add-Ins' permission can change this column.
Load on Startup	This column allows each user to specify that they want to use any optional Add-Ins that are available to their group. If users are not part of a listed group, or the status is not optional, this has no effect.

Signal Reference Library

All the broadcasts Enterprise Architect can send to an add-in are defined in a self-contained pattern that provides an easy way to implement each signal in your model Add-Ins.

Import the Broadcast Types Pattern

Step	Action
1	Click on the  icon and select the 'Management > Model Add-Ins' Perspective. This automatically opens the Start Page 'Create from Pattern' tab (Model Wizard) at the Model Add-Ins Perspective page.
2	Click on the target Package in the Browser window.
3	Click on the 'Broadcast Types' pattern.
4	Click on the Create Model(s) button.

Sample Add-Ins

There are two working examples of Model Add-Ins in the Enterprise Architect Example Model.

These samples demonstrate how to:

- Add code to Receptions
- Call Functions defined as class operations from Reception code
- Use class Attributes as global variables
- Create menus and menu items in an Add-in
- Respond to selection of Add-in menu items

To open the Example Model, select the ribbon option 'Start > Help > Help > Open the Example Model'. Once the Example Model has loaded, search for 'Model Based Add-Ins'.

Enterprise Architect Add-In Model

The Add-In facility provides a means of extending Enterprise Architect, allowing the programmer to enhance the user interface by adding new menus, sub menus, windows and other controls to perform a variety of functions. An Add-In is an ActiveX COM object that is notified of events in the user interface, such as mouse clicks and element selections, and has access to the repository content through the Object Model. Add-Ins can also be integrated with the license management system.

Using this facility, you can extend Enterprise Architect to create new features not available in the core product, and these can be compiled and easily distributed to a community of users within an organization, or more broadly to an entire industry. Using the Add-In facility it is even possible to create support for modeling languages and frameworks not supported in the core product.

Add-Ins have several advantages over stand-alone automation clients:

- Add-Ins can (and should) be written as in-process (DLL) components; this provides lower call overhead and better integration into the Enterprise Architect environment
- Because a current version of Enterprise Architect is already running there is no requirement to start a second copy of Enterprise Architect via the automation interface
- Because the Add-In receives object handles associated with the currently running copy of Enterprise Architect, more information is available about the current user's activity; for example, which diagram objects are selected
- You are not required to do anything other than to install the Add-In to make it usable; that is, you do not have to configure Add-Ins to run on your systems
- Because Enterprise Architect is constantly evolving in response to customer requests, the Add-In interface is flexible
- The Add-In interface does not have its own version, rather it is identified by the version of Enterprise Architect it first appeared in; for example, the current version of the Enterprise Architect Add-In interface is version 2.1
- When creating your Add-In, you do not have to subscribe to a type-library (Add-Ins created before 2004 are no longer supported - if an Add-In subscribes to the Addn_Tmpl.tlb interface (2003 style), it fails on load; in this event, contact the vendor or author of the Add-In and request an upgrade)
- Add-Ins do not have to implement methods that they never use
- Add-Ins prompt users via context menus in the tree view and the diagram
- Menu check and disable states can be controlled by the Add-In

Add-Ins enhance the existing functionality of Enterprise Architect through a variety of mechanisms, such as Scripts, UML Profiles and the Automation Interface. Once an Add-In is registered, it can be managed using the Add-In Manager.

The Add-In Manager

If you want to check what Add-Ins are available on your system, and enable or disable them for use, you can review the 'Add-In Manager' dialog. This dialog lists the Add-Ins that have been registered on your system, and their current status (Enabled or Disabled).

Access

Ribbon	Specialize > Add-Ins > Manage Addin

Enable/Disable Add-Ins

Action	Detail
Enable an Add-In	To enable an Add-In so that it is available for use, select the 'Load on Startup' checkbox corresponding to the name. Click on the OK button. • Any Add-In specific features, facilities and Help are made available through the 'Specialize | <add-in name>' context menu option • Any defined Add-In windows are populated with information; select the 'Specialize > Add-Ins > Addin Windows' ribbon option
Disable an Add-In	To disable an Add-In so that it is not available for use, clear the 'Load on Startup' checkbox corresponding to the name. Click on the OK button. All menu options, features and facilities specific to the Add-In are hidden and made inactive.

Notes

- When you enable or disable an Add-In, you must re-start Enterprise Architect to action the change

Create and Deploy Add-Ins

This topic directs you to information on creating, testing, deploying and managing Add-Ins.

Create an Add-In

Task	Information
Create the Add-In.	Some basic steps on using an IDE to create your Add-in. See the *Create Add-Ins* topic.
Define Menu Items.	Some examples for defining menu items in an Add-in. See the *Define Menu Items* topic.
Respond to Menu Events.	Description and syntax on using the EA_MenuClick. See the *EA_MenuClick* topic.
Handle Add-In Events.	See the *Add-In Events* topic.

Deploy your Add-In

Consideration	Information
Deploy the Add-in	For details on registering the add-in DLL see the Deploy *Add-In* topic.
Tricks and Traps	See the *Tricks and Traps* topic.

Manage Add-Ins

Task	Information
Register an Add-In (developed in-house or brought-in).	Brought-in applications are referred to as Commercial Off The Shelf (COTS) software. See the *Register Add-In* topic.
The Add-In Manager.	See *The Add-In Manager* topic.

Create Add-Ins

Before you start you must have an application development tool that is capable of creating ActiveX COM objects supporting the IDispatch interface, such as:

- Embarcadero Delphi, or Borland Delphi
- Microsoft Visual Basic
- Microsoft Visual Studio .NET

You should consider how to define menu items. To help with this, you could review some examples of Automation Interfaces - examples of code used to create Add-Ins for Enterprise Architect - on the Sparx Systems web page.

Create an Enterprise Architect Add-In

Step	Action
1	Use a development tool to create an ActiveX COM DLL project. Visual Basic users, for example, choose File-Create New Project-ActiveX DLL.
2	Connect to the interface using the syntax appropriate to the language.
3	Create a COM Class and implement each of the general Add-In Events applicable to your Add-In. You only have to define methods for events to respond to.
4	Add a registry key that identifies your Add-In to Enterprise Architect, as described in the Deploy Add-Ins topic.

Define Menu Items

Tasks

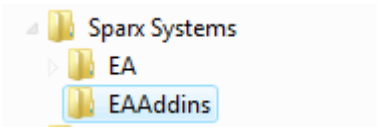
Task	Detail
Define Menu Items	Menu items are defined by responding to the GetMenuItems event. The first time this event is called, MenuName is an empty string, representing the top-level menu. For a simple Add-In with just a single menu option you can return a string. Function EA_GetMenuItems(Repository as EA.Repository, MenuLocation As String, MenuName As String) As Variant EA_GetMenuItems = "&Joe's Add-In" End Function
Define Sub-Menus	To define sub-menus, prefix a parent menu with a dash. Parent and sub-items are defined in this way: Function EA_GetMenuItems(Repository as EA.Repository, MenuLocation As String, MenuName As String) As Variant Select Case MenuName Case "" 'Parent Menu Item EA_GetMenuItems = "-&Joe's Add-In" Case "-&Joe's Add-In" 'Define Sub-Menu Items using the Array notation. 'In this example, "Diagram" and "Treeview" compose the "Joe's Add-In" sub-menu. EA_GetMenuItems = Array("&Diagram", "&Treeview") Case Else MsgBox "Invalid Menu", vbCritical End Select End Function
Define Further Sub-Menus	Similarly, you can define further sub-items: Function EA_GetMenuItems(Repository as EA.Repository, MenuLocation As String, MenuName As String) As Variant Select Case MenuName Case "" EA_GetMenuItems = "-Joe's Add-In" Case "-Joe's Add-In" EA_GetMenuItems = Array("-&Diagram", "&TreeView") Case "-&Diagram" EA_GetMenuItems = "&Properties" Case Else MsgBox "Invalid Menu", vbCritical End Select

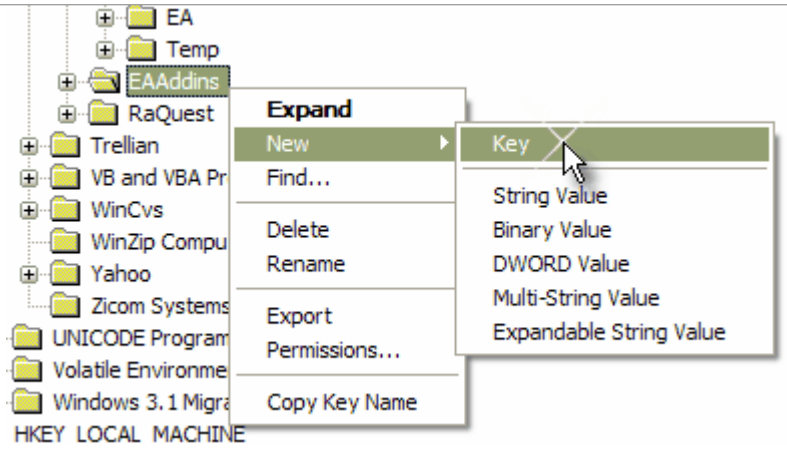
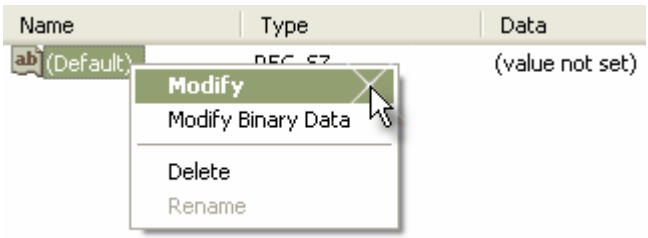
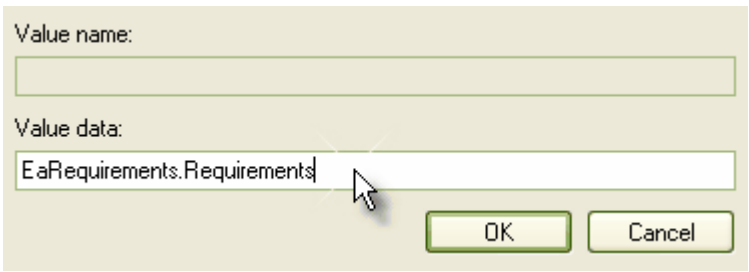
	End Function

Enable/Disable menu options	To enable or disable menu options by default, you can use this method to show particular items to the user:
	Sub EA_GetMenuState(Repository As EA.Repository, Location As String, MenuName As String, ItemName As String, IsEnabled As Boolean, IsChecked As Boolean)
	Select Case Location
	Case "TreeView"
	'Always enable
	Case "Diagram"
	'Always enable
	Case "MainMenu"
	Select Case ItemName
	Case "&Translate", "Save &Project"
	If GetIsProjectSelected() Then
	IsEnabled = False
	End If
	End Select
	End Select
	IsChecked = GetIsCurrentSelection()
	End Sub

Deploy Add-Ins

Deploy Add-Ins to Users' Sites

Step	Action
1	Add the Add-In DLL file to an appropriate directory on the user's computer; that is: C:\Program Files\(new dir)
2	Register the DLL as appropriate to your platform: • If compiled as a native Win32 DDL, such as VB or C++, register the DDL using the regsvr32 command from the command prompt regsvr32 "C:\Program Files\MyCompany\EAAddin\EAAddin.dll" • If compiled as a .NET DLL, such as C# or VB.NET, register the DLL using the RegAsm command from the command prompt C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\RegAsm.exe "C:\Program Files\MyCompany\EAAddin\EAAddin.dll" /codebase
3	Place a new entry into the registry using the registry editor (run regedit) so that Enterprise Architect recognizes the presence of your Add-In.
4	Add a new key 'EAAddIns' under one of these locations: • For the current user only - On Enterprise Architect 32 bit [HKEY_CURRENT_USER\Software\Sparx Systems\EAAddins] - On Enterprise Architect 64 bit [HKEY_CURRENT_USER\Software\Sparx Systems\EAAddins64] • For multiple users on a machine - On Enterprise Architect 32 bit [HKEY_LOCAL_MACHINE\Software\Sparx Systems\EAAddins] - On Enterprise Architect 64 bit [HKEY_LOCAL_MACHINE\Software\Sparx Systems\EAAddins64]  Note: the Enterprise Architect 32 and 64 bit editions will only attempt to load Add-Ins under the corresponding key - EAAddIns or EAAddIns64, respectively.
5	Add a new key under this key with the project name.

	 (ProjectName) is not necessarily the name of your DLL, but the name of the Project; in Visual Basic, this is the value for the property Name corresponding to the project file.
6	Specify the default value by modifying the default value of the key. 
7	Enter the value of the key by typing in the (project name).(class name), such as: EaRequirements.Requirements where *EaRequirements* is the project name, as shown in this example: 

Tips and Tricks

Considerations

Item	Detail
GUID	The registration GUID and name used to register an Add-In should be the same for Enterprise Architect 32 bit and 64 bit versions. Only the name and/or location of the DLL should be different.
Enterprise Architect 64 bit and C++ Add-Ins	Apart from using the correct registration key (see step 4 in the *Deploy Ad-Ins* Help topic) there is no special configuration for getting a 64 bit COM object running under Enterprise Architect 64 bit.
.NET Add-Ins	When generating a .NET assembly, you must explicitly set the 'Target Platform' to x86/x64. Leaving it on 'Any CPU' could cause issues when Enterprise Architect 32 bit is run on a 64 bit version of windows. *Add a x64 Target to your project and rebuild the project.* Visual Studio has an issue when attempting to register a .NET assembly when the Interop.EA is included in a project. It will attempt to use regasm for the architecture Interop.EA was built for. We have found adding to the Post-Build Script helps, if you uncheck the 'Register for COM interop' option in the project settings (assuming you have permission to write to the registry). if $(PlatformName) == x64 ("%Windir%\Microsoft.NET\Framework64\v4.0.30319\regasm" "$(TargetPath)") if $(PlatformName) == x86 ("%Windir%\Microsoft.NET\Framework64\v4.0.30319\regasm" "$(TargetPath)") Note: At the time of writing, .NET Add-ins will not work under Wine and using Wine-Mono.
Java API	The Java API loads the last installed Enterprise Architect and isn't affected when using either the 32 or 64 version of the dll, as long as the SSJavaCOM DLL can be found by the Java runtime.
Visual Basic 5/6 Users Note	Visual Basic users should note that the version number of the Enterprise Architect interface is stored in the VBP project file in a form similar to this: Reference=*\G{64FB2BF4-9EFA-11D2-8307-C45586000000}#2.2#0#..\..\..\..\Program Files\Sparx Systems\EA\EA.TLB#Enterprise Architect Object Model 2.02 If you experience problems moving from one version of Enterprise Architect to another, open the VBP file in a text editor and remove this line. Then open the project in Visual Basic and use Project-References to create a new reference to the Enterprise Architect Object model.
Holding State Information	It is possible for an Add-In to hold state information, meaning that data can be stored in member variables in response to one event and retrieved in another. There are some dangers in doing this: • Enterprise Architect Automation Objects do not update themselves in response to user activity, to activity on other workstations, or even to the actions of other objects in the same automation client; retaining handles to such objects between calls can result in the second event querying objects that have no

	relationship with the current state of Enterprise Architect • When you close Enterprise Architect, all Add-Ins are asked to shut down; if there are any external automation clients Enterprise Architect must stay active, in which case all the Add-Ins are reloaded, losing all the data • Enterprise Architect acting as an automation client does not close if an Add-In still holds a reference to it (releasing all references in the Disconnect() event avoids this problem) It is recommended that unless there is a specific reason for doing so, the Add-In should use the repository parameter and its method and properties to provide the necessary data.
Enterprise Architect Not Closing	.NET Specific Issues Automation checks the use of objects and will not allow any of them to be destroyed until they are no longer being used. As noted in the *Automation Interface* topic, if your automation controller was written using the .NET framework, Enterprise Architect does not close even after you release all your references to it. To force the release of the COM pointers, call the memory management functions as shown: GC.Collect(); GC.WaitForPendingFinalizers(); Additionally, because automation clients hook into Enterprise Architect, which creates Add-Ins that in turn hook back into Enterprise Architect, it is possible to get into a deadlock situation where Enterprise Architect and the Add-Ins will not let go of one another and keep each other active. An Add-In might retain hooks into Enterprise Architect because: • It keeps a private reference to an Enterprise Architect object (see the earlier *Holding State Information*), or • It has been created by .NET and the GC mechanism has not yet released it There are two actions required to avoid deadlock situations: • Automation controllers must call Repository.CloseAddins() at some point (perhaps at the end of processing) • Add-Ins must release all references to Enterprise Architect in the Disconnect() event; see the *Add-In Events* topic for details It is possible that your Automation client controls a running instance of Enterprise Architect where the Add-Ins have not complied with the rules. In this case you could call Repository.Exit() to terminate Enterprise Architect. **Miscellaneous** In developing Add-Ins using the .NET framework you must select COM Interoperability in the project's properties in order for it to be recognized as an Add-In. Some development environments do not automatically register COM DLLs on creation. You might have to do that manually before Enterprise Architect recognizes the Add-In. You can use your private Add-In key (as required for Add-In deployment) to store configuration information pertinent to your Add-In.

Add-In Search

Enterprise Architect enables Extensions to integrate with the Model Search. Searches can be defined that execute a method within your Add-In and display your results in an integrated way.

Details

Item
The method that runs the search must be structured in this way.
Defines the XML structure expected by Enterprise Architect to specify search results.
In addition to the displayed results, two additional hidden fields can be passed into the XML that provide special functionality. • CLASSTYPE - Returning a field of CLASSTYPE, containing the Object_Type value from the t_object table, displays the appropriate icon in the column in which you place the field • CLASSGUID - Returning a field of CLASSGUID, containing an ea_guid value, enables the Model Search to track the object in the Browser window and open the Properties window for the element by double-clicking in the Model Search

EA_SampleSearch

This defines the signature required for the function Enterprise Architect calls when executing an Add-In search. The name can be changed to any valid function name in your target programming language.

Syntax

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the Enterprise Architect model about to be closed. Poll its members to retrieve model data and user interface status information.
SearchText	String Direction: IN Description: Provides the value (if any) entered by the user in the search term field in the model search window.
XMLResults	String Direction: OUT Description: Provides the value (if any) entered by the user in the search term field in the model search window.

Return Value

The method must return any non-empty value for the results to be displayed.

XML Format (Search Data)

This example XML provides the format for the sSearchData parameter of the RunModelSearch method.

```
<ReportViewData UID=\"MySearchID\">
    <!--
    //The UID attribute enables XML type searches to persist column information. That is, if you run the search, group by column or adjust
    //column widths, then close the window and run the search again, the format/organization changes are retained. To avoid persisting column
    //arrangements, leave the attribute value blank or remove it altogether. Use this section to declare all possible fields - columns that appear
    //in Enterprise Architect's Search window - that are used below in <Rows/>. The order of the columns of information to be appended here must
    //match the order that the search run in Enterprise Architect would normally display. Furthermore, if you append results onto a custom SQL
    //Search, then the order used in your Custom SQL must match the order used here.
    -->
  <Fields>
    <Field name=""/>
    <Field name=""/>
    <Field name=""/>
    <Field name=""/>
  </Fields>
  <Rows>
    <Row>
      <Field name="" value=""/>
      <Field name="" value=""/>
      <Field name="" value=""/>
      <Field name="" value=""/>
    </Row>
    <Row>
      <Field name="" value=""/>
      <Field name="" value=""/>
      <Field name="" value=""/>
      <Field name="" value=""/>
    </Row>
    <Row>
      <Field name="" value=""/>
      <Field name="" value=""/>
      <Field name="" value=""/>
      <Field name="" value=""/>
    </Row>
  </Rows>
```

```
</ReportViewData>
```

Add-In Events

All Enterprise Architect Add-Ins can choose to respond to general Add-In events.

Events

Event
EA_Connect - Add-Ins can use this to identify their type and to respond to Enterprise Architect start up.
EA_Disconnect - Add-Ins can use this to respond to user requests to disconnect the model branch from an external project.
EA_GetMenuItems - Add-Ins can use this to provide the Enterprise Architect user interface with additional Add-In menu options in various context menus.
EA_GetMenuState - Add-Ins can use this to set a particular menu option to either enabled or disabled.
EA_GetRibbonCategory - Add-Ins can use this to identify the Ribbon panel in which to house their calling icon.
EA_MenuClick - received by an Add-In in response to user selection of a menu option.
EA_OnOutputItemClicked - informs Add-Ins that the user has clicked on a list entry in the system tab or one of the user defined output tabs.
EA_OnOutputItemDoubleClicked - informs Add-Ins that the user has used the mouse to double-click on a list entry in one of the user-defined output tabs.
EA_ShowHelp - Add-Ins can use this to show a Help topic for a particular menu option.

EA_OnAddinPropertiesTabChanging

Indicates that a value in a properties list added via Repository.AddPropertiesTab has been changed by the user.

Syntax

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects describing the field changed: • TabName: The name of the Add-Ins window tab changing • PropID: Unique ID assign to Property item within the xml definition. • ChangeValue: The value the Property is changing to. • OriginalValue: The original value assigned to the Property

Return Value

- Return False to indicate that this change was rejected
- Return True to indicate that the change is accepted

EA_Connect

Add-Ins can use EA_Connect events to identify their type and to respond to Enterprise Architect start up.

This event occurs when Enterprise Architect first loads your Add-In. Enterprise Architect itself is loading at this time so that while a Repository object is supplied, there is limited information that you can extract from it.

There are two key uses for EA_Connect:

- Initializing global Add-In data, along with identifying the Add-In as an MDG Add-In
- Initializing a Workflow script.

Syntax

Function EA_Connect (Repository As EA.Repository) As String

The EA_Connect function syntax has this parameter:

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

Return Value

A string identifying a specialized type of Add-In:

Type	Details
"MDG"	MDG Add-Ins receive MDG Events and extra menu options.
"Workflow"	Workflow add-ins receive additional events to control user ability to change specific fields.
""	A non-specialized Add-In.

EA_Disconnect

Add-Ins can use the EA_Disconnect event to respond to user requests to disconnect the model branch from an external project.

This function is called when Enterprise Architect closes. If you have stored references to Enterprise Architect objects (not recommended anyway), you must release them here.

In addition, .NET users must call memory management functions as shown:

```
GC.Collect();
GC.WaitForPendingFinalizers();
```

Syntax

```
Sub EA_Disconnect()
```

Return Value

None.

EA_GetMenuItems

The EA_GetMenuItems event enables the Add-In to provide the Enterprise Architect user interface with additional Add-In menu options in various context menus. When a user selects an Add-In menu option, an event is raised and passed back to the Add-In that originally defined that menu option.

This event is raised just before Enterprise Architect has to show particular menu options to the user, and its use is described in the *Define Menu Items* topic.

Syntax

Function EA_GetMenuItems (Repository As EA.Repository, MenuLocation As String, MenuName As String) As Variant

The EA_GetMenuItems function syntax has these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
MenuLocation	String Direction: IN Description: A string representing the part of the user interface that brought up the context menu. This can be TreeView, MainMenu, Diagram or Other. You can add further values for MenuLocation at any time. A MenuLocation of 'TreeView' would indicate that the menu was displayed in the Browser window; 'MainMenu' would indicate that the menu was displayed from a ribbon option, and 'Diagram' that the menu was displayed within a diagram. 'Other' would indicate an unspecified location, which might be one of these: • Calendar • Dialog • Element List • Gantt • Model View • Project View • Relationship Matrix • Reviews • Search • Specification Manager
MenuName	String Direction: IN Description: The name of the parent menu for which sub-items are to be defined. In the case of the top-level menu this is an empty string.

Return Value

One of these types:

- A string indicating the label for a single menu option
- An array of strings indicating multiple menu options
- Empty (Visual Basic/VB.NET) or null (C#) to indicate that no menu should be displayed

In the case of the top-level menu it should be a single string or an array containing only one item, or empty/null.

EA_GetMenuState

Add-Ins can use the EA_GetMenuState event to set a particular menu option to either enabled or disabled. This is useful when dealing with locked Packages and other situations where it is convenient to show a menu option, but not enable it for use.

This event is raised just before Enterprise Architect has to show particular menu options to the user. Its use is further described in the *Define Menu Items* topic.

Syntax

Sub EA_GetMenuState (Repository as EA.Repository, MenuLocation As String, MenuName as String, ItemName as String, IsEnabled as Boolean, IsChecked as Boolean)

The EA_GetMenuState function syntax has these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
MenuLocation	String Direction: IN Description: A string representing the part of the user interface that brought up the menu. This can be TreeView, MainMenu or Diagram.
MenuName	String Direction: IN Description: The name of the parent menu for which sub-items must be defined. In the case of the top-level menu it is an empty string.
ItemName	String Direction: IN Description: The name of the option actually clicked; for example, 'Create a New Invoice'.
IsEnabled	Boolean Direction: OUT Description: Set to False to disable this particular menu option.
IsChecked	Boolean Direction: OUT Description: Set to True to check this particular menu option.

Return Value

None.

EA_GetRibbonCategory

Add-Ins can use EA_GetRibbonCategory events to identify the Ribbon in which the Add-In should place its menu icon.

This event occurs when Enterprise Architect first loads your Add-In. Enterprise Architect itself is loading at this time so that while a Repository object is supplied, there is limited information that you can extract from it.

The chief use for EA_GetRibbonCategory is in initializing the Add-In access point.

Syntax

Function EA_GetRibbonCategory (Repository As EA.Repository) As String

The EA_GetRibbonCategory function syntax has this parameter:

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

Return Value

A string matching the name of the selected ribbon (in English if you are using a translated version). The possible names are:

- Start
- Design
- Layout
- Publish
- Specialize
- Construct
- Code
- Simulate
- Execute
- Manage

It is not possible to include Add-Ins in the 'Specification - Specify' ribbon or 'Documentation - Edit' ribbon.

If the function isn't implemented (or if an invalid name is returned) the 'Add-In' menu will be available from the 'Specialize' ribbon, 'Add-Ins' panel.

EA_MenuClick

EA_MenuClick events are received by an Add-In in response to user selection of a menu option.

The event is raised when the user clicks on a particular menu option. When a user clicks on one of your non-parent menu options, your Add-In receives a MenuClick event, defined as:

```
Sub EA_MenuClick(Repository As EA.Repository, ByVal MenuLocation As String, ByVal MenuName As String, ByVal ItemName As String)
```

This code is an example of use:

```
If MenuName = "-&Diagram" And ItemName = "&Properties" then
    MsgBox Repository.GetCurrentDiagram.Name, vbInformation
Else
    MsgBox "Not Implemented", vbCritical
End If
```

Notice that your code can directly access Enterprise Architect data and UI elements using Repository methods.

Syntax

Sub EA_MenuClick (Repository As EA.Repository, MenuLocation As String, MenuName As String, ItemName As String)

The EA_GetMenuClick function syntax has these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
MenuLocation	String Direction: IN Description: A string representing the part of the user interface that brought up the menu. This can be TreeView, MainMenu or Diagram.
MenuName	String Direction: IN Description: The name of the parent menu for which sub-items are to be defined. In the case of the top-level menu this is an empty string.
ItemName	String Direction: IN Description: The name of the option actually clicked; for example, 'Create a New Invoice'.

Return Value

None.

EA_OnOutputItemClicked

EA_OnOutputItemClicked events inform Add-Ins that the user has clicked on a list entry in the system tab or one of the user defined output tabs.

Usually an Add-In responds to this event in order to capture activity on an output tab they had previously created through a call to Repository.AddTab().

Note that every loaded Add-In receives this event for every click on an output tab in Enterprise Architect, irrespective of whether the Add-In created that tab. Add-Ins should therefore check the TabName parameter supplied by this event to ensure that they are not responding to other Add-Ins' events.

Syntax

EA_OnOutputItemClicked (Repository As EA.Repository, TabName As String, LineText As String, ID As Long)

The EA_OnOutputItemClicked function syntax has these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
TabName	String Direction: IN Description: The name of the tab that the click occurred in. Usually this would have been created through 'Repository.AddTab()'.
LineText	String Direction: IN Description: The text that had been supplied as the String parameter in the original call to 'Repository.WriteOutput()'.
ID	Long Direction: IN Description: The ID value specified in the original call to Repository.WriteOutput().

Return Value

None.

EA_OnOutputItemDoubleClicked

EA_OnOutputItemDoubleClicked events inform Add-Ins that the user has used the mouse to double-click on a list entry in one of the user-defined output tabs.

Usually an Add-In responds to this event in order to capture activity on an output tab they had previously created through a call to Repository.AddTab().

Note that every loaded Add-In receives this event for every double-click on an output tab in Enterprise Architect, irrespective of whether the Add-In created that tab; Add-Ins should therefore check the TabName parameter supplied by this event to ensure that they are not responding to other Add-Ins' events.

Syntax

EA_OnOutputItemDoubleClicked (Repository As EA.Repository, TabName As String, LineText As String, ID As Long)

The EA_OnOutputItemClicked function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model; poll its members to retrieve model data and user interface status information.
TabName	String Direction: IN Description: The name of the tab that the click occurred in; usually this would have been created through 'Repository.AddTab()'.
LineText	String Direction: IN Description: The text that had been supplied as the String parameter in the original call to 'Repository.WriteOutput()'.
ID	Long Direction: IN Description: The ID value specified in the original call to Repository.WriteOutput().

Return Value

None.

EA_ShowHelp

Add-Ins can use the EA_ShowHelp event to show a Help topic for a particular menu option. When the user has an Add-In menu option selected, pressing F1 can be related to the required Help topic by the Add-In and a suitable Help message shown.

This event is raised when the user presses F1 on a menu option that is not a parent menu.

Syntax

Sub EA_ShowHelp (Repository as EA.Repository, MenuLocation As String, MenuName as String, ItemName as String)

The EA_ShowHelp function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
MenuLocation	String Direction: Description: A string representing the part of the user interface that brought up the menu. This can be Treeview, MainMenu or Diagram.
MenuName	String Direction: Description: The name of the parent menu for which sub-items are to be defined. In the case of the top-level menu this is an empty string.
ItemName	String Direction: Description: The name of the option actually clicked; for example, 'Create a New Invoice'.

Return Value

None.

Broadcast Events

Overview

Broadcast events are sent to all loaded Add-Ins. For an Add-In to receive the event, they must first implement the required automation event interface. If Enterprise Architect detects that the Add-In has the required interface, the event is dispatched to the Add-In.

MDG Events add a number of additional events, but the Add-In must first have registered as an MDG-style Add-In, rather than as a generic Add-In.

Event Type
Add-In License Management Events
Custom Table Events
Compartment Events
Context Item Events
File Close Event
File New Event
File Open Event
Model Validation Events
On Tab Changed Event
Post Close Diagram Event
Post Initialization Event
Post New Events
Post Open Diagram Event
Pre-Deletion Events
Pre-Exit Instance (not currently used)
On the creation of new objects
Retrieve Model Template Event
Schema Composer Events
Tagged Value Events

Technology Events
Transformation Event

Add-In License Management Events

Enterprise Architect Add-Ins can respond to events associated with Add-In License Management.

License Management Events

Event
EA_AddinLicenseValidate
EA_AddinLicenseGetDescription
EA_GetSharedAddinName

EA_AddinLicenseValidate

When a user directly enters into the 'License Management' dialog a license key that doesn't match a Sparx Systems key, EA_AddInLicenseValidate is broadcast to all Enterprise Architect Add-Ins, providing them with a chance to use the Add-In key to determine the level of functionality to provide. When a key is retrieved from the Sparx Systems Keystore only the target Add-In will be called with the key.

For the Add-In to validate itself against this key, the Add-In's EA_AddinLicenseValidate handler should return confirmation that the license has been validated. As the EA_AddinLicenseValidate event is broadcast to all Add-Ins, one license can validate many Add-Ins.

If an Add-In elects to handle a license key by returning a confirmation to EA_AddinLicenseValidate, it is called upon to provide a description of the license key through the EA_AddinLicenseGetDescription event. If more than one Add-In elects to handle a license key, the first Add-In that returns a confirmation to EA_AddinLicenseValidate is queried for the license key description.

Syntax

Function EA_AddInLicenseValidate (Repository As EA.Repository, AddinKey As String) As Boolean

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
AddinKey	String Direction: IN Description: The Add-In license key that has been entered in the 'License Management' dialog.

Return Value

Returns True if the license key is validated for the current Add-In. Returns False otherwise.

EA_AddinLicenseGetDescription

Before the Enterprise Architect 'License Management' dialog is displayed, EA_AddInLicenseGetDescription is sent once for each Add-In key to the first Add-In that elected to handle that key.

The value returned by EA_AddinLicenseGetDescription is used as the key's plain text description.

Syntax

Function EA_AddinLicenseGetDescription (Repository as EA.Repository, AddinKey as String) As String

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open model. Poll its members to retrieve model data and user interface status information.
AddinKey	String Direction: IN Description: The Add-In license key that Enterprise Architect requires a description for.

Return Value

A String containing a plain text description of the provided AddinKey.

EA_GetSharedAddinName

As an Add-In writer you can distribute keys to your Add-In via the Enterprise Architect Keystore, provided that your keys are added using a prefix that allows the system to identify the Add-In to which they belong. EA_GetSharedAddinName is called to determine what prefix the Add-In is using. If a matching key is found in the keystore the 'License Management' dialog will display the name returned by EA_AddinLicenseGetDescription to your users. Finally, when the user selects a key, that key will be passed to your Add-In to validate by calling EA_AddinLicenseValidate.

Syntax

Function EA_GetSharedAddinName (Repository as EA.Repository) As String

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open model. Poll its members to retrieve model data and user interface status information.

Return Value

A String containing a product name code for the provided Add-In, such as MYADDIN. This will be shown in plain text in any keys added to the keystore.

Notes

Shared Add-In keys have the format:

EASK-YOURCODE-REALKEY

- EASK - Constant string that identifies a shared key for an Enterprise Architect Add-In
- YOURCODE - The code you select and verify with us:
  - Displayed to the administrator of the keystore
  - Recommended length of 6-10 characters
  - Contains ASCII characters 33-126, except for '-' (45)
- REALKEY - Encoding of the actual key or checksums
  - Recommended length of 8-32 characters
  - Contains ASCII characters 33-126

We recommend that you contact Sparx Systems directly with proposed values to ensure that you don't clash with any other Add-Ins.

For example, these keys would all be interpreted as belonging to an Add-In returning MYADDIN from this function:

- EASK-MYADDIN-Test
- EASK-MYADDIN-{7AC4D426-9083-4fa2-93B7-25E2B7FB8DC5}
- EASK-MYADDIN-7AC4D426-9083-4fa2-93B7
- EASK-MYADDIN-25E2B7FB8DC5
- EASK-MYADDIN-2hDfHKA5jf0GAjn92UvqAnxwC13dxQGJtH7zLHJ9Ym8=

Custom Table Events

The Custom Table element has an Operation called 'script', reserved for script execution, that can be used in two different, mutually exclusive ways, either:

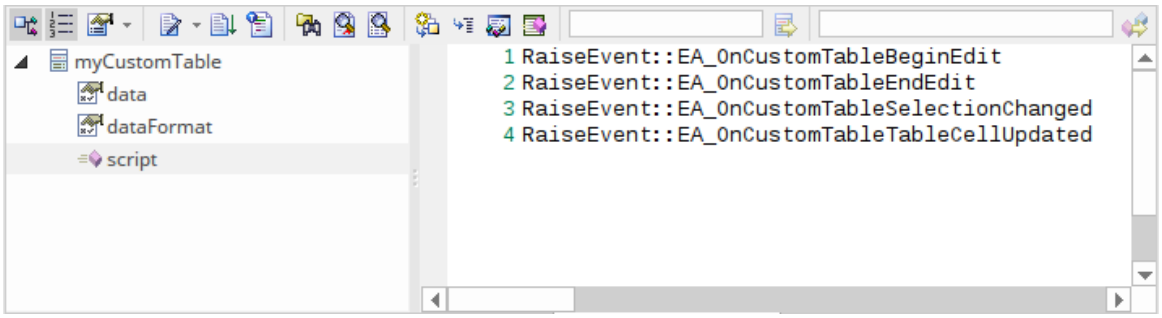
- To contain a script in JavaScript that can be executed from the element context menu; see the *Custom Table Artifact* Help topic, or
- To contain RaiseEvent broadcast calls to trigger actions from an Add-In written to read or update the Custom Table

Broadcasts

There are four reserved Add-In broadcast events that can only be enabled by listing the event in the 'script' Operation of the Custom Table element. To raise the broadcast events, list any or all of these broadcast calls in the operation named 'script'.

Syntax:

RaiseEvent::EA_OnCustomTableBeginEdit



EA_OnCustomTableBeginEdit

EA_OnCustomTableBeginEdit notifies Add-Ins that the Custom Table is beginning edit mode. This broadcast event can only be enabled by the Custom Table's operation 'script' behavior.

Syntax

Function EA_OnCustomTableBeginEdit (Repository As EA.Repository, Info As EA.EventProperties)

The EA_OnCustomTableBeginEdit function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the Custom Table that is under edit: • ObjectID - A long value corresponding to the ElementID of the object

EA_OnCustomTableEndEdit

EA_OnCustomTableEndEdit notifies Add-Ins that a Custom Table element is ending edit mode. This broadcast event can only be enabled by the Custom Table's operation 'script' behavior.

Syntax

Function EA_OnCustomTableEndEdit (Repository As EA.Repository, Info As EA.EventProperties)

The EA_OnCustomTableEndEdit function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the Custom Table that is under edit: • ObjectID - A long value corresponding to the ElementID of the object

Return Value

This function allows validation of the table data, and returns a Boolean value:

- True to save the current data in the grid, or
- False to abandon the current data

EA_OnCustomTableSelectionChanged

EA_OnCustomTableSelectionChanged notifies Add-Ins that a cell of the Custom Table has changed. This broadcast event can only be enabled by the Custom Table's operation 'script' behavior.

Syntax

Function EA_OnCustomTableSelectionChanged (Repository As EA.Repository, Info As EA.EventProperties)

The EA_OnCustomTableSelectionChanged function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the Custom Table that has been changed: • ObjectID - A long value corresponding to the ElementID of the object • RowID - A long value corresponding to the selected row id • ColID - A long value corresponding to the selected column id

EA_OnCustomTableCellUpdated

EA_OnCustomTableCellUpdated notifies Add-Ins that a cell value has been updated. This broadcast event can only be enabled by the Custom Table's operation 'script' behavior.

Syntax

Function EA_OnCustomTableCellUpdated (Repository As EA.Repository, Info As EA.EventProperties)

The EA_OnCustomTableCellUpdated function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the Custom Table cell that has been changed: • ObjectID - A long value corresponding to the ElementID of the object • RowID - A long value corresponding to the selected row id • ColID - A long value corresponding to the selected column id • Value - A variant value of the changed cell data

Schema Composer Events

Enterprise Architect Add-Ins can respond to events associated with the Schema Composer to provide custom schema export formats.

The requirements for an Add-In to participate consist of implementing these three functions:

- EA_IsSchemaExporter
- EA_GetProfileInfo
- EA_GenerateFromSchema

EA_GenerateFromSchema

Respond to a 'Generate' request from the Schema Composer when using the profile type specified by the EA_IsSchemaExporter event. The SchemaComposer object can be used to traverse the schema. Export formats that have been requested by the user for generation will be listed in the exports parameter.

Syntax

Sub EA_GenerateFromSchema (Repository as EA.Repository, composer as EA.SchemaComposer, exports as String)

Parameter	Details
Repository	Type: EA.Repository Direction: IN Description: An EA.Repository object representing the currently open model. Poll its members to retrieve model data and user interface status information.
composer	Type: EA.SchemaComposer Direction: IN Description: Provides access to the types defined in the schema currently being generated. Use the *SchemaTypes* attribute to enumerate through the types and output to the appropriate export format.
exports	Type: String Direction: IN Description: Comma-separated list of export formats that the user has requested in the 'Generate' dialog.

Return Value

None.

EA_GetProfileInfo

Add-Ins can optionally implement this function to define the capabilities of the Schema Composer when working with the profile type specified by the EA_IsSchemaExporter event.

Syntax

Sub EA_GetProfileInfo (Repository as EA.Repository, profile as EA.SchemaProfile)

Parameter	Details
Repository	Type: EA.Repository Direction: IN Description: An EA.Repository object representing the currently open model. Poll its members to retrieve model data and user interface status information.
profile	Type: EA.SchemaProfile Direction: IN Description: An EA.SchemaProfile object representing the currently active profile type. Call the *SetCapability* function to enable or disable various capabilities of the Schema Composer. Call the *AddExportFormat* function to define additional export formats that this profile will support.

Return Value

None.

EA_IsSchemaExporter

Enterprise Architect Add-Ins can integrate with the Schema Composer by providing alternatives to offer users for the generation of schemas and sub models.

The Add-In must implement this function to be listed in the Schema Composer.

Syntax

Function EA_IsSchemaExporter(Repository as EA.Repository, ByRef displayName as String) As Boolean

Parameter	Details
Repository	Type: EA.Repository Direction: IN Description: An EA.Repository object representing the currently open model. Poll its members to retrieve model data and user interface status information.
displayName	Type: String Direction: OUT Description: The name of the custom schema set that will be provided by this Add-In.

Return Value

Return True to indicate that this Add-In will provide schema export functionality and be listed as a Schema Set when defining a new profile in the Schema Composer.

Compartment Events

Enterprise Architect Add-Ins can respond to various events associated with user-generated element compartments.

Compartment Broadcast Events

Event
EA_QueryAvailableCompartments
EA_GetCompartmentData

EA_QueryAvailableCompartments

This event occurs when Enterprise Architect's diagrams are refreshed. It is a request for the Add-In to provide a list of user-defined compartments.

The EA_GetCompartmentData event then queries each object for the data to display in each user-defined compartment.

Syntax

Function EA_QueryAvailableCompartments (Repository As EA.Repository) As Variant

The EA_QueryAvailableCompartments function syntax contains this parameter.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

Return Value

A String containing a comma-separated list of user-defined compartments.

Example

```
Function EA_QueryAvailableCompartments(Repository As EA.Repository) As Variant
    Dim sReturn As String
    sReturn = ""
    If m_FirstCompartmentVisible = True Then
        sReturn = sReturn + "first,"
    End If
    If m_SecondCompartmentVisible = True Then
        sReturn = sReturn + "second,"
    End If
    If m_ThirdCompartmentVisible = True Then
        sReturn = sReturn + "third,"
    End If
    If Len(sReturn) > 0 Then
        sReturn = Left(sReturn, Len(sReturn)-1)
    End If
    EA_QueryAvailableCompartments = sReturn
End Function
```

EA_GetCompartmentData

This event occurs when Enterprise Architect is instructed to redraw an element. It requests that the Add-In provide the data to populate the element's compartment.

Syntax

Function EA_GetCompartmentData (Repository As EA.Repository, sCompartment As String, sGUID As String, oType As EA.ObjectType) As Variant

The EA_QueryAvailableCompartments function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
sCompartment	String Direction: IN Description: The name of the compartment for which data is being requested.
sGUID	String Direction: IN Description: The GUID of the element for which data is being requested.
oType	ObjectType Direction: IN Description: The type of the element for which data is being requested.

Return Value

A variant containing a formatted string. The format is illustrated in this example:

Example

```
Function EA_GetCompartmentData(Repository As EA.Repository, sCompartment As String, sGUID As String, oType As EA.ObjectType) As Variant


    If Repository Is Nothing Then
        Exit Function
    End If
```

```
Dim sCompartmentData As String
Dim oXML As MSXML2.DOMDocument
Dim Nodes As MSXML2.IXMLDOMNodeList
Dim Node1 As MSXML2.IXMLDOMNode
Dim Node As MSXML2.IXMLDOMNode
Dim sData As String

sCompartmentData = ""
Set oXML = New MSXML2.DOMDocument
sData = ""
On Error GoTo ERR_GetCompartmentData
oXML.loadXML (Repository.GetTreeXMLByGUID(sGUID))
Set Node1 = oXML.selectSingleNode("//ModelItem")
If Node1 Is Nothing Then
    Exit Function
End If

sCompartmentData = sCompartmentData + "Name=" + sCompartment + ";"
sCompartmentData = sCompartmentData + "OwnerGUID=" + sGUID + ";"
sCompartmentData = sCompartmentData + "Options=SkipIfOnDiagram&_eq_^1&_sc_^"
Select Case sCompartment
Case "parts"
Set Nodes = Node1.selectNodes("ModelItem(@Metatype=""Part"")")
For Each Node In Nodes
    sData = sData + "Data&_eq_^" + Node.Attributes.getNamedItem("Name").nodeValue + "&_sc_^"
    sData = sData + "GUID&_eq_^" + Node.Attributes.getNamedItem("GUID").nodeValue + "&_sc_^,"
    Next
Case "ports"
Set Nodes = Node1.selectNodes("ModelItem(@Metatype=""Port"")")
For Each Node In Nodes
    sData = sData + "Data&_eq_^" + Node.Attributes.getNamedItem("Name").nodeValue + "&_sc_^"
    sData = sData + "GUID&_eq_^" + Node.Attributes.getNamedItem("GUID").nodeValue + "&_sc_^,"
    Next
End Select
If there is no data to display, then don't return any compartment data
If sData <> "" Then
    sCompartmentData = sCompartmentData + "CompartmentData=" + sData + ";"
Else
    sCompartmentData = ""
End If
EA_GetCompartmentData = sCompartmentData
Exit Function
```

```
ERR_GetCompartmentData:
EA_GetCompartmentData = ""
End Function
```

Context Item Events

Enterprise Architect Add-Ins can respond to events associated with changing context.

Context Item Broadcast Events

Event
EA_OnContextItemChanged
EA_OnContextItemDoubleClicked
EA_OnNotifyContextItemModified

EA_OnContextItemChanged

EA_OnContextItemChanged notifies Add-Ins that a different item is now in context.

This event occurs after a user has selected an item anywhere in the Enterprise Architect GUI. Add-Ins that require knowledge of the current item in context can subscribe to this broadcast function. If ot = otRepository, then this function behaves in the same way as EA_FileOpen.

Syntax

Sub EA_OnContextItemChanged (Repository As EA.Repository, GUID As String, ot as EA.ObjectType)

The EA_OnContextItemChanged function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
GUID	String Direction: IN Description: Contains the GUID of the new context item. The value corresponds to these properties, depending on the value of the ot parameter: • ot (ObjectType) - GUID value • otElement - Element.ElementGUID • otPackage - Package.PackageGUID • otDiagram - Diagram.DiagramGUID • otAttribute - Attribute.AttributeGUID • otMethod - Method.MethodGUID • otConnector - Connector.ConnectorGUID • otRepository - NOT APPLICABLE, the GUID is an empty string
ot	EA.ObjectType Direction: IN Description: Specifies the type of the new context item.

Return Value

None.

EA_OnContextItemDoubleClicked

EA_OnContextItemDoubleClicked notifies Add-Ins that the user has double-clicked the item currently in context.

This event occurs when a user has double-clicked (or pressed the Enter key) on the item in context, either in a diagram, in the Browser window or in a custom compartment. Add-Ins to handle events can subscribe to this broadcast function.

Syntax

Function EA_OnContextItemDoubleClicked (Repository As EA.Repository, GUID As String, ot as EA.ObjectType)

The EA_OnContextItemDoubleClicked function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
GUID	String Direction: IN Description: Contains the GUID of the new context item. The value corresponds to these properties, depending on the value of the ot parameter: • otElement - Element.ElementGUID • otPackage - Package.PackageGUID • otDiagram - Diagram.DiagramGUID • otAttribute - Attribute.AttributeGUID • otMethod - Method.MethodGUID • otConnector - Connector.ConnectorGUID
ot	EA.ObjectType Direction: IN Description: Specifies the type of the new context item.

Return Value

- Return True to notify Enterprise Architect that the double-click event has been handled by an Add-In
- Return False to enable Enterprise Architect to continue processing the event

EA_OnNotifyContextItemModified

EA_OnNotifyContextItemModified notifies Add-Ins that the current context item has been modified.

This event occurs when a user has modified the context item. Add-Ins that require knowledge of when an item has been modified can subscribe to this broadcast function.

Syntax

Sub EA_OnNotifyContextItemModified (Repository As EA.Repository, GUID As String, ot as EA.ObjectType)

The EA_OnNotifyContextItemModified function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
GUID	String Direction: IN Description: Contains the GUID of the new context item. The value corresponds to these properties, depending on the value of the ot parameter: • ot(ObjectType) - GUID value • otElement - Element.ElementGUID • otPackage - Package.PackageGUID • otDiagram - Diagram.DiagramGUID • otAttribute - Attribute.AttributeGUID • otMethod - Method.MethodGUID • otConnector - Connector.ConnectorGUID
ot	EA.ObjectType Direction: IN Description: Specifies the type of the new context item.

Return Value

None.

EA_FileClose

The EA_FileClose event enables the Add-In to respond to a File Close event. When Enterprise Architect closes an opened Model file, this event is raised and passed to all Add-Ins implementing this method.

This event occurs when the model currently opened within Enterprise Architect is about to be closed (when another model is about to be opened or when Enterprise Architect is about to shutdown).

Syntax

Sub EA_FileClose (Repository As EA.Repository)

The EA_FileClose function syntax contains this parameter:

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the Enterprise Architect model about to be closed. Poll its members to retrieve model data and user interface status information.

Return Value

None.

EA_FileNew

The EA_FileNew event enables the Add-In to respond to a File New event. When Enterprise Architect creates a new model file, this event is raised and passed to all Add-Ins implementing this method.

The event occurs when the model being viewed by the Enterprise Architect user changes, for whatever reason (through user interaction or Add-In activity).

Syntax

Sub EA_FileNew (Repository As EA.Repository)

The EA_FileNew function syntax contains this parameter.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

Return Value

None.

EA_FileOpen

The EA_FileOpen event enables the Add-In to respond to a File Open event. When Enterprise Architect opens a new model file, this event is raised and passed to all Add-Ins implementing this method.

The event occurs when the model being viewed by the Enterprise Architect user changes, for whatever reason (through user interaction or Add-In activity).

Syntax

Sub EA_FileOpen (Repository As EA.Repository)

The EA_FileOpen function syntax contains this parameter.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

Return Value

None.

EA_OnPostCloseDiagram

EA_OnPostCloseDiagram notifies Add-Ins that a diagram has been closed.

Syntax

Function EA_OnPostCloseDiagram (Repository As EA.Repository, DiagramID As Integer)

The EA_OnPostCloseDiagram function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the Enterprise Architect model about to be closed. Poll its members to retrieve model data and user interface status information.
DiagramID	Integer Direction: IN Description: Contains the Diagram ID of the diagram that was closed.

Return Value

None.

EA_OnPostInitialized

EA_OnPostInitialized notifies Add-Ins that the Repository object has finished loading and any necessary initialization steps can now be performed on the object.

For example, the Add-In can create an 'Output' tab using Repository.CreateOutputTab.

Syntax

Sub EA_OnPostInitialized (Repository As EA.Repository)

The EA_OnPostInitialized function syntax contains this parameter.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

Return Value

None.

EA_OnPostOpenDiagram

EA_OnPostOpenDiagram notifies Add-Ins that a diagram has been opened.

Syntax

Function EA_OnPostOpenDiagram (Repository As EA.Repository, DiagramID As Integer)

The EA_OnPostOpenDiagram function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
DiagramID	Integer Direction: IN Description: Contains the Diagram ID of the diagram that was opened.

Return Value

None.

EA_OnPostTransform

EA_OnPostTransform notifies Add-Ins that an MDG transformation has taken place with the output in the specified target Package.

This event occurs when a user runs an MDG transform on one or more target Packages; the notification is provided for each transform/target Package immediately after all transform processes have completed.

Syntax

Function EA_OnPostTransform (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPostTransform function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty Objects for the transform performed: • Transform: A string value corresponding to the name of the transform used • PackageID: A long value corresponding to Package.PackageID of the destination Package

Return Value

Reserved for future use.

EA_OnPreExitInstance

EA_OnPreExitInstance is not currently used.

Syntax

Sub EA_OnPreExitInstance (Repository As EA.Repository)

The EA_OnPreExitInstance function syntax contains this parameter.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

Return Value

None.

EA_OnRetrieveModelTemplate

EA_OnRetrieveModelTemplate requests that an Add-In pass a model template to Enterprise Architect. This event occurs when a user executes the 'Add a New Model Using Wizard' command to add a model that has been defined by an MDG Technology.

Syntax

Function EA_OnRetrieveModelTemplate (Repository As EA.Repository, sLocation As String) As String

The EA_OnRetrieveModelTemplate function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
sLocation	String Direction: IN Description: The name of the template requested; this should match the location attribute in the <ModelTemplates> section of an MDG Technology File.

Return Value

Return a string containing the XMI export of the model that is being used as a template.

Return an empty string if access to the template is denied; the Add-In is to handle user notification of the error.

Example

```
Public Function EA_OnRetrieveModelTemplate(ByRef Rep As EA.Repository, ByRef sLocation As String) As String
Dim sTemplate As String
Select Case sLocation
Case "Templates\Template1.xml"
sTemplate = My.Resources.Template1
Case "Templates\Template2.xml"
sTemplate = My.Resources.Template2
Case "Templates\Template3.xml"
sTemplate = My.Resources.Template3
Case Else
MsgBox("Path for " & sLocation & " not found")
sTemplate = ""
```

```
End Select
EA_OnRetrieveModelTemplate = sTemplate
End Function
```

EA_OnTabChanged

EA_OnTabChanged notifies Add-Ins that the currently open tab has changed.

Diagrams do not generate the message when they are first opened - use the broadcast event EA_OnPostOpenDiagram for this purpose.

Syntax

Function EA_OnTabChanged (Repository As EA.Repository, TabName As String, DiagramID As Integer)

The EA_OnTabChanges function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
TabName	String Direction: IN Description: The name of the tab to which focus has been switched.
DiagramID	Long Direction: IN Description: The diagram ID, or 0 if switched to an Add-In tab.

Return Value

None

EA_LoadWindowManager

Enterprise Architect provides a set of Portals, each of which is a collection of shortcuts and information on performing specific areas of work on a project. The Portals help both new and experienced users quickly identify and set up the facilities they most often use in their assigned tasks.

You can add your own Portal to the system-installed set, to provide a convenient and concise call-up of one or more groups of facilities available in your Add-In.

Syntax

Function EA_Connect (Repository As EA.Repository) As String

The EA_Connect function syntax has this parameter:

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

Example Code

```
public String EA_LoadWindowManager(EA.Repository Repository)
{
return Resource1.WindowManager;
}
```

Where Resource1.WindowManager is a resource file with these contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<perspectives>
<perspective name="Add-In">
<category name="Add-In" type="commandlist" projectrequired="true">
<item name="Hello World" command="CallAddin" addin="CS_AddinFramework" function="HelloWorld"/>
<item name="Model Dump" command="CallScript" group="Local Scripts" script="JScript - Recursive Model Dump Example"/>
</category>
<category name="Open Diagrams" type="currentdiagramlist" state = "open"/>
<category name="Recent Diagrams" type="recentdiagramlist" state = "open"/>
<category name="Other Windows" type="otherwindowlist" state = "open"/>
</perspective>
</perspectives>
```

Note that the Add-In cannot specify the icon used.

Model Validation Events

Perform Model Validation from an Add-In

Using Enterprise Architect broadcasts, it is possible to define a set of rules that are evaluated when the user instructs Enterprise Architect to perform model validation. An Add-In that performs model validation would involve these broadcast events.

Command	Detail
EA_OnInitializeUserRules	EA_OnInitializeUserRules is intercepted in order to define rule categories and rules.
EA_OnStartValidation	EA_OnStartValidation can be intercepted to perform any required processing prior to validation.
EA_OnEndValidation	EA_OnEndValidation can be intercepted to perform any required clean-up after validation has completed.
Validate Request	These functions intercept each request to validate an individual element, Package, diagram, connector, attribute and method.
Validate Element	EA_OnRunElementRule
Validate Package	EA_OnRunPackageRule
Validate Diagram	EA_OnRunDiagramRule
Validate Connector	EA_OnRunConnectorRule
Validate Attribute	EA_OnRunAttributeRule
Validate Method	EA_OnRunMethodRule
Validate Parameter	EA_OnRunParameterRule

EA_OnInitializeUserRules

EA_OnInitializeUserRules is called on Enterprise Architect start-up and requests that the Add-In provide Enterprise Architect with a rule category and list of rule IDs for model validation.

This function must be implemented by any Add-In that is to perform its own model validation. It must call Project.DefineRuleCategory once and Project.DefineRule for each rule; these functions are described in the *Project Interface* topic.

Syntax

Sub EA_OnInitializeUserRules (Repository As EA.Repository)

The EA_OnInitializeUserRules function syntax contains this parameter.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

EA_OnStartValidation

EA_OnStartValidation notifies Add-Ins that a user has invoked the model validation command from Enterprise Architect.

Syntax

Sub EA_OnStartValidation (Repository As EA.Repository, ParamArray Args() as Variant)

The EA_OnStartValidation function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Args	ParamArray of Variant Direction: IN Description: Contains a list of Rule Categories that are active for the current invocation of model validation.

EA_OnEndValidation

EA_OnEndValidation notifies Add-Ins that model validation has completed.

Use this event to arrange any clean-up operations arising from the validation.

Syntax

Sub EA_OnEndValidation (Repository As EA.Repository, ParamArray Args() as Variant)

The EA_OnEndValidation function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Args	ParamArray of Variant Direction: IN Description: Contains a list of Rule Categories that were active for the invocation of model validation that has just completed.

EA_OnRunElementRule

This event is triggered once for each rule defined in EA_OnInitializeUserRules to be performed on each element in the selection being validated.

If you don't want to perform the rule defined by RuleID on the given element, then simply return without performing any action.

On performing any validation, if a validation error is found, use the Repository.ProjectInterface.PublishResult method to notify Enterprise Architect.

Syntax

Sub EA_OnRunElementRule (Repository As EA.Repository, RuleID As String, Element As EA.Element)

The EA_OnRunElementRule function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String Direction: IN Description: The ID that was passed into the 'Project.DefineRule' command.
Element	EA.Element Direction: IN Description: The element to potentially perform validation on.

EA_OnRunPackageRule

This event is triggered once for each rule defined in EA_OnInitializeUserRules to be performed on each Package in the selection being validated.

If you don't want to perform the rule defined by RuleID on the given Package, then simply return without performing any action.

On performing any validation, if a validation error is found, use the Repository.ProjectInterface.PublishResult method to notify Enterprise Architect.

Syntax

Sub EA_OnRunPackageRule (Repository As EA.Repository, RuleID As String, PackageID As Long)

The EA_OnRunElementRule function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String Direction: IN Description: The ID that was passed into the 'Project.DefineRule' method.
PackageID	Long Direction: IN Description: The ID of the Package to potentially perform validation on. Use the 'Repository.GetPackageByID' method to retrieve the Package object.

EA_OnRunDiagramRule

This event is triggered once for each rule defined in EA_OnInitializeUserRules to be performed on each diagram in the selection being validated.

If you don't want to perform the rule defined by RuleID on the given diagram, then simply return without performing any action.

On performing any validation, if a validation error is found, use the Repository.ProjectInterface.PublishResult method to notify Enterprise Architect.

Syntax

Sub EA_OnRunDiagramRule (Repository As EA.Repository, RuleID As String, DiagramID As Long)

The EA_OnRunDiagramRule function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String Direction: IN Description: The ID that was passed into the 'Project.DefineRule' command.
DiagramID	Long Direction: IN Description: The ID of the diagram to potentially perform validation on. Use the Repository.GetDiagramByID method to retrieve the diagram object.

EA_OnRunConnectorRule

This event is triggered once for each rule defined in EA_OnInitializeUserRules to be performed on each connector in the selection being validated.

If you don't want to perform the rule defined by RuleID on the given connector, then simply return without performing any action.

On performing any validation, if a validation error is found, use the Repository.ProjectInterface.PublishResult method to notify Enterprise Architect.

Syntax

Sub EA_OnRunConnectorRule (Repository As EA.Repository, RuleID As String, ConnectorID As Long)

The EA_OnRunConnectorRule function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String Direction: IN Description: The ID that was passed into the 'Project.DefineRule' command.
ConnectorID	Long Direction: IN Description: The ID of the connector to potentially perform validation on. Use the 'Repository.GetConnectorByID' method to retrieve the connector object.

EA_OnRunAttributeRule

This event is triggered once for each rule defined in EA_OnInitializeUserRules to be performed on each attribute in the selection being validated.

If you don't want to perform the rule defined by RuleID on the given attribute, then simply return without performing any action.

On performing any validation, if a validation error is found, use the Repository.ProjectInterface.PublishResult method to notify Enterprise Architect.

Syntax

Sub EA_OnRunAttributeRule (Repository As EA.Repository, RuleID As String, AttributeGUID As String, ObjectID As Long)

The EA_OnRunAttributeRule function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String Direction: IN Description: The ID that was passed into the 'Project.DefineRule' command.
AttributeGUID	String Direction: IN Description: The GUID of the attribute to potentially perform validation on. Use the 'Repository.GetAttributeByGuid' method to retrieve the attribute object.
ObjectID	Long Direction: IN Description: The ID of the object that owns the given attribute. Use the 'Repository.GetElementByID' method to retrieve the object.

EA_OnRunMethodRule

This event is triggered once for each rule defined in EA_OnInitializeUserRules to be performed on each method in the selection being validated.

If you don't want to perform the rule defined by RuleID on the given method, then simply return without performing any action.

On performing any validation, if a validation error is found, use the Repository.ProjectInterface.PublishResult method to notify Enterprise Architect.

Syntax

Sub EA_OnRunMethodRule (Repository As EA.Repository, RuleID As String, MethodGUID As String, ObjectID As Long)

The EA_OnRunMethodRule function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String Direction: IN Description: The ID that was passed into the 'Project.DefineRule' command.
MethodGUID	String Direction: IN Description: The GUID of the method to potentially perform validation on. Use the 'Repository.GetMethodByGuid' method to retrieve the method object.
ObjectID	Long Direction: IN Description: The ID of the object that owns the given method. Use the 'Repository.GetElementByID' method to retrieve the object.

EA_OnRunParameterRule

This event is triggered once for each rule defined in EA_OnInitializeUserRules to be performed on each parameter in the selection being validated.

If you don't want to perform the rule defined by RuleID on the given parameter, then simply return without performing any action.

On performing any validation, if a validation error is found, use the Repository.ProjectInterface.PublishResult method to notify Enterprise Architect.

Syntax

Sub EA_OnRunParameterRule (Repository As EA.Repository, RuleID As String, ParameterGUID As String, MethodGUID As String, ObjectID As Long)

The EA_OnRunMethodRule function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String Direction: IN Description: The ID that was passed into the 'Project.DefineRule' command.
ParameterGUID	String Direction: IN Description: The GUID of the parameter to potentially perform validation on. Use this to retrieve the parameter by iterating through the 'Method.Parameters' collection.
MethodGUID	String Direction: IN Description: The GUID of the method that owns the given parameter. Use the 'Repository.GetMethodByGuid' method to retrieve the method object.
ObjectID	Long Direction: IN Description: The ID of the object that owns the given parameter. Use the 'Repository.GetElementByID' method to retrieve the object.

Model Validation Example

This example code is written in C# and provides a skeleton model validation implementation that you might want to use as a starting point in writing your own model validation rules.

Main.cs

```
using System;
namespace myAddin
{
    public class Main
    {
        public Rules theRules;
        public Main()
        {
            theRules = new Rules();
        }
        public string EA_Connect(EA.Repository Repository)
        {
            return "";
        }
        public void EA_Disconnect()
        {
            GC.Collect();
            GC.WaitForPendingFinalizers();
        }
        private bool IsProjectOpen(EA.Repository Repository)
        {
            try
            {
                EA.Collection c = Repository.Models;
                return true;
            }
            catch
            {
                return false;
            }
        }
        public object EA_GetMenuItems(EA.Repository Repository, string MenuLocation, string MenuName)
        {
            switch (MenuName)
            {
```

```
            case "":
                return "-&myAddin";
            case "-&myAddin":
                string() ar = { "&Test" };
                return ar;
        }
        return "";
    }
    public void EA_GetMenuState(EA.Repository Repository, string MenuLocation, string MenuName,
    string ItemName, ref bool IsEnabled, ref bool IsChecked)
    {
        // if no open project, disable all menu options
        if (IsProjectOpen(Repository))
            IsEnabled = true;
        else
            IsEnabled = false;
    }
    public void EA_MenuClick(EA.Repository Repository, string MenuLocation, string MenuName, string
ItemName)
    {
        switch (ItemName)
        {
            case "&Test";
                DoTest(Repository);
                break;
        }
    }
    public void EA_OnInitializeUserRules(EA.Repository Repository)
    {
        if (Repository != null)
        {
            theRules.ConfigureCategories(Repository);
            theRules.ConfigureRules(Repository);
        }
    }
    public void EA_OnRunElementRule(EA.Repository Repository, string RuleID, EA.Element element)
    {
        theRules.RunElementRule(Repository, RuleID, element);
    }
    public void EA_OnRunDiagramRule(EA.Repository Repository, string RuleID, long lDiagramID)
    {
        theRules.RunDiagramRule(Repository, RuleID, lDiagramID);
```

```
        }
        public void EA_OnRunConnectorRule(EA.Repository Repository, string RuleID, long lConnectorID)
        {
            theRules.RunConnectorRule(Repository, RuleID, lConnectorID);
        }
        public void EA_OnRunAttributeRule(EA.Repository Repository, string RuleID, string AttGUID, long lObjectID)
        {
            return;
        }
        public void EA_OnDeleteTechnology(EA.Repository Repository, EA.EventProperties Info)
        {
            return;
        }
        public void EA_OnImportTechnology(EA.Repository Repository, EA.EventProperties Info)
        {
            return;
        }
        private void DoTest(EA.Repository Rep)
        {
            // TODO: insert test code here
         }
    }
}
```

Rules.cs

```
using System;
using System.Collections;
namespace myAddin
{
    public class Rules
    {
        private string m_sCategoryID;
        private System.Collections.ArrayList m_RuleIDs;
        private System.Collections.ArrayList m_RuleIDEx;
        private const string cRule01 = "Rule01";
        private const string cRule02 = "Rule02";
        private const string cRule03 = "Rule03";
        // TODO: expand this list as much as necessary
        public Rules()
        {
            m_RuleIDs = new System.Collections.ArrayList();
```

```
    m_RuleIDEx = new System.Collections.ArrayList();
}
private string LookupMap(string sKey)
{
    return DoLookupMap(sKey, m_RuleIDs, m_RuleIDEx);
}
private string LookupMapEx(string sRule)
{
    return DoLookupMap(sRule, m_RuleIDEx, m_RuleIDs);
}
private string DoLookupMap(string sKey, ArrayList arrValues, ArrayList arrKeys)
{
    if (arrKeys.Contains(sKey))
        return arrValues(arrKeys.IndexOf(sKey)).ToString();
    else
        return "";
}
private void AddToMap(string sRuleID, string sKey)
{
    m_RuleIDs.Add(sRuleID);
    m_RuleIDEx.Add(sKey);
}
private string GetRuleStr(string sRuleID)
{
    switch (sRuleID)
    {
        case cRule01:
            return "Error Message 01";
        case cRule02:
            return "Error Message 02";
        case cRule03:
            return "Error Message 03";
        // TODO: add extra cases as much as necessary
    }
    return "";
}
public void ConfigureCategories(EA.Repository Repository)
{
    EA.Project Project = Repository.GetProjectInterface();
    m_sCategoryID = Project.DefineRuleCategory("Enterprise Collaboration Architecture (ECA) Rules");
}
public void ConfigureRules(EA.Repository Repository)
```

```
        {
            EA.Project Project = Repository.GetProjectInterface();
            AddToMap(Project.DefineRule(m_sCategoryID, EA.EnumMVErrorType.mvError, GetRuleStr(cRule01)),
cRule01);
            AddToMap(Project.DefineRule(m_sCategoryID, EA.EnumMVErrorType.mvError, GetRuleStr(cRule02)),
cRule02);
            AddToMap(Project.DefineRule(m_sCategoryID, EA.EnumMVErrorType.mvError, GetRuleStr(cRule03)),
cRule03);
            // TODO: expand this list
        }
        public void RunConnectorRule(EA.Repository Repository, string sRuleID, long lConnectorID)
        {
            EA.Connector Connector = Repository.GetConnectorByID((int)lConnectorID);
            if (Connector != null)
            {
                switch (LookupMapEx(sRuleID))
                {
                    case cRule02:
                        // TODO: perform rule 2 check
                        break;
                    // TODO: add more cases
                }
            }
        }
        public void RunDiagramRule(EA.Repository Repository, string sRuleID, long lDiagramID)
        {
            EA.Diagram Diagram = Repository.GetDiagramByID((int)lDiagramID);
            if (Diagram != null)
            {
                switch (LookupMapEx(sRuleID))
                {
                    case cRule03:
                        // TODO: perform rule 3 check
                        break;
                    // TODO: add more cases
                }
            }
        }
        public void RunElementRule(EA.Repository Repository, string sRuleID, EA.Element Element)
        {
            if (Element != null)
            {
                switch (LookupMapEx(sRuleID))
```

```
            {
                case cRule01:
                    DoRule01(Repository, Element);
                    break;
                // TODO: add more cases
            }
        }
    }
    private void DoRule01(EA.Repository Repository, EA.Element Element)
    {
        if (Element.Stereotype != "myStereotype")
            return;
        // TODO: validation logic here
        // report validation errors
        EA.Project Project = Repository.GetProjectInterface();
        Project.PublishResult(LookupMap(cRule01), EA.EnumMVErrorType.mvError, GetRuleStr(cRule01));
    }
  }
}
```

Post-New Events

Enterprise Architect Add-Ins can respond to the creation of new elements, connectors, objects, attributes, methods and Packages using these broadcast events:

Post-New Broadcast Events

Event
EA_OnPostNewElement
EA_OnPostNewConnector
EA_OnPostNewDiagram
EA_OnPostNewDiagramObject
EA_OnPostNewAttribute
EA_OnPostNewMethod
EA_OnPostNewPackage
EA_OnPostNewGlossaryTerm

EA_OnPostNewElement

EA_OnPostNewElement notifies Add-Ins that a new element has been created on a diagram. It enables Add-Ins to modify the element upon creation.

This event occurs after a user has dragged a new element from the Toolbox or 'Resources' tab of the Browser window onto a diagram. The notification is provided immediately after the element is added to the model.

Set Repository.SuppressEADialogs to True to suppress Enterprise Architect from showing its default 'Properties' dialog.

Syntax

Function EA_OnPostNewElement (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPostNewElement function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the new element: • ElementID: A long value corresponding to Element.ElementID

Return Value

Return True if the element has been updated during this notification. Return False otherwise.

EA_OnPostNewConnector

EA_OnPostNewConnector notifies Add-Ins that a new connector has been created on a diagram. It enables Add-Ins to modify the connector upon creation.

This event occurs after a user has dragged a new connector from the Toolbox or 'Resources' tab of the Browser window onto a diagram. The notification is provided immediately after the connector is added to the model.

Syntax

Function EA_OnPostNewConnector (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPostNewConnector function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the new connector: • ConnectorID: A long value corresponding to Connector.ConnectorID

Return Value

Return True if the connector has been updated during this notification. Return False otherwise.

EA_OnPostNewDiagram

EA_OnPostNewDiagram notifies Add-Ins that a new diagram has been created. It enables Add-Ins to modify the diagram upon creation.

Syntax

Function EA_OnPostNewDiagram (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPostNewDiagram function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the new diagram: • DiagramID: A long value corresponding to Diagram.PackageID

Return Value

Return True if the diagram has been updated during this notification. Return False otherwise.

EA_OnPostNewDiagramObject

EA_OnPostNewDiagramObject notifies Add-Ins that a new object has been created on a diagram. It enables Add-Ins to modify the object upon creation.

This event occurs after a user has dragged a new object directly from the Browser window or from the 'Resources' tab of the Browser window onto a diagram. The notification is provided immediately after the object is added to the diagram.

Syntax

Function EA_OnPostNewDiagramObject (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPostNewDiagramObject function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the new element: • ID: A long value corresponding to the ElementID of the object that has been added to the diagram • DiagramID: A long value corresponding to the DiagramID of the diagram to which the object has been added • DUID: A string value for the DUID; can be used with Diagram.GetDiagramObjectByID to retrieve the new DiagramObject

Return Value

Return True if the element has been updated during this notification. Return False otherwise.

EA_OnPostNewAttribute

EA_OnPostNewAttribute notifies Add-Ins that a new attribute has been created on a diagram. It enables Add-Ins to modify the attribute upon creation.

This event occurs when a user creates a new attribute on an element by either drag-and-dropping from the Browser window, using the 'Attributes' tab of the Features window, or using the in-place editor on the diagram. The notification is provided immediately after the attribute is created.

Syntax

Function EA_OnPostNewAttribute (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPostNewAttribute function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the new attribute: • AttributeID: A long value corresponding to Attribute.AttributeID

Return Value

Return True if the attribute has been updated during this notification. Return False otherwise.

EA_OnPostNewMethod

EA_OnPostNewMethod notifies Add-Ins that a new method has been created on a diagram. It enables Add-Ins to modify the method upon creation.

This event occurs when a user creates a new method on an element by either drag-dropping from the Browser window, using the method's 'Properties' dialog, or using the in-place editor on the diagram. The notification is provided immediately after the method is created.

Syntax

Function EA_OnPostNewMethod (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPostNewMethod function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the new method: • MethodID: A long value corresponding to Method.MethodID

Return Value

Return True if the method has been updated during this notification. Return False otherwise.

EA_OnPostNewPackage

EA_OnPostNewPackage notifies Add-Ins that a new Package has been created on a diagram. It enables Add-Ins to modify the Package upon creation.

This event occurs when a user drags a new Package from the Toolbox or 'Resources' tab of the Browser window onto a diagram, or by selecting the New Package icon from the Browser window.

Syntax

Function EA_OnPostNewPackage (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPostNewPackage function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the new Package: • PackageID: A long value corresponding to Package.PackageID

Return Value

Return True if the Package has been updated during this notification. Return False otherwise.

EA_OnPostNewGlossaryTerm

EA_OnPostNewGlossaryTerm notifies Add-Ins that a new glossary term has been created. It enables Add-Ins to modify the glossary term upon creation.

The notification is provided immediately after the glossary term is added to the model.

Syntax

Function EA_OnPostNewGlossaryTerm (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPostNewGlossaryTerm function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the new glossary term: • TermID: A string value corresponding to Term.TermID • Term: A string value corresponding to the name of the glossary term being created • Meaning: A string value corresponding to meaning of the glossary term being created

Return Value

Return True if the glossary term has been updated during this notification. Return False otherwise.

Pre-Deletion Events

Enterprise Architect Add-Ins can respond to requests to delete elements, attributes, methods, connectors, diagrams, Packages and glossary terms using these broadcast events:

Pre-Deletion Broadcast Events

Event
EA_OnPreDeleteElement
EA_OnPreDeleteAttribute
EA_OnPreDeleteMethod
EA_OnPreDeleteConnector
EA_OnPreDeleteDiagram
EA_OnPreDeletePackage
EA_OnPreDeleteGlossaryTerm
EA_OnPreDeleteTechnology (Deprecated)

EA_OnPreDeleteElement

EA_OnPreDeleteElement notifies Add-Ins that an element is to be deleted from the model. It enables Add-Ins to permit or deny deletion of the element.

This event occurs when a user deletes an element from the Browser window or on a diagram. The notification is provided immediately before the element is deleted, so that the Add-In can disable deletion of the element.

Syntax

Function EA_OnPreDeleteElement (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreDeleteElement function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the element to be deleted: • ElementID: A long value corresponding to Element.ElementID

Return Value

- Return True to enable deletion of the element from the model
- Return False to disable deletion of the element

EA_OnPreDeleteAttribute

EA_OnPreDeleteAttribute notifies Add-Ins that an attribute is to be deleted from the model. It enables Add-Ins to permit or deny deletion of the attribute.

This event occurs when a user attempts to permanently delete an attribute from the Browser window. The notification is provided immediately before the attribute is deleted, so that the Add-In can disable deletion of the attribute.

Syntax

Function EA_OnPreDeleteAttribute (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreDeleteAttribute function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the attribute to be deleted: • AttributeID: A long value corresponding to Attribute.AttributeID

Return Value

- Return True to enable deletion of the attribute from the model
- Return False to disable deletion of the attribute

EA_OnPreDeleteMethod

EA_OnPreDeleteMethod notifies Add-Ins that a method (operation) is to be deleted from the model. It enables Add-Ins to permit or deny deletion of the method.

This event occurs when a user attempts to permanently delete a method from the Browser window. The notification is provided immediately before the method is deleted, so that the Add-In can disable deletion of the method.

Syntax

Function EA_OnPreDeleteMethod (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreDeleteMethod function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the method to be deleted: • MethodID: A long value corresponding to Method.MethodID

Return Value

- Return True to enable deletion of the method from the model
- Return False to disable deletion of the method

EA_OnPreDeleteConnector

EA_OnPreDeleteConnector notifies Add-Ins that a connector is to be deleted from the model. It enables Add-Ins to permit or deny deletion of the connector.

This event occurs when a user attempts to permanently delete a connector on a diagram. The notification is provided immediately before the connector is deleted, so that the Add-In can disable deletion of the connector.

Syntax

Function EA_OnPreDeleteConnector (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreDeleteConnector function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the connector to be deleted: • ConnectorID: A long value corresponding to Connector.ConnectorID

Return Value

- Return True to enable deletion of the connector from the model
- Return False to disable deletion of the connector

EA_OnPreDeleteDiagram

EA_OnPreDeleteDiagram notifies Add-Ins that a diagram is to be deleted from the model. It enables Add-Ins to permit or deny deletion of the diagram.

This event occurs when a user attempts to permanently delete a diagram from the Browser window. The notification is provided immediately before the diagram is deleted, so that the Add-In can disable deletion of the diagram.

Syntax

Function EA_OnPreDeleteDiagram (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreDeleteDiagram function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the diagram to be deleted: • DiagramID: A long value corresponding to Diagram.DiagramID

Return Value

- Return True to enable deletion of the diagram from the model
- Return False to disable deletion of the diagram

EA_OnPreDeleteDiagramObject

EA_OnPreDeleteDiagramObject notifies Add-Ins that a diagram object is to be deleted from the model. It enables Add-Ins to permit or deny deletion of the element.

This event occurs when a user attempts to permanently delete an element from a diagram. The notification is provided immediately before the element is deleted, so that the Add-In can disable deletion of the element.

Syntax

Function EA_OnPreDeleteDiagramObject (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreDeleteDiagramObject function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the element to be deleted: • ID: A long value corresponding to DiagramObject.ElementID

Return Value

- Return True to enable deletion of the element from the model
- Return False to disable deletion of the element

EA_OnPreDeletePackage

EA_OnPreDeletePackage notifies Add-Ins that a Package is to be deleted from the model. It enables Add-Ins to permit or deny deletion of the Package.

This event occurs when a user attempts to permanently delete a Package from the Browser window. The notification is provided immediately before the Package is deleted, so that the Add-In can disable deletion of the Package.

Syntax

Function EA_OnPreDeletePackage (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreDeletePackage function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the Package to be deleted: • PackageID: A long value corresponding to Package.PackageID

Return Value

- Return True to enable deletion of the Package from the model
- Return False to disable deletion of the Package

EA_OnPreDeleteGlossaryTerm

EA_OnPreDeleteGlossaryTerm notifies Add-Ins that a glossary term is to be deleted from the model. It enables Add-Ins to permit or deny deletion of the glossary term.

The notification is provided immediately before the glossary term is deleted, so that the Add-In can disable deletion of the glossary term.

Syntax

Function EA_OnPreDeleteGlossaryTerm (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreDeleteGlossaryTerm function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the glossary term to be deleted: • TermID: A long value corresponding to Term.TermID

Return Value

- Return True to enable deletion of the glossary term from the model
- Return False to disable deletion of the glossary term

Pre New-Object Events

When you create an Add-In, you can include broadcast events to intercept and respond to requests to create new objects, including elements, connectors, diagram objects, attributes, methods and Packages.

Events to intercept

Event
Creation of a new element
Creation of a new connector
Creation of a new diagram
Creation of a new diagram object
Creation of a new element by dropping onto a diagram from the Browser window.
Creation of a new attribute
Creation of a new method
Creation of a new Package
Creation of a new glossary term

EA_OnPreNewElement

EA_OnPreNewElement notifies Add-Ins that a new element is about to be created on a diagram. It enables Add-Ins to permit or deny creation of the new element.

This event occurs when a user drags a new element from the Toolbox or 'Resources' tab of the Browser window onto a diagram. The notification is provided immediately before the element is created, so that the Add-In can disable addition of the element.

Syntax

Function EA_OnPreNewElement (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreNewElement function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the element to be created: • Type: A string value corresponding to Element.Type • FQStereotype: A string value corresponding to Element.FQStereotype • Stereotype: A string value corresponding to Element.Stereotype • ParentID: A long value corresponding to Element.ParentID • DiagramID: A long value corresponding to the ID of the diagram to which the element is being added

Return Value

- Return True to enable addition of the new element to the model
- Return False to disable addition of the new element

EA_OnPreNewConnector

EA_OnPreNewConnector notifies Add-Ins that a new connector is about to be created on a diagram. It enables Add-Ins to permit or deny creation of a new connector.

This event occurs when a user drags a new connector from the Toolbox or 'Resources' tab of the Browser window, onto a diagram. The notification is provided immediately before the connector is created, so that the Add-In can disable addition of the connector.

Syntax

Function EA_OnPreNewConnector (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreNewConnector function syntax contains these elements:

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the connector to be created: • Type: A string value corresponding to Connector.Type • Subtype: A string value corresponding to Connector.Subtype • Stereotype: A string value corresponding to Connector.Stereotype • ClientID: A long value corresponding to Connector.ClientID • SupplierID: A long value corresponding to Connector.SupplierID • DiagramID: A long value corresponding to Connector.DiagramID

Return Value

- Return True to enable addition of the new connector to the model
- Return False to disable addition of the new connector

EA_OnPreNewDiagram

EA_OnPreNewDiagram notifies Add-Ins that a new diagram is about to be created. It enables Add-Ins to permit or deny creation of the new diagram.

The notification is provided immediately before the diagram is created, so that the Add-In can disable addition of the diagram.

Syntax

Function EA_OnPreNewDiagram (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreNewDiagram function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the diagram to be created: • Type: A string value corresponding to Diagram.Type • ParentID: A long value corresponding to Diagram.ParentID • PackageID: A long value corresponding to Diagram.PackageID

Return Value

- Return True to enable addition of the new diagram to the model
- Return False to disable addition of the new diagram

EA_OnPreNewDiagramObject

EA_OnPreNewDiagramObject notifies Add-Ins that a new diagram object is about to be dropped on a diagram. It enables Add-Ins to permit or deny creation of the new object.

This event occurs when a user drags an object directly from the Enterprise Architect Browser window or from the 'Resources' tab of the Browser window onto a diagram. The notification is provided immediately before the object is created, so that the Add-In can disable addition of the object.

Syntax

Function EA_OnPreNewDiagramObject (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreNewDiagramObject function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the object to be created: • Type: A string value corresponding to the Type of object being added to the diagram • Stereotype: A string value corresponding to the Stereotype of the object being added to the diagram • ID: A long value corresponding to the ID of the element, Package or diagram being added to the diagram • DiagramID: A long value corresponding to the ID of the diagram to which the object is being added

Return Value

- Return True to enable addition of the object to the model
- Return False to disable addition of the object

EA_OnPreDropFromTree

When a user drags any kind of element from the Browser window onto a diagram, EA_OnPreDropFromTree notifies the Add-In that a new item is about to be dropped onto a diagram. The notification is provided immediately before the element is dropped, so that the Add-In can override the default action that would be taken for this drag.

Syntax

Function EA_OnPreDropFromTree (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreDropFromTree function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the element to be created: • ID: A long value of the type being dropped • Type: A string value corresponding to type of element being dropped • DiagramID: A long value corresponding to the ID of the diagram to which the element is being added • PositionX: The X coordinate into which the element is being dropped • PositionY: The Y coordinate into which the element is being dropped • DroppedID: A long value corresponding to the ID of the element the item has been dropped onto

Return Value

- Return True to allow the default behavior to be executed
- Return False to override this behavior

EA_OnPreNewAttribute

EA_OnPreNewAttribute notifies Add-Ins that a new attribute is about to be created on an element. It enables Add-Ins to permit or deny creation of the new attribute.

This event occurs when a user creates a new attribute on an element by either drag-dropping from the Browser window, using the 'Attributes' tab of the Features window, or using the in-place editor on the diagram. The notification is provided immediately before the attribute is created, so that the Add-In can disable addition of the attribute.

Syntax

Function EA_OnPreNewAttribute (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreNewAttribute function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the attribute to be created: • Type: A string value corresponding to Attribute.Type • Stereotype: A string value corresponding to Attribute.Stereotype • ParentID: A long value corresponding to Attribute.ParentID • ClassifierID: A long value corresponding to Attribute.ClassifierID

Return Value

- Return True to enable addition of the new attribute to the model
- Return False to disable addition of the new attribute

EA_OnPreNewMethod

EA_OnPreNewMethod notifies Add-Ins that a new method is about to be created on an element. It enables Add-Ins to permit or deny creation of the new method.

This event occurs when a user creates a new method on an element by either drag-dropping from the Browser window, using the 'Operations' tab of the Features window, or using the in-place editor on the diagram. The notification is provided immediately before the method is created, so that the Add-In can disable addition of the method.

Syntax

Function EA_OnPreNewMethod (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreNewMethod function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the method to be created: • ReturnType: A string value corresponding to Method.ReturnType • Stereotype: A string value corresponding to Method.Stereotype • ParentID: A long value corresponding to Method.ParentID • ClassifierID: A long value corresponding to Method.ClassifierID

Return Value

- Return True to enable addition of the new method to the model
- Return False to disable addition of the new method

EA_OnPreNewPackage

EA_OnPreNewPackage notifies Add-Ins that a new Package is about to be created in the model. It enables Add-Ins to permit or deny creation of the new Package.

This event occurs when a user drags a new Package from the Toolbox or 'Resources' tab of the Browser window onto a diagram, or by selecting the New Package icon from the Browser window. The notification is provided immediately before the Package is created, so that the Add-In can disable addition of the Package.

Syntax

Function EA_OnPreNewPackage (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreNewPackage function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the Package to be created: • Stereotype: A string value corresponding to Package.Stereotype • ParentID: A long value corresponding to Package.ParentID • DiagramID: A long value corresponding to the ID of the diagram to which the Package is being added

Return Value

- Return True to enable addition of the new Package to the model
- Return False to disable addition of the new Package

EA_OnPreNewGlossaryTerm

EA_OnPreNewGlossaryTerm notifies Add-Ins that a new glossary term is about to be created. It enables Add-Ins to permit or deny creation of the new glossary term.

The notification is provided immediately before the glossary term is created, so that the Add-In can disable addition of the element.

Syntax

Function EA_OnPreNewGlossaryTerm (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreNewGlossaryTerm function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the glossary term to be created: • TermID: A string value corresponding to Term.TermID • Term: A string value corresponding to the name of the glossary term being created • Meaning: A string value corresponding to meaning of the glossary term being created

Return Value

- Return True to enable addition of the new glossary term to the model
- Return False to disable addition of the new glossary term

Tagged Value Events

Enterprise Architect includes the Addin Broadcast Tagged Value type that allows an Add-In to respond to attempts to edit it. The function that is called depends on the type of object the Tagged Value is on.

Tagged Value Events

Event
EA_OnAttributeTagEdit
EA_OnConnectorTagEdit
EA_OnElementTagEdit
EA_OnMethodTagEdit

EA_OnAttributeTagEdit

EA_OnAttributeTagEdit is called when the user clicks the  button for a Tagged Value of type AddinBroadcast on an attribute.

The Add-In displays fields to show and change the value and notes; this function provides the initial values for the Tagged Value notes and value, and takes on any changes on exit of the function.

Syntax

Sub EA_OnAttributeTagEdit (Repository As EA.Repository, AttributeID As Long, String TagName, String TagValue, String TagNotes)

The EA_OnAttributeTagEdit function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
AttributeID	Long Direction: IN Description: The ID of the attribute that this Tagged Value is on.
TagName	String Direction: IN Description: The name of the Tagged Value to edit.
TagValue	String Direction: INOUT Description: The current value of the tag; if the value is updated, the new value is stored in the repository on exit of the function.
TagNotes	String Direction: INOUT Description: The current value of the Tagged Value notes; if the value is updated, the new value is stored in the repository on exit of the function.

EA_OnConnectorTagEdit

EA_OnConnectorTagEdit is called when the user clicks the  button for a Tagged Value of type AddinBroadcast on a connector.

The Add-In displays fields to show and change the value and notes; this function provides the initial values for the Tagged Value notes and value, and takes on any changes on exit of the function.

Syntax

Sub EA_OnConnectorTagEdit (Repository As EA.Repository, ConnectorID As Long, String TagName, String TagValue, String TagNotes)

The EA_OnConnectorTagEdit function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
ConnectorID	Long Direction: IN Description: The ID of the connector that this Tagged Value is on.
TagName	String Direction: IN Description: The name of the Tagged Value to edit.
TagValue	String Direction: INOUT Description: The current value of the tag; if the value is updated, the new value is stored in the repository on exit of the function.
TagNotes	String Direction: INOUT Description: The current value of the Tagged Value notes; if the value is updated, the new value is stored in the repository on exit of the function.

EA_OnElementTagEdit

EA_OnElementTagEdit is called when the user clicks the  button for a Tagged Value of type AddinBroadcast on an element.

The Add-In displays fields to show and change the value and notes; this function provides the initial values for the Tagged Value notes and value, and takes on any changes on exit of the function.

Syntax

Sub EA_OnElementTagEdit (Repository As EA.Repository, ObjectID As Long, String TagName, String TagValue, String TagNotes)

The EA_OnElementTagEdit function syntax contains these elements:

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
ObjectID	Long Direction: IN Description: The ID of the object (element) that this Tagged Value is on.
TagName	String Direction: IN Description: The name of the Tagged Value to edit.
TagValue	String Direction: INOUT Description: The current value of the tag; if the value is updated, the new value is stored in the repository on exit of the function.
TagNotes	String Direction: INOUT Description: The current value of the Tagged Value notes; if the value is updated, the new value is stored in the repository on exit of the function.

EA_OnMethodTagEdit

EA_OnMethodTagEdit is called when the user clicks the  button for a Tagged Value of type AddinBroadcast on an operation.

The Add-In displays fields to show and change the value and notes; this function provides the initial values for the Tagged Value notes and value, and takes on any changes on exit of the function.

Syntax

Sub EA_OnMethodTagEdit (Repository As EA.Repository, MethodID As Long, String TagName, String TagValue, String TagNotes)

The EA_OnMethodTagEdit function syntax contains these elements:

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
MethodID	Long Direction: IN Description: The ID of the method that this Tagged Value is on.
TagName	String Direction: IN Description: The name of the Tagged Value to edit.
TagValue	String Direction: INOUT Description: The current value of the tag; if the value is updated, the new value is stored in the repository on exit of the function.
TagNotes	String Direction: INOUT Description: The current value of the Tagged Value notes; if the value is updated, the new value is stored in the repository on exit of the function.

Technology Events

Enterprise Architect Add-Ins can respond to events associated with the use of MDG Technologies.

Technology Broadcast Events

Event
EA_OnInitializeTechnologies
EA_OnPreActivateTechnology
EA_OnPostActivateTechnology
EA_OnPreDeleteTechnology (Deprecated)
EA_OnDeleteTechnology (Deprecated)
EA_OnImportTechnology (Deprecated)

EA_OnInitializeTechnologies

EA_OnInitializeTechnologies requests that an Add-In pass an MDG Technology to Enterprise Architect for loading.

This event occurs on Enterprise Architect start up. Return your technology XML to this function and Enterprise Architect loads and enables it.

Syntax

Function EA_OnInitializeTechnologies (Repository As EA.Repository) As Object

The EA_OnInitializeTechnologies function syntax contains this parameter:

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

Return Value

Return the MDG Technology as a single XML string.

Example

```
Public Function EA_OnInitializeTechnologies(ByVal Repository As EA.Repository) As Object
    EA_OnInitializeTechnologies = My.Resources.MyTechnology
End Function
```

EA_OnPreActivateTechnology

EA_OnPreActivateTechnology notifies Add-Ins that an MDG Technology resource is about to be activated in the model.

This event occurs when a user selects to activate an MDG Technology resource in the model (by clicking on the Set Active button on the 'MDG Technologies' dialog or by selecting the technology in the list box in the Default Tools toolbar).

The notification is provided immediately after the user attempts to activate the MDG Technology, so that the Add-In can permit or disable activation of the Technology.

Syntax

Function EA_OnPreActivateTechnology (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreActivateTechnology function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the MDG Technology to be activated: • TechnologyID: A string value corresponding to the MDG Technology ID

Return Value

- Return True to enable activation of the MDG Technology resource in the model
- Return False to disable activation of the MDG Technology resource

EA_OnPostActivateTechnology

EA_OnPostActivateTechnology notifies Add-Ins that an MDG Technology resource has been activated in the model.

This event occurs when a user activates an MDG Technology resource in the model (by clicking on the Set Active button on the 'MDG Technologies' dialog, or by selecting the technology in the list box in the Default Tools toolbar).

The notification is provided immediately after the user succeeds in activating the MDG Technology, so that the Add-In can update the Technology if necessary.

Syntax

Function EA_OnPostActivateTechnology (Repository As EA.Repository, Info As EA.EventProperties)

The EA_OnPostActivateTechnology function syntax contains these parameters:

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the MDG Technology to be activated: • TechnologyID: A string value corresponding to the MDG Technology ID

Return Value

Return True if the MDG Technology resource is updated during this notification. Return False otherwise.

EA_OnPreDeleteTechnology

Deprecated - refers to deleting a technology through the 'Resources' tab of the Browser window; this process is no longer recommended. See *Deploy An MDG Technology* for information on recommended methods for using technologies.

EA_OnPreDeleteTechnology notifies Add-Ins that an MDG Technology resource is about to be deleted from the model.

This event occurs when a user deletes an MDG Technology resource from the model.

The notification is provided immediately after the user confirms their request to delete the MDG Technology, so that the Add-In can disable deletion of the MDG Technology.

Related Broadcast Events

Event
EA_OnInitializeTechnologies
EA_OnPreActivateTechnology
EA_OnPostActivateTechnology

Syntax

Function EA_OnPreDeleteTechnology (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA_OnPreDeleteTechnology function syntax contains these elements:

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the MDG Technology to be deleted: • TechnologyID: A string value corresponding to the MDG Technology ID

Return Value

- Return True to enable deletion of the MDG Technology resource from the model
- Return False to disable deletion of the MDG Technology resource

EA_OnDeleteTechnology

Deprecated - refers to deleting a technology through the 'Resources' tab of the Browser window; this process is no longer recommended. See *Deploy An MDG Technology* for information of recommended methods for using technologies.

EA_OnDeleteTechnology notifies Add-Ins that an MDG Technology resource has been deleted from the model.

This event occurs after a user has deleted an MDG Technology resource from the model. Add-Ins that require an MDG Technology resource to be loaded can catch this event to disable certain functionality.

Related Events

Event
EA_OnInitializeTechnologies
EA_OnPreActivateTechnology
EA_OnPostActivateTechnology

Syntax

Sub EA_OnDeleteTechnology (Repository As EA.Repository, Info As EA.EventProperties)

The EA_OnDeleteTechnology function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects: • TechnologyID: A string value corresponding to the MDG Technology ID

Return Value

None.

EA_OnImportTechnology

Deprecated - refers to importing a technology into the 'Resources' tab of the Browser window; this process is no longer recommended. See *Deploy An MDG Technology* for information of recommended methods for using technologies.

EA_OnImportTechnology notifies Add-Ins that you have imported an MDG Technology resource into the model.

This event occurs after you have imported an MDG Technology resource into the model. Add-Ins that require an MDG Technology resource to be loaded can catch this Add-In to enable certain functionality.

Related Events

Event
EA_OnInitializeTechnologies
EA_OnPreActivateTechnology
EA_OnPostActivateTechnology

Syntax

Sub EA_OnImportTechnology (Repository As EA.Repository, Info As EA.EventProperties)

The EA_OnImportTechnology function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects: • TechnologyID: A string value corresponding to the MDG Technology ID

Return Value

None.

Technology Rules

The Technology Rules set of events provides hooks for Add-Ins to customize the behavior for their own modeling languages beyond that which can be specified through MDG Technologies alone. These events have been subdivided into categories to assist in exploring the events that are available.

Technology Rule Events

Events
The EARules_Initialize event is called for all Add-Ins during initialization. Specifying one or more profiles for which to define rules is a pre-requisite for all rule calls.
Diagram Appearance Rule events modify some facet of how elements are rendered on diagrams.
User Interface Rule events are used to show, hide or customize aspects of the user interface, or to customize available actions to work naturally with a language.

EARules_Initialize

EARules_Initialize enables Add-Ins to override internal behavior for one or more technologies.
This event occurs during Add-In initialization.

Syntax

Function EARules_Initialize (Repository As EA.Repository, RuleIndex As Integer, Base As String) As String
The EARules_Initialize function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleIndex	Integer Direction: IN Description: Provides a count of the number of calls to this function. This allows you to define multiple rule sets without maintaining your own state for previous calls to this function.
Base	String Direction: OUT Description: This parameter can be assigned the name of an existing set of rules that will be treated as a superclass of the rule set defined by your Add-In. The rules are named by the profile name containing the stereotypes or diagram types that are being modified. You will usually use this value if you are extending the stereotypes within that profile. The customized rules that are built-in to Enterprise Architect are: • ArchiMate3.0 • BPMN2.0 • DMN1.1 • MARTE • Modelica • SPEM • SysML1.2 • SysML1.3 • SysML1.4 • SysPhS

Return Value

- Returns a non-empty string that matches the name of a profile or diagram profile to define rules for that profile; this

function will be called again to allow rules for additional profiles

- Returns an empty string to specify that the Add-In does not define any additional technology-specific rules

Diagram Appearance Rule Events

Diagram Appearance Events

Event
EARules_ClosePartitionName is called to allow control over rendering a closing line after the name for an Activity Partition.
EARules_ElementDisplayName is called to allow alternative text to be displayed as the name during the rendering of an element.
EARules_GetCompartmentItem is called to allow altering the text to display for an item in a built-in compartment.
EARules_GetCompartmentName is called to determine the name displayed for built-in compartments.
EARules_GetNameUnderline is called by Enterprise Architect to determine if the name should be drawn with an underline.
EARules_GetPropertyString is called to provide alternative or additional text to the default element property string.
EARules_GetShapeScript is called to allow a custom shape script for elements without one defined.
EARules_ShowStereotype is called to allow control over stereotype visibility for the default rendering of an Activity Partition.
EARules_StereotypeDisplayName is called by Enterprise Architect to provide a custom keyword to display on the diagram in place of the stereotype name.

EARules_ClosePartitionName

EARules_ClosePartitionName allows an Add-In that is registered to provide rules for a language to determine if the name for an Activity Partition is displayed with a closing line.

This event occurs during diagram drawing.

Syntax

Function EARules_ClosePartitionName (Repository As EA.Repository, Language As String, Element as EA.Element) As Integer

The EARules_ClosePartitionName function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Element	EA.Element Direction: IN Description: Specifies the element currently being drawn.

Return Value

- Return a positive value to specify that the name should be closed
- Return zero to specify the name should not be closed
- Return a negative value to use the behavior from the base rules

EARules_ElementDisplayName

EARules_ElementDisplayName allows an Add-In registered to provide rules for a language to override the text displayed for the name in the default notation. An example of where you might want to use this is if an element should take its name automatically from another element.

This event occurs during the drawing of Activity Partitions and Shape Scripts.

Syntax

Function EARules_ElementDisplayName (Repository As EA.Repository, Language As String, Element as EA.Element, CurrentName as String, Base as Integer) As String

The EARules_ElementDisplayName function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Element	EA.Element Direction: IN Description: Specifies the element currently being drawn.
CurrentName	String Direction: IN Description: Specifies the name currently being used for this element.
Base	Integer Direction: OUT Description: Controls whether the base rules will be called if you return an empty value. A non-zero value and return of an empty string means that the parent rules will determine the display of the element name.

Return Value

A string to override the displayed name of the element on a diagram.

EARules_GetCompartmentItem

EARules_GetCompartmentName allows an Add-In registered to provide rules for a language to override the text displayed for individual items in a compartment.

This event occurs during diagram drawing.

Syntax

Function EARules_GetCompartmentItem (Repository As EA.Repository, Language As String, Element as EA.Element, Compartment as String, Item as String, Base as Integer) As String

The EARules_GetCompartmentItem function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Element	EA.Element Direction: IN Description: Specifies the element currently being drawn.
Compartment	String Direction: IN Description: Specifies the compartment being rendered.
Item	String Direction: IN Description: Specifies the item being rendered.
Base	Integer Direction: OUT Description: Controls whether the base rules will be called if you return an empty value. A non-zero value and return of an empty string means that the parent rules will determine the display of the compartment item.

Return Value

A string defining the text to be displayed for this compartment item.

EARules_GetCompartmentName

EARules_GetCompartmentName allows an Add-In registered to provide rules for a language to override the label displayed at the top of a compartment.

This event occurs during diagram drawing.

Syntax

Function EARules_GetCompartmentName (Repository As EA.Repository, Language As String, Element as EA.Element, Compartment as String, Base as Integer) As String

The EARules_GetCompartmentName function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Element	EA.Element Direction: IN Description: Specifies the element currently being drawn.
Compartment	String Direction: IN Description: Specifies the compartment being rendered.
Base	Integer Direction: OUT Description: Controls whether the base rules will be called if you return an empty value. A non-zero value and return of an empty string means that the parent rules will determine the display of the compartment name.

Return Value

A string defining the name to be rendered for the specified compartment.

EARules_GetNameUnderline

EARules_GetNameUnderline allows an Add-In registered to provide rules for a language to control if the name of an element is rendered with an underline.

This event occurs during diagram drawing.

Syntax

Function EARules_GetNameUnderline (Repository As EA.Repository, Language As String, Element as EA.Element) As Integer

The EARules_GetNameUnderline function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Element	EA.Element Direction: IN Description: Specifies the element currently being drawn.

Return Value

- Return a positive value to specify that the name should be underlined
- Return zero to specify that the name should not be underlined
- Return a negative value to use the behavior from the base rules

EARules_GetPropertyString

EARules_GetPropertyString allows an Add-In registered to provide rules for a language to override the text for the property string of an element. In standard UML notation this is rendered between '{' and '}' near the name.

This event occurs during diagram drawing.

Syntax

Function EARules_GetPropertyString (Repository As EA.Repository, Language As String, Element as EA.Element, Order as Integer) As String

The EARules_GetPropertyString function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Element	EA.Element Direction: IN Description: Specifies the element currently being drawn.
Order	Integer Direction: OUT Description: Controls the order in which text provided by base rules is added to the return value. • Assign a negative value to place the base property string ahead of the return value • Assign a positive value to place the base property string after the return value • Assign zero to prevent the base property string from being displayed

Return Value

A string defining the contents of the property string used in the default element notation.

EARules_GetShapeScript

EARules_GetShapeScript allows an Add-In registered to provide rules for a language to customize the rendering of an element for the modeling language of the diagram being drawn.

This event occurs during diagram drawing.

Syntax

Function EARules_GetShapeScript (Repository As EA.Repository, Language As String, Element as EA.Element) As String

The EARules_GetShapeScript function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Element	EA.Element Direction: IN Description: Specifies the element currently being drawn.

Return Value

- Return a string containing the Shape Script to use for this element
- Return an empty string to defer to the parent rules

EARules_ShowStereotype

EARules_ShowStereotype allows an Add-In registered to provide rules for a language to determine if the stereotype for an Activity Partition is displayed with the name.

This event occurs during diagram drawing.

Syntax

Function EARules_ShowStereotype (Repository As EA.Repository, Language As String, Element as EA.Element) As Integer

The EARules_ShowStereotype function syntax contains these parameters:

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Element	EA.Element Direction: IN Description: Specifies the element currently being drawn.

Return Value

- Return a positive value to specify that the name should be closed
- Return zero to specify that the name should not be closed
- Return a negative value to use the behavior from the base rules

EARules_StereotypeDisplayName

EARules_StereotypeDisplayName allows an Add-In registered to provide rules for a language to override the text displayed for the stereotype in the default notation. In standard UML notation, this is rendered between '«' and '»' near the name.

This event occurs during diagram drawing.

Syntax

Function EARules_StereotypeDisplayName (Repository As EA.Repository, Language As String, Stereotype as String, Base as Integer) As String

The EARules_StereotypeDisplayName function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Stereotype	String Direction: IN Description: Specifies the fully qualified name of the stereotype being queried.
Base	Integer Direction: OUT Description: Controls whether the base rules will be called if you return an empty value. A non-zero value and return of an empty string means that the parent rules will determine the display of the stereotype.

Return Value

A string to override the displayed name of the stereotype on a diagram.

User Interface Rule Events

User Interface Behavior Events

Event
EARules_AllowNesting is called to determine if creating a visual nesting of two elements on a diagram results in setting the ownership in the browser.
EARules_AppendChildDiagrams is called to build the context menu for adding child diagrams to an element.
EARules_AppendChildElements is called to build the context menu for adding child elements to an element.
EARules_CanOverrideStereotype is called to determine if the stereotype on a Type is assigned to a property when assigning the type.
EARules_CanProxy is called to determine if this element can be represented by another element.
EARules_CanReparent is called to determine if visually nesting on a diagram is intended to update ownership in the Browser.
EARules_CreateModel is called to allow creation of a wrapping element during creation of a diagram.
EARules_EnableElementProperty is called to determine if a particular property in the docked Properties window should allow edits for this element.
EARules_ForceLength is used to allow particular diagrams to require elements determined by EARules_IsAdjustable to have their length fixed to the size of the diagram.
EARules_AppendChildDiagrams is called to override a request for a UML diagram when the context specifies a different language is being used.
EARules_IsAdjustable is called to determine which elements can have their length automatically adjusted on diagrams where EARules_ForceLength has allowed the operation.
EARules_PropagateStereotype is called to determine if a classifier stereotype can be applied to an instance using that classifier.
EARules_ShowElementProperty is called to determine if a particular property in the docked Properties window should be displayed for this element.
EARules_ShowFrame is called to automatically insert the parent of the diagram as a frame.
EARules_ShowParentFrame is called to automatically insert the parent of the diagram as a frame.

EARules_AllowNesting

EARules_AllowNesting allows an Add-In registered to provide rules for a language to specify if two elements should be nested in the Browser window after being nested on a diagram.

This event occurs while a user drags one element onto another in a diagram.

Syntax

Function EARules_AllowNesting (Repository As EA.Repository, Language As String, Child as EA.Element, Parent as EA.Element, Diagram as EA.Diagram) As Integer

The EARules_AllowNesting function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Child	EA.Element Direction: IN Description: Specifies the element that has been graphically nested within another element.
Parent	EA.Element Direction: IN Description: Specifies the target element of a drag-and-drop operation within a diagram.
Diagram	EA.Diagram Direction: IN Description: The diagram that the move event has occurred on.

Return Value

- Return a positive value to allow the nesting to occur
- Return a negative value to use the base rules
- Return zero to prevent the nesting from occurring

EARules_AppendChildDiagrams

EARules_AppendChildDiagrams allows an Add-In registered to provide rules for a language to specify the list of items to be shown in the '[New Child Diagram' menu.

This event occurs when a context menu is shown that includes 'New Child Diagram'.

Syntax

Function EARules_AppendChildDiagrams (Repository As EA.Repository, Language As String, Parent as EA.Element, Diagram as EA.Diagram, Order as Integer) As Variant

The EARules_AppendChildDiagrams function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Parent	EA.Element Direction: IN Description: Specifies the element showing a context menu that includes the option for new child diagrams.
Diagram	EA.Diagram Direction: IN Description: The diagram that is showing the parent element. If a user is showing a context menu outside a diagram, this could be null.
Order	EA.Diagram Direction: OUT Description: Allows controlling if and where the child diagrams specified in the parent rules are shown. A positive value means they will be shown after the items specified in this function. Zero means they are not shown at all. A negative value means that they are shown after the items specified in this function.

Return Value

This function supports returning either a single string with multiple items specified by a ';', or an array of strings.

Each item can be one of these:

- "-" - inserts a separator

- Any other text - shows that text as the item, and if the user clicks on it the Add-In is responsible for creating the requested diagram in EA_OnMenuClick

EARules_AppendChildElements

EARules_AppendChildElements allows an Add-In registered to provide rules for a language to specify the list of items to be shown in the 'New Child Element' menu.

This event occurs when a context menu is shown that includes 'New Child Element'.

Syntax

Function EARules_AppendChildElements (Repository As EA.Repository, Language As String, Parent as EA.Element, Diagram as EA.Diagram, Order as Integer) As Variant

The EARules_AppendChildElements function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Parent	EA.Element Direction: IN Description: Specifies the element showing a context menu that includes the option for new child diagrams.
Diagram	EA.Diagram Direction: IN Description: The diagram that is showing the parent element. If a user is showing a context menu outside a diagram, this could be null.
Order	EA.Diagram Direction: OUT Description: Allows the control of if and where the child diagrams specified in the parent rules are shown. A positive value means they will be shown after the items specified in this function. Zero means they are not shown at all. A negative value means that they are shown after the items specified in this function.

Return Value

This function supports returning either a single string with multiple items specified by a ';', or an array of strings.

Each item can be one of these:

- "-" - inserts a separator

- A valid toolbox string including an alias - should be of the form <profile>::<stereotype>(UML::<base>)=<alias>
- Any other text - shows that text as the item, and if the user clicks on it the Add-In is responsible for creating the requested element in EA_OnMenuClick

EARules_CanOverrideStereotype

EARules_CanOverrideStereotype allows an Add-In registered to provide rules for a language to control when the stereotype from a newly assigned type is propagated to the property.

This event occurs when a type is assigned to a property element.

Syntax

Function EARules_CanOverrideStereotype (Repository As EA.Repository, Language As String, Element as EA.Element) As Integer

The EARules_CanOverrideStereotype function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Element	EA.Element Direction: IN Description: Specifies the element showing a context menu that includes the option for new child diagrams.

Return Value

- Return a positive value to allow the stereotype from the type to override the current stereotype
- Return zero to prevent any change to the element stereotype
- Return a negative value to use the behavior from the base rules

EARules_CanProxy

EARules_CanProxy allows an Add-In registered to provide rules for a language to specify that one element is a proxy for another.

This event should only be handled for rules extending BPMN.

Syntax

Function EARules_CanProxy (Repository As EA.Repository, Language As String, Element as EA.Element) As Integer

The EARules_CanProxy function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Target	EA.Element Direction: IN Description: Specifies the element showing a context menu that includes the option for new child diagrams.

Return Value

- Return a positive value to specify that the target element is a proxy
- Return zero to specify that the target element is not a proxy
- Return a negative value to use the behavior from the base rules

EARules_CanReparent

EARules_CanReparent allows an Add-In registered to provide rules for a language to specify that the child diagrams for an element can be changed to other diagrams.

Syntax

Function EARules_CanReparent (Repository As EA.Repository, Language As String, Element as EA.Element) As Integer

The EARules_CanReparent function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Element	EA.Element Direction: IN Description: Specifies the element showing a context menu that includes the option for new child diagrams.

Return Value

- Return a positive value to allow reparenting to occur
- Return zero to prevent reparenting
- Return a negative value to use the behavior from the base rules

EARules_CreateModel

EARules_CreateModel allows an Add-In registered to provide rules for a language to create a wrapping element when a new diagram is created.

This event occurs during diagram creation.

Syntax

Function EARules_CreateModel (Repository As EA.Repository, Language As String, Diagram as EA.Diagram) As Integer

The EARules_CreateModel function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Diagram	EA.Diagram Direction: IN Description: Specifies the diagram currently being created.

Return Value

- Return zero or a positive value if no further action is required
- Return a negative value to use the behavior from the base rule

EARules_EnableElementProperty

EARules_EnableElementProperty allows an Add-In registered to provide rules for a language to control if individual properties should be displayed as read-only in the docked Properties window.

This is called during selection of elements when the Properties window is visible.

Syntax

Function EARules_EnableElementProperty (Repository As EA.Repository, Language As String, Element as EA.Element, Namespace as String, Class as String, Property as String) As Integer

The EARules_EnableElementProperty function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Element	EA.Element Direction: IN Description: Specifies the element currently being drawn.
Namespace	String Direction: IN Description: Specifies the top level language this property comes from. This will be either "UML" or the name of a profile.
Class	String Direction: IN Description: Specifies the type this property was defined in. In the UML namespace that means the metaclass defined in UML. Otherwise it will be a stereotype.
Property	String Direction: IN Description: Specifies the metaclass or stereotype property to enable or disable.

Return Value

- Return a positive value to allow edits to the property

- Return zero to disable edits to the property
- Return a negative value to use the behavior from the base rules

EARules_ForceLength

EARules_ForceLength allows an Add-In registered to provide rules for a language to specify that some elements are sized to the width or height of the diagram.

This event occurs during diagram load and resize events.

Syntax

Function EARules_ForceLength (Repository As EA.Repository, Language As String, Diagram as EA.Diagram) As Integer

The EARules_ForceLength function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Diagram	EA.Diagram Direction: IN Description: Specifies the diagram currently being loaded.

Return Value

- Return a positive value to specify that this diagram should enforce the length of elements
- Return zero to specify that no elements should have their length enforced
- Return a negative value to use the behavior from the base rules

EARules_GetEquivalentDiagram

EARules_GetEquivalentDiagram allows an Add-In registered to provide rules for a language to override the diagram type created when a UML diagram would otherwise be created.

This event occurs during user requests to create a structure that includes a diagram.

Syntax

Function EARules_GetEquivalentDiagram (Repository As EA.Repository, Language As String, DiagramType as String) As String

The EARules_GetEquivalentDiagram function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
DiagramType	String Direction: IN Description: Specifies the type of diagram that has been requested.

Return Value

A string containing the qualified name of a diagram type to replace the UML diagram requested.

Return an empty string to allow the base rules to control the diagram type.

EARules_IsAdjustable

EARules_IsAdjustable allows an Add-In registered to provide rules for a language to specify which elements are sized to the width or height of the diagram.

This event occurs during diagram load and resize events.

Syntax

Function EARules_IsAdjustable (Repository As EA.Repository, Language As String, Element as EA.Element) As Integer

The EARules_IsAdjustable function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Element	EA.Element Direction: IN Description: Specifies the element.

Return Value

- Return a positive value to allow automatic resize to occur
- Return zero to prevent automatic resize
- Return a negative value to use the behavior from the base rules

EARules_PropagateStereotype

EARules_PropagateStereotype allows an Add-In registered to provide rules for a language to control if a particular stereotype from a classifier should be applied to an instance with that classifier.

This event occurs when a classifier is assigned to an instance.

Syntax

Function EARules_PropagateStereotype (Repository As EA.Repository, Language As String, Element as EA.Element, Stereotype as String) As Integer

The EARules_PropagateStereotype function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Element	EA.Element Direction: IN Description: Specifies the element showing a context menu that includes the option for applying the stereotype.
Stereotype	String Direction: IN Description: Specifies the qualified name of the stereotype to apply.

Return Value

- Return a positive value to allow the stereotype from the type to be applied
- Return zero to prevent the stereotype from being applied
- Return a negative value to use the behavior from the base rules

EARules_ShowElementProperty

EARules_ShowElementProperty allows an Add-In registered to provide rules for a language to control visibility for individual properties in the docked Properties window.

This is called during selection of elements when the Properties window is visible.

Syntax

Function EARules_ShowElementProperty (Repository As EA.Repository, Language As String, Element as EA.Element, Namespace as String, Class as String, Property as String) As Integer

The EARules_ShowElementProperty function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize
Element	EA.Element Direction: IN Description: Specifies the element currently being drawn.
Namespace	String Direction: IN Description: Specifies the top level language this property comes from. Will either be "UML" or the name of a profile.
Class	String Direction: IN Description: Specifies the type this property was defined in. In the UML namespace that means the metaclass defined in UML. Otherwise it will be a stereotype.
Property	String Direction: IN Description: Specifies the metaclass or stereotype property to display or hide.

Return Value

- Return a positive value to display the property

- Return zero to hide the property
- Return a negative value to use the behavior from the base rules

EARules_ShowFrame

EARules_ShowFrame allows an Add-In registered to provide rules for a language to determine if the owner of a diagram should always be displayed on the diagram as a diagram frame.

This event occurs during diagram load.

Syntax

Function EARules_ShowFrame (Repository As EA.Repository, Language As String, Diagram as EA.Diagram) As Integer

The EARules_ShowFrame function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Diagram	EA.Diagram Direction: IN Description: Specifies the diagram currently being loaded.

Return Value

- Return a positive value to specify that the parent element should be shown as a frame on this diagram
- Return zero to specify the frame should not be displayed on the diagram
- Return a negative value to use the behavior from the base rules

EARules_ShowParentFrame

EARules_ShowParentFrame allows an Add-In registered to provide rules for a language to determine if the owner of a diagram should always be displayed on the diagram as a diagram frame.

This event occurs during diagram load.

Syntax

Function EARules_ShowParentFrame (Repository As EA.Repository, Language As String, Diagram as EA.Diagram) As Integer

The EARules_ShowParentFrame function syntax contains these parameters.

Parameter	Description
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Language	String Direction: IN Description: Specifies the language for the rule that Enterprise Architect is requesting. This will match one of the values returned from EARules_Initialize.
Diagram	EA.Diagram Direction: IN Description: Specifies the diagram currently being loaded.

Return Value

- Return a positive value to specify that the parent element should be shown as a frame on this diagram
- Return zero to specify the name should not be displayed on the diagram
- Return a negative value to use the behavior from the base rules

Custom Views

Enterprise Architect enables custom windows to be inserted as a tab within the Diagram View that appears at the center of the Enterprise Architect frame.

Creating a custom view helps you to easily display a custom interface within Enterprise Architect, alongside other diagrams and built-in views for quick and easy access.

Uses for this facility include:

- Reports and graphs showing summary data of the model
- Alternative views of a diagram
- Alternative views of the model
- Views of external data related to model data
- Documentation tools

Bear in mind that the 'Open Diagrams in Single Window' option in the 'Application Look' dialog will swap diagrams in the Diagram View rather than open more diagram tabs.

Create a Custom View

A custom view must be designed as an ActiveX Custom Control and inserted via the Automation Interface. ActiveX Custom Controls can be created using most well-known programming tools, including Microsoft Visual Studio. See the documentation provided by the relevant vendor on how to create a custom control to produce an OCX file.

Once the custom control has been created and registered on the target system, it can be added through the AddTab() method of the Repository object. While it is possible to call AddTab() from any automation client, it is likely that you would call it from an Add-In, and that the Add-In is defined in the same OCX that provides the custom view.

C# Code Example

```
public class Addin
{
    UserControl1 m_MyControl;
    public void EA_Connect(EA.Repository Rep)
    {
    }
    public object EA_GetMenuItems(EA.Repository Repository, string Location, string MenuName)
    {
        if(MenuName == "")
             return "-&C# Control Demo";
        else
        {
            String() ret = {"Show Custom View", "Show Button"};
            return ret;
        }
    }
    public void EA_MenuClick(EA.Repository Rep, string Location, string MenuName, string ItemName)
    {
        if(ItemName == "Show Custom View")
            m_MyControl = (UserControl1) Rep.AddTab("C# Demo","ContDemo.UserControl1");
        else if(ItemName == "Show Button")
            m_MyControl.ShowButton();
    }
}
```

Custom Docked Window

Custom docked windows can be added into the Enterprise Architect user interface. Once added, they can be shown and docked in the same way as other built-in Enterprise Architect docked windows.

A custom docked window must be designed as an ActiveX Custom Control and inserted via the Automation Interface. ActiveX Custom Controls can be created using most well-known programming tools, including Microsoft Visual Studio. See the documentation provided by the relevant vendor on how to create a custom control to produce an OCX file.

Once the custom control has been created and registered on the target system, it can be added using the AddWindow() method of the Repository object. While it is possible to call AddWindow() from any automation client, it is likely that you would call it from an Add-In, and that the Add-In is defined in the same OCX that provides the custom view.

To view custom docked windows that have been added, select the 'Specialize > Add-Ins > Addin Windows' ribbon option.

Custom docked windows can also be made visible by the automation client or Add-In using the ShowAddinWindow() method, or hidden by using the HideAddinWindow() method.

C# Code Example

```
public class Addin
{
    UserControl1 m_MyControl;
    public void EA_Connect(EA.Repository Rep)
    {
        m_MyControl = (UserControl1) Rep.AddWindow
        ("C# Demo","ContDemo.UserControl1");
    }
    public object EA_GetMenuItems(EA.Repository Repository, string Location, string MenuName)
    {
        if(MenuName == "")
            return "-&C# Control Demo";
       else
        {
            String() ret = {"Show Window", "Show Button"};
            return ret;
        }
    }
    public void EA_MenuClick(EA.Repository Rep, string Location, string MenuName, string ItemName)
    {
        if(ItemName == "Show Window")
            Rep.ShowAddinWindow("C# Demo");
        else if(ItemName == "Show Button")
            m_MyControl.ShowButton();
    }
}
```

MDG Add-Ins

MDG Add-Ins are specialized types of Add-In that have additional features and extra requirements, for Add-In authors who want to contribute to Enterprise Architect's goal of Model Driven Generation.

One of the additional responsibilities of an MDG Add-In is to take ownership of a branch of an Enterprise Architect model, which is done through the MDG_Connect event. Unlike general Add-In events, MDG Add-In events are only sent to the Add-In that has taken ownership of an Enterprise Architect model branch on a particular workstation.

MDG Add-Ins identify themselves as such during EA_Connect by returning the string 'MDG'.

Unlike ordinary Add-Ins, responding to MDG Add-In events is not optional, and methods must be published for each of the MDG Events.

MDG Events

An MDG Add-In must respond to all MDG Events. These events usually identify processes such as Build, Run, Synchronize, PreMerge and PostMerge, amongst others.

An MDG Link Add-In is expected to implement some form of forward and reverse engineering capability within Enterprise Architect, and as such requires access to a specific set of events, all to do with generation, synchronization and general processes concerned with converting models to code and code to models.

MDGAdd-In Events

Event
MDG_BuildProject
MDG_Connect
MDG_Disconnect
MDG_GetConnectedPackages
MDG_GetProperty
MDG_Merge
MDG_NewClass
MDG_PostGenerate
MDG_PostMerge
MDG_PreGenerate
MDG_PreMerge
MDG_PreReverse
MDG_RunExe
MDG_View

MDG_BuildProject

Add-Ins can use MDG_BuildProject to handle file changes caused by generation. This function is called in response to a user selecting the 'Execute > Source > Build > Build' ribbon option.

Respond to this event by compiling the project source files into a running application.

Syntax

Sub MDG_BuildProject (Repository As EA.Repository, PackageGuid As String)

The MDG_BuildProject function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the Add-In.

Return Value

None.

MDG_Connect

An Add-In uses MDG_Connect to handle a user driven request to connect a model branch to an external application. The function is called when the user attempts to connect a particular Enterprise Architect Package to an as yet unspecified external project. The Add-In calls the event to interact with the user to specify such a project.

The Add-In is responsible for retaining the connection details, which should be stored on a per-user or per-workstation basis. That is, users who share a common Enterprise Architect model over a network should be able to connect and disconnect to external projects independently of one another.

The Add-In should therefore not store connection details in an Enterprise Architect repository. A suitable place to store such details would be:

SHGetFolderPath(..CSIDL_APPDATA..)\AddinName

The PackageGuid parameter is the same identifier as is required for most events relating to the MDG Add-In. Therefore it is recommended that the connection details be indexed using the PackageGuid value.

The PackageID parameter is provided to aid fast retrieval of Package details from Enterprise Architect, should this be required.

Syntax

Function MDG_Connect (Repository As EA.Repository, PackageID as Long, PackageGuid As String) As Long

The MDG_Connect function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageID	Long Direction: IN Description: The PackageID of the Enterprise Architect Package the user has requested to have connected to an external project.
PackageGuid	String Direction: IN Description: The unique ID identifying the project provided by the Add-In when a connection to a project branch of an Enterprise Architect model was first established.

Return Value

- Return a non-zero to indicate that a connection has been made
- Return a zero to indicate that the user has not nominated a project and connection should not proceed

MDG_Disconnect

Add-Ins can use MDG_Disconnect to respond to user requests to disconnect the model branch from an external project.

This function is called when the user attempts to disconnect an associated external project. The Add-In is required to delete the details of the connection.

Syntax

Function MDG_Disconnect (Repository As EA.Repository, PackageGuid As String) As Long

The MDG_Disconnect function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the Add-In.

Return Value

- Return a non-zero to indicate that a disconnection has occurred, enabling Enterprise Architect to update the user interface
- Return a zero to indicate that the user has not disconnected from an external project

MDG_GetConnectedPackages

Add-Ins can use MDG_GetConnectedPackages to return a list of current connections between Enterprise Architect and an external application.

This function is called when the Add-In is first loaded, and is expected to return a list of the available connections to external projects for this Add-In.

Syntax

Function MDG_GetConnectedPackages (Repository As EA.Repository) As Variant

The MDG_GetConnectedPackages function syntax contains this parameter.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

Return Value

Returns an array of GUID strings representing individual Enterprise Architect Packages.

MDG_GetProperty

MDG_GetProperty provides miscellaneous Add-In details to Enterprise Architect.

This function is called by Enterprise Architect to poll the Add-In for information relating to the PropertyName. This event should occur in as short a duration as possible, as Enterprise Architect does not cache the information provided by the function.

Values corresponding to these PropertyNames must be provided:

- IconID - Return the name of a DLL and a resource identifier in the format #ResID, where the resource ID indicates an icon
  c:\program files\myapp\myapp.dlll#101
- Language - Return the default language that Classes should be assigned when they are created in Enterprise Architect
- HiddenMenus - Return one or more values from the MDGMenus enumeration to hide menus that do not apply to your Add-In
  if(PropertyName == "HiddenMenus")
  return mgBuildProject + mgRun;

Syntax

Function MDG_GetProperty (Repository As EA.Repository, PackageGuid As String, PropertyName As String) As Variant

The MDG_GetProperty function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the Add-In.
PropertyName	String Direction: IN Description: The name of the property that is used by Enterprise Architect. See the start of this topic for the possible values.

Return Value

See the start of this topic.

MDG_Merge

Add-Ins can use MDG_Merge to jointly handle changes to both the model branch and the code project that the model branch is connected to.

This event should be called whenever the user has asked to merge their model branch with its connected code project, or whenever the user has established a new connection to a code project.

The purpose of this event is to make the Add-In interact with the user to perform a merge between the model branch and the connected project.

Syntax

Function MDG_Merge (Repository As EA.Repository, PackageGuid As String, SynchObjects As Variant, SynchType As String, ExportObjects As Variant, ExportFiles As Variant, ImportFiles As Variant, IgnoreLocked As String, Language As String) As Long

The MDG_Merge function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the Add-In.
SynchObjects	Variant Direction: OUT Description: A string array containing a list of objects (Object ID format) to be jointly synchronized between the model branch and the project. See *Object ID Format* for the format of the Object IDs.
SynchType	String Direction: OUT Description: The value determining the user-selected type of synchronization to take place. See *Synchronize Type* for a list of valid values.
ExportObjects	Variant Direction: OUT Description: The string array containing the list of new model objects (in Object ID format) to be exported by Enterprise Architect to the code project.
ExportFiles	Variant Direction: OUT Description: A string array containing the list of files for each model object chosen

	for export by the Add-In. Each entry in this array must have a corresponding entry in the ExportObjects parameter at the same array index, so ExportFiles(2) must contain the filename of the object by ExportObjects(2).
ImportFiles	Variant Direction: OUT Description: A string array containing the list of code files made available to the code project to be newly imported to the model. Enterprise Architect imports each file listed in this array for import into the connected model branch.
IgnoreLocked	String Direction: OUT Description: A value indicating whether to ignore any files locked by the code project (that is, 'True' or False').
Language	String Direction: OUT Description: The string value containing the name of the code language supported by the code project connected to the model branch.

Object ID Format

Each of the Object IDs listed in the 'SynchObjects' string arrays should have this format:
(@namespace)*(#class)*($attribute|%operation|:property)*

Return Value

- Return a non-zero if the merge operation completed successfully
- Return a zero when the operation has been unsuccessful

Merge

A merge consists of three major operations:

- Export: where newly created model objects are exported into code and made available to the code project
- Import: where newly created code objects, Classes and such things are imported into the model
- Synchronize: where objects available both to the model and in code are jointly updated to reflect changes made in either the model, code project or both

Synchronize Type

The Synchronize operation can take place in one of four different ways. Each of these ways corresponds to a value returned by 'SynchType':

- None: (SynchType' = 0) No synchronization is to be performed
- Forward: ('SynchType' = 1) Forward synchronization, between the model branch and the code project is to occur
- Reverse: ('SynchType = 2) Reverse synchronization, between the code project and the model branch is to occur
- Both: ('SynchType' = 3) Reverse, then Forward synchronizations are to occur

MDG_NewClass

Add-Ins can use MDG_NewClass to alter details of a Class before it is created.

This method is called when Enterprise Architect generates a new Class, and requires information relating to assigning the language and file path. The file path should be passed back as a return value and the language should be passed back via the language parameter.

Syntax

Function MDG_NewClass (Repository As EA.Repository, PackageGuid As String, CodeID As String, Language As String) As String

The MDG_NewClass function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the Add-In.
CodeID	String Direction: IN Description: A string used to identify the code element before it is created.
Language	String Direction: OUT Description: A string used to identify the programming language for the new Class. The language must be supported by Enterprise Architect.

Return Value

Return a string containing the file path that should be assigned to the Class.

MDG_PostGenerate

Add-Ins can use MDG_PostGenerate to handle file changes caused by generation.

This event is called after Enterprise Architect has prepared text to replace the existing contents of a file. Responding to this event enables the Add-In to write to the linked application's user interface rather than modify the file directly.

When the contents of a file are changed, Enterprise Architect passes FileContents as a non-empty string. New files created as a result of code generation are also sent through this mechanism, so the Add-Ins can add new files to the linked project's file list.

When new files are created Enterprise Architect passes FileContents as an empty string. When a non-zero is returned by this function, the Add-In has successfully written the contents of the file. A zero value for the return indicates to Enterprise Architect that the file must be saved.

Syntax

Function MDG_PostGenerate (Repository As EA.Repository, PackageGuid As String, FilePath As String, FileContents As String) As Long

The MDG_PostGenerate function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the Add-In.
FilePath	String Direction: IN Description: The path of the file Enterprise Architect intends to overwrite.
FileContents	String Direction: IN Description: A string containing the proposed contents of the file.

Return Value

The return value depends on the type of event that this function is responding to (see introduction). This function is required to handle two separate and distinct cases.

MDG_PostMerge

MDG_PostMerge is called by Enterprise Architect after a merge process has been completed.

File save checking should not be performed with this function, but should be handled by MDG_PreGenerate, MDG_PostGenerate and MDG_PreReverse.

Syntax

Function MDG_PostMerge (Repository As EA.Repository, PackageGuid As String) As Long

The MDG_PostMerge function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the Add-In.

Return Value

- Return a non-zero to indicate that the post-merge has been successful
- Return a zero if the post-merge process has failed

Enterprise Architect assumes a non-zero return if this method is not implemented.

MDG_PreGenerate

Add-Ins can use MDG_PreGenerate to deal with unsaved changes.

This function is called immediately before Enterprise Architect attempts to generate files from the model. A possible use of this function would be to prompt the user to save unsaved source files.

Return Value

- Return a zero to abort generation
- Return any other value to enable the generation to continue

Syntax

Function MDG_PreGenerate (Repository As EA.Repository, PackageGuid As String) As Long

The MDG_PreGenerate function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the Add-In.

MDG_PreMerge

MDG_PreMerge is called after a merge process has been initiated by the user and before Enterprise Architect performs the merge process.

This event is called after a user has performed their interactions with the merge screen and has confirmed the merge with the OK button, but before Enterprise Architect performs the merge process using the data provided by the MDG_Merge call, before any changes have been made to the model or the connected project.

This event is made available to provide the Add-In with the opportunity to generally set internal Add-In flags to augment the MDG_PreGenerate, MDG_PostGenerate and MDG_PreReverse events.

File save checking should not be performed with this function, but should be handled by MDG_PreGenerate, MDG_PostGenerate and MDG_PreReverse.

Syntax

Function MDG_PreMerge (Repository As EA.Repository, PackageGuid As String) As Long

The MDG_PreMerge function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the Add-In.

Return Value

- Return a zero to indicate that the merge process can not occur
- Return a non-zero if the merge process proceeds

If this method is not implemented then it is assumed that a merge process is used.

MDG_PreReverse

Add-Ins can use MDG_PreReverse to save file changes before they are imported into Enterprise Architect.

This function operates on a list of files that are about to be reverse-engineered into Enterprise Architect. If the user is working on unsaved versions of these files in an editor, you could either prompt the user or save automatically.

Syntax

Sub MDG_PreReverse (Repository As EA.Repository, PackageGuid As String, FilePaths As Variant)

The MDG_PreReverse function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the Add-In.
FilePaths	String array Direction: IN Description: An array of filepaths pointed to the files that are to be reverse engineered.

Return Value

None.

MDG_RunExe

Add-Ins can use MDG_RunExe to run the target application.

This function is called when the user selects the 'Execute > Run > Start > Run' ribbon option.

Respond to this event by launching the compiled application.

Syntax

Sub MDG_RunExe (Repository As EA.Repository, PackageGuid As String)

The MDG_RunExe function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the Add-In.

Return Value

None.

MDG_View

Add-Ins can use MDG_View to display user specified code elements.

This function is called by Enterprise Architect when the user asks to view a particular code element. The Add-In can then present that element in its own way, usually in a code editor.

Syntax

Function MDG_View (Repository As EA.Repository, PackageGuid As String, CodeID as String) As Long

The MDG_View function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the Add-In.
CodeID	String Direction: IN Description: Identifies the code element in this format: <type>ElementPart<type>ElementPart... where each element is proceeded with a token identifying its type: @ -namespace # - Class $ - attribute % - operation For example, if a user has selected the m_Name attribute of Class1 located in namespace Name1, the Class ID would be passed through in this format: @Name1#Class1%m_Name

Return Value

- Return a non-zero value to indicate that the Add-In has processed the request
- Return a zero value for Enterprise Architect to employ the standard viewing process, which is to launch the associated source file

Workflow Add-In Events

Enterprise Architect provides this set of four additional events that are sent only to workflow Add-Ins.

To use these the Workflow Add-In must be initialized with EA_Connect set to type: "Workflow". For more details see the *EA_Connect* Help topic.

Workflow Add-In Events

Event
EA_AllowPropertyUpdate This event is sent to workflow Add-Ins after a user has changed a built-in property value.
EA_AllowTagUpdate This event is sent to workflow Add-Ins after a user has changed a Tagged Value.
EA_CanEditProperty This event is sent to workflow Add-Ins when a property is being displayed that allows the property to block all edits.
EA_CanEditTag This event is sent to workflow Add-Ins when a Tagged Value is being displayed that allows the property to block all edits.

EA_AllowPropertyUpdate

This event is sent to workflow Add-Ins after a user has changed a built-in property value.

Syntax

Function EA_AllowPropertyUpdate (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects describing the requested property update: • Type: A string value corresponding to Element.Type • Stereotype: A string value corresponding to Element.Stereotype • Property: The name of the property field to enable or disable • OldValue: The previous value of the property • NewValue: The new value of the property

Return Value

- Return False to prevent this change to the described property
- Return True to allow this change

EA_AllowTagUpdate

This event is sent to Workflow Add-Ins after a user has changed a Tagged Value.

Syntax

Function EA_AllowTagUpdate (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects describing the requested Tagged Value update: • Type: A string value corresponding to Element.Type • Stereotype: A string value corresponding to Element.Stereotype • TagName: The name of the Tagged Value field to enable or disable • OldValue: The previous value of the tag • NewValue: The new value of the tag

Return Value

- Return False to prevent this change to the described Tagged Value
- Return True to allow this change

EA_CanEditProperty

This event is sent to Workflow Add-Ins when a property is being displayed that allows the property to block all edits.

Syntax

Function EA_CanEditProperty (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects describing the property: • Type: A string value corresponding to Element.Type • Stereotype: A string value corresponding to Element.Stereotype • PropertyName: The name of the property field to enable or disable

Return Value

- Return False to prevent all edits to the described property
- Return True to allow changes

EA_CanEditTag

This event is sent to Workflow Add-Ins when a Tagged Value is being displayed that allows the property to block all edits.

Syntax

Function EA_CanEditTag (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects describing the Tagged Value: • Type: A string value corresponding to Element.Type • Stereotype: A string value corresponding to Element.Stereotype • TagName: The name of the tag to enable or disable

Return Value

- Return False to prevent all edits to the described Tagged Value
- Return True to allow changes

Enterprise Architect Object Model

The Enterprise Architect Object Model gives the scripter or programmer access to the underlying objects that you can use to query or manipulate the repository. The Object Model is accessible either from internal or external scripting environments or through Add-Ins. This is a convenient feature that ensures that a programmer is insulated from the underlying database where the repository is stored, protecting them from changes to the database structure or content. The objects are grouped into Packages and contain a useful, extensive and well documented set of properties and methods that are intuitive to use and allow access to elements, features, diagrams and project metadata.

Automation provides a way for other applications to access the information in an Enterprise Architect model using Windows OLE Automation (ActiveX). Typically this involves scripting clients such as MS Word™ or Visual Basic, or using scripts created within Enterprise Architect using the Scripting window.

The Automation Interface provides a way of accessing the internals of Enterprise Architect models. Examples of things you can do using the Automation Interface include:

- Perform repetitive tasks, such as update the version number for all elements in a model
- Generate code from a StateMachine diagram
- Produce custom reports
- Perform ad hoc queries

Features

Feature	Description
Connecting to the Automation Interface	All development environments capable of generating ActiveX COM clients should be able to connect to the Enterprise Architect Automation Interface. This guide provides detailed instructions on connecting to the interface using Microsoft Visual Basic 6.0, Borland Delphi 7.0, Microsoft C# and Java. There are also more detailed steps on how to set-up Visual Basic; the principles are applicable to other languages.
Examples and Tips	Instruction on how to use the Automation Interface is provided by means of sample code. See pointers to the samples and other available resources. Also, consult the extensive Reference Section.
Calling Executables from Enterprise Architect	Enterprise Architect can be set up to call an external application. You can pass parameters on the current position selected in the Browser window to the application being called. For instructions, go to the *Call from Enterprise Architect* topic. A more sophisticated method is to create Add-Ins, which are discussed in a separate section.

Using the Automation Interface

This section provides instructions on how to connect to and use the Automation Interface, including:

- Connecting to the interface
- Setting references in Visual Basic
- Examples and Tips

Connect to the Interface

All development environments capable of generating ActiveX Com clients can connect to the Enterprise Architect Automation Interface.

By way of example, these sections describe how to connect using several such tools. The procedure might vary slightly with different versions of these products.

Microsoft Visual Basic 6.0

This procedure caters for the syntax and frameworks of version 6.0. More recent versions have the same framework as other .Net languages with only syntax differences, and therefore use a similar process to that described for Microsoft C#, later in this topic.

Step	Action
1	Create a new project.
2	Select the 'Project | References' menu option.
3	Select Enterprise Architect Object Model 2.0 from the list. If this does not appear, go to the command line and re-register Enterprise Architect using: EA.exe /unregister then EA.exe /register
4	See the general library documentation on the use of Classes. This example creates and opens a repository object: Public Sub ShowRepository() Dim MyRep As New EA.Repository MyRep.OpenFile "c:\eatest.eap" End Sub

Borland Delphi 7.0

Note that recent versions of Delphi are developed by Embarcadero.

Step	Action
1	Create a new project.
2	Select the 'Project | Import Type Library' menu option.
3	Select Enterprise Architect Object Model 2.0 from the list. If this does not appear, go to the command line and re-register Enterprise Architect using: EA.exe /unregister then EA.exe /register

4	Click on the Create Unit button.
5	Include EA_TLB in Project1's Uses clause.

6	See the general library documentation on the use of Classes. This example creates and opens a repository object:
	procedure TForm1.Button1Click(Sender: TObject);
	var
	r: TRepository;
	b: boolean;
	begin
	r:= TRepository.Create(nil);
	b:= r.OpenFile('c:\eatest.eap');
	end;

Microsoft C#

Step	Action
1	Select the 'Visual Studio Project | Add Reference' menu option.
2	Click on the 'Browse' tab.
3	Navigate to the folder in which you installed Enterprise Architect; usually: Program Files/Sparx Systems/EA Select Interop.EA.dll

4	See the general library documentation on the use of Classes. This example creates and opens a repository object:
	private void button1_Click(object sender, System.EventArgs e)
	{
	EA.Repository r = new EA.Repository();
	r.OpenFile("c:\\eatest.eap");
	}

Java

Step	Action
1	Copy the file: SSJavaCOM.dll

	from the Java API subdirectory of your installed directory, usually: Program Files/Sparx Systems/EA into any location within the Windows PATH windows\system32 directory. Note: The Java API loads the last installed Enterprise Architect and isn't affected when using either the 32 or 64 Version of DLL, as long as the SSJavaCOM dll can be found by the java runtime.
2	Copy the file eaapi.jar from the Java API subdirectory of your installed directory, usually: Program Files/Sparx Systems/EA to a location in the Java CLASSPATH or where the Java class loader can find it at run time.

3	All of the Classes described in the documentation are in the Package org.sparx. See the general library documentation for their use. This example creates and opens a repository object:
	public void OpenRepository()
	{
	org.sparx.Repository r = new org.sparx.Repository();
	r.OpenFile("c:\\eatest.eap");
	}

Set References In Visual Basic

It is possible to use the Enterprise Architect ActiveX interface with Visual Basic (VB). Use is ensured for Visual Basic version 6, but might vary slightly with versions other than version 6.

It is assumed that you have accessed VB through a Microsoft Application such as VB 6.0, MS Word™ or MS Access. If the code is not called from within Word, the Word VB reference must also be set.

On creating a new VB project, you set a reference to an Enterprise Architect Type Library and a Word Type Library.

Set References

Step	Action
1	Select the 'Tools | References' menu option.
2	Select the 'Enterprise Architect Object Model 2.10' checkbox from the list.
3	Do the same for VB or VB Word: select the checkbox for the 'Microsoft Word 10.0 Object Library'.
4	Click on the OK button.

Notes

- If 'Enterprise Architect Object Model 2.10' does not appear in the list, go to the command line and manually re-enter Enterprise Architect using:
  - (To unregister Enterprise Architect) ea.exe /unregister
  - (To register Enterprise Architect) ea.exe /register
- Visual Basic 5/6 users should also note that the version number of the Enterprise Architect interface is stored in the VBP project file in a form similar to this:
  Reference=*\G{64FB2BF4-9EFA-11D2-8307-C45586000000}#2.2#0#..\..\..\..\Program Files\
  Sparx Systems\EA\EA.TLB#Enterprise Architect Object Model 2.02
  If you experience problems moving from one version of Enterprise Architect to another, open the VBP file in a text editor and remove this line, then open the project in Visual Basic and use Project-References to create a new reference to the Enterprise Architect Object model
  Reference to objects in Enterprise Architect and Word should now be available in the Object Browser, which can be accessed from the main menu by pressing F2
  The drop-down list on the top-left of the window should now include Enterprise Architect and Word; if MS-Project is installed, also set this up

Examples and Tips

Points to consider

Subject	Points
Examples	Instructions for using the interface are provided through sample code. There are several sets of examples: • VB 6 and C# examples are available in the Code Samples folder under your Enterprise Architect installation (default: C:\Program Files\Sparx Systems\EA\Code Samples) • Enterprise Architect can be set up to call an external application • Several VB.NET code snippets are provided in the reference section • A comprehensive example of using Visual Basic to create MS Word™ documentation is available from the internet at **sparxsystems.com/resources/developers/autint_vb.html** • Additional samples are available from the Sparx Systems website; see the *Available Resources* topic
Tips and Tricks	Also note these tips and tricks: • An instance of the Enterprise Architect (EA.exe) process is executed when you initialize a new repository object - this process must remain running in order to perform automation tasks; if the main window is visible, you can safely minimize it, but it must remain running • The Enterprise Architect ActiveX Interface is a functional interface rather than a data interface; when you load data through the interface there is a noticeable delay as Enterprise Architect user interface elements (such as Windows and menus) are loaded and the specified database connection is established • Collections use a zero-based index; for example, Repository.Models(0) represents the first model in the repository • During the development of your client software your program might terminate unexpectedly and leave EA.exe running in such a state that it is unable to support further interface calls; if your program terminates abnormally, ensure that Enterprise Architect is not left running in the background (see the Windows 'Task Manager / Process' tab) • A handle to a currently running instance of Enterprise Architect can be obtained through the use of a GetObject() call (see the reference page for the App object); accessing your Enterprise Architect model via the App object enables querying the current User Interface status, such as using GetContextItem() on the Repository object to detect the current selection by the user, allowing for rapid prototyping and testing

Enterprise Architect Not Closing	After all processing by an automation controller is complete, it is recommended to call CloseFile() and Exit() on the Repository object, then set all references to the repository object to null.
	repository.CloseFile();
	repository.Exit();
	repository = null;
	If your automation controller was written using the .NET framework, Enterprise Architect does not close even after you release all your references to it. To force the

	release of the COM pointers, call the memory management functions: GC.Collect(); GC.WaitForPendingFinalizers(); There are additional concerns when controlling a running instance of Enterprise Architect that loads Add-Ins - see the *Tricks and Traps* topic for details.

Call from Enterprise Architect

Enterprise Architect can be set up to call an external application. You can pass parameters on the current position selected in the Browser window to the application being called. This helps you to:

- Add a command line for an application
- Define parameters to pass to this application

The parameters required for running the AutInt executable are:

- The Enterprise Architect file parameter $f and
- The current PackageID $p

Hence the arguments should simply contain: $f,$p.

Once this has been set up, the application can be called from the 'Extend' ribbon in Enterprise Architect using the 'Extend > <YourApplication>' option.

Access

Ribbon	Start > Appearance > Preferences > Other Options > Tools

Parameters to pass information to external applications

Parameter	Description
$d	Diagram ID Notes: ID for accessing associated diagram.
$D	Diagram GUID Notes: GUID for accessing the associated diagram.
$e	Comma separated list of element IDs Notes: All elements selected in the current diagram.
$E	Comma separated list of element GUIDs Notes: All elements selected in the current diagram.
$f	Project Name Notes: For example: C:\projects\EAexample.eap.
$F	Calling Application (Enterprise Architect) Notes: 'Enterprise Architect'.
$p	Current Package ID Notes: For example: 144.
$P	Package GUID

	Notes: GUID for accessing this Package.

Available Resources

Resources

Available resources include:

Resource	Download Link
VB 6 Add-In for generating MS Word documentation.	sparxsystems.com/resources/developers/autint_vb.html
VB 6 Add-In to display a custom ActiveX graph control within the Enterprise Architect window as a new view.	sparxsystems.com/resources/developers/autint_vb_custom_view.html
A basic Add-In framework written in C#. Useful as a starting point for authoring your own custom Enterprise Architect Add-In.	sparxsystems.com/bin/CS_AddinFramework.zip
An extension on the CS_AddinFramework example showing how to export Tagged Values to a .csv file.	sparxsystems.com/bin/CS_AddinTaggedCSV.zip
A basic Add-In skeleton written in Delphi.	sparxsystems.com/bin/DelphiDemo.zip
A simple example Add-In written in C#.	sparxsystems.com/bin/CS_Sample.zip

Reference

This section provides detailed information on all the objects available in the object model provided by the Automation Interface, including:

Object Groups

Group
App Object
Enumerations
Repository Package
Element Package
Element Features Package
Connector Package
Diagram Package
Project Interface Package
Document Generator Interface Package
Mail Interface Package
Code Samples

Interface Overview

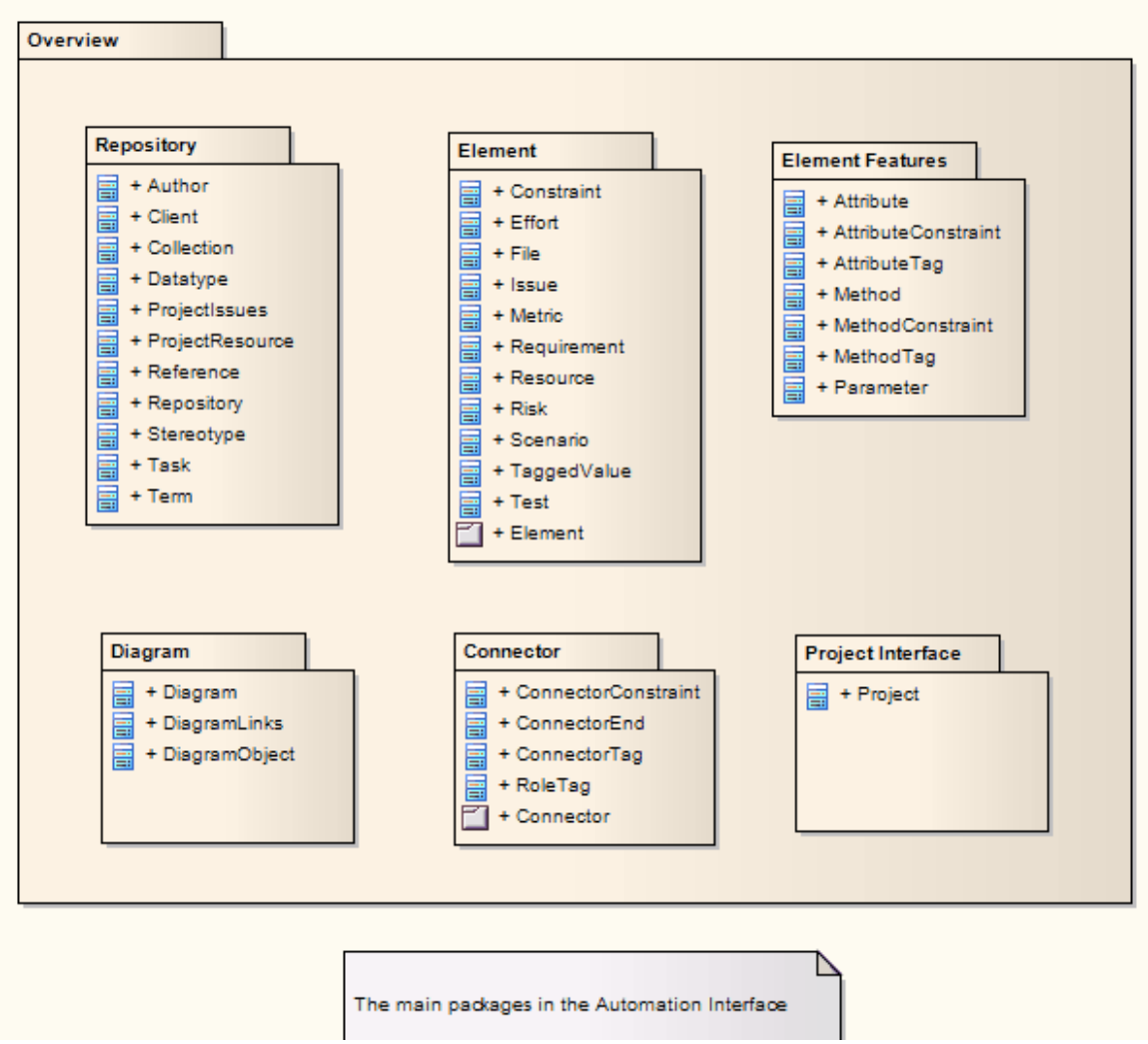
This section provides an overview of the main components of the Automation Interface.

Main Packages of Automation Interface

Package	Detail
Repository Package	Represents the model as a whole and provides entry to model Packages and collections.
Element Package	Identifies the basic structural units (such as Class, Use Case and Object).
Element Features Package	Identifies the attributes and operations defined on an element.
Diagram Package	Describes the visible drawings contained in the model.
Connector Package	Defines the relationships between elements.

Packages and Contents

This diagram illustrates the main interface Packages and their associated contents. Each UML element in this User Guide can be created by Automation and can be accessed either through the various collections that exist or, in some cases, directly.



The Repository Class is the starting point for all use of the Automation Interface. It contains the high level system objects and entry point into the model itself using the Models collection and the other system-level collections.

App Object

The App object represents a running instance of Enterprise Architect. Its object provides access to the Automation Interface.

Attributes

Attribute	Type
Project	Project Notes: Read only Provides a handle to the Project Interface.
Repository	Repository Notes: Read only Provides a handle to the Repository object.
Visible	Boolean Notes: Read/Write Whether or not the application is visible.

GetObject() Support

The App object is createable and a handle can be obtained by creating one. In addition, clients can use the equivalent of Visual Basic's GetObject() to obtain a reference to a currently running instance of Enterprise Architect.

Use this method to more quickly test changes to Add-Ins and external clients, as the Enterprise Architect application and data files do not have to be constantly re-loaded.

For example:

```
Dim App as EA.App
Set App = GetObject(,"EA.App")
MsgBox App.Repository.Models.Count
```

Another example, which uses the App object without saving it to a variable:

```
Dim Rep as EA.Repository
Set Rep = GetObject(, "EA.App").Repository
MsgBox Rep.ConnectionString
```

Enumerations

These enumerations are defined by the Automation Interface:

Automation Interface Enumerations

Enumeration	Link
Constant Layout Styles	Constant Layout Styles
Create Baseline Flag	Create Baseline Flag
Create Model Type	Create Model Type
Document Break	Document Break
Document Page Orientation	Document Page Orientation
Document Type	Document Type
Enterprise Architect Edition Types	Enterprise Architect Edition Types
Enumeration Relation Set Type	Enumeration Relation Set Type
Export Package XMI Flag	Export Package XMI Flag
Mail Interface Message Flag	Mail Interface Message Flag
MDG Menus	MDG Menus
Object Type	Object Type
PropType	PropType
Reload Type	Reload Type
Scenario Diagram Type	Scenario Diagram Type
Scenario Step Type	Scenario Step Type
Scenario Test Type	Scenario Test Type
XMI Type	XMI Type

ConstLayoutStyles

The enum values defined here are used exclusively for the 'Lay Out a Diagram' method. You use these values to define the layout options as provided by the 'Layout > Tools > Diagram Layout ' ribbon option.

Enum Values

Value	Meaning
lsCrossReduceAggressive	Perform aggressive Cross-reduction in the layout process (time consuming).
lsCycleRemoveDFS	Use the Depth First Cycle Removal algorithm.
lsCycleRemoveGreedy	Use the Greedy Cycle Removal algorithm.
lsDiagramDefault	Use existing layout options specified for this diagram.
lsInitializeDFSIn	Initialize the layout using the Depth First Search Inward algorithm.
lsInitializeNaive	Initialize the layout using the Naïve Initialize Indices algorithm.
lsInitializeDFSOut	Initialize the layout using the Depth First Search Outward algorithm.
lsLayeringLongestPathSink	Layer the diagram using the Longest Path Sink algorithm.
lsLayeringLongestPathSource	Layer the diagram using the Longest Path Source algorithm.
lsLayeringOptimalLinkLength	Layer the diagram using the Optimal Link Length algorithm.
lsLayoutDirectionDown	Direct connectors to point down.
lsLayoutDirectionLeft	Direct connectors to point left.
lsLayoutDirectionRight	Direct connectors to point right.
lsLayoutDirectionUp	Direct connectors to point up.
lsProgramDefault	Use factory default layout options as specified by Enterprise Architect.

CreateBaselineFlag

The CreateBaselineFlag enumeration is used in Baseline Management, when creating a Baseline.

Enum Values

Value	Meaning
cbSaveToStub	Baseline this Package with only immediate children (child Packages are included as stubs only).

CreateModelType

The CreateModelType enumeration is used in the CreateModel method on the Repository Class.

Enum Values

Value	Meaning
cmEAPFromBase	Create a copy of the EABase model file to the specified file path.
cmEAPFromSQLRepository	Create a .eap file shortcut to an SQL-based repository; requires user interaction to provide SQL connection details.

DocumentBreak

The DocumentBreak enumeration is used in the InsertBreak method on the DocumentGenerator Class.

Enum Values

Value	Meaning
breakPage	Insert a page break in the document.
breakSection	Insert a section break in the document.

DocumentPageOrientation

The DocumentPageOrientation enumeration is used in the SetPageOrientation method on the DocumentGenerator Class.

Enum Values

Value	Meaning
pagePortrait	Sets the current page orientation to Portrait.
pageLandscape	Sets the current page orientation to Landscape.

DocumentType

The DocumentType enumeration is used in the SaveDocument method on the DocumentGenerator Class.

Enum Values

Value	Meaning
dtRTF	Save the document file to disk as an RTF document.
dtHTML	Save the document file to disk as a HTML document.
dtPDF	Save the document file to disk as a PDF document.
dtDOCX	Save the document file to disk as a DOCX document.

EAEditionTypes

The EAEditionTypes enumeration identifies the current level of licensed functionality available.

```
EAEditionTypes theEdition = theRepository.GetEAEdition();
if (theEdition == EAEditionTypes.piProfessional)
...
else if (theEdition == EAEditionTypes.piCorporate)
...
```

The enumeration defines these formal values:

- piLite
- piProfessional
- piCorporate
- piBusiness
- piSystemEng
- piUltimate

There is no separate value for the Trial Edition; the Repository.GetEAEdition() function returns the appropriate EAEditionTypes value for whichever edition the user has selected to trial.

EnumRelationSetType

This enumeration represents values returned from the GetRelationSet method of the Element object.

Enum Values

Value	Meaning
rsDependEnd	List of elements that depend on the current element.
rsDependStart	List of elements that the current element depends on.
rsGeneralizeEnd	List of elements that are generalized by the current element.
rsGeneralizeStart	List of elements that the current element generalizes.
rsParents	List of all parent elements of the current element.
rsRealizeEnd	List of elements that are realized by the current element.
rsRealizeStart	List of elements that the current element realizes.

ExportPackageXMIFlag

The ExportPackageXMIFlag enumeration is used in Package control, when exporting to XMI.

Enum Values

Value	Meaning
epExcludeEAExtensions	Export this Package without any tool specific information.
epSaveToStub	Export this Package with only immediate children (child Packages are included as stubs only).

MDGMenus

Use this enumeration when providing the 'HiddenMenus' property to MDG_GetProperty.
These options are exclusive of one another and can be read or added to hide more than one menu.

Enum Values

Value	Meaning
mgBuildProject	'Hide Build Project' menu option.
mgMerge	'Hide Merge' menu option.
mgRun	'Hide Run' menu option.

MessageFlag

The MessageFlag enumeration is used in both the SendMailMessage and ComposeMailMessage methods of the MailInterface, to specify a flag to attach to the message.

Enum Values

Value	Meaning
mfNone	Do not flag the message.
mfComplete	Flag the message as 'Complete'.
mfPurple	Flag the message with a 'Purple' flag.
mfOrange	Flag the message with an 'Orange' flag.
mfGreen	Flag the message with a 'Green' flag.
mfYellow	Flag the message with a 'Yellow' flag.
mfBlue	Flag the message with a 'Blue' flag.
mfRed	Flag the message with a 'Red' flag.

ObjectType

The ObjectType enumeration identifies Enterprise Architect object types even when referenced through a Dispatch interface. For example:

```
var treeSelectedType = Repository.GetTreeSelectedItemType();
switch (treeSelectedType)
{
    case otElement :
    {
        // Code for when an element is selected
        var theElement as EA.Element;
        theElement = Repository.GetTreeSelectedObject();
        break;
    }
    case otPackage :
    {
        // Code for when a Package is selected
        var thePackage as EA.Package;
        thePackage = Repository.GetTreeSelectedObject();
        break;
    }
}
```

Valid Enumeration Values

otAttribute
otAttributeConstraint
otAttributeTag
otAuthor
otClient
otCollection
otConnector
otConnectorConstraint
otConnectorEnd
otConnectorTag
otConstraint
otCustomProperty
otDatatype
otDiagram
otDiagramLink
otDiagramObject
otEffort

otElement
otEventProperties
otEventProperty
otFile
otIssue
otMailInterface
otMethod
otMethodConstraint
otMethodTag
otMetric
otModel
otNone
otPackage
otParameter
otParamTag
otPartition
otProject
otProjectIssues
otProjectResource
otProperties
otProperty
otPropertyType
otReference
otRepository
otRequirement
otResource
otRisk
otRoleTag
otScenario
otScenarioExtension
otScenarioStep
otStereotype
otSwimlane
otSwimlaneDef
otSwimlanes
otTaggedValue
otTask
otTerm
otTest
otTransition

PropType

The PropType enumeration gives the automation programmer an indication of what sort of data is going to be stored by this property.

Enum Values

Value	Meaning
ptArray	An array containing values of any type.
ptBoolean	True or False.
ptEnum	A string being an entry in the semi-colon separated list specified in the validation field of the Property.
ptFloatingPoint	4 or 8 byte floating point value.
ptInteger	16 bit or 32 bit signed integer.
ptString	Unicode string.

ReloadType

The ReloadType enumeration represents values returned from the GetReloadItem and PeekReloadItem methods of the ModelWatcher Class. It has four possible values, which define the type of change that was made to a model.

Enum Values

Value	Meaning
rtElement	The Item parameter represents a particular element that must be reloaded.
rtEntireModel	Entire model must be reloaded to ensure that all changes are reloaded.
rtNone	No change in the model.
rtPackage	The Item parameter represents a particular Package that must be reloaded.

ScenarioDiagramType

The ScenarioDiagramType enumeration provides these enumeration values to the Project.GenerateDiagramFromScenario() method. They specify the type of diagram to generate.

Enum Values

Value	Meaning
sdActivity	Generate an Activity diagram.
sdActivityWithAction	Generate an Activity diagram with an Action.
sdActivityWithActionPin	Generate an Activity diagram with an ActionPin.
sdActivityWithActivityParameter	Generate an Activity diagram with an ActivityParameter.
sdRobustness	Generate a Robustness diagram.
sdRuleFlow	Generate a RuleFlow diagram.
sdSequence	Generate a Sequence diagram.
sdState	Generate a StateMachine diagram.

ScenarioStepType

The ScenarioStepType enumeration is used to identify the steps of a scenario, and the entity performing the step.

Enum Values

Value	Meaning
stActor	Identify that the step is an action performed by an actor.
stSystem	Identify that the step is an action performed by the system.

ScenarioTestType

The ScenarioTestType enumeration provides these enumeration values to the Project.GenerateTestFromScenario() method, to specify the type of test to generate.

Enum Values

Value	Meaning
stHorizontalTestSuite	Generate a horizontal Test Suite diagram.
stVerticalTestSuite	Generate a vertical Test Suite diagram.
stExternal	Generate an external Test Case element.
stInternal	Generate an internal test.

XMIType

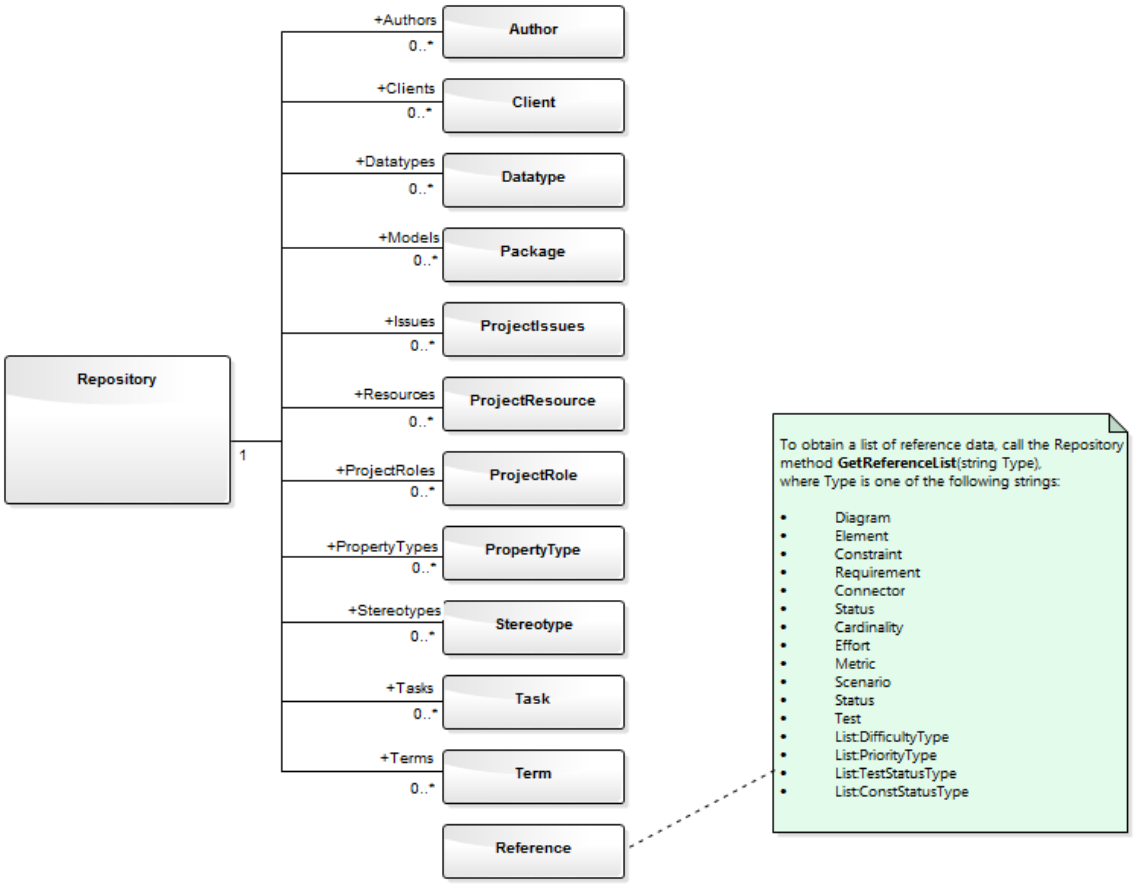
These enumeration values are used in the Project.ExportPackageXMI() and Project.ExportPackageXMIEx() methods, to specify the XMI export type.

- xmiEADefault = 0
- xmiRoseDefault = 1
- xmiEA10 = 2
- xmiEA11 = 3
- xmiEA12 = 4
- xmiRose10 = 5
- xmiRose11 = 6
- xmiRose12 = 7
- xmiMOF13 = 8
- xmiMOF14 = 9
- xmiEA20 = 10
- xmiEA21 = 11
- xmiEA211 = 12
- xmiEA212 = 13
- xmiEA22 = 14
- xmiEA23 = 15
- xmiEA24 = 16
- xmiEA241 = 17
- xmiEA242 = 18
- xmiEcore = 19
- xmiBPMN20 = 20
- xmiXPDL22 = 21
- xmiEA251 = 22
- xmiARCGIS = 23
- xmiNative = 24
- xmiEA2511 = 25
- xmiNativeXEA = 26

Repository Package

The Repository Package contains the high level system objects and the entry point into the model itself, using the Models collection and the other system level collections.

This diagram shows the collections of the Repository interface. Association Target roles correspond to member variable names in the Repository interface. The associated Classes represent the object type used in each collection.



Author Class

An Author object represents a named model author. Authors can be accessed using the Repository Authors collection.

Associated table in repository

t_authors

Author Attributes

Attribute	Remarks
Name	String Notes: Read/Write The Author name.
Notes	String Notes: Read/Write Notes about the author.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Roles	String Notes: Read/Write Roles the author might play in this project.

Author Methods

Method	Remarks
GetLastError ()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update ()	Boolean Notes: Updates the current Author object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Client Class

A Client represents one or more people or organizations related to the project. Clients can be accessed using the Repository Clients collection.

Associated table in repository

t_clients

Client Attributes

Attribute	Remarks
EMail	String Notes: Read/Write The client's email address.
Fax	String Notes: Read/Write The client's fax number.
Mobile	String Notes: Read/Write The client's mobile phone number, if available.
Name	String Notes: Read/Write The client's name.
Notes	String Notes: Read/Write Notes about the client.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through the Dispatch interface.
Organization	String Notes: Read/Write The client's associated organization.
Phone1	String Notes: Read/Write The client's main phone number.

Phone2	String Notes: Read/Write The client's second phone number.
Roles	String Notes: Read/Write Roles this client might play in the project.

Client Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Updates the current Client object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Collection Class

Collection is the main collection Class used by all elements within the Automation Interface. It contains methods to iterate through the collection, refresh the collection and delete an item from the collection.

It is important to realize that when the 'AddNew' function is called, the item is not automatically added to the current collection. The typical steps are:

- Call AddNew to add a new item
- Modify the item as required
- Call Update on the item to save it to the database
- Call Refresh on the collection to include it in the current set

Delete is the same; until Refresh is called, the collection still contains a reference to the deleted item, which should not be called.

Each method can be used to iterate through the collection for languages that support this type of construct.

Collection Attributes

Attribute	Remarks
Count	Short Notes: Read only The number of objects referenced by this list.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.

Collection Methods

Method	Remarks
AddNew(string Name, string Type)	Object Notes: Adds a new item to the current collection. The interface is the same for all collections; you must provide a Name and Type argument. What these arguments are used for depends on the actual collection being accessed. For example, when adding a new element to the Elements collection, the Type string can be either a basic UML element type or a fully qualified element type (stereotype) defined by a profile, such as SysML::Requirement, differentiating it from a standard requirement. Also note that you must call Update() on the returned object to complete the AddNew function. If Update() is not called the object is left in an indeterminate state. When an error occurs an exception will be thrown, including when the user does not have Security permission to modify the specify type. Parameters:

	• Name: String • Type: String (up to 30 characters long)
Delete(short index)	Void Notes: Deletes the item at the selected reference. Parameters: • index: Short
DeleteAt(short index, boolean Refresh)	Void Notes: Deletes the item at the selected index. The second parameter is currently unused. Parameters: • index: Short • Refresh: Boolean
GetAt(short index)	Object Notes: Retrieves the array object using a numerical index. If the index is out of bounds, an error occurs. Parameters: • index: Short
GetByName(string Name)	Object Notes: Gets an item in the current collection by name. Supported for Model, Package, Element, Diagram and element TaggedValue collections. If the collection does not contain any items (or, for the Tagged Value collection, if the collection contains items but the method cannot locate an object with the specified name) the method returns a null value. For other collections, if the method is unable to find an object with the specified name, it raises an exception. Parameters: • Name: String
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Refresh()	Void Notes: Refreshes the collection by re-querying the model and reloading the collection. Should be called after adding a new item or after deleting an item.
Update()	Boolean Notes: Updates the current Collection object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

The AddNew Function

The AddNew() function is used widely across the API to add new objects to a Collection. In all cases you must provide a Name and Type argument, but what these arguments are used for depends on the actual collection being accessed. For example, when adding a new element to the Elements collection, the 'Type' string can be either a basic UML element type or a fully qualified element type (stereotype) defined by a profile, such as SysML::Requirement differentiated from a standard requirement.

AddNew Attribute Arguments

This table provides guidance in specifying the AddNew arguments for each of the object attributes.

Attribute	Arguments
AttributeConstraints	Name - The name of the constraint. Type - The constraint type
Attributes	Name - The name of the attribute. Type - The attribute type.
AttributesEx	Name - The name of the attribute. Type - The attribute type.
AttributeTags	Name - The fully-qualified name, or plain text. Type - The value of the Tagged Value.
Authors	Name - The author name. Type - The author role.
Clients	Name - The client name. Type - The client role.
ConnectorConstraints	Name - The name of the constraint. Type - The constraint type.
ConnectorConveyedItems	Name - The GUID of an element. Type - *Not used.* Note: This does not return an object.
Connectors	Name - The name of the connector. Type - The connector type (for example 'Realization').
ConnectorTags	Name - The fully-qualified name, or plain text. Type - The value of the Tagged Value.
Constraints	Name - The name of the constraint. Type - The constraint type.
ConstraintsEx	Name - The name of the constraint.

	Type - The constraint type.
CustomProperties	You cannot create these.
DataTypes	Name - The datatype name. Type - The datatype type.
DiagramLinks	Name - *Not used*. Type - The style string (such as 'l=200;r=400;t=200;b=600;') (You might prefer to leave the Type empty and use the Functions on this interface for size, colors and so on).
DiagramObjects	Name - This can either be an empty string, or it can specify the initial Left, Right, Top and Bottom values for the new DiagramObject. For example: diagram.DiagramObjects.AddNew("l=200;r=400;t=200;b=600;", "") Note: Top and Bottom values should be specified here as positive numbers, but will be set in the repository as negative values. Type - Unused.
Diagrams	Name - The name of the diagram. Type - This can be either a standard UML metaclass type (such as 'Class' or 'UseCase') or a fully-qualified metatype defined by an MDG Technology (such as 'BPMN2.0::BusinessProcess' or 'SysML1.4::Block').
Efforts	Name - The name of the effort. Type - The effort type.
Elements	Name - The name of the new element. If the repository has an auto-name counter defined for the element type being created, pass an empty string to use the auto-name counter instead. Type - Can be either a standard UML metaclass type (such as 'Class' or 'UseCase') or a fully-qualified metatype defined by an MDG Technology (such as 'BPMN2.0::BusinessProcess' or 'SysML1.4::Block').
Files	Name - The full pathname of the file. Type - The file type (such as 'Local File' or 'Web Address').
Issues	Name - The name of the issue. Type - The problem type, (such as 'Issue' or 'Defect')
MethodPostConditions	Name - The name of the constraint. Type - The constraint type
MethodPreconditions	Name - The name of the constraint. Type - The constraint type.
Methods	Name - The name of the method. Type - The return value of the method.
MethodsEx	Name - The name of the method.

	Type - The return value of the method.
MethodTags	Name - The fully-qualified name, or plain text. Type - The value of the Tagged Value.
Metrics	Name - The name of the metric. Type - The metric type.
Models	Name - The name of the model. Type - Unused.
Packages	Name - The name of the Package. Type - Unused.
Parameters	Name - The parameter name. Type - The parameter type.
ParamTags	Name - The fully-qualified name or plain text. Type - The value of the Tagged Value.
Partitions	Name - The partition name. Type - The partition note.
ProjectIssues	Name - The name of the issue. Type - The issue type (such as 'Request', 'Defect', or 'Release')
ProjectResources	Name - The resource name. Type - The resource role.
ProjectRole	Name - The role name. Type - *Not used.*
PropertyTypes	Name - The tag name. Type - The description (limited to 50 characters).
Requirements	Name - The name of the requirement. Type - The requirement type.
RequirementsEx	Name - The name of the requirement. Type - The requirement type.
Resources	Name - The resource name. Type - The resource role.
Risks	Name - The name of the risk. Type - The risk type.
ScenarioExtension	Name - The extension name. Type - The scenario type

ScenarioStep	Name - The step name. Type - The ScenarioStep type value.
Scenarios	Name - The name of the scenario. Type - The scenario type.
Stereotypes	Name - The stereotype name. Type - The element this applies to. Note: You can only support multiple elements from within a Profile.
Tasks	Name - The task name. Type - The task type.
TemplateBindings	Name - The formal name of the binding. Type - The actual name of the binding or element GUID.
TemplateParameters	Name - The parameter name. Type - The parameter type
Terms	Name - The term name. Type - The term type.
Tests	Name - The name of the test. Type - The test type.
Transitions	Name - The transition name. Type - The transition value.

Datatype Class

A Datatype is a named type that can be associated with attribute or method types. It typically is related to either code engineering or database modeling. Datatypes also indicate which language or database system they relate to. Datatypes can be accessed using the Repository Datatypes collection.

Associated table in repository

t_datatypes

Datatype Attributes

Attribute	Remarks
DatatypeID	Long Notes: Read/Write The instance ID for this datatype within the current model; this is system maintained.
DefaultLen	Long Notes: Read/Write The default length (DDL only).
DefaultPrec	Long Notes: Read/Write The default precision (DDL only).
DefaultScale	Long Notes: Read/Write The default scale (DDL only).
GenericType	String Notes: Read/Write The associated generic type for this data type.
HasLength	String Notes: Read/Write Indicates whether the datatype has a length component.
MaxLen	Long Notes: Read/Write The maximum length (DDL only).
MaxPrec	Long Notes: Read/Write

	The maximum precision (DDL only).
MaxScale	Long Notes: Read/Write The maximum scale (DDL only).
Name	String Notes: Read/Write The datatype name (such as integer). This appears in the related drop-down datatype lists where appropriate.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Product	String Notes: Read/Write The datatype product, such as Java, C++ or Oracle.
Size	Long Notes: Read/Write The datatype size.
Type	String Notes: Read/Write The type can be DDL for database datatypes or Code for language datatypes.
UserDefined	Long Notes: Read/Write Indicates if the datatype is a user defined type or system generated. Datatypes distributed with Enterprise Architect are all system generated. Datatypes created in the 'Datatype' dialog are marked **1** (True).

Datatype Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Updates the current Datatype object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

EventProperties Class

An EventProperties object is passed to BroadcastFunctions to facilitate parameter passing.

EventProperties Attributes

Attribute	Remarks
Count	Long Notes: Read only The number of parameters being passed to this broadcast event.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.

EventProperties Methods

Method	Remarks
Get(object Index)	EventProperty Class Notes: Read only Returns an EventProperty in the list, raising an error if Index is out of range. Parameters: • Index: Variant - can either be a number representing a zero-based index into the array, or a string representing the name of the EventProperty: for example, Props.Get(3) or Props.Get("ObjectID")

EventProperty Class

EventProperty objects are always part of an EventProperties collection, and are passed to Add-In methods responding to broadcast events.

EventProperty Attributes

Attribute	Remarks
Description	String Notes: An explanation of what this property represents.
Name	String Notes: A string distinguishing this property from others in the list.
ObjectType	ObjectType Notes: Distinguishes objects referenced through a Dispatch interface.
Value	Variant Notes: A string, number or object reference representing the property value.

ModelWatcher Class

The ModelWatcher object enables an automation client to track changes in a particular model.

ModelWatcher Attributes

Attribute	Remarks
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.

ModelWatcher Methods

Methods	Remarks
GetReloadItem (object Item)	ReloadType Notes: The object that must be reloaded in order to see all changes is returned through the Item parameter. If there are no changes or the entire model must be reloaded, this value is returned as null (C#) or Nothing (VB). Calling this method clears the records so that the next time it is called the return values refer only to new changes. Returns a value from the ReloadType enumeration that specifies which type of change, if any, has occurred. Parameters: • Item: Object
PeekReloadItem	ReloadType Notes: This method behaves identically to 'GetReloadItem()' but does not clear the change record.

Notes

- After your model has been loaded, you only create the ModelWatcher once; if you reload the model, or load another model, the created ModelWatcher is still valid

Package Class

A Package object corresponds to a Package element in the Enterprise Architect Browser window. Packages can be accessed either through the Repository Models collection (a Model is a special form of Package) or through the Packages collection.

Note that a Package has an Element object as an attribute; this corresponds to an Enterprise Architect Package element in the t_object table and is used to associate additional information (such as scenarios and constraints) with the logical Package.

To set additional information for a Package, reference the Element object directly. Also note that if you add a Package to a diagram, you should add an instance of the element (not the Package itself) to the DiagramObject Class for a diagram.

Associated table in repository

t_package

Package Attributes

Attribute	Remarks
Alias	String Notes: Read only Alias
BatchLoad	Long Notes: Read/Write Flag to indicate that the Package is batch loaded during batch import from controlled Packages. Not currently used.
BatchSave	Long Notes: Read/Write Boolean value to indicate whether the Package is included in the batch XMI export list or not.
CodePath	String Notes: Read/Write The path where associated source code is found. Not currently used.
Connectors	Collection Notes: Read only The collection of connectors.
Created	Date Notes: Read/Write Date the Package was created.

Diagrams	Collection Notes: Read only A collection of diagrams contained in this Package.
Element	Element Notes: Read only The associated element object; use to get/set common information such as Stereotype, Complexity, Alias, Author, Constraints, Tagged Values and Scenarios.
Elements	Collection Notes: Read only A collection of elements that belong to this Package.
Flags	String Notes: Read/Write Extended information about the Package.
IsControlled	Boolean Notes: Read/Write Indicates if the Package has been marked as Controlled.
IsModel	Boolean Notes: Read only Indicates if the Package is a model or a Package.
IsNamespace	Boolean Notes: Read/Write True indicates that 'Package is a Namespace root'. Use 0 and 1 to set False and True.
IsProtected	Boolean Notes: Read/Write Indicates if the Package has been marked as 'Protected'.
IsVersionControlled	Boolean Notes: Read only Indicates whether or not this Package is under Version Control.
LastLoadDate	Date Notes: Read/Write The date XML was last loaded for the Package.
LastSaveDate	Date Notes: Read/Write The date XML was last saved from the Package.
LogXML	Boolean

	Notes: Read/Write Indicates if XMI export information is to be logged.
Modified	Date Notes: Read/Write Date the Package was last modified.
Name	String Notes: Read/Write The name of the Package.
Notes	String Notes: Read/Write Notes about this Package.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Owner	String Notes: Read/Write. The Package owner when using controlled Packages.
PackageGUID	Variant Notes: Read only The global Package ID; valid across models.
PackageID	Long Notes: Read only The local Package ID number. Valid only in this model file.
Packages	Collection Notes: Read only A collection of contained Packages that can be walked through.
ParentID	Long Notes: Read/Write The ID of the Package that is the parent of this one. 0 indicates that this Package is a model (that is, it has no parent).
StereotypeEx	String Notes: Read/Write All the applied stereotypes of the element in a comma-separated list. Reading the value will provide the stereotype name only; assigning the value accepts either fully-qualified or simple names. When setting this attribute, LastError (from the GetLastError method) will be non-empty on error.

TreePos	Long Notes: Read/Write The relative position in the tree compared to other Packages (use to sort Packages).
TypeInfoProperties	Notes: Read only Returns an interface pointer of TypeInfoProperties.
UMLVersion	String Notes: Read/Write The UML version for XMI export purposes.
UseDTD	Boolean Notes: Read/Write Indicates if a DTD is to be used when exporting XMI.
Version	String Notes: Read/Write The version of the Package.
XMLPath	String Notes: Read/Write The path to which the XML is saved when using controlled Packages.

Package Methods

Method	Remarks
ApplyGroupLock (string aGroupName)	Boolean Notes: Applies a group lock to the Package object, for the specified group, on behalf of the current user. User Security applies to the use of this function; if the user does not have permission to apply or release locks on elements, diagrams and Packages, the operation will fail. Returns True if the operation is successful; returns False if the operation is unsuccessful. Use GetLastError() to retrieve error information. Parameters: • aGroupName: String - The name of the security group for which to apply the lock
ApplyGroupLockRecursive (string aGroupName, boolean IncludeElements, boolean IncludeDiagrams, boolean IncludeSubPackages)	Boolean Notes: Applies a group lock to the Package object, object, and all of the Package, diagrams and elements contained within that Package, for the specified group, on behalf of the current user. User Security applies to the use of this function; if the user does not have permission to apply or release locks on elements, diagrams and Packages, the operation will fail. Returns True if the operation is successful; returns False if the operation is unsuccessful. Use 'GetLastError()' to retrieve error information.

	Parameters • aGroupName: String - The name of the security group for which to apply the lock • IncludeElements: Boolean - Recursively apply group lock to child elements • IncludeDiagrams: Boolean - Recursively apply group lock to child diagrams • IncludeSubPackages: Boolean - Recursively apply group lock to child Packages
ApplyUserLock ()	Boolean Notes: Applies a user lock to the Package object for the current user. User Security applies to the use of this function; if the user does not have permission to apply or release locks on elements, diagrams and Packages, the operation will fail. Returns True if the operation is successful; returns False if the operation is unsuccessful. Use 'GetLastError()' to retrieve error information.
ApplyUserLockRecursive (boolean IncludeElements, boolean IncludeDiagrams, boolean IncludeSubPackages)	Boolean Notes: Applies user locks to the Package object, and all of the Packages, diagrams and elements contained within that Package. User Security applies to the use of this function; if the user does not have permission to apply or release locks on elements, diagrams and Packages, the operation will fail. Returns True if the operation is successful; returns False if the operation is unsuccessful. Use GetLastError() to retrieve error information. Parameters • IncludeElements: Boolean - Recursively apply user lock to child elements • IncludeDiagrams: Boolean - Recursively apply user lock to child diagrams • IncludeSubPackages: Boolean - Recursively apply user lock to child Packages
Clone	LDISPATCH Notes: Inserts a copy of the Package into the same parent as the original Package. Returns the newly-created Package.
FindObject (string DottedID)	LPDISPATCH Notes: Returns a Package, element, attribute or operation matching the parameter DottedID. If the DottedID is not found, an error is returned: *Can't find matching object.* Parameters • DottedID: String - Is in the form 'object.object.object' where object is replaced by the name of a Package, element attribute or operation; examples include MyNamespace.Class1, CStudent.m_Name, MathClass.DoubleIt(int)
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
GetTXAlias (string Code, long Flag)	String Notes: Returns the Alias of the element for a given language. Parameters • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Flag: Long

	- 0 = Get the currently-stored translated Alias - 1 = Get the currently-stored translated Alias, and auto translate if the original Alias has changed - 2 = Always fetch the translated Alias from online
GetTXNote (string Code, long Flag)	String Returns the Notes of the element for a given language. Parameters • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Flag: Long - 0 = Get the currently-stored translated Notes - 1 = Get the currently-stored translated Notes, and auto translate if the original Notes have changed - 2 = Always fetch the translated Notes from online
SetTXAlias (string Code, string Translation)	String Notes - Set the translated Alias of the element for a given language. • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Translation: String - The translated Alias
SetTXName (string Code, string Translation)	String Notes - Set the translated name of the element for a given language. • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Translation: String - The translated name
SetTXNote (string Code, string Translation)	String Notes - Set the translated Notes of the element for a given language. • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Translation: String - The translated Notes
GetTXName (string Code, long Flag)	String Notes: Returns the name of the element for a given language. Parameters • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Flag: Long - 0 = Get the currently-stored translated name - 1 = Get the currently-stored translated name, and auto translate if the original name has changed - 2 = Always fetch the translated name from online
ReleaseUserLock ()	Boolean Notes: Releases user locks and group locks from the Package object, and all of the Packages, diagrams and elements contained within that Package. User Security applies to the use of this function; if the user does not have permission to apply or release locks on elements, diagrams and Packages, the operation will fail. Returns True if the operation is successful; returns False if the operation is

	unsuccessful. Use GetLastError() to retrieve error information.
ReleaseUserLockRecursive (boolean IncludeElements, boolean IncludeDiagrams, boolean IncludeSubPackages)	Boolean Notes: Releases user locks from the Package object, and all of the Packages, diagrams and elements contained within that Package. User Security applies to the use of this function; if the user does not have permission to apply or release locks on elements, diagrams and Packages, the operation will fail. Returns True if the operation is successful; returns False if the operation is unsuccessful. Use GetLastError() to retrieve error information. Parameters IncludeElements: Boolean - Recursively release user locks from child elements IncludeDiagrams: Boolean - Recursively release user locks from child diagrams IncludeSubPackages: Boolean - Recursively release user locks from child Packages
SetReadOnly (boolean ReadOnly, boolean IncludeSubPkgs)	Void Notes: Sets a Package Flag to mark a Package as ReadOnly=1. If Project Security is enabled, the user must have 'Configure Packages' permission to use this method. Throws an exception if the operation fails due to the user not having 'Configure Packages' permission; use 'GetLastError()' to retrieve error information. Parameters • ReadOnly: Boolean - Sets or clears the Read Only flag on the Package(s); if: False, any Read Only flag is removed from the Package True, a Read Only flag is applied to the Package • IncludeSubPkgs: Boolean - Indicates whether to set/reset the Read Only flag on just the object Package, or on the object Package and all of the nested sub-Packages that it contains; if: False, only the flag on the object Package is set or cleared True, flags are set (or cleared, according to the ReadOnly parameter) for the object Package plus all of the nested sub-Packages that it contains When working with Version Controlled Packages, the Read Only flag can be applied to Packages whether they are checked-in or checked-out. User Security applies to setting this flag - if you are prevented from editing the Package, you are also prevented from setting the flag.
Update ()	Boolean Notes: Updates the current Package object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information. Note that a Package object also has an element component that must be taken into account; the Package object contains information about the Package attributes such as hierarchy or contents. The element attribute contains information about, for example, Stereotypes, Constraints or Files - all the attributes of a typical element.
VersionControlAdd (string ConfigGuid, string XMLFile, string Comment, boolean KeepCheckedOut)	Void Notes: Places the Package under Version Control, using the specified Version Control Configuration and the specified XMI filename. Throws an exception if the operation fails; use GetLastError() to retrieve error

	information. It is recommended that the Package be saved using Update() before calling VersionControlAdd(), so that any outstanding changes are not lost. Parameters • ConfigGuid: String - Name corresponding to the Unique ID of the Version Control configuration to use • XMLFile: String - Name of the XML file to use for this Package; this filename is relative to the Working Copy folder specified for the Config • Comment: String - Log message that is added to the Version Controlled file's history (where applicable) • KeepCheckedOut: Boolean - Specify True to add to Version Control and keep the Package checked-out
VersionControlCheckin (string Comment)	Void Notes: Perform checkin of the Version Controlled Package (also see VersionControlCheckinEx). Throws an exception if the operation fails; use GetLastError() to retrieve error information. Parameters • Comment: String - Log message that is added to the Version Controlled file's history (where applicable)
VersionControlCheckinEx (string Comment, boolean PreserveCrossPkgRefs)	Void Notes: Perform check-in of the Version Controlled Package. Throws an exception if the operation fails; use GetLastError() to retrieve error information. Parameters • Comment: String - Log message that is added to the Version Controlled file's history (where applicable) • PreserveCrossPkgRefs: Boolean - Flag to indicate whether to preserve or discard pre-existing Cross Package References when checking-in; this parameter overrides the setting in the 'Preferences' dialog, 'XML Specifications' page Unsatisfied cross-Package references are preserved or discarded according to this setting, without prompting the user; see *Learn more*
VersionControlCheckout (string Comment)	Void Notes: Perform checkout of the Version Controlled Package. Throws an exception if the operation fails; use GetLastError() to retrieve error information. Parameters: • Comment: String - Log message that is added to the Version Controlled file's history (where applicable) When working in an environment that uses a Private Model deployment and your model contains a significant number of cross-Package references, it is recommended that you invoke the Repository.ScanXMIAndReconcile() method from time to time, following the re-importation of controlled Packages - for example, after using Package.VersionControlGetLatest() to update a number of Packages, or after performing a number of Package check-outs.
VersionControlGetLatest	Void

(boolean ForceImport)	Notes: Updates the local working copy of the Package file associated with the object Package, before re-importing the Package data from the Package file. Parameters: • ForceImport: Boolean - Used if the Package data in the model is found to be up-to-date with respect to the Version Controlled Package file; if: - False, the Package data that exists in the model is accepted as being up-to-date and no attempt is made to re-import data from the Package file - True, the system re-imports the Package from the Package file regardless See also the menu option 'Version Control | Get Latest'. When working in an environment that uses a Private Model deployment and your model contains a significant number of cross-Package references, it is recommended that you invoke the 'Repository.ScanXMIAndReconcile()' method from time to time, following the re-importation of controlled Packages - for example, after using 'Package.VersionControlGetLatest()' to update a number of Packages, or after performing a number of Package check-outs.

VersionControlGetStatus ()	Long
	Notes: Returns the Version Control status of the Package, as recorded in the current project database.
	Throws an exception if the operation fails; use GetLastError() to retrieve error information.
	Return value maps to this enumerated type:
	enum EnumCheckOutStatus
	{
	csUncontrolled = 0,
	csCheckedIn,
	csCheckedOutToThisUser,
	csReadOnlyVersion,
	csCheckedOutToAnotherUser,
	csOfflineCheckedIn,
	csCheckedOutOfflineByUser,
	csCheckedOutOfflineByOther,
	csDeleted,
	};
	• csUncontrolled - Either unable to communicate with the Version Control provider associated with the Package, or the Package file is unknown to the provider
	• csCheckedIn - The Package is not checked-out to anybody in the current project database
	• csCheckedOutToThisUser - The Package is marked as checked-out to the current user, in the current project database
	• csReadOnlyVersion - The Package is marked as read-only; an earlier revision of the Packagehas been retrieved from Version Control
	• csCheckedOutToAnotherUser - The Package is marked as checked-out in the current project database, by a user other than the current user
	• csOfflineCheckedIn - The Package is not checked-out to anybody in the current project database; however, the Version Control configuration associated with the Package was unable to connect to the VC server
	• csCheckedOutOfflineByUser - The Package was 'checked out' in this database,

	by this user, whilst disconnected from Version Control • csCheckedOutOfflineByOther - The Package was checked out in this project database, by another user, whilst disconnected from Version Control • csDeleted - The Package file has been deleted from Version Control
VersionControlPutLatest (string CheckInComment)	Void Notes: Perform a checkin of the Version Controlled Package, whilst keeping the Package checked-out. Throws an exception if the operation fails; use GetLastError() to retrieve error information. When a Package that was previously marked as Checked Out Offline, is successfully 'Put' (checkedin) to Version Control, that Package's flags are updated to clear the Checked Out Offline indicator. Parameters: • Comment: String - Log message added to the Version Controlled file's history (where applicable)
VersionControlRemove ()	Void Notes: Removes Version Control from the Package. Throws an exception if the operation fails; use 'GetLastError()' to retrieve error information.
VersionControlResynchPkgStatus (boolean ClearSettings)	Notes: Synchronizes the Version Control status of the single object Package recorded in your current model with the Package status reported by your Version Control provider. Parameters: • ClearSettings: Boolean - used if the Package file associated with the specified Package is reported by the Version Control provider as uncontrolled; if ClearSettings is: True, the Version Control settings are cleared from the Package False, the Version Control settings remain unchanged

ProjectIssues Class

A ProjectIssue is a system-level Issue that indicates a problem or risk associated with the system as a whole. ProjectIssues can be accessed using the Repository Issues collection.

Associated table in repository

t_issues

ProjectIssues Attributes

Attribute	Remarks
Category	String Notes: Read/Write The category this issue belongs to.
Date	Date Notes: Read/Write The date the issue item was created.
DateResolved	Date Notes: Read/Write The date the issue was resolved.
Name	String Notes: Read/Write The issue name (that is, the issue itself).
IssueID	Long Notes: Read only The ID of this issue.
Notes	String Notes: Read/Write The associated description of the issue.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Owner	String Notes: Read/Write The owner of the issue.

Priority	String Notes: Read/Write The issue priority - Low, Medium or High.
Resolution	String Notes: Read/Write A description of the resolution.
Resolver	String Notes: Read/Write The name of the person resolving the issue.
Severity	String Notes: Read/Write The issue severity - Low, Medium or High.
Status	String Notes: Read/Write The current status of the issue.

ProjectIssues Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Updates the current Issue object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

ProjectResource Class

A Project Resource is a named person who is available to work on the current project in any capacity. ProjectResources can be accessed using the Repository Resources collection.

Associated table in repository

t_resources

ProjectResource Attributes

Attribute	Remarks
Email	String Notes: The resource's email address.
Fax	String Notes: The resource's fax number.
Mobile	Variant Notes: The resource's mobile number, if available.
Name	String Notes: The name of the resource.
Notes	String Notes: A description of the resource, if appropriate.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Organization	Package Class: String Notes: The organization the resource is associated with.
Phone1	Variant Notes: The resource's main telephone number.
Phone2	Variant Notes: The resource's alternative telephone number.
Roles	String Notes: The roles this resource can play in the current project.

ProjectResource Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Updates the current Resource object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

ProjectRole Class

A ProjectRole object represents a named project role. ProjectRoles can be accessed using the Repository ProjectRole collection.

Associated table in repository

t_projectroles

ProjectRole Attributes

Attribute	Remarks
Description	String Notes: Read/Write The project role item description.
Notes	String Notes: Read/Write Notes about the project role item.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Role	String Notes: Read/Write The project role item name.

ProjectRole Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Updates the current ProjectRole object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

PropertyType Class

A PropertyType object represents a defined property that can be applied to UML elements as a Tagged Value. PropertyTypes can be accessed using the Repository PropertyTypes collection.

Each PropertyType corresponds to one of the predefined Tagged Values for the model.

Associated table in repository

t_propertytypes

PropertyType Attributes

Attribute	Remarks
Description	String Notes: Read/Write A short description of the property.
Detail	String Notes: Read/Write Configuration information for the property.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Tag	String Notes: Read/Write The name of the property (Tag Name).

PropertyType Methods:

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Updates the current PropertyType object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Reference Class

This Interface provides access to the various lookup tables within Enterprise Architect. Use the Repository GetReferenceList() method to get a handle to a list.

GetReferenceList (string Type)

Notes: Uses the list type to get a pointer to a Reference List object.

Parameters:

Type: String - specifies the list type to get; valid list types are:

- Diagram
- Element
- Constraint
- Requirement
- Connector
- Status
- Cardinality
- Effort
- Metric
- Scenario
- Status
- Test
- List:DifficultyType
- List:PriorityType
- List:TestStatusType
- List:ConstStatusType

Example:

```
var statusList as EA.Reference;
statusList = Repository.GetReferenceList("Status");
Session.Output("Status Count: " + statusList.Count);
for (var i=0; i < statusList.Count; i++)
{
        Session.Output("#" + (i+1) + ": " + statusList.GetAt(i));
}
```

Reference Attributes

Attribute	Remarks
Count	Short Notes: A count of items in the list.
ObjectType	ObjectType

	Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Type	String Notes: The list type (for example, DiagramTypes).

Reference Methods

Method	Remarks
GetAt(short Index)	String Notes: Get the item at the specified index. Parameters: • Index: Short - The index of the item to retrieve from the list
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Refresh()	Short Notes: Refresh the current list and return the count of items.

Repository Class

The Repository is the main container of all structures such as models, Packages and elements. You can begin accessing the model iteratively using the Models collection. The Repository also has some convenient methods to directly access the structures without having to locate them in the hierarchy first.

Associated table in repository

<none>

Repository Attributes

Attribute	Remarks
Authors	Collection Notes: Read only This is the system Authors collection containing 0 or more Author objects, each of which can be associated with, for example, elements or diagrams as the item author or owner. Use AddNew(), Delete() and GetAt() to manage Authors.
BatchAppend	Boolean Notes: Read/Write Set this property to True when your automation client has to rapidly insert many elements, operations, attributes and/or operation parameters. Set to False when work is complete. This can result in 10- to 20-fold improvement in adding new elements in bulk.
Clients	Collection Notes: Read only A list of Clients associated with the project. You can modify, delete and add new Client objects using this collection.
ConnectionString	String Notes: Read only The filename/connection string of the current Repository. For a connection string, the DBMS repository type is identified by "DBType=n;" where n is a number corresponding to the DBMS type, as shown: 0 - MYSQL 1 - SQLSVR 3 - ORACLE 4 - POSTGRES 8 - ACCESS2007 9 - FIREBIRD 10 - SQLITE

CurrentSelection	Notes: Read only Provides information on what is selected, and in what location without making any requests to the database.
DataMinerManager	Data Miner object Notes: Returns a pointer to the EA.DataMinerManager interface.
Datatypes	Collection Notes: Read only The Datatypes collection. This contains a list of Datatype objects, each representing a data type definition for either data modeling or code generation purposes.
EAEdition	EAEditionTypes Notes: Read only Returns the current level of core licensed functionality available. This property returns **Corporate** when the edition is Unified or Ultimate. Use 'EAEditionEx' to identify which of these extended editions is available.
EAEditionEx	EAEditionTypes Notes: Read only Returns the current level of extended licensed functionality available (Unified or Ultimate).
EnableCache	Boolean Notes: Read/Write An optimization for pre-loading Package objects when dealing with large sets of automation objects.
EnableUIUpdates	Boolean Notes: Read/Write Set this property to False to improve the performance of changes to the model; for example, bulk addition of elements to a Package. To reveal changes to the user, call 'Repository.RefreshModelView()'.
FlagUpdate	Boolean Notes: Read/Write Instructs Enterprise Architect to update the Repository with the LastUpdate value.
InstanceGUID	String Notes: Read only The identifier string identifying the Enterprise Architect runtime session.
IsSecurityEnabled	Boolean Notes: Read only Indicates whether User Security is enabled for the current repository.
Issues	Collection Notes: Read only The System Issues list. Contains ProjectIssues objects, each detailing a particular

	issue as it relates to the project as a whole.
LastUpdate	String Notes: Read only The identifier string identifying the Enterprise Architect runtime session and the timestamp for when it was set.
LibraryVersion	Long Notes: Read only The build number of the Enterprise Architect runtime.
Models	Collection of type Package Notes: Read only Models are of type Package and belong to a collection of Packages. This is the top level entry point to an Enterprise Architect project file. Each model is a root node in the Browser window and can contain items such as Views and Packages. A model is a special form of a Package; it has a ParentID of 0. By iterating through all models, you can access all the elements within the project hierarchy. You can also use the AddNew() function to create a new model. A model can be deleted, but remember that everything contained in the model is deleted as well.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through the Dispatch interface.
ProjectGUID	String Notes: Read only Returns the unique ID for the project.
ProjectRoles	Collection Notes: Read only The system Roles collection containing 0 or more Role objects, each of which can be associated with, for example, elements or diagrams as the item author or owner. Use AddNew(), Delete() and GetAt() to manage Roles.
PropertyTypes	Collection Notes: Read only Collection of Property Types available to the Repository.
Resources	Collection Notes: Read only Contains available ProjectResource objects to assign to work items within the project. Use the 'Add New()', 'Modify()' and 'Delete()' functions to manage resources.
SearchWindow	Notes: Read only Returns a reference to the Enterprise Architect Search Window.
SecurityUser	Notes: Read only

	Provides information about the currently logged in security user.
Stereotypes	Collection Notes: Read only The Stereotype collection. A list of Stereotype objects that contain information on a stereotype and the elements it can be applied to.
SuppressEADialogs	Boolean Notes: Read/Write Set this property in the EA_OnPostNewElement broadcast event to control whether Enterprise Architect should suppress showing the default 'Properties' dialog to the user when an element is created.
SuppressSecurityDialog	Boolean Notes: Read/Write Suppress the login prompt dialog that appears by default when username and password parameters passed to OpenFile2 are invalid. For use by external automation clients only.
Tasks	Collection Notes: Read only A list of system tasks (to do list). Each entry is a Task Item; you can modify, delete and add new tasks.
Terms	Collection Notes: Read only The Project Glossary Terms. Each Term object is an entry in the Glossary. Add, modify and delete Terms to maintain the Glossary.

Repository Methods

Method	Remarks
ActivateDiagram (long DiagramID)	Notes: Activates an already open diagram (that is, makes it the active tab) in the main Enterprise Architect user interface. Parameters: • DiagramID: Long - the ID of the diagram to make active
ActivatePerspective (string long)	Boolean Notes: Deprecated - no longer in use.
ActivateTab (string Name)	Notes: Activates an open Enterprise Architect tabbed view. Parameters: • Name: String - the name of the view to activate
ActivateTechnology (string TechnologyID)	Notes: Activates an enabled MDG Technology. Parameters:

	• TechnologyID: String - the ID of the Technology to activate, as assigned in the MDG Technology Wizard
ActivateToolbox (string Toolbox, long Options)	Boolean Notes: Activates a Toolbox page in the GUI. The returned value is reserved for future use. Parameters: • Toolbox: String - the name of the Toolbox page to activate • Options: Long - reserved for future use
AddDefinedSearches (string sXML)	Notes: Used to enter a set of defined searches that last in Enterprise Architect for the life of the application; when Enterprise Architect loads again they must be inserted again by your Add-In. Parameters: • sXML: String - the XML of the defined searches; you can get this XML by performing an export of the searches from the 'Manage Searches' dialog in Enterprise Architect
AddDocumentationPath (string Name, string Path, long Type)	Notes: Provides an Add-In with the ability to insert a book path into the Enterprise Architect installation directory, to display Learning Center pages on user-authored subjects (such as use of the Add-In). Parameters: • Name: String - the top-level (root) name for the Learning Center documentation hierarchy for the Add-In (for example, Enterprise Architect) • Path: String - the directory path to the folder to contain the Learning Center documentation structure (for example, C:\Program Files (86)\Sparx Systems\EA\Books • Type: Long - reserved for future use; set to 0
AddPerspective (string Perspective, long Options)	Boolean Notes: Deprecated - no longer in use.
AddPropertiesTab (string TabName, string PropXML)	Notes: Create a Properties tab. Returns a PropertiesTab interface if a tab was created successfully, otherwise NULL. Parameters: • TabName: String - Name of the Properties tab • PropXML: String - An XML string defining the values in the tab **Example XML string.** <?xml version='1.0'?> <properties> <group name='theGroup1'> <property id='1' type='text' default='' readonly='false' > <name>TestText</name> <description>this has id=1</description> </property> <property id='2' type='combobox' default='' readonly='false' > <name>TestCombo</name>

```
      <value>Two</value>
      <description>this has id=2</description>
      <valuelist>
        <item>One</item>
        <item>Two</item>
        <item>Three</item>
      </valuelist>
    </property>
    <property id='3' type='date' default='currentdate' showcheckbox='false'
readonly='false' >
      <name>TestDate</name>
      <value></value>
      <description>this has id=3</description>
    </property>
    <property id='4' type='checkbox' default='true' readonly='false' >
      <name>TestCheckbox</name>
      <description>this has id=4</description>
    </property>
    <property id='5' type='spin' default='1' min='0' max='100' readonly='false' >
      <name>TestSpin</name>
      <value>7</value>
      <description>this has id=5</description>
    </property>
    <property id='6' type='int' default='1' readonly='false' >
      <name>TestInt</name>
      <value>100</value>
      <description>this has id=6</description>
    </property>
    <property id='7' type='double' default='1' readonly='false' >
      <name>TestDouble</name>
      <value>3.333</value>
      <description>this has id=7</description>
    </property>
    <property id='8' type='memo' default='' readonly='false' >
      <name>TestMemo</name>
      <value></value>
      <description>this has id=8</description>
    </property>
  </group>
  <group name='theGroup2'>
    <property id='22' type='text' default='' readonly='false' >
      <name>Test1</name>
      <value></value>
      <description>this has id=22</description>
      <valuelist>
```

	<item></item> </valuelist> </property> </group> </properties>
AddTab (string TabName, string ControlID)	activeX custom control Notes: Adds an ActiveX custom control as a tabbed window. Enterprise Architect creates a control and, if successful, returns its Unknown pointer, which can be used by the caller to manipulate the control. Parameters: • TabName: String - used as the tab caption • ControlID: String - the ProgID of the control; for example, "CS_AddinFramework.UserControl1"
AddWindow (string WindowName, string ControlID)	activeX custom control Notes: Adds an ActiveX custom control as a window to the Add-Ins docked window. Enterprise Architect creates a control and, if successful, returns its Unknown pointer, which can be used by the caller to manipulate the control. Parameters: • WindowName: String - used as the window title • ControlID: String - the ProgID of the control; for example, "CS_AddinFramework.UserControl1"
AdviseConnectorChange (long ConnectorID)	Notes: Provides an Add-In or automation client with the ability to advise the Enterprise Architect user interface that a particular connector has changed and, if it is visible in any open diagram, to reload and refresh that connector for the user. Parameters: • ConnectorID: Long - the ID of the connector
AdviseElementChange (long ObjectID)	Notes: Provides an Add-In or automation client with the ability to advise the Enterprise Architect user interface that a particular element has changed and, if it is visible in any open diagram, to reload and refresh that element for the user. Parameters: • ObjectID: Long - the ID of the element
CallSBPI (string sbpiPrefix, string Method, string packedParameters)	Notes: Returns a JSON string with the result from the external server. Parameters: • sbpiPrefix: String - Prefix value of the external server • Method: String - Name of the function to call on the external server • [Optional] packedParameters: String - For SBPI Integrations this must match the expected parameters for the specified method; for Custom Services this can pass generic data to the SBPI in any format, but it is suggested you use the packing methods to ensure a correct JSON string structure
ChangeLoginUser (string Name, string Password)	Boolean Notes: Sets the currently logged on user to be the one specified by a name and password; this logs the user into the repository when security is enabled. If security is not enabled an exception (Security not enabled) is thrown. Parameters:

	• Name: String - the name of the user • Password: String - the password of the user
ClearAuditLogs (Object StartDateTime, Object EndDateTime)	Boolean Notes: Clears all Audit Logs from the model. If StartDateTime and EndDateTime are not null then only log items that fall into this period are cleared. Returns True for success, False for failure. • This method cannot be undone; it is strongly advised that you call 'SaveAuditLogs' first to backup the logs • This method might fail if the user logged into the model does not have the correct access permission Parameters: • StartDateTime: Variant (DateTime) - the earliest date and time of log entries to clear • EndDateTime: Variant (DateTime) - the latest date and time of log entries to clear
ClearOutput (string Name)	Notes: Removes all the text from a tab in the System Output window. Parameters: • Name: String - the name of the tab to remove text from
CloseAddins ()	Notes: Called by automation controllers to ensure that Add-Ins created in .NET do not linger after all controller references to Enterprise Architect have been cleared.
CloseDiagram (long DiagramID)	Notes: Closes a diagram in the current list of diagrams that Enterprise Architect has open. Parameters: • DiagramID: Long - the ID of the diagram to close
CloseFile ()	Notes: Closes any open file.
CreateDocumentGenerator( )	Document Generator Notes: Returns a pointer to the EA.DocumentGenerator interface.
CreateModel (CreateModelType CreateType, string FilePath, long ParentWnd)	Boolean Notes: Creates a new .eap model file based on the standard Enterprise Architect Base model, or a shortcut .eap based on a provided SQL connection. Returns True when the new file is created, otherwise returns False. Parameters: • CreateType: CreateModelType - Specify whether to make a new copy of the EABase.eap model, or create a .eap file shortcut to a DBMS repository; the latter option requires a dialog to be opened for the user to provide SQL connection details • FilePath: String - Destination for new .eap file • ParentWnd: Long - Window handle to act as the parent for the 'SQL connection' dialog; only required when using cmEAPFromSQLRepository
CreateOutputTab (string Name)	Notes: Creates a tab in the System Output window. Parameters:

	• Name: String - the name of the tab to create
DeletePerspective (string Perspective, long Options)	Boolean Notes: Deprecated - no longer in use.
DeleteTechnology (string ID)	Boolean Notes: Removes a specified MDG Technology resource from the repository. Returns True if the technology is successfully removed from the model. Returns False otherwise. • This applies to technologies imported into pre-7.0 versions of Enterprise Architect (imported technologies), not to technologies referenced in version 7.0 and later (referenced technologies) Parameters: • ID: String - the ID of the technology
EnsureOutputVisible (string Name)	Notes: Checks that a specified tab in the System Output window is visible to the user. The System Output window is made visible if it is hidden. Parameters: • Name: String - the name of the tab to make visible
ExecutePackageBuildScript (long ScriptOptions, string PackageGuid)	Notes: Helps you to run the active Package build script based on your current selection in the Browser window. You can also run a script by passing in the Package GUID. Parameters: • ScriptOptions: Long - the script type; can be any one of these numerical values: 1 = Build 2 = Test 3 = Run 4 = Create Workbench Instance 5 = Debug • PackageGuid: String - the ID of the Package for which to run the script
Exit	Notes: Shuts down Enterprise Architect immediately. Used by .NET programmers where the garbage collector does not immediately release all referenced COM objects.
ExtractImagesFromNote (string Notes, string WriteImagePath, string RelativeImagePath)	String Notes: Writes any Image Manager links to the WriteImagePath directory. Returns a modified notes text, which contains links to the images using the RelativeImagePath parameter. Parameters: • Notes: String - the notes of the selected Package, diagram or element • WriteImagePath: String - the path where the image file links will be stored; this path must exist • RelativeImagePath: String - the path to be inserted into the modified string indicating where the images can be found (for example, "..\images\")
ExtractSBPIParameter (string packedParameters,	Notes: Returns the value of the parameter name as a string.

string name)	Parameters: • packedParameters: String - The JSON string to append the Name/Value to; cannot be empty • name: String - The name of the parameter
GenerateMDGTechnology (string Filename)	Boolean Notes: Generates an MDG Technology file using the settings in the given MTS file. The returned value indicates success or failure. Parameters: • Filename: String - the name and path of the MTS file to use
GetActivePerspective ()	String Notes: Deprecated - no longer in use.
GetAttributeByGuid (string Guid)	Attribute Notes: Returns a pointer to an attribute in the repository, located by its GUID. This is usually found using the AttributeGUID property of an attribute. Parameters: • Guid: String - the GUID of the attribute to locate
GetAttributeByID (long AttributeID)	Attribute Notes: Returns a pointer to an attribute in the repository, located by its ID. This is usually found using the AttributeID property of an attribute. Parameters: • AttributeID: Long - the ID of the attribute to locate
GetConnectorByGuid (string Guid)	Connector Notes: Returns a pointer to a connector in the repository, located by its GUID. This is usually found using the ConnectorGUID property of a connector. Parameters: • Guid: String - the GUID of the connector to locate
GetConnectorByID (long ConnectorID)	Connector Notes: Searches the repository for a connector with a specific ID. Parameters: • ConnectorID: Long - the ID of the connector to locate
GetContextItem (object Item)	ObjectType Notes: Sets a pointer to an item in context within Enterprise Architect. Also returns the corresponding ObjectType. For additional information about ContextItems and the supported ObjectTypes see the 'GetContextItemType' method. Parameters: • Item: Object - the item to point to
GetContextItemType ()	ObjectType Notes: Returns the ObjectType of an item in context within Enterprise Architect. A ContextItem is defined as an item selected anywhere within the Enterprise

	Architect GUI including: • An item selected in the Browser window • An item selected in an open diagram • An item selected in certain dialogs, such as the attribute 'Properties' dialog The supported ObjectTypes can be any one of these values: • otElement • otPackage • otDiagram • otAttribute • otMethod • otConnector
GetContextObject ()	Object Notes: Returns the current context Object.
GetCounts ()	String Notes: Returns a set of counts from a number of tables within the base Enterprise Architect repository. These can be used to determine whether records have been added or deleted from the tables for which information is retrieved.
GetCurrentDiagram ()	Diagram Notes: Returns a selected diagram.
GetCurrentLoginUser (boolean GetGuid)	String Notes: If security is not enabled in the repository, an error is generated. If 'GetGuid' is True, a GUID generated by Enterprise Architect representing the user is returned; otherwise the text as entered in System Users/User Details/Login is returned.
GetDiagramByGuid (string Guid)	Diagram Notes: Returns a pointer to a diagram using the global reference ID (global ID). This is usually found using the diagram GUID property of an element, and stored for later use to open a diagram without using the collection GetAt() function. Parameters: • Guid: String - the GUID of the diagram to locate
GetDiagramByID (long DiagramID)	Diagram Notes: Gets a pointer to a diagram using an absolute reference number (local ID). This is usually found using the DiagramID property of an element, and stored for later use to open a diagram without using the collection GetAt() function. Parameters: • DiagramID: Long - the ID of the diagram to locate
GetElementByGuid (string Guid)	Element Notes: Returns a pointer to an element in the repository, using the element's GUID reference number (global ID). This is usually found using the ElementGUID property of an element, and stored for later use to open an element without using the collection 'GetAt ()' function. Parameters:

	• Guid: String - the GUID of the element to locate
GetElementByID (long ElementID)	Element Notes: Gets a pointer to an element using an absolute reference number (local ID). This is usually found using the ElementID property of an element, and stored for later use to open an element without using the collection GetAt () function. Parameters: • ElementID: Long - the ID of the element to locate
GetElementsByQuery (string QueryName, string SearchTerm)	Collection (of type Element) Notes: Helps you to run a search in Enterprise Architect, returning the result as a collection. For example: GetElementsByQuery('Simple','Class1'), where the results list elements with 'Class1' in the 'Name' and 'Notes' fields. Parameters: • QueryName: String - the name of the search to run, for example 'Simple' • SearchTerm: String - the term to search for
GetElementSet (string IDList, long Options)	Collection (of type Element) Notes: Returns a set of elements as a collection based on a comma-separated list of ElementID values. By default, if no values are provided in the IDList parameter, all objects for the entire project are returned. Parameters • IDList: String - a comma-separated list of ElementID values • Options: Long - modifies default behavior of this method 1. Returns empty collection when empty IDList parameter is given. 2. Use IDList string as an SQL query to populate this collection.
GetFieldFromFormat (string Format, string Text)	String Notes: Converts a field from your preferred format to Enterprise Architect's internal format; returns the field in that format. Parameters: • Format: String - The format to convert the field from; valid formats are: - HTML - Full HTML - RTF - Rich Text Format - TXT - Plain text • Text: String - The field to be converted
GetFormatFromField (string Format, string Text)	String Notes: After accessing a field that contains formatting, use this method to convert it to your preferred format; returns the field in the format specified. Parameters: • Format: String - The format to convert the field to; valid formats are: - HTML - Full HTML - RTF - Rich Text Format - TXT - Plain text • Text: String - The field to be converted
GetFormattedName (string Guid, long FlagInclude,	String Notes: Provides special formatting for the name of the specified object; for

string Separator, long FlagFormat)	example, the fully qualified name of a specific element or feature. Parameters: • Guid: String - The GUID of the object to be formatted • FlagInclude: Long - Items to be included in the formatted name: - fiFeature = &H01 - fiClass = &H02 - fiParents = &H04 - fiPackage = &H08 - fiRootNS = &H10 - fiHiddenNS = &H20 - fiDiagram = &H40 - fiElemAlias = &H80 • Separator: String - The string to use for separating each included item (such as Packages or elements) • FlagFormat: Long - Additional formatting options: - ffReplaceSpaces = &H01 - ffLowercase = &H02 - ffURLEncode = &H04 Example: FormattedName = Repository.GetFormattedName (Element.ElementGUID, fiFeature Or fiClass Or fiParents Or fiPackage Or fiDiagram, "::", 0)
GetGapAnalysisMatrix ()	String Notes: Read Only Returns all Gap Analyses as an XML document.
GetLastError ()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
GetLocalPath (string Type, string Path)	String Notes: Returns the expanded local file path for code generated from an element, with reference to the Type and Path defined in the 'Local Paths' dialog. Parameters: • Type: String - the coding language for the element, such as Java, C or C++ • Path: String - the local path to be expanded; for example: %Desk%\Javacode\Motor.java For example: Repository.GetLocalPath (Java, %Desk%\Javacode\Motor.java) This could return: C:\Users\fbloggs\Desktop\Javacode\Motor.java.
GetMailInterface ()	MailInterface Notes: Returns an instance of the EA.MailInterface; use this interface to automate the process of creating and sending Model Mail messages.
GetMethodByGuid (string Guid)	Method Notes: Returns a pointer to a method in the repository; this is usually found using the MethodGUID property of a method. Parameters:

	• Guid: String - the GUID of the method to look for
GetMethodByID (long MethodID)	Method Notes: Returns a pointer to a method in the repository; this is usually found using the MethodID property of a method. Parameters: • MethodID: Long - the ID of the method to look for
GetPackageByGuid (string Guid)	Package Notes: Returns a pointer to a Package in the repository using the Package's GUID reference number (global ID). This is usually found using the PackageGUID property of the Package. Each Package in the model also has an associated element with the same GUID, so if you have an element with Type="Package" then you can load the Package by calling: GetPackageByGuid(Element.ElementGUID) Parameters: • Guid: String - the GUID of the Package to look for
GetPackageByID (long PackageID)	Package Notes: Get a pointer to a Package using an absolute reference number (local ID). This is usually found using the PackageID property of a Package, and stored for later use to open a Package without using the collection GetAt () function. Parameters: • PackageID: Long - the ID of the Package to locate
GetProjectInterface ()	Project Notes: Returns a pointer to the EA.Project interface (the XML-based automation server for Enterprise Architect). Use this interface to work with Enterprise Architect using XML, and also to access utility functions for loading diagrams, running reports and so on.
GetPropertiesTab (string TabName)	Notes: Finds an existing Properties tab. Returns a PropertiesTab interface if the tab exists, otherwise NULL. Parameters: • TabName: String - The name of the 'Properties' tab.
GetReferenceList (string Type)	Reference Notes: Uses the list type to get a pointer to a Reference List object. Parameters: • Type: String - specifies the list type to get; valid list types are: - Diagram - Element - Constraint - Requirement - Connector - Status - Cardinality - Effort - Metric - Scenario - Status

	- Test - List:DifficultyType - List:PriorityType - List:TestStatusType - List:ConstStatusType
GetRelationshipMatrix ()	String Notes: Returns an XML document (as a string), containing definitions of all Relationship Matrix profiles saved in the current model.
GetTechnologyVersion (string ID)	String Notes: Returns the version of a specified MDG Technology resource. Parameters: • ID: String - the specified technology ID
GetTreeSelectedElements ()	Collection Notes: Returns the set of elements currently selected in the Browser window as a collection.
GetTreeSelectedItem (object SelectedItem)	ObjectType Notes: Gets an object variable and type corresponding to the currently selected item in the tree view. To use this function, create a generic object variable and pass this as the parameter. Depending on the return type, cast it to a more specific type. The object passed back through the parameter can be a Package, element, diagram, attribute or operation object. Parameters: • SelectedItem: Object - the object to get the variable and type for
GetTreeSelectedItemType ()	ObjectType Notes: Returns the type of the object currently selected in the tree. One of: • otDiagram • otElement • otPackage • otAttribute • otMethod
GetTreeSelectedObject ()	Object Notes: The related method GetTreeSelectedItem () has an output parameter that is inaccessible by some scripting languages. As an alternative, this method provides the selected item through the return value.
GetTreeSelectedPackage ()	Package Notes: Returns the Package in which the currently selected tree view object is contained.
HasPerspective (string Perspective)	String Notes: Deprecated - no longer in use.
HideAddinWindow ()	Notes: Hides the docked Add-In window.

ImportPackageBuildScripts (string PackageGuid, string BuildScriptXML)	Notes: Imports build scripts into a Package in Enterprise Architect. Parameters: • PackageGuid: String - the GUID of the Package into which to import the build scripts • BuildScriptXML: String - the build script XML data, which you can export from within Enterprise Architect
ImportRASAsset (string PackageGUID, string Protocol, string ServerName, string Model, string Storage, string RASGUID, string Password, string Version)	Notes: Imports the specified RAS asset. Returns True on success; check GetLastError on failure. Parameters: • PackageGUID: String - the GUID of the Package to import the asset to • Protocol: String - the protocol the server is using • ServerName: String - the name of the RAS server • Model: String - the name of the RAS model to use • Storage: String - the storage name of the RAS asset • RASGUID: String - the GUID of the RAS asset • Password: String - the password to access the RAS asset • Version: String - the version of the RAS asset to import
ImportTechnology (string Technology)	Boolean Notes: Installs a given MDG Technology resource into the repository. Returns True if the technology is successfully loaded into the model. Otherwise returns False. This applies to technologies imported into pre-7.0 versions of Enterprise Architect (imported technologies), not to technologies referenced in version 7.0 and later (referenced technologies). Parameters: • Technology: String - the contents of the technology resource file
InsertSBPIParameter (string packedParameters, string name, string value)	Notes: Returns a JSON string. Parameters: • packedParameters: String - The JSON string to append the Name/Value to; cannot be empty • name: String - The name of the parameter • value: String - The value of the parameter
InvokeConstructPicker (string ElementFilter)	String Notes: Invokes the 'Select <Item>' dialog with filters on the object type and, optionally, stereotype. Returns the ElementID of the selected object, or **0** if no object was selected when the dialog was closed. For example: elementid=Repository.InvokeConstructPicker ("IncludedTypes=Class,Component;StereoType=foo,bar") In this example, the 'Select <item>' dialog will allow the user to select any Class or Component element in the model that has a stereotype of 'foo' or 'bar'. The 'IncludedTypes' and 'StereoType' filters are separated by a semi-colon. Parameters: • ElementFilter: String - specifies which elements or Packages are to be made

	available for selection, based on element types and stereotypes identified by the IncludedTypes and StereoType filters
	- IncludedTypes - (mandatory) comma separated list of element types that can be selected in the dialog; for example:
	Package,Class,Component
	- MultiSelect - (optional) when set to True ("MultiSelect=True;") allows the Construct picker to select multiple elements
	- Selection (optional) - list of comma-separated element GUIDs that will be selected by default
	- GetNext (optional) - returns the next ID in the list of selected elements, or **0** when no more are available; this option will not display a dialog and assumes the first call was made with MultiSelect=True;
	- StereoType - (optional) comma separated list of stereotypes that can be selected in this dialog
	Do not use leading or trailing spaces between element type or stereotype values. Parameter values must be written with the correct case; element type names are also case sensitive.
	Example:
	`val = Repository.InvokeConstructPicker ("IncludedTypes=Class; MultiSelect=True;");`
	`while(val != 0)`
	`{`
	`val = Repository.InvokeConstructPicker("GetNext=True;");`
	`}`

InvokeFileDialog (string FilterString, long Filterindex, long Flags)	String Notes: Opens a standard 'Open File' dialog and returns a string containing the full path to the selected file on success. Returns an empty string if the dialog was canceled. Parameters: • FilterString: String - list of file type filters. • Filterindex: Long - one-based index of the filter to be used by default • Flags: Long - additional bit flags used to initialize the file dialog; see OPENFILENAME structure in MSDN documentation for accepted values
IsTabOpen (string TabName)	String Notes: Checks whether a named Enterprise Architect tabbed view is open and active. This includes open diagram windows or custom controls added using 'Repository.AddTab ()'. Returns: • 2 to indicate that a tab is open and active (top-most) • 1 to indicate that it is open but not top-most, or • 0 to indicate that it is not visible at all Parameters: • TabName: String - the name of the tab to check for; TabName is case sensitive
IsTechnologyEnabled (string ID)	Boolean Notes: Checks whether the specified string matches the ID of an enabled MDG Technology in Enterprise Architect.

	Returns True if the string matches the ID of an enabled Technology. Otherwise returns False. Parameters: ID: String - the technology ID to check for; built-in technology IDs include: • ArcGIS ArcGIS • BABOK BABOK • BIZBOK BIZBOK Guide • BPSim BPSim • BRM Business Rule Model • CMMN Case Management Model & Notation • CODEENG Code Engineering • Database Modeling Database Modeling • DMN1.1 DMN1.1 • EAExtended Core Extensions • ERD Entity Relationship Diagram • GML GML • MYSQLTECH MySqlTech • EAReview Review • SIMF SIMF Technology • SOAML SOAML • SysML1.1 SysML1.1 • SysML1.2 SysML1.2 • SysML1.3 SysML1.3 • SysML1.4 SysML1.5 • UML2 Basic UML2 Technology • SYSENG System Engineering • 262139 MDG Technology Builder • TOGAF TOGAF • UAF UAF • UPDM2 UPDM 2.0 • Win32UI Win 32 User Interface Modeling • ZF Zachman Framework Technically, any combination of technologies integrated with or added to Enterprise Architect - including user-developed technologies - could appear in this list. In practice you would only check for one or two technologies at a time.
IsTechnologyLoaded (string ID)	Boolean Notes: Checks whether a specified technology is loaded into the repository. Returns True if the MDG Technology resource is loaded into the repository. Otherwise returns False. Parameters: • ID: String - the technology ID to check for
LoadAddins ()	Notes: Loads all Add-Ins from a repository when Enterprise Architect is opened from automation.
MarkupNotes (string	String

Notes, string GlossaryType, string replacement)	Notes: Returns a string containing the translation of the term. Parameters • Notes: String - a value to perform a translation markup on • GlossaryType: String - a comma-separated list of glossary types; for example, 'tx-french,tx-global' • replacement: String - the value to replace the TERM when found; "<span class=\"notranslate\">#TERM#,/span>"
OpenDiagram (long DiagramID)	Notes: Provides a method for an automation client or Add-In to open a diagram. The diagram is added to the tabbed list of open diagrams in the main Enterprise Architect view. Parameters: • DiagramID: Long - the ID of the diagram to open
OpenFile (string Filename)	Boolean Notes: This is the main point for opening an Enterprise Architect project file from an automation client, and working with the contained objects. If the required project is a DBMS or Cloud based repository, you will require a valid Enterprise Architect connection string. This can be obtained in one of two ways; both methods require you to first make and open a connection to the model in question with Enterprise Architect: 1) Using the 'Save as Shortcut' menu item, create a shortcut .eap file containing the database connection string; you can call this shortcut file to access the repository. 2) Alternatively, you can right-click on the model's connection entry in the 'Open Project' screen and select 'Edit connection string', this connection string can then be used direct by OpenFile. Parameters: • Filename: String - the filename (or connection string) of the Enterprise Architect project to open
OpenFile2 (string FilePath, string Username, string Password)	Boolean Notes: As for 'OpenFile ()' except this provides for the specification of a password. Parameters: • Filepath: String - the file path of the Enterprise Architect project to open • Username: String - the user login ID • Password: String - the user password
OpenFileInEditor(string FilePath)	Boolean Notes: Displays a document or source code file in the EA editor Parameters: • FilePath: String - the file path of the document or file to display in the editor
OpenFileInEditorAtLine(string FilePath, integer LineNumber)	Boolean Notes: Displays a document or source code file in the EA editor Parameters: • FilePath: String - the file path of the document or file to display in the editor • LineNumber: Integer - the line number to highlight.

RefreshModelView (long PackageID)	Notes: Reloads a Package or the entire model, updating the user interface. Parameters: • PackageID: Long - the ID of the Package to reload: if 0, the entire model is reloaded; if a valid Package ID, only that Package is reloaded
RefreshOpenDiagrams (boolean FullReload)	Notes: Reloads the diagram contents for all open diagrams from the repository. Parameters: • FullReload: Boolean - if False only the contents of element compartments are reloaded; if True the full content of each diagram is reloaded
ReloadDiagram (long DiagramID)	Notes: Reloads a specified diagram. This would commonly be used to refresh a visible diagram after code import/export or other batch process where the diagram requires complete refreshing. Calling this method within a call to *EA_OnNotifyContextItemModified* is not supported Parameters: • DiagramID: Long - the ID of the diagram to be reloaded
ReloadPackage (long PackageID)	Notes: Reloads a Package and its open child diagrams. Parameters: PackageID: Long - The ID of the Package to reload; if a valid Package ID, only that Package is reloaded.
RemoveOutputTab (string Name)	Notes: Removes a specified tab from the System Output window. Parameters: • Name: String - the name of the tab to be removed
RemoveWindow (string WindowName)	Boolean Notes: Removes an Add-In window that matches the specified WindowName. Parameters: • WindowName: String - the name of the window to remove
RepositoryType ()	String Notes: Returns the currently open database/repository type. Can return one of these values: • JET (.EAP file, MS Access 97 to 2013 format) • FIREBIRD • ACCESS2007 (.accdb file, MS Access 2007+ format) • ASA (Sybase SQL Anywhere) • SQLSVR (Microsoft SQL Server) • MYSQL (MySQL) • ORACLE (Oracle) • POSTGRES (PostgreSQL)
RunModelSearch (string sQueryName, string sSearchTerm, string sSearchOptions, string sSearchData)	Notes: Runs a search, displaying the results in Enterprise Architect's Model Search window. Parameters: • sQueryName: String - the name of the search to run, for example Simple

	• sSearchTerm: String - the term to search for • sSearchOptions: String - currently not being used • sSearchData: String - a list of results in the form of XML, which is appended onto the result list in Enterprise Architect - see the *XML Format* topic; this parameter is not mandatory so pass in an empty string to run the search as per normal
SaveAllDiagrams ()	Notes: Saves all open diagrams.
SaveAuditLogs (string FilePath, object StartDateTime, object EndDateTime)	Boolean Notes: Saves the Audit Logs contained within a model to a specified file. If 'StartDateTime' and 'EndDateTime' are not null then only log items that fall into this period are saved. Returns True for success, False for failure. • This might fail if the user logged into the model does not have the correct access permission Parameters: • FilePath: String - the file to save the Audit Logs to • StartDateTime: Variant (DateTime) - the earliest date and time of log entries to save • EndDateTime; Variant (DateTime) - the latest date and time of log entries to save
SaveDiagram (long DiagramID)	Notes: Saves an open diagram; assumes the diagram is open in the main user interface Tab list. Parameters: • DiagramID: Long - the ID of the diagram to save
SaveDiagramAsUMLProfile (string DiagramGUID, string Filename)	Boolean Notes: Saves a given diagram as a UML Profile, using the settings from the previous time that the specific diagram was saved manually. The returned value indicates success or failure. Parameters: • DiagramGUID: String - the GUID of the Profile diagram to save • Filename: String - the name and path of the file to create; if left blank, the method will use the filename from the previous time the specified diagram was saved
SavePackageAsUMLProfile (string PackageGUID, string Filename)	Boolean Notes: Saves a given Package as a UML Profile, using the settings from the previous time that the specific Package was saved manually. The returned value indicates success or failure. Parameters: • PackageGUID: String - the GUID of the Profile Package to save • Filename: String - the name and path of the file to create; if left blank, the method will use the filename from the previous time the specified Package was saved
ScanXMIAndReconcile ()	Notes: Scans the Package XMI files associated with each of the project's controlled Packages and restores any diagram objects or cross-references that are detected as

	missing from the project. This function is useful in team environments where each user maintains their own private copy of the model database (that is, multiple private EAP files) and model updates are propagated through the use of controlled Packages; it provides no benefit when the model is hosted in a single shared database that is accessed by all team members. Each controlled Package is compared with its associated XMI file and, if the cross-reference information in the model does not match the XMI, Enterprise Architect updates the model with the information from the XMI and records the update in the System Output window. You can roll back such updates by right-clicking on the entry in the System Output window and selecting the 'Rollback Update' option (or 'Rollback Selected Updates' if multiple entries are selected). Closing the model clears the entries in the System Output window; an entry in this window is also cleared as and when you roll-back the update for it. This functionality is invoked automatically as part of the 'Get All Latest' operation. When working in an environment that uses a Private Model deployment and your model contains a significant number of cross-Package references, it is recommended that you invoke this function from time to time, following the re-importation of controlled Packages - for example, after using 'Get Latest' to update a number of Packages, or after performing a number of Package check-outs. As a general rule, avoid running this function while you have uncommitted changes in your model. Generally, you: • Check-out a number of Packages • Invoke 'ScanXMIAndReconcile' • Make your modifications • Commit any outstanding changes before you check-out more Packages and run 'ScanXMIAndReconcile' again
ShowAddinWindow (string TabName)	Boolean Notes: Shows the docked Add-In window on the specified page. Returns True if a tab of the specified name is now displayed. Parameters • TabName: String - specifies the tab
ShowDynamicHelp (string Topic)	Notes: Shows a Help topic as a view. Parameters: • Topic: String - specifies the Help topic
ShowInProjectView (object Item)	Notes: Selects a specified object in the Browser window. Accepted object types are Package, Element, Diagram, Attribute, and Method; an exception is thrown if the object is of an invalid type. Parameters: • Item: Object - the object to highlight
ShowWindow (long Show)	Notes: Shows or hides the Enterprise Architect User Interface. Parameters: • Show: Long
SQLQuery (string SQL)	String

	Notes: Enables execution of a SQL select statement against the current repository. Returns an XML formatted string value of the resulting record set. Parameters: • SQL: String - contains the SQL Select statement
SynchProfile (string Profile, string Stereotype)	Boolean Notes: Synchronizes Tagged Values and constraints of a UML Profile item using the 'Synch Profiled Elements' dialog. Parameters: • Profile: String - the name of the profile that contains the stereotype • Stereotype: String - the name of the profile stereotype for which the default tags and constraints are to be synchronized
VCRPS	Type **VersionControlResynchPkgStatuses (boolean ClearSettings)** Notes: Synchronizes the Version Control status of each Version Controlled Package within the current model with the status reported by your Version Control provider. Parameters: • ClearSettings: Boolean - if True, clear the Version Control settings from Packages that are reported by the Version Control provider as uncontrolled - if False, leave the Version Control settings unchanged for Packages reported as uncontrolled
WriteOutput (string Name, string Output, long ID)	Notes: Writes text to a specified tab in the System Output window, and associates the text with an ID. Parameters: • Name: String - specifies the tab on which to display the text • Output: String - specifies the text to display • ID: Long - specifies a numeric ID value to associate with this output item for further handling by Add-Ins; can be set to **0** if no handling is required

SecurityUser Class

A SecurityUser object represents a named security user.

Associated table in repository

None.

SecurityUser Attributes

Attribute	Remarks
Department	String Notes: Read only Returns the current user's department.
FirstName	String Notes: Read only Returns the current user's first name.
FullName	String Notes: Read only Returns the current user's full name.
Login	String Notes: Read only Returns the current user's login name.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Surname	String Notes: Read only Returns the current user's surname.

SecurityUser Methods

Method	Remarks
IsMemberOf (string GroupId)	Boolean Returns True if the user is part of the specified security group.

	Parameter: • GroupId: String - Name of the security group to check.

Stereotype Class

The Stereotype element corresponds to a UML stereotype, which is an extension mechanism for varying the behavior and type of a model element. Use the Repository Stereotypes collection to add new elements and delete existing ones.

Associated table in repository

t_stereotypes

Stereotype Attributes

Attribute	Description
AppliesTo	String Notes: Read/Write A reference to the stereotype Base Class; that is, which element it applies to.
MetafileLoadPath	String Notes: Read/Write The path to an associated metafile. The Automation Interface does not yet support loading metafiles. To do this you must use the 'Stereotype' tab of the 'UML Types' dialog in Enterprise Architect.
Notes	String Notes: Read/Write. Notes about the stereotype.
Name	String Notes: Read/Write The stereotype name, which appears in the Stereotype drop list for elements that match the AppliesTo attribute.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
StereotypeGUID	String Notes: Read/Write A unique identifier for stereotype, generally set and maintained by Enterprise Architect.
Style	String Notes: Read/Write An additional style specifier for the stereotype.
VisualType	String

	Notes: Read/Write Indicates an inbuilt visual style associated with a stereotype. Not currently implemented.

Stereotype Methods

Method	Description
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Updates the current stereotype object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Task Class

A Task is an entry in the System Task list. Tasks can be accessed using the Repository Tasks collection.

Associated table in repository

t_tasks

Task Attributes

Attribute	Remarks
ActualTime	Long Notes: Read/Write The time already expended on the task, in hours, days or other units.
AssignedTo	String Notes: Read/Write The person this task is assigned to; that is, the responsible resource.
EndDate	Date Notes: Read/Write The date the task is scheduled to finish.
History	String Notes: Read/Write A memo field to hold, for example, task history or notes.
Name	Variant Notes: Read/Write The task name.
Notes	Variant Notes: Read/Write A description of the task.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Owner	String Notes: Read/Write The task owner.
Percent	Long

	Notes: Read/Write The percentage completion of the task.
Phase	String Notes: Read/Write The phase of the project the task relates to.
Priority	String Notes: Read/Write The priority of this task.
StartDate	Date Notes: Read/Write The date the task is to start.
Status	Variant Notes: Read/Write The current status of the task.
TaskID	Long Notes: Read only The local ID of the task.
TotalTime	Long Notes: Read/Write The total expected time the task might run, in hours, days or some other unit.
Type	String Notes: Read/Write Sets or returns a string representing the type.

Task Methods

Method	Type
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Updates the current Task object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Term Class

A Term object represents one entry in the system glossary. Terms can be accessed using the Repository Terms collection.

Associated table in repository

t_glossary

Term Attributes

Attribute	Remarks
Meaning	String Notes: Read/Write The description of the term; its meaning.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Term	String Notes: Read/Write The glossary item name.
TermID	Long Notes: Read only A local ID number to identify the term in the model.
Type	String Notes: Read/Write The type this term applies to (for example, business or technical).

Term Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Refresh	Void Notes: Forces Enterprise Architect to reload the Glossary terms from the database. If an element is selected, it will have to be re-selected before the 'Note' fields and

	windows reflect the updated Glossary terms.
Update()	Boolean Notes: Updates the current Term object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Properties Tab Package

The Properties Tab Package contains:

- A function to retrieve a pointer to the interface
- Functions to create or find a Properties tab
- Utility functions for modifying Properties values

You can get a pointer to this interface using the methods Repository.AddPropertiesTab and Repository.GetPropertiesTab.

PropertiesTab Class

PropertiesTab Attributes

Attribute	Remarks

PropertiesTab Methods

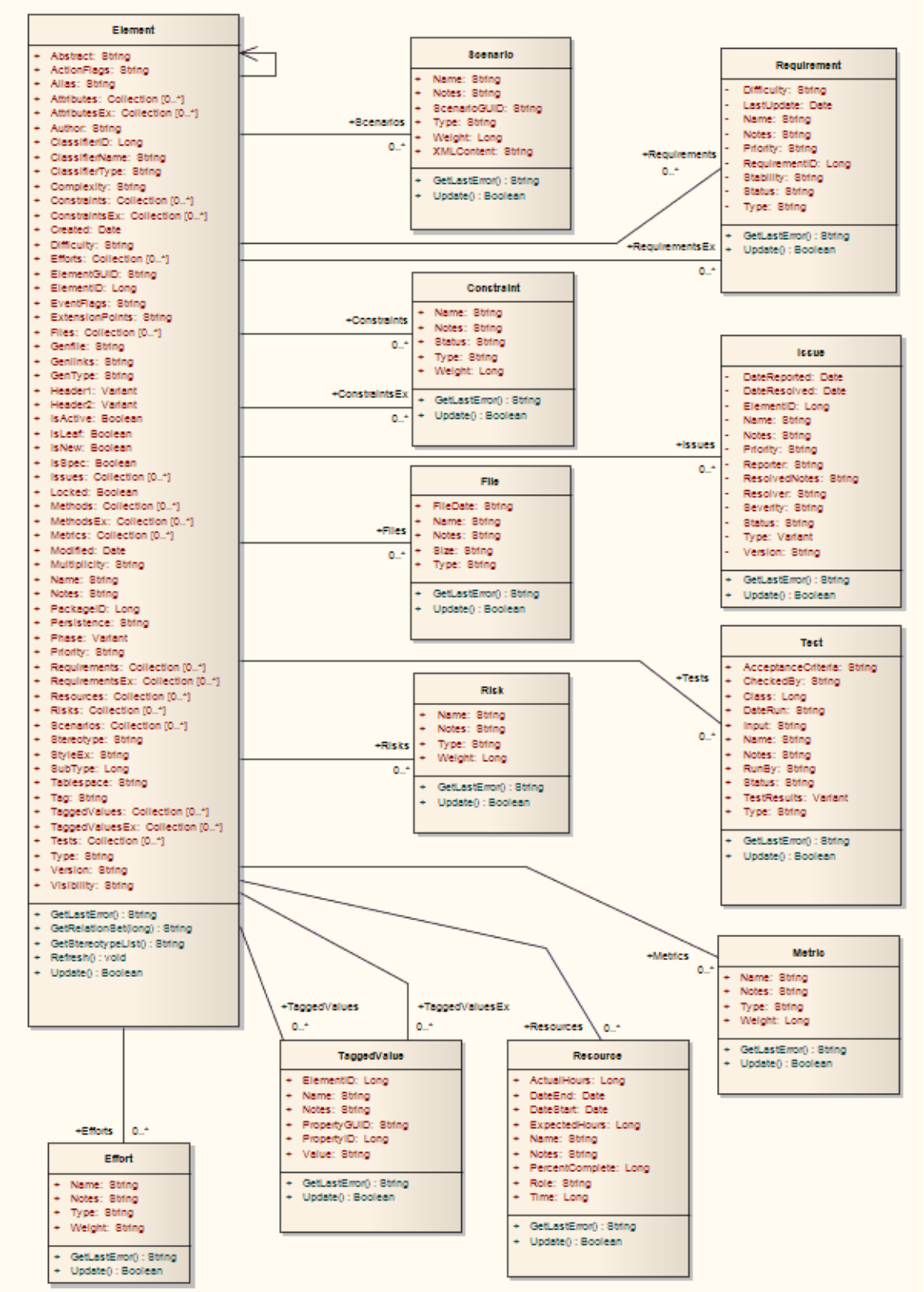
Method	Remarks
AddPropertiesTab (string TabName, string PropXML)	Adds a Properties tab. Returns TRUE if the tab was added. Parameters: • TabName: String - The name of the Properties tab • PropXML: String - An XML string defining the values in the tab
GetLastError ()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
GetPropertiesTab (string TabName)	Notes: Locates a Properties tab. Returns TRUE if the tab is found. Parameters: • TabName: String - The name of the Properties tab
GetPropertiesXML ()	Notes: Returns the XML string of the properties.
GetProperty (long PropID)	Notes: Returns a string of the Property value. Parameters: • PropID: long - The ID value of the property
RemovePropertiesTab ()	Notes: Removes a Properties tab. Returns TRUE if the tab is removed.
SetPropertiesXML (string PropXML)	Notes: Sets the Properties values in the tab. Returns TRUE if the properties were set successfully. Parameters: • PropXML: String - An XML string defining the values in the tab
SetProperty (long PropID, string Value)	Notes: Returns TRUE if the value was set successfully. Parameters: • PropID: long - The ID value of the property to set • Value: String - The value to set the property to

Element Package

The Element Package contains information about an element and its associated extended properties such as testing and project management information. An element is the basic item in an Enterprise Architect model. Classes, Use Cases and Components are all different types of UML element.

This diagram illustrates the relationships between an element and its associated extended information. The related information is accessed through the collections owned by the element (for example, Scenarios and Tests). It also includes a full description of the element object (the basic model structural unit).

Example


Element
+ Abstract: String
+ ActionFlags: String
+ Alias: String
+ Attributes: Collection [0..*]
+ AttributesEx: Collection [0..*]
+ Author: String
+ ClassifierID: Long
+ ClassifierName: String
+ ClassifierType: String
+ Complexity: String
+ Constraints: Collection [0..*]
+ ConstraintsEx: Collection [0..*]
+ Created: Date
+ Difficulty: String
+ Efforts: Collection [0..*]
+ ElementGUID: String
+ ElementID: Long
+ EventFlags: String
+ ExtensionPoints: String
+ Files: Collection [0..*]
+ Genfile: String
+ Genlinks: String
+ GenType: String
+ Header1: Variant
+ Header2: Variant
+ IsActive: Boolean
+ IsLeaf: Boolean
+ IsNew: Boolean
+ IsSpec: Boolean
+ Issues: Collection [0..*]
+ Locked: Boolean
+ Methods: Collection [0..*]
+ MethodsEx: Collection [0..*]
+ Metrics: Collection [0..*]
+ Modified: Date
+ Multiplicity: String
+ Name: String
+ Notes: String
+ PackageID: Long
+ Persistence: String
+ Phase: Variant
+ Priority: String
+ Requirements: Collection [0..*]
+ RequirementsEx: Collection [0..*]
+ Resources: Collection [0..*]
+ Risks: Collection [0..*]
+ Scenarios: Collection [0..*]
+ Stereotype: String
+ StyleEx: String
+ SubType: Long
+ Tablespace: String
+ Tag: String
+ TaggedValues: Collection [0..*]
+ TaggedValuesEx: Collection [0..*]
+ Tests: Collection [0..*]
+ Type: String
+ Version: String
+ Visibility: String
+ GetLastError() : String
+ GetRelationSet(long) : String
+ GetStereotypeList() : String
+ Refresh() : void
+ Update() : Boolean
+Scenarios
0..*
Scenario
+ Name: String
+ Notes: String
+ ScenarioGUID: String
+ Type: String
+ Weight: Long
+ XMLContent: String
+ GetLastError() : String
+ Update() : Boolean
+Requirements
0..*
Requirement
- Difficulty: String
- LastUpdate: Date
- Name: String
- Notes: String
- Priority: String
- RequirementID: Long
- Stability: String
- Status: String
- Type: String
+ GetLastError() : String
+ Update() : Boolean
+RequirementsEx
0..*
+Constraints
0..*
Constraint
+ Name: String
+ Notes: String
+ Status: String
+ Type: String
+ Weight: Long
+ GetLastError() : String
+ Update() : Boolean
+ConstraintsEx
0..*
+Issues
0..*
Issue
- DateReported: Date
- DateResolved: Date
- ElementID: Long
- Name: String
- Notes: String
- Priority: String
- Reporter: String
- ResolvedNotes: String
- Resolver: String
- Severity: String
- Status: String
- Type: Variant
- Version: String
+ GetLastError() : String
+ Update() : Boolean
+Files
0..*
File
+ FileDate: String
+ Name: String
+ Notes: String
+ Size: String
+ Type: String
+ GetLastError() : String
+ Update() : Boolean
+Tests
0..*
Test
+ AcceptanceCriteria: String
+ CheckedBy: String
+ Class: Long
+ DateRun: String
+ Input: String
+ Name: String
+ Notes: String
+ RunBy: String
+ Status: String
+ TestResults: Variant
+ Type: String
+ GetLastError() : String
+ Update() : Boolean
+Risks
0..*
Risk
+ Name: String
+ Notes: String
+ Type: String
+ Weight: Long
+ GetLastError() : String
+ Update() : Boolean
+Metrics
0..*
Metric
+ Name: String
+ Notes: String
+ Type: String
+ Weight: Long
+ GetLastError() : String
+ Update() : Boolean
+TaggedValues
0..*
+TaggedValuesEx
0..*
TaggedValue
+ ElementID: Long
+ Name: String
+ Notes: String
+ PropertyGUID: String
+ PropertyID: Long
+ Value: String
+ GetLastError() : String
+ Update() : Boolean
+Resources
0..*
Resource
+ ActualHours: Long
+ DateEnd: Date
+ DateStart: Date
+ ExpectedHours: Long
+ Name: String
+ Notes: String
+ PercentComplete: Long
+ Role: String
+ Time: Long
+ GetLastError() : String
+ Update() : Boolean
+Efforts
0..*
Effort
+ Name: String
+ Notes: String
+ Type: String
+ Weight: String
+ GetLastError() : String
+ Update() : Boolean

Constraint Class

A Constraint is a condition imposed on an element. Constraints are accessed through the Element Constraints collection.

Associated table in repository

t_objectconstraints

Constraint Attributes

Attribute	Remarks
Name	String Notes: Read/Write The name of the constraint (that is, the constraint).
Notes	String Notes: Read/Write Notes about the constraint.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
ParentID	Long Notes: Read only The ElementID of the element to which this constraint applies.
Status	String Notes: Read/Write The current status of the constraint.
Type	String Notes: Read/Write The constraint type.
Weight	Long Notes: Read/Write A weighting factor.

Constraint Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Update the current Constraint object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Effort Class

An Effort is a named item with a weighting that can be associated with an element for purposes of building metrics about the model. Efforts are accessed through the Element Efforts collection.

Associated table in repository

t_objecteffort

Effort Attributes

Attribute	Remarks
Name	String Notes: Read/Write The name of the effort.
Notes	String Notes: Read/Write Notes about the effort.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Type	String Notes: Read/Write The effort type.
Weight	Long Notes: Read/Write A weighting factor.
Weight2	Float Notes: Read/Write A weighting factor.

Effort Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in

	relation to this object.
Update()	Boolean Notes: Update the current Effort object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Element Class

An Element is the main modeling unit, corresponding to (for example) a Class, Use Case, Node or Component. You create new elements by adding to the Package Elements collection. Once you have created an element, you can add it to the DiagramObject Class of a diagram to include it in the diagram.

Elements have a collection of connectors. Each entry in this collection indicates a relationship to another element.

There are also some extended collections for managing addition information about the element, including properties such as Tagged Values, Issues, Constraints and Requirements.

Associated table in repository

t_object

Element Attributes

Attribute	Remarks
Abstract	String Notes: Read/Write Indicates if the element is Abstract (1) or Concrete (0).
ActionFlags	String Notes: Read/Write A structure to hold flags concerned with Action semantics.
Alias	String Notes: Read/Write An optional alias for this element.
AssociationClassConnector ID	Long Notes: Read only If the element is an AssociationClass, AssociationClassConnectorID contains the Connector ID of the respective Association connector.
Attributes	Collection Notes: Read only A collection of attribute objects for the current element; use the AddNew and Delete functions to manage attributes.
AttributesEx	Collection Notes: Read only A collection of attribute objects belonging to the current element and its parent elements.
Author	String Notes: Read/Write

	The element author.
BaseClasses	Collection Notes: Read only A list of Base Classes for this element, presented as a collection for convenience.
ClassfierID	Long Notes: Deprecated See **ClassifierID**
ClassifierID	Long Notes: Read/Write The ElementID of a Classifier associated with this element; that is, the base type. Only valid for instance type elements (such as Object or Sequence).
ClassifierName	String Notes: Read/Write Name of associated Classifier (if any).
ClassifierType	String Notes: Read only Type of associated Classifier.
Complexity	String Notes: Read/Write A complexity value indicating how complex the element is; used for metric reporting and estimation. Valid values are: 1 for Easy, 2 for Medium, 3 for Difficult.
CompositeDiagram	Diagram Notes: Read only If the element is Composite, returns its associated diagram; otherwise returns null.
Connectors	Collection Notes: Read only Returns a collection containing the connectors to other elements.
Constraints	Collection Notes: Read only A collection of Constraint objects.
ConstraintsEx	Collection Notes: Read only Collection of Constraint objects belonging to the current element and its parent elements.
Created	Date Notes: Read/Write

	The date the element was created.
CustomProperties	Collection Notes: Read only List of advanced properties for an element. The collection of advanced properties differs depending on element type; for example, an Action and an Activity have different advanced properties. Currently only editable from the user interface.
Diagrams	Collection Notes: Read only Returns a collection of sub-diagrams (child diagrams) attached to this element as seen in the tree view.
Difficulty	String Notes: Read/Write A difficulty level associated with this element for estimation/metrics; only useable for Requirement, Change and Issue element types, otherwise ignored. Valid values are: Low, Medium, High.
Efforts	Collection Notes: Read only A collection of Effort objects.
ElementGUID	String Notes: Read only A globally unique ID for this element; that is, unique across all model files.
ElementID	Long Notes: Read only The local ID of the element; valid for this file only.
Elements	Collection Notes: Read only Returns a collection of child elements (sub-elements) attached to this element as seen in the tree view.
EmbeddedElements	Collection Notes: Read only A list of elements that are embedded into this element, such as Ports, Parts, Pins and Parameter Sets.
EventFlags	String Notes: Read/Write A structure to hold a variety of flags to do with signals or events.
ExtensionPoints	String Notes: Read/Write

	Optional extension points for a Use Case as a comma-separated list.
Files	Collection Notes: Read only A collection of File objects.
FQName	String Notes: Read only The fully-qualified name of the element, consisting of a dot-separated list of names including all parent elements and Packages up to the first namespace root that is encountered.
FQStereotype	String Notes: Read only The fully-qualified stereotype name in the format "Profile::Stereotype". One or more fully-qualified stereotype names can be assigned to StereotypeEx.
GenFile	String Notes: Read/Write The file associated with this element for code generation and synchronization purposes; can include macro expansion tags for local conversion to full path.
Genlinks	String Notes: Read/Write Links to other Classes discovered at code reversing time; Parents and Implements connectors only.
GenType	String Notes: Read/Write The code generation type; for example, Java, C++, C#, VBNet, Visual Basic, Delphi.
Header1	Variant Notes: Read/Write A user defined string for inclusion as header in the source files generated.
Header2	Variant Notes: Read/Write Same as for Header1, but used in the CPP source file.
IsActive	Boolean Notes: Read/Write Boolean value indicating whether the element is active or not. 1 = True, 0 = False.
IsComposite	Boolean Notes: Read/Write Indicates whether the element is composite or not. 1 = True, 0 = False.

IsLeaf	Boolean Notes: Read/Write Indicates whether or not the element is a leaf node (and therefore cannot be a parent for any other elements). 1 = True, 0 = False.
IsNew	Boolean Notes: Read/Write Boolean value indicating whether the element is new or not. 1 = True, 0 = False.
IsRoot	Boolean Notes: Read/Write Indicates whether or not the element is a root node (and therefore cannot be descended from another element). 1 = True, 0 = False.
IsSpec	Boolean Notes: Read/Write; Note that this attribute is no longer used in UML 2.0 and later releases, and is provided only to support models maintained in releases of UML prior to 2.0. Boolean value indicating whether the element is a specification or not. 1 = True, 0 = False.
Issues	Collection Notes: Read only Collection of Issue objects.
Locked	Boolean Notes: Read/Write Indicates if the element has been locked against further change.
MetaType	String Notes: Read only The element's domain-specific meta type, as defined by an applied stereotype from an MDG Technology.
Methods	Collection Notes: Read only Collection of Method objects for current element.
MethodsEx	Collection Notes: Read only Collection of Method objects belonging to the current element and its parent elements.
Metrics	Collection Notes: Read only

	Collection of Metric elements for current element.
MiscData	String Notes: Read only This low-level property provides information about the contents of the PData x fields. These database fields are not documented, and developers must gain understanding of these fields through their own endeavors to use this property. MiscData is zero based, therefore: • MiscData(0) corresponds to PData1 • MiscData(1) to PData2, and so on
Modified	Date Notes: Read/Write The date the element was last modified.
Multiplicity	String Notes: Read/Write Multiplicity value for this element.
Name	String Notes: Read/Write The element name; should be unique within the current Package.
Notes	String Notes: Read/Write Further descriptive text about the element.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
PackageID	Long Notes: Read/Write A local ID for the Package containing this element.
ParentID	Long Notes: Read/Write If this element is a child of another, used to set or retrieve the ElementID of the other element; if not, returns 0.
Partitions	Collection Notes: Read only List of logical partitions into which an element can be divided. Only valid for elements that support partitions, such as Activities and States.
Persistence	String Notes: Read/Write

	The persistence associated with this element; can be Persistent or Transient.
Phase	String Notes: Read/Write The phase this element is scheduled to be constructed in; any string value.
Priority	String Notes: Read/Write The priority of this element as compared to other project elements; only applies to Requirement, Change and Issue types, otherwise ignored. Valid values are: Low, Medium and High.
Properties	Properties Notes: Returns a list of specialized properties that apply to the element that might not be available using the automation model. The properties are purposely undocumented because of their obscure nature and because they are subject to change as progressive enhancements are made to them.
PropertyType	Long Notes: Read/Write The ElementID of a Type associated with this element; only valid for Port and Part elements.
PropertyTypeName	String Notes: Read The name of a Type associated with this element; only valid for Port and Part elements.
Realizes	Collection Notes: Read only List of Interfaces realized by this element for convenience.
Requirements	Collection Notes: Read only Collection of Requirement objects.
RequirementsEx	Collection Notes: Read only Collection of Requirement objects belonging to the current element and its parent elements.
Resources	Collection Notes: Read only Collection of Resource objects for current element.
Risks	Collection Notes: Read only Collection of Risk objects.

RunState	String Notes: Read/Write The object's runstate list as a string. The string consists of a set of statements in the form: string = '@VAR;Variable=<string>;Value=<string>;Op=<string>;@ENDVAR;' Where: Op = ['=','>','<','>=','<=', '!=','<>'] For example: A set of run states can be created by looping through a set of attributes and forming a concatenated string: eRunState = eRunState + "@VAR;Variable="+ attrib.name + ";Value=" + attrib.value +";Op==;@ENDVAR;";
Scenarios	Collection Notes: Read only Collection of Scenario objects for current element.
StateTransitions	Collection Notes: Read only List of State Transitions that an element can support; applies in particular to Timing elements.
Status	String Notes: Read/Write Sets or gets the status, such as Proposed or Approved.
Stereotype	String Notes: Read/Write The primary element stereotype; the first of the list of stereotypes you can access using the 'StereotypeEx' attribute. When setting this attribute, LastError (for the GetLastError method) will be non-empty if an error occurs.
StereotypeEx	String Notes: Read/Write All the applied stereotypes of the element in a comma-separated list. Reading the value will provide the stereotype name only; assigning the value accepts either fully-qualified or simple names. When setting this attribute, LastError (for the GetLastError method) will be non-empty if an error occurs.
StyleEx	String Notes: Read/Write Advanced style settings; reserved for the use of Sparx Systems.
Subtype	Long

	Notes: Read/Write A numeric subtype that qualifies the Type of the main element • For Event: 0 = Receiver, 1 = Sender • For Class: 1 = Parameterised, 2 = Instantiated, 3 = Both, 0 = Neither, 17 = Association Class If 17, because an Association Class has been created through the user interface, MiscData(3) contains the ID of the related Association; as MiscData is read-only, you cannot create an Association Class through the Automation Interface. • For Note: 1 = Note linked to connector, 2 = Constraint linked to connector • For StateNode: 100 = ActivityIntitial, 101 = ActivityFinal • For Activity: 0 = Activity, 8 = composite Activity (also set to 8 for other composite elements such as Use Cases) • For Synchronization: 0 = Horizontal, 1 = Vertical Note that there are many more Types than indicated in these examples.
Tablespace	String Notes: Read/Write Associated tablespace for a Table element.
Tag	String Notes: Read/Write Corresponds to the 'Keywords' field in the Enterprise Architect user interface.
TaggedValues	Collection Notes: Read only Returns a collection of TaggedValue objects.
TaggedValuesEx	Collection Notes: Read only Returns a collection of TaggedValue objects belonging to the current element and the elements specialized or realized by the current element.
TemplateParameters	Collection Notes: Read Only A collection of TemplateParameter objects.
Tests	Collection Notes: Read only A collection of Test objects for the current element.
TreePos	Long Notes: Read/Write Sets or gets the tree position.
Type	String Notes: Read/Write The element type (such as Class, Component). Note that Type is case sensitive inside Enterprise Architect and should be provided

	with an initial capital (proper case); valid types are: • Action • Activity • ActivityPartition • ActivityRegion • Actor • Artifact • Association • Boundary • Change • Class • Collaboration • Component • Constraint • Decision • DeploymentSpecification • DiagramFrame • EmbeddedElement • Entity • EntryPoint • Event • ExceptionHandler • ExitPoint • ExpansionNode • ExpansionRegion • Feature • GUIElement • InteractionFragment • InteractionOccurrence • InteractionState • Interface • InterruptibleActivityRegion • Issue • Node • Note • Object • Package • Parameter • Part • Port • ProvidedInterface • Report • RequiredInterface • Requirement

	• Screen • Sequence • State • StateNode • Synchronization • Text • TimeLine • UMLDiagram • UseCase
TypeInfoProperties	Notes: Read only Returns an interface pointer of TypeInfoProperties.
Version	String Notes: Read/Write The version of the element.
Visibility	String Notes: Read/Write The Scope of this element within the current Package. Valid values are: Public, Private, Protected or Package.

Element Methods

Method	Remarks
ApplyGroupLock(string aGroupName)	Boolean Notes: Applies a group lock to the element object, for the specified group, on behalf of the current user. Returns True if the operation is successful; returns False if the operation is unsuccessful. Use 'GetLastError()' to retrieve error information. Parameters: • aGroupName: String - the name of the user group for which to set the group lock
ApplyUserLock()	Boolean Notes: Applies a user lock to the element object for the current user. Returns True if the operation is successful; returns False if the operation is unsuccessful. Use 'GetLastError()' to retrieve error information.
Clone ()	LDISPATCH Notes: Inserts a copy of the selected element under the same parent as the selected element. Returns the newly-created element.
CreateAssociationClass(lon	

g ConnectorID)	Boolean Notes: Makes this element an AssociationClass of the Association with the provided Connector ID; the return value indicates whether the function succeeded in converting the element to an AssociationClass. AssociationClasses are created only where: • The current element is valid • The current element is a Class • The current element is not already an AssociationClass • The specified connector exists • The specified connector is an Association • The specified connector is not already in an AssociationClass pair • The current element is not at either end of the specified connector Parameters: • ConnectorID: Long - the Connector ID of an Association connector
DeleteLinkedDocument()	Boolean Notes: Removes the Linked Document for the element. This method does not display a confirmatory prompt. Returns True if a document was deleted.
GetBusinessRules()	String Notes: Read Only. Returns all the Business Rules for the element.
GetChart	LDISPATCH Notes: For chart elements returns an interface to the chart
GetDecisionTable()	String Notes: Provides read-only access to a Decision Table XML string. Returns the XML data for the Decision Table as a string.
GetElementGrid()	String Notes: Returns an object of type ElementGrid (a Custom Table Artifact element).
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
GetLinkedDocument()	String Notes: Returns a string value containing the element's Linked Document contents, in Rich Text Format. If the element contains no Linked Document, an empty string is returned.
GetRelationSet(EnumRelationSetType Type)	String Notes: Returns a string containing a comma-separated list of ElementIDs of directly- and indirectly-related elements based on the given type. Recurses using the same relation type on all elements it finds, retrieving all dependencies and sub-dependencies of the current element; for example, Object1 depends on Object2, which depends on Object3, therefore this method returns

	Object2 and Object3. To obtain only the direct relationships of the element, use the Connector collection instead.
GetStereotypeList()	String Notes: Returns a comma-separated list of stereotypes allied to this element.
GetTXAlias (string Code, long Flag)	String Notes: Returns the Alias of the element for a given language. Parameters • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Flag: Long - 0 = Get the currently-stored translated Alias - 1 = Get the currently-stored translated Alias, and auto translate if the original Alias has changed - 2 = Always fetch the translated Alias from online
GetTXName (string Code, long Flag)	String Notes: Returns the name of the element for a given language. Parameters • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Flag: Long - 0 = Get the currently-stored translated name - 1 = Get the currently-stored translated name, and auto translate if the original name has changed - 2 = Always fetch the translated name from online
GetTXNote (string Code, long Flag)	String Returns the Notes of the element for a given language. Parameters • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Flag: Long - 0 = Get the currently-stored translated Notes - 1 = Get the currently-stored translated Notes, and auto translate if the original Notes have changed - 2 = Always fetch the translated Notes from online
HasStereotype(string Stereotype)	Boolean Notes: Returns true if the current element has the specified stereotype applied to it. Accepts either qualified or unqualified stereotype names; for example, 'block' or 'SysML1.3::block'. Parameters: • Stereotype: String - the name of the stereotype to search for
IsAssociationClass	Boolean Notes: Returns whether or not the current element is an AssociationClass.
LoadLinkedDocument(stri	Boolean

ng Filename)	Notes: Loads the document from the specified file into the element's Linked Document. Parameters: • FileName: String - the name of the file from which to load the document; both RTF and DOCX input formats are supported
Refresh()	Void Notes: Refreshes the element features in the Browser window. Usually called after adding or deleting attributes or methods, when the user interface is required to be updated as well.
ReleaseUserLock()	Boolean Notes: Releases a user lock or group lock on the element object. Returns True if the operation is successful; returns False if the operation is unsuccessful. Use GetLastError() to retrieve error information.
SaveLinkedDocument(string Filename)	Boolean Notes: Saves the Linked Document for this element to the specified file. Returns False if the element does not have a Linked document or fails to save the file. Parameters: • FileName: String - the name of the file to save to disk The output format will be determined by the file's extension - currently rtf, docx and pdf are supported; if an invalid extension is used, it will write the file in RTF format regardless of the extension
SetAppearance(long Scope, long Item, long Value)	Void Notes: Sets the visual appearance of the element. Parameters: • Scope: Long - Scope of appearance set to modify 1 - Base (Default appearance across entire model) To set appearance for the element (diagram object) in a selected diagram only, see *Setting The Style* in the *DiagramObject Class* topic • Item: Long - Appearance feature to modify 0 - Background color 1 - Font Color 2 - Border Color 3 - Border Width • Value: Long - Value to set appearance to
SetCompositeDiagram()	Boolean Notes: Sets the composite diagram of the element. Parameters: • GUID: String - the GUID of the composite diagram; a blank GUID will remove the link to the composite diagram
SetCreated(Date NewVal)	Void Notes: Deprecated This method is no longer supported.
SetModified(Date NewVal)	Void

	Notes: Deprecated This method is no longer supported.
SetTXAlias (string Code, string Translation)	String Notes - Set the translated Alias of the element for a given language. • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Translation: String - The translated Alias
SetTXName (string Code, string Translation)	String Notes - Set the translated name of the element for a given language. • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Translation: String - The translated name
SetTXNote (string Code, string Translation)	String Notes - Set the translated Notes of the element for a given language. • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Translation: String - The translated Notes
SynchConstraints(string Profile, string Stereotype)	Boolean Notes: Synchronizes the constraints of a UML Profile item for this element, only if the specified stereotype has been applied. Parameters: • Profile: String - Name of the profile that contains the stereotype • Stereotype: String - Name of the profile stereotype for which the default constraints are to be synchronized
SynchTaggedValues(string Profile, string Stereotype)	Boolean Notes: Synchronizes the Tagged Values of a UML Profile item for this element, only if the specified stereotype has been applied. Parameters: • Profile: String - Name of the profile that contains the stereotype • Stereotype: String - Name of the profile stereotype for which the default tags are to be synchronized
UnlinkFromAssociation	Boolean Notes: Performs the opposite of CreateAssociationClass().
Update()	Boolean Notes: Updates the current element object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

ElementGrid Class

The ElementGrid object represents a Custom Table, which is used to display custom data in tabular format on a diagram, the data being provided by the user rather than generated by the system.

The ElementGrid object is accessible from an Element object, using the GetElementGrid() method.

Associated table in repository

t_object

ElementGrid Methods

Method	Remarks
GetCell (int nrow, int ncell)	Variant Notes: The cell value is return as a variant value. Parameters: • nRow: Integer - the number of the row containing the cell • nCell: Integer - the number of the cell in the row (the column number)
GetColumnCount ()	Integer Notes: Returns the number of columns in the grid.
GetRowCount ()	Integer Notes: Returns the number of rows in the grid.
SetCell (int nRow, int nCell, variant sValue)	Boolean Notes: Sets a value in the specified cell. Parameters: • nRow: Integer - specifies the row into which to insert the value • nCell: Integer - specifies the cell (column number) into which to insert the value • sValue: Variant - specifies the value to set in the cell
SetGridSize (int nRows, int nColumns)	Boolean Notes: Sets the size of the grid in rows and columns. The size can be set and reset; any data outside the bounds of the new grid size will be lost on resize. Parameters: • nRows: Integer - the number of rows in the table grid • nColumns: Integer - the number of columns in the table grid

File Class

A File represents an associated file for an element. Files are accessed through the Element Files collection.

Associated table in repository

t_objectfiles

File Attributes

Attribute	Remarks
FileDate	String Notes: Read/Write The file date when the entry was created.
Name	String Notes: Read/Write The file name can be a logical file or a reference to a web address (using http://).
Notes	String Notes: Read/Write Notes about the file.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Size	String Notes: Read/Write The file size.
Type	String Notes: Read/Write The file type.

File Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in

	relation to this object.
Update()	Boolean Notes: Updates the current File object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Issue (Maintenance) Class

An Issue is either a Change or a Defect, is associated with the containing element, and is accessed through the Issues collection of an element.

Associated table in repository

t_objectproblems

Issue Attributes

Attribute	Remarks
DateReported	Date Notes: Read/Write The date the issue was reported.
DateResolved	Date Notes: Read/Write The date the issue was resolved.
ElementID	Long Notes: Read/Write The ID of the element associated with this issue.
Name	String Notes: Read/Write The Issue name; that is, the Issue itself.
Notes	String Notes: Read/Write The Issue description.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Priority	String Notes: Read/Write The priority of the Issue - Low, Medium or High.
Reporter	String Notes: Read/Write The user ID of the person reporting the issue.

Resolver	String Notes: Read/Write The user ID of the person resolving the issue.
ResolverNotes	String Notes: Read/Write Notes entered by the resolver about resolution of the Issue.
Severity	String Notes: Read/Write The Issue severity - Low, Medium or High.
Status	String Notes: Read/Write The current status of the issue.
Type	Variant Notes: Read/Write The Issue type - Defect, Change, Issue or Task.
Version	String Notes: Read/Write The version associated with the issue. Note that this method is only available through a Dispatch interface. Object ob = Issue; Print ob.Version;

Issue Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Updates the current Issue object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Metric Class

A Metric is a named item with a weighting that can be associated with an element for purposes of building metrics about the model. Metrics are accessed through the Element Metrics collection.

Associated table in repository

t_objectmetrics

Metric Attributes

Attribute	Remarks
Name	String Notes: Read/Write The name of the metric.
Notes	String Notes: Read/Write Notes about this metric.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Type	String Notes: Read/Write The metric type.
Weight	Long Notes: Read/Write A user-defined weighting for estimation or metric purposes.

Metric Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Updates the current Metric object after modification or appending a new

	item. If False is returned, check the 'GetLastError()' function for more information.

Requirement Class

An Element Requirement object holds information about the requirements of an element in the context of the model. Requirements can be accessed using the Element Requirements collection.

Associated table in repository

t_objectrequires

Requirement Attributes

Attribute	Remarks
Difficulty	String Notes: Read/Write The estimated difficulty of implementing the requirement.
LastUpdate	Date Notes: Read/Write The date the requirement was last updated.
Name	String Notes: Read/Write The requirement itself.
Notes	String Notes: Read/Write Further notes on the requirement.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
ParentID	Long Notes: Read only The ElementID of the element to which this requirement applies.
Priority	String Notes: Read/Write The assigned priority of the requirement.
RequirementID	Long Notes: Read only A local ID for this requirement.

Stability	String Notes: Read/Write The estimated stability of the requirement. This is an indication of the probability of the requirement - or understanding of the requirement - changing. High stability indicates a low probability of the requirement changing.
Status	String Notes: Read/Write The current status of the requirement.
Type	String Notes: Read/Write The requirement type.

Requirement Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Updates the current Requirement object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Resource Class

An element Resource is a named person/task pair with timing constraints and percent complete indicators. Use this to manage the work associated with delivering an element.

Associated table in repository

t_objectresources

Resource Attributes

Attribute	Description
ActualHours	Long Notes: Read/Write The time already expended on the task, in hours, days or other units.
DateEnd	Date Notes: Read/Write The expected end date.
DateStart	Date Notes: Read/Write The date to start work.
ExpectedHours	Long Notes: Read/Write The total expected time the task might run, in hours, days or other units.
History	String Notes: Read/Write Gets or sets history text.
Name	String Notes: Read/Write The name of the resource (for example, a person's name).
Notes	String Notes: Read/Write Descriptive notes.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.

PercentComplete	Long Notes: Read/Write The current percent complete figure.
Role	String Notes: Read/Write The role the resource plays in implementing the element.
Time	Long Notes: Read/Write The time expected to complete the task; a numeric indicating the number of days.

Resource Methods

Method	Description
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used as an exception is thrown when an error occurs.
Update()	Boolean Notes: Update the current Resource object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Risk Class

A Risk object represents a named risk associated with an element. It is used for project management purposes. Risks can be accessed through the Element Risks collection.

Associated table in repository

t_objectrisks

Risk Attributes

Attribute	Description
Name	String Notes: Read/Write The name of the risk.
Notes	String Notes: Read/Write Further notes describing the risk.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Type	String Notes: Read/Write The risk type associated with this element.
Weight	Long Notes: Read/Write A weighting for estimation or metric purposes.

Risk Methods

Method	Description
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Update the current Risk object after modification or appending a new item.

	If False is returned, check the 'GetLastError()' function for more information.

Scenario Class

A Scenario corresponds to a Collaboration or Use Case instance. Each Scenario is a path of execution through the logic of a Use Case. Scenarios can be added to using the Element Scenarios collection.

Associated table in repository

t_objectscenarios

Scenario Attributes

Attribute	Description
Name	String Notes: Read/Write The Scenario name.
Notes	String Notes: Read/Write A description of the Scenario, usually containing the steps to execute the scenario.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
ScenarioGUID	String Notes: Read/Write A unique ID for the Scenario, used to identify the Scenario unambiguously within a model.
Steps	Collection of ScenarioStep Class Notes: Read only A collection of step objects for this Scenario. Use the 'AddNew' and 'Delete' functions to manage steps. 'AddNew' passes the step name and '1' as the type for an actor step.
Type	String Notes: Read/Write The scenario type (for example, Basic Path).
Weight	Long Notes: Read/Write Currently used to position scenarios in the scenario list (that is, List Position).
XMLContent	String

	Notes: Read/Write A structured field that can contain scenario details in XML format. It is recommended that you use the 'Steps' collection to read or modify this field.

Scenario Methods

Method	Description
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Update the current Scenario object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

ScenarioExtension Class

ScenarioExtension Attributes

Attribute	Description
ExtensionGUID	String Notes: Read/Write A unique GUID for this Extension.
Join	String Notes: Read/Write The GUID of the step where this Extension rejoins the Scenario.
JoiningStep	ScenarioStep Notes: Read only The actual step where this Extension rejoins the Scenario, if any.
Level	String Notes: Read only The number of this Extension as shown in the scenario editor. This is derived from the value of Pos for this object and the owning step.
Name	String Notes: Read/Write The Extension name. This should match the name of the linked scenario.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Pos	Long Notes: Read/Write The position of the Extension in the Extensions list.
Scenario	Scenario Notes: Read only The scenario that is executed as an alternative path for this Extension.

ScenarioStep Class

ScenarioStep Attributes

Attribute	Description
Extensions	Collection of ScenarioExtension Notes: Read only A collection of ScenarioExtension objects that specify how the scenario is extended from this step. The arguments to 'AddNew' should match the name and GUID of the alternative scenario being linked to.
Level	String Notes: Read only The number of this Step as shown in the scenario editor. This is derived from the value of Pos.
Link	String Notes: Read/Write The GUID of a Use Case that is relevant to this step.
LinkedElement	Element Notes: Read only The actual element specified by Link, if any.
Name	String Notes: Read/Write The step name.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Pos	Long Notes: Read/Write The position of the 'Step' in the 'Scenario Step' list.
Results	String Notes: Read/Write Any results that are given from this step.
State	String Notes: Read/Write A description of the state the system enters when this Step is executed.
StepGUID	String

	Notes: Read/Write A unique GUID for this Step.
StepType	ScenarioStepType Notes: Read/Write Identifies whether this step is being performed by a user or the system.
Uses	String Notes: Read/Write The input and requirements that are relevant to this step.
UsesElementList	Collection of Element Notes: Read only Indicates that the Scenarios view 'Uses' field is a linked element list.

ScenarioStep Methods

Method	Description
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Updates the current ScenarioStep object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

ScenarioExtension Methods

Method	Description
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Updates the current ScenarioExtension object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

TaggedValue Class

A TaggedValue is a named property and value associated with an element. Tagged Values can be accessed through the TaggedValues collection.

Associated table in repository

t_objectproperties

TaggedValue Attributes

Attribute	Description
ElementID	Long Notes: Read/Write The local ID of the associated element.
FQName	String Notes: Read only The fully-qualified name of the tag.
Name	String Notes: Read/Write The name of the tag.
Notes	String Notes: Read/Write Further descriptive notes about this tag. If 'Value' is set to '<memo>', then 'Notes' should contain the actual Tagged Value content.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
PropertyGUID	String Notes: Read/Write The global ID of the tag.
PropertyID	Long Notes: Read only The local ID of the tag.
Value	String Notes: Read/Write

	The value assigned to this tag. This field has a 255 character limit. If the value is greater than 255 characters long, set the value to "<memo>" and insert the body of text in the 'Notes' attribute. When reading existing Tagged Values, if 'Value" = "<memo>" then the developer should read the actual body of text from the 'Notes' attribute.

TaggedValue Methods

Method	Description
GetAttribute(string propName)	String Notes: Returns the text of a single named property within a structured Tagged Value. Parameters: • propName: String - the name of the property for which the text is being returned
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
HasAttributes()	Boolean Notes: Returns True if the Tagged Value is a structured Tagged Value with one or more properties.
SetAttribute(string propName, string propValue)	Boolean Notes: Sets the text of a single named property within a structured Tagged Value. Parameters: • propName: String - the name of the property for which the text is being set • propValue: the value of the property
Update()	Boolean Notes: Updates the current TaggedValue object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Test Class

A Test is a single Test Case applied to an element. Tests are added and accessed through the Element Tests collection.

Associated table in repository

t_objecttests

Test Attributes

Attribute	Description
AcceptanceCriteria	String Notes: Read/Write The acceptance criteria for successful execution.
CheckedBy	String Notes: Read/Write User ID of the person confirming the results.
Class	Long Notes: Read/Write The test Class: 1 = Unit Test 2 = Integration Test 3 = System Test 4 = Acceptance Test 5 = Scenario Test 6 = Inspection Test
DateRun	Date Notes: Read/Write The date the test was last run.
Input	String Notes: Read/Write Input data for the test.
Name	String Notes: Read/Write The test name.
Notes	String Notes: Read/Write

	Detailed notes about test to be carried out.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
RunBy	String Notes: Read/Write The user ID of the person conducting the test.
Status	String Notes: Read/Write The current status of the test.
TestResults	Variant Notes: Read/Write Results of test.
Type	String Notes: Read/Write The test type, such as Load or Regression.

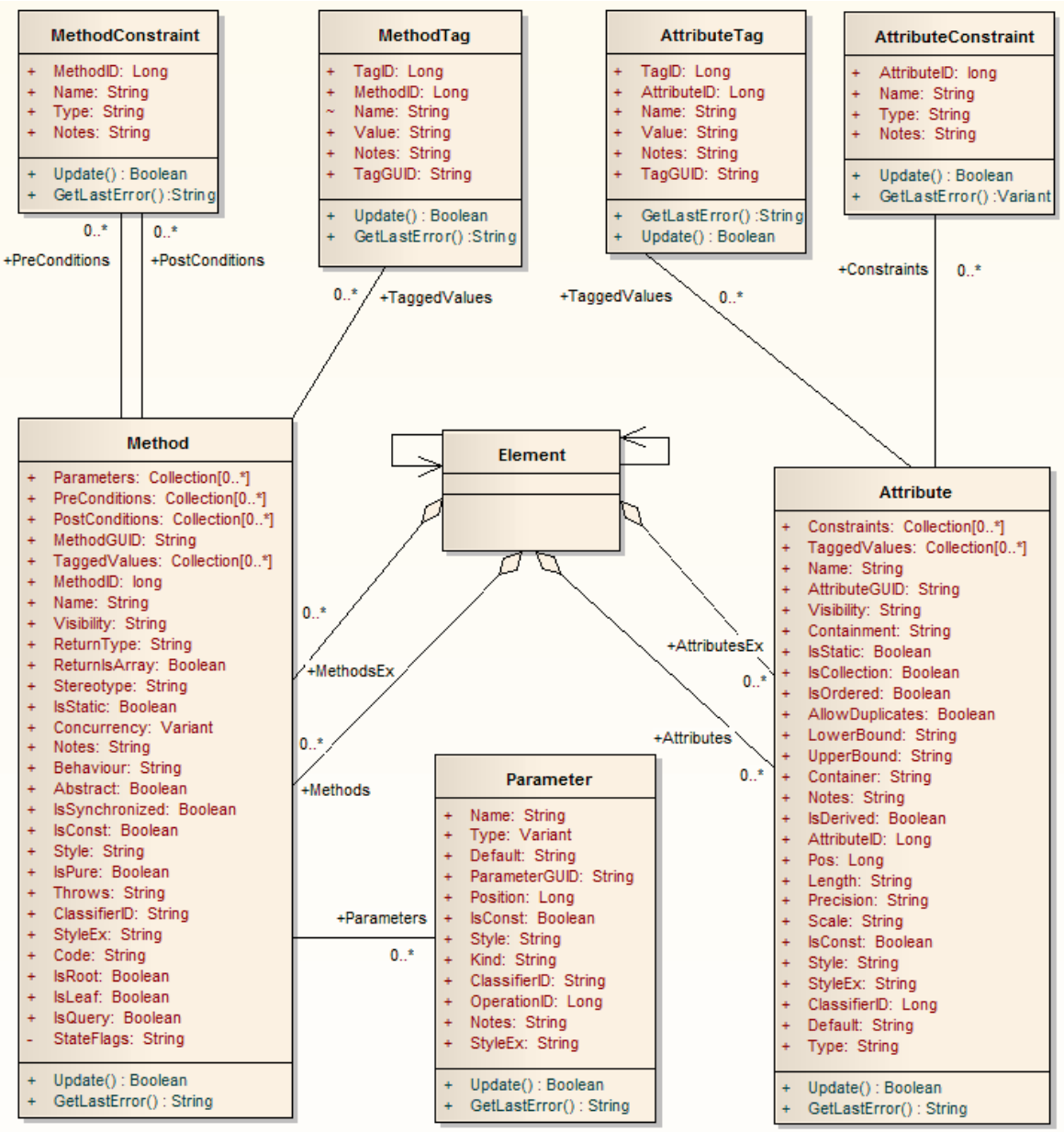
Test Methods

Method	Description
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Update the current Test object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Element Features Package

The ElementFeatures Package contains descriptions of the model interfaces that enable access to operations and attributes, and their associated Tagged Values and constraints.

This diagram illustrates the components associated with element features. These include attributes and methods, and their associated constraints and Tagged Values. It also includes the Parameter object that defines the arguments associated with an operation (Method).



Attribute Class

An attribute corresponds to a UML Attribute. It contains further collections for constraints and Tagged Values. Attributes are accessed from the element Attributes collection.

Associated table in repository

t_attribute

Attribute Attributes

Attribute	Remarks
Alias	String Notes: Read/Write Contains the (optional) 'Alias' property for this attribute. This can be used interchangeably with the Style attribute.
AllowDuplicates	Boolean Notes: Read/Write Indicates if duplicates are allowed in the collection. If the attribute represents a database column this, when set, represents the 'Not Null' option.
AttributeGUID	String Notes: Read only A globally unique ID for the current attribute. This attribute is system generated.
AttributeID	Long Notes: Read only The local ID number of the attribute.
ClassifierID	Long Notes: Read/Write The classifier ID, if appropriate, indicating the base type associated with the attribute, if not a primitive type.
Constraints	Collection Notes: Read only A collection of AttributeConstraint objects, used to access and manage constraints associated with this attribute.
Container	String Notes: Read/Write The container type.

Containment	String Notes: Read/Write The type of containment - Not Specified, By Reference or By Value.
Default	String Notes: Read/Write The initial value assigned to this attribute.
FQStereotype	String Notes: Read Only The fully-qualified stereotype name in the format "Profile::Stereotype". One or more fully-qualified stereotype names can be assigned to StereotypeEx.
IsCollection	Boolean Notes: Read/Write Indicates if the current feature is a collection or not. If the attribute represents a database column this, when set, represents a Foreign Key.
IsConst	Boolean Notes: Read/Write A flag indicating if the attribute is Const or not.
IsDerived	Boolean Notes: Read/Write Indicates if the attribute is derived (that is, a calculated value).
IsID	Boolean Notes: Read/Write Indicates if the attribute uniquely identifies an instance of the containing Class, or not.
IsOrdered	Boolean Notes: Read/Write Indicates if a collection is ordered or not. If the attribute represents a database column this, when set, represents a Primary Key.
IsStatic	Boolean Notes: Read/Write Indicates if the current attribute is a static feature or not. If the attribute represents a database column this, when set, represents the 'Unique' option.
Length	String Notes: Read/Write The attribute length, where applicable.
LowerBound	String Notes: Read/Write A value for the collection lower boundary.

Name	String Notes: Read/Write The attribute name.
Notes	String Notes: Read/Write Further notes on this attribute.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
ParentID	Long Notes: Read only Returns the ElementID of the element that this attribute is a part of.
Pos	Long Notes: Read/Write The position of the attribute in the Class attribute list.
Precision	String Notes: Read/Write The precision value.
RedefinedProperty	String Notes: Read/Write Corresponds to the 'Redefined Property' field on the 'Detail' page of the attribute 'Properties' dialog, or the UML *redefinedProperty* attribute. Contains a comma separated list of GUIDs.
Scale	String Notes: Read/Write The scale value.
Stereotype	String Notes: Read/Write Sets or gets the stereotype for this attribute. When setting this attribute, LastError (for the GetLastError method) will be non-empty if an error occurs.
StereotypeEx	String Notes: Read/Write Provides all the applied stereotypes of the attribute, in a comma-separated list. Reading the value will provide the stereotype name only; assigning the value accepts either fully-qualified or simple names. When setting this attribute, LastError (for the GetLastError method) will be non-empty if an error occurs.

Style	String Notes: Read/Write Contains the (optional) Alias property for this attribute. This can be used interchangeably with the Alias attribute.
StyleEx	String Notes: Read/Write Advanced style settings, reserved for the use of Sparx Systems.
SubsettedProperty	String Notes: Read/Write Corresponds to the 'Subsetted Property' field on the 'Detail' page of the attribute 'Properties' dialog, or the UML *subsettedProperty* attribute. Contains a comma separated list of GUIDs.
TaggedValues	Collection of type AttributeTag Notes: Read only A collection of AttributeTag objects, used to access and manage Tagged Values associated with this attribute.
TaggedValuesEx	Collection of type TaggedValue Notes: Read only A collection of TaggedValue objects belonging to the current attribute and the TaggedValuesEx property of its classifier.
Type	String Notes: Read/Write The attribute type (by name; also see *ClassifierID*).
TypeInfoProperties	Notes: Read only Returns an interface pointer of TypeInfoProperties.
UpperBound	String Notes: Read/Write A value for the collection upper boundary.
Visibility	String Notes: Read/Write Identifies the scope of the attribute - Private, Protected, Public or Package.

Attribute Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in

	relation to this object.
GetTXAlias (string Code, long Flag)	String Notes: Returns the Alias of the element for a given language. Parameters • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Flag: Long - 0 = Get the currently-stored translated Alias - 1 = Get the currently-stored translated Alias, and auto translate if the original Alias has changed - 2 = Always fetch the translated Alias from online
GetTXName (string Code, long Flag)	String Notes: Returns the name of the element for a given language. Parameters • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Flag: Long - 0 = Get the currently-stored translated name - 1 = Get the currently-stored translated name, and auto translate if the original name has changed - 2 = Always fetch the translated name from online
GetTXNote (string Code, long Flag)	String Returns the Notes of the element for a given language. Parameters • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Flag: Long - 0 = Get the currently-stored translated Notes - 1 = Get the currently-stored translated Notes, and auto translate if the original Notes have changed - 2 = Always fetch the translated Notes from online
SetTXAlias (string Code, string Translation)	String Notes - Set the translated Alias of the element for a given language. • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Translation: String - The translated Alias
SetTXName (string Code, string Translation)	String Notes - Set the translated name of the element for a given language. • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Translation: String - The translated name
SetTXNote (string Code, string Translation)	String Notes - Set the translated Notes of the element for a given language. • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog)

	• Translation: String - The translated Notes
Update()	Boolean Notes: Updates the current attribute object after modifying or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

AttributeConstraint Class

An AttributeConstraint is a constraint associated with the current Attribute.

Associated table in repository

t_attributeconstraints

AttributeConstraint Attributes

Attribute	Remarks
AttributeID	Long Notes: Read/Write The ID of the attribute this constraint applies to.
Name	String Notes: Read/Write The name of the constraint.
Notes	String Notes: Read/Write Descriptive notes about the constraint.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Type	String Notes: Read/Write The type of constraint.

AttributeConstraint Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Update the current AttributeConstraint object after modification or appending a new item.

	If False is returned, check the 'GetLastError()' function for more information.

AttributeTag Class

An AttributeTag represents a Tagged Value associated with an attribute.

Associated table in repository

t_attributetag

AttributeTag Attributes:

Attribute	Remarks
AttributeID	Long Notes: Read/Write The local ID of the attribute associated with this Tagged Value.
FQName	String Notes: Read only The fully-qualified name of the tag.
Name	String Notes: Read/Write The name of the tag.
Notes	String Notes: Read/Write Further descriptive notes about this tag. If 'Value' is set to '<memo>', then 'Notes' should contain the actual Tagged Value content.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
TagGUID	String Notes: Read/Write A globally unique ID for this Tagged Value.
TagID	Long Notes: Read only The local ID to identify the Tagged Value.
Value	String Notes: Read/Write The value assigned to this tag.

	This field has a 255 character limit. If the value is greater than 255 characters long, set the value to "<memo>" and insert the body of text in the 'Notes' attribute. When reading existing Tagged Values, if 'Value' = "<memo>" then the developer should read the actual body of text from the 'Notes' attribute.

AttributeTag Methods:

Method	Remarks
GetAttribute(string propName)	String Notes: Returns the text of a single named property within a structured Tagged Value.
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used as an exception is thrown when an error occurs.
HasAttributes()	Boolean Notes: Returns True if the Tagged Value is a structured Tagged Value with one or more properties.
SetAttribute(string propName, string propValue)	Boolean Notes: Sets the text of a single named property within a structured Tagged Value.
Update()	Boolean Notes: Updates the current AttributeTag object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

CustomProperties Collection

The CustomProperties collection contains 0 or more CustomProperties associated with the current element. These properties provide advanced UML configuration options, and must not be added to or deleted. The value of each property can be set.

CustomProperty

Attribute	Remarks
Name	String Notes: Read only The CustomProperty name.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Value	String Notes: Read/Write The value associated with this CustomProperty. This can be: • A string • The Boolean values True or False, or • An enumeration value from a defined list The UML 2.5 specification in general provides information on the kinds of enumeration relevant here.

Notes

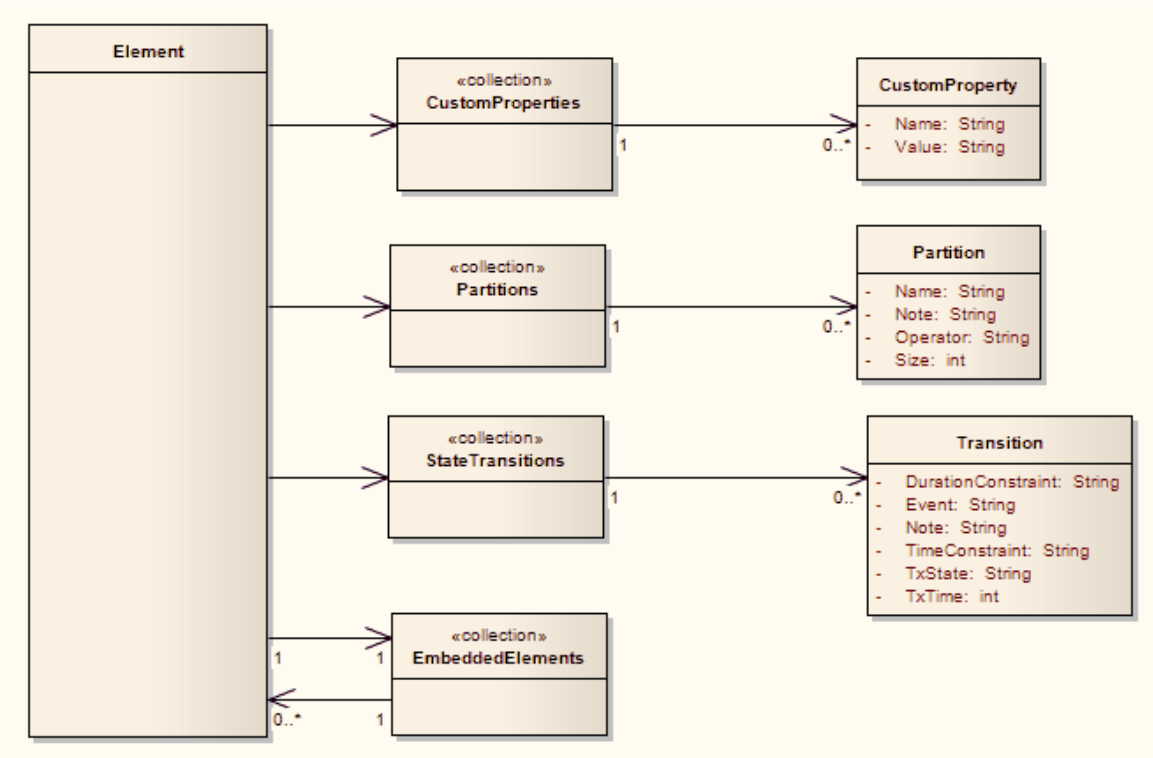
- The number and type of properties vary depending on the actual element

EmbeddedElements Collection

In UML 2.5 an element can have one or more embedded elements such as Ports, Pins, Parameters or ObjectNodes. These are attached to the boundary of the host element and cannot be moved off the element. They are owned by their host element. This collection gives easy access to the set of elements embedded on the surface of an element. Note that some embedded elements can have their own embedded element collection (for example, Ports can have Interfaces embedded on them).

The EmbeddedElements collection contains Element objects.

Example



Method Class

A method represents a UML operation. It is accessed from the Element Methods collection and includes collections for parameters, constraints and Tagged Values.

Associated table in repository

t_operation

Method Attributes

Attribute	Remarks
Abstract	Boolean Notes: Read/Write A flag indicating if the method is abstract (1) or not (0).
Behavior	String Notes: Read/Write Some further explanatory behavior notes (for example, pseudocode). In earlier releases of Enterprise Architect this attribute had the UK/Australian spelling 'Behaviour'; this is still present for backwards compatibility, but please now use the 'Behavior' attribute for consistency.
ClassifierID	String Notes: Read/Write The Classifier ID that applies to the ReturnType.
Code	String Notes: Read/Write An optional field to hold the method code (used for the 'Initial Code' field).
Concurrency	Variant Notes: Read/Write Indicates the concurrency type of the method.
FQStereotype	String Notes: Read Only The fully-qualified stereotype name in the format "Profile::Stereotype". One or more fully-qualified stereotype names can be assigned to StereotypeEx.
IsConst	Boolean Notes: Read/Write A flag indicating that the method is Const.
IsLeaf	Boolean

	Notes: Read/Write A flag to indicate if the method is a Leaf (cannot be overridden).
IsPure	Boolean Notes: Read/Write A flag indicating that the method is defined as 'Pure' in C++.
IsQuery	Boolean Notes: Read/Write A flag to indicate if the method is a query (that is, does not alter Class variables).
IsRoot	Boolean Notes: Read/Write A flag to indicate if the method is Root.
IsStatic	Boolean Notes: Read/Write A flag to indicate a static method.
IsSynchronized	Boolean Notes: Read/Write A flag indicating a Synchronized method call.
MethodGUID	String Notes: Read/Write A globally unique ID for the current method. This is system generated.
MethodID	Long Notes: Read only A local ID for the current method, only valid within this .eap file.
Name	String Notes: Read/Write The method name.
Notes	String Notes: Read/Write Descriptive notes on the method.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Parameters	Collection Class Notes: Read only The Parameters collection for the current method, used to add and access parameter objects for the current method.

ParentID	Long Notes: Read only Returns the ElementID of the element that this method belongs to.
Pos	Long Notes: Read/Write Specifies the position of the method within the set of operations defined for a Class.
PostConditions	Collection Class Notes: Read only The PostConditions (constraints) as they apply to this method. This returns a MethodConstraint object of type 'post'.
PreConditions	Collection Class Notes: Read only The PreConditions (constraints) as they apply to this method. This returns a MethodConstraint object of type 'pre'.
ReturnIsArray	Boolean Notes: Read/Write A flag to indicate that the return value is an array.
ReturnType	String Notes: Read/Write The return type for the method; this can be a primitive data type or a Class or Interface type.
StateFlags	String Notes: Read/Write Some flags as applied to methods in State elements.
Stereotype	String Notes: Read/Write The method stereotype (optional). When setting this attribute, LastError (for the GetLastError method) will be non-empty if an error occurs.
StereotypeEx	String Notes: Read/Write All the applied stereotypes of the method in a comma-separated list. Reading the value will provide the stereotype name only; assigning the value accepts either fully-qualified or simple names. When setting this attribute, LastError (for the GetLastError method) will be non-empty if an error occurs.
Style	String Notes: Read/Write Contains the Alias property for this method.

StyleEx	String Notes: Read/Write Advanced style settings, reserved for the use of Sparx Systems.
TaggedValues	Collection Class of type MethodTag Class Notes: Read only The TaggedValues collection for the current method. This accesses a list of MethodTag objects.
Throws	String Notes: Read/Write Exception information. Valid input for setting the Throws is: • GUID String - the GUID of an element in the model or a comma-separated list of element GUIDS • <none> - removes the existing Throws set
TypeInfoProperties	Notes: Read only Returns an interface pointer of TypeInfoProperties.
Visibility	String Notes: Read/Write The method scope - Public, Protected, Private or Package.

Method Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
GetTXAlias (string Code, long Flag)	String Notes: Returns the Alias of the element for a given language. Parameters • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Flag: Long - 0 = Get the currently-stored translated Alias - 1 = Get the currently-stored translated Alias, and auto translate if the original Alias has changed - 2 = Always fetch the translated Alias from online
GetTXName (string Code, long Flag)	String Notes: Returns the name of the element for a given language. Parameters • Code: String - Two-letter language code (found on the 'Translations' page of

	the 'Manage Model Options' dialog) • Flag: Long - 0 = Get the currently-stored translated name - 1 = Get the currently-stored translated name, and auto translate if the original name has changed - 2 = Always fetch the translated name from online
GetTXNote (string Code, long Flag)	String Returns the Notes of the element for a given language. Parameters • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Flag: Long - 0 = Get the currently-stored translated Notes - 1 = Get the currently-stored translated Notes, and auto translate if the original Notes have changed - 2 = Always fetch the translated Notes from online
SetTXName (string Code, string Translation)	String Notes - Set the translated name of the element for a given language. • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Translation: String - The translated name
SetTXAlias (string Code, string Translation)	String Notes - Set the translated Alias of the element for a given language. • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Translation: String - The translated Alias
SetTXNote (string Code, string Translation)	String Notes - Set the translated Notes of the element for a given language. • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Translation: String - The translated Notes
Update()	Boolean Notes: Update the current method object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

MethodConstraint Class

A MethodConstraint is a condition imposed on a method. It is accessed through either the Method PreConditions or Method PostConditions collection.

Associated table in repository

t_operationpres and t_operationposts

MethodConstraint Attributes

Attribute	Remarks
MethodID	Long Notes: Read/Write The local ID of the associated method.
Name	String Notes: Read/Write The name of the constraint.
Notes	String Notes: Read/Write Descriptive notes about this constraint.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Type	String Notes: Read/Write The constraint type.

MethodConstraint Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used as an exception is thrown when an error occurs.
Update()	Boolean

	Notes: Update the current MethodConstraint object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

MethodTag Class

A MethodTag is a Tagged Value associated with a method.

Associated table in repository

t_operationtag

MethodTag Attributes:

Attribute	Remarks
FQName	String Notes: Read only The fully-qualified name of the tag.
MethodID	Long Notes: Read/Write The ID of the associated method.
Name	String Notes: Read/Write The tag or name of the property.
Notes	String Notes: Read/Write Further descriptive notes about this tag. If 'Value' is set to '<memo>', then 'Notes' should contain the actual Tagged Value content.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
TagGUID	String Notes: Read/Write A unique GUID for this Tagged Value.
TagID	Long Notes: Read only A unique ID for this Tagged Value.
Value	String Notes: Read/Write The value assigned to this tag.

	This field has a 255 character limit. If the value is greater than 255 characters long, set the value to "<memo>" and insert the body of text in the 'Notes' attribute. When reading existing Tagged Values, if 'Value' = "<memo>" then the developer should read the actual body of text from the 'Notes' attribute.

MethodTag Methods:

Method	Remarks
GetAttribute(string propName)	String Notes: Returns the text of a single named property within a structured Tagged Value.
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used as an exception is thrown when an error occurs.
HasAttributes()	Boolean Notes: Returns True if the Tagged Value is a structured Tagged Value with one or more properties.
SetAttribute(string propName, string propValue)	Boolean Notes: Sets the text of a single named property within a structured Tagged Value.
Update()	Boolean Notes: Updates the current MethodTag object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Parameter Class

A Parameter object represents a method argument and is accessed through the Method Parameters collection.

Associated table in repository

t_operationparams

Parameter Attributes

Attribute	Remarks
Alias	String Notes: Read/Write An optional alias for this parameter.
ClassifierID	String Notes: Read/Write A ClassifierID for the parameter, if known.
Default	String Notes: Read/Write A default value for this parameter.
IsConst	Boolean Notes: Read/Write A flag indicating that the parameter is Const (cannot be altered).
Kind	String Notes: Read/Write The parameter kind - in, inout, out, or return.
Name	String Notes: Read/Write The parameter name; this must be unique for a single method.
Notes	String Notes: Read/Write Descriptive notes.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
OperationID	Long

	Notes: Read only The ID of the method associated with this parameter.
ParameterGUID	String Notes: Read/Write A system generated, globally unique ID for the current Parameter.
Position	Long Notes: Read/Write The position of the parameter in the argument list.
Stereotype	String Notes: Read/Write The first stereotype of the parameter. When setting this attribute, LastError (for the GetLastError method) will be non-empty if an error occurs.
StereotypeEx	String Notes: Read/Write All the applied stereotypes of the parameter in a comma-separated list. Reading the value will provide the stereotype name only; assigning the value accepts either fully-qualified or simple names. When setting this attribute, LastError (for the GetLastError method) will be non-empty if an error occurs.
Style	String Notes: Read/Write Some style information.
StyleEx	String Notes: Read/Write Advanced style settings, reserved for the use of Sparx Systems.
TaggedValues	Collection Class of type ParamTag Class Notes: Read/Write The GUID of the parameter with which this ParamTag is associated.
Type	Variant Notes: Read/Write The parameter type; can be a primitive type or a defined classifier.
TypeInfoProperties	Notes: Read only Returns an interface pointer of TypeInfoProperties.

Parameter Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Update the current Parameter object after modifying or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

ParamTag Class

A ParamTag is a Tagged Value associated with a method parameter.

Associated table in repository

t_taggedvalue

ParamTag Attributes

Attribute	Remarks
ElementGUID	String Notes: Read/Write The GUID of the parameter with which this ParamTag is associated.
FQName	String Notes: Read only The fully qualified name of the tag.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
PropertyGUID	String Notes: Read/Write A system generated GUID to identify the Tagged Value.
Tag	String Notes: Read/Write The actual tag name.
Value	String Notes: Read/Write The value associated with this tag.

ParamTag Methods

Method	Remarks
GetAttribute(string propName)	String Notes: Returns the text of a single named property within a structured Tagged

	Value.
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
HasAttributes()	Boolean Notes: Returns True if the Tagged Value is a structured Tagged Value with one or more properties.
SetAttribute(string propName, string propValue)	Boolean Notes: Sets the text of a single named property within a structured Tagged Value.
Update()	Boolean Notes: Updates the current ParamTag object after modifying or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Partitions Collection

A collection of internal element partitions (regions). This is commonly seen in Activity, State, Boundary, Diagram Frame and similar elements. Not all elements support partitions.

This collection contains a set of Partition elements. The set is read/write: information is not saved until the host element is saved, so ensure that you call the Element.Save method after making changes to a Partition.

Partition Attributes

Attribute	Remarks
Name	String Notes: Read/Write The partition name; this can represent a condition or constraint in some cases.
Note	String Notes: Read/Write A free text note associated with this partition.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Operator	String Notes: Read/Write An optional operator value that specifies the partition type.
Size	String Notes: Read/Write The vertical or horizontal width of the partition in pixels.

Properties Class

Properties

Properties Attributes

Attribute	Remarks
Count	Long Notes: The number of properties that are available for this object.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.

Properties Methods

Property

Method	Remarks
Item(object Index)	Property Notes: Returns a property either by name or by a zero-based integer offset into the list of properties. Parameter: • Index: Variant - either a string representing the property name or an integer representing the zero-based offset into the property list

Property Attributes

Attribute	Remarks
Name	String Notes: Read only The name of the property. The object to which the properties list applies can have an automation property with the same name, in which case the data accessed through Value is identical to that obtained through the automation property.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.

Type	PropType Notes: Read only Provides an indication of what sort of data is going to be stored by this property. This restriction can be further defined by the Validation attribute.
Validation	String Notes: Read only An optional string that is used to validate any data that is passed to the Value attribute. This string is used by the programmer at run time to provide an indication of what is expected, and by Enterprise Architect to ensure that the submitted data is appropriate.
Value	Variant Notes: Read/write The value of the property as defined in the other fields.

TemplateParameter Class

A TemplateParameter for a template signature specifies a formal parameter that will be substituted by an actual parameter (or the default) in a TemplateBinding relationship on a Class element.

Associated table in repository

t_xref

TemplateParameter Attributes

Attribute	Remarks
Constraint	String Notes: Read/Write The name of the Classifier that acts as the constraint value.
Default	String Notes: Read/Write The name of the Classifier that acts as the default value.
Name	String Notes: Read/Write The name of the Template Parameter.
ObjectType	ObjectType Notes: Read Only Distinguishes objects referenced through a Dispatch interface.
TemplateParameterID	String Notes: Read Only The Enterprise Architect Globally Unique ID (GUID) of the current Template Parameter, in the XrefID column of t_xref.
Type	String Notes: Read/Write The Template Parameter type.

TemplateParameter Methods

Method	Remarks
GetLastError()	String

	Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Updates the current TemplateParameter object after modifying or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Transitions Collection

The Transitions collection applies only to Timeline elements.

A Timeline element displays 0 or more state transitions at set times on its extent. This collection enables you to access the transition set. You can also access additional information by referring to the connectors associated with the Timeline, and by referencing messages passed between timelines. Note that any changes made to elements in this collection are only saved when the main element is saved.

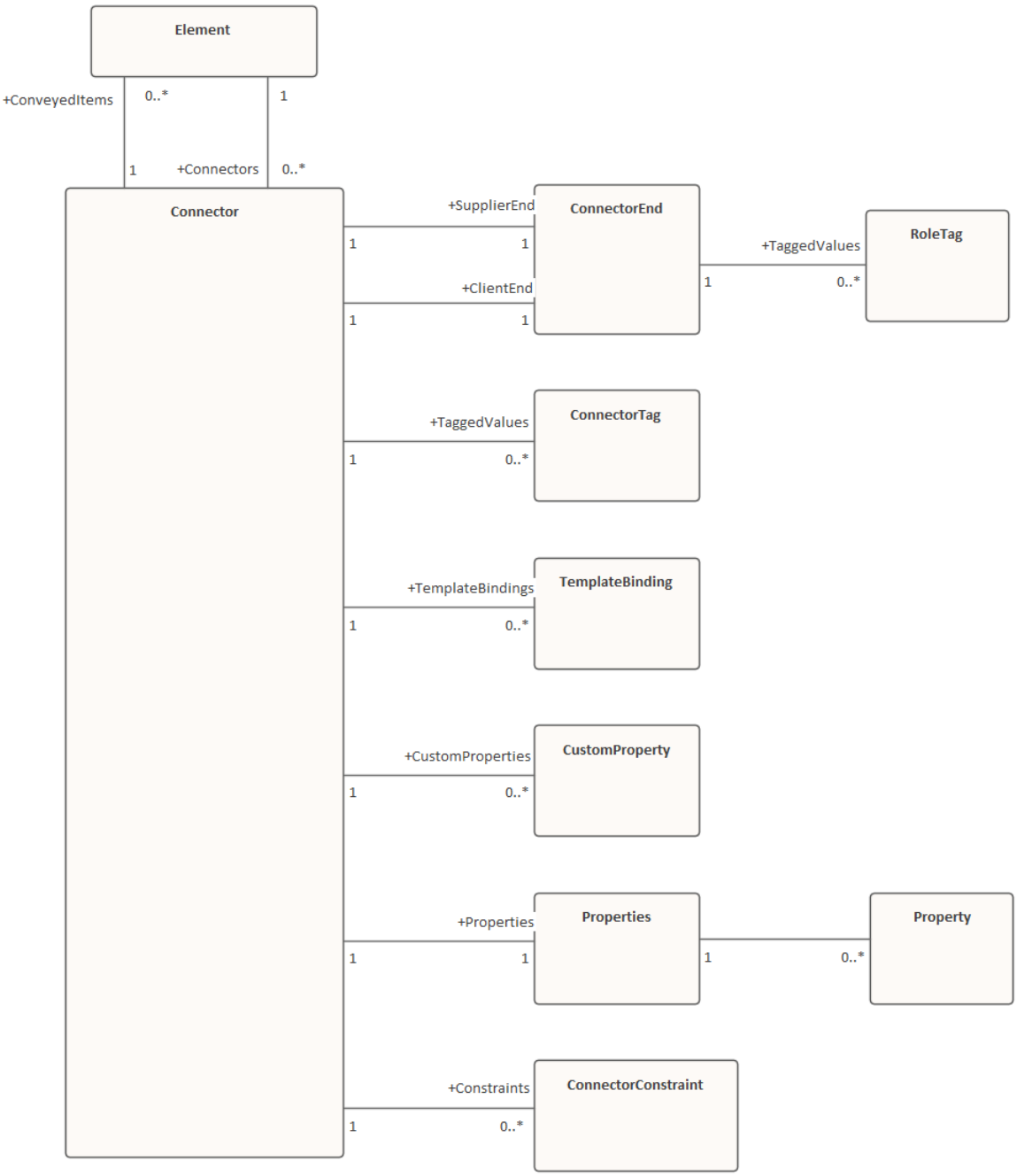
Transition Attributes

Attribute	Remarks
DurationConstraint	String Notes: Read/Write A constraint on the time duration of the transition.
Event	String Notes: Read/Write The event (optional) that initiated the transition.
Note	String Notes: Read/Write A free text note.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
TimeConstraint	String Notes: Read/Write A constraint on when the transition has to be completed.
TxState	String Notes: Read/Write The state to transition to, as defined in the 'Timeline Properties' dialog.
TxTime	String Notes: Read/Write. The time that the transition occurs. The value depends on a range set in the diagram.

Connector Package

The Connector Package details how connectors between elements are accessed and managed.

This diagram shows the Connector Class, its collections, and its relationships to the Element Class. Association Target roles correspond to member variable names in the source interface. The associated Classes represent the object type used in each collection.



Connector Class

To represent the various kinds of connectors between UML elements, you use a Connector object. You can access this from either the Client or Supplier element, using the Connectors collection of that element. When creating a new connector you assign to it a valid type from this list:

- Aggregation
- Assembly
- Association
- Collaboration
- CommunicationPath
- Connector
- ControlFlow
- Delegate
- Dependency
- Deployment
- ERLink
- Generalization
- InformationFlow
- Instantiation
- InterruptFlow
- Manifest
- Nesting
- NoteLink
- ObjectFlow
- Package
- Realization
- Sequence
- StateFlow
- TemplateBinding
- UseCase

Associated table in repository

t_connector

Connector Attributes

Attribute	Remarks
Alias	String Notes: Read/Write An optional alias for this connector.

AssociationClass	Element Notes: Read Only Returns the Association Class element if the connector has one; otherwise NULL/.
ClientEnd	ConnectorEnd Notes: Read Only A pointer to the ConnectorEnd object representing the source end of the relationship.
ClientID	Long Notes: Read/Write The ElementID of the element at the source end of this connector.
Color	Long Notes: Read/Write Sets the color of the connector.
ConnectorGUID	String Notes: Read Only A system generated, globally unique ID for the current connector.
ConnectorID	Long Notes: Read Only A system generated local identifier for the current connector.
Constraints	Collection Notes: Read Only A collection of constraint objects.
ConveyedItems	Collection of type Element Notes: Read Only Returns a collection of elements that have been conveyed. To add another element to the conveyed Collection, use 'AddNew (ElementGUID,NULL)', where 'ElementGUID' is the GUID of the element to be added.
CustomProperties	Collection Notes: Read Only Returns a collection of advanced properties associated with an element in the form of CustomProperty objects.
DiagramID	Long Notes: Read/Write The DiagramID of the connector.
Direction	String Notes: Read/Write The connector direction, which can be set to one of:

	• Unspecified • Bi-Directional • Source -> Destination or • Destination -> Source If the connector is non-navigable, set the 'sourceNavigability' and/or 'targetNavigability' attributes.
EndPointX	Long Notes: Read/Write The x-coordinate of the connector's end point. Connector end points are specified in Cartesian coordinates with the origin to the top left of the screen.
EndPointY	Long Notes: Read/Write The y-coordinate of the connector's end point. Connector end points are specified in Cartesian coordinates with the origin to the top left of the screen.
EventFlags	String Notes: Read/Write A structure to hold a variety of flags concerned with event signaling on messages.
FQStereotype	String Notes: Read Only The fully-qualified stereotype name in the format "Profile::Stereotype". One or more fully-qualified stereotype names can be assigned to StereotypeEx.
ForeignKeyInformation	String Notes: Read Only Returns the Foreign Key information.
IsLeaf	Boolean Notes: Read/Write A flag indicating that the connector is a leaf.
IsRoot	Boolean Notes: Read/Write A flag indicating that the connector is a root.
IsSpec	Boolean Notes: Read/Write A flag indicating that the connector is a specification.
MessageArguments	String Notes: Read Only The connector Message arguments.

MetaType	String Notes: Read Only The connector's domain-specific meta type, as defined by an applied stereotype from an MDG Technology.
MiscData	String Notes: Read Only This low-level property returns an array providing information about the contents of the PData x fields. These database fields are not documented and developers must gain understanding of these fields through their own endeavors to use this property. MiscData is zero based, therefore: • MiscData(0) corresponds to PData1 • MiscData(1) corresponds to PData2, and so on
Name	String Notes: Read/Write The connector name.
Notes	String Notes: Read/Write Descriptive notes about the connector.
ObjectType	ObjectType Notes: Read Only Distinguishes objects referenced through a Dispatch interface.
Properties	Properties Notes: Returns a list of specialized properties applicable to the connector that might not be available using the automation model. The properties are purposely undocumented because of their obscure nature and because they are subject to change as progressive enhancements are made to them.
ReturnValueAlias	String Notes: Shows the 'Return Value Alias' field of the operation.
RouteStyle	Long Notes: Read/Write The route style.
SequenceNo	Long Notes: Read/Write The SequenceNo of the connector.
StartPointX	Long Notes: Read/Write The x-coordinate of the connector's start point. Connector end points are specified in Cartesian coordinates with the origin to the

	top left of the screen.
StartPointY	Long Notes: Read/Write The y-coordinate of the connector's start point. Connector end points are specified in Cartesian coordinates with the origin to the top left of the screen.
StateFlags	String Notes: Read/Write A structure to hold a variety of flags concerned with State signaling on messages; the list is delimited by semi-colons.
Stereotype	String Notes: Read/Write Sets or gets the stereotype for this connector end.
StereotypeEx	String Notes: Read/Write All the applied stereotypes of the connector in a comma-separated list. Reading the value will provide the stereotype name only; assigning the value accepts either fully-qualified or simple names.
StyleEx	String Notes: Read/Write Advanced style settings; reserved for the use of Sparx Systems.
Subtype	String Notes: Read/Write A possible subtype to refine the meaning of the connector.
SupplierEnd	ConnectorEnd Notes: Read Only A pointer to the ConnectorEnd object representing the target end of the relationship.
SupplierID	Long Notes: Read/Write The ElementID of the element at the target end of this connector.
TaggedValues	Collection of type ConnectorTag Notes: Read Only The collection of ConnectorTag objects.
TemplateBindings	Collection of type TemplateBinding Notes: Read Only A collection of TemplateBinding objects.
TransitionAction	String

	Notes: Read/Write See the *Transition* topic for appropriate values.
TransitionEvent	String Notes: Read/Write See the *Transition* topic for appropriate values.
TransitionGuard	String Notes: Read/Write See the *Transition* topic for appropriate values.
Type	String Notes: Read/Write The connector type; valid types are held in the t_connectortypes table in the .eap file.
TypeInfoProperties	Notes: Read only Returns an interface pointer of TypeInfoProperties.
VirtualInheritance	String Notes: Read/Write For Generalization, indicates if the inheritance is virtual.
Width	Long Notes: Read/Write Specifies the width of the connector.

Connector Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
GetTXAlias (string Code, long Flag)	String Notes: Returns the Alias of the element for a given language. Parameters • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Flag: Long - 0 = Get the currently-stored translated Alias - 1 = Get the currently-stored translated Alias, and auto translate if the original Alias has changed - 2 = Always fetch the translated Alias from online
GetTXName (string Code,	String

long Flag)	Notes: Returns the name of the element for a given language. Parameters • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Flag: Long - 0 = Get the currently-stored translated name - 1 = Get the currently-stored translated name, and auto translate if the original name has changed - 2 = Always fetch the translated name from online
GetTXNote (string Code, long Flag)	String Returns the Notes of the element for a given language. Parameters • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Flag: Long - 0 = Get the currently-stored translated Notes - 1 = Get the currently-stored translated Notes, and auto translate if the original Notes have changed - 2 = Always fetch the translated Notes from online
IsConnectorValid()	Boolean Notes: Queries Enterprise Architect's internal relationship validation schema on the current connector. If False is returned, check the 'GetLastError()' function for more information.
SetTXAlias (string Code, string Translation)	String Notes - Set the translated Alias of the element for a given language. • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Translation: String - The translated Alias
SetTXName (string Code, string Translation)	String Notes - Set the translated name of the element for a given language. • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Translation: String - The translated name
SetTXNote (string Code, string Translation)	String Notes - Set the translated Notes of the element for a given language. • Code: String - Two-letter language code (found on the 'Translations' page of the 'Manage Model Options' dialog) • Translation: String - The translated Notes
Update()	Boolean Notes: Updates the current ConnectorObject after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

ConnectorConstraint Class

A ConnectorConstraint holds information about special conditions that apply to a connector. It is accessed through the Connector Constraints collection.

Associated table in repository

t_connectorconstraints

ConnectorConstraint Attributes

Attribute	Remarks
ConnectorID	Long Notes: Read/Write A local ID value (long) - system generated.
Name	String Notes: Read/Write The constraint name.
Notes	String Notes: Read/Write Notes about this constraint.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Type	String Notes: Read/Write The constraint type.

ConnectorConstraint Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Update the current ConnectorConstraint object after modification or

	appending a new item. If False is returned, check the 'GetLastError()' function for more information.

ConnectorEnd Class

A ConnectorEnd contains information about a single end of a connector. A ConnectorEnd is accessed from the connector as either the ClientEnd or SupplierEnd.

Associated table in repository

derived from t_connector

ConnectorEnd Attributes

Attribute	Remarks
Aggregation	Long Notes: Read/Write The type of Aggregation as it applies to this end; valid values are: 0 = None 1 = Shared 2 = Composite
Alias	String Notes: Read/Write An optional alias for this connector end.
AllowDuplicates	Boolean Notes: Read/Write For multiplicities greater than 1, indicates that duplicate entries are possible.
Cardinality	String Notes: Read/Write The cardinality associated with this end.
Constraint	String Notes: Read/Write A constraint that can be applied to this connector end.
Containment	String Notes: Read/Write The containment type applied to this connector end.
Derived	Boolean Notes: Read/Write Indicates that the value of this end is derived.
DerivedUnion	Boolean

	Notes: Read/Write Indicates the value of this role derived from the union of all roles that subset this.
End	String Notes: Read only The end this ConnectorEnd object applies to - Client or Supplier.
IsChangeable	String Notes: Read/Write Flag indicating whether this end is changeable or not - 'frozen', 'addOnly' or none.
IsNavigable	**Note: This property is not used** Boolean Notes: Read/Write A flag indicating this end is navigable from the other end.
Navigable	String Notes: Read/Write Indicates whether this role of an association is navigable from the opposite classifier - Navigable, Non-Navigable or Unspecified.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Ordering	Long Notes: Read/Write Ordering for this connector end.
OwnedByClassifier	Boolean Notes: Read/Write Indicates that this Association end corresponds to an attribute on the opposite end of the Association.
Qualifier	String Notes: Read/Write A qualifier that can apply to the connector end.
Role	String Notes: Read/Write The connector end role.
RoleNote	String Notes: Read/Write Notes associated with the role of this connector end.
RoleType	String Notes: Read/Write

	The role type applied to this end of the connector.
Stereotype	String Notes: Read/Write Sets or gets the stereotype for this connector end.
StereotypeEx	String Notes: Read/Write All the applied stereotypes of the connector end in a comma-separated list. Reading the value will provide the stereotype name only; assigning the value accepts either fully qualified or simple names.
TaggedValues	Collection of type RoleTag Notes: Read only A collection of RoleTag objects.
Visibility	String Notes: Read/Write The Scope associated with this connector end - Public, Private, Protected or Package.

ConnectorEnd Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Update the current ConnectorEnd object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

ConnectorTag Class

A ConnectorTag is a Tagged Value for a connector and is accessed through the Connector TaggedValues collection.

Associated table in repository

t_connectortag

ConnectorTag Attributes

Attribute	Remarks
ConnectorID	Long Notes: Read/Write The local ID of the associated connector.
FQName	String Notes: Read only The fully qualified name of the tag.
Name	String Notes: Read/Write The tag or name.
Notes	String Notes: Read/Write Further descriptive notes on this tag. If 'Value' is set to '<memo>', then 'Notes' should contain the actual Tagged Value content.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
TagGUID	String Notes: Read/Write A globally unique ID for this Tagged Value.
TagID	Long Notes: Read only A local ID to identify the Tagged Value.
Value	String Notes: Read/Write The value assigned to this tag.

	This field has a 255 character limit. If the value is greater than 255 characters long, set the value to "<memo>" and insert the body of text in the 'Notes' attribute. When reading existing Tagged Values, if 'Value' = "<memo>" then the developer should read the actual body of text from the 'Notes' attribute.

ConnectorTag Methods

Method	Remarks
GetAttribute(string propName)	String Notes: Returns the text of a single named property within a Structured Tagged Value.
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
HasAttributes()	Boolean Notes: Returns True if the Tagged Value is a Structured Tagged Value with one or more properties.
SetAttribute(string propName, string propValue)	Boolean Notes: Sets the text of a single named property within a Structured Tagged Value.
Update()	Boolean Notes: Update the current ConnectorTag object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

RoleTag Class

The RoleTag interface provides access to an Association's Role Tagged Values. Each connector end has a RoleTag collection that can be accessed to add, delete and access the RoleTags.

You might use this in creating code that resembles this fragment for accessing a RoleTag in VB.NET (where con is a Connector Object):

```
client = con.ClientEnd
client.Role = "m_client"
client.Update()
tag = client.TaggedValues.AddNew("tag", "value")
tag.Update()
tag = client.TaggedValues.AddNew("tag2", "value2")
tag.Update()
client.TaggedValues.Refresh()
For idx = 0 To client.TaggedValues.Count - 1
tag = client.TaggedValues.GetAt(idx)
Console.WriteLine(tag.Tag)
client.TaggedValues.DeleteAt(idx, False)
Next
tag = Nothing
```

Associated table in repository

t_taggedvalue

RoleTag Attributes

Attribute	Description
BaseClass	String Notes: Read/Write Indicates the role end; set to ASSOCIATION_SOURCE or ASSOCIATION_TARGET.
ElementGUID	String Notes: Read/Write The GUID of the connector with which this role tag is associated.
FQName	String Notes: Read only The fully qualified name of the tag.
ObjectType	ObjectType

	Notes: Read only Distinguishes objects referenced through a Dispatch interface.
PropertyGUID	String Notes: Read/Write A system generated GUID to identify the Tagged Value.
Tag	String Notes: Read/Write The actual tag name.
Value	String Notes: Read/Write The value associated with this tag.

RoleTag Methods

Method	Description
GetAttribute(string propName)	String Notes: Returns the text of a single named property within a Structured Tagged Value.
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
HasAttributes()	Boolean Notes: Returns True if the Tagged Value is a Structured Tagged Value with one or more properties.
SetAttribute(string propName, string propValue)	Boolean Notes: Sets the text of a single named property within a Structured Tagged Value.
Update()	Boolean Notes: Update the RoleTag after changes or on initial creation. If False is returned, check the 'GetLastError()' function for more information.

TemplateBinding Class

A TemplateBinding defines the connector between a binding Class and a parameterized Class, and the binding expression on that connector.

TemplateBinding Attributes

Attribute	Remarks
ActualGUID	String Notes: Read/Write The GUID of the element classifier set as the Actual Template Binding parameter. If the Actual Template Binding parameter is set as a string expression only, this will be an empty string. Assigning a GUID value will automatically change the ActualName attribute after Update() has been called.
ActualName	String Notes: Read/Write The name of the Actual Template Binding parameter. Assigning a new value will clear any current ActualGUID value.
BindingExpression	String Notes: Read only The Binding Expression as shown in Enterprise Architect.
ConnectorGUID	String Notes: Read only The Globally Unique ID of the associated connector.
ConnectorType	String Notes: Read only The type of the associated connector.
FormalName	String Notes: Read/Write The name of the Formal Template Binding parameter.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch Interface.
Pos	String Notes: Read only The position of the Template Binding in the list (as on the 'Bindings' page of the connector 'Properties' dialog).

TemplateBindingID	String Notes: Read only The Globally Unique ID of the current Template Binding.

TemplateBinding Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
Update()	Boolean Notes: Update the current TemplateBinding object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

Diagram Package

The Diagram Package has information on a diagram and on DiagramObject and DiagramLink, which are the instances of elements within a diagram.

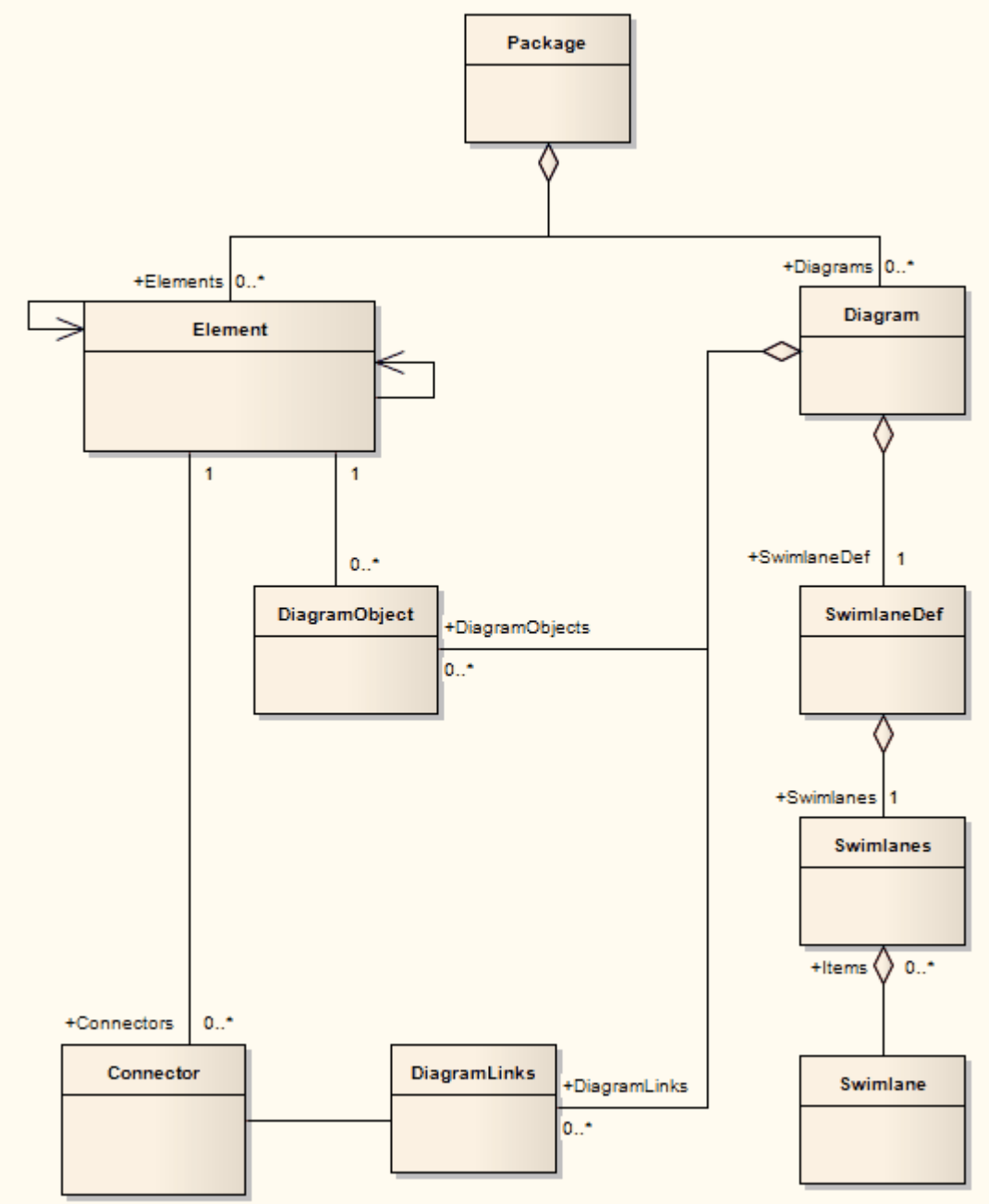


Diagram Class

A Diagram corresponds to a single UML diagram. It is accessed through the Package Diagrams collection and in turn contains a collection of diagram objects and diagram connectors. Adding to the DiagramObject Class adds an existing element to the diagram. When adding a new diagram, you must set the diagram type to one of the valid types:

- Activity
- Analysis
- Component
- Custom
- Deployment
- Logical
- Sequence
- Statechart
- Use Case

For a Collaboration (Communication) diagram, use the Analysis type.

Associated table in repository

t_diagram

Diagram Attributes

Attribute	Remarks
Author	String Notes: Read/Write The name of the author.
CreatedDate	Date Notes: Read/Write The date the diagram was created.
cx	Long Notes: Read/Write The X dimension of the diagram (the default is 800).
cy	Long Notes: Read/Write The Y dimension of the diagram (the default is 1100).
DiagramGUID	Variant Notes: Read/Write A globally unique ID for this diagram.

DiagramID	Long Notes: Read only A local ID for the diagram.
DiagramLinks	Collection Notes: Read only A list of DiagramLink objects, each containing information about the display characteristics of a connector in a diagram.
DiagramObjects	Collection Notes: Read only A collection of references to DiagramObjects. A DiagramObject is an instance of an element in a diagram, and includes size and display characteristics.
ExtendedStyle	String Notes: Read/Write An extended style attribute.
FilterElements	String Notes: Read/Write Applies a comma-separated list of object ids (from SelectedObjects) to the currently-applied diagram filter, overriding the filter. The effect persists until another filter is applied, or the diagram is closed.
HighlightImports	Boolean Notes: Read/Write A flag to indicate that elements from other Packages should be highlighted. Corresponds with the 'Show Namespace' option in the diagram 'Properties' dialog.
IsLocked	Boolean Notes: Read/Write A flag indicating whether this diagram is locked or not.
MetaType	String Notes: Read/Write The diagram's domain-specific meta type, as defined by an MDG Technology. When writing, the meta type must be fully qualified and from an existing profile.
ModifiedDate	Variant Notes: Read/Write The date the diagram was last modified.
Name	String Notes: Read/Write The diagram name.
Notes	String Notes: Read/Write Set or retrieve notes for this diagram.

ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Orientation	String Notes: Read/Write The page orientation: P for Portrait or L for Landscape.
PackageID	Long Notes: Read/Write The ID of the Package that this diagram belongs to.
PageHeight	Long Notes: Read The number of pages high the diagram is.
PageWidth	Long Notes: Read The number of pages wide the diagram is.
ParentID	Long Notes: Read/Write The optional ID of an element that 'owns' this diagram; for example, a Sequence diagram owned by a Use Case.
Scale	Long Notes: Read/Write The zoom scale (the default is 100).
SelectedConnector	Connector Notes: Read/Write The currently selected connector on this diagram. Null if there is no currently selected diagram.
SelectedObjects	Collection Notes: Read only Gets a collection representing the currently selected elements on the diagram. You can remove objects from this collection to deselect them, and add elements to the collection by passing the Object ID as a name to select them.
ShowDetails	Long Notes: Read/Write A flag to indicate that the Diagram Details text should be shown: 1 = Show, 0 = Hide.
ShowPackageContents	Boolean Notes: Read/Write A flag to indicate that the Package contents should be shown in the current

	diagram.
ShowPrivate	Boolean Notes: Read/Write A flag to show or hide Private features.
ShowProtected	Boolean Notes: Read/Write A flag to show or hide Protected features.
ShowPublic	Boolean Notes: Read/Write A flag to show or hide Public features.
Stereotype	String Notes: Read/Write Sets or gets the stereotype for this diagram.
StyleEx	String Notes: Read/Write Advanced style settings, reserved for the use of Sparx Systems.
Swimlanes	String Notes: Read/Write Information on swimlanes contained in the diagram. Please note that this property is superseded by SwimlaneDef.
SwimlaneDef	SwimlaneDef Notes: Read/Write Information on swimlanes contained in the diagram.
Type	String Notes: Read only The diagram type; see the t_diagramtypes table in the .eap file for more information.
Version	String Notes: Read/Write The version of the diagram.

Diagram Methods

Method	Details
ApplyGroupLock (string	Boolean

aGroupName)	Notes: Applies a group lock to this diagram object, for the specified group, on behalf of the current user. Returns True if the operation is successful; returns False if the operation is unsuccessful. Use GetLastError() to retrieve error information. Parameter: • aGroupName: String - the name of the user group for which to set the group lock
ApplyUserLock ()	Boolean Notes: Applies a user lock to this diagram object, for the current user. Returns True if the operation is successful; returns False if the operation is unsuccessful. Use GetLastError() to retrieve error information.
FindElementInDiagram (long ElementID)	Boolean Notes: This function activates the Diagram View and displays the diagram with the diagram object selected. If the diagram is too large to display all of it on the screen, the portion of the diagram containing the object is displayed with the object shown in the center of the screen. Diagram objects flagged as non-selected are shown but are not selected Returns True if the diagram object was found, the diagram displayed and the object selected (or at least displayed) in the view. Returns False if the diagram object was not found in the diagram and the diagram not displayed. Parameter • ElementID: Long - the element ID of the diagram object to locate
GetDiagramObjectByID (long ID, string DUID)	DiagramObject Notes: Returns the DiagramObject object, if it exists on the diagram. Parameters: • ID: Long - the ElementID of the diagram object • DUID: String - the optional Diagram Unique ID of the diagram object
GetElementByGrid (string GridX, string GridY)	Element Notes: Uses the Excel type of format to specify the column and row of a grid at which an element should be found: A 5, CB 1. Returns null if no element is at the specified position. Parameters: • GridX: string - Column A to Z • GridY: string - Number of row
GetElementByName (string Name)	Element Notes: Locates an element with the specified name. Returns null if no element is found with that name. Parameters: • name: string - Name of the element to find
GetObjectByGrid (string GridX, string GridY)	DiagramObject Notes: Uses the Excel type of format to specify the column and row of a grid at which an object should be found: A 5, CB 1. Returns null if no element is at the specified position.

	Parameters: • GridX: string - Column A to Z • GridY: string - Number of row
GetLastError ()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
ReadStyle (string StyleName)	String Notes: Returns the current value of the named diagram style. Use GetLastError() to retrieve error information. Parameters: • StyleName: String - the name of the diagram style whose value is to be retrieved; valid StyleNames are: - Show Element Property String - Show Connector Property String - Show Feature Property String
ReleaseUserLock ()	Boolean Notes: Releases a group lock or user lock on this diagram object. Returns True if the operation is successful; returns False if the operation is unsuccessful. Use GetLastError() to retrieve error information.
ReorderMessages ()	Void Notes: Resets the display order of Sequence and Collaboration messages. This is typically used after inserting or deleting messages in the diagram.
SaveAsPDF (string FileName)	Boolean Notes: Exports the diagram to a PDF document. Returns True on success. Parameters: • FileName: String - full path to file location
SaveImagePage(long x, long y, long sizeX, long sizeY, string filename, long flags)	Boolean Notes: Saves a page of the diagram to disk. Returns True if the operation is successful; returns False if the operation is unsuccessful. Use GetLastError() to retrieve error information. Parameters: • x: Long - the horizontal page • y: Long - the vertical page • sizeX: Long - currently unused; pass a value of 0 to ensure behavior does not change in a future build • sizeY: Long - currently unused; pass a value of 0 to ensure behavior does not change in a future build • filename: String - the filename and path to save the image • flags: Long - additional options, currently unused; pass a value of 0 to ensure behavior does not change in a future build The image type is determined by the extension of the filename. Currently only .emf, .bmp and .png formats are supported.

ShowAsElementList (bool ShowAsList, bool Persist)	Boolean Notes: Toggles the diagram display between diagram format and Diagram List depending on the value of ShowAsList. If Persist is set, the display format is written to the database so the diagram always opens in that format (diagram or list). Otherwise, the display format falls back to the default (diagram) once the display is closed. Parameters: • ShowAsList: Boolean - indicates diagram or Diagram List • Persist: Boolean - indicates set (maintain ShowAsList value) or not (revert to default)
Update ()	Boolean Notes: Updates this diagram object after modification or appending a new item. If False is returned, use GetLastError() to retrieve error information.

VirtualizeConnector (int ConnectorID, int Action, int X, int Y)	Boolean
	Notes: Creates a virtual copy of the source or target element on a connector, and sets its location on the diagram as a waypoint on the connector. If the source element is being virtualized, the waypoint is created as the first on the connector, and if the target element is being virtualized, the waypoint is created as the last on the connector.
	If called again on the same connector, removes the virtual element. However, the waypoint remains in place.
	As waypoints and therefore virtual elements can only be created on connectors with the Custom line-style, if the connector does not have this line style the method sets it. So, after this method executes, an Update function should be called for the connector as well as for the diagram. All parameters are required for the function to complete successfully.
	Returns True if the operation is successful; returns False if the operation is unsuccessful.
	Parameters:
	• ConnectorID - Integer: the ID of the connector on which to create the virtual element
	• Action - Integer: the element to be virtualized; 1 for the source element, 2 for the target element
	• X - Integer: the position on the X axis that the element's center point will be aligned with
	• Y - Integer: the position on the Y axis that the element's center point will be aligned with
	For example, to virtualize the source element of the selected connector:
	function main()
	{
	var diagram as EA.Diagram;
	var conn as EA.Connector;
	diagram = Repository.GetCurrentDiagram();
	if(diagram != null)
	{
	var connector as EA.Connector.
	connector = diagram.SelectedConnector;

	diagram.VirtualizeConnector(connector.ConnectorID, 1, 100, 150);
	connector.Update();
	diagram.Update();
	Repository.ReloadDiagram(diagram.DiagramID);
	}
	else
	{
	Session.Output("Script requires a diagram to be visible");
	}
	}
	main();

WriteStyle (string StyleName, string StyleValue)	Void Notes: Sets the value of the named diagram style. Use GetLastError() to retrieve error information. Parameters: • StyleName: String - the name of the diagram style whose value is to be retrieved; valid StyleNames are: - Show Element Property String - Show Connector Property String - Show Feature Property String • StyleValue: String - the value to be set in the named diagram style; valid values for the StyleNames listed are **0** and **1**

DiagramLink Class

A DiagramLink is an object that holds display information on a connector between two elements in a specific diagram. It includes, for example, the custom points and display appearance. It can be accessed from the Diagram DiagramLinks collection.

Associated table in repository

t_diagramlinks

DiagramLink Attributes

Attribute	Remarks
ConnectorID	Long Notes: Read/Write The ID of the associated connector.
DiagramID	Long Notes: Read/Write The local ID for the associated diagram.
Geometry	String Notes: Read/Write The geometry associated with the current connector in this diagram.
HiddenLabels	Boolean Notes: Indicates if this connector's labels are hidden on the diagram.
InstanceID	Long Notes: Read only The connector identifier for the current model.
IsHidden	Boolean Notes: Read/Write Indicates if this item is hidden or not.
LineColor	Long Notes: Sets the line color of the connector. Set to -1 to reset to the default color in the model.
LineStyle	Long Notes: Sets the line style of the connector. 1 = Direct 2 = Auto Routing

	3 = Custom Line 4 = Tree Vertical 5 = Tree Horizontal 6 = Lateral Vertical 7 = Lateral Horizontal 8 = Orthogonal Square 9 = Orthogonal Rounded
LineWidth	Long Notes: Sets the line width of the connector.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Path	String Notes: Read/Write The path of the connector in this diagram.
SourceInstanceUID	String Notes: Read only Returns the Unique Identifier of the source object.
SuppressSegment	Long Notes: Read/Write Returns the index of the line segment that has been suppressed. Returns 0 when no segments are suppressed.
Style	String Notes: Read/Write Additional style information; for example, color or thickness.
TargetInstanceUID	String Notes: Read only Returns the Unique Identifier of the target object.

DiagramLink Methods

Method	Remarks
GetLastError()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used as an exception is thrown when an error occurs.

Update()	Boolean Notes: Update the current DiagramLink object after modification or appending a new item. If False is returned, check the 'GetLastError()' function for more information.

DiagramObject Class

The DiagramObject Class stores presentation information that indicates what is displayed in a diagram and how it is shown.

Associated Table in Repository

t_diagramobjects

DiagramObject Attributes

Attribute	Remarks
BackgroundColor	Long Notes: The background color of the object on the diagram. Set to -1 to re-set to the default color in the model.
BorderColor	Long Notes: The border line color of the object on the diagram. Set to -1 to re-set to the default color in the model.
BorderLineWidth	Long Notes: The border line width of the object on the diagram. Valid values are 1 (narrowest) to 5 (thickest); a default of 1 is applied if an invalid value is passed in.
Bottom	Long Notes: Read/Write The bottom edge position of the object on the diagram. Enterprise Architect uses a cartesian coordinate system, with {0,0} being the top-left corner of the diagram. For this reason, Y-axis values (Top and Bottom) should always be negative.
DiagramID	Long Notes: Read/Write The ID of the associated diagram.
ElementDisplayMode	Long Notes: Indicates how to adjust the element features if the element is resized. 1 = Resize to longest feature 2 = Wrap features 3 = Truncate features Defaults to 1 if an invalid value is supplied.
ElementID	Long Notes: Read/Write

	The ElementID of the object instance in this diagram.
FeatureStereotypesTo Hide	String Notes: Lists the stereotypes to hide on the object on the diagram.
FontBold	Boolean Notes: Get or Set the status of the object text font as Bold.
FontColor	Long Notes: The color of the font of the object text on the diagram.
FontItalic	Boolean Notes: Get or Set the status of the object text font as Italic.
FontName	String Notes: The name of the font used for the object text.
FontSize	String Notes: The size of the font used for the object text.
FontUnderline	Boolean Notes: Get or Set the status of the object text font as Underlined.
InstanceGUID	String Notes: The instance GUID for the object on the diagram (the DUID).
InstanceID	Long Notes: Read Holds the connector identifier for the current model.
IsSelectable	Boolean Notes: Indicates whether this object on the diagram can be selected.
Left	Long Notes: Read/Write The left edge position of the object on the diagram.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Right	Long Notes: Read/Write The right edge position of the object on the diagram.
Sequence	Long Notes: Read/Write The sequence position when loading the object into the diagram (this affects its Z

	order). The Z-order is one-based and the lowest value is in the foreground.
ShowComposedDiagram	Boolean Notes: Indicates whether the object's composite diagram should be displayed by default when the object is selected.
ShowConstraints	Boolean Notes: Show constraints for this object on the diagram.
ShowFormattedNotes	Boolean Notes: Show any formatting applied to the notes, for this object on the diagram. ShowNotes must be True for the formatted notes to be displayed.
ShowFullyQualifiedTags	Boolean Notes: Show fully qualified Tagged Values for this object on the diagram.
ShowInheritedAttributes	Boolean Notes: Show inherited attributes for this object on the diagram.
ShowInheritedConstraints	Boolean Notes: Show inherited constraints for this object on the diagram.
ShowInheritedOperations	Boolean Notes: Show inherited operations for this object on the diagram.
ShowInheritedResponsibilities	Boolean Notes: Show the inherited requirements within the Requirements compartment for this object on the diagram.
ShowInheritedTags	Boolean Notes: Show inherited Tagged Values for this object on the diagram.
ShowNotes	Boolean Note: Show the notes for this object on the diagram.
ShowPackageAttributes	Boolean Notes: Show Package attributes for this object on the diagram.
ShowPackageOperations	Boolean Notes: Show Package operations for this object on the diagram.
ShowPortType	Boolean Notes: Show the Port type.
ShowPrivateAttributes	Boolean Notes: Show private attributes for this object on the diagram.
ShowPrivateOperations	Boolean

	Notes: Show private operations for this object on the diagram.
ShowProtectedAttributes	Boolean Notes: Show protected attributes for this object on the diagram.
ShowProtectedOperations	Boolean Notes: Show protected operations for this object on the diagram.
ShowPublicAttributes	Boolean Notes: Show public attributes for this object on the diagram.
ShowPublicOperations	Boolean Notes: Show public operations for this object on the diagram.
ShowResponsibilities	Boolean Notes: Show the requirements compartment for this object on the diagram.
ShowRunstates	Boolean Notes: Show Runstates for this object on the diagram.
ShowStructuredCompartments	Boolean Note: Indicates whether to display the Structure Compartments for this object on the diagram.
ShowTags	Boolean Notes: Show Tagged Values for this object on the diagram.
Style	Variant Notes: Read/Write The style information for this object. Returns a semi-colon delimited string that defines the current style settings. Changing a value will completely overwrite the previously existing value, so caution is advised to avoid losing existing style information that you want to keep. See *Setting the Style*.
TextAlign	Long Notes: Indicates the alignment of text on a Text element on the diagram. 1 = Left aligned 2 = Center aligned 3 = Right aligned Defaults to 1 if an invalid value is supplied.
Top	Long Notes: Read/Write The top edge position of the object on the diagram. Enterprise Architect uses a cartesian coordinate system, with {0,0} being the top-left corner of the diagram. For this reason, Y-axis values (Top and Bottom) should always be negative.

DiagramObject Methods

Method	Remarks
GetLastError ()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
MoveElementToGridPosition (GridX, GridY)	Notes: Currently not implemented.
ResetFont	Notes: Resets the font of the object text on the diagram back to the model default.
SetFontStyle (FontName, FontSize, Bold, Italic, Underline)	Boolean Notes: Sets the font of the object text on the diagram to the specified values.
SetStyleEx (string Parameter, string Value)	Void Notes: Sets an individual parameter of the Style string. Parameters: • Parameter: String - the name of the style parameter to modify; for example: "BCol" = background color "BFol" = font color "LCol" = line color "LWth" = line width • Value: String - the new value for the style parameter
Update ()	Boolean Notes: Updates the current DiagramObject after modification or appending a new item If False is returned, check the GetLastError function for more information.

Setting the Style

The Style attribute contains various settings that affect the appearance of a DiagramObject. However, it is not recommended to directly edit this attribute string. Instead, use either the SetStyleEx method or one of the individual DiagramObject attributes such as BackgroundColor, FontColor or BorderColor.

For example, the Style string might contain a series of values in a format such as:

BCol=n;BFol=n;LCol=n;LWth=n;

where:

- BCol = Background Color
- BFol = Font Color
- LCol = Line Color
- LWth = Line Width

The value assigned to each of the Style color properties is a decimal representation of the hex RGB value, where Red=FF, Green=FF00 and Blue=FF0000.

This code snippet shows how you might change the style settings for all of the objects in the current diagram, changing the background color to red (FF=255) and the font and line colors to yellow (FFFF=65535):

```
For Each aDiagObj In aDiag.DiagramObjects
    aDiagObj.BackgroundColor=255
    aDiagObj.FontColor=65535
    aDiagObj.BorderColor=65535
    aDiagObj.BorderLineWidth=1
    aDiagObj.Update
    aRepos.ReloadDiagram aDiagObj.DiagramID
Next
```

SwimlaneDef Class

A SwimlaneDef object makes available attributes relating to a single row or column in a list of swimlanes.

SwimlaneDef Attributes

Attribute	Description
Bold	Boolean Notes: Read/Write Show the title text in bold.
FontColor	Long Notes: Read/Write The RGB color used to draw the titles.
HideClassifier	Boolean Notes: Read/Write Removes any classifier from the title display.
HideNames	Boolean Notes: Read/Write Set to True to hide the swimlane titles.
LineColor	Long Notes: Read/Write The RGB color used to draw swimlane borders.
LineWidth	Long Notes: Read/Write The width, in pixels, of the line used to draw swimlanes. Valid values are 1, 2 or 3.
Locked	Boolean Notes: Read/Write If set to True, disables user modification of the swimlanes via the diagram.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Orientation	String Notes: Read/Write Indicates whether the swimlanes are vertical or horizontal.
ShowInTitleBar	Boolean Notes: Read/Write

	Enables vertical swimlane titles to be shown in the title bar.
Swimlanes	Swimlanes Notes: Read/Write A list of individual swimlanes.

Swimlanes Class

A Swimlanes object is attached to a diagram's SwimlaneDef object and provides a mechanism to access individual swimlanes.

Swimlanes Attributes

Attribute	Description
Count	Long Notes: Read/Write Gives the number of swimlanes.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.

Swimlanes Methods

Method	Description
Add(string Title, long Width)	Swimlane Notes: Adds a new swimlane to the end of the list, and returns a swimlane object representing the newly added entry. Parameters: • Title: String - The title text that appears at the top of the swimlane; this can be the same as an existing swimlane title • Width: Long - The width of the swimlane in pixels
Delete(object Index)	Void Notes: Deletes a selected swimlane. If the string matches more than one entry, only the first entry is deleted. Parameter: • Index: Object - Either a string representing the title text or an integer representing the zero-based index of the swimlane to delete
DeleteAll()	Void Notes: Removes all swimlanes.
Insert(long Index, string Title, long Width)	Swimlane Notes: Inserts a swimlane at a specific position, and returns a swimlane object representing the newly added entry. Parameters: • Index: Long - The zero-based index of the existing Swimlane before which this

	new entry is inserted • Title: String - The title text that appears at the top of the swimlane; this can be the same as an existing swimlane title • Width: Long - The width of the swimlane in pixels
Items(object Index)	Swimlane collection Notes: Accesses an individual swimlane. If the string matches more than one swimlane title, the first matching swimlane is returned. Parameter: • Index: Object - Either a string representing the title text or an integer representing the zero-based index of the swimlane to get

Swimlane Class

A Swimlane object makes available attributes relating to a single row or column in a list of swimlanes.

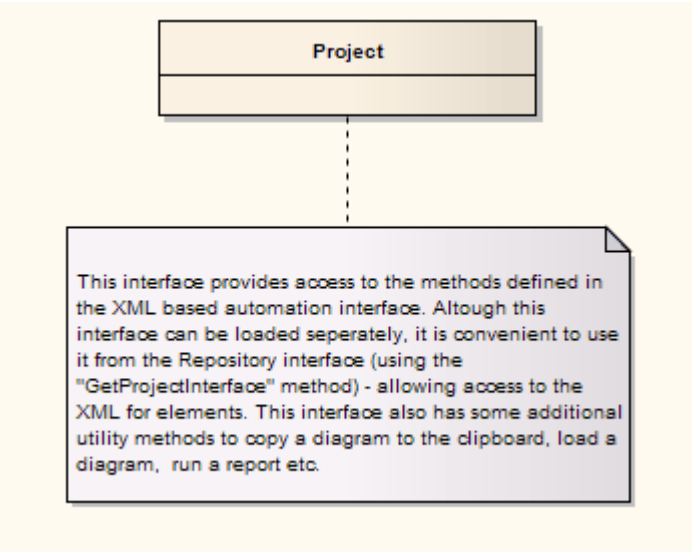
Swimlane Attributes

Attribute	Description
BackColor	Long Notes: Read/Write The RGB color that the swimlane is filled with.
ClassifiedGuid	String Notes: Read/Write The GUID of the classifier Class. This can be obtained from the corresponding element object via the ElementGUID property.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Title	String Notes: Read/Write The text at the head of the swimlane.
Width	Long Notes: Read/Write The width of the swimlane, in pixels.

Project Interface Package

The Enterprise Architect.Project interface. This is the interface to Enterprise Architect elements; it also includes some utility functions. You can get a pointer to this interface using the Repository.GetProjectInterface method.

Example



Project Class

The Project interface can be accessed from the Repository using GetProjectInterface(). The returned interface provides access to the XML-based Enterprise Architect Automation Interface. Use this interface to get XML for the various internal elements and to run some utility functions to perform tasks such as load diagrams or run reports.

Project Attributes

Attribute	Remarks
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.

Notes

- The Project methods listed here all require input GUIDs in XML format; use **GUIDtoXML** to change the Enterprise Architect GUID to an XML GUID

Project Methods

Method	Remarks
BuildExecutableStatemachine (string ElementGUID, string ExtraOptions)	Boolean Notes: Builds Executable StateMachine code for an <<executable statemachine>> Artifact element. Parameters: • ElementGUID: String - the GUID (in XML format) of the element to generate • ExtraOptions: String - enables extra options to be given to the command (currently unused)
CancelValidation ()	Void Notes: Cancels a validation process.
CanValidate ()	Boolean Notes: Returns a value to indicate that the Model Validation component is loaded.
CreateBaseline (string PackageGUID, string Version, string Notes)	Boolean Notes: Creates a Baseline of a specified Package. Parameters: • PackageGUID: String - the GUID (in XML format) of the Package to Baseline • Version: String - the version of the Baseline • Notes: String - any notes concerning the Baseline

CreateBaselineEx (string PackageGUID, string Version, string Notes, EA.CreateBaselineFlag Flags)	Boolean Notes: Creates a Baseline of a specified Package, with a flag to exclude Package contents below the first level. Parameters: • PackageGUID: String - the GUID (in XML format) of the Package to be Baselined • Version: String - the version of the Baseline • Notes: String - any notes concerning the Baseline • Flags: EA.CreateBaselineFlag - whether or not to exclude the Package contents below the first level
DefineRule (string CategoryID, EA.EnumMVErrorType ErrorType, string ErrorMessage)	String Notes: Defines the individual rules that can be performed during model validation. It must be called once for each rule from the EA_OnInitializeUserRules broadcast handler. The return value is a RuleId, which can be used for reference purposes when an individual rule is executed by Enterprise Architect during model validation. See the *Model Validation Example* for a detailed example of the use of this method. Parameters: • CategoryId: String - should be passed the return value from the DefineRuleCategory method • ErrorType: EA.EnumMVErrorType - depending on the severity of the error being validated, can be: - mvErrorCritical - mvError - mvWarning, or - mvInformation • ErrorMessage: String - can contain a default error string, although this is probably overridden by the PublishResult call
DefineRuleCategory (string CategoryName)	String Notes: Defines a category of rules that can be performed during model validation (there is typically one category per Add-In). It must be called once from the EA_OnInitializeUserRules broadcast handler. The return value is a CategoryId that must to be passed to the DefineRule method. See the *Model Validation Example* for a detailed example of the use of this method. Parameters: • CategoryName: String - a text string that is visible in the 'Model Validation Configuration' dialog
DeleteBaseline (string BaselineGUID)	Boolean Notes: Deletes a Baseline, identified by the BaselineGUID, from the repository. If the repository is configured to store Baselines in a Reusable Asset Service Registry, then it is not possible to delete the Baseline and a False value is returned. Parameters: • BaselineGUID: String - the GUID (in XML format) of the Baseline to delete
DoBaselineCompare (string PackageGUID, string Baseline, string	String Notes: Performs a Baseline comparison using the supplied Package GUID and Baseline GUID (obtained in the result list from GetBaselines).

ConnectString)	Optionally you can include the connection string required to find the Baseline if it exists in a different model file. This method returns a log file of the status of all elements found and compared in the difference procedure. You can use this log information as input to DoBaselineMerge - automatically merging information from the Baseline. Parameters: • PackageGUID: String - the GUID (in XML format) of the Package to run the comparison on • Baseline: String - the GUID (in XML format) of the Baseline to run the comparison on • ConnectString: String - not currently used
DoBaselineMerge (string PackageGUID, string Baseline, string MergeInstructions, string ConnectString)	String Notes: Performs a batch merge based on instructions contained in an XML file (MergeInstructions). You can supply an optional connection string if the Baseline is located in another model. In the MergeInstructions file, each MergeItem node supplies the GUID of a differenced item from the XML difference log. As the merge is uni-directional and actioned in only one possible way, no additional arguments are required. Enterprise Architect chooses the correct procedure based on the 'Difference' results. <Merge> <MergeItem guid="{XXXXXX}" /> <MergeItem guid="{XXXXXX}" /> </Merge> Alternatively, you can supply a single Mergeitem with a GUID of RestoreAll. In this case, Enterprise Architect batch-processes ALL differences. <Merge> <MergeItem guid="RestoreAll" changed="true" baselineOnly="true" modelOnly="true" moved="true" fullRestore="false" /> </Merge> Parameters: • PackageGUID: String - the GUID (in XML format) of the Package to merge the Baseline into • Baseline: String - the GUID of the Baseline (in XML format) to merge into the Package • MergeInstructions: String - the file containing the GUID of each differenced item from the XML difference log returned by DoBaselineCompare() • ConnectString: String - not currently used
EnumDiagramElements (string DiagramGUID)	protected abstract: String Notes: Gets an XML list of all elements in a diagram. Parameters: • DiagramGUID: String - the GUID (in XML format) of the diagram to get elements for
EnumDiagrams (string PackageGUID)	protected abstract: String Notes: Gets an XML list of all diagrams in a specified Package. Parameters: • PackageGUID: String - the GUID (in XML format) of the Package to list diagrams for

EnumElements (string PackageGUID)	protected abstract: String Notes: Gets an XML list of elements in a specified Package. Parameters: • PackageGUID: String - the GUID (in XML format) of the Package to get a list of elements for
EnumLinks (string ElementGUID)	protected abstract: String Notes: Gets an XML list of connectors for a specified element. Parameters: • ElementGUID: String - the GUID (in XML format) of the element to get all associated connectors for
EnumPackages (string PackageGUID)	protected abstract: String Notes: Gets an XML list of child Packages inside a parent Package. Parameters: • PackageGUID: String - the GUID (in XML format) of the parent Package
EnumProjects ()	protected abstract: String Notes: Gets a list of projects in the current file; corresponds to Models in Repository.
EnumViewEx (string ProjectGUID)	protected abstract: String Notes: Gets a list of Views in the current project. Parameters: • ProjectGUID: String - the GUID (in XML format) of the project to get views for
EnumViews ()	protected abstract: String Notes: Enumerates the Views for a project. Returned as an XML document.
Exit ()	protected abstract: String Notes: Exits the current instance of Enterprise Architect; this function is maintained for backward compatibility and should never be called. Enterprise Architect automatically exits when you are no longer using any of the provided objects.
ExportPackageXMI (string PackageGUID, enumXMIType XMIType, long DiagramXML, long DiagramImage, long FormatXML, long UseDTD, string FileName)	protected abstract: String Notes: Exports XMI for a specified Package. Parameters: • PackageGUID: String - the GUID (in XML format) of the Package to be exported • XMIType: EnumXMIType - specifies the XMI type and version information; see *XMIType Enum* for accepted values • DiagramXML: Long - True if XML for diagrams is required; accepted values: 0 = Do not export diagrams 1 = Export diagrams 2 = Export diagrams along with alternative images • DiagramImage: Long - the format for diagram images to be created at the same time; accepted values:

	-1 = NONE 0 = EMF 1 = BMP 2 = GIF 3 = PNG 4 = JPG • FormatXML: Long - True if XML output should be formatted prior to saving • UseDTD: Long - True if a DTD should be used • FileName: String - the filename to output to
ExportPackageXMIEx (string PackageGUID, enumXMIType XMIType, long DiagramXML, long DiagramImage, long FormatXML, long UseDTD, string FileName, ea.ExportPackageXMIFlag Flags)	protected abstract: String Notes: Exports XMI for a specified Package, with a flag to determine whether the export includes Package content below the first level. Parameters: • PackageGUID: String - the GUID (in XML format) of the Package to be exported • XMIType: EnumXMIType - specifies the XMI type and version information; see *XMIType Enum* for accepted values • DiagramXML: Long - true if XML for diagrams is required; accepted values: 0 = Do not export diagrams 1 = Export diagrams 2 = Export diagrams along with alternative images • DiagramImage: Long - the format for diagram images to be created at the same time; accepted values: -1 =NONE 0 =EMF 1 =BMP 2 =GIF 3 =PNG 4 =JPG • FormatXML: Long - True if XML output should be formatted prior to saving • UseDTD: Long - True if a DTD should be used. • FileName: String - the filename to output to • Flags: ea.ExportPackageXMIFlag - specify whether or not to include Package content below the first level (currently supported for xmiEADefault), whether or not to exclude tool-specific information from export
ExportProjectXML (string DirectoryPath)	Boolean Notes: Exports the entire current project to Native XML files in the specified directory. The contents of the directory will be deleted prior to exporting the project data Parameters: • DirectoryPath: String - directory path to save the exported Native XML files
ExportReferenceData (string FileName, string Tables)	Boolean Notes: Exports Reference Data. Parameters: • FileName: String - the name of the file to output the reference data to • Tables: String - the list of reference data tables to be output; the data table delimeter is ";" If the string is empty, Enterprise Architect will prompt with a dialog to select the tables to output

GenerateBuildRunExecutableStateMachine (string ElementGUID, string ExtraOptions)	Boolean Notes: Generates, builds and runs Executable StateMachine code for an <<executable statemachine>> Artifact element, which will start simulation of the StateMachine. Parameters: • ElementGUID: String - the GUID (in XML format) of the element to generate • ExtraOptions: String - enables extra options to be given to the command (currently unused)
GenerateClass (string ElementGUID, string ExtraOptions)	Boolean Notes: Generates the code for a single Class. Parameters: • ElementGUID: String - the GUID (in XML format) of the element to generate • ExtraOptions: String - enables extra options to be given to the command; currently unused
GenerateDiagramFromScenario (string ElementGUID, EnumScenarioDiagramType DiagramType, long OverwriteExistingDiagram)	Boolean Notes: Generates various diagrams from the scenario specification of an element. Parameters: • ElementGUID: String - the GUID (in XML format) of the element containing the scenario specification • DiagramType: EnumScenarioDiagramType - the type of diagram to generate; see ScenarioDiagramType Enum for accepted values • OverwriteExistingDiagram: Long - determines whether to overwrite the existing diagram or synchronize the existing elements with the scenario steps 0 = Delete the existing diagram and elements, and create a new diagram and elements 1 = Synchronize existing elements with the scenario steps and preserve the diagram layout 2 = Synchronize existing elements with the scenario steps and re-cast the diagram layout 3 = Do not generate a diagram if one already exists
GenerateElementDDL (string ElementGUID, string FileName, string ExtraOptions)	Boolean Notes: Generates DDL for an element using the options that are currently set on the Generate DDL screen.
GenerateExecutableStatemachine (string ElementGUID, string ExtraOptions)	Boolean Notes: Generates Executable StateMachine code for an <<executable statemachine>> Artifact element. Parameters: • ElementGUID: String - the GUID (in XML format) of the element to generate • ExtraOptions: String - enables extra options to be given to the command (currently unused)
GeneratePackage (string PackageGUID, string ExtraOptions)	Boolean Notes: Generates the code for all Classes within a Package. For example: recurse=1;overwrite=1;dir=C:\

	Parameters: • PackageGUID: String - the GUID (in XML format) of the Package to generate code for • ExtraOptions: String - enables extra options to be given to the command; currently enables: - Generation of all sub-Packages (recurse) - Force overwrite of all files (overwrite) and - Specification to auto generate all paths (dir)
GeneratePackageDDL (string PackageGUID, string FileName, string ExtraOptions)	Boolean Notes: Generates DDL for all elements in a Package using the options that are currently set on the Generate DDL screen.
GenerateTestFromScenario (string ElementGUID, EnumScenarioTestType TestType)	Boolean Notes: Generates a Vertical Test Suite, a Horizontal Test Suite, an Internal test or an External test from the scenario specification of an element. Parameters: • ElementGUID: String - the GUID (in XML format) of the element containing the scenario specification • TestType: EnumScenarioTestType - the type of test to generate; see *ScenarioTestType Enum* for accepted values
GenerateWSDL(string WSDLComponentGUID, string Filename, string Encoding, string ExtraOptions)	Boolean Notes: Generates WSDL for the specified WSDL stereotyped Component. Parameters: • WSDLComponentGUID: String - the GUID (in XML format) of the WSDL stereotyped Component • Filename: String - the target file path • Encoding: String - the XML encoding for the code page instruction • ExtraOptions: String - enables extra options to be given to the command; currently unused
GenerateXSD (string PackageGUID, string FileName, string Encoding, string Options)	Boolean Notes: Creates an XML schema for a Package, specified by its GUID. Returns True on success. Parameters: • PackageGUID: String - the GUID (in XML format) of the Package • FileName: String - the target filepath • Encoding: String - the XML encoding for the code page instruction • Options: String - enables extra options to be given to the command, in a comma-separated string; currently enables: - GenGlobalElement - turn the generation of global elements for all global ComplexTypes On or Off; for example: GenGlobalElement=1 - UseRelativePath - turns on or off the option to use a relative path in the XSD import or XSD include statement when referencing external Package, provided the schemaLocation tag is empty on the referenced Packages; for example: UseRelativePath=1
GetAllDiagramImagesAnd	Boolean

Map (string Directory)	Notes : Saves the image and image-map for every diagram in the model, in the specified directory location. The image files will be saved in PNG format and each will have the diagram GUID as the image name. The image-map files will be saved as .txt files and each will have the diagram GUID as the image map name. The 'Auto Create Diagram Image and Image Map' option must be selected in the model options for this function to save the images and image-maps. Parameters: • Directory – the location of the directory into which the images and image-maps are to be saved
GetBaselines (string PackageGUID, string ConnectString)	String Notes: Returns a list (in XML format) of Baselines associated with the supplied Package GUID. Parameters: • PackageGUID: String - the GUID (in XML format) of the Package to get Baselines for • ConnectString: String - not currently used
GetDiagram (string DiagramGUID)	protected abstract: String Notes: Gets the diagram details, in XML format. Parameters: • DiagramGUID: String - the GUID (in XML format) of the diagram to get details for
GetDiagramImageAndMap (string DiagramGUID, string Directory)	Boolean Notes: Saves the image and image-map for the diagram with the specified GUID, in the specified directory location. The image will be saved in PNG format and will have the DiagramGUID as the image name. The image-map will be saved as a .txt file and will have the DiagramGUID as the image-map name. The 'Auto Create Diagram Image and Image Map' option must be selected in the model-specific options for this function to save the image and image-map. Parameters: • DiagramGUID – the GUID of the diagram for which the image and image-map are to be saved • Directory – the directory into which the image and image-map are to be saved
GetElement (string ElementGUID)	protected abstract: String Notes: Gets XML for the specified element. Parameters: • ElementGUID: String - the GUID (in XML format) of the element to retrieve XML for
GetElementConstraints (string ElementGUID)	protected abstract: String Notes: Gets constraints for an element, in XML format. Parameters: • ElementGUID: String - the GUID (in XML format) of the element
GetElementEffort (string	protected abstract: String

ElementGUID)	Notes: Gets efforts for an element, in XML format. Parameters: • ElementGUID: String - the GUID (in XML format) of the element
GetElementFiles (string ElementGUID)	protected abstract: String Notes: Gets metrics for an element, in XML format. Parameters: • ElementGUID: String - the GUID (in XML format) of the element
GetElementMetrics (string ElementGUID)	protected abstract: String Notes: Gets files for an element, in XML format. Parameters: • ElementGUID: String - the GUID (in XML format) of the element
GetElementProblems (string ElementGUID)	protected abstract: String Notes: Gets a list of issues (problems) associated with an element, in XML format. Parameters: • ElementGUID: String - the GUID (in XML format) of the element
GetElementProperties (string ElementGUID)	protected abstract: String Notes: Gets Tagged Values for an element, in XML format. Parameters: • ElementGUID: String - the GUID (in XML format) of the element
GetElementRequirements (string ElementGUID)	protected abstract: String Notes: Gets a list of requirements for an element, in XML format. Parameters: • ElementGUID: String -the GUID (in XML format) of the element
GetElementResources (string ElementGUID)	protected abstract: String Notes: Gets a list of resources for an element, in XML format. Parameters: • ElementGUID: String - the GUID (in XML format) of the element
GetElementRisks (string ElementGUID)	protected abstract: String Notes: Gets a list of risks associated with an element, in XML format. Parameters: • ElementGUID: String - the GUID (in XML format) of the element
GetElementScenarios (string ElementGUID)	protected abstract: String Notes: Gets a list of scenarios for an element, in XML format. Parameters: • ElementGUID: String - the GUID (in XML format) of the element
GetElementTests (string ElementGUID)	protected abstract: String Notes: Gets a list of tests for an element, in XML format. Parameters:

	• ElementGUID: String - the GUID (in XML format) of the element
GetFileNameDialog (string Filename, string FilterString, long FilterIndex, long Flags, string InitialDirectory, long OpenOrSave)	String Notes: Opens a standard 'File Open' or 'Save As' dialog and returns a string containing the full path to the selected file on success. Returns an empty string if the dialog was canceled. For example: Filename = "" FilterString = "CSV Files (*.csv)|*.csv|All Files (*.*)|*.*||" Filterindex = 1 Flags = &H2 'OFN_OVERWRITEPROMPT InitialDirectory = "" OpenOrSave = 1 filepath = Project.GetFileNameDialog (Filename, FilterString, Filterindex, Flags, InitialDirectory, OpenOrSave) In this example, the 'Save As' dialog will prompt for a CSV file. Parameters: • Filename: String - default filename specified in the dialog • FilterString: String - delimited list of available file type filters • Filterindex: Long - one-based index of the filter to be used by default • Flags: Long - additional bit flags used to initialize the file dialog; see the OPENFILENAME structure in MSDN documentation for accepted values • InitialDirectory: String - directory path to open this dialog • OpenOrSave: Long - show dialog as an 'Open' or 'Save As' style dialog; accepted values: 0 = Open, 1 = Save As
GetLastError ()	protected abstract: String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
GetLink (string LinkGUID)	protected abstract: String Notes: Gets connector details, in XML format. Parameters: • LinkGUID: String - the GUID (in XML format) of the connector to get details of
GUIDtoXML (string GUID)	String Notes: Changes an internal GUID to the form used in XML. Parameters: • GUID: String - the Enterprise Architect style GUID to convert to XML format
ImportDirectory (string PackageGUID, string Language, string DirectoryPath, string ExtraOptions)	Boolean Notes: Imports a source code directory into the model. Parameters: • PackageGUID: String - the GUID (in XML format) of the Package to reverse engineer code into • Language: String - specifies the language of the code to be imported • DirectoryPath: String - specifies the path where the code is found on the

	computer • ExtraOptions: String - enables extra options to be given to the command; currently enables import of source from all child directories (recurse) - for example: recurse=1
ImportFile (string PackageGUID, string Language, string FileName, string ExtraOptions)	Boolean Notes: Imports an individual file or binary module into the model, in a Package per namespace style import. Parameters: • PackageGUID: String - the GUID (in XML format) of the Package to reverse engineer code into; this is expected to be a namespace root Package • Language: String - specifies the language of the code to be imported Use the value 'DNPE' to import a binary module; this imports a .NET assembly or Java .class file, but not a .jar file • Filename: String - specifies the path where the code or module is found on the computer • ExtraOptions: String - enables extra options to be given to the command; currently unused
ImportPackageXMI (string PackageGUID, string Filename, long ImportDiagrams, long StripGUID)	String Notes: Imports an XMI file at a point in the tree. Returns an empty string if successful, or returns an error message on failure. Parameters: • PackageGUID: String - the GUID (in XML format) of the target Package to import the XMI file into (or overwrite with the XMI file) • Filename or XMLText: String - the name of the XMI file; if the String is of type filename it is interpreted as a source file, otherwise the String is imported as XML text • ImportDiagrams: Long - 1 for importing diagrams and 0 to skip importing diagrams • StripGUID: Long - 1 to replace the element UniqueIDs on import; if stripped, then a copy of the Package could be imported into the same Enterprise Architect model as two different versions - 0 to retain the element UniqueIDs on import; a duplicate copy of the Package cannot be created in the same model of Enterprise Architect
ImportReferenceData (string FileName, string DataSets)	Boolean Notes: Imports Reference Data. Parameters: • FileName: String - the name of the reference data file to import from • DataSets: String - the list of reference data sets to import from; the data set delimeter is ";" If the string is empty, Enterprise Architect displays a dialog prompt to select the data sets to import
ImportVisualStudioSolution (string PackageGUID, string SolutionPath)	Boolean Notes: Imports a Visual Studio Solution into the model. Parameters: • PackageGUID: string - the GUID (in XML format) of the Package to reverse

	engineer the solution into • SolutionPath: string - specifies the path of the Visual Studio Solution file on the computer
LayoutDiagram (string DiagramGUID, long LayoutStyle)	Boolean Notes: Deprecated. Use **LayoutDiagramEx**. Calls the function to automatically layout a diagram in hierarchical fashion. It is only recommended for Class and Object diagrams. Parameters: • DiagramGUID: String - the GUID (in XML format) of the diagram to lay out • LayoutStyle: Long - always ignored
LayoutDiagramEx (string DiagramGUID, long LayoutStyle, long Iterations, long LayerSpacing, long ColumnSpacing, boolean SaveToDiagram)	Boolean Notes: Calls the function to automatically layout a diagram in hierarchical fashion. It is only recommended for Class and Object diagrams. LayoutStyle accepts these options • Default Options: - lsDiagramDefault - lsProgramDefault • Cycle Removal Options: - lsCycleRemoveGreedy - lsCycleRemoveDFS • Layering Options: - lsLayeringLongestPathSink - lsLayeringLongestPathSource - lsLayeringOptimalLinkLength • Initialize Options: - IsInitializeNaive - IsInitializeDFSOut - IsInitializeDFSIn • Crossing Reduction Option: - lsCrossReduceAggressive • Layout Options - Direction - lsLayoutDirectionUp - lsLayoutDirectionDown - lsLayoutDirectionLeft - lsLayoutDirectionRight Parameters: • DiagramGUID: String - the GUID (in XML format) of the diagram to lay out • LayoutStyle: Long - the layout style • Iterations: Long - the number of layout iterations the Layout process should take to perform cross reduction (Default value = 4) • LayerSpacing: Long - the per-element layer spacing the Layout process should use (Default value = 20) • ColumnSpacing: Long - the per-element column spacing the Layout process should use (Default value = 20) • SaveToDiagram: Boolean - specifies whether or not Enterprise Architect

	should save the supplied layout options as default to the diagram in question
LoadControlledPackage (string PackageGUID)	String Notes: Loads a Package that has been marked and configured as controlled. The filename details are stored in the Package control data. Parameters: • PackageGUID: String - the GUID (in XML format) of the Package to load
LoadDiagram (string DiagramGUID)	protected abstract: Boolean Notes: Loads a diagram by its GUID. Parameter: • DiagramGUID: String - the GUID (in XML format) of the diagram to load; if you retrieve the GUID using the Diagram interface, use the GUIDtoXML function to convert it to XML format
LoadProject (string FileName)	protected abstract: Boolean Notes: Loads an Enterprise Architect project file. Do not use this method if you have accessed the Project interface from the Repository, which has already loaded a file. Parameters: • FileName: String - the name of the project file to load
Migrate (string GUID, string SourceType, string DestinationType)	Void Notes: Migrates a model (or part of a model) from one BPMN, ArchiMate, UPDM or SysML format to an upgraded format. Parameters: • GUID: String - the GUID of the Package or element for which the contents are to be migrated • SourceType: String - the type of model to be upgraded; accepted values: - BPMN - BPMN1.1 - UPDM - SysML1.1 - SysML1.2 - SysML1.3 - ArchiMate - ArchiMate2 - UPDM2 • DestinationType: String - the type of model to upgrade to; accepted values: - BPMN1.1 - BPMN1.1::BPEL - BPMN2.0 - UPDM2 - SysML1.2 - SysML1.3 - SysML1.4 - ArchiMate2 - ArchiMate3 - UAF
MigrateToBPMN11 (string GUID, string Type)	Void Notes: Migrates every BPMN 1.0 construct in a Package or an element (including elements, attributes, diagrams and connectors) to BPMN 1.1.

	Parameters • GUID: String - the GUID of the Package or element for which the contents are to be migrated to BPMN 1.1 • Type: String - the type of upgrade, either just to BPMN 1.1 or to BPMN 1.1 and BPEL. Accepted values are: - BPMN = migrate to BPMN 1.1 - BPEL = migrate to BPMN 1.1 and update: - any diagram with stereotype BPMN to BPEL - any element with stereotype BusinessProcess to BPELProcess Migrating to BPEL is possible in the Ultimate and Unified Editions of Enterprise Architect.
ProjectTransfer (string SourceFilePath, string TargetFilePath, string LogFilePath)	Boolean Notes: Transfers the project from a source .eap file or DBMS to a target .eap file, .eapx file or .feap file. Parameters: • SourceFilePath: String - the path of the source file to transfer • TargetFilePath: String - the path of the target file, including the file type extension; Enterprise Architect creates a new Base project in this location (using the TargetFilePath as its name) and then transfers the content of the source project into that file • LogFilePath: String - the path of the log file where the status of the transfer process is updated In automation, the target file must not previously exist. Enterprise Architect creates a new, empty Base.* file using the specified target name and extension, and transfers the source project into it.
PublishResult (string CategoryID, EA.EnumMVErrorType ErrorType, string ErrorMessage)	String Notes: Returns the results of each rule that can be performed during model validation. It must be called once for each rule from the EA_OnInitializeUserRules broadcast handler. The return value is a RuleId, which can be used for reference purposes when an individual rule is executed by Enterprise Architect during model validation. See the Model Validation Example for a detailed example of the use of this method. Parameters: • CategoryId: String - should be passed the return value from the DefineRuleCategory method • ErrorType: EA.EnumMVErrorType - depending on the severity of the error being validated, can be: - mvErrorCritical - mvError - mvWarning, or - mvInformation • ErrorMessage: String - contains an error string
PutDiagramImageOnClipboard (string DiagramGUID, long Type)	protected abstract: Boolean Notes: Copies an image of the specified diagram to the clipboard. Parameters: • DiagramGUID: String - the GUID (in XML format) of the diagram to copy • Type: Long - the file type - If Type = 0 then it is a metafile - If Type = 1 then it is a Device Independent Bitmap

PutDiagramImageToFile (string Diagram GUID, string FileName, long Type)	protected abstract: Boolean Notes: Saves an image of the specified diagram to file. Parameters: • DiagramGUID: String - the GUID (in XML format) of the diagram to save • FileName: String - the name of the file to save the diagram into • Type: Long - the file type - If type = 0 then it is a metafile - If type = 1 then it uses the file type from the name extension (that is, .bmp, .jpg, .gif, .png, .tga)
ReloadProject ()	protected abstract: Boolean Notes: Reloads the current project. This is a convenient method to refresh the current loaded project (in case of outside changes to the .eap file).
RunExecutableStatemachine (string ElementGUID, string ExtraOptions)	Boolean Notes: Runs Executable StateMachine code for an <<executable statemachine>> Artifact element, which will start simulation of the StateMachine Parameters: • ElementGUID: String - the GUID (in XML format) of the element to generate • ExtraOptions: String - enables extra options to be given to the command (currently unused)
RunModelSearch (string Search, string SearchTerm, bool ShowInEA)	Void Notes: Invokes the Model Search component. Parameters: • Search: String - the name of an Enterprise Architect defined search • SearchTerm: String - the term to search for in the project • ShowInEA: Boolean - execute the search and output in the Model Search window
RunReport (string PackageGUID, string TemplateName, string Filename)	protected abstract: Void Notes: Runs a named document report. Parameters: • PackageGUID: String - the GUID of the Package or master document to run the report on • TemplateName: String - the document report template to use; if the PackageGUID has a stereotype of MasterDocument, the template is not required • FileName: String - the file name and path to store the generated report; the file extension specified will determine the format of the generated document - for example, RTF, PDF
RunHTMLReport (string PackageGUID, string ExportPath, string ImageFormat, string Style,	String Notes: Runs an HTML report (as for 'Documentation | Publish as HTML' when you click on a Package in the Browser window and on the  icon). Parameters: • PackageGUID: String - the GUID (in XML format) of the Package or master

string Extension)	document to run the report on • ExportPath: String - the directory path to store the generated report files • ImageFormat: String - file format in which to store images - .png or .gif • Style: String - name of the web style template to apply; use <default> for the standard, system-provided template • Extension: String - file extension for generated HTML files (example: .htm)
SaveControlledPackage (string PackageGUID)	String Notes: Saves a Package that has been configured as a controlled Package, to Native/XMI format. Only the Package GUID is required, Enterprise Architect picks the rest up from the Package control information. Parameter: • PackageGUID: String - the GUID (in XML format) of the Package to save
SaveDiagramImageToFile (string Filename)	protected abstract: String Notes: Saves a diagram image of the current diagram to file. Parameters: • FileName: String - the filename of the image to save
ShowWindow (long Show)	protected abstract: Void Notes: Shows or hides the Enterprise Architect User Interface. Parameters: • Show: Long
SynchronizeClass (string ElementGUID, string ExtraOptions)	Boolean Notes: Synchronizes a Class with the latest source code. Parameters: • ElementGUID: String - the GUID (in XML format) of the element to update from code • ExtraOptions: String - enables extra options to be given to the command; currently unused
SynchronizePackage (string PackageGUID, string ExtraOptions)	Boolean Notes: Synchronizes each Class in a Package with the latest source code. Parameters: • PackageGUID: String - the GUID (in XML format) of the Package containing the elements to update from code • ExtraOptions: String - enables extra options to be given to the command; currently enables synchronization of all child Packages (children) - for example: children=1
TransformElement (string TransformName, string ElementGUID, string TargetPackage, string ExtraOptions)	Boolean Notes: Transforms an element into a Package. Parameters: • TransformName: String - specifies the transformation that should be executed • ElementGUID: String - the GUID (in XML format) of the element to transform • TargetPackageGUID: String - the GUID (in XML format) of the Package to transform into • ExtraOptions: String - enables extra options to be given to the command:

	- GenCode=True / False - articulate code generation from the transformed elements; this option supercedes the current model setting
TransformPackage (string TransformName, string SourcePackage, string TargetPackage, string ExtraOptions)	Boolean Notes: Runs a transformation on the contents of a Package. Parameters: • TransformName: String - specifies the transformation that should be executed • SourcePackageGUID: String - the GUID (in XML format) of the Package to transform • TargetPackageGUID: String - the GUID (in XML format) of the Package to transform into • ExtraOptions: String - enables extra options to be given to the command: - GenCode=True/False - articulate code generation from the transformed elements; this option supercedes the current model setting - SubPackages=True/False - specify if the child Packages are to be included whilst transforming a Package
ValidateDiagram (string DiagramGUID)	Boolean Notes: Invokes the Enterprise Architect Model Validation component, then validates the diagram (for correctness) and the elements and connectors within the diagram. Output can be viewed through 'Start > Application > Design > System Output > Model Validation'. Returns a Boolean value to indicate the success or failure of the process, regardless of the results of the validation. Parameters: • DiagramGUID: String - the GUID of the Diagram Class object
ValidateElement (string ElementGUID)	Boolean Notes: Invokes the Enterprise Architect Model Validation component, then validates the element and all child elements, diagrams, connectors, attributes and operations. Output can be viewed through 'Start > Application > Design > System Output > Model Validation'. Returns a Boolean value to indicate the success or failure of the process, regardless of the results of the validation. Parameters: • ElementGUID: String - the GUID of the Element Class object
ValidatePackage (string PackageGUID)	Boolean Notes: Invokes the Enterprise Architect Model Validation component, then validates the Package and all sub-Packages, elements, connectors and diagrams within it. Output can be viewed through 'Start > Application > Design > System Output > Model Validation'. Returns a Boolean value to indicate the success or failure of the process, regardless of the results of the validation. Parameters:

	• PackageGUID: String - the GUID of the Package Class object
XMLtoGUID (string GUID)	String Notes: Changes a GUID in XML format to the form used inside Enterprise Architect. Parameters: • GUID: String - the XML style GUID to convert to Enterprise Architect internal format

Chart Package

The Chart interface can be used to dynamically construct any of the supported Chart types, using the functions provided in the Chart Package. The interface is obtained using the GetChart method on a Dynamic Chart element. A Dynamic Chart element can be created from the 'Charts' page of the Diagram Toolbox, and is typically used on a Dashboard diagram.

Chart Enumerations

These enumerations, used specifically by methods in the Chart interface, are described in the topics of this section. Click on the enumeration name in the list to the left of this text.

ChartAxisCrossType

Enum Values

Enum	Value
Auto	value: 0
MaximumAxisValue	value: 1
MinimumAxisValue	value: 2
AxisValue	value : 3
Ignore	value: 4
FixedDefaultPos	value: 5

ChartAxisIndex

Enum Values

Enum	Value
Unknown	value: -1
X	value: 0
Y	value: 1
Z	value: 2

ChartAxisLabelType

Enum Values

Enum	Value
NoLabels	value: 0
NextToAxis	value: 1
High	value: 2
Low	value: 3

ChartAxisTickMarkType

Enum Values

Enum	Value
NoTicks	value: 0
Inside	value: 1
Outside	value: 2
Cross	value: 3

ChartAxisType

A set of constants that refer to the various axes used in charts.

Enum Values

Enum	Value
CHART_Y_PRIMARY_AXIS	value: 0
CHART_Y_SECONDARY_AXIS	value: 1
CHART_X_PRIMARY_AXIS	value: 2
CHART_X_SECONDARY_AXIS	value: 3
CHART_Z_PRIMARY_AXIS	value: 4
CHART_Z_SECONDARY_AXIS	value: 5
CHART_Y_POLAR_AXIS	value: 6
CHART_X_POLAR_AXIS	value: 7
CHART_A_TERNARY_AXIS	value: 8
CHART_B_TERNARY_AXIS	value: 9
CHART_C_TERNARY_AXIS	value: 10

ChartBarShape

Enum Values

Enum	Value
Box	value: 0
Pyramid	value: 1
PyramidPartial	value: 2

ChartCategory

Enum Values

Enum	Value
chartDefault	value: 0
chartLine	value: 1
chartPie	value: 2
chartPie3D	value: 3
chartPyramid	value: 4
chartPyramid3D	value: 5
chartFunnel	value: 6
chartFunnel3D	value: 7
chartColumn	value: 8
chartBar	value: 9
chartHistogram	value: 10
chartArea	value: 11
chartStock	value: 12
chartBubble	value: 13
chartLongData	value: 14
chartHistoricalLine	value: 15
chartPolar	value: 16
chartDoughnut	value: 17
chartDoughnut3D	value: 18
chartTorus3D	value: 19
chartTernary	value: 20
chartColumn3D	value: 21

chartBar3D	value: 22
chartLine3D	value: 23
chartArea3D	value: 24
chartSurface3D	value: 25
chartDoughnutNested	value: 26
chartBoxPlot	value: 27
chartBarSmart	value: 28
chartBar3DSmart	value: 29

ChartColorMode

Enum Values

Enum	Values
Single	value: 0
Multiple	value: 1
Palette	value: 2
Custom	value: 3
Series	value: 4

ChartCurveType

Enum Values

Enum	Value
NoLine	value: 0
Line	value: 1
Spline	value: 2
SplineHermite	value: 3
Step	value: 4
ReversedStep	value: 5

ChartDashStyle

Enum Values

Enum	Value
Solid	value: 0
Dash	value: 1
Dot	value: 2
DashDot	value: 3
DashDotDot	value: 4
Custom	value: 5

ChartFrameStyle

Enum Values

Enum	Value
None	value: 0
Mesh	value: 1
Contour	value: 2
ContourMesh	value: 3

ChartGradientType

Enum Values

Enum	Value
None	value: 0
Horizontal	value: 1
Vertical	value: 2
DiagonalLeft	value: 3
DiagonalRight	value: 4
CenterHorizontal	value: 5
CenterVertical	value: 6
RadialTop	value: 7
RadialCenter	value: 8
RadialBottom	value: 9
RadialLeft	value: 10
RadialRight	value: 11
RadialTopLeft	value: 12
RadialTopRight	value: 13
RadialBottomLeft	value: 14
RadialBottomRight	value: 15
Bevel	value: 16
PipeVertical	value: 17
PipeHorizontal	value : 18

ChartMarkerShape

Enum Values

Enum	Value
Circle	value: 0
Triangle	value: 1
Rectangle	value: 2
Rhombus	value: 3

ChartStockSeriesType

Enum Values

Enum	Value
Bar	value: 0
Candle	value: 1
LineOpen	value: 2
LineHigh	value: 3
LineLow	value: 4
LineClose	value: 5
LineCustom	value: 6

ChartType

Enum Values

Enum	Value
chartTypeDEFAULT	value: 0
chartTypeSIMPLE	value: 1
chartTypeSTACKED	value: 2
chartType100STACKED	value: 3
chartTypeRANGE	value: 4

ChartWallOptions

Enum Values

Enum	Value
None	value: 0, 0x0000
FillLeftWall	value: 1, 0x0001
OutlineLeftWall	value: 2, 0x0002
FillRightWall	value: 4, 0x0004
OutlineRightWall	value: 8, 0x0008
FillFloor	value: 16, 0x0010
OutlineFloor	value: 32, 0x0020
DrawAll	value: 65535, 0xFFFF
DrawLeftWall	FillLeftWall | OutlineLeftWall
DrawRightWall	FillRightWall | OutlineRightWall
DrawFloor	FillFloor | OutlineFloor
DrawAllWalls	DrawLeftWall | DrawRightWall
OutlineAllWalls	OutlineLeftWall | OutlineRightWall
OutlineAll	OutlineAllWalls | OutlineFloor
FillAllWalls	FillLeftWall | FillRightWall
FillAll	FillAllWalls | FillFloor
Default	OutlineAll

Chart Class

The Chart Class is the primary interface for Chart elements; it is used to create a series, add datapoints to a series and configure the chart appearance.

Chart Attributes

Attribute	Description
Title	String Notes: Read/Write The title of the chart.
Category	ChartCategory Notes: Read only The chart category; provided in the SetChartType method.
Type	ChartType Notes: Read only The chart type; provided in the SetChartType method.

Chart Methods

Method	Description
AddChartDataYXZ(double Y, double X, double Z, long seriesIndex)	long Adds a datapoint to an existing series. Parameters: • Y: double, the primary Y axis value • X: double, the primary X axis value • Z: double, the primary Z axis value • seriesIndex: long, the index of the series (returned by the CreateSeries methods)
AddChartDataYY1(string category, double Y, double Y1, long seriesIndex)	long Adds a datapoint to an existing series. Parameters: • category: string - the x axis group, column or label • Y: double, the primary Y axis value • Y1: double, the secondary Y axis value • seriesIndex: long, the index of the series (returned by the CreateSeries and CreateSeriesEx methods)

CreateSeries(string name)	LDISPATCH Adds a new series to the chart. Returns an interface that can be used to add data to the series and configure its appearance. Parameters: • name: string, the displayed name of the series
CreateSeriesEx(string name, long color, ChartType type, ChartCategory category)	LDISPATCH Creates a series of a particular chart category and type and returns an IChartSeries interface. This allows charts to form multiple series in various ways. The CombinedCharts in the EAExample Model are an example of this, displaying three series for the Area, Column and Line categories respectively. Parameters: • name: string, the name of the series • color: long, RGB color value,-1 for default • type: ChartType, one of the ChartType enumerations • category: ChartCategory, one of the ChartCategory enumerations
EnableResizeAxes(boolean bEnable)	void Grants or denies the ability to resize axes. Parameters: • bEnable: boolean
GetChartAxis(ChartAxisType type)	LDISPATCH Returns an IChartAxis interface for the specified axis. Parameters: • type: ChartAxisType, one of the ChartAxisType enumerations
GetDiagram3D()	LDISPATCH Returns an IChartDiagram interface that can be used to specify the rendering engine; Software or OpenGL.
GetSeries(long index)	LDISPATCH Returns an IChartSeries interface for the given index. Indices for series begin at zero. Parameters: • index: long
GetSeriesCount()	long Returns the number of series represented by the chart.
Redraw()	void Redraws the chart.
SetChartType(ChartType type, ChartCategory category, boolean bRedraw, boolean bResizeAxis)	void This is typically the first call made on the Chart interface. It defines the style and appearance of the Chart when it is rendered. Parameters: • type: ChartType

	• category: ChartCategory • bRedraw: Boolean • bResizeAxis: Boolean
SetCurveType(ChartCurveType type)	void Sets the interpolation method of drawing the curve. Parameters: • type: ChartCurveType, one of the ChartCurveType enumerations, such as Line, Spline or SplineHermite.
SetSeriesShadow(boolean bShow)	void Displays or hides shadows on series. Parameters: • bShow: Boolean
SetThemeOpacity(long percentage)	void Sets the opacity of the chart. Parameters: • percentage: long, chart transparency as a percentage
ShowAxis(long index)	void Shows the axis for the given index. Parameters: • index: long, one of the ChartAxisType enumerations
ShowDataLabels(boolean show, boolean border, boolean dropLineTomarker)	void Shows or hides data labels on the chart. Parameters: • show: Boolean, show or hide labels • border: Boolean, show or hide border on labels • dropLineTomarker: Boolean, changes position of label with respect to line
ShowDataMarkers(boolean show, long size, ChartmarkerShape shape)	void Shows or hides data markers on the chart. Also allows setting the appearance of markers. Parameters: • show: Boolean, show or hide markers • size: long, size of markers in pixels • shape: ChartmarkerShape, one of the ChartmarkerShape enumerations

ChartAxisIndex Class

ChartAxisIndex Attributes

Attribute	Description
Visible	Boolean Shows or hides the axis.

ChartAxisIndex Methods

Method	Description
EnableMajorUnitIntervalInterlacing(boolean binterlace)	void Turns interlacing on or off.
GetGuid()	string Returns the guid of the axis. Uniquely identifies an axis.
GetLabel()	string Returns the value of the label of the axis.
SetAxisName(string label, boolean showonaxis)	void Sets the label for the axis and whether it should be displayed on the chart. Parameters: • label: string, the text for the label • showonaxis: Boolean, a true value indicates that the label is displayed
SetCrossType(long type)	void Provides a directive or hint for use when calculating the position of labels on an axis. Parameters: • type: long, one of the ChartAxisCrossType enumerations
SetDataFormat(string format, boolean formatAsDate)	void Sets the format string for the conversion of values to strings (e.g. "%.4f"). If the datapoints represent datetime values, the formatAsDate argument should be true, and the format string set appropriately (e.g. "%H:%M") Parameters: • format: string, the format to use when converting datapoint values to string • formatAsDate: Boolean, a true value indicates the datapoint represent a datetime
SetDisplayUnits(double	

units)	void Sets the display units on the axis. Basically, the datapoint values are divided by this figure to give a major unit value. For example, if the datapoint contains meter values, a value of 1000 would result in kilometers being used as the major unit on the axis. Parameters: • units: double, the value of a single unit on the axis
SetFixedDisplayRange(double fmin, double fmax)	void Sets a fixed range for the axis. Parameters: • fmin: double, the minimum value • fmax: double, the maximum value
SetLabelType(long labelpos)	void Sets the position of labels on the axis. Parameters: • labelpos: long, one of the ChartAxisLabelType enumerations
SetTickMark(long tickmarkpos)	void Sets the position of tick marks on the axis. Parameters: • tickmarkpos: long, one of the ChartAxisTickMarkType enumerations
ShowMajorGridLines(boolean show)	void Shows or hides grid lines.

ChartDataValue Class

The ChartDataValue class provides an interface that allows values to be obtained from points in a series.

ChartDataValue Methods

Method	Description
GetValue()	double Returns the value associated with the datapoint.
IsEmpty()	Boolean True if no value exists for the datapoint.
SetEmpty(boolean empty)	void Sets a datapoint on a series to be empty. Parameters: • empty: Boolean, true if the datapoint is to be considered as empty, having no value
SetValue(double value)	void Sets the value of a datapoint. Parameters: • value: double, the value of the datapoint; setting a value makes a datapoint non-empty

ChartDiagram3D Class

ChartDiagram3D Methods

Method	Description
SetDrawWallOptions(long options, boolean redraw)	void Sets the option for how walls and floors - if any - are displayed on the 3D chart. The options parameter is a bitmask of one or more values from the ChartWallOptions enum. Parameters: • options: Long, bitmask of wall and floor display options • redraw: Boolean, redraws the chart after the function completes
SetRenderingType(long engine)	void Parameters: • engine: long, 0 for software,1 for openGL

ChartFormatSeries Class

A helper class for the ChartSeries class that allows setting appearance options.

ChartFormatSeries Methods

Method	Description
SetCurveType(ChartCurveType type)	void Sets the graphic option for rendering lines. Parameters: • type: long, one of the ChartCurveType enumerations
SetSeriesLineWidth(long width)	void Sets the line width in pixels. Parameters: • width: long, a pixel value
SetSeriesOutlineDashStyle(ChartDashStyle dashstyle)	void Sets the dash style of the line on the chart/graph. Parameters: • dashstyle: ChartDashStyle, one of the ChartDashStyle enumerations

ChartSeries Class

ChartSeries Methods

Method	Description
AddBoxPlotData(double ave, double min, double q1, double q2, double q3, double max, double notched)	long For a chart having the BoxPlot category, adds a single datapoint to the series. Parameters: • ave: double, the mean value at this point • min: double, the minimum value at this point • q1: double, the first quartile value • q2: double, the second quartile value • q3: double, the third quartile value • max: double the maximum value at this point • notched: double, for a series with notched style, the notched value to express at this point
AddDataPoint(double Y)	long Adds a datapoint to the series. Returns the index of the point, which is the number of points -1. Parameters: • Y: double, the Y axis value
AddDataPoint2(double Y, double X)	long Adds a datapoint to the series. Returns the index of the point, which is the number of points -1. Parameters: • Y: double, the Y axis value • X: double, the X axis value
AddDataPoint3(string category, double Y)	long Adds a Y axis value for a given category on the X axis. Parameters: • category: string, the category or column name • Y: double, the value
AddStockData(double open, double high, double low, double closing, VARIANT timestamp)	void Adds data to a series for a chart of the Stock category. Parameters: • open: double, opening value • high: double, high value • low: double, low value • closing: double, closing value • timestamp: {datetime, double utcsecs} either VARIANT date value or double, in which case the value is interpreted as the number of seconds since midnight

	on January 1st, 1970, UTC time
AddSurfaceColors(VARIANT colors)	void Adds one or more colors to the series. Parameters: • colors: long, or long[], a single RGB color or an array of RGB color values
CloseShape(boolean close, boolean fill)	void Connects the first and last datapoints and fills the shape if 'fill' is true. Parameters • close: Boolean, if true closes the series • fill: Boolean, fills the shape
GetDataPointCount()	long Returns the number of datapoints in the series.
GetDataPointValue(long index)	LDISPATCH Returns a ChartDataValue interface for the datapoint with the given index. Parameters: • index: long, the index of the datapoint (typically returned by AddDataPoint functions; a value in the range 0 to n-1, where n is the number of points returned by the *GetDataPointCount* function)
GetSeriesFormat()	LDISPATCH Returns a ChartFormatSeries interface that allows the chart appearance to be changed.
SetBarShape(long barshape)	void Sets the shape for Bar charts, 0 for Box, 1 for Pyramid, 2 for PyramidPartial. Parameters: • barshape: ChartBarShape, one of the ChartBarShape enumerations
SetColorMapCount(long count)	void Sets the number of colors used when rendering the series. Typical values are 4, 8, 16 and 32 Parameters: • count: long, the number of colors to use
SetColorMode(ChartColorMode mode)	void For 3D charts, sets the interpolation method for filling shapes. Single, for example, would result in the 3D object being filled by varying the color slightly. The level of variation will depend on the number of colors used by the chart (see *SetColorMapCount*). Parameters: • mode: ChartColorMode
SetDrawFlat(boolean flat)	void Draws the shape flattened when set to true. Parameters:

	• flat: Boolean, draw flat
SetFrameColor(long color)	void Sets the color of the frame for 3D objects. Parameters: • color: long, the RGB color value for coloring the frame
SetFrameStyle(ChartFrameStyle style)	void Sets the frame style for the chart - none, mesh, contour or both. Parameters: • style: ChartFrameStyle, one of the ChartFrameStyle enumerations
SetGradientType(long type)	void Sets the gradient type to use. Parameters: • type: long, one of the ChartGradientType enumerations
SetGroupID(long id)	void Groups series on a stacked chart having the same id. Must be a non-negative number. Parameters: • id: long, a non-negative number used to group the series on a chart
SetLevelRangeMode(long mode)	void Sets the mode for ranges in series. • 0 - Minimum and maximum for Series • 1 - Minimum and maximum for Y axis • 2 - Custom Parameters: • mode: long, either 0 or 1 supported
SetRelatedAxis(string axis, long index)	void Sets the related axis for a series. The related axis is created using the Split function of the ChartAxis interface. The axis is first created using Split, then a new series is created, and this function called on it to one of its axes. The axis is specified by the index parameter; the value is one of the ChartAxisIndex enumerations (0 for X, 1 for Y or 2 for Z) Parameters: • axis: string, the guid of the axis returned by a ChartAxis.Split method call; returned by the ChartAxis.GetGuid method • index: long, one of the ChartAxisIndex enumerations
SetStockSeriesType(ChartStockSeriesType type)	void For Stock charts, sets the graphic used to render the series. Parameters: • type: ChartStockSeriesType, one of the ChartStockSeriesType enumerations
SetWireFrame(boolean	void

wired)	Sets the wireframe option on or off. When set to true, the chart is no longer rendered as a solid object but is instead rendered as a frame composed of wires. Parameters: • wired: Boolean, displays as a wireframe object if true

Document Generator Interface Package

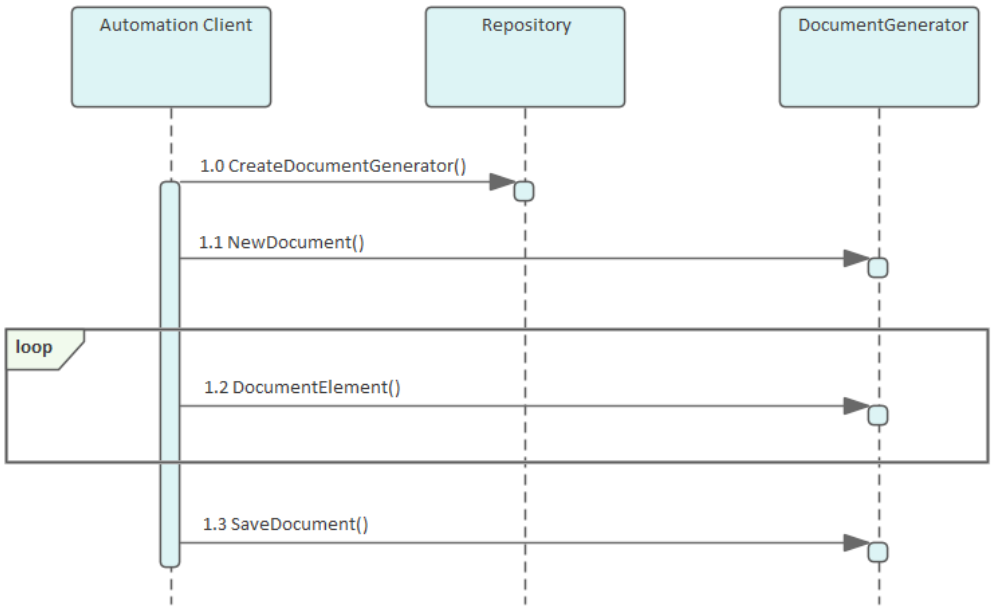
The DocumentGenerator Class provides an interface to the document and web reporting facilities, which you can use to generate reports on specific Packages, diagrams and elements in your model.

Access

Repository Class	You can create a pointer to this interface using the method Repository.CreateDocumentGenerator.

Example

This diagram illustrates how you might use the Document Generator interface in generating a report through the Automation Interface.



Also look at the:

- Document Generation scripting example in the Scripting window ('Specialize > Tools > Script Library', then expand the 'Local Scripts' folder and double-click on 'JScript - Documentation Example')
- RunReport method in the Project Interface

DocumentGenerator Class

The DocumentGenerator Class provides an interface to the document and web reporting facilities, which you can use to generate reports on specific Packages, diagrams and elements in your model. This Class is accessed from the Repository Class using the CreateDocumentGenerator() method.

DocumentGenerator Attributes

Attribute	Remarks
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.

DocumentGenerator Methods

Method	Remarks
DocumentConnector (long connectorID, long nDepth, string templateName)	Boolean Notes: Documents a connector. Parameters: • connectorId: Long - the ID of the connector • nDepth: Long - the depth by which to adjust the heading level • templateName: String - the name of a template to use when documenting connectors; this can be blank
DocumentCustomData (string XML, long nDepth, string templateName)	Boolean Notes: Documents information based on the data supplied. Parameters: • XML: String - the XML of the data to be documented • nDepth: Long - the depth by which to adjust the heading level • templateName: String - the name of a template to use when documenting custom data; this can be blank
DocumentDiagram (long diagramID, long nDepth, string templateName)	Boolean Notes: Documents a diagram. Parameters: • diagramId: Long - the ID of the diagram • nDepth: Long - the depth by which to adjust the heading level • templateName: String - the name of a template to use when documenting diagrams; this can be blank
DocumentElement (long elementID, long nDepth,	Boolean Notes: Documents an element.

string templateName)	Parameters: • elementId: Long - the ID of the element • nDepth: Long - the depth by which to adjust the heading level • templateName: String - the name of a template to use when documenting elements; this can be blank
DocumentModelAuthor (string name, long nDepth, string templateName)	Boolean Notes: Documents a model author. Parameters: • name: String - the name of the author • nDepth: Long - the depth by which to adjust the heading level • templateName: String - a template to use when documenting model authors; this can be blank
DocumentModelClient (string name, long nDepth, string templateName)	Boolean Notes: Documents a single model client. Parameters: • name: String - the name of the client • nDepth: Long - the depth by which to adjust the heading level • templateName: String - a template to use when documenting model clients; this can be blank
DocumentModelGlossary (long id, long nDepth, string templateName)	Boolean Notes: Documents a single model glossary term. Parameters: • id: Long - the ID of the term • nDepth: Long - the depth by which to adjust the heading level • templateName: String - a template to use when documenting model glossary terms; this can be blank
DocumentModelIssue (long id, long nDepth, string templateName)	Boolean Notes: Documents a single model issue. Parameters: • id: Long - the ID of the issue • nDepth: Long - the depth by which to adjust the heading level • templateName: String - a template to use when documenting model issues; this can be blank
DocumentModelResource (string name, long nDepth, string templateName)	Boolean Notes: Documents a single model resource. Parameters: • name: String - the name of the resource • nDepth: Long - the depth by which to adjust the heading level • templateName: String - a template to use when documenting model resources; this can be blank
DocumentModelRole (string name, long nDepth,	Boolean Notes: Documents a single model role.

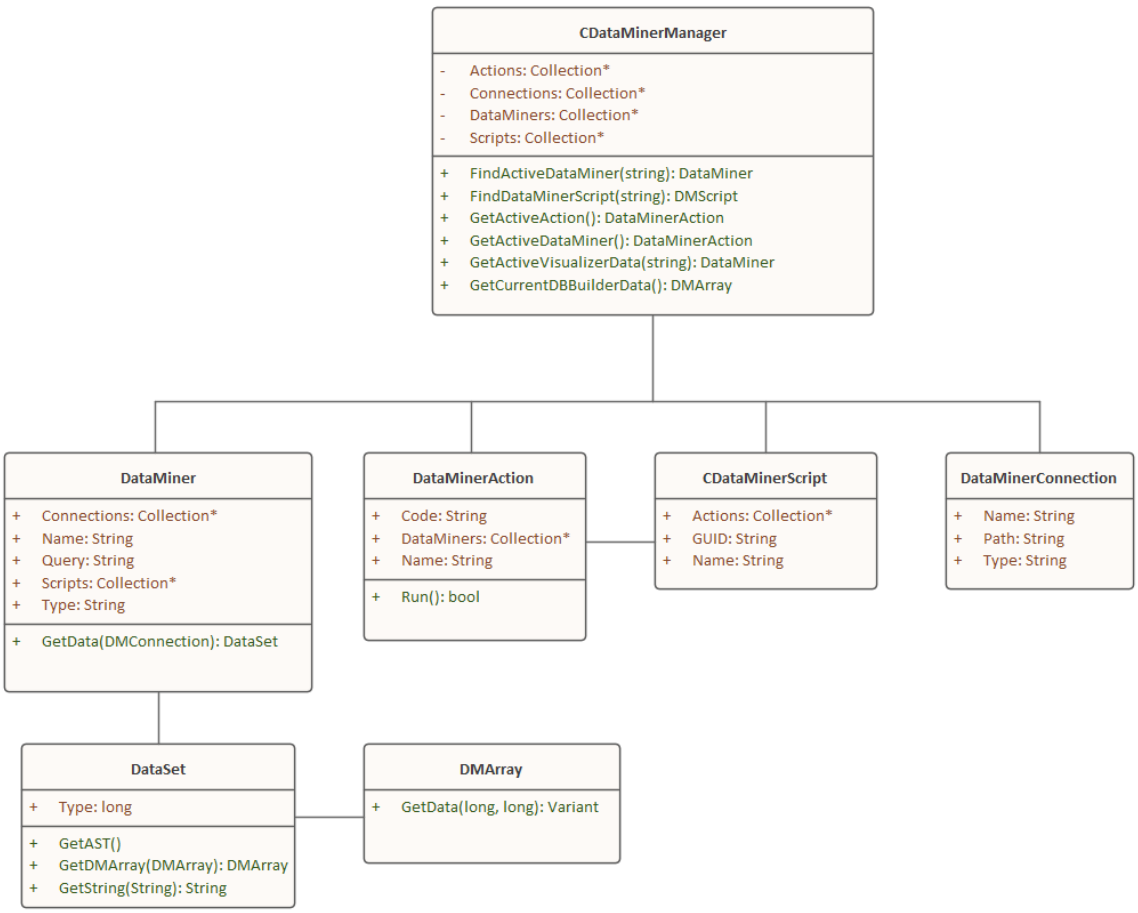
string templateName)	Parameters: • name: String - the name of the role • nDepth: Long - the depth by which to adjust the heading level • templateName: String - a template to use when documenting model roles; this can be blank
DocumentModelTask (long id, long nDepth, string templateName)	Boolean Notes: Documents a single model task. Parameters: • id: Long - the ID of the task • nDepth: Long - the depth by which to adjust the heading level • templateName: String - a template to use when documenting model tasks; this can be blank
DocumentPackage (long packageID, long nDepth, string templateName)	Boolean Notes: Documents a Package. Parameters: • packageId: Long - the ID of the Package • nDepth: Long - the depth by which to adjust the heading level • templateName: String - a template to use when documenting Packages; this can be blank
GetDocumentAsRTF()	Read Only. Returns a string value of the document in raw Rich Text Format.
GetProjectConstant (string nameVal)	String Notes: Returns the value of a Project Constant. Parameters: • nameVal: String - the name of the Project Constant for which to extract the value.
GetLastError ()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
InsertBreak (long breakType)	Boolean Notes: Inserts a break into the report at the current location. Parameters: • breakType: Long - 0 = page break, 1 = section break
InsertCoverPageDocument (string Name)	Boolean Notes: Inserts the Coverpage into the document at the current location. The style sheet is applied to the document before it is insert into the generated document. Parameters: • Name: String - the name of the Cover page document found in the Resource tree
InsertHyperlink (string	

Name, string URL)	Boolean Notes: Inserts a hyperlink at the current location. If you use a URL with the #BOOKMARKNAME syntax, the hyperlink will link to another part of the document. Parameters: • Name: String - the link text to insert into the report • URL: String - The URL of the website to link to
InsertLinkedDocument (string guid)	Boolean Notes: Inserts a Linked Document into the report at the current location. A Linked Document can used to set the header and footer of the report. These are taken from the first Linked Document added to the report. Parameters: • guid: String - the GUID of the element that has a Linked Document
InsertTableOfContents	Boolean Notes: Inserts a Table of Contents at the current position.
InsertTeamReviewPost (string path)	Boolean Notes: Inserts a Model Library posting into the report at the current location. Parameters: • path: String - the path of the Model Library post
InsertTemplate (string templateName)	Notes: Inserts the contents of the template directly into the report. Parameters: • templateName: String - the name of the template to use
InsertText (string text, string style)	Boolean Notes: Inserts static text into the report at the current location. A carriage return is not included; if you need to use one, you can add it manually. Parameters: • text: String - the static text to be inserted • style: String - the name of the style in the template; defaults to Normal style
InsertTOCDocument (string name)	Boolean Notes: Inserts the Table of Contents into the document at the current location. Note: The stylesheet is applied to the document before it is insert into the generated document. Parameters: • name: String - the name of the Table of Contents document found in the Resource tree
LoadDocument(string FileName)	Boolean Notes: Inserts an external document into the currently generated file. Parameters: • FileName: String - the filename of an external document file to insert into the document.

NewDocument (string templateName)	Boolean Notes: Starts a new document; you call this before attempting to document anything else. Parameters: • templateName: String - the name of a template to use when documenting elements; this can be blank
ReplaceField (string fieldname, string fieldvalue)	Boolean Notes: Replaces the 'Section' field identified by the fieldname parameter with the value provided in fieldvalue. For example: ReplaceField ("Element.Alias", "MyAlias") If you call this function more than once with the same fieldname, the field only has the most recent value set. Parameters: • fieldname: String - the field name to find (this does not include the {} braces) • fieldvalue: String - the value to insert into the field; this can be a constant or a derived value
SaveDocument (string filename, long nDocType)	Boolean Notes: Saves the document to disk. Parameters: • filename: String - the filename to save the file to • nDocType: Long - 0 = RTF, 1 = HTML, 2 = PDF, 3 = DOCX
SetPageOrientation (long pageOrientation)	Boolean Notes: Sets the current page orientation. Parameters: • pageOrientation: Long - 0 = Portrait, 1 = Landscape
SetProjectConstant (string newNameVal, string newValue)	Boolean Notes: Sets a Project Constant for the documentation generator; this is saved in the current model. Parameters: • newNameVal: String - the name of the Project Constant • newValue: String - the value of the Project Constant
SetStyleSheetDocument (string name)	Boolean Notes: Sets the Stylesheet to be used for TOC, Coverpage and templates used. This can be called before NewDocument. Parameters: • name: String - the name of the stylesheet found in the Resource tree
SetSuppressProfile (name)	Boolean Notes: Sets the Suppress Profile to be used during report generation. Parameters: • Name: String - The name of the Suppress Profile, as created on the 'Suppress Sections' tab of the 'Document Generation' dialog.

Data Miner Package

The Data Miner Package provides the Automation Interface to the Data Miner elements. It contains these Classes:



For an overview of using the Data Miner see the *Data Miner* Help topic under the *Model Exchange* group of topics.

Notes

- The Data Miner is available in the Unified and Ultimate Editions

DataMinerManager Class

DataMinerManager Attributes

Attribute	Remarks
Actions	Collection Notes: Returns a pointer to the EA.DMAction objects.
Connections	Collection Notes: Returns a Collection of EA.DMConnection objects.
DataMiners	Collection Notes: Returns a Collection of EA.DataMiner objects
Scripts	Collection Notes: Returns a Collection of EA.DMScript objects.

DataMinerManager Methods

Method	Remarks
FindActiveDataMiner (string guid)	DataMiner Object Loads the DataMiner object from the model specified by its GUID. Returns an EA.DataMiner object or NULL if the current selected object isn't a DataMiner object. Parameters: • GUID: string - GUID of the Data Miner object to look up
FindDataMinerScript (string guid)	DMScript object Returns an EA.DMScript object in the model. Parameters: • GUID: string - GUID of DMScript object.
GetActiveAction ()	DMAction Object When you run an Action (operation), from a diagram, this returns the Action's EA.DMAction object. Note: This is generally used for an Action to work out what DataMiner and DMConnections it is linked to.
GetActiveDataMiner ()	DataMiner Object Returns a pointer to an EA.DataMiner object, or NULL if the currently selected object is not a DataMiner object.
GetActiveVisualizerData	

(string name)	DataSet Object Get the EA.DataSet of the currently open Visualizer. Parameters: • Name: string - Name of Open Visualizer Note: Passing in a blank name will return the first Visualizer tab.
GetCurrentDBBuilderData ()	DMArray Object Get the current data from the Database Builder's latest SQL query. Returns the current output of the SQL scratch window. Accessible via: Ribbon: Develop > Data Modeling > Database Builder > SQL Scratch Pad. Return Type: DMArray Returns a pointer to an EA.DMArray object, or NULL if there is not a current Database Builder window with returned data. See The Database Builder Help topic for more information on how to get data into this window.

DataMiner Class

DataMiner Attributes

Attribute	Remarks
Connections	Collection A collection of EA.DMConnection's, Notes: Read Only
Name	String Name of the Script object. Notes: Read Only
Query	String Query of the Data miner object Notes: Read Only
Scripts	Collection A collection of EA.DMScript's, Notes: Read Only
Type	String Type of the Data miner object Notes: Read Only

DataMiner Methods

Method	Remarks
GetData (DMCconnection Connection)	DataSet Returns an EA.DataSet object that represents the query on the connection. Parameters: • connection: DMConnection - A DMConnection object

DataSet Class

DataSet Attributes

Attribute	Remarks
Type	long Type of data contained in this data set. 1. Safe Array 2. Abstract Data type 3. JSon 4. Text Notes: Read Only

DataSet Methods

Method	Remarks
GetAST ()	Currently not supported
GetDMArray ()	DMArray Returns an EA.DMArray object Note: Only supported when Type = 1
GetString ()	String Returns a string of the data. NOTE: Only supported when Type = 3 or 4.

DMArray Class

DMArray Attributes

Attribute	Remarks
ColumnCount	long Notes: Read Only Number of Columns returned in this dataset
RowCount	long Notes: Read Only Number of rows returned in this dataset

DMArray Methods

GetData (long row, long column)	Variant Notes: When the database returns a NULL value, this will return an empty string. Return: Variant. Parameters: • row: Row number of data • column: Column number of data

DMAction Class

DMAction Attributes

Attribute	Remarks
Code	String The code on the Action Notes: Read Only
DataMiners	Collection A Collection of DMDataminer objects Notes: Read Only
Name	String Name of the Action. Notes: Read Only

DMAction Methods

Run ()	Boolean Returns TRUE if the script was run successfully.

DMScript Class

DMScript Attributes

Attribute	Remarks
Actions	Collection Returns a Collection of EA.DMAction's
GUID	String Guid of the Script object. Notes: Read Only
Name	String Name of the Script object. Notes: Read Only

DMConnection Class

DMConnection Attributes

String

Sets the type that the connect object is.

Notes: Read Only

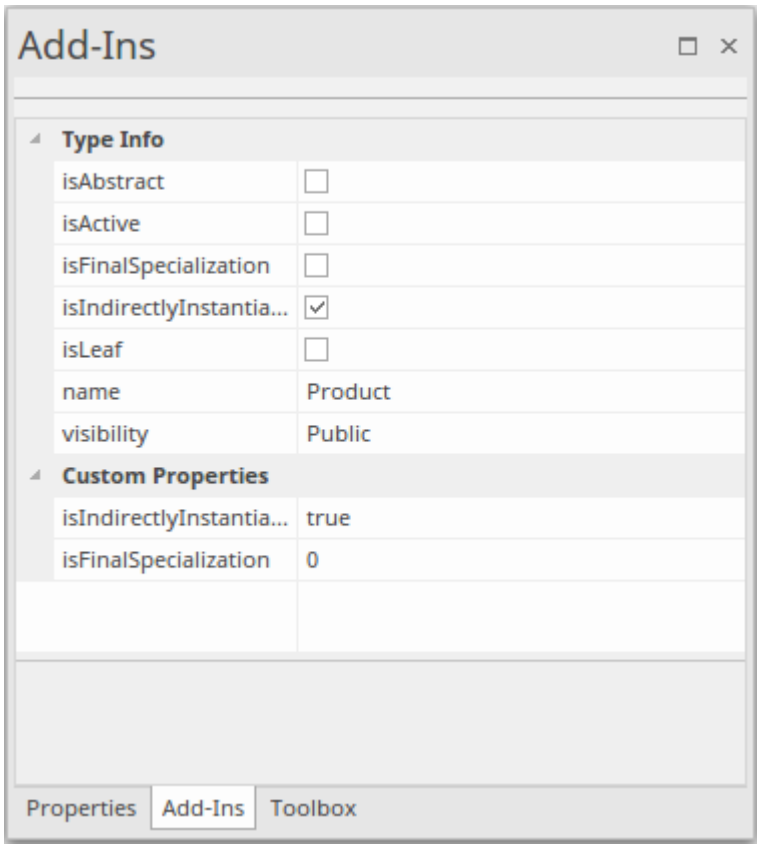
Attribute	Remarks
Name	Type: String Notes: Read Only Name of the Connection object.
Path	Type: String Path to the data we are connecting to. Notes: Read Only
Type	Type: String Notes: Read Only Type of Connection. Options: • ODBC • EA Repository • File • URL

TypeInfoProperties Package

The TypeInfoProperties Package provides an interface to the properties of an object from the perspective of the technology rather than the Enterprise Architect database, allowing read and write access to those properties. It effectively shows the properties contained in the technology-specific and custom categories of the Properties window for the object (and omits the Enterprise Architect specific properties such as the General and Project properties). The interface hides the origin of the properties - whether they are from the base object directly, a Tagged Value, or are MOF properties.

You can see this interface in action in the EA.Example model ('Start > Help > Help > Open the Example Model'). When you open this model:

1. Select the 'Specialize > Manage Addin' ribbon option.
2. Select the checkbox against 'Type Info' and click on the OK button. An icon for 'Type Info' displays on the right of the Add-Ins panel.
3. Click on the drop-down arrow and select the 'Show Type Info' option. The Add-Ins window displays, showing the type information (properties) for the currently-selected object.
4. If you also want to display custom properties in the Add-Ins window, click on the 'Type-Info' icon again and select the 'Include Custom Properties option'. The window resembles this illustration, which is for a UML Component element.



5. Browse the EA.Example model, clicking on different types of object. You will see a different list of properties for, say, an Action than for a Class. Then you can both read and write to those properties. Also compare the list with the Properties window for the same objects.

TypeInfoProperties Class

TypeInfoProperties Attributes

Attribute	Remarks
Count	long Returns the number of TypeInfo Properties.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.

TypeInfoProperties Methods

Method	Remarks
GetLastError ()	String Notes: Returns a string value describing the most recent error that occurred in relation to this object.
GetProperty (String PropName)	Returns the property value as a string. Parameters: • PropName : String - Name of the property
HasProperty (String PropName)	Returns True if the object has the property. Parameters: • PropName : String - Name of the property
Items (object Index)	TypeInfoProperty collection Notes: Accesses an individual TypeInfoProperty. Parameters: • Index: Object - Either a string representing the title text or an integer representing the zero-based index of the TypeInfoProperty to get
SetProperty (String PropName, String Value)	Returns True if the property was set. Parameters: • PropName : String - Name of property • Value : String - Value of property

TypeInfoProperty Class

TypeInfoProperty Attributes

Attribute	Remarks
Name	String Notes: Readonly. Name of the property.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Value	String Get/Sets the Property value.

TypeInfoProperty Methods

<None.>

Method	Remarks

Mail Interface Package

The MailInterface Package contains:

- A function to retrieve a pointer to the interface
- Functions to create and send a mail message within the current mode
- Utility functions for creating hyperlinks to selected model elements

You can get a pointer to this interface using the method Repository.GetMailInterface.

MailInterface Class

The MailInterface interface can be accessed from the Repository using GetMailInterface(). The returned interface provides access to the Enterprise Architect Model Mail Interface. Use this interface to automate the process of creating and sending messages using Enterprise Architect's Model Mail system.

MailInterface Attributes

Attribute	Remarks
MessagingEnabled	Boolean Notes: Read Only Advises whether messaging is enabled on the current model.
ObjectType	ObjectType Notes: Read Only Distinguishes objects referenced through a dispatch interface.

MailInterface Methods

Method	Remarks
ComposeMailMessage(string InitialRecipientGUID, string InitialSubject, messageflag InitialFlag, string InitialMessageText)	Boolean Notes: Creates a new mail message using the values specified in the input parameters; the message is displayed in the composition window, ready for sending. This method does NOT send the message. Parameters: • InitialRecipientGUID: String - Initial value for the GUID of the addressee user (an Enterprise Architect user defined in the current model) • InitialSubject: String - Initial value for the Subject text to display for this message • InitialFlag: MessageFlag - Initial value for the flag type/color to attach to this message • InitialMessageText: String - Initial value for the text that is the body of the message
GetAttributeHyperlink(string AttributeGUID, string LinkText)	String Notes: Returns a string containing a hyperlink to the attribute specified by the input parameter AttributeGUID. Parameters: • AttributeGUID: String - The GUID of the attribute for which a hyperlink is required • LinkText: String - The text to display for the hyperlink (such as the attribute name)

GetDiagramHyperlink (string DiagramGUID, string LinkText)	String Notes: Returns a string containing a hyperlink to the diagram specified by the input parameter DiagramGUID. Parameters: • DiagramGUID: String - The GUID of the diagram for which a hyperlink is required • LinkText: String - The text to display for the hyperlink (such as the diagram name)
GetElementHyperlink (string ElementGUID, string LinkText)	String Notes: Returns a string containing a hyperlink to the element specified by the input parameter ElementGUID. Parameters: • ElementGUID: String - The GUID of the element for which a hyperlink is required • LinkText: String - The text to display for the hyperlink (such as the element name)
GetFileHyperlink (string FilePath, string LinkText)	String Notes: Returns a string containing a hyperlink to the file specified by the input parameter FilePath. Parameters: • FilePath: String - The path name of the file for which a hyperlink is required • LinkText: String - The text to display for the hyperlink (such as the file name)
GetLastError ()	String Notes: Returns the last error message set for the MailInterface.
GetMethodHyperlink (string MethodGUID, string LinkText)	String Notes: Returns a string containing a hyperlink to the method specified by the input parameter MethodGUID. Parameters: • MethodGUID: String - The GUID of the method for which a hyperlink is required • LinkText: String - The text to display for the hyperlink (such as the method name)
GetPackageHyperlink (string PackageGUID, string LinkText)	String Notes: Returns a string containing a hyperlink to the Package specified by the input parameter PackageGUID. Parameters: • PackageGUID: String - The GUID of the Package for which a hyperlink is required • LinkText: String - The text to display for the hyperlink (such as the Package name)
GetRecipientGUID (string UserName)	String Notes: Returns the GUID of the specified Enterprise Architect user. Parameters:

	• UserName: String - The name of a user defined in the current model
GetWebHyperlink (string URL, string LinkText)	String Notes: Returns a string containing a hyperlink to the URL specified by the input parameter URL. Parameters: • URL: String - The URL of the item for which a hyperlink is required • LinkText: String - The text to display for the hyperlink
SendMailMessage (string RecipientGUID, string Subject, messageflag Flag, string MessageText)	Boolean Notes: Creates and sends a new mail message using the values specified in the input parameters. Parameters: • RecipientGUID: String - The GUID of the addressee user (an Enterprise Architect user defined in the current model) • Subject: String - The Subject text to display for this message • Flag: MessageFlag - The flag type/color to attach to this message • MessageText: String - The text that is the body of the message

Search Window Package

The Search Window Package contains:

- The EAContext Class, which provides a description of a single selected item
- The EASelection Class, which provides optimized functions to access information about the current selection
- The SearchWindow Class, which provides a method for displaying the results of your operation using the Search Window

EAContext Class

The EAContext Class provides a description of a single selected item. The fields with values depend on the location of the selected item.

EAContext Attributes

Atttribute	Remarks
Alias	String Notes: Read only The Alias of the context item.
BaseType	String Notes: Read only Returns the base UML type of the context item.
ContextType	ContextType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
ElementGUID	String Notes: Read only The GUID of the current context object; empty if an object isn't selected. That is: • ElementGUID if an element has context • AttributeGUID if an attribute has context • MethodGUID if an operation has context. • DiagramGUID if a diagram has context • PackageGUID if a Package has context
ElementID	Long Notes: Read only The ID of the current context object; 0 if an object isn't selected. That is: • ElementID if an element has context • AttributeID if an attribute has context • MethodID if an operation has context. • DiagramID if a diagram has context • PackageID if a Package has context
Locked	Boolean Notes: Read only Indicates if the context item is locked.
MetaType	String Notes: Read only

	Returns the metatype of the context item.
Name	String Notes: Read only The name of the context item.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.

EAContext Methods

Method	Remarks
HasStereotype (String stereo)	Boolean Returns: True if the stereotype is applied to an object. Parameters • stereo: String - the stereotype to check against the context object, to see if has been applied

EASelection Class

The EASelection Class provides optimized functions to access information on the current selection. It should be used when building Add-In menus and setting the menu state, as almost all properties can be used without any database queries being made.

EASelection Attributes

Attribute	Remarks
Context	EAContext Notes: Describes the currently focused element without requiring any database calls.
ElementSet	Collection Notes: When the selection consists of one or more objects of type otElement, this provides a collection giving optimized access to all of those elements.
List	Collection Notes: For any window where multiple selection is supported, this provides a list describing the type of every selected element without requiring any database calls.
Location	String Notes: Provides the type of window that contains the current selection. Possible values are: • Calendar • Diagram • Dialog • Element List • Gantt • Model View • Browser window • Project View • Relationship Matrix • Reviews • Search • Specification Manager Further values could be added to this list in the future.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.

EASelection Methods

None.

SearchWindow Class

The SearchWindow Class provides a method for displaying the results of your operation using the Search Window.

SearchWindow Attributes

Attribute	Remarks
FieldChooserVisible	Boolean Shows or hides the search Field Chooser.
FiltersVisible	Boolean Shows or hides the search filters.
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.

SearchWindow Methods

Method	Remarks
AddColumn (string Name, long Width)	Adds the column into the current Search window. Returns the column number, or -1 on error. Parameters: • Name: String - Name of the column • Width: Long - Width of the column
AddRow (ObjectType ot, String ElementGUID, Long ElementID, String ClassType, VARIANT Values)	Returns the row inserted into the search. Parameters: • ot: ObjectType - the Object Type • ElementGUID: String - GUID of the element • ElementID: long - Object ID of the element • ClassType: String - the type of object • Values: an array of values
ClearGrouping ()	Clear all groupings in the search. Returns FALSE on error.
ClearSorting ()	Clear all column sorting in the search. Returns FALSE on error.
EnsureVisible ()	Make the Search window visible.

	Returns FALSE, if the Search window isn't open.
GetCell (long Row, long Column)	Returns the value of the cell. Parameters: • Row: long - Row number • Column: long - Column number
GroupByColumn (long Column)	Sets the group order by column. Returns FALSE if it cannot group by the specified column. Parameters: • Column: Long - Column number
LoadLayout (string LayoutGUID)	Set the layout of the Search window. Returns FALSE if the layout cannot be set. Parameters: • LayoutGUID: String - Layout GUID
NewLayout (string LayoutGUID)	Saves the layout of the Search window. Parameters: • LayoutGUID: String - Layout GUID
SetCellString (long Row, long Column, String Data)	Sets a value in a cell. Parameters: • Row: long - Row number • Column: long - Column number • Data: String - Value to set the cell to
SetCellVariant (long Row, long Column, VARIANT Data)	Sets an alternative value in a cell. Parameters: • Row : long - Row number • Column : long - Column number • Data: Value to set the cell to
SortByColumn (long Column)	Sets the column to sort by. Returns FALSE if it cannot sort by the specified column. Parameters: • Column: Long - Column number

Simulation Package

The Simulation Package contains:

- An attribute to set, increase and decrease the speed of the simulation
- A function to check if a simulation is currently running
- Functions to Start, Stop, Step Into, Step Out of, Step Over and Pause a simulation
- A function to send a broadcast signal to the simulation that is currently running

Simulation Class

The Simulation Class provides an interface to the Enterprise Architect Model Simulation facilities.

Simulation Attributes

Attribute	Description
ObjectType	ObjectType Notes: Read only Distinguishes objects referenced through a Dispatch interface.
Speed	Long Notes: Read/Write Retrieve or set the current simulation running speed.

Simulation Methods

Method	Description
BroadcastSignal(string sSignalName, string sParameters)	Boolean Notes: Send a signal into the running simulation. If the simulation is stopped, do nothing. Parameters: • sSignalName: String - the name of the signal OR the GUID of the Signal element • sParameters: String - a string of one or more signal parameters, in this format: {parameter1: 5, parameter2: "test", parameter3: 3.2}
IsSimulatorRunning()	Boolean Notes: Check the state of the simulation. Returns True if the simulation is running; returns False if the simulation is stopped.
Pause()	Boolean Notes: Pause the simulation if it is running.
Start()	Boolean Notes: Start the simulation based on the current selection. If the current simulation is in a paused state, then the simulation is resumed.
StepIn()	Boolean Notes: Step In to the routine in the current simulation.
StepOut()	Boolean

	Notes: Step Out of the routine in the current simulation.
StepOver()	Boolean Notes: Step Over the routine in the current simulation.
Stop()	Boolean Notes: Stop the simulation.

Schema Composer Package

The Schema Composer can be accessed from the Enterprise Architect automation interface. A client (script or Add-In) can obtain access to the interface using the SchemaComposer property of the Repository object. This interface is available when a Schema Composer has a profile loaded.

SchemaProperty Class

SchemaProperty Attributes

Attribute	Description
TypeID	long Notes: Read only The classifier ID of the property.
PropID	long Notes: Read only The property ID.
Guid	string Notes: Read only The unique model GUID of the property.
Name	string Notes: Read only The name of the property.
Cardinality	string Notes: Read only The cardinality of the element.
UMLType	string Notes: Read only The UML type, such as attribute, association or aggregation.
Parent	long Notes: Read only The classifier of the owner Class.
PrimitiveType	string Notes: Read only The property's primitive type if property represents a simple type.
Annotation	string Notes: Read only The model notes for the property.
Stereotype	string Notes: Read only The stereotype of the property.

Choices	SchemaTypeEnum Returns an iterator allowing navigation of choice elements in *model,* defined for this property in the Schema Composer. Combine with SchemaChoices attribute to obtain all available choices.
SchemaChoices	SchemaTypeEnum Returns an iterator allowing navigation of choice elements in *schema,* defined for this property in the Schema Composer. Combine with Choices attribute to obtain all available choices.
TypeName	string Returns a string naming the type of the property
Type	SchemaType Returns an interface to the property's type for complex types.

SchemaProperty Methods

Method	Description
IsInline	Boolean If true, the property is marked as 'Inline'. XML schema generators would emit an inline definition when detecting this attribute.
IsPrimitive	Boolean Returns true for a property whose type is maps to a built in type such as xs:integer, xs:string, xs:date or other XML Schema built-in type.
IsByReference	Boolean Returns true for a property marked as 'By Reference' in the profile.

SchemaProfile Class

The interface representing the technology governing the naming and design rules on which the schema is built.

SchemaProfile Methods

Method	Description
AddExportFormat(string description)	void Notes: Use this function to add entries that are offered by the Schema Composer when the user clicks on the Generate button. Parameters: • description: describes the export format provided by the Add-In
SetCapability(string name,boolean enabled)	void Notes: Use this function to enable/disable capabilities. Parameters: • name: name of the capability • enabled: True or False Capabilities: 'allowCardinality' - allows/denies restrictions to cardinality 'allowRootElement' - allows/denies setting root element 'allowPropByRef' - allows/denies By Reference restriction 'allowRedefine' - allows/denies ability to redefine an element
SetProperty(string name, string value)	void Notes: Sets properties displayed in the Schema Composer. Parameters: • name: property name • value: property value Properties: 'Namespace' - Target namespace for XML schema 'Namespace Prefix' - Namespace prefix for XML schema 'Qualifier' - string qualifier that prepends schema type names

SchemaComposer Class

The SchemaComposer Class provides the interface to the Enterprise Architect Schema Composer facility.

SchemaComposer Attributes

Attribute	Description
ModelReference	String Notes: The model ref listed in the Schema Composer for the current profile.
Namespace	String Notes: The namespace listed in the Schema Composer for the current profile.
NamespacePrefix	String Notes: The namespace prefix listed in the Schema Composer for the current profile.
TargetDirectory	String Notes: The target directory selected by the user after clicking on the Generate button.
SchemaName	String Notes: Returns the name of the schema profile currently being generated.
SchemaSet	String Notes: Returns the schema set used when the schema was created.
SchemaType	String Notes: The schema type listed in the Schema Composer for the current profile, either 'schema' or 'transform'.
SchemaTypes	SchemaTypeEnum Notes: Read only Enumerator for the type collection represented in the currently open schema.
Namespaces	SchemaNamespaceEnum Notes: Read only Enumerator for the namespaces referenced by schema

SchemaComposer Methods

Method	Description
FindInSchema(long	SchemaType

typeID)	Notes: Obtains an interface to a Class as represented in the schema for a given model Class ID. Parameters: • typeID: the model Class ID
FindInModel(long typeID)	ModelType Notes: Obtains an interface to a Class as represented in the UML model for a given model Class ID Parameters: • typeID: the model Class ID
FindSchemaTypeByName(string typename)	SchemaType Notes: Returns an interface to the schema type that matches the type specified or null if no type exists. Parameters: • name : the name of the type
GetNamespacePrefixForTy pe(long typeID)	String Notes: Returns the schema namespace prefix for a given type Parameters: • typeID: the model Class ID
GetNamespaceForPrefix(st ring prefix)	String Notes: Returns the URI for a given schema namespace prefix Parameters: • name: the namespace prefix

ModelTypeEnum Class

An enumerator interface for schema types as represented in the UML model.

ModelTypeEnum Methods

Method	Description
GetCount()	long Returns the number of types present in the collection.
GetFirst()	ModelType Returns the first type interface in a collection of types.
GetNext()	ModelType Returns the next type in the collection of types or null if end is reached.

ModelType Class

Provides an interface to the Class of a schema type as represented in the model.

ModelType Attributes

Attribute	Description
PropertyCount	long Notes: Read only The total number of properties for this Class available in the Properties collection.
Properties	SchemaPropEnum Notes: Enumerator Collection of properties for the Class as defined in the model.
TypeID	long Notes: Read only The Class ID of the type.
Guid	string Notes: Read only A GUID that uniquely identifies a type in the model.
Typename	string Notes: Read only The name of the type as represented in the model.
ClassifierPath	string Notes: Read only The qualified path of the type in the model.
ClassifierPathID	string Notes: Read only A GUID that uniquely identifies a ClassifierPath in the model.
Stereotype	string Notes: Read only The stereotype of the Class as defined in the model.
Annotation	string Notes: Read only Any notes present in the model describing the Class.

ModelType Methods

Method	Description
GetSuperClassEnum(SearchType searchtype)	ModelTypeEnum Notes: Enumerator Returns an enumerator that can be used to traverse the Class ancestry. Parameters: • searchtype: the type of traversal to use, breadth first or depth first
GetSubClassEnum(SearchType searchType)	ModelTypeEnum Notes: Enumerator Returns an enumerator that can be used to iterate over any descendents of the Class. Parameters: • searchtype: the type of traversal to use, breadth first or depth first
IsEnumeration	True where type represents an enumeration element

SchemaTypeEnum Class

An enumerator interface for schema types as represented in XML schema.

Methods

Method	Description
GetCount()	Returns the number of properties for an element.
GetFirst()	Returns the first property for the element in alphabetical order.
GetNext()	Returns the first property for the element in alphabetical order or null if no more are present.

SchemaType Class

Represents a type as it is defined in the schema.

Methods

Method	Description
GetFacet(BSTR name)	Returns the value of the named facet. 'Root', for example' returns a value indicating whether a type is a root element.
GetRestriction(BSTR guid)	Returns the restriction as a string for the property having the supplied guid.
IsRoot()	True if Class is marked as 'root' in the Composer.
IsEnumeration()	True if the type represents an enumeration element

Properties

Property	Description
PropertyCount [type: long]	Returns the number of properties held by 'type'.
Properties [type: IEASchemaPropEnum]	Returns an enumerator for 'type's' properties.
TypeID	The model Class ID.
Guid	The unique model GUID of the type.
Typename	The type's name.
Parent	The parent type - if any - that this Class extends. Could be null depending on composition method.

SchemaPropEnum Class

An enumerator for properties of a UML model type or XML schema type.

Methods

Method	Description
GetCount()	Returns the number of properties for an element.
GetFirst()	Returns the first property for the element in alphabetical order.
GetNext()	Returns the first property for the element in alphabetical order or null if no more are present.

SearchType Enumeration

SearchType Attributes

Attribute	Description
searchDepthFirst	Navigate children before siblings.
searchBreadthFirst	Navigate siblings before children.

SchemaNamespace Class

An interface presenting namespace information

SchemaNamespace Attributes

Name	string Notes: Read only The namespace prefix.
URI	string Notes: Read only The URI of the namespace.

SchemaNamespaceEnum Class

An enumerator interface for namespaces referenced by schema.

SchemaNamespaceEnum Methods

GetFirst()	SchemaNamespace Returns the first namespace interface in a collection of namespaces.
GetNext()	SchemaNamespace Returns the next namespace interface in a collection of namespaces

Code Samples

As you write or edit code for using the Automation Interface, you might want to review these public Object examples, written in VB.Net.

Examples

Name
Open the Repository
Iterate Through a .eap File
Add and Manage Packages
Add and Manage Elements
Add a Connector
Add and Manage Diagrams
Add and Delete Features
Element Extras
Repository Extras
Stereotypes
Work with Attributes
Work with Methods

Open the Repository

This is an example of the VB.Net code to open an Enterprise Architect repository.

```
Public Class AutomationExample
    "Class level variable for Repository
    Public m_Repository As Object

    Public Sub Run()
        try
            "create the repository object
            m_Repository = CreateObject("EA.Repository")

            "open an EAP file
            m_Repository.OpenFile("F:\Test\EAAuto.EAP")

            "use the Repository in any way required
            "DumpModel

            "close the repository and tidy up
            m_Repository.Exit()
            m_Repository = Nothing

        catch e as exception
            Console.WriteLine(e)
        End try
    End Sub
end Class
```

Iterate Through a .EAP File

This is an example of the VB.Net code to iterate through a .eap file starting at the Model level, after the repository has been opened.

```
Sub DumpModel()
    Dim idx as Integer
    For idx=0 to m_Repository.Models.Count-1
        DumpPackage("",m_Repository.Models.GetAt(idx))
    Next
End Sub

"output Package name, then element contents, then process child Packages
Sub DumpPackage(Indent as String, Package as Object)
    Dim idx as Integer
    Console.WriteLine(Indent + Package.Name)
    DumpElements(Indent + "", Package)

    For idx = 0 to Package.Packages.Count-1
        DumpPackage(Indent + "", Package.Packages.GetAt(idx))
    Next
End Sub

"dump element name
Sub DumpElements(Indent as String, Package as Object)
    Dim idx as Integer
    For idx = 0 to Package.Elements.Count-1
        Console.WriteLine(Indent + "::" + Package.Elements.GetAt(idx).Name)
    Next
End Sub
```

Add and Manage Packages

This example illustrates how to add a model or a Package to the project.

```
Sub TestPackageLifecycle
    Dim idx as integer
    Dim idx2 as integer
    Dim package as object
    Dim model as object
    Dim o as object

    "first add a new Model

    model = m_Repository.Models.AddNew("AdvancedModel","")
    If not model.Update() Then
        Console.WriteLine(model.GetLastError())
    End If

    "refresh the models collection
    m_Repository.Models.Refresh

    "now work through models collection and add a Package

    For idx = 0 to m_Repository.Models.Count -1
        o = m_Repository.Models.GetAt(idx)
        Console.WriteLine(o.Name)
        If o.Name = "AdvancedModel" Then
            package = o.Packages.Addnew("Subpackage","Nothing")
            If not package.Update() Then
                Console.WriteLine(package.GetLastError())
            End If

            package.Element.Stereotype = "system"
            package.Update

            "for testing purposes just delete the
            "newly created Model and its contents
            "m_Repository.Models.Delete(idx)

        End If
    Next
```

```
End Sub
```

Add and Manage Elements

This is an example of the code for adding and deleting elements in a Package.

```
Sub ElementLifeCycle
    Dim package as Object
    Dim element as Object

    package = m_Repository.GetPackageByID(2)
    element = package.elements.AddNew("Login to Website","UseCase")
    element.Stereotype = "testcase"
    element.Update
    package.elements.Refresh()

    Dim idx as integer

    "Note the repeated calls to "package.elements.GetAt."
    "In general you should make this call once and assign to a local
    "variable - in this example, Enterprise Architect loads the
    "element required every time a call is made - rather than loading once
    "and keeping a local reference.

    For idx = 0 to package.elements.count-1
        Console.WriteLine(package.elements.GetAt(idx).Name)
        If (package.elements.GetAt(idx).Name = "Login to Website" and _
            package.elements.GetAt(idx).Type = "UseCase") Then
                package.elements.deleteat(idx, false)
        End If
    Next
End Sub
```

Add a Connector

This is an example of code to add a connector and set its values.

```
Sub ConnectorTest
    Dim source as object
    Dim target as object
    Dim con as object
    Dim o as object

    Dim client as object
    Dim supplier as object

    "Use ElementIDs to quickly load an element in this example
    "... you must find suitable IDs in your model

    source = m_Repository.GetElementByID(129)
    target = m_Repository.GetElementByID(169)

    con = source.Connectors.AddNew ("test link 2", "Association")

    "again, replace ID with a suitable one from your model
    con.SupplierID = 169

    If not con.Update Then
        Console.WriteLine(con.GetLastError)
    End If
    source.Connectors.Refresh

    Console.WriteLine("Connector Created")

    o = con.Constraints.AddNew ("constraint2","type")
    If not o.Update Then
        Console.WriteLine(o.GetLastError)
    End If

    o = con.TaggedValues.AddNew ("Tag","Value")
    If not o.Update Then
        Console.WriteLine(o.GetLastError)
    End If
```

```
    "Use the client and supplier ends to set
    "additional information

    client = con.ClientEnd
    client.Visibility = "Private"
    client.Role = "m_client"
    client.Update
    supplier = con.SupplierEnd
    supplier.Visibility = "Protected"
    supplier.Role = "m_supplier"
    supplier.Update

    Console.WriteLine("Client and Supplier set")

    Console.WriteLine(client.Role)
    Console.WriteLine(supplier.Role)

End Sub
```

Add and Manage Diagrams

This is an example of the code for creating a diagram and adding an element to it. Note the optional use of the element rectangle setting, using left, right, top and bottom dimensions in the AddNew call.

```
Sub DiagramLifeCycle

    Dim diagram as object
    Dim v as object
    Dim o as object
    Dim package as object

    Dim idx as Integer
    Dim idx2 as integer

    package = m_Repository.GetPackageByID(5)

    diagram = package.Diagrams.AddNew("Logical Diagram","Logical")
    If not diagram.Update Then
        Console.WriteLine(diagram.GetLastError)
    End if

    diagram.Notes = "Hello there this is a test"
    diagram.update()

    o = package.Elements.AddNew("ReferenceType","Class")
    o.Update

    " add element to diagram - supply optional rectangle co-ordinates

    v = diagram.DiagramObjects.AddNew("l=200;r=400;t=200;b=600;","")
    v.ElementID = o.ElementID
    v.Update

    Console.WriteLine(diagram.DiagramID)

End Sub
```

Add and Delete Features

An example of code to add and delete Features of an object.

```
Dim element as object
Dim idx as integer
Dim attribute as object
Dim method as object

'just load an element by ID - you must
'substitute a valid ID from your model
element = m_Repository.GetElementByID(246)

"create a new method
method = element.Methods.AddNew("newMethod", "int")
method.Update
element.Methods.Refresh
'now loop through methods for Element - and delete our addition
For idx = 0 to element.Methods.Count-1
    method =element.Methods.GetAt(idx)
    Console.Writeline(method.Name)
    If(method.Name = "newMethod") Then
        element.Methods.Delete(idx)
    End if
Next

'create an attribute
attribute = element.attributes.AddNew("NewAttribute", "int")
attribute.Update
element.attributes.Refresh

'loop through and delete our new attribute
For idx = 0 to element.attributes.Count-1
    attribute =element.attributes.GetAt(idx)
    Console.Writeline(attribute.Name)
    If(attribute.Name = "NewAttribute") Then
        element.attributes.Delete(idx)
    End If
Next
```

Element Extras

These are examples of code to access and use element extras, such as scenarios, constraints and requirements.

```
Sub ElementExtras
    Dim element as object
    Dim o as object
    Dim idx as Integer
    Dim bDel as boolean
    bDel = true

    try
        element = m_Repository.GetElementByID(129)

        'manage constraints for an element
        'demonstrate addnew and delete
        o = element.Constraints.AddNew("Appended","Type")
        If not o.Update Then
            Console.WriteLine("Constraint error:" + o.GetLastError())
        End if
        element.Constraints.Refresh
        For idx = 0 to element.Constraints.Count -1
            o = element.Constraints.GetAt(idx)
            Console.WriteLine(o.Name)
            If(o.Name="Appended") Then
                If bDel Then element.Constraints.Delete (idx)
            End if
        Next

        'efforts
        o = element.Efforts.AddNew("Appended","Type")
        If not o.Update Then
            Console.WriteLine("Efforts error:" + o.GetLastError())
        End if
        element.Efforts.Refresh
        For idx = 0 to element.Efforts.Count -1
            o = element.Efforts.GetAt(idx)
            Console.WriteLine(o.Name)
            If(o.Name="Appended") Then
                If bDel Then element.Efforts.Delete (idx)
            End if
```

```
Next

'Risks
o = element.Risks.AddNew("Appended","Type")
If not o.Update Then
    Console.WriteLine("Risks error:" + o.GetLastError())
End if
element.Risks.Refresh
For idx = 0 to element.Risks.Count -1
    o = element.Risks.GetAt(idx)
    Console.WriteLine(o.Name)
    If(o.Name="Appended") Then
        If bDel Then element.Risks.Delete (idx)
    End if
Next

'Metrics
o = element.Metrics.AddNew("Appended","Change")
If not o.Update Then
    Console.WriteLine("Metrics error:" + o.GetLastError())
End if
element.Metrics.Refresh
For idx = 0 to element.Metrics.Count -1
    o = element.Metrics.GetAt(idx)
    Console.WriteLine(o.Name)
    If(o.Name="Appended") Then
        If bDel Then element.Metrics.Delete (idx)
    End if
Next

'TaggedValues
o = element.TaggedValues.AddNew("Appended","Change")
If not o.Update Then
    Console.WriteLine("TaggedValues error:" + o.GetLastError())
End if
element.TaggedValues.Refresh
For idx = 0 to element.TaggedValues.Count -1
    o = element.TaggedValues.GetAt(idx)
    Console.WriteLine(o.Name)
    If(o.Name="Appended") Then
        If bDel Then element.TaggedValues.Delete (idx)
    End if
```

```
Next

'Scenarios
o = element.Scenarios.AddNew("Appended","Change")
If not o.Update Then
    Console.WriteLine("Scenarios error:" + o.GetLastError())
End if
element.Scenarios.Refresh
For idx = 0 to element.Scenarios.Count -1
    o = element.Scenarios.GetAt(idx)
    Console.WriteLine(o.Name)
    If(o.Name="Appended") Then
        If bDel Then element.Scenarios.Delete (idx)
    End if
Next

'Files
o = element.Files.AddNew("MyFile","doc")
If not o.Update Then
    Console.WriteLine("Files error:" + o.GetLastError())
End if
element.Files.Refresh
For idx = 0 to element.Files.Count -1
    o = element.Files.GetAt(idx)
    Console.WriteLine(o.Name)
    If(o.Name="MyFile") Then
        If bDel Then element.Files.Delete (idx)
    End if
Next

'Tests
o = element.Tests.AddNew("TestPlan","Load")
If not o.Update Then
    Console.WriteLine("Tests error:" + o.GetLastError())
End if
element.Tests.Refresh
For idx = 0 to element.Tests.Count -1
    o = element.Tests.GetAt(idx)
    Console.WriteLine(o.Name)
    If(o.Name="TestPlan") Then
        If bDel Then element.Tests.Delete (idx)
    End if
```

```
        Next

        'Defect
        o = element.Issues.AddNew("Broken","Defect")
        If not o.Update Then
            Console.WriteLine("Issues error:" + o.GetLastError())
        End if
        element.Issues.Refresh
        For idx = 0 to element.Issues.Count -1
            o = element.Issues.GetAt(idx)
            Console.WriteLine(o.Name)
            If(o.Name="Broken") Then
                If bDel Then element.Issues.Delete (idx)
            End if
        Next

        'Change
        o = element.Issues.AddNew("Change","Change")
        If not o.Update Then
            Console.WriteLine("Issues error:" + o.GetLastError())
        End if
        element.Issues.Refresh
        For idx = 0 to element.Issues.Count -1
            o = element.Issues.GetAt(idx)
            Console.WriteLine(o.Name)
            If(o.Name="Change") Then
                If bDel Then element.Issues.Delete (idx)
            End if
        Next

    catch e as exception
        Console.WriteLine(element.Methods.GetLastError())
        Console.WriteLine(e)
    End try

End Sub
```

Repository Extras

These are examples of code for accessing repository collections for system-level information.

```
Sub RepositoryExtras

    Dim o as object
    Dim idx as integer

    'issues
    o = m_Repository.Issues.AddNew("Problem","Type")
    If(o.Update=false) Then
        Console.WriteLine (o.GetLastError())
    End if
    o = nothing
    m_Repository.Issues.Refresh
    For idx = 0 to m_Repository.Issues.Count-1
        Console.Writeline(m_Repository.Issues.GetAt(idx).Name)
        If(m_Repository.Issues.GetAt(idx).Name = "Problem") then
            m_Repository.Issues.DeleteAt(idx,false)
            Console.WriteLine("Delete Issues")
        End if
    Next

    "tasks
    o = m_Repository.Tasks.AddNew("Task 1","Task type")
    If(o.Update=false) Then
        Console.WriteLine ("error - " + o.GetLastError())
    End if
    o = nothing
    m_Repository.Tasks.Refresh
    For idx = 0 to m_Repository.Tasks.Count-1
        Console.Writeline(m_Repository.Tasks.GetAt(idx).Name)
        If(m_Repository.Tasks.GetAt(idx).Name = "Task 1") then
            m_Repository.Tasks.DeleteAt(idx,false)
            Console.WriteLine("Delete Tasks")
        End if
    Next

    "glossary
    o = m_Repository.Terms.AddNew("Term 1","business")
```

```
If(o.Update=false) Then
    Console.WriteLine ("error - " + o.GetLastError())
End if
o = nothing
m_Repository.Terms.Refresh
For idx = 0 to m_Repository.Terms.Count-1
    Console.Writeline(m_Repository.Terms.GetAt(idx).Term)
    If(m_Repository.Terms.GetAt(idx).Term = "Term 1") then
        m_Repository.Terms.DeleteAt(idx,false)
        Console.WriteLine("Delete Terms")
    End if
Next

'authors
o = m_Repository.Authors.AddNew("Joe B","Writer")
If(o.Update=false) Then
    Console.WriteLine (o.GetLastError())
End if
o = nothing
m_Repository.Authors.Refresh
For idx = 0 to m_Repository.authors.Count-1
    Console.Writeline(m_Repository.Authors.GetAt(idx).Name)
    If(m_Repository.authors.GetAt(idx).Name = "Joe B") then
        m_Repository.authors.DeleteAt(idx,false)
        Console.WriteLine("Delete Authors")
    End if
Next

o = m_Repository.Clients.AddNew("Joe Sphere","Client")
If(o.Update=false) Then
    Console.WriteLine (o.GetLastError())
End if
o = nothing
m_Repository.Clients.Refresh
For idx = 0 to m_Repository.Clients.Count-1
    Console.Writeline(m_Repository.Clients.GetAt(idx).Name)
    If(m_Repository.Clients.GetAt(idx).Name = "Joe Sphere") then
        m_Repository.Clients.DeleteAt(idx,false)
        Console.WriteLine("Delete Clients")
    End if
Next
```

```
    o = m_Repository.Resources.AddNew("Joe Worker","Resource")
    If(o.Update=false) Then
        Console.WriteLine (o.GetLastError())
    End if
    o = nothing
    m_Repository.Resources.Refresh
    For idx = 0 to m_Repository.Resources.Count-1
        Console.Writeline(m_Repository.Resources.GetAt(idx).Name)
        If(m_Repository.Resources.GetAt(idx).Name = "Joe Worker") then
            m_Repository.Resources.DeleteAt(idx,false)
            Console.WriteLine("Delete Resources")
        End if
    Next

End Sub
```

Stereotypes

This is some example code for adding and deleting stereotypes.

```
Sub TestStereotypes

    Dim o as object
    Dim idx as integer

    "add a new stereotype to the Stereotypes collection
    o = m_Repository.Stereotypes.AddNew("funky","class")
    If(o.Update=false) Then
        Console.WriteLine (o.GetLastError())
    End if
    o = nothing

    "make sure you refresh
    m_Repository.Stereotypes.Refresh

    "then iterate through - deleting our new entry in the process
    For idx = 0 to m_Repository.Stereotypes.Count-1
        Console.Writeline(m_Repository.Stereotypes.GetAt(idx).Name)
        If(m_Repository.Stereotypes.GetAt(idx).Name = "funky") then
            m_Repository.Stereotypes.DeleteAt(idx,false)
            Console.WriteLine("Delete element")
        End if
    Next

End Sub
```

Work With Attributes

This is an example of code for working with attributes.

```
Sub AttributeLifecycle

    Dim element as object
    Dim o as object
    Dim t as object
    Dim idx as Integer
    Dim idx2 as integer
    try
        element = m_Repository.GetElementByID(129)

        For idx = 0 to element.Attributes.Count -1

            Console.WriteLine("attribute=" + element.Attributes.GetAt(idx).Name)

            o = element.Attributes.GetAt(idx)
            t = o.Constraints.AddNew("> 123","Precision")
            t.Update()
            o.Constraints.Refresh
            For idx2 = 0 to o.Constraints.Count-1
                t = o.Constraints.GetAt(idx2)
                Console.WriteLine("Constraint: " + t.Name)
                If(t.Name="> 123") Then
                    o.Constraints.DeleteAt(idx2, false)
                End if
            Next

            For idx2 = 0 to o.TaggedValues.Count-1
                t = o.TaggedValues.GetAt(idx2)
                If(t.Name = "Type2") Then
                    'Console.WriteLine("deleteing")
                    o.TaggedValues.DeleteAt(idx2, true)
                End if
            Next

            t = o.TaggedValues.AddNew("Type2","Number")
            t.Update
            o.TaggedValues.Refresh
```

```
            For idx2 = 0 to o.TaggedValues.Count-1
                t = o.TaggedValues.GetAt(idx2)
                Console.WriteLine("Tagged Value: " + t.Name)
            Next

            If(element.Attributes.GetAt(idx).Name = "m_Tootle") Then
                Console.WriteLine("delete attribute")
                element.Attributes.DeleteAt(idx, false)
            End If

        Next

    catch e as exception
        Console.WriteLine(element.Attributes.GetLastError())
        Console.WriteLine(e)
    End try
End Sub
```

Work With Methods

This is an example of code for working with the Methods collection of an element and with Method collections.

```
Sub MethodLifeCycle

    Dim element as object
    Dim method as object
    Dim t as object
    Dim idx as Integer
    Dim idx2 as integer

    try
        element = m_Repository.GetElementByID(129)

        For idx = 0 to element.Methods.Count -1
            method = element.Methods.GetAt(idx)
            Console.WriteLine(method.Name)

            t = method.PreConditions.AddNew("TestConstraint","something")
            If t.Update = false Then
                Console.WriteLine("PreConditions: " + t.GetLastError)
            End if

            method.PreConditions.Refresh
            For idx2 = 0 to method.PreConditions.Count-1
                t = method.PreConditions.GetAt(idx2)
                Console.WriteLine("PreConditions: " + t.Name)
                If t.Name = "TestConstraint" Then
                    method.PreConditions.DeleteAt(idx2,false)
                End If
            Next

            t = method.PostConditions.AddNew("TestConstraint","something")
            If t.Update = false Then
                Console.WriteLine("PostConditions: " + t.GetLastError)
            End if

            method.PostConditions.Refresh
            For idx2 = 0 to method.PostConditions.Count-1
                t = method.PostConditions.GetAt(idx2)
```

```
            Console.WriteLine("PostConditions: " + t.Name)
            If t.Name = "TestConstraint" Then
                method.PostConditions.DeleteAt(idx2, false)
            End If
        Next

        t = method.TaggedValues.AddNew("TestTaggedValue","something")
        If t.Update = false Then
            Console.WriteLine("Tagged Values: " + t.GetLastError)
        End if

        For idx2 = 0 to method.TaggedValues.Count-1
            t = method.TaggedValues.GetAt(idx2)
            Console.WriteLine("Tagged Value: " + t.Name)
            If(t.Name= "TestTaggedValue") Then
                method.TaggedValues.DeleteAt(idx2,false)
            End If
        Next

        t = method.Parameters.AddNew("TestParam","string")
        If t.Update = false Then
            Console.WriteLine("Parameters: " + t.GetLastError)
        End if

        method.Parameters.Refresh
        For idx2 = 0 to method.Parameters.Count-1
            t = method.Parameters.GetAt(idx2)
            Console.WriteLine("Parameter: " + t.Name)
            If(t.Name="TestParam") Then
                method.Parameters.DeleteAt(idx2, false)
            End If
        Next

        method = nothing
    Next
catch e as exception
    Console.WriteLine(element.Methods.GetLastError())
    Console.WriteLine(e)
End try

End Sub
```