ENTERPRISE ARCHITECT

用户指南系列

软件工程

Author: Sparx Systems

Date: 2022-08-30

Version: 16.0

创建于 ENTERPRISE ARCHITECT

目录

软件工程 7
开始 9
示例图表 11
综合发展 12
特征概述 14
生成源代码 15
生成一个单类 17
生成一组类 18
生成一个包 19
更新包内容 21
同步模型和代码 23
命名空间 24
导入源代码 25
导入项目 27
导入源代码 29
注记on源代码导入 30
导入资源脚本 32
导入目录结构 34
导入二进制模块 36
导入过程中找不到类 37
编辑源代码 38
支持的语言 41
配置文件关联 42
比较编辑 43
代码编辑器工具栏 44
代码编辑器上下文菜单 46
创建用例for方法 49
代码编辑器函数 50
函数详情 51
智能感知 53
查找和替换 55
在文件中搜索 58
查找文件 60
搜索智能感知 62
代码编辑器键绑定 64
应用模式（模型+代码） 68
MDG集成和代码工程 70
行为模型代码生成 71
代码生成 -活动图表 73
代码图表-交互 74
代码生成 -状态机 75
旧版状态机模板 79
Java Code Generated From Legacy状态机模板 81
状态机建模For HDLs 87
Win32用户接口对话框 89
建模UI对话框 91
从 RC文件导入单个对话框 93

从 RC文件导入所有对话框 94
导出对话框到 RC文件 95
设计一个新的对话框 96
四人帮 (GoF)模式 99
ICONIX 101
配置设置 103
源代码工程选项 104
代码生成选项 106
导入部件 107
源代码选项 108
选项-代码编辑器 110
编辑器语言属性 112
选项-物件Lifetimes 114
选项 - 属性/操作 115
建模约定 116
ActionScript 约定 118
Ada 2012 年公约 120
C 约定 123
C语言中的面向物件编程 125
C# 约定 127
C++ 约定 130
托管 C++ 约定 133
C++/CLI 约定 134
德尔福约定 136
Java约定 138
AspectJ 约定 140
PHP 约定 141
Python 约定 143
SystemC 约定 144
VB.NET 约定 146
Verilog 约定 149
VHDL 约定 151
Visual Basic 约定 154
语言选项 156
ActionScript 选项 -用户 158
ActionScript 选项 -模型 159
Ada 2012 选项-用户 160
Ada 2012 选项-模型 161
ArcGIS 选项 -用户 162
ArcGIS 选项 -模型 163
C 选项 -用户 164
C选项-模型 165
C# 选项 -用户 166
C# 选项 -模型 167
C++ 选项 -用户 168
C++ 选项-模型 169
Delphi Options -用户 171
Delphi Options -模型 172
德尔福属性 173
Java选项 -用户 174
Java选项 -模型 175

MySQL选项 -用户 176
MySQL选项 -模型 177
PHP 选项 -用户 178
PHP 选项-模型 179
Python 选项 -用户 180
Python 选项 -模型 181
SystemC 选项 -用户 182
SystemC 选项-模型 183
Teradata 选项 -用户 184
Teradata Options -模型 185
VB.NET 选项 -用户 186
VB.NET 选项-模型 187
Verilog 选项 -用户 188
Verilog 选项-模型 189
VHDL 选项 -用户 190
VHDL 选项 -模型 191
Visual Basic 选项 -用户 192
Visual Basic 选项 -模型 193
MDG 技术语言选项 194
重置选项 195
设置集合类 196
集合类的示例使用 197
本地路径 200
本地路径对话框 201
语言宏 203
开发编程语言 205
代码模板框架 206
代码模板定制化 207
代码和转换模板 208
基础模板 210
导出代码生成和变换模板 212
导入代码生成和变换模板 213
同步代码 214
同步现有部分 216
加新版块 217
加新特征和元素 218
代码模板编辑器 219
创建新的自定义模板 221
代码模板语法 222
文字文本 223
变量 224
宏 226
模板替换宏 228
字段替换宏 230
替换示例 231
属性字段替换宏 233
类字段替换宏 235
代码生成选项字段替换宏 238
连接器字段替换宏 242
约束字段替代宏 246
工作量字段替换宏 247

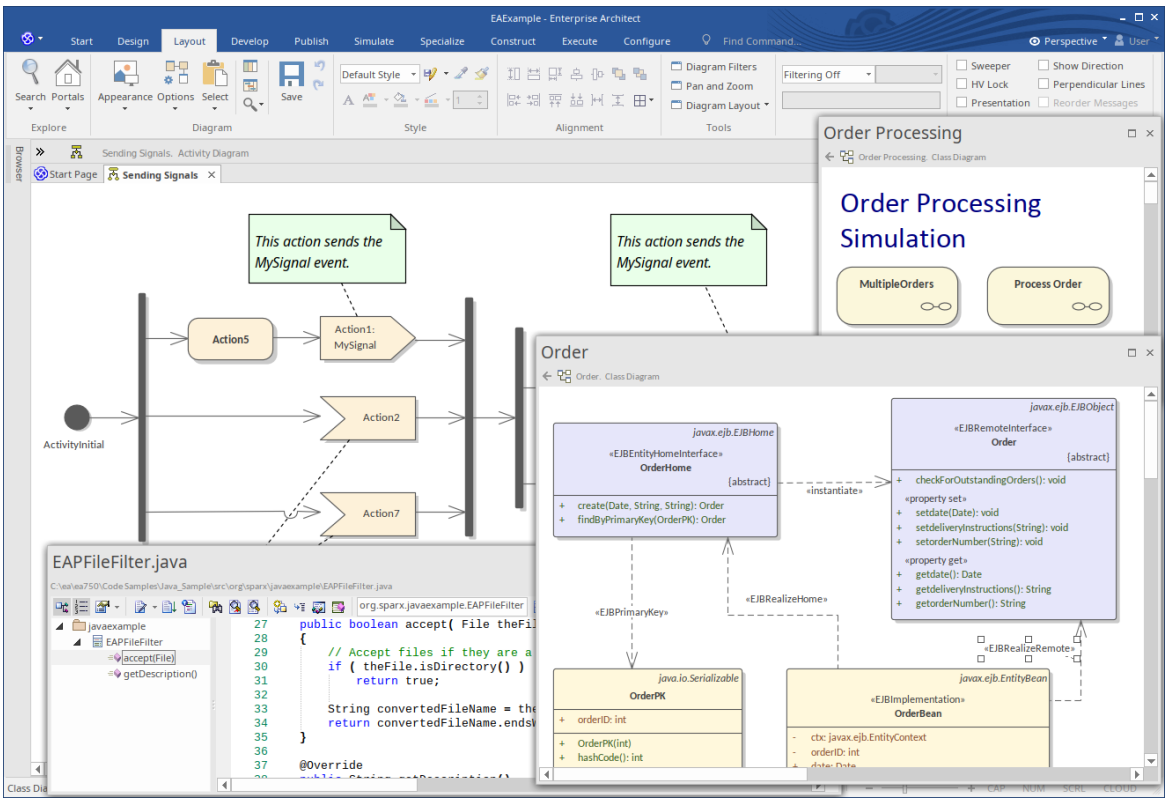
文件字段替换宏 248
文件导入字段替换宏 249
链接字段替换宏 250
链接文件字段替换宏 252
度量字段替换宏 253
操作字段替换宏 254
包字段宏 256
参数字段替换宏 257
问题字段替换宏 258
需求字段替换宏 259
资源字段替换宏 260
风险字段替换宏 261
字段替换宏 262
标记值替换宏 263
模板参数替换宏 265
测试字段替换宏 266
函数宏 267
控件宏 273
列表宏 274
分支宏 276
同步宏 278
处理指令 (PI) 宏 279
用于可执行状态机的代码生成宏 280
EASL 代码生成宏 292
EASL 系列 295
EASL属性 299
模板模板调用 306
MDG开发中的代码模板编辑器 307
创建自定义模板 308
自定义基础模板 310
加新Stereotyped模板 311
覆盖默认模板 312
语法框架 313
语法句法 314
语法说明 315
语法规则 316
语法术语 317
语法命令 318
AST 节点 320
编辑语法 328
解析 AST 结果 329
分析语法分析 330
宏编辑器 331
示例语法 332
代码分析器 333
代码矿工框架 341
代码矿工库 342
创建新的代码矿工数据库 345
Code Miner Queries 349
代码矿工查询语言 (mFQL) 350
mFQL 语言 351

集提取 358
设置遍历 360
设置加入 362
Sparx 英特尔服务 364
Sparx 英特尔服务配置 365
Sparx Intel 服务自动更新 370
服务配置 373
客户端配置- 将Enterprise Architect配置为使用代码矿工服务 374

软件工程

创建和管理有效且富有成效的软件结构和行为模型

软件工程是设计、实施和维护软件的学科。软件工程的过程以需求和约束作为输入开始，并产生部署到各种平台的编程代码和模式，从而创建运行系统。



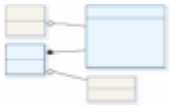
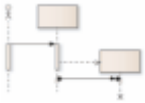
Enterprise Architect拥有一套丰富的工具和特征，可帮助软件工程师有效地执行工作并减少已实施解决方案中的错误数量。特征包括用于创建软件模型的设计工具、自动代码生成、源代码、二进制文件和模式的逆向工程，以及将源代码与设计模型同步的工具。程序代码可以直接在Enterprise Architect智能感知内的集成代码编辑器中查看和编辑，它提供了智能代码和其他特征来帮助编码。

环境的另一个引人注目的方面是能够将实现类追溯到设计元素和架构，然后追溯到需求和约束以及其他规范，并最终回到利益相关者及其目标和愿景。

Enterprise Architect支持广泛的编程语言和平台，并提供与两种最流行的集成开发环境的轻量级无缝集成：Visual Studio 和 Eclipse。此外，还有一个功能齐全的执行分析器，允许软件工程师在Enterprise Architect中设计、构建调试和测试软件模块。

功能

功能	描述
开发工具	发现具有出色工具和功能的紧密集成开发环境。

代码、编译和调试 	在建模环境中对应用程序进行模型、开发、调试、分析和管理。
执行代码的可视化分析 	通过可视化分析运行代码来了解您的代码库。使用测试点、分析和自动图表生成。
生成源代码	探索为单个类、选择的类或整个包生成源代码的一些方法。从结构或行为模型生成。
导入源代码	通过将源代码导入Enterprise Architect来检查现有系统。视图并修改对话框定义。将模型与源代码的最新更新同步。

开始

配置设置

选择蓝图

Enterprise Architect将工具的广泛特征划分为蓝图，确保您可以聚焦于特定任务并使用您需要的工具，而不会分散其他特征的注意力。要使用软件模型特征，您首先需要选择以下蓝图：

软件工程集：

<透视名称> >软件工程> 代码工程

<透视名称> >软件工程> GoF模式

<透视名称> >软件工程> ICONIX

UX设计集：

<透视名称> > UX设计>赢32 UI模型

设置蓝图可确保默认情况下可以使用案例管理模型和符号图、它们的工具箱和蓝图的其他特征。

示例图表

示例图提供了对该主题的可视化介绍，并允许您查看一些重要元素和连接器，这些元素和连接器用于指定或描述用于软件可视化的类以及与各种编程语言之间的正向和反向工程.

综合发展

在本主题中，您将学习如何使用功能齐全的集成开发环境。您将学习如何在丰富的代码编辑器中创建软件工件的结构和行为模型、生成和逆向工程代码、自定义代码生成方式、运行分析器脚本以优化代码、使用调试器和设置单元测试等等。

行为模型

行为模型

在本主题中，您将学习如何直接从行为图生成软件、系统和硬件描述语言的代码，包括：状态机、序列和活动图表。这将为您使用软件和工程系统的方式增加新的维度和精度。

四人帮 (GoF)模式

本主题介绍了著名的 23 种设计模式，这些设计模式汇集在一起，称为 Gang of Four (GoF) 模式，指的是它们的四位作者。您将获得软件工程师面临的常见问题的解决方案，并能够将这些模式注入您自己的模型中，从而为您的软件系统增加质量和严谨性。

Win32用户接口对话框

在本主题中，您将学习如何使用Enterprise Architect的用户接口建模功能，该功能允许您使用用户控件来模型用户界面屏幕。这些模型可以进行正向或反向工程，还可以为状态机和活动图模拟提供接口，允许它们接收和处理用户输入。

代码模板框架

在本主题中，您将学习如何使用代码模板框架来管理模型和转换为代码的方式。有一组标准模板，但您可以扩展这些模板以创建自己的模板并生成代码以满足您的需求。还有一些模板可以控制转换和数据库定义语言 (DDL.

语法框架

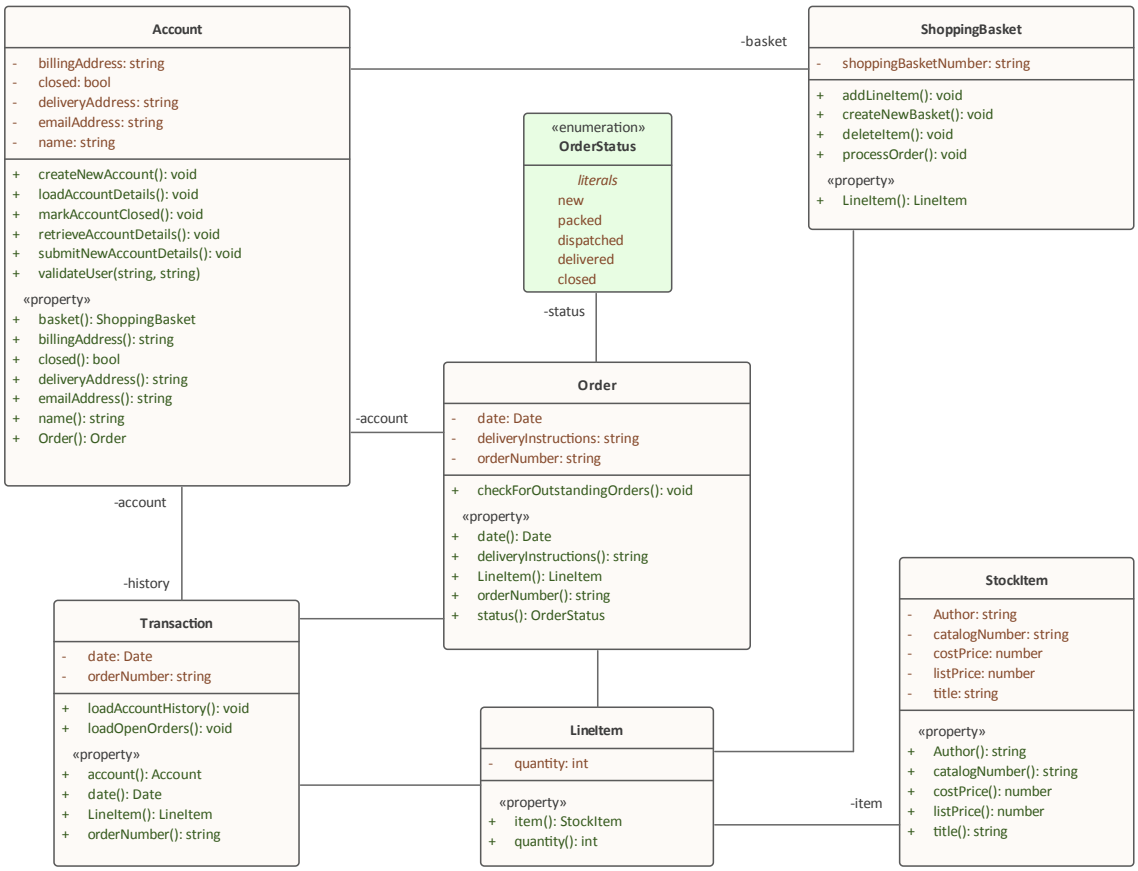
在本主题中，您将学习如何创建语法以将不受支持的编程语言转换为UML模型。 Enterprise Architect内置了对多种编程语言的支持，但如果您需要使用不受支持的语言，您可以使用语法框架来编写自己的解析器。该语法用于对文本形式的编程代码进行逆向工程，是代码模板框架的直接补充，您可以指定如何将不受支持的语言的UML模型转换为代码。

代码矿工框架

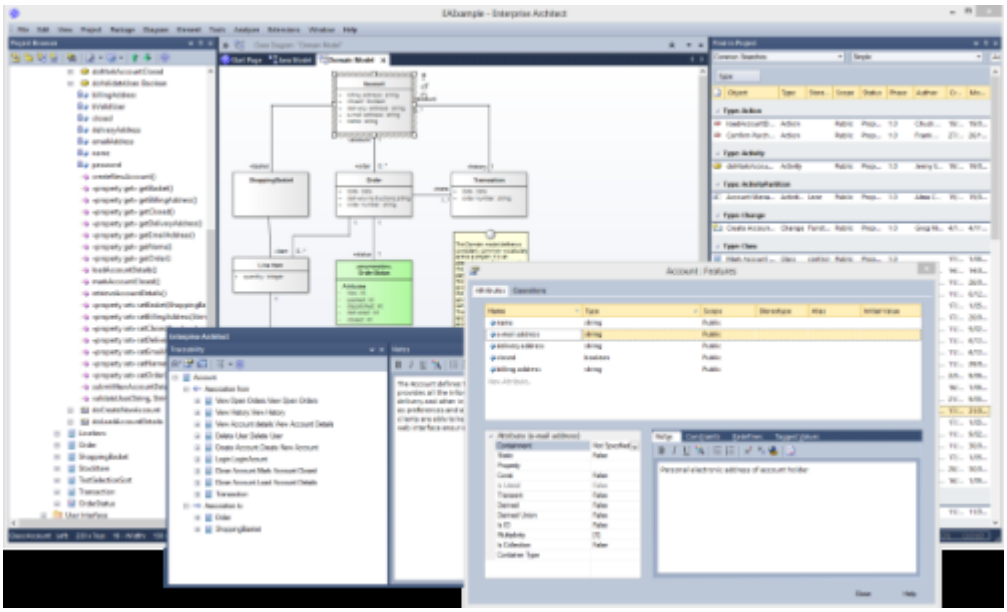
在本主题中，您将学习如何使用源代码数据库，它可以及时有效地访问隐藏在源代码中的数据。源代码被解析创建一个树结构，可用于分析程序结构、计算指标、跟踪关系甚至执行重构。

示例图表

软件图表允许您对模型的结构和行为进行建模，包括用户界面。 Enterprise Architect具有对建模软件的核心基础支持，并且该工具支持广泛的编程语言和范例。在此图中，我们看到用于模型在线商店的类，包括包含属性隔间的类和操作属性。枚举也被用于模型状态。



综合发展



Enterprise Architect为软件工程师提供了一套无与伦比的工具和特征，以帮助创建健壮且无错误的软件系统。工程师可以从定义架构开始，并确保它追溯到需求和规范。技术中性模型可以转换为针对全面的编程语言范围。模型驱动的开发环境适合各种技术。

特征

开发工具

- 使用一流的UML工具进行模型驱动的开发
- 生成和逆向工程代码
- 使用模板自定义代码生成
- 分析器管理应用程序的脚本
- 代码作者的代码编辑器
- 调试器来调查行为
- 用于可视化行为的分析器
- 记录行为的分析器
- 用于验证编程合同的测试点
- 与 jUnit 和 nUnit集成
- Eclipse 或 Visual Studio 在需要的地方集成

可追溯性

概括、实现、关联、依赖等的可追溯性一目了然。自定义关系视图。轻松导航模型中的相关元素。

用途

快速浏览所有图表中的元素用法。使用复杂的查询执行有效的元素搜索。

流行语言

- C/C++
- Java
- 微软.NET系列
- ADA
- Python

- Perl
- PHP

工具箱 为大量建模技术和编程语言提供了工具箱。

应用模式 Enterprise Architect为完成基本的应用程序类型提供了完整的启动项目，包括模型信息、代码和构建脚本。

特征概述

Enterprise Architect的代码工程广泛地涵盖了从您的UML模型中设计、生成和转换代码的各种过程。

特征

模型驱动代码工程

- 许多流行语言的源代码生成和逆向工程，包括 C++、C#、 Java 、Delphi、VB.Net、Visual Basic、ActionScript、Python 和 PHP
- A的 语法高亮"源代码编辑器
- 代码生成模板，可让您根据公司规范自定义生成的源代码

快速发展转型

- 使用转换模板的高级模型驱动架构（MDA）转换
- DDL、C#、 Java 、EJB 和 XSD 的内置转换
- 一个平台独立模型可用于生成和同步多个平台特定模型，显着提高生产力
- XSL 转换图、工具箱、编辑器和调试器。

可视化执行分析/调试、验证和可视化

- 执行构建、测试、调试、运行和部署脚本
- 将UML开发和建模与源开发和编译集成
- 使用 MDA 转换从源类生成NUnit 和 JUnit 测试类
- 将测试过程直接集成到Enterprise Architect IDE
- 调试.NET 、Mono、 Java和 Microsoft Native（C、C++ 和 Visual Basic）应用程序
- 基于契约式编程原则设计和执行测试套件
- XSL 样式表调试

数据库建模

Enterprise Architect使您能够：

- 来自许多流行 DBMS 的逆向工程，包括#
  服务器，我的#
  ,访问, PostgreSQL 和 Oracle
- 使用UML和内置数据建模配置文件的模型数据库库表、列、键、外键和复杂关系
- 前向生成 DDL 脚本以创建目标数据库结构

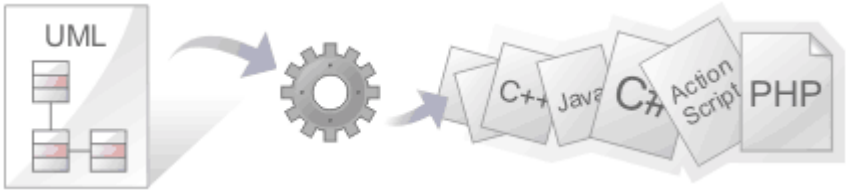
XML技术工程

Enterprise Architect使您能够快速模型、正向工程和反向工程两种关键的 W3C XML 技术：

- XML Schema (XSD)
- 网络服务定义语言（WSDL）

XSD 和 WSDL 支持对于开发完整的完成服务架构(SOA) 至关重要，而UML 2.5 和 XML 的耦合提供了在组织内实现基于 XML 的 SOA 工件的自然机制。

生成源代码



源代码生成是从UML模型创建编程代码的过程。采用这种方法有很大的好处，因为源代码包、类和接口是自动创建的，并使用变量和方法进行详细说明。

Enterprise Architect还可以从许多行为模型生成代码，包括状态机、序列和活动图。有一个高度灵活的模板机制，允许工程师完全定制生成源代码的方式，包括方法中的注释头和使用的集合类。

从工程和质量的角度来看，这种方法最引人注目的优势是UML模型以及因此架构和设计与编程代码同步。可以创建从目标、业务驱动程序和利益相关者的需求一直到编程代码中的方法的完整可追溯路径。

功能

功能	描述
语言	Enterprise Architect支持以下每种软件语言的代码生成： • 行动脚本 • 艾达 • ArcGIS • C • C#（适用于.NET 1 . 1 、 .NET 2.0 和.NET 4.0） • C++（标准，加上.NET托管的 C++ 扩展） • 德尔福 • Java （包括Java 1 .5、方面和泛型） • JavaScript • mFQL • mysql • PHP • Python • 兆数据# • 视觉基础 • Visual Basic .NET • 工作流脚本 您还可以使用以下语言生成硬件定义语言代码： • 高密度脂蛋白 • Verilog • 系统C
元素	代码是从类或接口模型元素生成的，因此您必须创建所需的类和接口元素才能从中生成。对代码有贡献的所有其他类型的元素（例如状态机或活动）必须是类的子元素。 添加属性（成为变量）和操作（成为方法）。代码中还支持约束和信号接

	收。
设置	在生成代码之前，您应该确保代码生成的默认设置符合您的要求；设置默认值以匹配您所需的语言和偏好。 您可以定义的首选项包括默认构造函数和析构函数、接口方法和创建语言的统一码选项。 Java等语言支持 "命名空间"，可以配置为指定命名空间根。 除了生成代码的默认设置外， Enterprise Architect还有助于为每种支持的语言设置特定的生成选项。
代码模板框架	代码模板框架 (CTF) 使您能够自定义Enterprise Architect生成源代码的方式，还可以生成Enterprise Architect不特别支持的语言。
本地路径	本地路径名使您能够用标签替换目录名。
行为准则	您还可以从三个UML行为建模范例生成软件代码： ● （交互序列图 ● 活动图 ● 状态机图（在 "任务"下的代码生成操作中使用 Legacy状态机模板） ● 状态机（使用可执行状态机工件）
实时代码生成	在 "开发 >源代码 > 选项"下拉菜单中，您可以选择在更改模型时立即更新源代码。
任务	生成代码时，您将执行以下一项或多项任务： ● 生成一个单类 ● 生成一组类 ● 生成一个包 ● 更新包内容

注记

- Enterprise Architect提供的大部分代码工程和调试工具在Enterprise Enterprise Architect专业版及更高版本中可用；统一版和终极版中提供了行为代码生成
- 启用安全性后，您需要访问权限 "生成源代码和 DDL"和 "从 DDL 和源代码逆向工程师"

生成一个单类

在为单个类生成代码之前，您：

- 模型元素（类或接口）的完全设计
- 创建到父级的继承连接器并关联到其他使用的类
- 创建继承连接器到您的类实现的接口；系统提供了一个选项来为类实现的所有接口方法生成函数存根

生成单个类的代码

节	行动
1	打开包含要为其生成代码的类或接口的图表。
2	单击所需的类或接口并选择 开发>源代码>生成>生成单个元素”功能区选项，或按 F11。 将显示 生成代码”对话框，您可以通过该对话框控制生成源代码的方式和位置。
3	在 路径”字段中，单击 ... 按钮并选择要生成的源代码的路径名称。
4	在 目标语言”字段中，单击下拉箭头并选择要生成的语言；这成为那个类的永久选项，所以如果你只用另一种语言做一遍，就把它改回来。
5	单击高级按钮。 显示 物件选项”对话框，在 首选项”对话框中提供 源代码工程”和代码语言选项页面的子集。
6	设置任何自定义选项（仅针对这个类），然后单击关闭按钮返回到 生成代码”对话框。
7	在 导入(s) / Header(s)”字段中，键入任何导入语句、#includes 或其他标题信息。 注记在 Visual Basic 的情况下，此信息将被忽略；在Java的情况下，两个导入文本框被合并；在 C++ 的情况下，第一个导入文本区域放在头文件中，第二个放在正文 (.cpp) 文件中。
8	单击生成按钮以创建源代码。
9	完成后，单击视图按钮查看已生成的内容。 注记您应该首先为每种语言类型设置默认查看器/编辑器；您还可以在 Preferences 窗口的 代码编辑器”页面上设置默认编辑器（ 开始>应用程序> 首选项 > 首选项 >源代码工程 >代码编辑器”）。

生成一组类

除了可以为单个类生成代码外，还可以选择一组类进行批量代码生成。执行此操作时，您接受集合中每个类的所有默认代码生成选项。

生成类组

节	细节
1	在图表中选择一组类和/或接口。
2	单击组中的一个元素并选择 开发>源代码>生成>生成选定元素”功能区选项（或按 Shift+F11）。 如果所选元素不存在代码，则会显示 另存为”对话框，您可以在其中指定每个代码文件的文件路径和名称；输入此信息并单击 保存”按钮。
3	将显示 批量生成”对话框，显示状态执行时的状态（进程可能太快而无法看到此对话框）。 如果所选类元素的代码已经存在，并且类名或结构已更改，则可能还会显示 同步元素<包名>.<元素名>”对话框；此对话框有助于同步模型和代码。

注记

- 如果选择的任何元素不是类或接口，则生成代码的选项不可用

生成一个包

除了从单个类和类组生成源代码之外，您还可以从一个包中生成代码。此特征提供了从子包递归生成代码并根据包层次结构自动生成目录结构的选项。这有助于您一步生成项目模型的整个分支的代码。

访问

功能区	开发>源代码>生成>生成全部
键盘快捷键	Ctrl+Alt+K

从包中生成代码，在生成包源代码对话框中

节	行动
1	在 同步"字段中，单击下拉箭头并选择适当的同步选项： ● 同步模型和代码"：具有现有文件的类代码与该文件前向同步；为显示的目标文件生成不存在文件的类的代码 ● 'Overwrite code'：所有选定的目标文件都被覆盖（向前生成） ● '不生成'：只为那些没有现有文件的选定类生成代码；所有其他类都被忽略
2	突出显示要为其生成代码的类；不选择任何不为其生成代码。 如果要在布局中显示更多信息，可以调整对话框及其列的大小。
3	要使Enterprise Architect根据包层次结构自动生成目录和文件名，请选中 自动生成文件"复选框；这将启用 根目录"字段，您可以在其中选择要在其下生成源目录的根目录。 默认情况下， 自动生成文件"特征忽略任何已经 类关联的文件路径；您还可以通过选中 保留现有文件路径"复选框来更改此行为。
4	要在输出中包含所有子包的代码，请选中 包含子包"复选框。
5	单击生成按钮开始生成代码。 随着代码生成的进行， Enterprise Architect会显示进度消息。如果某个类需要输出文件名，系统会提示您在适当的时间输入一个（假设未选择自动生成文件）。例如，如果选定的类包括部分类，则会显示一个提示以输入要为第二个部分类生成代码的文件名。

有关对话框选项的更多信息

选项	行动
根包	选择要为其生成代码的包的名称。

同步	选择指定如何重新生成现有文件的选项。
自动生成文件	指定Enterprise Architect是否应根据包层次结构自动生成文件名和目录。
根目录	如果选择自动生成文件，则显示创建生成的目录结构的路径。
保留现有文件路径	如果选择自动生成文件，请指定是否使用与类关联的现有文件路径。 如果未选择自动生成文件， Enterprise Architect将生成类代码以自动确定路径，而不管源文件是否已经与类相关联。
包括所有子包	还为列表中目标包的所有子包中的所有类生成代码。 此选项有助于递归生成给定包及其子包的代码。
选择要生成的对象	列出目标包下所有可用于代码生成的类；仅生成选定（突出显示）类的代码。 类与它们的目标源文件一起列出。
全选	将列表中的所有类标记为选中。
选择无	将列表中的所有课程标记为未选中。
生成	开始为所有选定的类生成代码。
取消	退出 "生成包源代码"对话框；不生成类代码。

更新包内容

除了生成和导入代码之外， Enterprise Architect还提供同步模型和源代码的选项，创建代表源代码中最新更改的模型，反之亦然。您可以使用模型作为源，也可以使用代码作为源。

同步的行为和动作取决于您在 "首选项"对话框的 "属性和操作"页面上选择的设置。使用这些设置，您可以保护或自动丢弃模型中代码中不存在的信息，并提示有关模型中不存在的代码特征的决定。在这两个示例中，已选择适当的复选框以最大限度地保护数据：

- 您生成了一些源代码，但随后对模型进行了更改；当您再次生成代码时， Enterprise Architect将任何新的属性或方法添加到现有的源代码中，而保留已经存在的内容，这意味着开发人员可以处理源代码，然后根据模型的需要生成其他方法，而无需他们的代码覆盖或销毁
- 您可能对源代码文件进行了更改，但模型有详细的注记和您不想丢失的特征；通过从源代码同步到模型中，您可以导入其他属性和方法，但不更改其他模型元素

使用同步方法，很容易使源代码和模型元素保持最新和同步。

访问

功能区	开发>源代码>同步>同步包

包内容与源代码同步

字段/按钮	行动
更新类型	根据需要选择单选按钮以对包类进行正向工程或反向工程。
在生成中包含子包	选中复选框以在同步中包含子包。
确定	单击按钮开始同步。 Enterprise Architect使用第一次导入/生成项目源时指定的目录名称，并根据选择的选项更新模型或源代码。如果： ● 执行前向同步 AND ● 模型和代码之间存在差异 AND ● 在 "选项 - 属性和操作"对话框中选中 "在前向同步时，提示删除代码特征不在模型中"复选框 然后显示 "同步元素<包名>.<元素名>"对话框。 否则，无需进一步操作。

注记

- 代码同步不会改变方法体；行为代码无法同步，代码生成仅在生成整个文件时有效
- 在Enterprise Architect的企业版、统一版和终极版中，如果启用了安全性，则必须具有 "生成源代码和DDL"权限才能将源代码 模型元素同步

同步模型和代码

您可以：

- 根据浏览器窗口中的模型同步一包类的代码，或
- 从模型中的一批类重新生成代码

在此类过程中，代码中可能存在模型中不存在的项目。

如果要捕获这些项目并手动解决它们，请在 选项 - 属性和操作"对话框中选中 在前向同步时，提示删除代码特征不在模型中"复选框，以便 同步元素<包名> .<元素名称>' 对话框显示，提供响应每个项目的选项。

同步项

按钮	细节
全选	突出显示并选择特征列中的所有项目。
全部清除	取消选择并删除特征列中所有项目的突出显示。
删除	将所选代码特征标记为要从代码中删除（行动列中的值更改为删除）。
重新分配	将选定的代码特征标记为要重新分配给模型中的元素。 这只有在存在代码中尚未定义的适当模型元素时才有可能。 将显示 选择相应的类特征"对话框，您可以从中选择要重新分配特征的类。单击确定按钮以标记要重新分配的特征。
忽视	将模型中不存在的选定代码元素标记为完全忽略（默认值；行动列中的值保持为 <none> 或更改为<none> ）。
重置为默认	将所选项目重置为 Ignore（行动列中的值更改为<none> ）。
确定	对项目进行分配的更改，然后关闭对话框。

命名空间

Java等语言支持包结构或名称空间。在Enterprise Architect中，您可以指定一个包作为命名空间根，它表示您的类模型的命名空间结构从哪里开始；命名空间根下的所有从属包将形成包含的类和接口的命名空间层次结构。

要将包定义为命名空间根，请单击浏览器窗口中的包并选择 开发 >源代码 > 选项 > 设置为命名空间的根”功能区选项。浏览器窗口中的包图标变为显示一个彩色角，表示该包是一个名称空间根。

比如生成的Java源代码，会在生成文件的开头自动添加一个包声明，指明类在包层级中的位置在namespace根下。

要清除现有的命名空间根，请单击浏览器窗口中的命名空间根包，然后取消选择 开发 >源代码 > 选项 > 设置为命名空间的根”功能区选项

要查看命名空间列表，请选择 设置 >参考> 设置 >命名空间根”功能区选项；将显示 命名空间”对话框。如果双击列表中的命名空间，该包会在浏览器窗口中突出显示；或者，右键单击命名空间并选择 定位包在浏览器”选项。

您也可以通过选择 清除命名空间属性”选项来清除选定的命名空间根。

要从命名空间定义中省略从属包，请选择 开发 >源代码 > 选项 > 抑制命名空间”功能区选项；要将包再次包含在命名空间中，请取消选择功能区选项。

注记

- 执行代码生成时，任何包含空格字符的包名都会被自动视为命名空间根

导入源代码

同时查看编程代码及其衍生模型的能力使系统设计更加清晰。 Enterprise Architect方便的代码工程特征之一是将源代码逆向工程成UML模型的能力。支持广泛A编程语言，并且有一些选项可以控制模型的生成方式。一旦代码在模型中，无论更改是直接在代码中还是模型本身中进行，都可以使其与模型保持同步。代码结构被映射到它们的UML表示中；例如， Java类被映射为UML类元素，变量被定义为属性，方法被建模为操作，以及由适当的连接器表示的Java类之间的交互。

将编程代码表示为模型结构有助于您更好地了解代码的结构以及它如何实现设计、架构和需求，以及最终如何交付业务价值。

需要注记的是，如果系统设计得不好，简单地将源导入Enterprise Architect并不能将其变成易于理解的UML模型。当使用设计不佳的系统时，通过检查从代码生成的单个模型包或元素来评估可管理单元中的代码是有用的；例如，将感兴趣的特定类拖到图表上，然后在一个级别使用 插入相关元素”选项来确定该类与其他类之间的直接关系。从这一点开始，可以创建标识源代码类之间交互的使用案例，提供应用程序操作的概述。

有几个选项指导如何对代码进行逆向工程，包括是否将注释导入注记以及如何格式化、如何识别属性方法以及是否为操作返回和参数类型创建依赖关系。

版权所有权

通常适合逆向工程的情况倾向于在以下源代码上运行：

- 你已经开发了
- 是您已获得使用权限的第三方库的一部分
- 是您的组织使用的框架的一部分
- 您的开发人员每天都在开发

如果您正在检查您或您的组织不拥有或没有复制和编辑特定权限的代码，您必须确保在开始逆向工程过程之前了解并遵守该代码的版权限制。

逆向工程支持的语言

语
行动脚本
Ada 2012（统一和终极版）
C
C #
C++
CORBA IDL（ MDG 技术）

德尔福
Java
PHP
Python
SystemC（统一和终极版）
Verilog（统一和终极版）
VHDL（统一和终极版）
视觉基础
Visual Basic .NET

注记

- Enterprise Architect专业版、企业版、统一版和终极版支持逆向工程
- 如果启用了安全性，您必须具有 从 DDL 和源代码进行反向工程"权限才能对源代码进行反向工程并根据代码同步模型元素
- 使用Enterprise Architect ，您还可以导入某些类型的二进制文件，例如Java .jar 文件和.NET PE 文件
- 目前可通过使用Sparx Systems网站的MDG 技术页面上列出的 MDG MDG 技术获得其他语言的逆向工程

导入项目

Enterprise Architect支持导入在 Visual Studio、Mono、Eclipse 和 NetBeans 中创作的软件项目。在Enterprise Architect中导入和处理项目有很多好处，不仅可以立即访问Enterprise Architect著名的建模工具和管理特征，还可以访问开发工具，例如模拟、调试和分析。

访问

功能区	开发>源代码>解决方案>导入a <项目类型>

导入Visual Studio 解决方案

此选项允许您从现有的 Visual Studio 解决方案文件或正在运行的 Visual Studio 实例导入一个或多个项目。该向导将为每个项目生成一个类模型，并为每个 Visual Studio 配置生成适当的分析器脚本。

导入Mono 解决方案

此选项允许您从解决方案文件导入 Mono 项目。显示的对话框与 "Visual Studio导入"对话框相同，但您可以选择针对 Linux 或窗口。该向导将为每个项目生成一个类模型并配置它们以进行调试。生成的分析器脚本引用 msbuild 来构建项目。

导入一个 Eclipse 项目

Eclipse 'Wizard' 可以对由其 Eclipse .project 文件和 ANT 构建描述的Java项目进行逆向工程。特征将为您选择的每个 ANT 目标生成UML类模型和分析器脚本。该过程还将为您通过 "向导"选择的每个调试协议生成一个脚本。您将看到适用于服务器的 JDWP（ Java调试有线协议）和适用于独立Java应用程序的 JVMTI（ Java虚拟机工具接口）的选择。这些脚本应该用于在Enterprise Architect中调试项目。

导入NetBeans 项目

NetBeans "向导"可以对由 NetBeans XML 项目文件和 ANT 构建描述的Java项目进行反向工程。 "向导"将为您选择的每个 ANT 目标创建项目的UML类模型和分析器脚本。该过程还将为您通过 "向导"选择的每个调试协议生成一个脚本。这些脚本应该用于在Enterprise Architect中调试项目。您将看到适用于服务器的 JDWP（ Java调试有线协议）和适用于独立Java应用程序的 JVMTI（ Java虚拟机工具接口）的选择。

导入选项

当您选择导入 Visual Studio 或 Mono 解决方案时，将显示 "Visual Studio 解决方案导入"对话框。按照此表中的指示完全字段。

当您选择导入 Eclipse 或 Netbeans 解决方案时，将显示相应的向导开始屏幕。按照每个屏幕上的提示指示浏览屏幕。

选项	描述
<项目列表>	选择解决方案文件后，解决方案中的项目将列在面板中。选择要由向导导入的项目。 您可以使用 All 按钮选择所有项目，使用 None 按钮清除项目选择。
选择解决方案文件	浏览并选择要从中导入的解决方案文件。 Mono 解决方案文件和 Visual Studio 解决方案文件具有 .sln 文件扩展名。
执行 Dry运行	选择此选项以干运行方式执行导入，以检查过程或输出中的任何错误，然后再重复导入以更改模型内容。单击视图日志按钮以检查导入log 。
创建包per文件	选择此选项以更精细的粒度执行导入，为每个文件创建一个单独的包。
导入	单击此按钮开始导入过程。
提示缺少宏定义	不适用于 Mono 解决方案导入。 对于 Visual Studio 中的 C++ 项目，解析器可能会遇到无法识别的宏。如果选择此选项，当发生此类事件时，系统会提示您，并有机会定义宏。如果不选择此选项，生成的类模型可能会缺少某些项目。
为每个包创建图表	选择后，将创建一个描述每个包的类模型的类图。结果是一个更大但更丰富多彩的模型。取消选择此选项将导致图表创建被跳过并且导入运行得更快。
生成分析器Scripts	对于 Visual Studio 解决方案，选择此选项将为每个项目配置生成分析器脚本，以及每个解决方案配置的脚本。脚本将允许在导入完成后立即构建和调试解决方案描述的程序。选择 窗口”复选框；如果不选择此选项，则不会配置执行分析器特征。 对于 Mono 解决方案，此选项允许您以 Linux 或窗口为目标。如果您选择 Linux，则假定运行Enterprise Architect的机器是 Linux，平台（ Java或 Mono）安装在那里，并且编译的程序在运行上运行。
启动项目	选择此选项后，该项目的脚本将成为模型的默认设置。调试工具、执行功能区和工具栏按钮将自动针对该程序。

导入源代码

您可以将源代码导入Enterprise Architect模型，对模块进行逆向工程。随着导入的进行， Enterprise Architect会提供进度信息。导入所有文件后， Enterprise Architect进行第二次传递以解决导入的类之间的关联和继承关系。

过程--导入源代码

节	行动
1	在浏览器窗口中，选择（或添加）要向其中导入类的图表。
2	单击图表背景，然后： ● 选择 开发 >源代码 > 文件"功能区选项并单击相应的语言，或 ● 如果显示代码生成工具栏，单击 导入"下拉箭头并选择要导入的语言 语言列表将包括您为其创建模型结构的任何自定义语言。
3	在出现的文件浏览器中，找到并选择一个或多个要导入的源代码文件。
4	单击 打开"按钮开始导入过程。

注记on源代码导入

您可以使用多种编程语言将代码导入Enterprise Architect项目。 Enterprise Architect支持每种编码语言的大多数结构和关键字。您为语言选择适当类型的源文件，作为要导入的源代码。

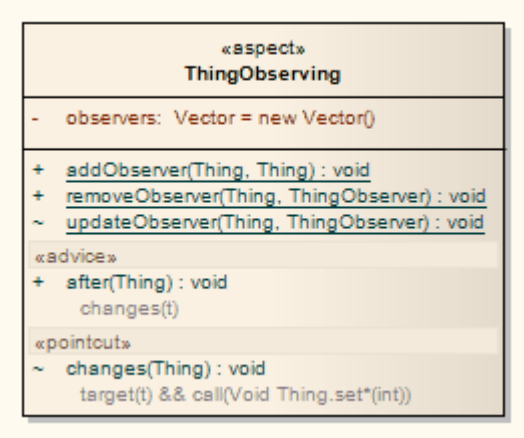
如果您认为缺少某个特定特征需要支持，请联系Sparx Systems 。

注记

- 当使用参数替换（模板化属性）对属性进行逆向工程时：
  - 如果找到具有适当模板参数定义的类，则为关联连接器
  创建并配置其参数替换
  - 如果关联条目被定义为 Collection类或
  在 附加集合类"选项中（对于 C#、C++ 和Java ）；例如，请参阅示例
  使用类的使用

编程语言注记

语	注记
动作脚本	合适的源文件类型：.as 代码文件。
C	合适的源文件类型：.h 头文件和/或 .c 文件。 当您选择头文件时， Enterprise Architect会根据 C 选项中指定的扩展名和搜索路径选项自动搜索要导入的相应 .c 实现文件。 Enterprise Architect不会扩展已使用的宏，这些必须添加到语言宏的内部列表中。
C++	合适的源文件类型：.h 头文件。 Enterprise Architect根据 C++ 选项中设置的扩展名和搜索路径自动搜索 .cpp 实现文件；当它找到实现文件时，它可以根据需要使用它来解析参数名称和方法注记。 导入 C++源代码时， Enterprise Architect会忽略函数指针声明。 要将它们导入您的模型，您可以创建一个 typedef 来定义函数指针类型，然后使用该类型声明函数指针；以这种方式声明的函数指针作为函数指针类型的属性导入。 Enterprise Architect不会扩展已使用的宏；这些必须添加到语言宏的内部列表中。
C #	合适的源文件类型：.cs。
德尔福	合适的源文件类型：.pas。
Java	合适的源文件类型：.java。 Enterprise Architect支持 AspectJ 语言扩展。

	 方面使用具有原型方面的类进行建模；这些方面可以像普通类一样包含属性和方法。 如果需要 intertype 属性或操作，您可以添加标签 "className"，其值是它所属的类的名称。 切入点被定义为具有构造型 <<pointcut>> 的操作，并且可以出现在任何 Java类、接口或方面；切入点的详细信息包含在方法的 "行为"字段中。 建议被定义为具有构造型 <<advice>> 的操作；此建议操作的切入点位于 "行为"字段中，并充当方法唯一签名的一部分。 afterAdvice 也可以有一个标记值的返回或抛出。
PHP	合适的源文件类型：.php、.php4 或 .inc。 如果启用条件语法，则嵌套。
Python	合适的源文件类型：.py。
视觉基础	合适的源文件类型：.cls类文件。
Visual Basic .NET	合适的源文件类型：.vb类文件。

导入资源脚本

Enterprise Architect支持 Microsoft 资源脚本（作为 .rc窗口文件）的导入和导出，其中包含应用程序图形用户界面的 Win32® 对话框定义（具有构造型 «win32Dialog» 的那些）。对话框资源被导入和导出用于一种特定的语言，默认为当前计算机系统的语言环境。

访问

功能区	开发>源代码>文件>导入资源脚本
键盘快捷键	F7（元素与代码同步）

从 .rc 文件导入对话框资源

选项	行动
资源文件	单击 ... 按钮并找到要从中导入屏幕元素的.rc 文件。
资源 ID	任何一个： • 保留默认值 "全部"以从文件中导入所有屏幕元素，或 • 单击下拉箭头并选择要导入的特定对话框的屏幕 ID
语	单击下拉箭头并选择要导入的对话框的语言版本（例如英语 - 美国）。
导入	单击此按钮可从资源文件中导入屏幕。 导入进度在 "语言"字段下方的字段中报告。

将对话框导出到 .rc 文件

选项	行动
屏幕ID	默认来自所选屏幕元素的屏幕标记值。 （如果对话框没有此 ID，则打开元素的 "属性"对话框的 "属性"页面并为 ID 标签提供一个值。）
资源文件	单击 ... 按钮并找到要将屏幕元素导出到其中的 .rc 文件。 如果之前已导入元素，则此字段默认为源文件。
语	单击下拉箭头并选择导出对话框的语言版本（例如英语 - 美国）。
导出	单击此按钮可从资源文件中导出屏幕。

	导出进度在 语言”字段下方的字段中报告。

注记

- 新对话框导出到现有的 .rc 文件
- 在导出到现有 .rc 文件时，不会从文件中删除任何对话框，即使它们已从模型中删除
- 在导入中，即使从原始 .rc 文件中省略，也不会从模型中删除任何对话框

导入目录结构

您可以从完成目录结构中的所有源文件中导入，这使您可以一次导入或同步目录树中的多个文件。Enterprise Architect在导入过程中创建必要的包和图表。

访问

功能区	开发>源代码>文件>导入源目录
键盘快捷键	Ctrl+Shift+U

使用 “导入源目录”对话框导入目录结构

字段	行动
根目录	类型输入或浏览要导入的目录的名称。
源类型	类型在或从下拉列表中选择源目录中要导入的文件的编码语言。
文件	类型在或从下拉列表中选择要包含在导入中的文件扩展名。使用a ';'分隔值。
执行 Dry运行	如果您想在单击确定按钮时以干运行的方式执行导入，请选中此复选框。处理完成后，单击视图日志按钮以检查处理的预测结果。
进程子目录	如果要在导入过程中包含子目录的内容，请选中此复选框。
从导入组件	如果要导入其他文件（如 “导入部件类型”对话框中所述），请选中此复选框。然后，您完成提示以指定组件的来源。
不要导入私人成员	如果要在导入库时从模型中排除私有成员，请选中此复选框。
提示缺少宏定义	在导入期间，解析器可能会遇到无法识别的宏。如果选中此复选框，当发生此类事件时，系统会提示您并有机会定义宏。如果您不选择此选项，则生成的包结构可能会缺少某些项目。
包结构	选择适当的单选按钮为每个目录、每个命名空间或每个文件创建一个包；这可能会受到所选源类型的限制。
为每个包创建图表	选中此复选框以在导入时创建的每个包中创建图表。单击 “选项”按钮以识别要包含在图表中的元素特征。
同步	选择适当的单选按钮以同步现有类或覆盖现有类。 如果找到与代码中匹配的模型类： ● “同步”更新模型类以包含代码中的详细信息，从而保留代码中未表示的信息，例如图表中类的位置

	• 'Overwrite' 删除模型类并从代码生成一个新的；不保留任何附加信息。 如果选择了 使用时间戳"选项，则具有最新时间戳（模型或代码）的表示将优先。
删除代码中找不到的类	选择适当的单选按钮以指定如何处理导入代码中不存在的现有模型类。 • 永不删除"保留模型中所有现有的类。 • 提示操作"使您能够单独审阅类 • 'Always' delete' 从模型中删除导入代码中不存在的任何类。
确定	单击此按钮开始导入。

导入二进制模块

Enterprise Architect使您能够对某些类型的二进制模块进行逆向工程。

访问

功能区	开发 >源代码 > 文件 >导入二进制模块

使用

目前允许的类型是：

- Java存档 (.jar)
- .NET PE 文件（.exe、.dll）- 不支持本机窗口和 EXE 文件，仅支持包含.NET程序集数据的 PE 文件
- 中间语言文件 (.il)

Enterprise Architect在导入过程中创建必要的包和图表；选择 不导入私人成员"复选框可将库中的私人成员排除在导入模型。

导入.NET文件时，可以通过反射或反汇编的方式导入，或者让系统选择最佳方法 - 这可能会导致两种类型都被使用。

基于反射的导入器依赖于.NET程序，并且需要安装.NET运行时环境。

基于反汇编程序的导入器依赖于名为 Ildasm.exe 的本机窗口程序，该程序是 MS .NET SDK 提供的工具； SDK 可以从微软网站下载。

A选择导入方式，因为有些文件不兼容反射（如mscorlib.dll），只能使用反汇编程序打开；但是，基于反射的导入器通常要快得多。

您还可以配置：

- 找到时是否同步或覆盖现有类；如果发现模型类与文件中的模型类匹配：
  - 同步更新模型类以包含文件中的详细信息，其中
  保留文件中未表示的信息，例如图表中类的位置
  - 覆盖删除模型类并从文件中生成一个新的模型类，删除并
  不替代附加信息
- 是否为每个包创建图表
- 导入创建的图表上显示的内容

导入过程中找不到类

从您的代码进行逆向工程时，有时可能会故意从您的源代码中删除类。

“导入源目录”功能跟踪它希望 之同步的类，并在“导入目录结构”对话框中提供如何处理未找到的类的选项。

您可以选择适当的选项使Enterprise Architect在导入结束时忽略丢失的类，自动删除它们或提示您管理它们。

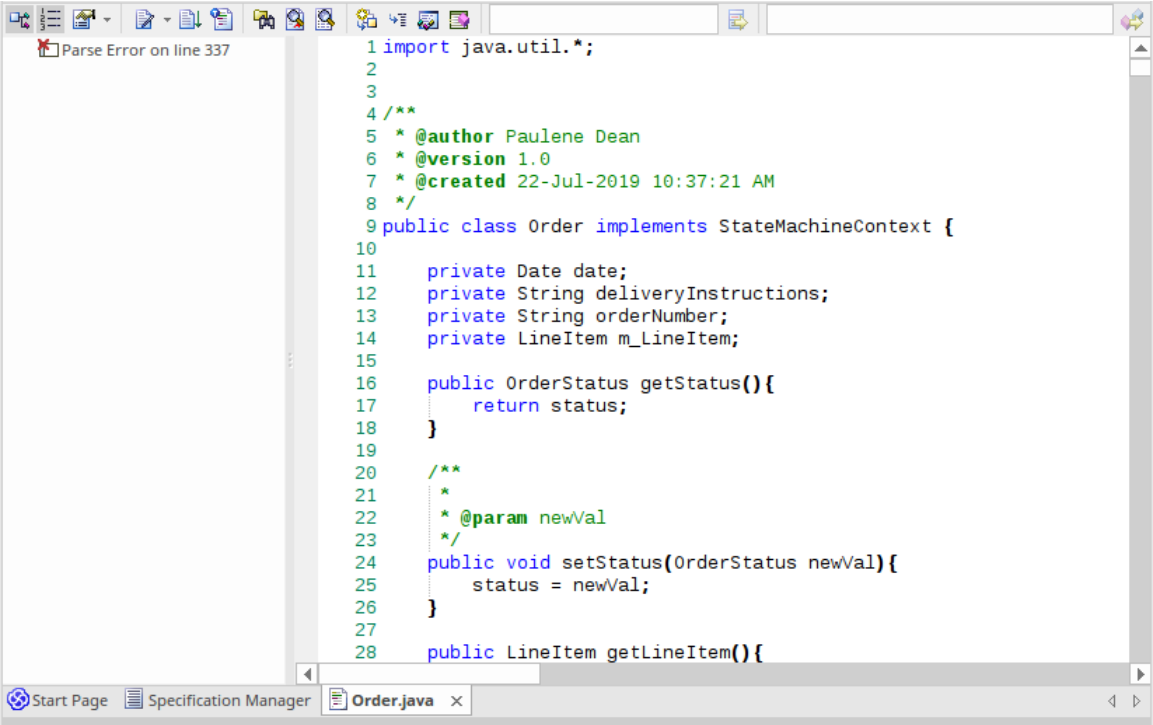
在“导入目录结构”对话框中，如果您选择“提示行动”单选按钮以手动审阅缺失的类，则会显示一个对话框，您可以在其中指定对导入代码中缺失的每个类的处理。

默认情况下，所有类都标记为删除；要保留一个或多个类，请选择它们并单击忽略按钮。

编辑源代码

Enterprise Architect包含一个功能丰富的源代码编辑器，可帮助您直接在工具内查看、编辑和维护您的源代码。一旦为一个或多个类生成了源代码，就可以在这个灵活的编辑环境中查看它。在上下文它的UML模型的时间中查看代码可以使代码和模型更加清晰，并弥合设计和实现之间的鸿沟，这种鸿沟在历史上已经将错误引入软件系统。

源代码编辑器功能齐全，具有结构树，可轻松导航属性、属性和方法。可以显示行号并且可以配置语法高亮选项。许多在他们最喜欢的 IDE 中熟悉的软件，例如智能感知特征和代码完成都包含在编辑器中。还有许多其他特征，例如宏录制，可以轻松管理Enterprise Architect中的源代码。还有许多用于管理代码的选项，可通过代码编辑器上下文菜单、工具栏和函数键使用。



对于大多数编程语言，单个文件是从UML类创建的，但在 C++ 的情况下，会创建头文件和实现类，并且源代码编辑器会在单独的选项卡中显示这些文件。

许多选项改变A源代码编辑器的工作方式；可以使用 开始”功能区中的 首选项”对话框更改它们：

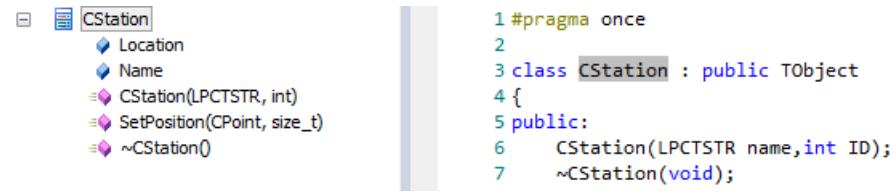
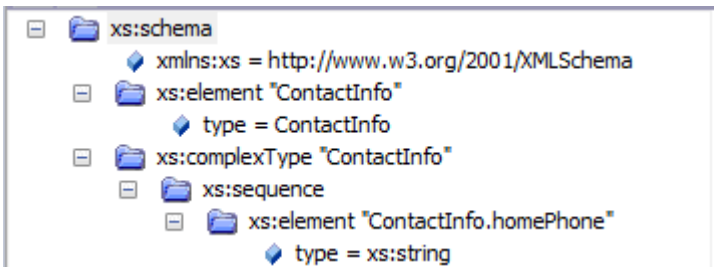
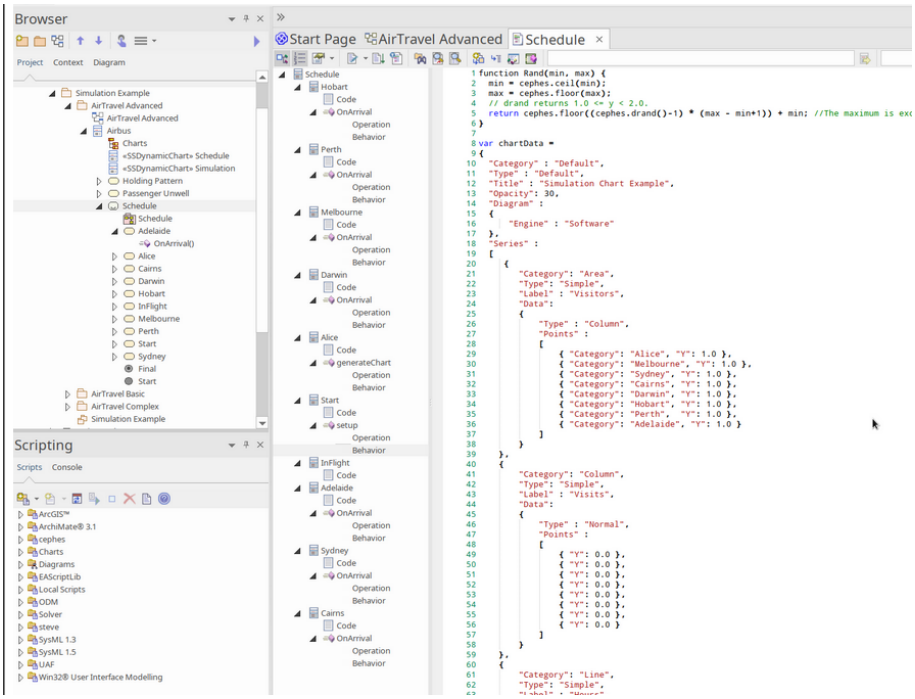
'开始>外观>首选项>首选项>源代码工程>代码编辑器'

源代码编辑器有多种变体，具有不同的访问方法。变体在比较编辑器主题中进行了讨论。

访问

功能区	执行>源>编辑>打开源文件（外部文件）或 执行 >源> 编辑 > 编辑元素源（对于现有的源文件）或 执行 >源> 编辑 > 编辑新源文件或 设计>元素>行为或 开发>源代码>行为
键盘快捷键	F12 或 Ctrl+E（用于模型元素的现有代码） Ctrl+Alt+O（定位外部文件）

功能

功能	描述
源代码编辑器	默认情况下，源代码编辑器设置为： ● 解析所有打开的文件，并显示结果树 ● 显示行号  如果您正在编辑 XML 文件，则结构树会反映文档的确切顺序和结构。 
结构树	文件结构树可用于支持的语言文件，例如 C++、C#、 Java和 XML。树有助于快速导航内容，就像内容表用于其他文档一样。
仿真行为	如果您正在编辑状态机或活动图表中元素的行为，代码编辑器允许您使用结构树一起列出和编辑图表中所有元素的行为。  在此图中，您可以看到一个状态机中的多个状态，每个状态都有操作和行为，所有这些状态都列在一起，无需离开或更改编辑器窗口即可选择。

注记

- 对于大多数选定的元素，您可以使用 F12 或 Ctrl+E 键查看源代码。
- 示例当您选择一个元素来查看源代码时，如果该元素没有生成文件（即，代码尚未生成或无法生成，例如用于使用元素）， Enterprise Architect将检查该元素是否具有指向用例成文件的链接另一个元素的操作或属性 - 如果存在这样的链接，并且另一个元素源代码，则显示该元素的代码
- 您还可以找到包含已在Enterprise Architect中创建或导入的源文件的目录，并使用记事本或 Visual Studio 等外部编辑器对其或其相关文件进行编辑；单击浏览器窗口中的元素，然后按 Ctrl+Alt+Y

支持的语言

源代码编辑器可以显示多种语言的代码，如下所示。对于每种语言，编辑器都会以彩色文本突出显示标准代码语法。

- 艾达（.ada、.ads、.adb）
- 动作脚本 (.as)
- BPEL 文档 (.bpel)
- C++（.h、.hh、.hpp、.c、.cpp、.cxx）
- C# (.cs)
- DDL 结构化查询语言 (.sql)
- 德尔福/帕斯卡 (.pas)
- 差异/补丁文件（.diff，补丁）
- 文档类型定义 (.dtd)
- DOS 批处理文件 (.bat)
- DOS 命令脚本 (.cmd)
- HTML (.html)
- 接口定义语言 (.idl, .odl)
- Java (.java)
- JavaScript (.javascript)
- JScript (.js)
- 修改的巴库斯-瑙尔形式语法 (.mbnf)
- PHP（.php、.php4、.inc）
- Python (.py)
- 标准通用标记语言 (.sgml)
- SystemC (.sc)
- Visual Basic 6 (.bas)
- VB.NET (.vb)
- VBScript (.vbs)
- Verilog (.v)
- VHSIC 硬件描述语言 (.vhdl)
- Visual Studio 资源配置(.rc)
- XML（可扩展标记语言）（.xml）
- XSD（ XML Schema定义）
- XSL（XML 样式表语言）

配置文件关联

如果您是 Windows® 用户，您可以将Enterprise Architect配置为您的语言源文件的默认文档处理程序。

访问

功能区	开始>外观>首选项>首选项>源代码工程>代码编辑器：配置 Enterprise Architect文件关联配置

行动

对于您希望在Enterprise Architect中打开的每种文件类型，单击文件类型名称左侧的复选框。选择所需的所有文档类型后，单击 保存”按钮。

在此之后，单击 Windows® Explorer 中的任何相应文件将在Enterprise Architect中打开它。

注记

- 您可以直接通过窗口®控件面板中的 默认程序”选项更改默认程序或由它们处理的文档。

比较编辑

Enterprise Architect提供了四种主要的代码编辑器变体，可通过多种访问路径获得。这些描述中确定了最直接的访问选项。

列出的前三个代码编辑器变体具有相同的显示格式、选项工具栏、上下文菜单选项和内部函数键。它们的访问方法和显示机制不同。

编辑器变体

变体	细节
源代码视图	F12 Ctrl+E 类上下文菜单 | '视图源代码' 描述：在图表视图的选项卡上显示代码；选项卡标签显示文件名和扩展名（例如 .java）；同样，对于 C++，头文件和实现文件有两个选项卡。 您可以通过重新选择下一个类上的菜单选项/键，在其他选项卡上显示其他类的源代码。
源代码窗口（可停靠）	Alt+7 '执行>源>编辑>打开源文件' 描述：显示所选类的源文件的内容（除非语言是 C++，当窗口显示 Header 文件的选项卡和 Implementation 文件的选项卡时）。 如果您选择不同的类，窗口会更改以显示新类的代码（除非第一个类调用第二个，在这种情况下，窗口会向下滚动到第二个类的代码）。
内部编辑，外部源代码	Ctrl+Alt+O '执行>源>编辑>打开源文件'功能区选项 描述：如果您打算编辑外部代码、XML 或 DDL 文件（即未导入或在Enterprise Architect中生成的代码），请使用此选项。 显示一个外部浏览器，然后打开特定选定的代码文件作为图表视图的选项卡（对于 C++，不是两个代码文件）；否则这与 F12 选项相同。
外部编辑器，内部或外部源代码	Ctrl+Alt+Y 类上下文菜单 |打开源目录 描述：显示一个外部文件浏览器，打开到包含所选类的源文件的目录；您可以在记事本、Visual Studio 或系统上可能拥有的其他工具中打开这些文件。

代码编辑器工具栏

当您在源代码编辑器中查看模型的一部分的代码时，您可以从编辑器工具栏访问范围广泛的显示和编辑功能。

代码编辑器工具栏

工具栏选项

结构树 单击此图标可显示或隐藏元素层次结构面板（源代码编辑器的左侧面板）。

行号 单击此图标可根据代码行显示或隐藏行号。

源代码工程属性 单击下拉箭头以显示选项菜单以选择 首选项"对话框的各个 源代码工程"页面，您可以从中配置源代码工程的显示和行为选项：

- 语
- 语法高亮选项
- 代码编辑器选项
- 代码工程选项
- 代码编辑器键绑定

编辑函数 单击下拉箭头以显示提供对一系列代码编辑功能的访问的菜单：

- 打开对应文件(Ctrl+Shift+O) - 打开与当前打开的文件关联的头文件或实现文件
- 转到匹配大括号 (Ctrl+E) - 对于选定的左大括号或右大括号，突出显示该对中相应的右大括号或左大括号
- 转到行 (Ctrl+G) - 显示一个对话框，您可以在其中选择要突出显示的行号；单击确定按钮将光标移动到该行
- Cursor History Previous (Ctrl+-) -源代码查看器保留前 50 个光标位置的历史记录，当光标从其先前位置移动超过 10 行或在查找和替换操作中时创建一条记录;菜单选项将光标移动到上一个光标历史记录中的位置
- Cursor History Next (Ctrl+Shift+-) - 如果您已移动到较早的光标位置，此选项将光标移动到紧随其后的光标历史记录中的位置
- 查找 (Ctrl+F) - 显示一个对话框，您可以在其中定义文本string和搜索选项以在代码中定位该文本string
- 替换 (Ctrl+R) - 显示一个对话框，您可以在其中定义文本string和搜索选项以在代码中定位该文本string并将其替换为另一个文本string ；该对话框有选项可以根据您的决定定位和替换每个匹配项，或立即替换所有匹配项
- Highlight Matching Words - (Ctrl+3) 启用或禁用在查找操作期间突出显示匹配的单词；默认情况下启用此选项
- 记录宏 - 记录您的下一次击键以保存为宏
- 停止记录并保存宏 - 停止记录击键并显示 保存宏"对话框，您可以在其中指定宏的名称
- 播放宏 - 显示 打开宏"对话框，您可以从中选择并执行保存的宏，以重复保存的击键

- Toggle Line注解(Ctrl+Shift+C) - 注释掉 (//) 或为突出显示文本的每一行重新建立代码
- Toggle Stream注解(Ctrl+Shift+X) - 在光标位置插入流注释（/* */）（仅注释掉突出显示的字符和行），或将注释文本重新建立为代码
- 切换空白字符 (Ctrl+Shift+W) - 显示或隐藏空格字符：-->（制表符空格）和 .（字符空间）
- 切换 EOL 字符 (Ctrl+Shift+L) - 显示或隐藏行尾字符：CR（回车）和 LF（换行）
- Toggle树- 在代码编辑器中随着上下文变化自动选择树项
- 打开包含文件夹 - 在包含代码文件的文件夹中打开文件浏览器；您可以在默认外部编辑器中打开其他文件进行比较和并行工作

保存源和重新同步类 单击此图标可保存源代码并重新同步模型中的代码和类。

代码模板 单击此图标可访问代码模板编辑器，以编辑或创建代码模板以生成代码。

在项目中查找浏览器 对于选定的代码行，单击此图标以突出显示浏览器窗口中的相应结构。如果存在多种可能性，则会显示 "可能的匹配"对话框，列出结构的出现情况，您可以从中选择所需的结构。

在文件中搜索 单击此图标可在关联文件中搜索所选object名称，并在文件搜索窗口中显示搜索结果。您可以通过在 "在文件中查找"窗口工具栏上指定条件来优化和刷新搜索。

在模型中搜索 单击此图标可在整个模型中搜索所选文本，并将搜索结果显示在在项目中查找视图中。

前往声明 单击此图标可在源代码中找到符号的声明。

转到定义 单击此图标可在源代码中找到符号的定义（适用于 C++ 和 Delphi 等语言，其中符号在单独的文件中声明和定义）。

自动完成列表 单击此图标可显示可能值的自动完成列表；双击一个值以选择它。

参数信息 当光标位于操作参数列表的括号之间时，单击此图标可显示操作的签名，突出显示当前参数。

在浏览器Window中查找当前类 单击该图标可显示代码中当前选择的类的名称，并在浏览器窗口中突出显示该名称；如果存在多种可能性，则会显示 "可能的匹配"对话框，列出出现的类，您可以从中选择所需的类。

查找会员 单击该图标可显示代码中当前选择的属性或方法的名称，并在浏览器窗口中突出显示该名称；如果存在多种可能性，则会显示 "可能的匹配"对话框，列出出现的特征，您可以从中选择所需的特征。

注记

- 录制宏时， "重新 "选项会禁用智能感知录制
- 您可以指定击键来执行宏，而不是使用工具栏下拉菜单和 "打开宏"对话框

代码编辑器上下文菜单

使用代码编辑器处理文件时，您可以执行许多代码搜索和编辑操作来审阅文件的内容。这些选项可通过编辑器上下文菜单获得，并且可能因您使用的代码编辑器而异。

访问

上下文菜单	右键单击您正在处理的代码文本string

选项

前往声明 在源代码中找到并突出显示符号的声明。

转到定义 在源代码中找到并突出显示符号的定义（适用于 C++ 和 Delphi 等语言，其中符号在不同的位置声明和定义）。

在语法编辑器中打开 打开一个视图，让您可以使用适当的语法检查或验证代码。

同步树到编辑器 在结构树中查找并显示当前元素（例如方法）。

自动同步树和编辑器 选中后，结构树将自动显示编辑器中正在处理的元素。

XML Schema验证 允许验证 XML 模式。

搜索 < string >” 显示一个子菜单，提供在一系列位置中定位选定文本string的选项。

- '在项目中查找浏览器' - 在浏览器窗口中突出显示包含所选文本的object
- “在打开的文件中搜索” - 在关联的打开文件中搜索选定的文本string ，并在 “在文件中查找”窗口中显示搜索结果；您可以通过在 “在文件中查找”窗口工具栏上指定条件来优化和刷新搜索
- “在文件中搜索” - 在所有关联文件（关闭或打开）中搜索选定的文本string ，并在 “在文件中查找”窗口中显示搜索结果；您可以通过在 “在文件中查找”窗口工具栏上指定条件来优化和刷新搜索（快捷键：F12）
- “在模型中搜索” - 在模型搜索功能中执行 “元素名称”搜索，并在模型搜索选项卡上显示结果
- 'Search in脚本' - （在脚本编辑器中工作时可用）打开 “在文件中查找”窗口，将 “搜索路径”字段设置为 “在脚本中搜索”，将 “搜索文本”字段设置为选定的文本，然后搜索所有文本string的脚本并显示搜索结果；您可以通过在 “在文件中查找”窗口工具栏上指定条件来优化和刷新搜索
- 'EA用户指南' - 在*Enterprise Architect* 用户指南中显示代码项的描述
- 'Google' - 在文本上显示 Google 搜索的结果
- 'MSDN' - 显示在 Microsoft Developer Network (MSDN) 中的文本搜索结果
- 'Sun Java SE' - 显示对 Sun Microsystems 'Sun Search'功能中文本的搜索结果
- 'Wikipedia' - 在 Wikipedia 网站上显示object上的任何条目
- 'Koders' - 显示在 Koders.com 上搜索文本string的结果

搜索智能感知	使用当前分析器中指定的代码矿工脚本或库对指定string进行搜索。结果显示在 在文件中查找"窗口的 代码矿工"选项卡中。 快捷键：Shift+F12
将调试器设置为 **Line**	（如果调试器正在执行并且已经到达断点。）将执行点移动到当前行。选择不要跳过任何影响下一段正在调试的代码的代码或声明。
显示变量	（如果调试器正在执行。）打开本地窗口并突出显示代码中当前点的局部变量。
在字符串查看器中显示	在字符串查看器中显示变量string的完整内容。
为 **'< string >'** 创建用例	显示 创建用例For Method"对话框，您可以通过该对话框为包含文本string的方法创建一个用例。
断点	显示用于在所选代码行上创建记录标记的选项子菜单。您可以添加的录音标记包括： • 断点 • 开始记录Marker • 结束记录标记 • 堆栈自动捕获标记 • 方法自动记录标记 • 跟踪点
测试点	显示选项以添加新测试点、显示测试点管理器（测试点窗口）或编辑现有测试点（如果已在选定位置定义了一个或多个测试点）。 （子选项取决于您正在查看的代码文件的类型。）
XML 验证	允许检查 XML 文档是否符合其自己的模式引用或使用用户指定的模式；本地模式文件或 URL。
打开（关闭）输入法	打开（或关闭）输入法编辑器，以便您可以在选定的外语脚本中输入文本，例如日语。您可以使用窗口功能控件面板 - 区域和语言选项设置功能语言。
复制位置超链接	将光标位置复制为可以粘贴到 Rich注记编辑器中的超链接，例如 聊天和邮件"窗口的 聊天"选项卡中的消息。只需使用消息中的 粘贴"上下文菜单选项，并指定链接文本。 读者可以点击链接打开源文件，将光标移动到文件中选定的光标位置。
复制文本超链接	将选定的文本string复制为可以粘贴到 Rich注记编辑器中的超链接，例如 聊天和邮件"窗口的 聊天"选项卡中的消息。只需使用消息中的 粘贴"上下文菜单选项即可。 读者可以单击链接打开源文件并将光标移动到文件中该文本string的第一次出现处。
行号	（仅限脚本编辑器。）显示或隐藏编辑器屏幕左侧的代码行号。
撤消 切 复制 粘贴 删除 全选	这六个选项提供了用于编辑代码的简单功能。

注记

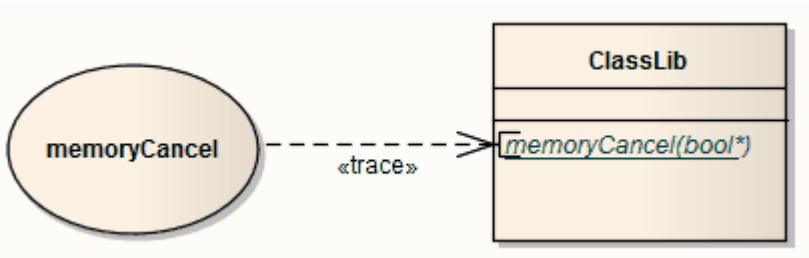
- 'Search for <string>' 子菜单下半部分的选项（在'Search in Scripts'之后）是可配置的；您可以通过编辑Sparx Systems > EA > Config 文件夹中的 searchProviders.xml 文件来添加新的搜索工具或删除现有的搜索工具 - 该文件采用 OpenSearch 描述文档格式

创建用例for方法

使用代码编辑上下文菜单，您可以为从代码中选择的方法创建一个用例元素。你也可以：

- 将用例直接链接到方法
- 将父类添加到图表中（如果它不在所选图表中）和/或将用例元素添加到图表中
- 特征块来自显示不是链接目标的任何属性或方法

通过代码编辑器为方法创建一个用例

节	行动
1	（如果您想在图表中描述用例及其与方法的链接）单击浏览器窗口中的图表名称。
2	在代码编辑器中，右键单击方法名称或方法主体的任何部分，然后选择 "为 <methodname> 创建方法"选项。 将显示 "创建用于方法的用例"对话框。
3	此对话框的基本函数是为选定的方法创建一个用例： ● 如果这就是所有要求，请单击确定按钮；用例元素是在浏览器窗口中创建的，与方法的父类在同一个包中，并且与方法同名 ● 如果您打算使关系有形，请继续进行过程
4	要创建跟踪连接器或将用例链接到方法，请选中 "链接用例到方法"复选框。
5	要将方法的父类添加到图表中，如果它不存在，请选中 "添加类到图表"复选框。
6	要将新创建的用例添加到图表中，请选中 "将用例添加到图表"复选框；这将现在在图表上显示用例、类和跟踪连接器。
7	要仅显示作为 "链接到特征"关系目标的父类的特征（属性和方法），请选中 "仅显示链接的特征类"复选框。 这些类可能包含任意数量的属性和方法，但那些没有 "到特征"关系的链接是隐藏的。
8	点击确定按钮创建和描述用例和关系；如果您选择了所有选项，图表现在包含类似于此插图的链接元素： 

代码编辑器函数

通用代码编辑器提供了多种功能来协助代码编辑过程，包括：

- 语法高亮
- 书签
- 光标历史
- 大括号匹配
- 自动缩进
- 评论选择
- 范围指南
- 缩放
- 线路选择
- 智能感知
- 查找和替换
- 在文件中查找

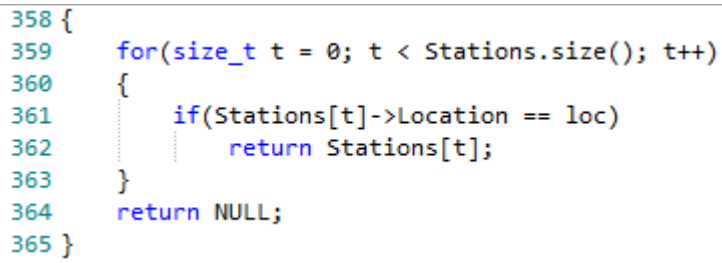
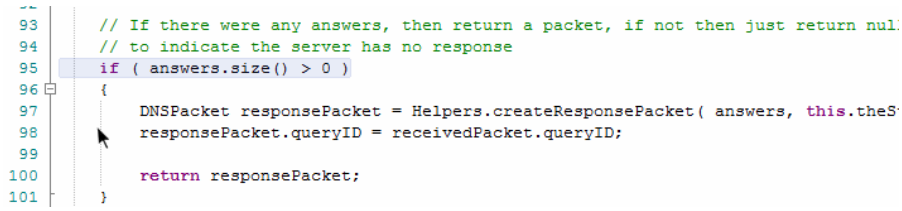
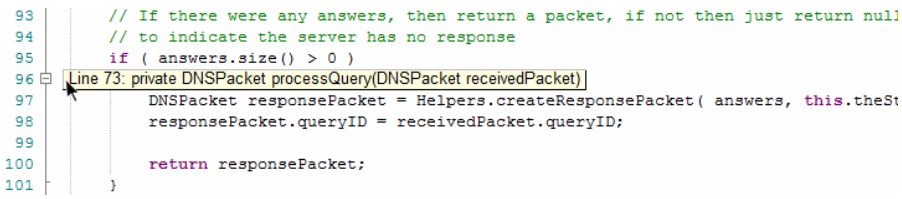
A这些功能可通过键盘组合键和/或上下文菜单选项获得。

您可以通过在代码编辑器配置文件中设置属性来自定义几个代码编辑器特征；例如，默认情况下，包含光标的行总是突出显示，但您可以关闭突出显示。

函数详情

代码编辑器函数

函数	描述
语法高亮	代码编辑器以彩色文本突出显示Enterprise Architect支持的所有语言文件格式的标准代码语法 1 #pragma once 2 #include "afxwin.h" 3 #include "afxcmn.h" 4 5 6 // CToolBox dialog 7 8 class CToolBox : public CDialog 9 { 10 DECLARE_DYNAMIC(CToolBox) 11 CRect m_rect; 12 int m_offset; 您可以通过 首选项"对话框的 代码编辑器"页面定义代码编辑器如何为每种语言实现语法突出显示。
书签	书签表示文档中感兴趣的行；您可以通过按 Ctrl+F2 为特定行打开和关闭它们。 此外，您可以按 F2 和 Shift+F2 导航到文档中的下一个或上一个书签。 要清除代码文件中的所有书签，请按 Ctrl+Shift+F2。
光标历史	代码编辑控件保留前 50 个光标位置的历史记录；历史列表中的条目在以下情况下创建： • 光标从之前的位置移动了 10 多行 • 光标在查找/替换操作中移动 您可以通过按 Ctrl+- 导航到光标历史记录中的较早点，然后按 Ctrl+Shift+- 导航到较晚的点。
大括号匹配	当您将光标放在大括号或方括号上时，代码编辑器会突出显示其对应的伙伴；然后，您可以按 Ctrl+E 导航到匹配的大括号。 28 function ProtectedFunctionTest: boolean; 29 procedure ProtectedProcedureTest(a: WideString);
自动缩进	对于每种支持的语言，代码编辑器会根据指向光标位置的行中是否存在控制语句或范围块标记来调整新行的缩进。

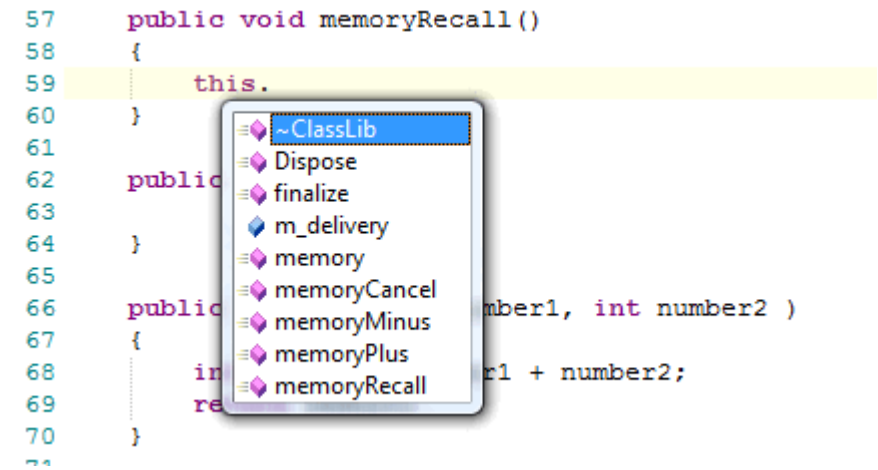
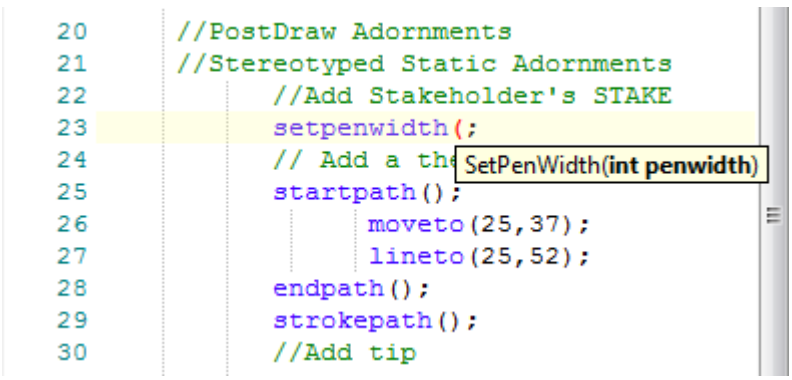
	 缩进的级别由浅水平线表示。 您还可以通过按 Tab 键手动缩进选定的行和代码块；要取消缩进所选代码，请按 Shift+Tab。
评论选择	对于支持注释的语言，代码编辑器可以注释整个代码选择。 代码编辑器识别两种类型的注释： • 行注释 - 从一开始就对整行进行注释（例如：// 这是一条评论） • 流注释 - 从指定的起点到指定的终点对行的各个部分进行注释（例如：/* 这是一条评论 */) 您可以通过按以下方式切换当前行或选择的注释： • Ctrl+Shift+C 用于行注释，或 • Ctrl+Shift+X 用于流评论
范围指南	如果光标放在缩进标记上，代码编辑器会执行 回溯”以查找在该缩进级别开始范围的行；如果找到该行并且当前在屏幕上，它会以浅蓝色突出显示。  或者，如果线路不在屏幕上，则会显示一个提示，告知线路编号和内容： 
缩放	您可以使用以下方法放大和缩小代码编辑器的内容： • Ctrl+小键盘 + 和 • Ctrl+键盘 - 使用 Ctrl+键盘 / 可以将缩放恢复到 100%。
线路选择	如果要将光标移动到特定的代码行，请按 Ctrl+G，然后在响应提示时输入行号。 按下确定按钮；编辑器显示指定的代码行，光标位于左侧。

智能感知

智能感知是一种特征，提供代码项和值的选择。并非所有代码编辑器都使用智能感知；例如，在您录制宏时，智能感知源代码查看器处于禁用状态。

智能感知通过自动提示列表、呼叫提示和鼠标悬停信息为您提供基于上下文的帮助

功能

功能	描述
自动完成列表	自动补全列表提供当前文本可能的补全列表；当您在包含成员的object或类型之后输入访问器标记（例如句点或指针访问器）时，将自动调用该列表。  您也可以通过按 Ctrl+Space 手动调用自动完成列表；代码编辑器然后搜索导致调用点的单词的匹配项。 从列表中选择一个项目，然后按 Enter 键或 Tab 键将该项目插入到代码中；要关闭自动完成列表，请按 Esc。
通话提示	Calltips 在您键入参数列表标记时显示当前方法的签名（例如，左括号）；如果方法重载，则调用提示会显示箭头，您可以使用这些箭头浏览不同的方法签名 
鼠标悬停信息	您可以通过将光标悬停在相关元素显示代码元素（例如，属性和方法）的支持文档。

```
11  dockable = "none";
12
13  Tagged V
14
15  //PreDraw Derived Attribute I
```

string
Dock elements together.
Valid Values: none, standard

查找和替换

Enterprise Architect的每个代码编辑器都通过 查找和替换"对话框促进在编辑器中搜索和替换术语。

访问

键盘快捷键	突出显示所需的文本string并按： • Ctrl+F 仅用于查找控件，或 • Ctrl+R 用于查找和替换控件 在每种情况下， 查找内容"字段都填充有当前在编辑器中选择的文本。如果在编辑器中未选择任何文本，则 查找内容"字段将填充当前光标位置处的单词。如果当前光标位置不存在任何词，则使用最后搜索的词。

基本操作- 命令

命令	行动
找下一个	找到并突出显示 查找内容"字段中指定的文本的下一个实例（相对于当前光标位置）。
代替	将 查找内容"字段中指定的文本的当前实例替换为 替换为"字段中指定的文本，然后定位并突出显示 查找内容"字段中指定的文本的下一个实例（相对于当前光标位置）查找什么字段。
全部替换	自动将 查找内容"字段中指定的文本的所有实例替换为 替换为"字段中指定的文本。

基本操作- 选项

选项	行动
相符	在代码中搜索匹配项时，指定 查找内容"字段中文本string中每个字符的大小写是否重要。
匹配整个单词	指定 查找内容"字段中的文本string是一个完成词，并且不应与构成较长string一部分的文本实例匹配。 例如，搜索 ARE 不应匹配单词 AREA 或 ARENA 实例中的那些字母。
向上搜索	执行从当前光标位置到文件开头的搜索，而不是按照当前光标位置到文件结尾的默认方向。
使用正则表达式	将 查找内容"和 替换为"字段中的特定字符序列评估为正则表达式。

概念

概念	描述
常用表达	A则表达式是搜索模式的正式定义，可用于匹配特定字符、单词或字符模式。 为简单起见，代码编辑器的 查找和替换"机制仅支持标准正则表达式语法的一个子集。 只有在 查找和替换"对话框中选中 使用正则表达式"复选框时， 查找内容"和 替换为"字段中的文本才会被解释为正则表达 。
元序列	如果选中 使用正则表达式"复选框，则 查找内容"字段中的大多数字符都将被视为文字（即，它们仅匹配自身）。 例外被称为元序列；在代码编辑器 查找和替换"对话框中识别的每个元序列在此表中进行了描述： • \< - 表示文本是单词的开头；例如： \<cat 匹配*catastrophe*和*cataclysm* ，但不连接 • \> - 表示文本是一个单词的结尾；例如： hat\> 与*that*和*chat*匹配，但不匹配*hat* • (...) - 表示可以匹配的替代单个字符 - 字符可以是特定的 (chr) 或在字母或数字范围内 (am)；例如： (hc) at 匹配到*hat*和*cat*但不匹配*bat* ，并且 (am)class 匹配到*aClass-mClass*类内的任何名称 • (^...) - 表示应从匹配中排除的替代单个字符 - 字符可以是特定的 (^chr) 或在字母或数字范围内 (^am)；例如： (^hc) at 匹配到*rat*和*bat* ，但不包括*hat*和*cat* ，并且 (^am)class 匹配到*nClass*到类范围内的任何名称，但不包括*aClass*到*mClass* • ^ - 匹配行首 • $ - 匹配行尾 • * - 匹配前面的字符（或字符集）0 次或多次；例如： ba*t 匹配*bt* 、 *bat* 、 *baat* 、 *baaat*等， b(ea) *t 匹配*bt* 、 *bet* 、 *bat* 、 *beat* 、 *beet* 、 *baat*等 • + - 匹配前面的字符（或字符集） 1次或多次；例如： ba+t 匹配到*bat* 、 *baat*和*baaat*但不匹配*bt* ，并且 b(ea) +t 匹配到*bet* 、 *bat* 、 *beat* 、 *beet*和*baat*但不匹配*bt* 如果单个字符元序列前面有反斜杠 (\)，则将其视为文字字符：c\(at\) 与 c(at) 匹配，因为括号是按字面处理的。 选中 使用正则表达式"复选框时， 查找内容"和 替换为"字段右侧都有一个元序列帮助菜单；从此菜单中选择元序列会将元序列插入到字段中，并根据需要替换或换行当前选定的文本。
标记区域	当使用正则表达式 查找和替换"时，最多可以将原始术语的九个部分替换为替换术语。 元序列 '\(' 和 '\)' 表示标记区域的开始和结束；位于标记区域内的匹配文本部分可以包含在替换文本中，并带有元序列 \n"（其中*n*是1到 9 之间的标记区域编号）。 例如： 查找：\((A-Za-z) +\)的东西 替换为属于 \ *1*的项目 原文：这些都是迈克尔的东西。 替换文本：这些都是属于迈克尔的项目。

在文件中搜索

文件文本搜索由 Find in Files 窗口和代码编辑器中提供，用于搜索文件中的数据名称和结构。这些文件可以是外部代码文件、您已经在Enterprise Architect中打开的代码文件、内部模型脚本或帮助子系统。

"文件搜索"选项卡维护您浏览过的文件路径的历史记录，帮助您快速返回文件系统中的常用文件夹。如果您需要多次重复搜索，您可以类似地选择以前使用的搜索string 。当您搜索代码文件时，您还可以通过选择文件扩展名将搜索限制在特定类型的文件中，并仅包括所选文件夹或其所有子文件夹。另一个有用的功能是能够选择将搜索结果显示为string的每个实例的列表，或包含该string的文件列表，其中实例分组在找到它们的文件下。

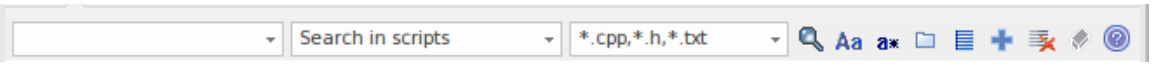
对于所有搜索，您可以将搜索限定为区分大小写和/或将搜索string与完成词匹配。

访问

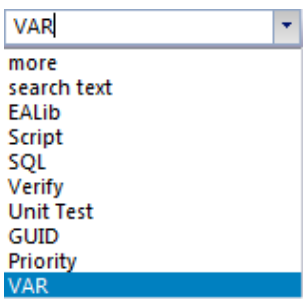
功能区	探索 > 搜索 > 文件 执行>源>查找 执行 >源> 编辑 > 在文件中搜索
上下文菜单	右键单击选定的文本 |搜索 <选定文本> |在文件中搜索
键盘快捷键	F12、Ctrl+Shift+Alt+F

搜索工具栏

您可以使用 "在文件中查找"窗口中的工具栏选项来控制搜索操作。每个按钮的状态都会随着时间的推移而持续存在，以始终反映您以前的搜索条件。



选项

选项	行动
	"搜索文本"字段。类型要搜索的文本string 。 您输入的任何文本都会自动保存在下拉列表中，最多十个字符串；之后添加的文本会覆盖列表中最旧的文本string 。如果愿意，您可以单击下拉箭头并选择其中一个已保存的文本字符串。
	"搜索路径"字段。指定要搜索的文件夹或搜索类型。 您可以直接在文本框中键入要搜索的文件夹路径，或单击下拉箭头并选择 "浏览文件夹"以使用 "浏览文件夹"对话框进行搜索。 您输入的任何路径都会自动保存在下拉列表中，最多十个；之后添加的路径

Search in local help... more Browse for folder... Search in scripts Search in open files... Search in local help... c:\ C:\EA\VEA\Microsoft Native\CityLoop	会覆盖列表中最旧的路径。如果您愿意，可以选择这些保存的路径之一。 除了 浏览文件夹”之外，下拉列表中还有其他三个固定选项： ● 'Search in脚本'，在脚本窗口中搜索本地和用户自定义的脚本 ● 在打开的文件中搜索”，它将搜索范围限制在您在Enterprise Architect中打开的文件中 ● 在本地帮助中搜索”，它搜索已从帮助Sparx Systems网站安装的本地帮助文件；结果列出了包含搜索词的帮助主题，以及文本出现的行号和行 这些选项禁用 搜索文件类型”列表框。
.cs,.txt *.cpp,*.h,*.txt *.c,*.h *.java,*.txt *.cs,*.txt	搜索文件类型”字段。单击下拉箭头并选择要搜索的文件类型（文件扩展名）。
	单击此图标开始搜索。 在搜索过程中，工具栏中的所有其他按钮都被禁用。您可以随时再次单击 搜索”按钮取消搜索。 如果切换这些切换按钮中的任何一个，则必须再次运行搜索以更改输出。
Aa	单击此图标可切换搜索是否区分大小写。工具提示消息标识当前设置。
a*	单击此图标可在搜索任何匹配项和仅搜索构成整个单词的匹配项之间切换。工具提示消息标识当前设置。
	单击此图标可在将搜索限制为单个路径和包括该路径下的所有子文件夹之间切换。工具提示消息标识当前设置。
	单击此图标可选择搜索结果的呈现格式；你有两个选择： ● 列表视图-（如图所示）每个结果行由文件路径和行号组成，然后是行文本；一个文件中的多行被列为单独的条目 ● 树视图 - (视图) 每个结果 包含与搜索条件匹配的文件路径，以及与该文件中的搜索文本匹配的行数；您可以展开条目以显示每行的行号和文本
	单击此图标可添加新的搜索选项卡。您最多可以创建四个新的搜索选项卡。搜索也运行同时运行。
	单击此图标可清除结果。
	如有必要，单击此图标可删除 搜索路径”、 搜索文本和 搜索文件类型”下拉列表中的所有条目。

查找文件

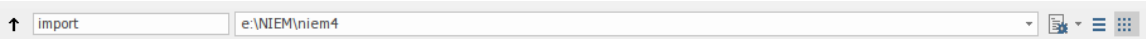
在文件中查找窗口的 查找文件”选项卡提供了一个工具，可以帮助您更快地查找文件。该选项卡充当文件系统资源管理器，并提供常见打开文件对话框的快速替代方案。文件搜索快速简单，允许您查找感兴趣的文件而不会丢失当前工作流程。显示可以在报告和列表视图之间切换。

访问

功能区	探索 > 搜索 > 文件 > 查找文件
键盘快捷键	Ctrl+Shift+Alt+F

工具栏

工具栏提供搜索过滤器和文件夹导航组合框。工具栏提供了记住搜索位置以及在列表和报告视图之间切换的选项。



选项

	单击以导航到父文件夹。
	过滤器控件允许您排除与您键入的条件不匹配的文件。通配符 * 会自动附加到文本中，因此无需自己添加。要搜索包含术语 jvm”的所有文件，只需键入 “jvm” 。要查找包含术语 red”的 .png 图像，您可以输入 *red*.png。按 Enter 键更新结果。
	输入目录的路径，然后按 Enter 键以显示该位置的文件 使用下拉列表从当前模型的书签位置中选择。可以使用工具栏菜单管理位置。
	允许您管理目录组合中显示的位置。 ● 记住路径 - 存储 目录”字段的当前值，以便稍后返回 在文件中查找”窗口时， 目录”字段默认为该值（如果它是唯一的 记住”值）或提供下拉列表中的值 ● 忘记路径 - 从内存中清除当前值，以便它不会作为 目录”字段的可能值提供 ● 记住过滤器- 将当前值存储在 过滤器”字段中，以便稍后返回 在文件中查找”窗口时， 过滤器”字段默认为该值 ● 忘记过滤器-从内存中删除 过滤器”字段值，以便下次访问窗口时不会将其放置在该字段中

	在此视图中，列表显示列 "名称"、 "修改日期"、 "类型"和 "大小"。 列可以按升序或降序排序。第三次单击该列以删除排序顺序。
	列表视图删除列，当文件夹包含许多文件时很方便。

键盘快捷键

	将聚焦设置为过滤器控件。
	导航到父文件夹。
	导航到父文件夹。
	如果选择了文件夹，则打开该文件夹，否则打开所选文件。

搜索智能感知

Enterprise Architect的智能感知功能是使用Sparx Systems的“代码矿工”工具构建的。代码矿工提供对现有代码库中信息的快速和全面的访问。系统提供对原始源代码所有方面的完整访问，可以“即时”在代码编辑器中搜索，也可以通过用矿代码矿工完成语言编写的查询生成的搜索结果。

访问

在“在文件中查找”窗口中，单击“代码矿工”选项卡。

功能区	探索 > 搜索 > 文件
键盘快捷键	Ctrl+Shift+Alt+F

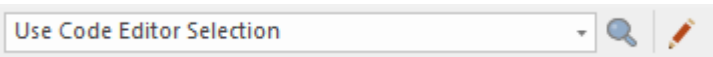
矿代码矿工控件

此控件提供了一个界面，用于一次对多个代码库执行查询。它使用的代码库是使用Enterprise Architect的代码矿工工具构建的数据库。这些数据库形成一个库，在部署为服务时也可以共享。可以使用工具栏列出和选择可以运行的查询，从而可以轻松访问查询的源代码，进行编辑和组合。查询不需要编译；可以像源代码文件一样查看、编辑和保存它们。采用单个参数的查询可以利用打开的代码编辑器中的任何选择。该接口还支持采用多个参数的查询的手动参数输入。

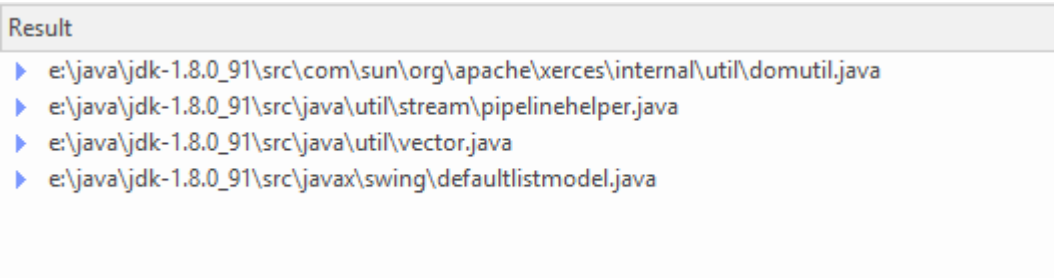
工具栏上的第一个控件列出了可用的命名空间。选择命名空间会将显示的查询限制为该命名空间内的查询。

下一个控件提供了一个下拉列表，其中包含所选命名空间的查询文件中的所有查询。

第三个控件是一个编辑组合框。默认情况下，从打开的代码编辑器中的选定文本中获取单个查询参数，但您也可以直接在此字段中键入参数。多个参数应以逗号分隔。紧随其后的是“搜索”按钮以运行查询。可以使用“搜索”按钮旁边的“编辑”按钮 时编辑查询。



“结果”面板是一个树形控件，列出了按文件分组的查询结果。



代码矿工库

代码矿工是数据库的集合，Enterprise Architect智能感知供应商可以使用这些数据库来获取和查询跨多个代码库的信息。每个数据库都是使用适合其语言（C++、Java或C#）的专用语法从代码库的根代码目录创建的。

在 分析器脚本器”中创建、更新、删除或添加这些库。使用此特征A典型场景是为开发项目创建一个数据库，并为项目引用的框架创建其他数据库。随着代码更改的增加，您的开发数据库可以经常更新，而静态框架的更新频率会降低。可以通过与 文件搜索”工具类似的方式搜索库，但代码矿工由于其 mFQL 语言而提供了高级搜索功能。

- 可以一次搜索多个域/框架
- 查询可以A文件搜索运行时间的一小部分内运行
- 可以对查询进行编码以协助复杂的搜索条件
- 查询可以采用多个参数
- 所有文件都基于等效的UML结构进行索引，允许智能搜索在建模设置中产生有意义的结果

代码矿工查询文件

代码矿工查询保存在单个源代码文件中，该文件应具有 .mFQL 扩展名。每个Enterprise Architect安装都提供了A组基本查询；这些可以位于 config\codeminer 子目录中。这个查询文件在你编辑的任何分析器脚本都应该默认命名。

在编辑任何查询之前，建议您将此文件复制到工作位置并在您使用的任何分析器脚本命名该副本。这样，您将始终有一个参考文件可以返回。

最好将查询视为用 mFQL 语言编写的函数。因此，它们具有唯一的名称，可以由单个命名空间限定并且可以指定参数。该文件提供智能感知控件工具栏中列出的查询。每当保存对查询文件的编辑时，搜索工具栏组合框中列出的查询都会相应更新。此图像是用 mFQL 编写的简单查询示例。

```
188
189 namespace java
190 {
191 //
192 // Find all references
193 //
194 query::findByName($param1)
195 {
196     distinct(GetByValue( $param1 +))
197 }
198
199 query::findMethodByName($name)
200 {
201     move( 1, "METHOD", intersect( GetByNode("NAME"), GetByValue( $name ) )  )
202 }
203
204 query::findMethodCall($name)
205 {
206     filter( "METHOD_ACCESS", intersect(GetByNode("NAME"), GetByValue( $name )) )
207 }
208
```

代码编辑器键绑定

钥匙

钥匙	描述
Ctrl+G	将光标移动到指定行
↓	将光标下移一行
Shift+↓	将选择向下延伸一行
Ctrl+↓	向下滚动一行
Alt+Shift+↓	将矩形选区向下延伸一行
↑	将光标上移一行
Shift+↑	将选择向上扩展一行
Ctrl+↑	向上滚动一行
Alt+Shift+↑	将矩形选区向上延伸一行
Ctrl+(	将光标上移一个段落
Ctrl+Shift+(	将所选内容向上扩展一段
Ctrl+)	将光标下移一段
Ctrl+Shift+)	将选择向下扩展一段
←	光标左移一个字符
Shift+←	扩展选择左一个字符
Ctrl+←	光标左移一个单词
Ctrl+Shift+←	扩展选择左一个字
Alt+Shift+←	将矩形选区向左扩展一个字符
→	将光标向右移动一个字符。
Shift+→	将选择向右扩展一个字符
Ctrl+→	光标右移一个字
Ctrl+Shift+→	将选择向右扩展一个单词

Alt+Shift+→	将矩形选择向右扩展一个字符
Ctrl+/	将光标左移一个单词部分
Ctrl+Shift+/	扩展选择左一个单词部分
Ctrl+\	光标右移一个单词部分
Ctrl+Shift+\	将选择向右扩展一个单词部分
家	将光标移动到当前行的开头
Shift+Home	将选择范围扩展到当前行的开头
Ctrl+主页	将光标移动到文档的开头
Ctrl+Shift+Home	将选择范围扩展到文档的开头
Alt+Home	将光标移动到行的绝对开头
Alt+Shift+Home	将矩形选区扩展到行首
结尾	将光标移动到当前行的末尾
Shift+结束	将选择范围扩展到当前行的末尾
Ctrl+结束	将光标移动到文档末尾
Ctrl+Shift+结束	将选择范围扩展到文档末尾
Alt+结束	将光标移动到行的绝对末尾
Alt+Shift+End	将矩形选区扩展到行尾
向上翻页	将光标上移一页
Shift+Page Up	将选择向上扩展一页
Alt+Shift+Page Up	将矩形选区向上扩展一页
向下翻页	将光标向下移动一页
Shift+Page Down	将选择向下扩展一页
Alt+Shift+Page Down	将矩形选区向下扩展一页
删除	删除光标右侧的字符
Shift+删除	剪切选择

Ctrl+删除	删除光标右侧的单词
Ctrl+Shift+删除	删除直到行尾
插入	切换改写
Shift+插入	粘贴
Ctrl+插入	复制选择
退格	删除光标左侧的字符
Shift+退格键	删除光标左侧的字符
Ctrl+退格	删除光标左侧的单词
Ctrl+Shift+退格	删除从行首到光标
Alt+退格键	撤销删除
标签	缩进光标一个选项卡
Ctrl+Shift+I	缩进光标一个选项卡
Shift+Tab	取消缩进光标一个选项卡
Ctrl+小键盘(+)	放大
Ctrl+小键盘(-)	缩小
Ctrl+键盘(/)	恢复缩放
Ctrl+Z	撤消
Ctrl+Y	重做
Ctrl+X	剪切选择
Ctrl+C	复制选择
Ctrl+V	粘贴
Ctrl+L	切割线
Ctrl+T	移调线
Ctrl+Shift+T	复制行
Ctrl+A	选择整个文档
Ctrl+D	重复选择

Ctrl+U	将所选内容转换为小写
Ctrl+Shift+U	将所选内容转换为大写
Ctrl+E	将光标移动到匹配的大括号
Ctrl+Shift+E	将选择扩展到匹配的大括号
Ctrl+Shift+C	切换选择的行注释
Ctrl+Shift+X	切换选择的流评论。
Ctrl+F2	切换书签
F2	转到下一个书签
Shift+F2	转到上一个书签
Ctrl+Shift+F2	清除当前文件中的所有书签
Ctrl+Shift+W	切换空白字符
Ctrl+Shift+L	切换 EOL 字符
Ctrl+空格	调用自动完成。
Ctrl+-	在光标历史记录中后退
Ctrl+Shift+-	在光标历史中前进
F12	开始/取消在文件中搜索关键字。
Ctrl+F	查找文本
Ctrl+R	替换文本

注记

- 除了这些键之外，您还可以将 (Ctrl+Alt+<n>) 组合键分配给您在源代码编辑器中定义的宏

应用模式（模型+代码）

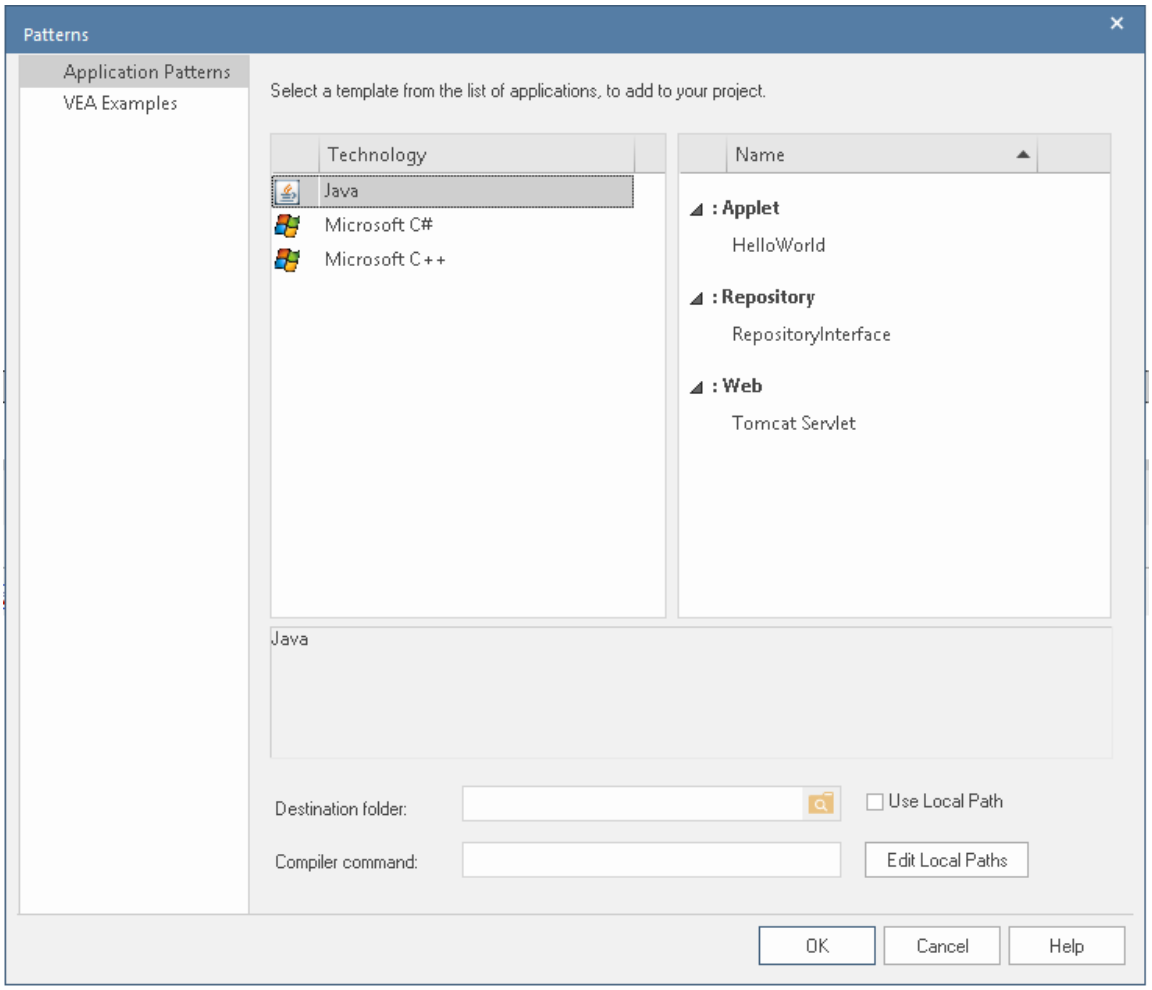
为了让您尽快完成基于代码的项目， Enterprise Architect可帮助您生成包含模型信息、代码和构建脚本的入门项目，用于几种基本应用程序类型之一。模式包括：

- MFC窗口应用程序
- Java程序
- ASP.NET 网络服务

访问

功能区	开发 >源代码 > Create From模式> Application模式

生成模型



选项	行动
技术	选择合适的技术。

名称	显示所选技术可用的应用程序模式；选择需要导入的模式。
<说明>	显示所选模式的描述。
目标文件夹	浏览并选择要加载应用程序源代码的目录。
使用本地路径	启用选择现有的本地路径来放置源代码；将“目标文件夹”字段更改为下拉选择。
编译器命令	显示所选技术的默认编译器命令路径；您必须： ● 确认可以在此路径中找到编译器，或者 ● 编辑编译器位置的路径
编辑本地路径	许多应用程序模式使用本地路径指定它们的编译器。 第一次使用任何模式时，您必须单击此按钮以确保本地路径指向正确的位置。 将显示“本地路径”对话框。

注记

- 如果需要，您可以通过将文件添加到安装Enterprise Architect的*AppPatterns*目录来发布自定义应用程序模式；顶级目录被列为技术并且可以包含一个图标文件来自定义为技术显示的图标
  下面的目录被定义为模式列表中的组；该模式通过存在四个具有匹配名称的文件来标识：压缩文件 (.zip)、XMI 文件 (.xml)、配置文件 (.cfg) 和可选图标 (.ico)
- 配置文件支持以下字段：
  - [provider], [language], [platform], [url], [description], [version] - 所有显示在 <description> 场地
  - [xmirootpaths] - 导出的XMI中源代码的根路径；这被替换为用户应用应用模式时选择的目标文件夹

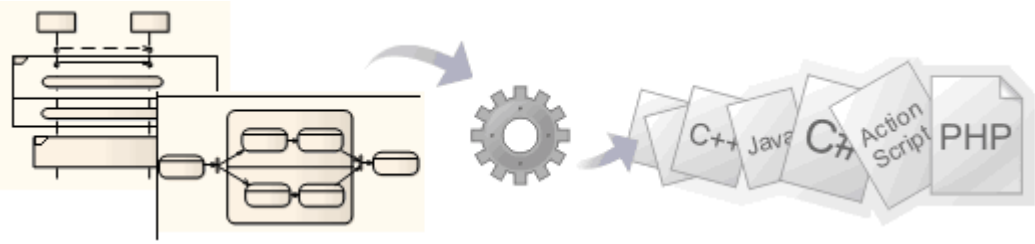
MDG集成和代码工程

用于 Eclipse 的MDG集成和用于 Visual Studio 的MDG集成是帮助您使用Enterprise Architect浏览器窗口直接在这两个流行的集成开发环境中创建和维护UML模型的产品。可以使用丰富而灵活的模板引擎将模型生成为源代码，该引擎使工程师可以完成控制代码的生成方式。现有的源代码也可以逆向工程并与UML模型同步。安装集成后，IDE 将成为一个功能丰富的建模平台，通过将需求管理、架构和设计与源代码工程联系起来，节省时间和精力并降低错误风险。

丰富而富有表现力的文档可以自动生成为各种格式，包括 DOCX、PDF 和 HTML。文档可以包括需求、设计和架构的图表以及源代码描述，将源代码放入上下文。

您可以从Sparx Systems网站购买 Eclipse ™的MDG集成和 Visual Studio ™的MDG集成或下载试用版。

行为模型代码生成



企业的多特征系统能力可用于软件、系统和硬件的工程语言直接从行为模型生成描述，如状态机、序列（交互）和活动图。支持的语言包括 C(OO)、C++、C# 、 Java 、VB.Net、VHDL、Verilog 和 SystemC。

软件代码可以从状态机、序列和活动图生成，硬件描述语言可以从状态机图生成（使用Legacy状态机模板）。

访问

功能区	开发>源代码>生成

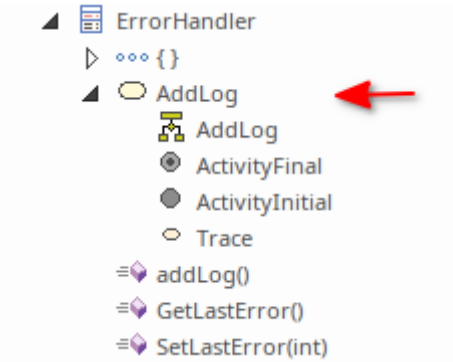
行为模型细节

三种关键类型的行为模型支持行为模型代码生成；但是，每种行为模型类型根据所涉及的元素类型都有自己的特征。这些主题为使用的核心元素类型提供指导和参考。

类型	描述
活动	关键操作类型的概述以及在代码生成中使用它们的详细信息。
交互	详细信息片段使用代码生成的交互（序列）图。
状态机	详细介绍了定义使用状态生成的代码的选项，包括行为 - 进入/退出/执行，以及状态机中的转换。

结构

行为模型代码生成主要要求所有行为构造都包含在类中（作为该类的子类）。



如果任何行为构造引用当前包之外的外部元素，则必须从当前包添加一个导入连接器到包含外部元素的包。更多细节见包图帮助主题中的导入连接器类型。

使用生成项目从行为图中生成代码

节	行动
1	通过选择 开始>帮助>帮助>打开示例模型"功能区选项打开帮助文件。
2	从浏览器窗口中，选择以下任何一个包： **软件语言示例：** ● 示例模型>软件工程> Java模型With Behaviors 生成账户和订单类 ● 示例模型>系统工程>实施模型>软件> C# 生成 DataProcessor类 ● 示例模型>系统工程> SysML示例>实施模型>软件> C++ 生成 IO类 ● 示例模型>系统工程> SysML示例> 实施模型> 软件 > Java 生成 IO类 ● 示例模型>系统工程> SysML示例> 实施模型> 软件 >示例 生成 IO类 **硬件语言示例：** ● 示例模型>系统工程> SysML示例：便携式音频播放器 > 实施模型> 硬件 > SystemC 生成 PlayBack类 ● 示例模型>系统工程> SysML示例：便携式音频播放器 > 实施模型> 硬件 > VHDL 生成 PlayBack类 ● 示例模型>系统工程> SysML示例：便携式音频播放器 > 实施模型> 硬件 > Verilog 生成 PlayBack类
3	完成时： ● 选择用于生成的类 ● 按 Ctrl+E 打开生成的源代码。 您应该会看到代码中生成的方法。

注记

- Enterprise Architect统一版和终极版中提供了从行为模型生成软件代码
- Enterprise Architect统一版和终极版中提供了从状态机模型生成硬件代码
- 对于 C(OO)，在 管理模型选项"对话框的 C 规格"页面上，将 物件导向支持"选项设置为True 。
  请参阅*C Options* -模型帮助主题。
- 行为代码不支持代码同步。

代码生成 -活动图表

从类中的活动图表生成代码需要一个验证相，在此过程中， Enterprise Architect使用系统工程图优化器来分析图表并将其呈现为可以生成代码的各种结构。 Enterprise Architect交互）。

行动

行动	描述
调用行动(行动调用行动)	用于调用活动图中的操作或行为；行为代码生成支持的调用行动的两个主要变体是： ● 行动——用于调用操作，可以在同一个类中，也可以在同一个包内的其他类中；如果在同一个包中引用其他类的操作，你必须有一个请求被传递到的目标 ● 行动——用于在活动流中调用另一个活动；被引用的活动应该在同一个类中 **论据** 调用行动可以指定与关联行为或行为特征中的参数对应的参数值。 您可以手动添加参数或使用 参数”对话框的 同步”按钮自动创建它们。
创建对象动作	用于表示活动流中的object创建；您可以使用行动元素的属性窗口将销的结果设置为要创建的object 。 CreateObjectAction 的分类器分类器要为其创建实例的分类器。
销毁对象动作	用于表示活动流中的object删除；您可以使用行动元素的属性窗口将DestroyObjectAction 的目标销设置为要销毁的object 。
循环	Enterprise Architect的系统工程图优化器还能够分析和识别循环；已识别的循环在内部呈现为行动循环，由 EASL 代码生成宏转换以生成所需的代码。 您可以有单个循环、嵌套循环和多层嵌套循环。
条件语句	对于一个条件语句模型，您使用决策/合并节点。 或者，您可以在内部暗示决策/合并；图优化器期望每个决策节点都有一个关联的合并节点，以促进对各种分支的有效跟踪和对其中代码结构的分析。

注记

- 为了能够从行为模型生成代码，所有行为构造都应该包含在一个类中

代码图表-交互

在从类（序列）生成代码的过程中， Enterprise Architect交互其系统工程图优化转换为程序化类。消息和片段根据它们的功能被标识为几种动作类型中的两种， Enterprise Architect使用代码生成模板来相应地呈现它们的行为。

行动

行动	描述
行动调用	调用操作A信息。
行动创造	具有生命周期A信息= 新。
行动毁灭	带有生命周期A信息=删除。
行动循环	类型= Alt A组合片段。
行动如果	类型= 循环A组合片段。
分配给	调用信息具有使用 分配给”字段设置A有效目标属性在代码中呈现为调用行动的目标属性。

注记

- 为了能够从行为模型生成代码，所有行为构造都应该包含在一个类中
- 对于一个交互（序列）图，行为代码生成引擎期望元素序列图和所有相关的交互信息片段被封装在一个交互内

代码生成 -状态机

状态机说明了A object （由一个类表示）如何改变状态，每个状态的改变都是由事件引起的触发器发起的转换，通常在定义为守卫的条件或约束下。当您模型object如何更改状态时，您可以使用适当的软件语言生成和构建（编译）代码并执行代码，通过模型模拟器将执行可视化。

在Enterprise Architect中，还可以组合独立但相关对象的状态机，以查看它们如何交互（通过广播事件），并从模型的变体快速创建和生成代码。例如，您可能会模型以下行为：

- 后轮驱动和前轮驱动模式下车辆的后轮外轮（一个状态机）
- 四轮驱动模式下车辆的方向盘和所有四个驱动轮（五状态机）
- 跑车的车轮和越野车的车轮（两个工件状态机、一个组合的实例）

对于这些设备的所有组件来说，生成和元素工件可执行状态机。这充当您的状态机模型的容器和代码生成单元。

您不使用此方法为硬件定义语言生成代码，但您也可以使用Enterprise Architect中的通用代码生成功能从状态机生成 HDL 代码和软件代码（请参阅生成源代码过程）。

先决条件

- 选择 设置>模型>选项>源代码工程”，对于适当的软件编码语言（ Java 、C、C#或ANSI C++），将 使用新的状态机模板”选项设置为 True ”
- 如果在 C++ 中工作，请选择 设置 >模型> 选项 >源代码工程 > C++”并将 C++版本”选项设置为 ANSI”

此代码生成方法不适用于在Enterprise Architect 11.0 之前开发的 Legacy状态机代码生成模板，也不适用于生成硬件定义语言代码。

访问

将工具图表工具页面中的 图表仿真工具箱”可执行状态机工件中。可以使用本表中列出的任何方法访问工具箱图表 仿真”页面。

功能区	设计>图表>工具箱>仿真
键盘快捷键	Ctrl+Shift+3 >仿真
其它	您可以通过单击 » « 图表工具箱显示或隐藏图形图表视图。

准备你的状态机图

节	行动
1	为每个想要模型的状态机创建一个类图。
2	从工具箱的 类”页面中，将 类”图标图表图表上并给元素起一个合适的名称。
3	右键单击类元素并选择 新子图表|状态机”上下文菜单选项。 给状态机图一个合适的名字。

4	创建状态机模型以反映状态之间的适当转换。

设置可执行状态机工件

节	行动
1	创建一个新的类图以包含您打算从中生成代码的建模状态机。
2	从名称 元素图表工具箱"页面，将 仿真可执行状态机"图表图表上，以创建工具元素并放大其工件。
3	从窗口中，首先将包含浏览器的元素状态机图类工件元素的浏览器窗口上。 '粘贴<元素名称>' 对话框显示。在 放下成"字段中，单击下拉箭头并选择值 属性"。 （如果对话框未显示，请在从浏览器窗口拖动类元素时按 Ctrl。）
4	点击确定按钮。工件在组件内部的类元素是一个部件。
5	对您想要组合并为其生成代码的任何其他具有状态机的类重复步骤 3 和 4。这些可能是： ● 重复相同类和状态机的 滴"，为平行对象建模 ● 不同的类和状态机，建模分离的交互对象
6	右键单击工件元素>属性属性选项，展开 高级"类别，在 语言"字段中，单击下拉箭头并将代码语言设置为与当前语言相同的语言为类元素定义。 您可以将可执行状态机从浏览器工件现在元素部件模型上，并用不同的编程语言将部件修改为任何数量的系统或过程的变体或相同的过程。

生成工件

节	行动
1	单击可执行状态机工件并选择 元素> 可执行状态仿真状态机 >生成"功能区选项。 将显示 可执行状态机代码生成"对话框。
2	在 项目输出目录"字段中，键入或浏览要在其下创建输出文件的目录路径。 在代码生成期间，此目录中的所有现有文件都将被删除。
3	选择目标系统。如果您在窗口上运行，请选择 本地"选项。如果您在 Linux 上工作，请选择 远程"选项。该选择会影响为支持仿真而生成的脚本。
4	在 <compiler> 安装目录位置"字段中，键入或浏览编译器安装目录的路径，以自动映射到本地路径（显示在字段左侧）。对于每种编程语言，路径可能类似于以下示例： ● Java JAVA_HOME C:\Program Files (x86)\ Java \jdk1.7.0_17 ● C/C++

	VC_HOME C:\Program Files (x86)\Microsoft Visual Studio 9.0 • C # 窗口C:\Window\Microsoft.NET\Framework\V3.5
5	点击生成按钮。代码文件是根据编程语言创建的。 系统输出显示 可执行状态机输出"选项卡，显示生成的进度和状态。 在代码生成期间，执行自动验证函数以根据UML约束检查图表或模型错误。任何错误都由 可执行状态机输出"选项卡上的错误消息标识。 双击错误信息可以显示发生错误的建模结构，并在重新生成代码之前更正错误。
6	当出现错误时，点击工件生成组件> 无仿真元素> 选择 可执行状态状态机>编译"功能区选项进行编译。 系统输出窗口显示一个 编译"选项卡，显示编译的进度和状态。请注意，编译包括模拟操作的配置。

代码生成宏

您还可以在状态机的代码生成中使用两个宏。

宏名称	描述
SEND_EVENT	将事件发送到接收器（部件）。例如： %SEND_EVENT("event1", "Part1")%
BROADCAST_EVENT	向所有接收者广播一个事件。例如： %BROADCAST_EVENT("event2")%

执行/仿真工件

节	行动
1	选择功能区选项 仿真>动态仿真仿真>模拟器>应用工作空间"，将仿真窗口和仿真事件窗口一起显示 将两个窗口停靠在屏幕的方便区域。
2	在图表窗口上，单击工件浏览器元素选择 仿真>运行状态> 状态机>"功能区选项。 该系列中的第一个状态机图显示了已经开始的过程模拟。在仿真窗口中，处理步骤以这种格式表示： [03516677] Part1[Class1].Initial_367_TO_State4_142影响 [03516683] Part1[Class1].StateMachine_State4 ENTRY [03516684] Part1[Class1].StateMachine_State4 DO [03518375] 被封锁
3	单击相应的仿真窗口工具栏按钮，按照您的喜好逐步完成仿真。 当仿真在退出或终止元素结束时，单击仿真窗口工具栏中的停止按钮。

4	在跟踪显示 Blocked 的情况下，模拟已达到必须发生触发器事件才能继续处理的点。在仿真事件窗口的 等待触发器"栏中，双击相应的 触发器"。 当触发器触发时，模拟继续到下一个暂停点，触发器或退出。

注记

- 如果您对现有状态机模型进行小幅更改，您可以通过选择 仿真> 可执行状态 > 状态机 >生成、构建和运行"功能区选项来组合代码生成、构建和运行操作
- 您还可以在JavaScript中生成代码

旧版状态机模板

代码生成使用一组生成模板进行操作。从Enterprise Architect 11.0 版开始，默认提供一组不同的模板，用于将软件代码从状态机图生成为Java 、C、ANSI C++ 或 C# 代码。如果您不想升级它们以获得新的模板功能，您仍然可以使用原始模板，如此处所述，用于在Enterprise Architect早期版本中开发的模型。

在 **Legacy** 和 **Release 11** 模板之间切换

访问

显示 管理模型选项"对话框，然后使用本表中列出的方法之一显示所选语言的 语言规范"页面。如有必要，展开 状态机工程（适用于当前模型）"分组并将 使用新状态机模板"选项设置为True （使用后面的模板）或False（使用旧版模板）。

功能区	设置>模型>选项>源代码工程> [语言名称]

旧版模板转换

类中A状态机在内部生成许多软件语言结构，以提供状态行为（执行、进入和退出）的有效执行，并在必要时对适当的转换效果进行编码。

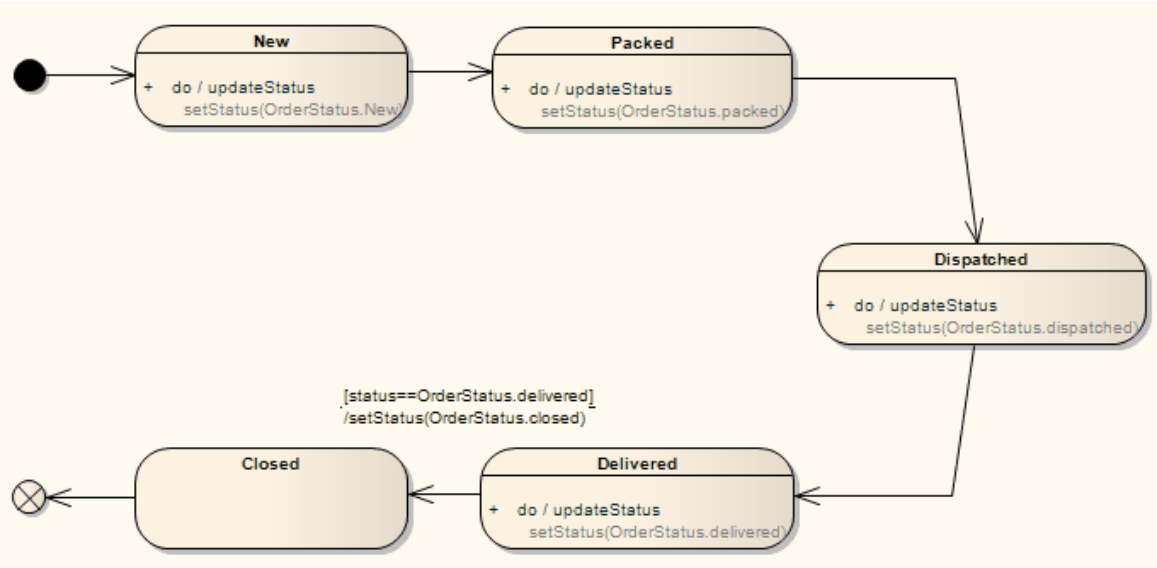
模型对象	代码对象
枚举	• StateType - 由状态机中包含的每个状态的枚举组成 • TransitionType – 由每个具有与其关联的有效效果的转换的枚举组成；例如，ProcessOrder_Delivered_to_ProcessOrder_Closed • CommandType – 包含一个状态可以包含的每个行为类型的枚举（Do、Entry、Exit）
属性	• currState:StateType - 保存当前状态信息的变量 • nextState:StateType - 保存下一个状态信息的变量，由每个状态的转换相应地设置 • currTransition:TransitionType - 保存当前转换信息的变量；如果过渡具有与之关联的有效效果，则设置此项 • transcend:Boolean - 一个标志，用于告知转换是否涉及不同状态机（或子机状态）之间的超越 • xx_history:StateType - 每个状态机/子机状态的历史变量，保存关于发生转换的最后状态的信息
操作	• 过程- 一个状态过程，包含一个状态的枚举和它的操作之间的映射；它取消引用当前状态的信息以调用相应状态的函数 • TransitionsProc - 一个过渡过程，包含转移 之间的映射转移 的列举及其影响；它调用了转移 的效果 • <<State>> -状态机中包含的每个状态的操作；这会根据输入

	CommandType 呈现状态的行为，并执行其转换 • initializeStateMachine - 一个初始化所有框架相关属性的函数 • 状态- 一个迭代每个状态的函数，并相应地执行它们的行为和转换

注记

- 为了能够从行为模型生成代码，所有行为构造都应该包含在一个类中

Java Code Generated From Legacy状态机模板



私有枚举状态类型： int

{

ProcessOrder_Delivered,

ProcessOrder_Packed,

ProcessOrder_Closed,

ProcessOrder_Dispatched,

ProcessOrder_New,

ST_NOSTATE

}

私有枚举TransitionType： int

{

ProcessOrder_Delivered_to_ProcessOrder_Closed,

TT_NOTRANSITION

}

私有枚举 CommandType

{

做，

入口，

出口

}

私有StateType currState；

私有状态类型下一个状态；

私有TransitionType currTransition；

私有布尔超越；

私有StateType ProcessOrder_history；

private void processOrder_Delivered（CommandType 命令）

```
{
开关（命令）
{
案例做：
{
// 做行为..
设置状态（已交付）；
//状态的转换
如果（（状态==已交付））
{
nextState = StateType.ProcessOrder_Closed;
currTransition = TransitionType.ProcessOrder_Delivered_to_ProcessOrder_Closed；
}
休息;
}
默认：
{
休息;
}
}
}
private void processOrder_Packed（CommandType 命令）
{
开关（命令）
{
案例做：
{
// 做行为..
设置状态（打包）；
//状态的转换
nextState = StateType.ProcessOrder_Dispatched；
休息;
}
默认：
{
休息;
}
}
}
private void processOrder_Closed（CommandType 命令）
{
开关（命令）
```

```
{
案例做：
{
// 做行为..
//状态的转换
休息;
}
默认：
{
休息;
}
}
}
private void processOrder_Dispatched(CommandType 命令)
{
开关（命令）
{
案例做：
{
// 做行为..
设置状态（已调度）；
//状态的转换
nextState = StateType.ProcessOrder_Delivered；
休息;
}
默认：
{
休息;
}
}
}
私人void processOrder_New（CommandType 命令）
{
开关（命令）
{
案例做：
{
// 做行为..
设置状态（新）；
//状态的转换
nextState = StateType.ProcessOrder_Packed；
休息;
```

```
}
默认：
{
休息;
}
}
}
私人void StatesProc（StateType currState，CommandType命令）
{
开关（当前状态）
{
案例 ProcessOrder_Delivered：
{
processOrder_Delivered（命令）；
休息;
}
案例 ProcessOrder_Packed：
{
processOrder_Packed（命令）；
休息;
}
案例 ProcessOrder_Closed：
{
processOrder_Closed（命令）；
休息;
}
案例 ProcessOrder_Dispatched：
{
processOrder_Dispatched（命令）；
休息;
}
案例 ProcessOrder_New：
{
processOrder_New（命令）；
休息;
}
默认：
休息;
}
}
私人void TransitionsProc（TransitionType过渡）
{
```

```
开关（过渡）
{
案例 ProcessOrder_Delivered_to_ProcessOrder_Closed：
{
设置状态（关闭）；
休息;
}
默认：
休息;
}
}
私人void初始化状态机（ ）
{
currState = StateType.ProcessOrder_New；
nextState = StateType.ST_NOSTATE；
currTransition = TransitionType.TT_NOTRANSITION;
}
私人void运行状态机（ ）
{
而（真）
{
if (currState == StateType.ST_NOSTATE)
{
休息;
}
currTransition = TransitionType.TT_NOTRANSITION;
StatesProc(currState, CommandType.Do);
// 然后检查在 do 行为之后是否分配了任何有效的转换
if (nextState == StateType.ST_NOSTATE)
{
休息;
}
if (currTransition != TransitionType.TT_NOTRANSITION)
{
TransitionsProc(currTransition);
}
if (currState != nextState)
{
StatesProc(currState, CommandType.Exit);
StatesProc(nextState, CommandType.Entry);
当前状态 = 下一个状态；
}
```

```
}
}
```

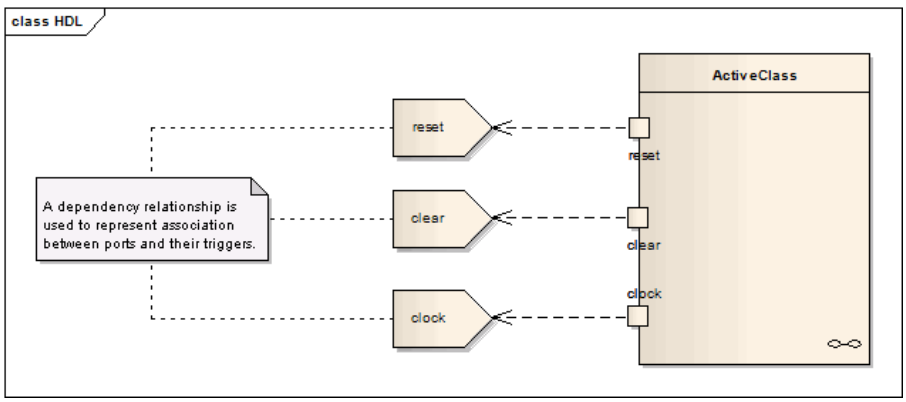
状态机建模For HDLs

要从状态机模型高效地生成硬件描述语言 (HDL) 代码，请应用本主题中描述的设计实践。硬件描述语言包括 VHDL、Verilog 和 SystemC。

在 HDL状态机模型中，您可能期望：

- 指定驾驶触发器
- 建立端口-触发器映射
- 添加到活动状态逻辑

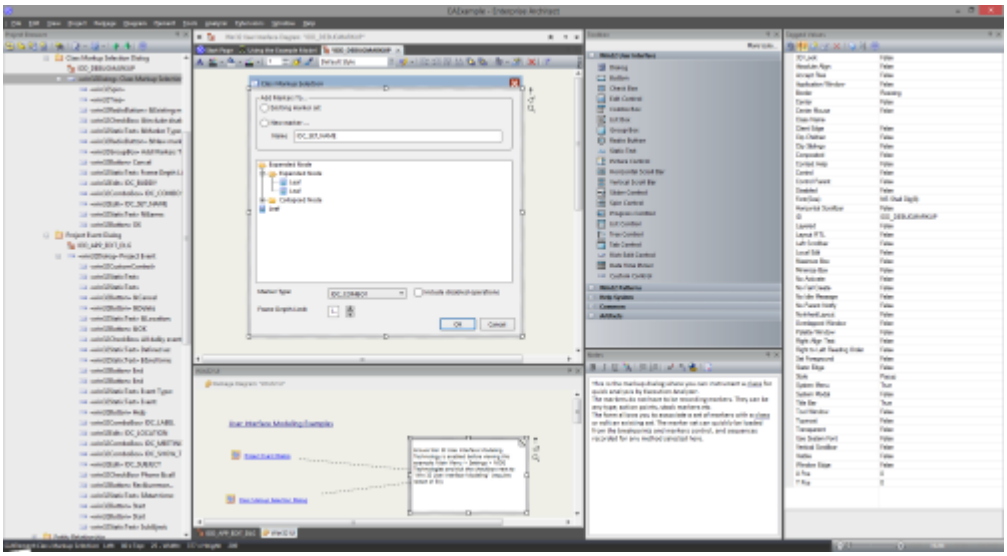
操作

手术	描述
指定驾驶触发器	• A以下情况下， 更改"触发器被视为异步触发器： - 从实际的状态（其中 封装了它触发的实际逻辑），并且 - 该转换的目标状态触发了自转换 同样触发器 • 触发器应该按照这种模式建模： - 简单的触发器应该是更改（规范： True / False ） - 活动状态（子机状态）应该有一个过渡 由它触发 - 触发转换的目标状态应该有一个自我 用同样的触发器 • A 触发器"时间，触发转换到活动状态（子机器状态），被认为是时钟；简单来说此触发器的规范应符合目标语言： - VHDL - 上升沿/下降沿 - Verilog - posedge / negedge - SystemC - 正/负
建立端口触发映射	在成功地对组件的不同操作模式进行建模，并将触发器与它们关联起来后，您必须将触发器与组件的端口相关联。 从端口到关联A触发器关系用于表示该关联。 
主动状态逻辑	指定驱动触发器并建立 Port-Trigger 映射为有效解释硬件组件所需的准备工作做好了准备。 我们现在模型Active (现在)状态中的实际状态机逻辑。

注记

- 为了能够从行为模型生成代码，所有行为构造都应该包含在一个类中
- 当前的代码生成引擎只支持一个组件的时钟触发器

Win32用户接口对话框



使用MDG Win32 UI技术，您可以设计呈现为 Win32® 控件的用户界面屏幕。生成的用户界面可用于任何资源定义脚本。资源定义脚本或 RC 文件是一种 Microsoft 技术，与其他代码一样，可以编译本机桌面应用程序使用的资产。用户界面屏幕或对话框可以从头开始创建或反向工程。用户界面模型也可以使用同步代码函数（F7）进行正向工程。接口建模以与在Enterprise Architect中使用任何技术完全相同的方式在图表上进行。 Enterprise Architect中用户接口设计的一个有趣方面是组件可以在状态机和活动的模拟中发挥积极作用，使模拟能够与用户交互，就像一个真实的程序一样！

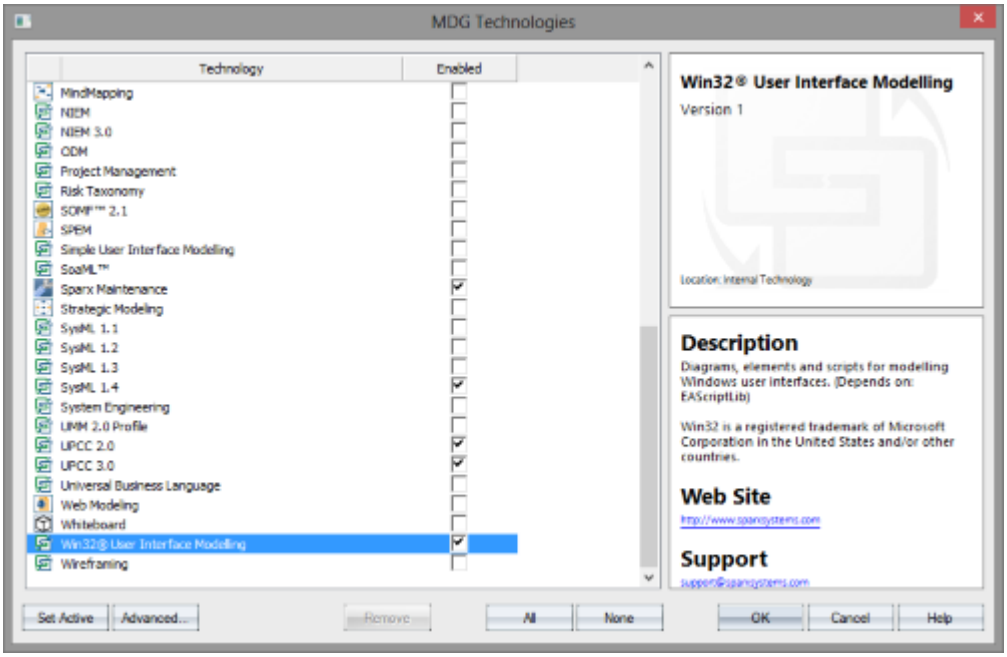
访问

功能区	设计>图表>添加图表>类型>用户接口Win32
上下文菜单	右键单击包|添加图表>类型|用户接口Win32
其它	浏览器窗口标题栏菜单 |新图表|用户接口Win32

支持

Enterprise Architect专业版、企业版、统一版和终极版中提供MDG Win32®用户接口技术

开启Win32用户接口技术



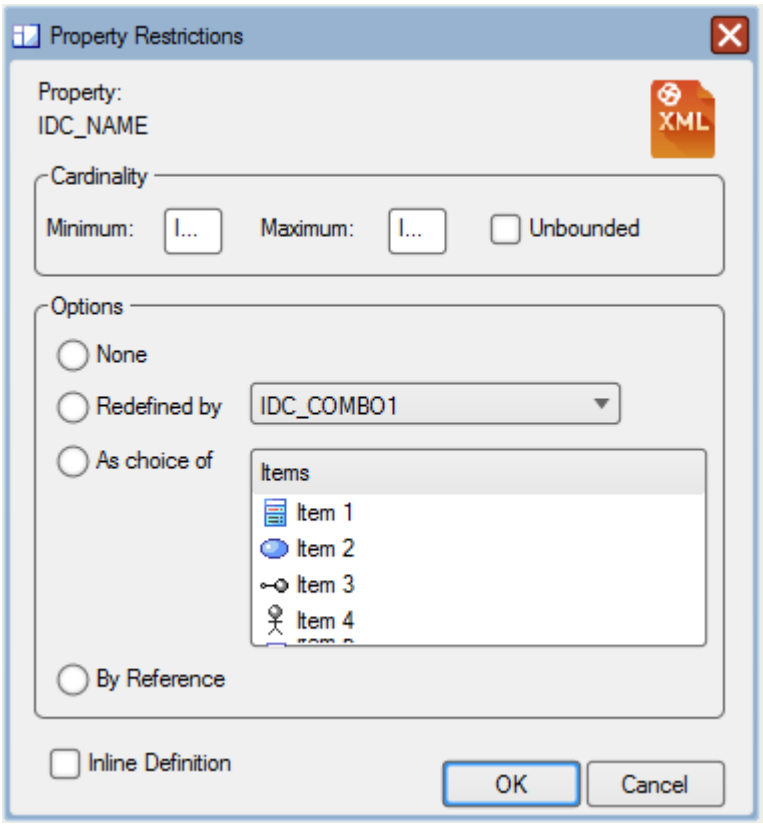
使用“MDG 技术”对话框（选择 特定技术> 管理技术”功能区选项）启用或禁用Enterprise Architect中的 Win32® UI技术。

默认技术

您可以将MDG Win32® UI技术设置为活动的默认技术，以直接访问工具箱页面。

建模UI对话框

Win32 用户接口用户MDG 技术提供了帮助您设计用户界面的工具，该界面与窗口对话框的视觉风格和可用选项非常相似。



Win32对话框

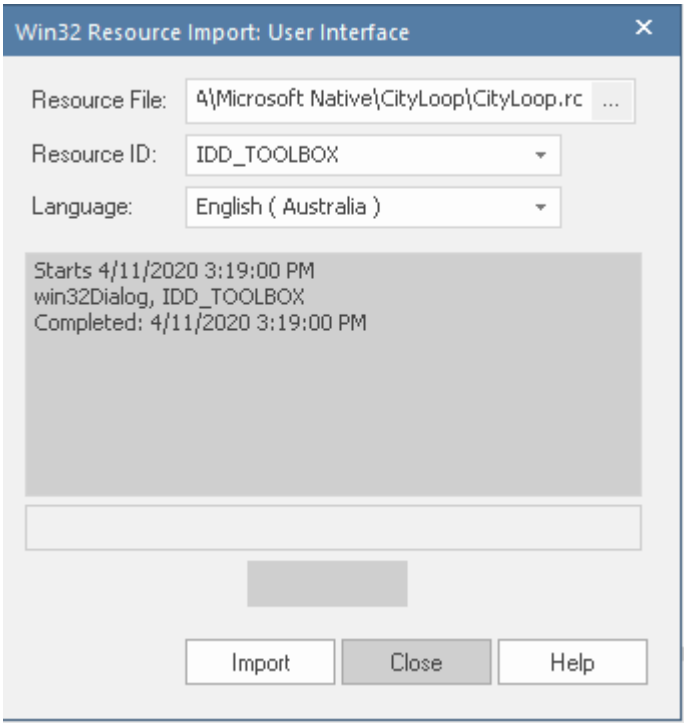
支持这些用户界面组件，每一个都匹配同名的 RC 资源。

部件	细节
win32对话框	相当于 RC 格式的 DIALOG 和 DIALOGEX 资源。
win32静态文本	相当于RC格式的LTEXT、RTEXT、CTEXT资源。
win32编辑	相当于 RC 格式的 EDITTEXT 资源。
win32按钮	相当于RC格式的BUTTON、DEFPUSHBUTTON等资源。
win32复选框	相当于 RC 格式的 CHECKBOX 资源。
win32滚动条H	SBS_HORZ 样式的 RC 格式 SCROLLBAR 资源的等效项
win32滚动条V	具有 SBS_VERT 样式的 RC 格式 SCROLLBAR 资源的等效项。
win32组框	相当于 RC 格式的 GROUPBOX 资源。

win32组合框	相当于 RC 格式的 COMBOBOX 资源。 注记：当您最初将“组合框”图标（类型为“下拉”或“下拉列表”）拖到图表上时，元素两侧的中间“跟踪手柄”为白色，表示您可以只调整元素的宽度。要调整元素的高度和宽度，请单击图像的下拉箭头部分；底部边缘中间的'tracking handle'现在是白色的，表示可以向下拖动base来设置虚拟高度（当元素展开以显示下拉列表中所有可能的值时的高度）。
win32列表框	相当于 RC 格式的 LISTBOX 资源。
win32单选按钮	相当于 RC 格式的 RADIOBUTTON 资源。
win32TabPane	RC 格式 TABPANE 资源的等效项。
win32图片	具有 SS_BITMAP 样式的 RC 格式 STATIC 资源的等效项。 从您的模型应用该控件时，可以渲染图像。可以通过先选择图像并按 Ctrl+Shift+W 显示图像管理器来应用图像。之后，您可能需要在相应的标记值。
win32自定义控件	相当于 RC 格式的 CONTROL 资源。

从 RC文件导入单个对话框

您可以按名称快速导入单个对话框。



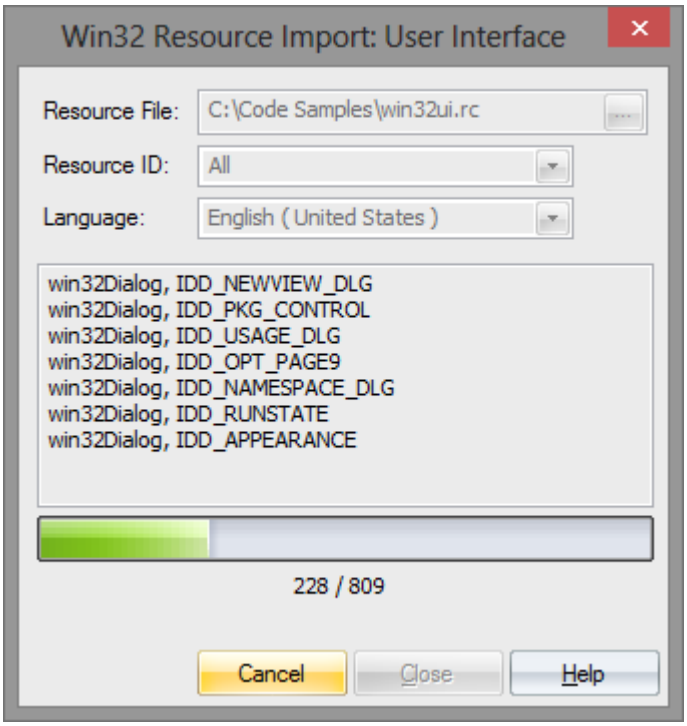
访问

在浏览器窗口中，单击目标包。

功能区	开发>源代码>文件>导入资源脚本

从 RC文件导入所有对话框

单个 RC 文件中的所有对话框都可以导入到您的模型中。此图像是在导入一分钟后捕获的，此时已导入 200 多个大型对话框定义。

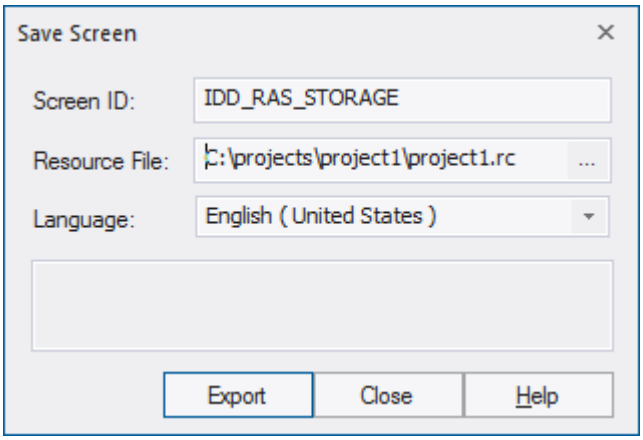


访问

功能区	开发>源代码>文件>导入资源脚本

导出对话框到 RC文件

一旦修改了屏幕设计或创建了新的屏幕设计，您可能希望将其恢复为用于构建应用程序的 RC 文件，以便您可以看到它在真实数据中的外观。首先在浏览器窗口中选择 Win32Dialog元素，然后使用功能区执行同步。



访问

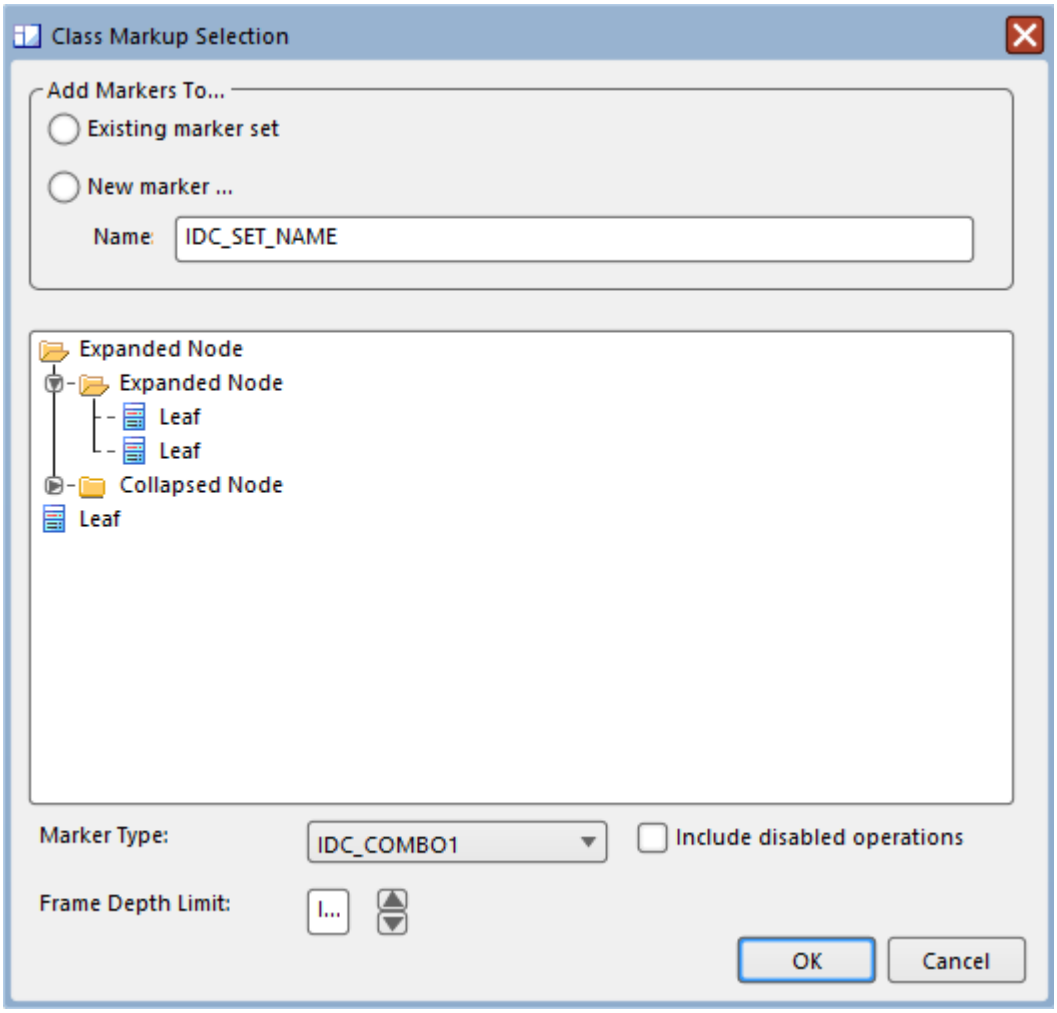
单击 win32Dialog元素。

功能区	开发>源代码>生成>生成单个元素
键盘快捷键	F11

设计一个新的对话框

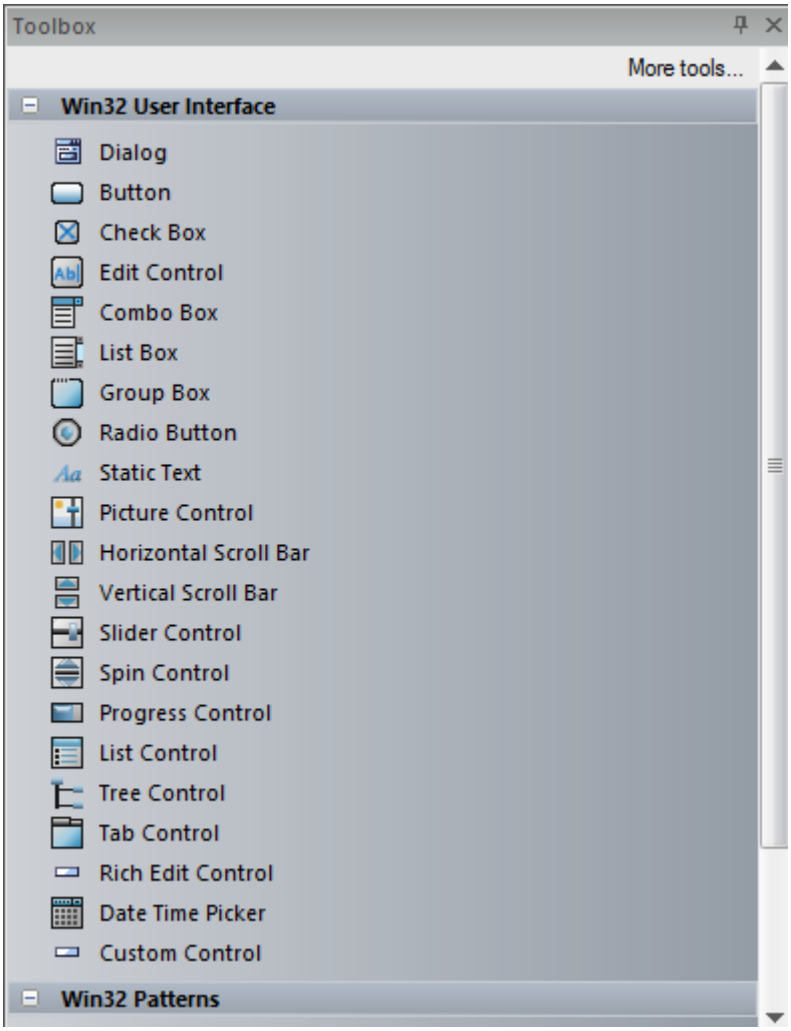
创建一个新的 Win32 对话框很容易，而且主要是可视化的。您可能需要一个显示：

- 新图表（选择'设计>图表>添加图表>用户接口- Win32 >用户接口- Win32'功能区路径）
- 接口用户工具箱（选择 设计>图表>工具箱"功能区选项）和
- 属性窗口的标记值选项卡



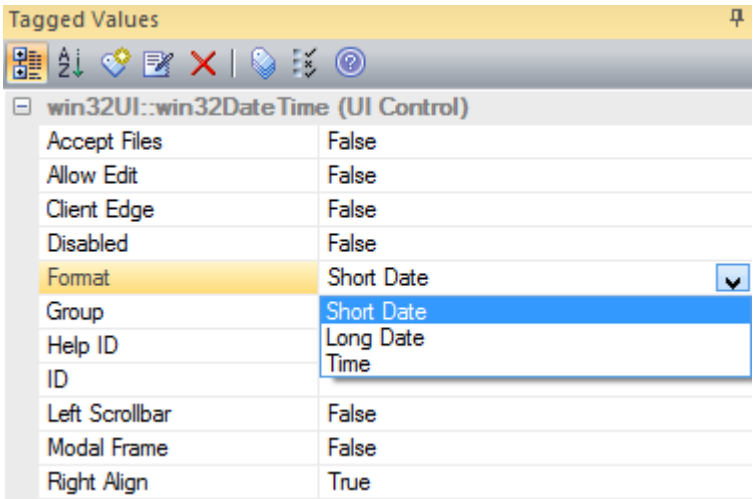
UI工具箱

所有常见的 RC 元素都可以在 UI 工具箱中找到



标签选项卡

此选项卡在object的属性窗口和 属性”对话框中提供，并且可以在其中查看和编辑控件的所有属性。



使用图片控件

可以通过选择对话框上的控件并按 Ctrl+Shift+W 来应用模型中的图像（请参阅图像管理器）。您可能需要在相应的标记值。

注记

- 可以复制粘贴对话框包

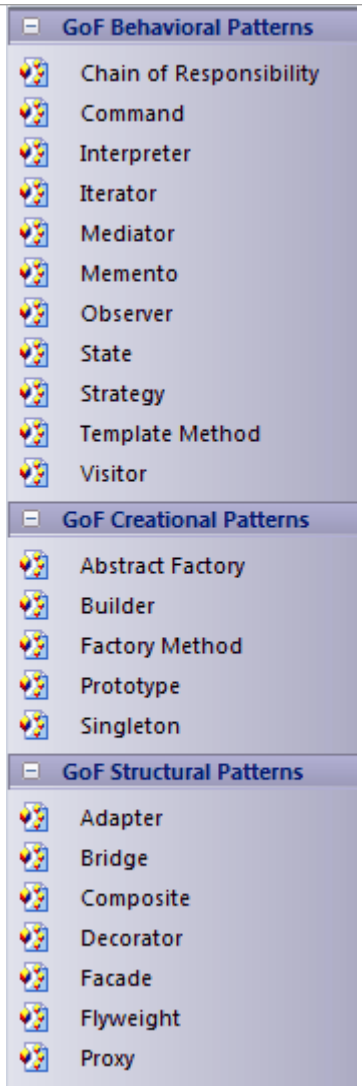
四人帮 (GoF)模式

设计模式是用于解决经常出现的设计问题A模板；它由一系列元素和连接器组成，可以在新的时间上下文使用。使用模式的优点是它们已经在许多上下文中经过测试和改进，因此通常是常见问题的稳健解决方案。Enterprise Architect将 Gang of Four模式作为MDG 技术提供，可以加载到当前存储库中。

四人组 (Gof)模式是一组 23 种设计模式，最初发表在一本名为《设计模式：元素可重用的面向对象软件》的开创性书籍中； 四人帮"一词指的是四位作者。 Enterprise Architect在其模式引擎中显示这些模式，帮助您可视化模式的元素并根据您的软件设计问题的上下文调整模式。

GoF模式在Enterprise Architect

特征	描述
GoF模式功能	GoFa模式以以下形式提供： ● 工具箱中的 GoF 行为模式、GoF 创建模式和 GoF 结构模式页面 ● 工具箱快捷菜单中的四种模式条目组 **GoF模式工具箱Pages** 您可以通过单击工具箱以显示 查找工具箱项"对话框并指定 GoF模式"来访问的 GoF模式"页面；这些图标可用：



当您将其中一个模式元素拖到新图表上时，将显示 模式<pattern group><pattern type>”对话框；如有必要，修改组件元素的动作和/或默认值，然后单击确定按钮以创建基于模式的图表。

ICONIX

ICONIX 过程是一种基于UML的专有软件开发方法。该过程是用例驱动的，并使用基于 UML 的图表来定义四个里程碑。该过程的主要特征是一个称为稳健性建模的概念，它基于 Ivar Jacobson 的早期工作，有助于弥合分析和设计之间的差距。

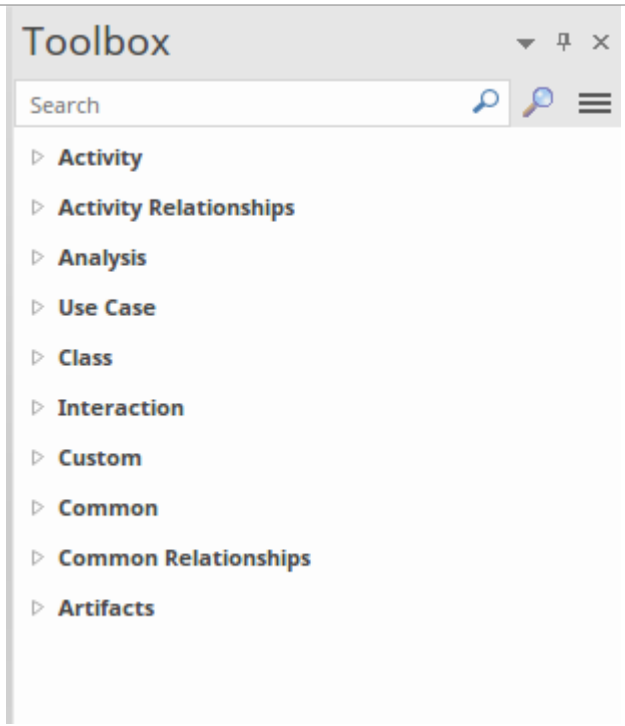
此文本来自在线 Wikipedia 中的 ICONIX 条目：

'ICONIX进程是一种极简、流线型的用例驱动UML建模方法，它使用UML图和技术的核心子集来提供面向对象分析和设计的全面覆盖。它的主要活动是稳健性分析，这是一种弥合分析和设计之间差距的方法。稳健性分析通过确保将用例描述写在相关域模型的上下文中来减少用例描述中的歧义。这个过程使用例更容易设计、测试和估计。

ICONIX进程由 Doug Rosenberg 开发；有关 ICONIX 的更多信息，请参阅 ICONIX软件工程公司。

方面

方面	细节
Enterprise Architect中的ICONIX	Enterprise Architect使您能够通过使用与Enterprise Architect安装程序集成的MDG 技术在 ICONIX 下快速简单地开发模型。 功能以以下形式提供： ● 工具箱中的A组工具箱页面 ● "工具箱快捷方 "菜单和快速链接器中的工具箱元素和关系条目 为了进一步帮助您开发和管理 ICONIX 下的项目， Enterprise Architect还提供了一份关于 ICONIX路线图的白皮书。
ICONIX工具箱Pages	在工具箱内， Enterprise Architect版本的页面为UML （序列）和活动图提供了分析、用例、类交互和自定义图（通常是鲁棒性图的基础）。 与标准工具箱页面相比，这些页面的元素和关系集略有不同；您可以通过以下任一方式访问它们： ● 在"查找工具箱项目"对话框中指定"ICONIX"并选择特定的工具箱页面 ● 在默认工具工具栏的下拉字段中选择"ICONIX"选项，这会将所有六个页面添加到工具箱；所有页面都关闭


Toolbox
Search
Activity
Activity Relationships
Analysis
Use Case
Class
Interaction
Custom
Common
Common Relationships
Artifacts

配置设置

您可以设置默认代码选项，例如Enterprise Architect可用的每种编程语言的编辑器，以及如何生成源代码或逆向工程的特殊选项。这些选项是根据它们是否适用于：

- 当前模型的所有用户，在 管理模型选项"对话框中设置，或
- 您访问的所有模型（其他用户可以定义适用于相同模型的自己的设置），在 首选项"对话框中设置

你也可以：

- 对于模型中使用的每种编程语言，对于使用模型的所有用户，定义集合类以从关联连接器生成代码，其中目标角色的多重性设置大于1
- 使用 本地路径"对话框为自己定义本地路径；这些设置适用于您访问的所有Enterprise Architect模型
- 在模型中定义语言宏，在逆向工程中很有用，并且可以从模型中导出和导入

源代码工程选项

源代码工程”选项适用 您从Enterprise Architect生成代码的语言。它们分为特定于模型的选项和特定于用户的选项，如此处所述。

特定于模型的选项

这些选项在 管理模型选项”对话框中定义。

访问

功能区	设置>模型>选项>源代码工程

期权类型

选项类型	细节
源代码生成选项	您可以在模型中定义许多用于生成代码的设置，例如用于生成代码的默认语言和用于代码生成的统一码字符集。
选项-物件Lifetimes	您可以配置有关物件生命周期的各种选项。
代码语言选项	对于Enterprise Architect支持的每种代码语言，您可以定义特定于模型的选项并设置所需的任何集合类。

用户特定选项

这些选项在 首选项”对话框中定义。

访问

在 首选项”对话框中，单击左侧列表中的 源代码工程”。

功能区	开始>外观>偏好>偏好
键盘快捷键	Ctrl+F9

期权类型

选项类型	细节
源代码生成选项	您可以定义许多设置以在您以相同用户 ID 访问的任何模型中生成代码。
代码编辑器	这些是访问和配置源代码编辑器的选项。
属性/操作	使用这些选项来配置属性和操作。
代码语言选项	对于Enterprise Architect支持的每种代码语言，您可以定义适用于您在用户 ID 下访问的任何模型的用户特定选项。

代码生成选项

当您为您的模型生成代码时，您可以设置某些选项。这些包括：

- 默认语言
- 是否为实现的接口生成方法
- 代码生成的统一码选项

访问

功能区	设置>模型>选项>源代码工程

配置代码生成选项

选项	行动
始终与现有文件同步（推荐）	选择单选按钮以将导入的代码与现有文件同步。
替换（覆盖）现有源文件	选择单选按钮以使用导入的代码覆盖现有的源文件。
部件类型	点击此按钮打开 导入组件类型”对话框，设置导入组件类型。
代码生成的默认语言	单击下拉箭头并选择代码生成的默认语言。
DDL名称模板	单击…按钮为主键、唯一约束、外键和外键索引名称模板定义模板名称。
关联属性的默认名称	类型是从导入的属性生成的默认名称。
实现接口的生成方法	选中复选框以指示为已实现的接口生成方法。
源编辑代码页	单击下拉箭头并选择要应用的适当统一码字符嵌入格式。

注记

- 配置这些设置是值得的，因为它们是模型中所有类的默认值；您可以使用自定义设置（来自 代码生成”对话框）在每个类的基础上覆盖其中的大部分

导入部件

使用 导入部件类型”对话框，您可以配置要为在导入源代码目录时找到的任何扩展名的文件创建哪些元素。

访问

功能区	设置>模型>选项>源代码工程：部件类型

定义导入部件

选项	行动
扩展	组件类型的扩展名中的类型。
类型	单击下拉箭头并选择组件类型。
构造型	类型在进一步标识此类型的组件的任何构造型名称中。
部件列表	列出当前定义的组件类型。
节省	单击此按钮可保存组件定义并将其添加到组件列表中。
新的	单击此按钮可清除对话框字段，以便您可以定义新的组件类型。
删除	单击此按钮可从组件列表中删除选定的组件类型。

注记

- 您可以使用 设置>模型>传输>导出参考”和 导入参考”功能区选项在模型之间传输这些导入组件类型

源代码选项

您可以设置多种选项以在您使用的模型中生成代码。这些包括：

- 如何格式化生成的代码
- 如何在代码生成期间响应某些事件
- 是否从代码生成图表

访问

在 首选项”对话框中，选择 源代码工程”选项

功能区	开始>外观>偏好>偏好
键盘快捷键	Ctrl+F9

配置代码生成选项

字段	行动
将长注释行换成	类型在将文本换行到下一行之前，注释行中允许的字符数。
图表导入自动布局	单击下拉箭头并选择是否以及何时在代码导入时自动生成图表。
默认布局图表	单击下拉箭头并选择布局类型以应用于从代码生成的图表。
输出文件同时使用 CR 和 LF	选中复选框以包括回车和换行；根据当前使用的操作系统设置此选项，因为代码可能无法正确呈现。
同步（倒车）时提示	选择复选框以在发生同步时显示提示。
从导入评论中删除硬中断	选中该复选框以从有关导入的评论部分中删除硬中断。
创建代码时自动生成角色名称	选中复选框以在创建代码时生成角色名称。
不要生成关联方向为 未指定”的成员	如果未指定关联方向，请选中该复选框以防止生成成员。
为操作返回和参数类型创建依赖关系	选中该复选框以生成操作返回和参数类型的依赖关系。
点评：生成	选中复选框以生成评论。
评论：反向	选中复选框以生成反向注释。
生成 Get/Set属性时删除	前缀中的类型，用分号分隔，用于变量命名约定，在变量对应的 get/set 函数

前缀	中删除。
当作后缀	选中该复选框以使用在 "生成获取/设置属性时删除前缀"字段中定义的前缀作为后缀。
属性的属性属性名称	选中复选框以属性属性的属性名称。
使用布尔属性Get()的'Is'	选中该复选框以对布尔属性Get() 使用 Is 关键字。

注记

- 配置这些设置是值得的，因为它们是模型中所有类的默认值；您可以使用自定义设置（来自 "代码生成"对话框）在每个类的基础上覆盖其中的大部分

选项-代码编辑器

您可以通过“首选项”对话框的“DDL”页面访问源代码编辑器选项。在此页面上，您可以为Enterprise Architect的内部编辑器以及 DDL 脚本的默认编辑器配置选项。您可以在每个语言选项页面上为代码语言配置外部编辑器。

访问

在“首选项”对话框中，选择“源代码工程>代码编辑器”选项。

功能区	开始>外观>偏好>偏好
键盘快捷键	Ctrl+F9

选项

选项	行动
DDL 编辑器	默认为空白，表示Enterprise Architect代码编辑器是正在使用的 DDL 编辑器。 如有必要，您可以选择不同的默认编辑器；单击[...]按钮以浏览并选择所需的 DDL 编辑器。然后编辑器名称显示在“DDL 编辑器”字段中。
默认数据库	单击下拉箭头并选择要使用的默认数据库。
MySQL储存引擎	单击下拉箭头并选择要使用的MySQL存储引擎。
如果没有设置外部编辑器，则使用内置编辑器	如果在用户特定选项中没有为该语言定义外部编辑器，则选中该复选框以将内置编辑器用于任何语言的代码。
显示行号	选中复选框以在编辑器中显示行号。
显示结构树	选中该复选框以显示带有解析打开文件结果的树（如果文件解析成功）。
对文件保存进行自动逆向工程	如果选中此复选框，则在源代码编辑器中按 Ctrl+S 保存会自动对代码进行逆向工程，其方式与保存源和重新同步类按钮所做的相同。
不要解析大于	单击下拉箭头并选择文件大小的上限进行解析。 设置此选项可防止由于解析非常大的文件而导致性能下降。
字体、样式和语法高亮	单击[...]按钮以显示“编辑器语言属性”对话框，您可以在其中设置全局和特定语言的编辑器语言属性。
配置Enterprise Architect文件关联	单击[...]按钮以显示“为程序设置关联”对话框，然后选择要通过Enterprise Architect文档处理程序打开的文件的文件扩展名。

编辑器语言属性

使用 "编辑器语言属性"对话框，您可以为Enterprise Architect在安装时支持的任何编程语言指定语法高亮属性。

访问

在 "首选项"对话框中，选择 "源代码工程|代码编辑器"选项并单击 "语法高亮选项"旁边的按钮。

功能区	开始>外观>首选项>首选项，选择 "源代码工程|代码编辑器"选项>点击 "语法高亮选项"旁边的按钮
其它	在代码编辑器窗口中，单击工具栏图标 |语法高亮选项

选项

控制板	描述
语言面板	对话框左侧的面板列出了您可以为其设置属性的语言。 列表顶部是三个非语言选项： ● （深色主题）- 为属性字段和代码编辑器屏幕中的代码面板分配深色背景（您可以为特定属性应用不同的颜色） ● （浅色主题）- 为属性字段和代码编辑器屏幕中的代码面板分配浅色背景（您可以为特定属性应用不同的颜色） 您还可以在 "应用程序外观"对话框中设置背景主题 ● （全局）提供可以为所有编程语言设置的属性；但是，您可以在特定语言的属性中将全局属性重置为特定语言的不同值 为一种语言重置全局属性不会影响该属性在其他语言中的值 单击列表中所需的语言，以显示该语言的属性： ● 以属性显示的属性表示这是可以定义该属性的最高级别（对于 "全局"以外的大多数语言选项，这实际上是定义该属性的唯一点） ● 以普通属性显示的属性通常是全局属性，您可以仅为当前语言重置
属性面板	滚动浏览语言的属性类别和单个属性。您可以根据需要折叠和展开类别，使用类别名称旁边的扩展框 ( )。 当您单击某个属性名称时，该属性的说明会显示在对话框右下方的面板中。 要定义属性，请单击属性名称后面的值字段；根据属性的类型，可以启用该字段以进行直接编辑，或者显示下拉箭头或按钮（如属性窗口的 "标签"选项卡所述），以便您可以选择值来定义属性. 选择或输入所需的值。 使用工具栏图标： ● 保存对属性的更改 ● 将所有属性字段重置为Enterprise Architect附带的默认设置 ● 将当前样式字段重置为默认设置（非样式字段不启用）

将键分配给宏	在属性的 宏”类别中，您可以将（Ctrl+Alt+<n>）按键组合分配给您在 源代码查看器”中自己创建的编码宏。 当您单击选定 宏”字段中的 浏览”按钮时，将显示 打开宏”对话框；此对话框列出现有的宏，如果已将组合键分配给宏，则该组合键是什么。 单击宏的名称，然后单击打开按钮将选定的键分配给宏。

注记

- 您目前无法为通过MDG 技术包含的任何其他语言设置属性
- 如果需要，您可以调整此对话框的大小

选项-物件Lifetimes

您可以使用这些选项来配置各种物件生命周期设置，例如：

- 生成代码时定义构造函数详细信息
- 指定是否创建复制构造函数
- 定义析构函数细节

访问

功能区	设置>模型>选项>源代码工程>物件Lifetimes

选项

选项	行动
构造函数	如有必要，请选中复选框以指定生成构造函数以及（对于 C++）该构造函数是内联的。 单击下拉箭头并选择默认构造函数的适当可见性 - Private、Protected 或 Public。
复制构造函数	如有必要，请选中复选框以指定生成复制构造函数以及（对于 C++）复制构造函数是内联的。 单击下拉箭头并选择默认复制构造函数的适当可见性 - Private、Protected 或 Public。
析构函数	如有必要，选中复选框以指定生成析构函数以及（对于 C++）析构函数是内联的和/或虚拟的。 单击下拉箭头并选择默认析构函数的适当可见性 - Private、Protected 或 Public。

选项 - 属性/操作

您可以通过多种方式配置属性和操作的使用。您可以将选项设置为：

- 删除反向同步时代码中不包含的模型属性
- 删除反向同步时代码中未包含的模型方法
- 在前向同步期间删除模型中包含的特征代码
- 删除模型反向同步时代码中未包含的属性对应的关联和聚合
- 定义逆向工程时是否包含方法体并保存在模型中
- 快速连续创建特征，单击 保存”时清除属性窗口，以便您可以输入另一个特征名称

您可以在 首选项”对话框的 属性/操作”页面上配置这些选项。

访问

在 首选项”对话框中，选择 源代码工程 > 属性/操作”选项。

功能区	开始>外观>偏好>偏好
键盘快捷键	Ctrl+F9

选项

字段	行动
在反向同步时，删除代码中没有的模型属性	选中该复选框以指示在反向同步时，模型中未包含在代码中的属性会自动从模型中删除。
在反向同步时，删除不在代码中的模型关联	选中复选框以指示在反向同步时，模型中未包含在代码中的关联会自动从模型中删除。
在反向同步时，删除代码中没有的模型方法	选中该复选框以指示在反向同步时，模型中未包含在代码中的方法会自动从模型中删除。
逆向工程时在模型中包含方法体	选中复选框以指示在逆向工程代码中，代码中的方法主体包含在您的模型中。
保存后，重新选择编辑过的项目	选中复选框，表示保存属性或操作后，属性定义继续显示所选特征的详细信息。 如果取消选中，表示属性定义的字段将被清除，以便您可以立即输入另一个属性或操作名称和详细信息。
在前向同步时，提示删除模型中没有的代码特征	选中该复选框以指示在前向同步期间，将显示 同步元素<包名称>.<元素名称>”对话框，以便您可以忽略、重新分配或删除代码中不在模型中的特征。

建模约定

UML模型和编程代码之间的同步是使用UML结构和编程代码语法之间的一组建模约定（映射）来实现的。建议软件工程师熟悉这些约定，以便使用他们打算针对的编程语言的代码生成过程。使用了一系列构造，包括元素、特征、连接器、连接器末端、刻板印象和标记值。新手将需要一点时间来熟悉这些约定，但很快他们就会毫不费力地在编程代码和UML结构之间进行转换。

支持的语言

语
行动脚本
Ada 2012（统一和终极版）
C
C #
C++
德尔福
Java
PHP
Python
SystemC（统一和终极版）
Verilog（统一和终极版）
VHDL（统一和终极版）
视觉基础
Visual Basic .NET

注记

Enterprise Architect为其支持的语言合并了许多可见性指标或范围值；其中包括：

- 所有语言 - 公共 (+)、受保护 (#) 和私有 (-)
- Java包(~)
- 德尔福 - 已发布 (^)
- C# - 内部 (~)、受保护的内部 (^)
- ActionScript - 内部 (~)
- VB.NET - 朋友 (~)、受保护的朋友 (^)
- 包(~)
- Python-包(~)
- C-包(~)
- C++-包(~)

ActionScript 约定

Enterprise Architect支持使用这些约定的 ActionScript 2 和 3 的round工程。

构造型

构造型	适用于
文字	手术 对应于：由变量引用A文字方法。
属性get	手术 对应于： A读取"属性。
属性集	手术 对应于： A 写"属性。

标记值

标签	适用于
属性名	使用构造型属性get 或属性集进行操作 对应于：该属性后面的变量的名称。
动态的	类或接口 对应于： 劲态"关键字。
最后	ActionScript 3：操作 对应于："final"关键字。
固有的	动作脚本 2：类 对应于： 内在"关键字。
命名空间	ActionScript 3：类、接口、属性、操作 对应于：当前元素的命名空间。
覆盖	ActionScript 3：操作 对应于： 覆盖"关键字。
原型	ActionScript 3：属性 对应于： 原型"关键字。
休息	ActionScript 3：参数 对应于：其余参数（ ... ）

公共约定

- 包限定符（ActionScript 2）和包（ActionScript 3）在当前包不是命名空间根时生成
- 未指定的类型被建模为 "var"或空的 类型"字段

ActionScript 3 约定

- 类的Is Leaf属性对应于sealed关键字
- 如果指定了命名空间标记，它将覆盖指定的范围

Ada 2012 年公约

Enterprise Architect支持使用这些约定的 Ada 2012 的round工程。

构造型

构造型	适用于
adaPackage	类 对应于：Ada 2012 中A包规范，没有标记记录。
ada程序	类 对应于：Ada 2012 中A过程规范。
代表	手术 对应于：访问子程序。
枚举	内在类 对应于：枚举类型。
结构	内在类 对应于A记录定义。
类型定义	内在类 对应于A类型定义、子类型定义、访问类型定义、重命名。

标记值

标签	适用于
方面	带有构造型 typedef 的内部类 手术 对应于：方面规范（子程序类型'invariant'，子类型'predicate'的前置条件和后置条件）。
实例化单元类型	带有构造型 typedef 的内部类 对应于：实例化单元的类型（包/过程/函数）。
是访问	参数 对应于：判断参数是否为访问变量。
别名	函数参数 对应于：别名函数参数。

判别式	带有构造型 typedef 的内部类 对应于：类型的判别式。
零件类型	带有构造型 typedef 的内部类 对应于：零件类型（ 重命名"或 新"）。
类型	带有构造型 typedef 的内部类 对应：如果'Value' = '子类型'，设置'subtype' 如果 值"= 访问"，则设置 访问类型"。

其它

- 合适的源文件类型：Ada 规范文件、.ads
- Ada 2012 导入包定义为 <<adaPackage>>类或类，基于 Ada 2012 选项中的设置
- A包包含 Tagged Record，则将其作为类导入，其名称由选项 使用类名称for Tagged Record"和 Alternate Tagged Record名称"控制；该标记记录中定义的所有属性都被吸收为类的属性
- Ada 规范文件中A过程/函数被认为是类的成员函数，如果它的第一个参数满足选项 参考参数样式"、 忽略参考参数名称"和 参考参数名称"中指定的条件
- 如果启用选项 参考Tagged Record"，则为类创建一个引用类型，其名称由选项 参考名称类型"确定；例如：

HelloWorld.ads

包HelloWorld是

类型 HelloWorld 被标记记录

Att1：自然；

整数：坚持；

结束记录；

-- 公共函数

函数MyPublicFunction (P: HelloWorld) return字符串；

Process MyPublicFunction (P1: in out HelloWorld;过程: Boolean);

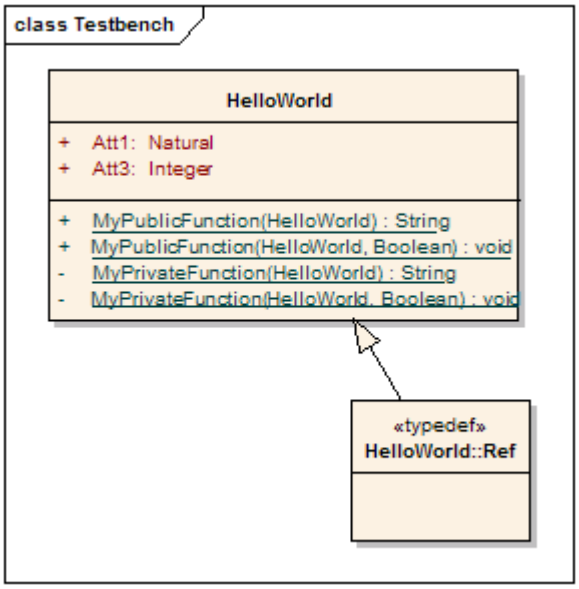
私人的

-- 私人函数

函数MyPrivateFunction (P: HelloWorld) return字符串；

过程MyPrivateFunction (P1: in out HelloWorld; AFlag: Boolean);

结束HelloWorld；



注记

- Enterprise Architect的统一版和终极版中提供了 Ada 2012 支持

C 约定

Enterprise Architect支持 C 的round工程，其中使用了以下约定：

构造型

构造型	适用于
枚举	内在类 对应于：枚举类型。
结构	内在类 对应于： A结构”类型。
属性	变量定义中A关键字结构。
类型定义	内在类 对应于A 'typedef' 语句，其中 parent 是原始类型名称。
联盟	内在类 对应于A联合类型。
属性	变量定义中A关键字 union。

标记值

标签	适用于
匿名的	还包含标记值typedef的类 对应于：这个类的名称仅由 typedef 语句定义。
位域	属性 对应于：允许存储此属性的大小（以位为单位）。
身体位置	手术 对应于：方法体生成的位置；预期值为 header、classDec 或 classBody。
类型定义	具有除 'typedef' 以外的构造型的类 对应于：在 类”语句中定义的此类。
类型同义词	类 对应于：“typedef”名称和/或此类型的字段。

UML模型的 C 代码生成

UML	C代码
A类	A对 C 文件 (.h + .c) 注记：文件名与类名相同
操作（公共和受保护）	.h 文件中的函数声明和 .c 文件中的定义 注记：
操作（私人）	仅在 .c 文件中的函数定义 注记：
操作（静态）	仅在 .c 文件中的函数定义 注记：不管作用域如何，静态函数只会出现在 .c 文件中。
属性（公共和受保护）	.h 文件中的变量定义 注记：
属性（私有）	.c 文件中的变量定义 注记：
内在类（无刻板印象）	A N ） 注记：这个inner类会被忽略

捕获要在 C 代码中生成的 #define 值

例如，#define PI 3.14。

节	进程
1	为类添加一个属性，名称= PI，初始值 = 3.14。
2	在 属性”页面的属性面板中，更新 静态”和 常量”字段。
3	在 属性”页面的 标记值”选项卡上，添加一个名为 定义”的标签，其值为True 。

注记

- 单独的约定适用于 C 中的面向物件的编程

C语言中的面向物件编程

在Enterprise Architect中，您在 C 中应用了许多面向对象编程的约定。

要将系统配置为支持使用 C 的面向对象编程，您必须在 "首选项"对话框的 "C 规范"页面上将 "物件面向支持"选项设置为True 。

构造型

构造型	适用于
枚举	类 对应于：枚举类型。
结构	类 对应于： A结构"类型。
属性	变量定义中A关键字结构。
类型定义	类 对应于A 'typedef' 语句，其中 parent 是原始类型名称。
联盟	类 对应于A联合类型。
属性	变量定义中A关键字 union。

标记值

标签	适用于
匿名的	具有 "枚举"、 "结构"或 "联合"刻板印象的类 对应于：这个类的名称仅由 typedef 语句定义。
身体位置	手术 对应于：方法体生成的位置；预期值为 "header"、 "classDec"或 "classBody"。
定义	属性 对应于：'#define' 语句。
类型定义	具有 "枚举"、 "结构"或 "联合"刻板印象的类 对应于：在 "类"语句中定义的此类。

UML模型的面向对象的 C 代码生成

在 C 代码中实现UML类的基本思想是将数据变量（ UML属性）分组为结构类型；此结构在 .h 文件中定义，以便其他类和引用它的客户端可以共享它。

UML类中的操作在 C 代码中实现为函数；函数名必须是一个完全限定名，由操作名和类名组成，以表明该操作是针对类的。

分隔符（ A “C 规范”页面上的 “命名空间分隔符”选项中指定）用于连接类名和函数（操作）名。

C 代码中的函数还必须具有类object的引用参数 - 您可以修改 “C 规范”页面上的 “参考操作参数”、 “参数样式”和 “参考参数名称”选项以支持此参考范围。

C 中面向对象编程的局限性

- 属性没有范围映射： UML类中的属性映射到 C 代码中的结构变量，其范围（私有、受保护或公共）被忽略
- 目前忽略了一个内部类：如果一个UML类是另一个UML类的内部类，则在生成C代码时忽略它
- 忽略初始值：在生成的 C 代码中忽略UML类中属性的初始值

C# 约定

Enterprise Architect支持使用这些约定的 C# 的round工程。

构造型

构造型	适用于
枚举	类 对应于：枚举类型。
事件	手术 对应：事件。
扩大	手术 对应： A类扩展方法，在代码中由签名中的'this'参数表示。
索引器	手术 对应于： A此类类的属性。
部分的	手术 对应于：操作上的 ‘部分”关键字。
属性	手术 对应于：可能包含读写代码A属性。
记录	类 对应于： A记录”类型。
结构	类 对应于： A结构”类型。

标记值

标签	适用于
参数名称	带有构造型扩展的操作 对应于：赋予此参数的名称。
属性名	使用构造型属性或事件进行操作 对应于：此属性或事件背后的变量名称。
班级名称	带有构造型扩展的操作 对应于：添加此方法的类。

常量	属性 对应于： const 关键字。
定义	使用刻板印象部分操作 对应于：这是方法的声明，还是定义。
代表	手术 对应于：'delegate' 关键字。
枚举类型	使用构造型属性操作 对应于：属性表示为的数据类型。
表达体	操作，使用构造型属性或索引器的操作 如果 行为代码”来自表达式主体函数成员，则对应于：“ True ”。
扩展属性	使用构造型扩展进行操作。 对应于：赋予此参数的属性。
外部	手术 对应于：“extern”关键字。
固定的	属性 对应于： 固定”关键字。
通用的	手术 对应于：此操作的通用参数。
通用约束	模板类或接口，带有 通用”标签的操作 对应于：对该类型或操作的泛型参数的约束。
工具	手术 对应于：此实现的方法的名称，包括接口名称。
实现显式	手术 对应于：此方法声明中源接口名称的存在。
初始化器	手术 对应于A构造函数初始化列表。
新的	类，接口，操作 对应于： 新”关键字。
覆盖	手术 对应于： 覆盖”关键字。
参数	参数 对应于：使用 “params”关键字A参数列表。

部分的	类,接口 对应于： 部分"关键字。
属性初始化器	使用构造型属性操作 对应于： A属性初始化器。
只读	操作, <<struct>>类 对应于： 只读"关键字。
位置参数	<<记录>>类 对应于：记录定义中的位置参数。
参考	操作, <<struct>>类 对应于："ref"关键字。
密封	手术 对应于： 密封"关键字。
静止的	类 对应于： 静态"关键字。
不安全	类，接口，操作 对应于： 不安全"关键字。
虚拟的	手术 对应于： 虚拟"关键字。
只写	使用构造型属性操作 对应于：此属性仅定义 写入"代码。

其它

- 为命名空间根下的每个包生成命名空间
- 属性的 const 属性对应 readonly 关键字，而标签 const 对应 const 关键字
- 一个参数的 Kind属性的 inout 的值对应 ref 关键字
- 一个参数的 Kind属性的 out 值对应 out 关键字
- 部分类可以建模为带有部分标签的两个单独的类
- 类的Is Leaf属性对应于sealed关键字

C++ 约定

Enterprise Architect支持 C++ 的round工程，包括使用这些约定的托管 C++ 和 C++/CLI 扩展。

构造型

构造型	适用于
枚举	类 对应于：枚举类型。
朋友	手术 对应于： 朋友”关键字。
属性get	手术 对应于： A读取”属性。
属性集	手术 对应于： A 写”属性。
结构	类 对应于： A结构”类型。
类型定义	类 对应于A 'typedef' 语句，其中 parent 是原始类型名称。
别名	类 对应于 别名”声明，其中父项是原始类型名称。
联盟	类 对应于A联合类型。

标记值

标签	适用于
afx_msg	手术 对应于： afx_msg 关键字。
匿名的	还包含标记值typedef的类 对应于：这个类的名称仅由 typedef 语句定义。
属性名	使用构造型属性get 或属性集进行操作 对应于：该属性后面的变量的名称。

位域	属性 对应于：允许存储此属性的大小（以位为单位）。
身体位置	手术 对应于：方法体生成的位置；预期值为 header、classDec 或 classBody。
打回来	手术 对应于：对 CALLBACK 宏A引用。
常量表达式	属性与操作 对应于： constexpr 关键字。
明确的	手术 对应于： 显 ”关键字。
初始化器	手术 对应于A构造函数初始化列表。
排队	属性与操作 对应于T 'inline' 关键字和成员变量定义和方法体的内联生成。
可变的	属性 对应于： 可变”关键字。
范围	具有构造型枚举的类 对应于： 类”或 结构”关键字。
投掷	手术 对应于：此方法引发的异常。
类型定义	具有除 'typedef' 以外的构造型的类 对应于：在 类”语句中定义的此类。
类型同义词	类 对应于：“typedef”名称和/或此类型的字段。
易挥发的	手术 对应于：“volatile”关键字。

其它

- 为命名空间根下的每个包生成命名空间
- 通过参考属性对应一个指向指定类型的指针
- 一个属性的属性对应的是volatile关键字
- 属性的抽象属性对应虚拟关键字

- 一个操作的Const属性对应const关键字，指定一个常量返回类型
- 操作的Is查询属性对应const关键字，指定方法不修改任何字段
- 操作的 Pure属性对应于使用 ≝0″语法的纯虚方法
- 一个参数的Fixed属性对应const关键字

托管 C++ 约定

这些约定用于 C++/CLI 之前的 C++ 托管扩展。为了将系统设置为生成托管 C++，您必须在 C++ 选项中修改 C++ 版本。

构造型

构造型	适用于
属性	手术 对应于："__property"关键字。
属性get	手术 对应于："__property"关键字和读取属性。
属性集	手术 对应于："_ _property"关键字和 写入"属性。
参考	类 对应于：'__gc' 关键字。
价值	类 对应于："__value"关键字。

标记值

标签	适用于
托管类型	具有原型引用、值或枚举的类；接口 对应于：该类型声明中使用的关键字；预期值为 类"或 结构"。

其它

- 不支持来自本机 C++ 的 typedef 和匿名标签
- 一个操作的 Pure属性对应于关键字 __abstract

C++/CLI 约定

这些约定用于对 C++/CLI 对 C++ 的扩展进行建模。为了将系统设置为生成托管 C++/CLI，您必须在 C++ 选项中修改 C++ 版本。

构造型

构造型	适用于
事件	手术 描述：定义一个事件以提供对此类事件处理程序的类。
属性	操作、属性 描述：这是一个可能包含读写代码的属性。
参考	类 描述：对应'ref class'或'ref struct'关键字。
价值	类 描述：对应 值类”或 值结构”关键字。

标记值

标签	适用于
属性名	使用构造型属性或事件进行操作 描述：此属性或事件背后的变量名称。
通用的	手术 描述：定义此操作的通用参数。
通用约束	模板类或接口，带有标签泛型的操作 描述：定义此操作的通用参数的约束。
初始化	属性 描述：对应于'initonly'关键字。
文字	属性 描述：对应文字关键字。
托管类型	具有原型引用、值或枚举的类；接口 描述：对应于 类”或 结构”关键字。

其它

- 不使用 typedef 和匿名标签
- 不使用属性get/属性型
- 一个操作的 Pure属性对应于关键字 abstract

德尔福约定

Enterprise Architect支持 Delphi 的round工程，其中使用了以下约定：

构造型

构造型	适用于
构造函数	手术 对应于A构造函数。
析构函数	手术 对应于A析构函数。
调度接口	类,接口 对应于： A调度接口。
枚举	类 对应于：枚举类型。
元类	类 对应： A元类类型。
object	类 对应于： object类型。
操作员	手术 对应于：操作员。
属性get	手术 对应于： A读取"属性。
属性集	手术 对应于： A 写"属性。
结构	类 对应于A记录类型。

标记值

标签	适用于
属性名	使用构造型属性get 或属性集进行操作 对应于：该属性后面的变量的名称。

超载	手术 对应于：'overload' 关键字。
覆盖	手术 对应于： 覆盖"关键字。
包装好的	类 对应于："packed"关键字。
属性	类 对应于： A属性；有关详细信息，请参阅*Delphi*属性。
重新引入	手术 对应于： 重新引入"关键字。

其它

- 属性或操作的静态属性对应于 类"关键字
- 参数的Fixed属性对应'const'关键字
- 参数的 Kind属性的 inout 值对应于 'Var' 关键字
- 参数的 Kind属性的 out 值对应于 输出"关键字

Java约定

Enterprise Architect支持使用这些约定的Java round工程——包括 AspectJ 扩展。

构造型

构造型	适用于
注解	接口 对应于：注释类型。
紧凑构造函数	手术 对应于：用于记录A'紧凑规范构造函数。
记录	类 对应于A记录类型。
默认	手术 对应于： 默认"关键字。
枚举	类定型枚举中的属性 对应于：一个枚举选项，区别于其他没有构造型的属性。
枚举	类 对应于：枚举类型。
操作员	手术 对应于：操作员。
属性get	手术 对应于： A读取"属性。
属性集	手术 对应于： A 写"属性。
静止的	类或接口 对应于： 静态"关键字。

标记值

标签	适用于
注释	任何事物 对应于：当前代码特征上的注释。

论据	带有构造型枚举的属性 对应于：适用于此枚举值的参数。
属性名	使用构造型属性get 或属性集进行操作 对应于：该属性后面的变量的名称。
动态的	类或接口 对应于： 劲态”关键字。
通用的	手术 对应于：此操作的泛型参数。
参数列表	参数 对应于：具有 ... 语法A参数列表。
记录头	<<记录>>类 对应于：记录定义的记录头。
投掷	手术 对应于：此方法引发的异常。
短暂的	属性 对应于： 瞬态”关键字。

其它

- 当前包不是命名空间时生成包语句根
- 属性或操作的属性对应于 final 关键字
- 一个属性的属性对应的是volatile关键字
- 一个参数的Fixed属性对应final关键字

AspectJ 约定

这些是用于支持Java的 AspectJ 扩展的约定。

构造型

构造型	适用于
建议	手术 对应于：AspectJ 方面A一条建议。
方面	类 对应于：AspectJ 方面。
切入点	手术 对应于：AspectJ 方面中A 切入点”。

标记值

标签	适用于
班级名称	类刻板印象方面内的属性或操作 对应于：此 AspectJ 互类型成员所属的类。

其它

- 切入点的规范包含在方法的 行为”字段中

PHP 约定

Enterprise Architect支持使用这些约定的 PHP 4 和 5 的round工程。

构造型

构造型	适用于
特征	类 对应于： A特征”。
属性get	手术 对应于： A读取”属性。
属性集	手术 对应于： A 写”属性。

标记值

标签	适用于
属性名	使用构造型属性get 或属性集进行操作 对应于：该属性后面的变量的名称。
最后	PHP 5 中的操作 对应于：“final”关键字。

公共约定

- 未指定的类型被建模为 var
- 通过将返回类型设置为 var* 来生成返回引用的方法
- 参考参数由参数 Kind 设置为 inout 或 out 的参数生成

PHP 5 约定

- final类修饰符对应于 Is Leaf属性
- abstract类修饰符对应于 Abstract属性
- 通过设置参数的类型来支持参数类型提示
- 参数的 Kind属性的 inout 或 out 的值对应一个引用参数

Python 约定

Enterprise Architect支持使用这些约定的 Python 的round工程。

标记值

标签	适用于
异步	手术 对应于：函数定义中的 “async”关键字。
装饰器	类, 操作 对应于：在源中应用到这个元素的装饰器。

其它

- 具有 Private Scope 的模型成员对应于具有两个前导下划线的代码成员
- 属性仅在初始值不为空时生成
- 所有类型都被反向工程为 var

SystemC 约定

Enterprise Architect支持使用这些约定的 SystemC 的往返工程。

构造型

构造型	适用于
代表	方法 对应A代表。
枚举	内在类 对应于：枚举类型。
朋友	方法 对应于A朋友方法。
属性	方法 对应于： A属性定义。
sc_ctor	方法 对应于： A SystemC 构造函数。
sc_module	类 对应于： A SystemC 模块。
sc_port	属性 对应： A端口。
sc_signal	属性 对应： A信号。
结构	内在类 对应于A结构或联合。

标记值

标签	适用于
种类	属性（端口） 对应于：端口种类（端口、fifo master、slave、resolved、vector）。
模式	属性（端口） 对应于：端口模式（in、out、inout）。

覆盖	方法 对应于：方法声明的继承列表。
扔	方法 对应于：方法的异常规范。

其它

- SystemC也继承了C++的大部分刻板印象和标记值

SystemC工具箱Pages

对于 SystemC 设计，将这些图标从模型工具箱的 图表Constructs"页面拖到图表上。

页	图标
系统C	模块 行动：定义一个 SystemC 模块。 一个 sc_module -stereotyped类元素。
SystemC特征	端口 行动：定义一个 SystemC端口。 一个 sc_port 原型属性。

访问

功能区	设计> 工具箱图表 查找工具箱"对话框中指定 "SystemC Constructs"
键盘快捷键	Ctrl+Shift+3 : > 在 查找工具箱项目"对话框中指定 "SystemC Constructs"
其它	您可以通过单击 图表工具箱显示或隐藏图形图表视图。

VB.NET 约定

Enterprise Architect支持使用这些约定的 Visual Basic.NET 的往返工程。支持早期版本的 Visual Basic 作为不同的语言。

构造型

构造型	适用于
事件	手术 对应于：事件声明。
进口	手术 对应于：要从另一个库导入的操作。
模块	类 对应于： A模块。
操作员	手术 对应于：运算符重载定义。
部分的	手术 对应于：操作上的 部分"关键字。
属性	手术 对应于：可能包含读写代码A属性。

标记值

标签	适用于
别名	使用构造型导入操作 对应于：此导入操作的别名。
属性名	使用构造型属性操作 对应于：该属性后面的变量的名称。
字符集	使用构造型导入操作 对应于：此导入的字符集子句 - 值 Ansi"、 统一码"或 自动"之一。
代表	手术 对应于：'delegate' 关键字。
枚举标签	使用构造型属性操作

	对应于：此属性表示为的数据类型。
把手	手术 对应于：此操作的 "handles"子句。
工具	手术 对应于：此操作的 "实现"子句。
库	使用构造型导入操作 对应于：此导入来自的库。
必须覆盖	手术 对应于："MustOverride"关键字。
收窄	使用构造型运算符进行操作 对应于："缩"关键字。
不可覆盖	手术 对应于："NotOverrideable"关键字。
重载	手术 对应于：'overloads' 关键字。
覆盖	手术 对应于："覆盖"关键字。
参数数组	参数 对应于：使用 "ParamArray"关键字A参数列表。
部分的	类,接口 对应于："部分"关键字。
只读	使用构造型属性操作 对应于：此属性仅定义 "读取"代码。
阴影	类，接口，操作 对应于："阴影"关键字。
共享	属性 对应于："共享"关键字。
加宽	使用构造型运算符进行操作 对应于："扩大"关键字。
只写	使用构造型属性操作 对应于：此属性仅定义 "写入"代码。

其它

- 为命名空间根下的每个包生成命名空间
- 类的 Is Leaf属性对应类关键字
- A类的Abstract属性对应MustInherit关键字
- 属性或操作的静态属性对应于 Shared 关键字
- 操作的 Abstract属性对应于 MustOverride 关键字
- 参数的 Kind属性的 in 值对应于 ByVal 关键字
- 参数的 Kind属性的 inout 或 out 的值对应 ByRef 关键字

Verilog 约定

Enterprise Architect支持使用这些约定的 Verilog 往返工程。

构造型

构造型	适用于
异步	方法 对应于A并发进程。
枚举	内在类 对应于：枚举类型。
初始化器	方法 对应于：一个初始化进程。
模块	类 对应于： A模块。
部分	属性 对应于A组件实例化。
港口	属性 对应： A端口。
同步	方法 对应于A顺序过程。

标记值

标签	适用于
种类	属性（信号） 对应：信号种类（如寄存器、总线）。
模式	属性（端口） 对应于：端口模式（in、out、inout）。
端口图	属性（部分） 对应于：实例化组件的通用/端口映射。
灵敏度	方法 对应于：顺序过程的敏感度列表。

类型	属性 对应于：属性的范围或类型值。

Verilog工具箱Pages

访问: '设计>图表>工具箱: '汉堡包' icon > HDL | Verilog 构造'
将这些图标拖到图表上以模型Verilog 设计。

物品	行动
模块	定义一个 Verilog 模块。 A模块定型的类元素。
枚举	定义一个枚举类型。枚举元素。
端口	定义一个 Verilog端口。 A原型属性。
部件	定义 Verilog 组件实例化。部分刻板印象A属性。
属性	定义一个属性。
过程	定义一个 Verilog 进程： • Concurrent - 一种异步定型方法 • Sequential - A同步定型方法 • 初始化器 - 初始化器刻板的方法

VHDL 约定

Enterprise Architect支持使用这些约定的 VHDL 的往返工程。

构造型

构造型	适用于
架构	类 对应于：架构。
异步	方法 对应于：一个异步进程。
配置	方法 对应于A配置。
枚举	内在类 对应于：枚举类型。
实体	接口 对应于：一个实体。
部分	属性 对应于A组件实例化。
港口	属性 对应： A端口。
信号	属性 对应于A信号声明。
结构	内在类 对应于A记录定义。
同步	方法 对应于A同步过程。
类型定义	内在类 对应于A类型或子类型定义。

标记值

标签	适用于

是通用的	属性（端口） 对应于：通用接口中的 端口”声明。
isSubType	内部类（typedef） 对应于： A类型定义。
种类	属性（信号） 对应于：信号种类（如 寄存器”、 总线”）。
模式	属性（端口） 对应：端口输入模式（'in'、'out'、'out'、'buffer'、'linkage'）。
端口映射	属性（部分） 对应于：实例化组件的通用/端口映射。
灵敏度	方法（同步） 对应于：同步进程的 敏感度”列表。
类型	内部类（typedef） 对应于： 类型”声 的 类型”指示。
类型名称空间	属性（部分） 对应于：实例化组件的 类型”命名空间。

VHDL工具箱Pages

访问

模型VHDL 设计，从 VHDL 工具箱页面拖动图标并将它们拖放到您的图表上。

功能区	设计> 工具箱图表 查找工具箱”对话框中指定 VHDL Constructs”
键盘快捷键	Ctrl+Shift+3 : > 在 查找工具箱项”对话框中指定 VHDL Constructs”
其它	您可以通过单击 图表工具箱显示或隐藏图形图表视图。

VHDL工具箱页面

物品	行动

架构	定义与 VHDL 实体关联的架构。 一种架构刻板印象的类元素。
实体	定义一个 VHDL 实体以包含端口定义。 实体原型接口元素。
枚举	定义一个枚举类型。 一个枚举元素。
结构	定义 VHDL 记录。 A结构刻板的类元素。
类型定义	定义 VHDL 类型或子类型。 A typedef-stereotyped类元素。

VHDL特征工具箱页面

物品	行动
部件	定义 VHDL 组件实例化。 部分刻板印象A属性。
端口	定义一个 VHDL端口。 A原型属性。
信号	定义 VHDL 信号。 信号定型属性A
过程	定义一个 VHDL 过程： • Concurrent - 一种异步定型方法 • Sequential - A同步定型方法 • 配置- 一种配置定型方法

Visual Basic 约定

Enterprise Architect支持使用这些约定的 Visual Basic 5 和 6 的round工程。
支持 Visual Basic .NET作为不同的语言。

构造型

构造型	适用于
全球的	属性 对应于： 全局"关键字。
进口	手术 对应于：要从另一个库导入的操作。
属性get	手术 对应于： A属性'get'。
属性集	手术 对应于： A属性'set'。
属性	手术 对应于： A属性'let'。
有事件	属性 对应于："WithEvents"关键字。

标记值

标签	适用于
别名	使用构造型导入操作 对应于：此导入操作的别名。
属性名	使用构造型属性get、属性或属性进行操作 对应于：该属性后面的变量的名称。
库	使用构造型导入操作 对应于：此导入来自的库。
新的	属性 对应于： 新"关键字。

其它

- 参数的 Kind属性的 in 值对应于 ByVal 关键字
- 参数的 Kind属性的 inout 或 out 的值对应 ByRef 关键字

语言选项

您可以为Enterprise Architect在生成和逆向工程代码时如何处理特定语言设置各种选项。这些选项要么特定于：

- 您的用户 ID，适用于所有型号或
- 为所有用户定义它们的模型

访问

功能区	开始>外观> Preferences > Preferences >源代码工程> <语言名> 设置>模型>选项>源代码工程><语言名称>
键盘快捷键	Ctrl+F9（ 首选项”对话框）

支持的语言

语
行动脚本
Ada 2012（在Enterprise Architect的统一和终极版中）
ArcGIS
ANSI C
C #
C++
德尔福
Java
PHP
Python
系统C
Verilog（统一和终极版）
VHDL（统一和终极版）
视觉基础

Visual Basic .NET

ActionScript 选项 -用户

如果您打算从您的模型生成 ActionScript 代码，您可以使用 "Preferences" 对话框的 "ActionScript Specifications" 页面配置代码生成选项，以：

- 指定默认源目录
- 指定 ActionScript 代码的编辑器

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > ActionScript
键盘快捷键	Ctrl+F9

选项

选项	行动
禁用语言	不选中此复选框以支持 ActionScript 代码生成。 选中此复选框可禁用 ActionScript 代码支持。
当前用户的选项	在 "默认源目录" 和 "编辑器" 字段中，单击 ... 按钮并浏览您将使用的源目录和外部文件编辑器。

注记

- 这些选项适用于您访问的所有模型

ActionScript 选项 -模型

如果您打算从您的模型生成 ActionScript 代码，您可以使用 管理模型选项”对话框的 ‘ActionScript 规范”页面配置特定于模型的代码生成选项，以：

- 指定要生成的默认 ActionScript 版本（AS2.0 或 AS3.0）
- 指定默认文件扩展名
- 指定关联连接器的集合类定义

访问

功能区	设置 >模型> 选项 >源代码工程 > ActionScript

选项

选项	行动
当前模型的选项	类型在生成 ActionScript源代码时应用的默认 ActionScript 版本和默认文件扩展名。
集合类	单击此按钮打开 关联角色的集合类”对话框，您可以通过该对话框指定关联连接器的集合类定义。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型

Ada 2012 选项-用户

如果您打算从您的模型生成 Ada 2012 代码，您可以使用 首选项"对话框的 Ada"页面配置代码生成选项，以：

- 通知逆向工程过程 Tagged Record 的名称是否与包名相同
- 建议替代标记记录名称的引擎找到
- 指定引擎是否应该为标记记录创建引用类型（如果未定义）
- 提供要创建的引用类型的名称（默认为 Ref）
- 指定参考/访问类型的参考参数
- 告诉引擎忽略引用参数的名称
- 指示要定位的参考参数的名称

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > Ada
键盘快捷键	Ctrl+F9

选项

选项	行动
禁用语言	不选中此复选框以支持 Ada 2012 代码生成。 选中此复选框可禁用 Ada 2012 代码支持。
当前用户的选项	指定用于当前用户的选项；这些选项适用于用户访问的所有模型。

注记

- Enterprise Architect的统一版和终极版中提供了 Ada 2012 支持

Ada 2012 选项-模型

如果您打算从您的模型生成 Ada 2012 代码，您可以使用 管理模型选项"对话框的 Ada"页面配置特定 模型的代码生成选项，以：

- 指定默认文件扩展名和
- 指定关联连接器的集合类定义

访问

功能区	设置 >模型> 选项 >源代码工程 > Ada

选项

选项	行动
当前模型的选项	生成 Ada源代码时应用的默认文件扩展名中的类型。
集合类	单击此按钮打开 关联角色的集合类"对话框，您可以通过该对话框指定关联连接器的集合类定义。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型
- Enterprise Architect的统一版和终极版中提供了 Ada 2012 支持

ArcGIS 选项 -用户

如果您打算从您的模型生成 ArcGIS 代码，您可以使用 "首选项"对话框的 "ArcGIS"页面配置代码生成选项，以：

- 指定默认源目录
- 指定 ArcGIS 代码的编辑器

必须在 "MDG 技术"对话框（ 特定>技术>管理技术"）中启用 ArcGIS，才能使用 "ArcGIS"页面。

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > ArcGIS
键盘快捷键	Ctrl+F9

选项

选项	行动
禁用语言	取消选中此复选框以支持 ArcGIS 代码生成。 选中此复选框可禁用 ArcGIS 代码支持。
当前用户的选项	指定用于当前用户的选项；这些选项适用于用户访问的所有模型。

ArcGIS 选项 -模型

如果您打算从您的模型生成 ArcGIS 代码，您可以使用 管理模型选项"对话框的 ArcGIS"页面配置特定 模型的代码生成选项，以：

- 指定默认文件扩展名
- 指定关联连接器的集合类定义

访问

功能区	设置 >模型> 选项 >源代码工程 > ArcGIS

选项

选项	行动
当前模型的选项	生成 ArcGIS源代码时应用的默认文件扩展名中的类型。
集合类	单击此按钮打开 关联角色的集合类"对话框，您可以通过该对话框指定关联连接器的集合类定义。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型

C 选项 -用户

如果您打算从您的模型生成 C 代码，您可以使用 "Preferences" 对话框的 "C Specifications" 页面配置代码生成选项。

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > C
键盘快捷键	Ctrl+F9

选项

选项	行动
禁用语言	不选中此复选框以支持 C 代码生成。 选择此选项可禁用 C 代码支持。
当前用户的选项	在值字段中，指定在您访问的所有模型中应用在您自己的用户 ID 下的选项： ● 要创建的默认属性类型（固定为int ） ● 是否将#define 常量作为导入的 C 代码中的属性导入（如果在 管理模型选项"对话框的 "C 规范"页面上将 物件导向编程"设置为True ） ● 是否为声明的 C 方法生成注释，以及从声明中逆向工程注释 ● 是否为实现的 C 方法生成注释，以及从实现中逆向工程注释 ● 从模型重新生成代码时是否更新注释 ● 从模型重新生成代码时是否更新实现文件 ● 默认源代码目录位置（点击 ... 按钮） ● 导入 C 代码目录时要读取的默认文件扩展名 ● 要使用的代码编辑器（单击 ... 按钮） ● 实现文件相对于头文件路径的搜索路径

C选项-模型

如果您打算从您的模型生成 C 代码，您可以使用 管理模型选项”对话框的 C 规范”页面配置特定于模型的代码生成选项，以：

- 指定默认文件扩展名（头和源）
- 定义对面向物件编程的支持
- 设置状态机工程选项
- 指定关联连接器的集合类定义

访问

功能区	设置>模型>选项>源代码工程> C

选项

选项	行动
当前模型的选项	在值字段中，指定以下选项： ● 代码文件的默认头文件和源文件扩展名 ● 支持面向物件的编程；如果这是True ，则设置： -命名空间分隔符 - 操作的第一个参数是否为类引用 - 生成的 C 代码中的参数引用样式 - 生成代码中的引用参数名称 - 生成代码中的默认构造函数名称 - 生成代码中的默认析构函数名称
状态机工程	在值字段中，使用下拉箭头将选项设置为True或False ；这些选项仅适用于从当前模型中的状态机模型生成代码： ● '使用新状态机模板' - 设置为True以使用Enterprise Architect Release 11 及更高版本中的代码生成模板，设置为False以应用 EASL Legacy 模板 ● 生成跟踪代码 - 设置为True生成跟踪代码，设置为False省略
集合类	单击此按钮打开 关联角色的集合类”对话框，您可以通过该对话框指定关联连接器的集合类定义。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型

C# 选项 -用户

如果您打算从您的模型生成 C# 代码，您可以使用 "Preferences" 对话框的 "C# Specifications" 页面配置代码生成选项

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > C#
键盘快捷键	Ctrl+F9

选项

选项	行动
禁用语言	不选中此复选框以支持 C# 代码生成。 选中此复选框可禁用 C# 代码支持。
当前用户的选项	在值字段中，指定在您访问的所有模型中应用在您自己的用户 ID 下的选项： ● 要创建的默认属性类型 ● 生成 C# 类时是否应生成命名空间 ● 导入 XML.NET 样式注释时是否从摘要标记中删除新行（硬回车） ● 为C#类生成代码时是否生成Finalizer方法 ● 为C#类生成代码时是否生成Dispose方法 ● 默认源代码目录位置（点击 ... 按钮） ● 要使用的代码编辑器（单击 ... 按钮）

C# 选项 -模型

如果您打算从您的模型生成 C# 代码，您可以使用 "模型"对话框的 "C# Specifications"页面配置特定 模型的代码生成选项，以：

- 指定默认文件扩展名
- 指示额外的集合类 - 定义自定义集合类，可以是简单的替换（例如 CArray<#TYPE#>）或其他字符串和替换的混合（例如 Cmap<CString,LPCTSTR,#TYPE#*,#TYPE #*>);这些集合类是默认定义的：
  - 列表<#TYPE#>;堆栈<#TYPE#>;队列<#TYPE#>;
- 设置状态机工程选项
- 指定关联连接器的集合类定义

访问

功能区	设置>模型>选项>源代码工程> C#

选项

选项	行动
当前模型的选项	类型在生成 C#源代码时应用的默认文件扩展名中，以及您要定义的任何其他集合类的列表。
状态机工程	在值字段中，使用下拉箭头将选项设置为True或False ；这些选项仅适用于从当前模型中的状态机模型生成代码： ● '使用新状态机模板' - 设置为True以使用Enterprise Architect Release 11 及更高版本中的代码生成模板，设置为False以应用 EASL Legacy 模板 ● '生成跟踪代码' - 设置为True生成跟踪代码，设置为False省略
集合类	单击此按钮打开 "关联角色的集合类"对话框，您可以通过该对话框指定关联连接器的集合类定义。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型

C++ 选项 -用户

如果您打算从您的模型生成 C++ 代码，您可以使用 首选项”对话框的 C++ 规范”页面配置代码生成选项。

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > C++
键盘快捷键	Ctrl+F9

选项

选项	行动
禁用语言	不选中此复选框以支持 C++ 代码生成。 选择此选项可禁用 C++ 代码支持。
当前用户的选项	在值字段中，指定在您访问的所有模型中应用在您自己的用户 ID 下的选项： ● 要创建的默认属性类型 ● 生成 C++ Classes 时是否应该生成 Namespaces ● 为 C++ 生成和处理注释时应用什么样式 ● 是否为声明的 C++ 方法生成注释，或从声明中逆向工程注释 ● 是否为实现的 C++ 方法生成注释，或从实现中逆向工程注释 ● 从模型重新生成代码时是否更新注释 ● 从模型重新生成代码时是否更新实现文件 ● 默认源代码目录位置（点击 ... 按钮） ● 导入 C++ 代码目录时要读取的默认文件扩展名 ● 要使用的代码编辑器（单击 ... 按钮） ● 实现文件相对于头文件路径的搜索路径

C++ 选项-模型

如果您打算从您的模型生成 C++ 代码，您可以使用 "管理模型选项"对话框的 "C++ 规范"页面配置特定于模型的代码生成选项，以：

- 指示要生成的 C++ 版本；这控制使用的模板集以及如何创建属性
- 指定通过引用指定类型时使用的默认引用类型
- 指定默认文件扩展名
- 指定默认的 Get/Set 前缀
- 指定关联连接器的集合类定义
- 定义额外的集合类 - 定义自定义集合类，可以是简单的替换（如 CArray<#TYPE#>）或其他字符串和替换的混合（如 Cmap<CString,LPCTSTR,#TYPE#*,#TYPE #*>);这些集合类是默认定义的：
  - CArray<#TYPE#>;CMap<CString,LPCTSTR,#TYPE#*,#TYPE#*>;
- 设置状态机工程选项

访问

功能区	设置>模型>选项>源代码工程> C++

选项

选项	行动
当前模型的选项	在值字段中，指定影响当前模型的所有用户的选项： ● 您正在使用的 C++ 版本（它决定了生成代码时要使用的模板） ● 通过引用为 C++ 属性创建属性时使用的默认引用类型 ● 代码文件的默认头文件和源文件扩展名 ● 默认的 "获取"前缀 ● 默认的 "设置"前缀 ● 额外的集合类
状态机工程选项	在值字段中，使用下拉箭头将选项设置为True或False ；这些选项仅适用于从当前模型中的状态机模型生成代码： ● '使用新状态机模板' - 设置为True以使用Enterprise Architect Release 11 及更高版本中的代码生成模板，设置为False以应用 EASL Legacy 模板 ● '生成跟踪代码' - 设置为True生成跟踪代码，设置为False省略
集合类	单击此按钮打开 "关联角色的集合类"对话框，您可以通过该对话框指定关联连接器的集合类定义。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型

Delphi Options -用户

如果您打算从您的模型生成 Delphi 代码，您可以使用 "Preferences"对话框的 "Delphi Specifications"页面配置代码生成选项，以：

- 设置默认属性类型
- 指明一个默认的源目录
- 设置用于编辑 Delphi源代码的默认代码编辑器

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > Delphi
键盘快捷键	Ctrl+F9

选项

选项	行动
禁用语言	不选中此复选框以支持 Delphi 代码生成。 选择此选项可禁用 Delphi 代码支持。
当前用户的选项	指定用于当前用户的选项；这些选项适用于用户访问的所有模型。

Delphi Options -模型

如果您打算从您的模型生成 Delphi 代码，您可以使用 "模型"对话框的 "Delphi Specifications"页面配置特定 模型的代码生成选项，以：

- 指定默认文件扩展名（头和源）
- 指定关联连接器的集合类定义

访问

功能区	设置>模型>选项>源代码工程> Delphi

选项

选项	行动
当前模型的选项	类型在生成 Delphi源代码时应用的默认文件扩展名。
集合类	单击此按钮打开 "关联角色的集合类"对话框，您可以通过该对话框指定关联连接器的集合类定义。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型

德尔福属性

Enterprise Architect对 Delphi属性有全面的支持。这些是作为标记值实现的，具有专门的属性编辑器来帮助创建和修改类属性。通过使用 特征可见性"元素上下文显示菜单选项，您可以显示包含属性的 标签"隔间。为方便起见，带有属性的导入的 Delphi 类会自动显示此特征。

手动激活属性编辑器

- 在选定的类中，将代码生成语言设置为 Delphi"
- 右键单击类并选择 Delphi属性"打开编辑器

使用Delphi属性编辑器，可以快速简单地构建属性；从这里您可以：

- 更改名称和范围（目前仅支持 Public 和 Published）
- 更改属性类型（下拉列表包括项目中所有定义的类）
- 设置读写信息（下拉列表有当前类的所有属性和操作，也可以输入自由文本）
- 将 已存储"设置为True或False
- 设置工具信息
- 设置默认值，如果存在

注记

- 当您使用 属性"屏幕中的 创建属性"对话框时，系统会生成一对属性Get 和 Set 函数以及所需的定义作为标记值；如果需要，您可以手动编辑这些标记值
- 公共属性以 +"符号前缀显示，并以 "发布
- 在 创建属性实现"对话框（通过 属性"对话框访问）中创建属性时，如果属性类型为 Delphi，您可以将范围设置为 已发布"
- 仅支持 公开"和 已发布"
- 如果您更改了某个属性和正向工程师的名称，则会添加一个新属性，但您必须手动从源文件中删除旧属性

Java选项 -用户

如果您打算从您的模型生成Java代码，您可以使用 首选项”对话框的 “Java规范”页面配置代码生成选项。

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > Java
键盘快捷键	Ctrl+F9

选项

选项	行动
禁用语言	不选中此复选框以支持Java代码生成。 选中此复选框可禁用Java代码支持。
当前用户的选项	在值字段中，指定在您访问的所有模型中应用在您自己的用户 ID 下的选项；这： ● 要创建的默认属性类型（从下拉列表中选择） ● 默认源代码目录位置（点击 ... 按钮） ● 要使用的代码编辑器（单击 ... 按钮）

Java选项 -模型

如果您打算从您的模型生成Java代码，您可以使用 管理模型选项"对话框的 "Java规范"页面配置特定于模型的代码生成选项，以：

- 指定默认文件扩展名
- 指定默认的 获取"前缀
- 指定默认的 设置"前缀
- 设置状态机工程选项
- 指定关联连接器的集合类定义
- 定义额外的集合类 - 定义自定义集合类，可以是简单的替换（如 CArray<#TYPE#>）或其他字符串和替换的混合（如 Cmap<CString,LPCTSTR,#TYPE#*,#TYPE #*>);这些集合类是默认定义的：
  - HashSet<#TYPE#>;Map<字符串,#TYPE#>;

访问

功能区	设置>模型>选项>源代码工程> Java

选项

选项	行动
当前模型的选项	在值字段中，指定影响当前模型的所有用户的选项；这： ● 代码文件的默认文件扩展名 ● 默认的 Get 和 Set 前缀 ● 默认和附加的集合类
状态机工程	在值字段中，使用下拉箭头将选项设置为True或False ；这些选项仅适用于从当前模型中的状态机模型生成代码： ● '使用新状态机模板' - 设置为True以使用Enterprise Architect Release 11 及更高版本中的代码生成模板，设置为False以应用 EASL Legacy 模板 ● '生成跟踪代码' - 设置为True生成跟踪代码，设置为False省略
集合类	单击此按钮打开 关联角色的集合类"对话框，您可以通过该对话框指定关联连接器的集合类定义。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型

MySQL选项 -用户

如果您打算从您的模型生成MySQL代码，您可以使用 首选项”对话框的 “MySQL ”页面配置代码生成选项，以：

- 指定默认属性类型
- 指定默认源目录
- 指定要导入的文件的文件扩展名
- 指定用于更改代码的编辑器
- 指定默认所有者

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > MySQL
键盘快捷键	Ctrl+F9

选项

选项	行动
禁用语言	不选中此复选框以支持MySQL代码生成。 选择此选项可禁用MySQL代码支持。
当前用户的选项	指定用于当前用户的选项；这些选项适用于用户访问的所有模型。

MySQL选项 -模型

如果您打算从您的模型生成MySQL代码，您可以使用 管理模型选项"对话框的 "MySQL "页面配置特定 模型的代码生成选项，以：

- 指定默认文件扩展名
- 指定关联连接器的集合类定义

访问

功能区	设置>模型>选项>源代码工程> MySQL

选项

选项	行动
当前模型的选项	类型在生成MySQL源代码时应用的默认文件扩展名。
集合类	单击此按钮打开 关联角色的集合类"对话框，您可以通过该对话框指定关联连接器的集合类定义。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型

PHP 选项 -用户

如果您打算从您的模型生成 PHP 代码，您可以使用 首选项”对话框的 PHP 规范”页面配置代码生成选项，以：

- 定义一个以分号分隔的扩展列表，以便在为 PHP 进行目录代码导入时查看
- 设置打开和保存 PHP源代码的默认目录
- 指定编辑 PHP 代码时使用的默认编辑器

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > PHP
键盘快捷键	Ctrl+F9 |源代码工程 | PHP

选项

选项	行动
禁用语言	不选中此复选框以支持 PHP 代码生成。 选择此选项可禁用 PHP 代码支持。
当前用户的选项	指定用于当前用户的选项；这些选项适用于用户访问的所有模型。

PHP 选项-模型

如果您打算从您的模型生成 PHP 代码，您可以使用 管理模型选项”对话框的 PHP 规范”页面配置特定于模型的代码生成选项，以：

- 指定要生成的默认 PHP 版本
- 定义默认文件扩展名
- 指定默认的 获取”前缀
- 指定默认的 设置”前缀

访问

功能区	设置>模型>选项>源代码工程> PHP

选项

选项	行动
当前模型的选项	默认 PHP 版本中的类型，生成 PHP源代码时应用的默认文件扩展名，以及默认的 'Get' 和 'Set' 前缀。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型

Python 选项 -用户

如果您打算从您的模型生成 Python 代码，您可以使用 首选项”对话框的 Python 规范”页面配置代码生成选项，以：

- 指定要使用的默认源目录
- 指定用于编写和编辑 Python 代码的默认编辑器

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > Python
键盘快捷键	Ctrl+F9

选项

选项	行动
禁用语言	不选中此复选框以支持 Python 代码生成。 选择此选项可禁用 Python 代码支持。
当前用户的选项	指定用于当前用户的选项；这些选项适用于用户访问的所有模型。

Python 选项 -模型

如果您打算从您的模型生成 Python 代码，您可以使用 管理模型选项"对话框的 Python 规范"页面配置特定于模型的代码生成选项，以：

- 指定默认文件扩展名

访问

功能区	设置>模型>选项>源代码工程> Python

选项

选项	行动
当前模型的选项	类型在生成 Python源代码时应用的默认文件扩展名。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型

SystemC 选项 -用户

如果您打算从您的模型生成 SystemC 代码，您可以使用 "首选项" 对话框的 "SystemC" 页面配置代码生成选项，以：

- 指定默认源目录
- 指定用于更改代码的编辑器

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > SystemC
键盘快捷键	Ctrl+F9

选项

选项	行动
禁用语言	不选中此复选框以支持 SystemC 代码生成。 选择此选项可禁用 SystemC 代码支持。
当前用户的选项	指定用于当前用户的选项；这些选项适用于用户访问的所有模型。

SystemC 选项-模型

如果您打算从您的模型生成 SystemC 代码，您可以使用 模型"对话框的 SystemC"页面配置特定 模型的代码生成选项，以：

- 指定默认文件扩展名
- 指定关联连接器的集合类定义

访问

功能区	设置>模型>选项>源代码工程> SystemC

选项

选项	行动
当前模型的选项	生成 SystemC源代码时应用的默认文件扩展名中的类型。
集合类	单击此按钮打开 关联角色的集合类"对话框，您可以通过该对话框指定关联连接器的集合类定义。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型

Teradata 选项 -用户

如果您打算从您的模型生成 Teradata 代码，您可以使用 "首选项"对话框的 "Teradata"页面配置代码生成选项，以：

- 指定默认属性类型
- 指定默认源目录
- 指定用于更改代码的编辑器

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > Teradata
键盘快捷键	Ctrl+F9

选项

选项	行动
禁用语言	不选中此复选框以支持 Teradata 代码生成。 选择此选项可禁用 Teradata 代码支持。
当前用户的选项	指定用于当前用户的选项；这些选项适用于用户访问的所有模型。

Teradata Options -模型

如果您打算从您的模型生成 Teradata 代码，您可以使用 管理模型选项"对话框的 Teradata"页面配置特定 模型的代码生成选项，以：

- 指定默认文件扩展名
- 指定关联连接器的集合类定义

访问

功能区	设置 >模型> 选项 >源代码工程 > Teradata

选项

选项	行动
当前模型的选项	生成 Teradata源代码时应用的默认文件扩展名中的类型。
集合类	单击此按钮打开 关联角色的集合类"对话框，您可以通过该对话框指定关联连接器的集合类定义。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型

VB.NET 选项 -用户

如果您打算从您的模型生成 VB.NET 代码，您可以使用 "首选项"对话框的 "VB.NET 规范"页面配置代码生成选项，以：

- 指定默认属性类型
- 指示是否生成命名空间
- 指定默认源目录
- 指定用于更改代码的编辑器

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > VB.Net
键盘快捷键	Ctrl+F9

选项

选项	行动
禁用语言	不选中此复选框以支持 VB.NET 代码生成。 选择此选项可禁用 VB.NET 代码支持。
当前用户的选项	指定用于当前用户的选项；这些选项适用于用户访问的所有模型。

VB.NET 选项-模型

如果您打算从您的模型生成 VB.NET 代码，您可以使用 管理模型选项”对话框的 “VB.Net 规范”页面配置特定于模型的代码生成选项，以：

- 指定默认文件扩展名
- 指定关联连接器的集合类定义

访问

功能区	设置>模型>选项>源代码工程> VB.Net

选项

选项	行动
当前模型的选项	生成 VB.Net源代码时应用的默认文件扩展名中的类型。
集合类	单击此按钮打开 关联角色的集合类”对话框，您可以通过该对话框指定关联连接器的集合类定义。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型

Verilog 选项 -用户

如果您打算从您的模型生成 Verilog 代码，您可以使用 "首选项" 对话框的 "Verilog" 页面配置代码生成选项，以：

- 指定默认源目录
- 指定用于更改代码的编辑器

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > Verilog
键盘快捷键	Ctrl+F9

选项

选项	行动
禁用语言	不选中此复选框以支持 Verilog 代码生成。 选择此选项可禁用 Verilog 代码支持。
当前用户的选项	指定用于当前用户的选项；这些选项适用于用户访问的所有模型。

Verilog 选项-模型

如果您打算从您的模型生成 Verilog 代码，您可以使用 管理模型选项”对话框的 “Verilog”页面配置特定 模型的代码生成选项，以：

- 指定默认文件扩展名
- 指定关联连接器的集合类定义

访问

功能区	设置>模型>选项>源代码工程> Verilog

选项

选项	行动
当前模型的选项	类型在生成 Verilog源代码时应用的默认文件扩展名。
集合类	单击此按钮打开 关联角色的集合类”对话框，您可以通过该对话框指定关联连接器的集合类定义。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型

VHDL 选项 -用户

如果您打算从您的模型生成 VHDL 代码，您可以使用 "Preferences"对话框的 "VHDL"页面配置代码生成选项，以：

- 指定默认源目录
- 指定用于更改代码的编辑器

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > VHDL
键盘快捷键	Ctrl+F9

选项

选项	行动
禁用语言	不选中此复选框以支持 VHDL 代码生成。 选择此选项可禁用 VHDL 代码支持。
当前用户的选项	指定用于当前用户的选项；这些选项适用于用户访问的所有模型。

VHDL 选项 -模型

如果您打算从您的模型生成 VHDL 代码，您可以使用 "模型"对话框的 "VHDL"页面配置特定 模型的代码生成选项，以：

- 指定默认文件扩展名
- 指定关联连接器的集合类定义

访问

功能区	设置>模型>选项>源代码工程> VHDL

选项

选项	行动
当前模型的选项	类型在生成 VHDL源代码时应用的默认文件扩展名。
集合类	单击此按钮打开 "关联角色的集合类"对话框，您可以通过该对话框指定关联连接器的集合类定义。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型

Visual Basic 选项 -用户

如果您打算从您的模型生成 Visual Basic 代码，您可以使用 "首选项"对话框的 "VB 规范"页面配置代码生成选项，以：

- 指定默认属性类型
- 定义默认源目录
- 定义文件扩展名以搜索要导入的代码文件
- 定义用于编辑源代码的默认编辑器

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > Visual Basic
键盘快捷键	Ctrl+F9

选项

选项	行动
禁用语言	不选中此复选框以支持 Visual Basic 代码生成。 选择此选项可禁用 Visual Basic 代码支持。
当前用户的选项	指定当前使用的选项；这些选项适用于用户访问的所有模型。

Visual Basic 选项 -模型

如果您打算从您的模型生成 Visual Basic 代码，您可以使用 管理模型选项”对话框的 ‘VB 规范”页面配置特定于模型的代码生成选项，以：

- 指定要生成的默认 Visual Basic 版本
- 读/写时指示默认文件扩展名
- 指示 MTS 对象的 Microsoft 事务服务器(MTS) 事务模式
- 指定一个类是否使用 Multi use（ True或False ）
- 指定一个类是否使用 Persistable属性
- 指示数据绑定和数据源行为
- 设置全局命名空间
- 设置暴露属性
- 指示 Createable 属性是True还是False
- 指定关联连接器的集合类定义

访问

功能区	设置>模型>选项>源代码工程> Visual Basic

选项

选项	行动
当前模型的选项	类型在生成 Visual Basic源代码时应用的默认文件扩展名中，然后单击其他每个字段中的下拉箭头并选择适当的值。
集合类	单击此按钮打开 关联角色的集合类”对话框，您可以通过该对话框指定关联连接器的集合类定义。

注记

- 这些选项影响当前模型的所有用户；但是，它们不适用于其他模型

MDG 技术语言选项

如果您已将指定代码模块的MDG 技术加载到*Sparx Systems* > *EA* > *MDG* 技术文件夹中，则该语言将包含在 首选项"对话框的 源代码工程"列表中。如果MDG 技术文件在您的模型中实际使用该语言，则该语言仅在 首选项"对话框中列出。

访问

功能区	开始>外观> Preferences > Preferences >源代码工程 > MDG
键盘快捷键	Ctrl+F9

选项

字段	行动
默认扩展	生成的源文件的默认扩展名；显示选项是否在技术中。 这是按项目保存的。
导入文件扩展	导入源文件的默认文件夹；显示该技术是否支持命名空间。 这为所有项目保存一次。
生成命名空间	指示是否生成命名空间。
默认源目录	保存生成的源文件的默认目录。 这总是显示出来的。
编辑	表示用于编辑源文件的编辑器。
态度类型	指示默认属性类型。

注记

- 这些选项在代码模块的 <CodeOptions> 标签内的技术中设置，如图所示：
  <CodeOption name="DefaultExtension">.rb</CodeOption>

重置选项

Enterprise Architect在第一次创建类时存储了类的一些选项。有些是全球性的；例如，$LinkClass 是在您第一次创建类时存储的，因此在现有类中， 首选项"对话框中的全局更改不会自动被拾取。您必须修改现有类的选项。

修改单个类的选项

节	行动
1	单击要更改的类，然后选择 开发>源代码>生成>生成单个元素"功能区选项。 将显示 生成代码"对话框。
2	单击高级按钮。 将显示 物件选项"对话框。
3	单击 属性/操作"选项。
4	更改选项，然后单击关闭按钮以应用更改。

修改包中所有类的选项

节	行动
1	单击浏览器窗口中的包，然后选择 开发 > 首选项 > 选项 > 重置源语言"功能区选项。 将显示 管理代码生成"对话框。
2	在 语言所在："字段中，单击下拉箭头并选择要更改的语言。
3	在 转换为："字段中，单击下拉箭头并选择要更改为的语言。
4	选中每个选项对应的复选框以应用于包中更改的类元素： ● 清除文件的文件名以生成代码 ● 重置每个类的默认选项 ● 选中包下的进程子包包
5	单击确定按钮以应用更改。

设置集合类

使用Enterprise Architect ，您可以定义集合类，以便从目标角色的多重性设置大于1的关联连接器生成代码。

任务

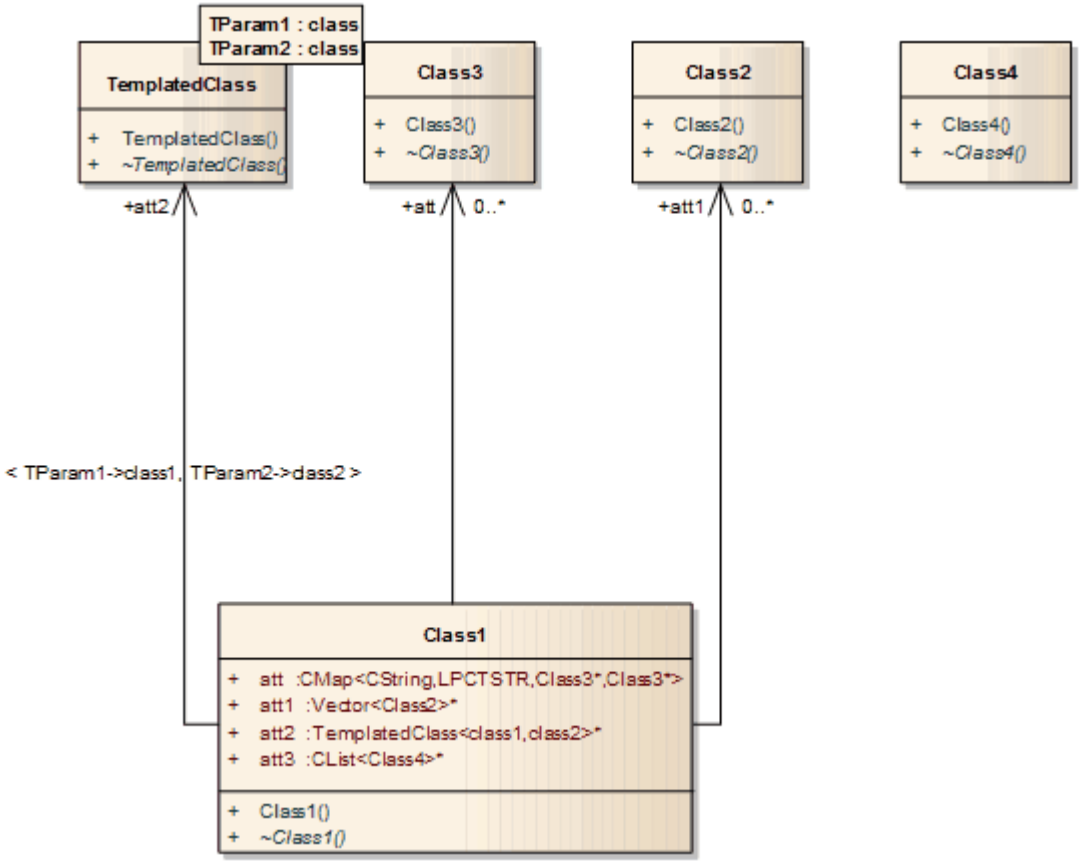
任务	细节
定义集合类	在 管理模型选项"对话框的 源代码工程"部分（选择 设置>模型>选项>源代码工程"功能区选项），在每个语言页面上单击集合类按钮。 将显示 关联角色的集合类"对话框。在此对话框中，您可以定义： ● 1 ..* 角色的默认 Collection类 ● 用于1 ..* 角色的有序集合类 ● 用于1 ..* 角色的限定 Collection类
为特定类	类特定的集合类可以通过单击元素的类的属性对话框中的集合类按钮来定义。
代码生成优先级	当Enterprise Architect为具有多重角色 > 1的连接器生成代码时： 1 .如果设置了限定符，请使用限定集合： - 如果设置了类 - else使用代码语言限定的集合 2.如果设置了'Order'选项，使用有序集合： - 如果设置了类 - else使用代码语言有序集合 3.否则使用默认集合： - 如果设置了类 - else使用代码语言默认集合
使用标记	您可以在集合名称中包含标记#TYPE#； Enterprise Architect将其替换为在源生成时收集的类的名称（例如，Vector<#TYPE#> 将变为 Vector<foo>）。 相反，在逆向工程时，如果将匹配条目（例如，如果在模型中找到 foo，则为 foo）定义为 Collection类，也会创建关联连接器。
附加集合类	可以在 C#、C++ 和Java的模型特定语言选项页面中定义其他集合类。
会员类型	在关联 属性"对话框的 角色"选项卡上（可从任何关联的右键单击上下文菜单访问），每个源和目标角色都有一个 成员类型"字段。 如果您设置此项，您输入的值将覆盖所有列出的选项。

集合类的示例使用

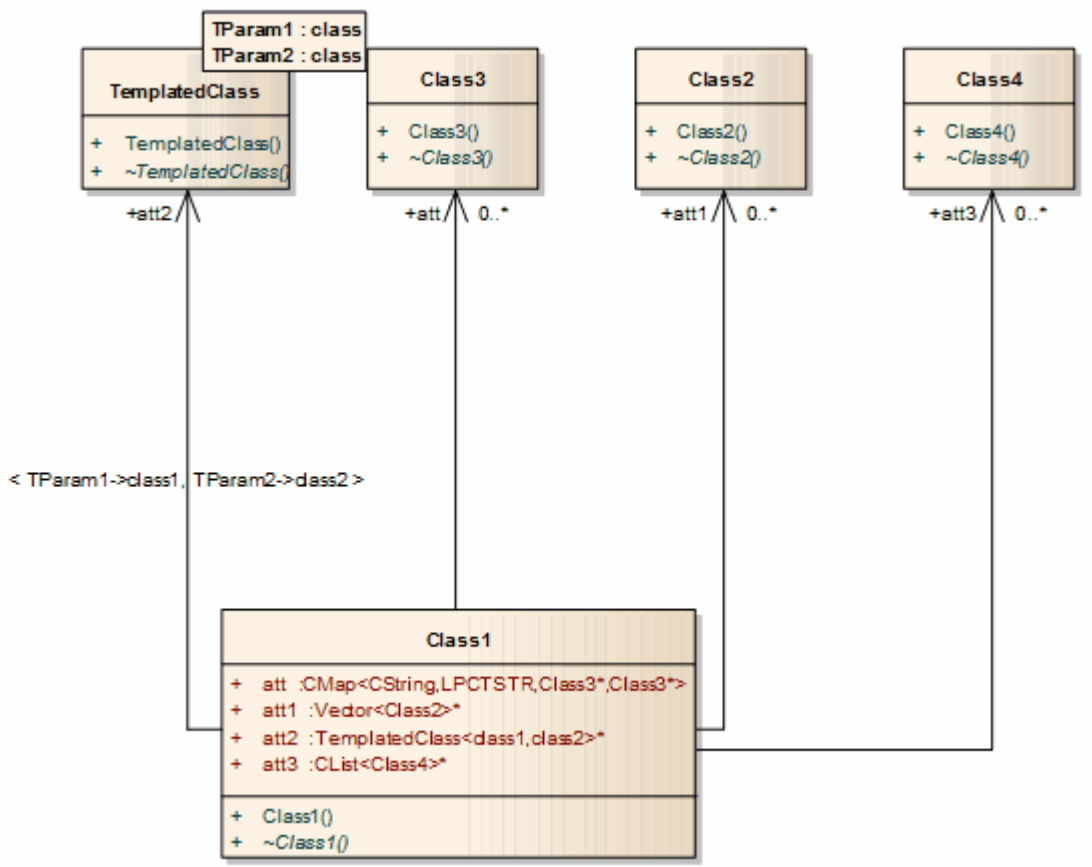
考虑这个源代码：

```
类 Class1
{
上市：
类 1();
虚拟～Class1（）；
CMap<CString,LPCTSTR,Class3*,Class3*> att;
向量<Class2> *att1;
TemplatedClass<class1,class2> *att2;
CList<Class4> *att3;
};
类 Class2
{
上市：
类 2();
虚拟～Class2（）；
};
类 Class3
{
上市：
类 3();
虚拟～Class3（）；
};
类 Class4
{
上市：
第 4 类（）；
虚拟～Class4（）；
};
模板<类 TParam1，类 TParam2>
类模板类
{
上市：
模板类（）{
}
虚拟～模板类（）{
}
};
```

如果使用默认导入选项将此代码导入系统，则会生成此图：



但是，如果您在特定于模型的语言选项页面（C#、 Java 、C++）的 Additional Collection Classes"字段中输入值 “CList<#Type#>”，则还会为类4 创建一个关联连接器：


TParam1 : class
TParam2 : class
TemplatedClass
+ TemplatedClass()
+ ~TemplatedClass()
Class3
+ Class3()
+ ~Class3()
Class2
+ Class2()
+ ~Class2()
Class4
+ Class4()
+ ~Class4()
+att2
+att
0..*
+att1
0..*
+att3
0..*
< TParam1->class1, TParam2->class2 >
Class1
+ att :CMap<CString,LPCTSTR,Class3*,Class3*>
+ att1 :Vector<Class2>*
+ att2 :TemplatedClass<class1,class2>*
+ att3 :CList<Class4>*
+ Class1()
+ ~Class1()

本地路径

当一组开发人员在同一Enterprise Architect模型上工作时，每个开发人员可能会将他们的源代码版本存储在他们的本地文件系统中，但并不总是与他们的开发人员处于相同的位置。要在Enterprise Architect中管理此场景，您可以在 "本地路径"对话框中为每个用户定义本地路径。

您可以使用本地路径生成代码和逆向工程，以及在版本控制、开发 XML 模式和生成文档和 Web 报告。

设置本地路径可能需要一些时间，但如果您想同时在源和模型上协同工作，那么努力是值得的。

例如，如果：

- 开发者A将她的 .java 文件存储在 C:\ Java \源目录中，而开发者B将他的文件存储在D :\源中，并且
- 两位开发人员都希望生成并逆向工程到位于共享（或复制）网络驱动器上的相同Enterprise Architect模型

开发人员A可能会定义以下本地路径：

JAVA_SOURCE = "C:\ Java \源"

Enterprise Architect项目中生成和存储的所有类都存储为：

%JAVA_SOURCE%\<xxx.java>

开发者B将本地路径定义为：

JAVA_SOURCE = D :\源"

现在， Enterprise Architect将所有 java 文件存储在这些目录中：

%JAVA_SOURCE%\<文件名>

在每个开发人员的机器上，文件名都扩展为正确的本地版本。

访问

功能区	开发>源代码>选项>配置本地路径

本地路径对话框

使用 "本地路径"对话框，您可以为特定机器上的单个用户设置本地路径。有关使用本地路径的说明，请参阅本地路径主题。

访问

功能区	开发>源代码>选项>配置本地路径

选项

选项	行动
小路	文件系统中本地目录的路径类型为in或browser（例如d:\java\源）。
ID	替换本地路径的共享 ID 中的类型（例如，JAVA_SRC）。
类型	单击下拉箭头并选择要应用的路径类型（例如Java ）。
相对路径	列出当前为模型定义的路径，顶部默认为最新。 如果要更改列表中路径的序列将路径在列表中向上或向下移动一个位置。
应用路径	单击 "相对路径"列表中的路径并单击此按钮以将模型中的任何现有完整路径名称更新为共享相对路径名称。例如： d:\java\源\main.java 可能会变成 %JAVA_SRC%\main.java
展开路径	单击 "相对路径"列表中的路径，然后单击此按钮以删除相对路径并替换为完整路径名称（与应用路径按钮的效果相反）。
新的	单击此按钮可清除数据字段，以便您可以定义另一个本地路径。
节省	定义本地路径后，单击此按钮将其保存并将其添加到 "相对路径"列表中。
删除	单击 "相对路径"列表中的路径，然后单击此按钮可将路径从列表中完全删除。
关	单击此按钮关闭对话框，保存对列表的任何更改。

注记

- 您还可以在图表上设置超链接（用于Enterprise Architect命令）以访问 "本地路径"对话框，以切换、更新或扩展您当前的本地路径

- 如果为链接文件扩展或应用路径的行为将创建重复记录，则该过程将跳过该记录并在过程结束时显示一条消息

语言宏

在对 C++ 等语言进行逆向工程时，您可能会发现预处理器指令分散在整个代码中。这可以使代码管理更容易，但会妨碍底层 C++ 语言的解析。

为了帮助解决这个问题，您可以包含任意数量的宏定义，这些宏定义在相工程的解析阶段会被忽略。如果您有此功能，最好先使用适当的编译器预处理代码；这样，复杂的宏定义和定义被扩展并且可以很容易地解析。如果你没有这个功能，那么这个选项提供了一个方便的替代品。

访问

功能区	设置 >参考> 设置 > 预处理器宏或 开发>源代码>选项>配置>定义预处理器宏

定义一个宏

节	行动
1	选择 预处理器宏”菜单选项。 将显示 语言宏”对话框。
2	点击加新按钮。
3	输入宏的详细信息。
4	点击确定按钮。

嵌入在声明中的宏

宏有时在类和操作的声明中使用，如下例所示：

```
类 __declspec Foo
{
int __declspec Bar( int p);
};
```

如果 declspec 定义为 C++ 宏，如前所述，导入的类和操作包含一个名为 DeclMacro1 的标记值，其值为 __declspec（后续宏将定义为 DeclMacro2、DeclMacro3 等）。

在正向工程期间，这些标记值用于重新生成代码中的宏。

定义复杂宏

有时为可以跨越多行的复杂宏定义规则很有用； Enterprise Architect忽略规则定义的整个代码段。

可以在Enterprise Architect中定义此类宏，如这两个示例中所示；这两种类型可以组合在一个定义中。

块宏

BEGIN_INTERFACE_PART ^ END_INTERFACE_PART

^ 符号代表宏的主体 - 这允许从一个宏跳到另一个宏； ^ 符号周围的空格是必需的。

函数宏

RTTI_EMULATION()

Enterprise Architect跳过了令牌，包括括号内的所有内容。

函数宏也可以包含函数体：

RTTI_EMULATION() {}

在这种情况下， Enterprise Architect跳过了标记，包括括号内和大括号内的所有内容。请注记，如果函数宏包含函数主体，则它不能与块宏组合。

注记

- 您可以使用 设置>模型>传输>导出参考”和 导入参考”选项在模型之间传输这些语言宏（或预处理器宏）定义；宏导出为宏列表

开发编程语言

您可以在Enterprise Architect中使用一系列已建立的编程语言，但如果这些语言不适合您的需求，您可以开发自己的。然后，您将通过您可能仅为此目的或更广泛目的而开发的MDG 技术将其应用到您的模型中。开发语言后，您还可以编写 MDA变换模板，将平台无关模型或另一种语言的模型转换为新语言的模型，反之亦然。

访问

功能区	开发 >源代码 > 选项 > 编辑代码模板
键盘快捷键	Ctrl+Shift+P

开发编程语言

节	描述
1	在代码模板编辑器中，单击 新语言"按钮，然后在 编程语言数据类型"对话框中，单击 添加产品"按钮。 输入新的编程语言名称并为其定义数据类型。在将至少一种数据类型添加到语言之前，您无法在代码模板编辑器中访问新语言。
2	定义完所有需要的数据类型后，单击关闭按钮，在代码模板编辑器的 语言"字段中选择语言，然后开始编辑或创建新语言的代码模板。 代码模板定义了系统应该如何执行： ● 使用新语言对模型进行正向代码工程 ● 行为代码生成（如果合适的话）
3	如果您愿意，您还可以为您的新语言定义源代码选项。这些是数据类型或代码模板未提供的语言附加设置，有助于定义系统在生成和逆向工程代码时如何处理该语言。 代码选项仅通过MDG 技术提供给您的模型。
4	为您的语言定义语法是一个可选步骤，它提供了两个主要好处： ● 将现有代码逆向工程到您的模型中 ● 在代码生成期间进行同步，以便自上次生成以来对文件所做的更改不会丢失。 要访问语法编辑器，请选择 开发 >源代码 > 语法编辑器"功能区选项。
5	如果您打算对您的新编程语言进行 MDA 转换（或从该语言转换），您还可以为其编辑和创建转换模板。创建转换模板的过程与创建代码模板的过程非常相似。
6	为您的新语言创建数据类型、代码模板、代码选项、语法和转换模板后，您可以将它们合并并分发到MDG 技术中。

代码模板框架

当您使用Enterprise Architect从模型生成代码或转换模型时，系统参考代码模板框架 (CTF) 的参数来定义它应该如何：

- 正向工程一个UML模型
- 生成行为准则
- 执行模型驱动架构（MDA）变换驱动
- 在数据库建模中生成DDL

A标准模板可用于直接生成代码和转换；如果您不想使用标准的 CTF 配置，您可以自定义它们以满足您的需求。

CTF模板

模板类型	细节
代码模板	当您对类模型进行正向工程时，代码模板定义了如何为给定的编程语言生成骨架代码。一种语言的模板会自动与该语言相关联。 这些模板以纯文本形式编写，其语法与标记语言和脚本语言的某些方面相同。
模型变换模板	模型变换模板了一种完全可配置的方法，用于定义模型驱动架构(MDA) 转换如何将模型元素和模型片段从一个域转换到另一个域。 这个过程是两层的。它创建一种中间语言（可以查看以进行调试），然后对其进行处理以创建对象。
行为代码生成模板	Enterprise Architect支持UML行为模型的用户可定义代码生成。 这适用于标准代码模板框架，但包括特定的Enterprise Architect仿真库 (EASL) 代码生成宏。
DDL模板	DDL模板与代码生成模板非常相似，但它们已被扩展为使用自己的一组基本模板、宏、函数宏和模板选项来支持 DDL 生成。

代码模板定制化

Enterprise Architect可帮助您从UML模型为各种编程语言生成源代码。提供开箱即用的标准模板（映射），但您可以使用实用且灵活的代码模板框架 (CTF) 自定义生成代码的方式。这个复杂的框架允许您自定义代码生成方式的每一个细节，包括为基础产品不支持的语言创建新模板的功能。例如， JavaScript不是受支持的语言之一，但可以快速编写一系列模板以从UML模型生成JavaScript 。在这些情况下，现有模板可作为新语言的有用起点和参考。

代码模板框架还提供了生成行为模型的机制，并用于转换模板。

特征

特征	细节
默认模板	默认代码模板内置于Enterprise Architect中，用于支持正向工程的语言。
代码模板编辑器	A代码模板编辑器用于创建和维护用户定义的代码模板。
自定义代码模板	模板语法的描述以及可用于控制模板效果的宏和函数。
同步代码	用于同步代码的默认代码模板A子集。

代码和转换模板

代码模板和变换 (模型变换) 模板定义了系统应如何以Enterprise Architect支持的一种或其他编程语言生成或转换代码。每种语言都有广泛的基础模板，每个模板都定义了特定代码结构的生成方式。您可以按原样使用这些基本模板，也可以自定义模板并将其添加到模板中，以更好地支持您使用标准语言或您可能为系统定义的其他语言。您可以通过代码模板编辑器或变换编辑器来审阅、更新和创建模板。

基本模板在两个编辑器中列出的顺序与对象的层次顺序及其要处理的部分有关。调用是从某些基本模板向其他模板发出的，您可以向基本模板和您自己的自定义模板添加更多调用。默认情况下，文件模板是通过模板生成代码的起点；一个文件由可以包含属性和操作的类组成。

访问

功能区	开发 >源代码 > 选项 > 编辑代码模板 设计>包>变换>变换模板
键盘快捷键	Ctrl+Shift+P（代码生成模板） Ctrl+Alt+H（MDA变换模板）

模板的应用

行动	细节

调用模板	在任何模板中，您都可以使用 %TemplateName% 调用其他模板。封闭的百分比 (%) 符号表示宏。
	您可以将它用于对 ClassBody模板%ClassBody% 的一次调用，如下所示：
	% list = "TemplateName" @separator= "\n" @indent= " " %
	%list 宏对当前模板范围内的所有对象执行迭代传递，并为每个对象调用 TemplateName：
	% list = "ClassBody" @separator= "\n" @indent= " " %
	在生成或转换之后，每个宏被替换以产生生成的输出；对于像 C++ 这样的语言，处理这个模板的结果可能是：
	/**
	* 这是一个使用代码模板生成的示例类注记
	* @author Sparx Systems
	*/
	类ClassA：公共ClassB
	{
	...
	}

代码模板的执行	每个模板可能只作用于特定的元素类型；例如，ClassNotes模板只作用于UML类和接口元素。 当前生成代码的元素被称为在范围内；如果范围内的元素是构造型的，则系统搜索已为该构造型定义的模板。如果找到专门的模板，则执行；否则使用

	基本模板的默认实现。 模板按顺序逐行处理，用模型中的基础文本值替换每个宏。
在项目之间转移模板	如果您编辑基本代码生成或变换模板，或创建自定义模板，您可以将它们从一个项目复制到另一个项目作为参考。

基础模板

代码模板框架由许多基本模板组成。每个基本模板将UML的特定方面转换为面向对象语言的相应部分。

基本模板形成一个层次结构，在不同的编程语言中略有不同。在与诸如 C# 或Java （没有头文件）等语言相关的典型模板层次结构中，模板可以建模为类，但通常只是纯文本。对于具有单独实现模板的 C++ 和 Delphi 等语言，这种层次结构会稍微复杂一些。

每个基本模板都必须专门用于代码工程；特别是，每个模板都专门用于支持的语言（或 产品”）。例如，有一个为 C++ 定义的 ClassBody模板，一个为 C# 定义的，另一个为Java定义的，以此类推；通过专门化模板，您可以定制为相应的UML实体生成的代码。

一旦基础模板专门用于给定语言，它们就可以进一步专门基于：

- A类的刻板印象，或
- 特征A刻板印象（特征可以是操作或属性）

例如，这种类型的特化使得被构造为 «property» 的 C# 操作具有与普通操作不同的操作体模板；然后可以根据类原型进一步专门化操作体模板。

CTF 中使用的基本模板

模板	描述
属性	从UML属性生成成员变量A顶级模板。
属性声明	由 Attribute模板用于生成成员变量声明。
属性注记	由 Attribute模板用来生成成员变量注记。
类	用于从UML类生成类A顶级模板。
类基地	由类模板用于在派生类的继承列表中生成基类名称，其中基类在模型中不存在。
类体	由类模板用来生成类的主体。
类声明	由类模板用来生成类的声明。
类接口	类模板用于在派生类的继承列表中生成接口名称，该接口在模型中不存在。
类注记	由类模板用来生成类注记。
文件	用于生成源文件A顶级模板。 对于 C++ 等语言，这对应于头文件。
导入部分	在文件模板中用于生成外部依赖项。
链接属性	用于生成从UML关联派生的属性A顶级模板。
关联属性注记	被链接属性模板用来生成属性注记。
链接属性声明	由链接属性模板用于生成属性声明。

链接类	由类模板用于在派生类的继承列表中为模型中的类元素生成一个基类名称，该类元素是当前类的父类。
链接类接口	由类模板用于在派生类的继承列表中生成接口名称，用于模型中作为当前类父级的接口元素。
命名空间	用于从UML包生成名称空间A顶级模板（尽管并非所有语言都有名称空间，但此模板可用于生成等效结构，例如Java中的包）。
命名空间体	由命名空间模板用来生成命名空间的主体。
命名空间宣言	由命名空间模板用来生成命名空间声明。
手术	用于从UML类的操作生成操作A顶级模板。
操作体	由模板用于生成UML操作的主体。
运营声明	操作模板用于生成操作声明。
操作注记	由操作模板用于生成操作文档。
参数	由操作声明模板用于生成参数。

模板用于为具有单独接口和实现部分的语言生成代码

模板	描述
类Impl	用于生成类的实现A顶级模板。
类体Impl	由类Impl模板用来生成类成员的实现。
文件实现	用于生成实现文件A顶级模板。
文件注记Impl	由文件Impl模板用于在源文件中生成注记。
导入Section Impl	由文件Impl模板用于生成外部依赖项。
操作实现	用于从UML类的操作生成操作A顶级模板。
操作体Impl	由模板用于生成UML操作的主体。
操作声明实现	操作模板用于生成操作声明。
操作注记Impl	由操作模板用于生成操作文档。

导出代码生成和变换模板

可以将模型中的代码生成和变换模板导出为 .xml 文件。然后，您可以将该文件以及模板导入到其他模型中，作为参考数据。您可以导出自定义模板，其中包括您或其他用户创建和更新的模板，以及已定制的基本（标准）模板。您不需要导出未更改的基本模板，因为这些在Enterprise Architect的每个安装中都可用。

访问

功能区	设置>>模型>传输>导出参考

导出代码生成模板或变换模板

节	行动
1	在 “导出参考”对话框的 “名称”列表中，选择要导出的模板。 该列表包括任何标准代码生成或变换模板，以及您创建或更改的任何自定义模板。 您可以选择一个或多个模板以导出到单个 XML 文件，方法是在单击模板名称时按 Ctrl 或 Shift。
2	点击导出按钮。
3	当系统提示您这样做时，输入带有 .xml 扩展名的有效文件名。
4	单击保存按钮和确定按钮。 这模板导出到文件中；您可以使用任何文本或 XML 查看器来检查文件。

导入代码生成和变换模板

如果您已经从Enterprise Architect模型中导出了代码生成和/或变换模板，您可以将它们作为参考数据导入到其他Enterprise Architect模型中。

访问

功能区	设置>模型>传输>导入参考

导入生成和/或变换模板

节	行动
1	在 “导入参考”对话框中，单击选择文件按钮并浏览到包含所需代码生成或变换模板的 .xml 文件。
2	选择一个或多个模板数据集的名称，然后单击导入按钮。

同步代码

Enterprise Architect在这些编程语言的前向同步期间使用代码模板：

- 动作脚本
- C
- C++
- C #
- 德尔福
- Java
- PHP
- Python
- VB
- VB.Net

当与UML模型同步时，源中会发生三种类型的变化：

- 现有部分是同步的：例如，更新操作声明中的返回类型
- 将新部分添加到现有特征中：例如，将注记添加到以前没有的类声明中
- 添加了新的特征和元素：例如，将新操作添加到类中

这些更改中的每一个对 CTF 都有不同的影响，并且必须由Enterprise Architect以不同方式处理，如以下主题中所述：

- 同步现有部分
- 加新部分到现有特征
- 加新特征和元素

可以同步的代码段

同步期间仅使用 CTF 基本模板的一个子集。该子集对应于Enterprise Architect在源代码中识别的不同部分。

代码模板	代码部分
类注记	类声明之前的注释。
类声明	直到并包括类父母。
属性注记	属性声明之前的注释。
属性声明	直到并包括终止字符。
操作注记	操作声明前的注释。
操作注记Impl	至于操作注记。
运营声明	直到并包括终止字符。
操作声明实现	直到并包括终止字符。
操作体	包括大括号之间的所有内容。

操作体Impl	至于操作体。

同步现有部分

当源代码中的现有部分与相应模板生成的结果不同时，该部分将被替换。

例如，考虑这个 C++类声明：

（asm） A类：公共B

现在假设你添加了一个从类A到类C的继承关系；整个类声明将被替换为类似这样的东西：

（asm） A类：公共B ，公共C

加新版块

这些部分可以作为新部分添加到源代码中的现有特征中：

- 类注记
- 属性注记
- 操作注记
- 操作注记Impl
- 操作体
- 操作体Impl

假设在这个例子中，类A在你最初生成代码时没有注记：

（asm） A类：公共B ，公共C

如果您现在在模型中为类A指定注记， Enterprise Architect会尝试在同步期间通过执行类注记模板从模型中添加新注记。

要为要插入的新部分腾出空间，您可以通过同步宏指定要附加到该部分的空白空间。

加新特征和元素

这些特征和元素可以在同步期间添加到源代码中：

- 属性
- 内部类
- 操作

它们是通过为模型中的每个新元素或特征执行相关模板来添加的。

Enterprise Architect试图通过查找类的列表宏中指定的缩进来保留代码中新特征的适当缩进；对于使用命名空间的语言，可以使用 “synchNamespaceBodyIndent” 。

在同步期间，在（非全局）命名空间中定义的类根据为此宏设置的值缩进。

该值被忽略：

- 对于在设置为根命名空间的包中定义的类，或
- 如果在 “首选项”对话框（“>开始外观>首选项>首选项>源代码工程> <语言>”）上的相应语言页面（C#、C++ 或 VB.Net）中的 “生成命名空间”选项设置为False

代码模板编辑器

代码编辑器提供公共代码编辑器的智能感知模板功能，包括各种宏的功能。有关智能感知代码和公共代码编辑器的详细信息，请参阅编辑源代码主题

访问

功能区	开发 >源代码 > 选项 > 编辑代码模板
键盘快捷键	Ctrl+Shift+P

选项

选项	行动
语	选择编程语言。
新语言	显示 编程语言数据类型"对话框，它使您能够包含Enterprise Architect支持的编程语言以外的编程语言，用于创建或编辑代码模板。
模板	显示活动模板的内容，并打开编辑器修改模板。
模板	列出基本代码模板；活动模板突出显示。 'Modified' 字段指示您是否更改了当前语言的默认模板。
构造型覆盖	列出活动基本模板。 "Modified"字段指示您是否修改了默认的原型模板。
加新自定义模板	调用一个对话框来创建一个自定义的原型模板。
加新Stereotyped Override	为当前选定的基本模板模板
获取默认模板	使用活动模板的默认版本更新编辑器显示。
节省	用编辑器的内容覆盖活动模板。
删除	如果您已经覆盖了活动模板，则覆盖将被删除并替换为相应的默认代码模板。

注记

- 用户修改和用户定义的代码模板可以作为参考数据导入和导出（参见共享参考主题）；为每种语言定义的模板在 导出参考"对话框中由带有后缀导出的语言名称指示 - 如果一种语言不存在模板，则对话框中没有该

语言的条目

创建新的自定义模板

Create New Custom Template 对话框提供了为当前编程或数据库管理系统(DBMS) 语言创建自定义模板的能力，具体取决于使用代码模板编辑器编辑的信息。

加载此对话框后，系统将提示您输入模板类型和模板名称的值。为了保存一个新的模板类型和名称都是必需的。

选项

选项	行动
模板类型	为新的自定义模板选择模板类型。
模板名称	输入新自定义模板的名称。
确定	保存新自定义模板的详细信息。
取消	关闭 Create New Custom Template 对话框并释放所有未保存的更改。

注记：

“ <none> ”类型的所有模板都被视为函数，因此Enterprise Architect将自动删除输入到名称中的所有空格字符。

代码模板语法

代码模板是使用Enterprise Architect的代码模板编辑器编写的。代码模板编辑器支持代码模板框架语言的语法高亮显示。

语法元素

元素	细节
基本构造	模板可以包含： ● 文字文本 ● 变量 ● 宏 ● 调用其他模板
注释	如果要向模板添加注释，请使用以下命令： $COMMENT="文本" 其中 "text" 是评论的文本；这必须用引号引起来。 该命令区分大小写，并且必须以大写形式键入。

文字文本

给定模板中不属于宏或变量定义/引用的所有文本都被视为文字文本。除了被忽略的空行之外，文字文本直接从模板替换到生成的代码中。

考虑Java类声明模板的这段摘录：

$bases = "基础"

类 % 类名 % $bases

在最后一行，单词 'class'，包括随后的空格，将被视为文字文本，因此对于名为 'foo' 的类将返回输出：

类 fooBase

变量 $bases 后面A空行对输出没有影响。

插入系统字符：

%、$、" 和 \ 字符在模板语法中具有特殊含义，不能始终用作文字文本。如果必须从模板中生成这些字符，则可以使用这些直接替换宏安全地复制它们：

宏	行动
%dl%	产生一个文字 $ 字符。
％个人电脑％	产生一个文字 % 字符。
%qt%	产生一个文字 " 字符。
%sl%	产生一个文字 \ 字符

注记

字符串联合运算符（“+”、 “+=”）不是必需的，但可以使用

变量

模板变量提供了一种在模板中存储和检索数据的便捷方式。本节说明如何定义和引用变量。

变量定义

变量定义采用基本形式：

$<名称> = <值>

其中 <name> 可以是任何字母数字序列，而 <value> 派生自宏或其他变量。

A简单的示例定义是：

$foo = %className%

可以使用以下值定义变量：

- 替换，函数或列表宏
- 字符串文字，用双引号括起来
- 变量引用

定义规则

这些规则适用于变量定义：

- 变量在定义它们的模板内具有全局范围，其他模板无法访问
- 每个变量必须在行首定义，没有任何中间空格
- 变量通过在名称前加上 $ 来表示，如 $foo
- 变量不必在定义之前声明
- 必须使用赋值运算符 (=) 或加法赋值运算符 (+=) 定义变量
- 可以使用加法运算符 (+) 将多个术语组合在一个定义中

例子

使用替换宏：

$foo = %opTag:"bar"%

使用文字string ：

$foo = "酒吧"

使用另一个变量：

$foo = $bar

使用列表宏：

$ops = %list="操作" @separator="\n\n" @indent="\t"%

使用加法赋值运算符 (+=)：

$body += %list="操作" @separator="\n\n" @indent="\t"%

该定义相当于：

$body = $body + %list="操作" @separator="\n\n" @indent="\t"%

使用多个术语：

$templateArgs = %list="ClassParameter" @separator=", "%
$template="模板<" + $templateArgs + ">"

变量引用

可以使用以下形式的引用来检索变量值：
$<名称>
其中 <name> 可以是先前定义的变量。
可以使用变量引用：

- 作为宏的一部分，例如函数宏的参数
- 作为变量定义中的一个术语
- 作为将变量值直接替换到输出中

在定义之前引用变量是合法的。在这种情况下，假定变量包含一个空string值：“”

变量引用 -示例1

使用变量作为宏的一部分。这是默认 C++ ClassNotes模板的摘录。
$wrapLen = %genOptWrapComment%
$style = %genOptCPPCommentStyle%（定义变量以存储样式和换行长度选项）
%if $style == "XML.NET"% (参考$style 作为条件的一部分)
%XML_COMMENT($wrapLen)%
％别的％
%CSTYLE_COMMENT($wrapLen)%（参考$wrapLen 作为函数宏的参数）
％万一％

变量引用 -示例

使用变量引用作为变量定义的一部分。
$foo = "foo" (定义我们的变量)
$bar = "酒吧"
$foobar = $foo + $bar ($foobar现在包含值 foobar)

变量引用 -示例

将变量值代入输出。
$bases=%classInherits%（将 ClassInherits模板的结果存储在 $bases 中）
类%className%$bases（现在在类名后面输出$bases的值）

宏

宏提供对UML模型中元素字段的访问，也用于构建生成的输出。所有宏都包含在百分号 (%) 内，如下所示：
%<宏名>%
在一般情况下，宏（包括 % 分隔符）被替换为输出中的文字。例如，考虑类声明模板中的这个项目：
... 类 %className% ...
字段替换宏 %className% 将导致当前类名在输出中被替换。因此，如果生成的类名为 Foo，则输出将是：
...类Foo ...
CTF 包含多种类型的宏：

- Template Substitution Macros
- Field Substitution Macros
- Substitution Examples
- Attribute Field Substitution Macros
- Class Field Substitution Macros
- Code Generation Option Field Substitution Macros
- Connector Field Substitution Macros
- Constraint Field Substitution Macros
- Effort Field Substitution Macros
- File Field Substitution Macros
- File Import Field Substitution Macros
- Link Field Substitution Macros
- Linked File Field Substitution Macros
- Metric Field Substitution Macros
- Operation Field Substitution Macros
- Package Field Substitution Macros
- Parameter Field Substitution Macros
- Problem Field Substitution Macros
- Requirement Field Substitution Macros
- Resource Field Substitution Macros
- Risk Field Substitution Macros
- Scenario Field Substitution Macros
- Tagged Value Substitution Macros
- Template Parameter Substitution Macros
- Test Field Substitution Macros
- Function Macros
- Control Macros
- List Macro
- Branching Macros
- Synchronization Macros
- The Processing Instruction (PI) Macro
- EASL Code Generation Macros

模板替换宏

模板替换宏对应于基本模板，并导致命名模板的执行。按照惯例，模板宏是根据 Pascal 大小写命名的。

结构：%<模板名称>%

其中 <TemplateName> 可以是本主题中列出的模板之一。

当从另一个模板中引用一个模板时，它是相对于当前范围内的元素生成的。特定模板是根据范围内元素的构造型选择的。

如前所述，各种模板之间存在隐式层次结构。应注意保持模板引用的合理层次结构。例如，在任何 Attribute 或 Operation 模板中使用 %ClassInherits% 宏是没有意义的。相反，Operation 和 Attribute 模板是为在 ClassBody模板中使用而设计的。

CTF 中的模板替换宏

- 属性
- 属性声明
- AttributeDeclarationImpl
- 属性注释
- 类
- 类库
- 类体
- ClassBodyImpl
- 类声明
- ClassDeclarationImpl
- ClassImpl
- 类继承
- 类接口
- 课堂笔记
- 类参数
- 文件
- FileImpl
- 进口节
- ImportSectionImpl
- 内部类
- 内部类实现
- 关联属性
- 链接属性声明
- LinkedAttributeNotes
- 链接类库
- 链接类接口
- 命名空间
- 命名空间体
- 命名空间声明
- 命名空间实现

- 手术
- 操作体
- OperationBodyImpl
- 操作声明
- OperationDeclarationImpl
- OperationImpl
- 操作说明
- 参数

字段替换宏

字段替换宏提供对模型中数据的访问。特别是，它们用于从以下位置访问数据字段：

- 包
- 课程
- 属性
- 操作，和
- 参数

字段替换宏根据 Camel 大小写命名。按照惯例，宏以相应模型元素的缩写形式作为前缀。例如，与属性相关的宏以 att 开头，就像在 %attName% 宏中一样，用于访问范围内的属性名称。

如果选中该框，则表示复选框的宏将返回T值。否则值为空。

此表列出了少量项目字段替换宏。此字段替换宏部分的子主题中列出了特定于类型的宏。

项目宏

宏名称	描述
日期时间	当前时间，格式为：DD-MMM-YYYY HH:MM:SS AM/PM。
EAGUID	这一代A唯一GUID 。
版本	程序版本（位于 关于Enterprise Architect″对话框中，选择 开始>帮助>帮助>关于 EA”）。

替换示例

字段替换宏可以通过以下两种方式之一使用：

- 直接替代或
- 条件替换

直接替代

这种形式直接将范围内元素的对应值代入输出。

结构：%<macroName>%

其中 <macroName> 可以是字段替换宏库表中列出的字段。

例子

- ％班级名称％
- %操作名称%
- %attName%

条件替换

这种形式的宏允许根据宏的值进行替代替换。

结构：%<macroName> (== "<text>") ? <subTrue> (: <subFalse>) %

在哪里：

- () 表示括号之间的值是可选的
- <text> 是一个string ，表示宏的可能值
- <subTrue> 和 <subFalse> 可以是带引号的字符串和关键字值的组合；在使用该值的地方，将其替换为输出中的宏值

例子

- %classAbstract==" T " ？ 纯的″ ：″″％
- %opStereotype=="操作员" ？ 操作员″ ：″″％
- %paramDefault != "" ? “=″值：“”%

如果条件失败，这三个示例不输出任何内容。在这种情况下，可以省略False条件，导致这种用法：

- %classAbstract==" T " ？ 纯的″％
- %opStereotype=="操作员" ？ 操作员″％
- %paramDefault != "" ? " = "值%

两个块的第三个示例显示了对非空值或存在的比较检查。这个测试也可以省略。

- % 参数默认值？ “=″值：“”%
- % 参数默认值？ “=″值%

所有这些包含 paramDefault 的示例都是等价的。如果作用域中的参数的默认值为 10，则每个参数的输出通常

为：

= 10

注记

- 在条件替换宏中，<macroName> 后面的任何空格都会被忽略；如果输出中需要空格，则应将其包含在带引号的替换字符串中

属性字段替换宏

此表列出了每个属性字段替换宏。
字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值 "T"。否则值为空。

属性宏

宏名称	描述
别名	属性"对话框：别名。
attAllowDuplicates	属性详细信息"对话框：允许重复"复选框。
attClassifierGUID	当前属性的分类器的唯一GUID。
属性集合	属性详细信息"对话框：属性是一个集合"复选框。
attConst	属性"对话框：常量"复选框。
attContainerType	属性详细信息"对话框：容器类型。
攻击遏制	属性"对话框：遏制。
属性派生	属性"对话框：派生"复选框。
attGUID	当前属性的唯一GUID。
姓名首字母	属性"对话框：初始。
attIsEnumLiteral	属性"对话框：是文字"复选框。
身份标识	属性详细信息"对话框："isID"复选框。
属性长度	列"对话框：长度。
attLowerBound	属性详细信息"对话框：下限。
名称	属性"对话框：名称。
附注	属性"对话框：注记。
attOrderedMultiplicity	属性详细信息"对话框：有多重性"复选框。
属性	属性"对话框：属性"复选框。
attQualType	由命名空间路径（如果生成命名空间）和分类器路径（点分隔）限定的属性类型。如果未设置属性分类器，则相当于attType宏。
攻击范围	属性"对话框：范围。

attStatic	属性"对话框： 静态"复选框。
态度刻板印象	属性"对话框：构造型。
属性类型	属性"对话框：类型。
attUpperBound	属性详细信息"对话框：上限。
挥发性的	属性详细信息"对话框： 瞬态"复选框。

类字段替换宏

此表提供了访问代码生成和变换模板中每个可用类属性的方法列表。

字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值 "T"。否则值为空。

类宏

宏名称	描述
元素类型	元素类型：接口或类。
类抽象	类'属性'对话框：'摘要'复选框（'详细信息'选项卡）。
类别名	类 属性"对话框： 别名"字段。
类参数	类'Detail' 对话框：C++模板：参数。
类作者	类'属性'对话框：'作者'字段。
类基名	'类型' 对话框：类名称（用于子类和基类之间不存在连接器的情况）。
类BaseScope	反向工程的继承范围。（用于子类和基类之间不存在连接器的情况。）
类BaseVirtual	作为反向工程的继承的虚拟属性。（用于子类和基类之间不存在连接器的情况。）
类复杂度	类'属性'对话框：'复杂性'字段。
类创建	创建类的日期和时间。
类GUID	当前类的唯一GUID 。
类HasConstructor	查看当前object中的方法列表，根据当前语言的约定，如果其中一个是默认构造函数，则返回T通常与 genOptGenConstructor 宏一起使用。
类HasCopy构造函数	查看当前object中的方法列表，根据当前语言的约定，如果其中一个是复制构造函数，则返回T通常与 genOptGenCopyConstructor 宏一起使用。
类HasDestructor	查看当前object中的方法列表，根据当前语言的约定，如果其中一个是析构函数，则返回T通常与 genOptGenDestructor 宏一起使用。
classHasParent	True ，如果作用域中的类具有一个或多个基类。
类HasStereotype	True ，如果作用域中的类具有与构造型名称匹配的构造型（您可以选择将其指定为完全限定）。因此，它检查 a类具有的所有构造型，如果其中任何一个是指定的构造型或它的特化，则返回 "T"。例如： • %classHasStereotype:"block"% 将为来自任何 SysML 版本的任何 block-stereotyped类返回 ' T '，包括 associationBlock • %classHasStereotype:"SysML1.4::block"% 将专门匹配 SysML 1 .4 版本

	稍后将其与 classStereotype 进行比较。
类进口	'Code Gen' 对话框：导入。
类是活动的	类'高级'对话框：'Is Active'复选框。
classIsAssociationClass	True ，如果关联是 AssociationClass 连接器。
类实例化	True ，如果类是实例化的模板类。
类IsLeaf	类'高级'对话框：'是叶子'复选框。
类IsRoot	类'高级'对话框：'是根'复选框。
类IsSpecification	类'高级'对话框：'是规范'复选框。
类关键字	类'属性'对话框：'关键字'字段。
类语言	类'属性'对话框：'语言'字段。
类宏	为类定义A以空格分隔的宏列表。
类修改	上次修改类的日期和时间。
类多重性	类'高级'对话框：多重性。
班级名称	类'属性'对话框：'名称'字段。
课堂笔记	类'属性'对话框：'注记'字段。
类参数默认值	类'详细信息'对话框。
类参数名称	类'详细信息'对话框。
类参数类型	类'详细信息'对话框。
类持久化	类的 属性"对话框： 持久性"字段（ 详细信息"选项卡）
类阶段	类'属性'对话框：'相'字段。
类QualName	以它的外部 Classes 为前缀的类名。类名用双冒号 (::) 分隔。
类作用域	类'属性'对话框：'范围'字段。
类刻板印象	类'属性'对话框：'构造型'字段。检索应用于类的第一个构造型的名称。在比较中使用时，它会检查第一个构造型是否与string完全匹配。 例如： %classStereotype=="enumeration" ？ 枚举"： 类"% 将此与前面的 classHasStereotype 进行比较。
类状态	类'属性'对话框：'状态'字段。

类版本	类'属性'对话框：'版本'字段。

代码生成选项字段替换宏

代码生成选项字段替换宏在以下任一选项的 "源代码工程"页面中定义的源代码生成选项上运行：

- 用于用户特定选项的 "首选项"对话框（ "开始>外观>首选项>首选项>源代码工程"），或
- '模型' 对话框（'Settings >model> Options'）用于模型特定的选项

有关选项划分的更多信息，请参阅源代码工程选项主题。

字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值 "T"。否则值为空。此表列出了每个代码生成选项字段替换宏。

代码生成选项宏

宏名称	描述
genOptActionScriptVersion	ActionScript 规范页面：默认版本。
genOptCDefaultAttributeType	C 规范页面：默认属性类型。
genOptCGenMethodNotesInBody	C 规范页面：方法注记在实施中。
genOptCGenMethodNotesInHeader	C 规格页面：方法注记在页眉中。
genOptCSynchNotes	C 规格页面：在生成中同步注记。
genOptCSynchCFile	C 规范页面：在生成中同步实现文件。
genOptCDefaultSourceDirectory	C 规格页面：默认源目录。
genOptCNamespaceDelimiter	C 规格页面：命名空间分隔符。
genOptCOOperationRefParam	C 规范页面：参考作为操作参数。
genOptCOOperationRefParamStyle	C 规格页面：参考参数样式。
genOptCOOperationRefParamName	C 规格页面：参考参数名称。
genOptCConstructorName	C 规范页面：默认构造函数名称。
genOptCDestructorName	C 规范页面：默认析构函数名称。
genOptCPPCommentStyle	C++ 规范页面：注解风格。

genOptCPPDefaultAttributeType	C++ 规范页面：默认属性类型。
genOptCPPDefaultReferenceType	C++ 规范页面：默认参考类型。
genOptCPPDefaultSourceDirectory	C++ 规范页面：默认源目录。
genOptCPPGenMethodNotesInHeader	C++ 规范页面： "方法注记在标题中"复选框。
genOptCPPGenMethodNotesInBody	C++ 规范页面：方法注记在体复选框。
genOptCPPGetPrefix	C++ 规范页面：获取前缀。
genOptCPPHeaderExtension	C++ 规范页面：标题扩展。
genOptCPPSetPrefix	C++ 规范页面：设置前缀。
genOptCPPSourceExtension	C++ 规范页面：源扩展。
genOptCPSynchNotes	C++ 规范页面：同步注记。
genOptCPPSynchCPPFile	C++ 规范页面：同步 CPP文件。
genOptCSDefaultAttributeType	C# 规范页面：默认属性类型。
genOptCSSourceExtension	C# 规范页面：默认文件扩展名。
genOptCSGenDispose	C#规范页面：生成Dispose。
genOptCSGenFinalizer	C# 规范页面：生成Finalizer。
genOptCSGen 命名空间	C#规范页面：生成命名空间。
genOptCSDefaultSourceDirectory	C#规范页面：默认源目录。
genOptDefaultAssocAttName	源代码工程页面：关联属性的默认名称。
genOptDefaultConstructorScope	物件生命周期页面：默认构造函数可见性。
genOptDefaultCopyConstructorScope	物件生命周期页面：默认复制构造函数可见性。
genOptDefaultDatabase	代码编辑器页面：默认数据库。

genOptDefaultDestructorScope	物件生命周期页面：默认析构函数构造函数可见性。
genOptGenCapitalizedProperties	'源代码工程'页面：'Capitalize Attribute Names for属性'复选框。
genOptGen评论	源代码工程”页面： 评论 -生成”复选框。
genOptGen构造函数	物件生命周期页面： 生成构造器”复选框。
genOptGenConstructorInline	物件生命周期页面： 构造函数内联”复选框。
genOptGenCopy构造函数	物件页面： 生成Copy Constructor”复选框。
genOptGenCopyConstructor内联	物件生命周期页面： 复制构造函数内联”复选框。
genOptGenDestructor	物件页面： 生成Destructor”复选框。
genOptGenDestructorInline	物件页面：“Destructor Inline”复选框。
genOptGenDestructorVirtual	物件生命周期页面： 虚拟析构函数”复选框。
genOptGenImplementedInterfaceOps	代码生成”页面： 实现接口的生成方法”复选框。
genOptGenPrefixBoolProperties	'源代码工程'页面：'使用'是'布尔属性Get（ ）'复选框。
genOptGenRoleNames	源代码工程”页面： 创建代码时自动生成角色名称”复选框。
genOptGenUnspecAssocDir	源代码工程”页面： 不生成未指定关联方向的成员”复选框。
genOptJavaDefaultAttributeType	Java规范页面：默认属性类型。
genOptJavaGetPrefix	Java规范页面：获取前缀。
genOptJavaDefaultSourceDirectory	Java规范页面：默认源目录。
genOptJavaSetPrefix	Java规范页面：设置前缀。
genOptJavaSourceExtension	Java规范页面：源代码扩展。
genOptPHPDefaultSourceDirectory	PHP 规范页面：默认源目录。

genOptPHPGetPrefix	PHP 规范页面：获取前缀。
genOptPHPSetPrefix	PHP 规范页面：设置前缀。
genOptPHPSourceExtension	PHP 规范页面：默认文件扩展名。
genOptPHP版本	PHP 规范页面：PHP版本。
genOptPropertyPrefix	源代码工程"页面：在生成 Get/Set属性时删除前缀。
genOptVB多用途	VB 规范页面： 多用途"复选框。
genOptVBPersistable	VB 规范页面："Persistable"复选框。
genOptVBDataBindingBehavior	VB 规范页面： 数据绑定行为"复选框。
genOptVBDataSourceBehavior	VB 规范页面： 数据源行为"复选框。
genOptVBGlobal	VB 规范页面： 全局命名空间"复选框。
genOptVBC可创建	VB 规范页面： 可创建"复选框。
genOptVB暴露	VB 规范页面： 公开"复选框。
genOptVBMTS	VB 规范页面：MTS 交易模式。
genOptVBNetGen 命名空间	VB.Net规范页面：生成命名空间。
genOptVB版本	VB 规范页面：默认版本。
genOptWrap 评论	源代码工程"页面：注释行的换行长度。

连接器字段替换宏

此表列出了每个连接器字段替换宏。

字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值 "T"。否则值为空。

连接器宏

宏名称	描述
连接器别名	连接器 "属性"对话框： "别名"字段。
连接器关联类ElemGUID	连接器的关联类元素的GUID 。
connectorAssociationClassElemName	连接器的关联类元素的名称。
连接器目的地访问	连接器的 "属性"对话框， "目标角色"选项卡：访问。
连接器目的地聚合	连接器的 "属性"对话框， "目标角色"选项卡：聚合。
connectorDestAlias	连接器的 "属性"对话框， "目标角色"选项卡：别名。
connectorDestAllowDuplicates	连接器的 "属性"对话框， "目标角色"选项卡： "允许重复"复选框。
connectorDestChangeable	连接器的 "属性"对话框， "目标角色"选项卡：可更改。
连接器目的地约束	连接器 "属性"对话框， "目标角色"选项卡：约束。
连接器DestContainment	连接器的 "属性"对话框， "目标角色"选项卡：包含。
连接器DestDerived	连接器的 "属性"对话框， "目标角色"选项卡： "派生"复选框。
connectorDestDerivedUnion	连接器的 "属性"对话框， "目标角色"选项卡："DerivedUnion"复选框。
连接器DestElem*	访问连接器目标端的元素属性的A组宏。 *（星号）是一个通配符，对应于类宏列表中的任何类替换宏。例如： • connectorDestElemAlias (classAlias) • connectorDestElemAuthor (classAuthor)
连接器DestElemType	连接器目标元素的元素类型。 （与 connectorDestElem* 宏分开，因为没有 classType 替换宏。）
连接器目的地特征*	访问连接器目标端的特征属性的A组宏。 *（星号）是一个通配符，对应于属性宏或操作宏列表中的任何属性或操作替换宏，具体取决于 connectorDestFeatureType。 例如： • connectorDestFeatureReturnClassifierGUID - 操作的返回分类器GUID

	• connectorDestFeatureContainment - 属性的包含
connectorDestFeatureType	连接器目标特征的类型。 • connectorDestFeatureType="属性" 或 "操作"
connectorDestMemberTypе	连接器 属性"对话框， 目标角色"选项卡：成员类型。
connectorDestMultiplicity	连接器的 属性"对话框， 目标角色"选项卡：多重性。
connectorDestNavigability	连接器的 属性"对话框， 目标角色"选项卡：可导航性。
connectorDestNotes	连接器的 属性"对话框， 目标角色"选项卡：角色注记。
connectorDestOrdered	连接器的 属性"对话框， 目标角色"选项卡： 有 "复选框。
连接器DestOwned	连接器的 属性"对话框， 目标角色"选项卡： 拥有"复选框。
连接器目的地限定符	连接器 属性"对话框， 目标角色"选项卡：限定符。
连接器目标角色	连接器的 属性"对话框， 目标角色"选项卡：角色。
连接器DestScope	连接器的 属性"对话框， 目标角色"选项卡：目标范围。
connectorDestStereotype	连接器的 属性"对话框， 目标角色"选项卡：构造型。
连接器方向	连接器属性：Direction。
连接器效果	'转移 约束"对话框： 影响"字段。
连接器防护	'物件流'和'转移 约束"对话： 守卫条件"字段。
连接器GUID	当前连接器的唯一GUID 。
连接器是关联类	True ，如果连接器是 AssociationClass 连接器。
连接器名称	连接器属性：名称。
连接器注意事项	连接器属性：注记。
连接器源访问	连接器的 属性"对话框， 源角色"选项卡：访问。
连接器源聚合	连接器的 属性"对话框， 源角色"选项卡：聚合。
连接器来源别名	连接器的 属性"对话框， 源角色"选项卡：别名。
connectorSourceAllowDuplicates	连接器的 属性"对话框， 源角色"选项卡：允许重复复选框。

connectorSourceChangeable	连接器的 属性"对话框， 源角色"选项卡：可更改。
连接器源约束	连接器 属性"对话框， 源角色"选项卡：约束。
连接器源遏制	连接器的 属性"对话框， 源角色"选项卡：包含。
连接器来源派生	连接器的 属性"对话框， 源角色"选项卡： 派生"复选框。
connectorSourceDerivedUnion	连接器的 属性"对话框， 源角色"选项卡："DerivedUnion"复选框。
connectorSourceElem*	A组访问连接器源端元素属性的宏。 *（星号）是一个通配符，对应于类宏列表中的任何类替换宏。例如： • connectorSourceElemAlias (classAlias) • connectorSourceElemAuthor (classAuthor)
connectorSourceElemType	连接器源元素的元素类型。 （与 connectorSourceElem* 宏分开，因为没有 classType 替换宏。）
连接器来源功能*	A组宏，用于访问连接器源端的特征属性。 *（星号）是一个通配符，它对应于属性宏或操作宏列表中的任何属性或操作替换宏，具体取决于 connectorSourceFeatureType。例如： • connectorSourceFeatureCode - 操作代码 • connectorSourceFeatureInitial - 属性的初始值
connectorSourceFeatureType	连接器源特征的类型。 • connectorSourceFeatureType="属性" 或 "操作"
连接器来源成员类型	连接器 属性"对话框， 源角色"选项卡：成员类型。
connectorSourceMultiplicity	连接器的 属性"对话框， 源角色"选项卡：多重性。
connectorSourceNavigability	连接器的 属性"对话框， 源角色"选项卡：可导航性。
connectorSourceNotes	连接器的 属性"对话框， 源角色"选项卡：角色注记。
connectorSourceOrdered	连接器的 属性"对话框， 源角色"选项卡： 有 "复选框。
connectorSourceOwned	连接器 属性"对话框， 源角色"选项卡： 拥有"复选框。
连接器源限定符	连接器 属性"对话框， 源角色"选项卡：限定符。
连接器源角色	连接器的 属性"对话框， 源角色"选项卡：角色。
连接器SourceScope	连接器 属性"对话框， 源角色"选项卡：目标范围。
connectorSourceStereotype	连接器的 属性"对话框， 源角色"选项卡：构造型。
连接器刻板印象	连接器 属性"对话框： 构造型"字段。

连接器触发器	'转移 约束”对话框： 触发器”字段。
连接器类型	连接器类型；例如，关联或概括。
连接器重量	物件流约束”对话框： 重量”字段。

约束字段替代宏

此表列出了每个 约束”字段替换宏。
字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值 “T ”。否则值为空。

约束宏指令

宏名称	描述
约束名	类”对话框， 约束”选项卡：名称。
约束说明	'类'对话框，'约束'标签：注记。
约束状态	类”对话框， 约束”选项卡：状态。
约束类型	'类'对话框，'约束'选项卡：类型。
约束权重	'类'对话框，'约束'选项卡：排序（手上/下）键。

工作量字段替换宏

此表列出了每个 "工作量"字段替换宏。
字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值 "T"。否则值为空。

工作量宏

宏名称	描述
努力名称	工作量窗口：工作量。
努力笔记	工作量窗口：注记（未标记）。
努力时间	工作量窗口：时间。
努力类型	工作量窗口：类型。

文件字段替换宏

此表列出了每个文件字段替换宏。

字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值 "T"。否则值为空。

文件宏

宏名称	描述
文件扩展名	正在生成的文件的文件类型扩展名。
文件名	正在生成的文件的名称。
文件名Impl	此生成的实现文件的文件名（如果适用）。
文件头	代码生成"对话框：标题。
文件导入	'Code Gen' 对话框：导入。对于支持的语言，这还包括从这些类型的关系派生的依赖项： ● 聚合 ● 关联 ● 属性分类器 ● 方法返回类型 ● 方法参数分类器 ● 概括 ● 实现（到接口） ● 模板捆绑(C++) ● 依赖
文件路径	正在生成的文件的完整路径。
文件路径Impl	此生成的实现文件的完整路径（如果适用）。

文件导入字段替换宏

此表列出了每个文件导入字段替换宏。
字段替换宏根据 Camel 大小写命名。如果选中该框，则表示复选框的宏将返回T值。否则值为空。

导入文件宏

宏名称	描述
导入类名	正在导入的类的名称。
导入文件名	正在导入的类的文件名。
导入文件路径	正在导入的类的完整路径。
importFromAggregation	如果类在此文件中具有一个聚合连接器，类T ，否则为 F。
从协会进口	T如果类与该文件中的某个类有关联连接器，否则为 F。
importFromAtt	如果当前文件中的一个类的属性是这个类的类型，则为T ，否则为 F。
importFromDependency	如果类在此文件中有一个到类的依赖连接器，则为T ，否则为 F。
importFromGeneralization	如果概括在此文件中有一个连接到 a类的连接器，类T ，否则为 F。
importFromMeth	如果当前文件中某个类的方法返回类型是这个类的类型，则为T ，否则为 F。
从参数导入	T如果当前文件中某个类的方法参数是这个类的类型；否则 F。
importFromPropertyType	如果该类有一个属性（部件/端口）类型到另一个类，类T ，否则为 F。
importFromRealization	如果类在此文件中有一个到类的实现连接器，则为T ，否则为 F。
importFromTemplateBinding	T如果类具有到此文件中的类的 TemplateBinding 连接器，否则为 F。
导入文件	如果类在当前文件中，则为T ，否则为 F。
导入包路径	带有'.'的包路径正在导入的类的分隔符。
导入相对文件路径	正在生成的文件的文件路径中导入的类的相对文件路径。

链接字段替换宏

如果您想提供对模型中有关连接器的数据的访问，特别是关联和概括，您可以使用 "链接字段替换"宏。宏名称采用 Camel 大小写。如果选中了复选框，则表示复选框的宏返回值 "T"；否则值为空。

链接宏

宏名称	描述/结果
链接AttAccess	关联'属性'对话框，目标角色：'访问'字段。
链接AttAggregation	关联'属性'对话框，源或目标角色：聚合。
链接AttCollectionClass	适用于范围内链接属性的集合。
链接AttContainment	关联'属性'对话框，目标角色：包含。
链接名称	"关联属性"对话框：目标。
链接AttNotes	关联'属性'对话框，目标角色：角色注记。
链接AttOwnedByAssociation	True ，如果未选中关联 "属性"对话框的 "角色"页面上的 "拥有"复选框。
链接AttOwnedByClass	True ，如果选中了关联 "属性"对话框的 "角色"页面上的 "拥有"复选框。
链接AttQualName	由命名空间路径（如果生成命名空间）和分类器路径（点分隔）限定的关联目标。
链接AttRole	关联'属性'对话框，目标角色：角色。
链接AttRoleAlias	'关联属性目标角色'对话框：别名
链接AttStereotype	关联 "属性"对话框，目标角色：构造型。
链接AttTargetScope	关联'属性'对话框，目标角色：目标范围。
链接卡	链接 "属性"对话框，目标角色：多重性。
链接GUID	当前连接器的唯一GUID 。
链接IsAssociationClass	True ，如果关联是 AssociationClass 连接器。
链接已绑定	如果在连接器上指定了任何 TemplateBinding，则返回T
链接参数Subs	返回指定参数的逗号分隔列表。
链接父名	概括属性'对话框：'目标'字段。

链接ParentQualName	概括由命名空间路径（如果生成命名空间）和分类器路径（点分隔）限定。
链接刻板印象	当前连接器的构造型。
链接虚拟继承	概括'属性'对话框：'虚拟继承'字段。

链接文件字段替换宏

此表列出了每个 "链接文件"字段替换宏。
字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值 "T"。否则值为空。

链接文件宏

宏名称	描述
链接文件最后写入	类的 "属性"对话框："文件"选项卡，"上次写入"字段。
链接文件注释	类'属性'对话框：'文件'选项卡，'注记'字段。
链接文件路径	类'属性'对话框：'文件'选项卡，'文件路径'字段。
链接文件大小	类'属性'对话框：'文件'选项卡，'大小'字段。
链接文件类型	类'属性'对话框：'文件'选项卡，'类型'字段。

度量字段替换宏

此表列出了每个度量字段替换宏。
字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值 "T"。否则值为空。

指标宏

宏名称	描述
指标名称	指标屏幕： 指标"字段。
公制注释	指标屏幕：（注记）字段。
公制类型	指标屏幕： 类型"字段。
公制重量	指标屏幕： 体重"字段。

操作字段替换宏

操作字段替换" 提供对模型中操作相关数据的访问。宏名称采用 Camel 大小写。如果选中了复选框，则表示复选框的宏返回值 "T"；否则值为空。

操作字段替换宏

宏名称	描述/结果
欧普文摘	操作"对话框： 虚拟"复选框。
别名	别名 操作"对话框：
操作行为	操作行为"对话框：行为。
操作码	操作行为"对话框：行为代码。
op并发	操作"对话框：并发。
opConst	操作"对话框： 常量"复选框。
操作GUID	当前操作的唯一GUID 。
opHasSelfRefParam	扫描当前操作中的参数列表，如果一种类型是类引用，则返回 "T"（这可能是 ClassA* 或 ClassA&，取决于 genOptCOperationRefParamStyle 代码生成选项字段替换宏的值）。
opImpl 宏	在此操作的实现中定义A以空格分隔的宏列表。
opIsQuery	操作"对话框："IsQuery"复选框。
操作宏	A此操作的声明中定义的以空格分隔的宏列表。
操作名称	操作"对话框：名称。
操作说明	操作"对话框：注记。
欧纯	操作"对话框： 纯"复选框。
操作返回数组	操作"对话框： 返回数组"复选框。
opReturnClassifierGUID	当前操作的分类器的唯一GUID 。
opReturnQualType	由命名空间路径（如果生成命名空间）和分类器路径（点分隔）限定的操作返回类型。如果没有设置返回类型分类器，则相当于 opReturnType 宏。
opReturnType	操作"对话框：返回类型。
示波器	操作"对话框：范围。

opStatic	操作"对话框： 静态"复选框。
操作刻板印象	操作"对话：构造型。
opSynchronized	操作"对话框： 同步"复选框。

包字段宏

此表列出了包字段替换宏。
字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值 "T"。否则值为空。

包宏

宏名称	描述
包摘要	包"对话：摘要。
包别名	包"对话：别名。
包作者	包"对话：作者。
包复杂度	包"对话框：复杂性。
包GUID	当前包的唯一GUID 。
包关键字	包"对话框：关键字。
包语言	包"对话框：语言。
包裹名字	包"对话框：名称。
包路径	表示包层级的string ，为作用域内的类。每个包名由点 (.) 分隔。
封装阶段	'包'对话：相。
包范围	包"对话框：范围。
包状态	'包'对话框：状态。
包装刻板印象	包"对话：构造型。
包版本	包"对话框：版本。

参数字段替换宏

此表列出了每个参数字段替换宏。

字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值 "T"。否则值为空。

参数宏

宏名称	描述
参数分类器GUID	当前参数的分类器的唯一GUID 。
参数默认值	操作 参数"对话框： 默认"字段。
参数固定	操作 参数"对话框： 固定"复选框。
参数GUID	当前参数的唯一GUID 。
参数枚举	True ，如果参数使用 enum 关键字 (C++)。
参数种类	操作 参数"对话框： 种类"字段。
参数名称	操作 参数"对话框： 名称"字段。
参数说明	操作 参数"对话框： 注记"字段。
参数QualType	由命名空间路径（如果生成命名空间）和分类器路径（点分隔）限定的参数类型。如果没有设置参数分类器，则相当于paramType宏。
参数类型	操作 参数"对话框： 类型"字段。

问题字段替换宏

此表列出了每个问题字段替换宏。
字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值 "T"。否则值为空。

问题宏

宏名称	描述
问题完成者	维护"对话框， 无素问题"选项卡：完成者。
问题完成日期	维护"对话框， 无素问题"选项卡：已完成。
问题历史	维护"对话框， 无素问题"选项卡：历史记录。
问题名称	维护"对话框， 无素问题"选项卡：名称。
问题笔记	维护"对话框， 无素问题"选项卡：描述。
问题优先	维护"对话框， 无素问题"选项卡：优先级。
问题提出者	维护"对话框， 无素问题"选项卡：引发者。
问题RaisedDate	维护"对话框， 无素问题"选项卡：已启动。
问题状态	维护"对话框， 无素问题"选项卡：状态。
问题版本	维护"对话框， 无素问题"选项卡：版本。

需求字段替换宏

此表列出了每个需求字段替换宏以及结果描述。
字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值 "T"。否则值为空。

需求宏

宏名称	描述
要求难度	属性"对话框： 要求"选项卡：难度。
要求上次更新	属性"对话框： 要求"选项卡：上次更新。
需求名称	属性"对话框： 要求"选项卡：简短描述。
需求备注	属性"对话框： 要求"选项卡：注记。
要求优先	属性"对话框： 要求"选项卡：优先级。
要求状态	属性"对话框： 要求"选项卡：状态。
需求类型	属性"对话框： 要求"选项卡：类型。

资源字段替换宏

此表列出了每个资源字段替换宏。
字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值 "T"。否则值为空。

资源宏

宏名称	描述
资源分配时间	资源分配窗口：分配时间。
资源结束日期	资源分配窗口：结束日期。
资源预期时间	资源分配窗口：预计时间。
资源消耗时间	资源分配窗口：时间消耗。
资源历史	资源分配窗口：历史。
资源名称	资源分配窗口：资源。
资源说明	资源分配窗口：描述。
资源百分比已完成	资源分配窗口：已完成 (%)。
资源角色	资源分配窗口：角色。
资源开始日期	资源分配窗口：开始日期。

风险字段替换宏

此表列出了每个风险字段替换宏。
字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值 "T"。否则值为空。

风险宏

宏名称	描述
风险名称	风险窗口：风险。
风险提示	风险窗口：（注记）。
风险类型	风险窗口：类型。
风险权重	风险窗口：权重。

字段替换宏

此表列出了每个场景字段替换宏以及结果说明。
字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值 "T"。否则值为空。

场景宏

宏名称	描述
场景GUID	方案的唯一 ID。在模型中明确识别场景。
场景名称	属性"对话框， 场景"选项卡：场景。
场景笔记	属性"对话框， 场景"选项卡：（注记）。
场景类型	'属性'对话框，'场景'选项卡：类型。

标记值替换宏

标记值宏是一种特殊形式的字段替换宏，它提供对元素标签和相应标记值的访问。它们可以通过以下两种方式之一使用：

- 直接替代
- 条件替换

直接替代

这种形式的宏直接将命名标签的值代入输出。

结构：%<macroName>:"<tagName>"%

<macroName> 可以是以下之一：

- 属性标签
- 类标签
- connectorDestElemTag
- connectorDestTag
- connectorSourceElemTag
- 连接器源标签
- 连接器标签
- 链接AttTag
- 链接标签
- opTag
- 包装标签
- 参数标签

这对应于属性、类、操作、包、参数、两端的连接器、连接器两端的元素和包括属性端的连接器的标签。

<tagName> 是表示特定标签名称的string 。

示例

%opTag:"属性"%

条件替换

这种形式的宏模拟了为字段替换宏定义的条件替换。

结构：%<macroName>:"<tagName>" (== "<test>") ? <subTrue> (: <subFalse>) %

注记：

- <macroName> 和 <tagName> 在这里定义
- (<text>) 表示 <text> 是可选的
- <test> 是一个string ，表示宏的可能值
- <subTrue> 和 <subFalse> 可以是带引号的字符串和关键字值的组合；在使用该值的地方，它将被输出中的宏值替换

例子

%opTag:"opInline" ？ 排队″ ： ””%
%opTag:"opInline" ？ 排队″%
%classTag:"unsafe" == "true" ？ 不安全″：“”%
%classTag:"unsafe" == "true" ？ 不安全″%
标记值宏使用与字段替换宏相同的命名约定。

模板参数替换宏

如果您想在转换模板中提供对与模型中模板捆绑连接器的绑定参数替换转换有关的数据的访问，您可以使用模板参数替换宏。宏名称采用 Camel 大小写。如果选中了复选框，则表示复选框的宏返回值 "T"；否则值为空。

模板参数替换宏

宏名称	描述
参数替换形式	模板捆绑属性"对话框， 捆绑参数"选项卡， 参数替换"面板：正式模板参数名称。
参数SubstitutionActual	模板捆绑属性"对话框， 捆绑参数"选项卡， 参数替换"面板：实际参数名称/表达式。
参数SubstitutionActualClassifier	模板捆绑属性"对话框， 捆绑参数"选项卡， 参数替换"面板：实际参数分类器。

测试字段替换宏

此表列出了每个测试字段替换宏以及结果说明。
字段替换宏根据 Camel 大小写命名。如果选中了复选框，则表示复选框的宏返回值“T”。否则值为空。

测试宏

宏名称	描述
测试验收标准	测试对话框：Acceptance Criteria。
testCheckedBy	测试案例窗口：检查者。
测试日期运行	测试案例窗口：上次运行。
测试类	测试案例窗口：测试类（定义的测试类型：Unit,集成,系统,Acceptance,Inspection,Scenario）
测试输入	测试对话框：输入。
测试名称	测试案例窗口：测试。
测试笔记	测试案例窗口：描述。
试验结果	测试对话框：结果。
testRunBy	测试案例窗口：运行By。（值来自“人物”对话框中的项目作者定义 - 设置>参考>模型类型 > 人物 > 项目作者”。）
测试状态	测试案例窗口：状态。
测试类型	测试案例窗口：类型。

函数宏

函数宏是一种操作和格式化各种元素数据项的便捷方式。每个函数宏都返回一个结果string 。使用函数宏的结果有两种主要方法：

- 将返回的string直接替换到输出中，如：%TO_LOWER(attName)%
- 将返回的string存储为变量定义的一部分，例如：$name = %TO_LOWER(attName)%

函数宏可以接受参数，这些参数可以传递给宏：

- 字符串文字，用双引号括起来
- 不带百分号的直接替换宏
- 变量引用
- 数字文字

使用逗号分隔的列表传递多个参数。

函数宏根据全大写样式命名，如下所示：

%CONVERT_SCOPE(操作范围)%

此处描述了可用的函数宏。参数用方括号表示，如下所示：

FUNCTION_NAME（[参数]）。

CONVERT_SCOPE([umlScope])

为了与支持的语言一起使用，将 [umlScope] 转换为正在生成的语言的适当范围关键字。此表显示了 [umlScope] 相对于给定语言的转换。

语	转换
C++	包==>公开 公开 ==> 公开 私人 ==> 私人 受保护 ==> 受保护
C #	包==>内部 公开 ==> 公开 私人 ==> 私人 受保护 ==> 受保护
德尔福	包==>受保护 公开 ==> 公开 私人 ==> 私人 受保护 ==> 受保护
Java	包==> {空白} 公开 ==> 公开 私人 ==> 私人 受保护 ==> 受保护
PHP	包==>公开 公开 ==> 公开

	私人 ==> 私人 受保护 ==> 受保护
VB	包==> 受保护 公开 ==> 公开 私人 ==> 私人 受保护 ==> 受保护
VB.Net	包==>朋友 公开 ==> 公开 私人 ==> 私人 受保护 ==> 受保护

COLLECTION_CLASS（[语言]）

为当前链接属性指定的语言提供适当的集合类。

CSTYLE_COMMENT([wrap_length])

使用 /* 和 */ 将当前范围内的元素的注记转换为纯 C 样式的注释。

DELPHI_PROPERTIES([范围], [分隔符], [缩进])

生成一个 Delphi属性。

DELPHI_COMMENT([wrap_length])

将当前范围内的元素的注记转换为 Delphi 注释。

EXEC_ADD_IN(, [function_name],, ...,)

调用Enterprise Architect插件
函数，它可以返回一个结果string 。
[插件
] 和 [function_name] 指定插件的名称插件
和函数被调用。
插件
的参数插件
函数可以通过参数 [prm_1] 到 [prm_n] 指定。
$result = %EXEC_ADD_IN("MyAddin", "ProcessOperation", classGUID, opGUID)%

由 EXEC_ADD_IN 宏调用的任何函数都必须有两个参数：一个EA.Repository object和一个包含来自 EXEC_ADD_IN 调用的任何附加参数的 Variant 数组。返回类型应该是 Variant。

Public函数ProcessOperation(存储库As EA.Repository , args As Variant) As Variant

查找（[src]，[子字符串]）

[subString] 第一个实例在 [src] 中的位置； -1 如果没有。

GET_ALIGNMENT()

返回一个string ，其中当前输出行上的所有文本都转换为空格和制表符。

JAVADOC_COMMENT([wrap_length])

将当前范围内的元素的注记转换为 javadoc 样式的注释。

左（[src]，[count]）

[src] 的前 [count] 个字符。

长度（[src]）

[src] 的长度。返回一个string 。

MATH_ADD(x,y) MATH_MULT(x,y) 和 MATH_SUB(x,y)

在代码模板或 DDL模板中，这三个宏分别执行以下数学函数：

- 加法 (x+y)
- 乘法 (x*y) 和
- 减法 (xy)

参数 x 和 y 可以是整数或变量，或两者的组合。考虑这些示例，用于 C++ 代码生成的 类”模板：

- $a = %MATH_ADD(3,4)%
- $b = %MATH_SUB(10,3)%
- $c = %MATH_MULT(2,3)%
- $d = %MATH_ADD($a,$b)%
- $e = %MATH_SUB($b,$c)%
- $f = %MATH_MULT($a,$b)%
- $g = %MATH_MULT($a,10)%
- $h = %MATH_MULT(10,$b)%

这些以相同的序列计算：

- a = 3 + 4 = $a
- b = 10 - 3 = $b
- c = 2 * 3 = $c
- d = a + b = $d
- e = b - c = $e
- f = a * b = $f
- g = a * 10 = $g
- h = 10 * b = $h

生成代码时，.h 文件（对于 C++）包含以下相应的字符串：

- a = 3 + 4 = 7
- b = 10 - 3 = 7
- c = 2 * 3 = 6
- d = a + b = 14
- e = b - c = 1
- f = a * b = 49
- g = a * 10 = 70
- h = 10 * b = 70

MID([src], [start]) MID([src], [start], [count])

[src] 的子字符串，从 [start] 开始，包括 [count] 个字符。其中 [count] 被省略， string的其余部分被包括在内。

PI（[选项]，[值]，{[选项]，[值]}）

将当前模板的 PI 设置为 [value]。 [value] 的有效值为：

- "\n"
- "\t"
- “”
- “”

<option> 控制新 PI 何时生效。 <option> 的有效值为：

- I , Immediate: 在下一个非空模板行之前生成新的 PI
- N , Next: 在下一个非空模板行之后生成新的 PI

一次调用中允许多对选项。使用 this 的一个例子是一个关键字总是在一个新的行上，如下所示：

%PI=""%
%类抽象？ 抽象的”%
%if classTag:"宏" != ""%
%PI(" I "、"\n"、" N "、"")%
%classTag:"宏"%
%万一%
班级
%班级名称%

有关更多详细信息，请参阅处理指令 *(PI)* 宏。

PROCESS_END_OBJECT（[模板名称]）

使与基类相距较远的一个类可以转化为基类的对象（如属性、操作、包、参数、列等）。 [template_name] 是指临时存储数据的工作模板。

REMOVE_DUPLICATES([源], [分隔符])

其中[源]是一个[分隔符]分隔列表；这将删除任何重复或空字符串。

替换（[string]，[旧]，[新]）

将给定string <string> 中所有出现的 [old] 替换为 [new]。

RESOLVE_OP_NAME()

解决两个方法源接口具有相同名称的接口名称冲突。

RESOLVE_QUALIFIED_TYPE() RESOLVE_QUALIFIED_TYPE([分隔符]) RESOLVE_QUALIFIED_TYPE([分隔符], [默认])

为当前属性、链接属性、链接父项、操作或参数生成限定类型。启用除以外的分隔符的规范。以及需要某个值时的默认值。

RIGHT([src], [count])

[src] 的最后 [count] 个字符。

TO_LOWER([string])

将 [string] 转换为小写。

TO_UPPER([string])

将 [string] 转换为大写。

TRIM([string]) TRIM([string], [trimChars])

从 [string] 中删除尾随和前导空格。如果指定了 [trimChars]，则删除 <trimChars> 集中的所有前导和尾随字符。

TRIM_LEFT([string]) TRIM_LEFT([string], [trimChars])

从 <string> 中删除指定的前导字符。

TRIM_RIGHT([string]) TRIM_RIGHT([string], [trimChars])

从 <string> 中删除指定的尾随字符。

VB_COMMENT([wrap_length])

将当前范围内的元素的注记转换为 Visual Basic 样式注释。

WRAP_COMMENT([comment], [wrap_length], [indent], [start_string])

以宽度 [wrap_length] 将文本 [comment] 换行，将 [indent] 和 [start_string] 放在每行的开头。

$behavior = %WRAP_COMMENT(opBehavior, "40", " ", "//")%

<wrap_length> 仍然必须作为string传递，即使 WRAP_COMMENT 将此参数视为整数。

WRAP_LINES([文本], [wrap_length], [start_string] {, [end_string] })

将 [text] 包装为指定为 [wrap_length]，将 [start_string] 添加到每行的开头，并将 [end_string] 添加到行尾（如果已指定）。

XML_COMMENT([wrap_length])

将当前范围内的元素的注记转换为 XML 样式的注释。

控件宏

控件macros用于控制模板的处理和格式化。控制宏的基本类型包括：

- 列表宏，用于生成多个元素特征，例如属性和操作
- 分支宏，形成 if-then-else 结构以有条件地执行部分模板
- 用于在输出中格式化新行的 PI 宏，从下一个非空行开始生效
- A PI函数宏，可以将 PI 设置为变量并添加设置在下一行之前生成的 PI 的功能
- 同步宏

在一般情况下，控制宏是根据 Camel 大小写命名的。

列表宏

如果您需要循环或迭代包含在当前object ，您可以使用*%list*宏来执行此操作。该宏对当前模板范围内的所有对象执行迭代传递，并调用另一个模板来处理每个对象。

基本结构是：

%list=<TemplateName> @separator=<string> @indent=<string> (<conditions>) %

其中 <string> 是双引号文字string ， <TemplateName> 可以是以下模板名称之一：

- 属性
- 属性实现
- 类
- 类库
- ClassImpl
- 类初始化器
- 类接口
- 约束
- 自定义模板（自定义模板使您能够定义自己的模板）
- 工作量
- 内部类
- 内部类实现
- 链接文件
- 公制
- 命名空间
- 手术
- OperationImpl
- 参数
- 问题
- 需求
- 资源
- 风险
- 设想
- 测试

<conditions> 是可选的，看起来与 'if' 和 'elseIf' 语句的条件相同。

示例

在类转换中，类可能包含多个属性；此示例调用属性转换并输出作用域内类的每个属性的转换处理结果。结果列表用一个新行分隔其项目，并将它们分别缩进两个空格。如果作用域中的类有任何原型属性，它们将使用适当的专用模板生成。

%list="属性" @separator="\n" @indent=" "%

由@separator 表示的分隔符属性指定列表项之间应使用的空间，不包括列表中的最后一项。

由@indent 表示的缩进属性指定生成的输出中的每一行应该缩进的空间。

特别案例

使用 %list 宏时需要考虑一些特殊情况：

- 如果 Attribute模板用作 %list 宏的参数，这也会通过执行适当的 LinkedAttribute模板生成从关联派生的属性
- 如果 ClassBase模板用作 %list 宏的参数，这也会通过执行适当的类模板生成从模型中的链接派生的类库
- 如果 ClassInterface模板用作 %list 宏的参数，这也会通过执行适当的类模板生成从模型中的链接派生的类库
- 如果 InnerClass 或 InnerClassImpl 用作 %list 宏的参数，则这些类分别使用类和 ClassImpl 模板生成；这些参数指示模板应基于范围内的类的内部类进行处理

分支宏

分支宏提供 if-then-else 结构。 CTF 通过这些宏支持有限形式的分支：

- 如果
- 否则如果
- else
- 万一
- endTemplate（退出当前模板）

if 和 elseIf 宏的基本结构是：

%if <测试> <操作员> <测试>%

其中 <operator> 可以是以下之一：

- ==
- !=
- <（数学比较，小于）
- >（数学比较，大于）
- <=（数学比较，小于等于）
- >=（数学比较，大于等于）

<test> 可以是以下之一：

- 一个string文字，用双引号括起来
- 直接替换宏，不带百分号
- 变量引用

注记，如果您使用其中一种数学比较运算符，<test> 必须是string格式的十进制数。

可以嵌套分支，并且可以使用以下之一指定多个条件：

- 和，或
- 或者

指定多个条件时，'and'和'or'的优先顺序相同，从左到右处理条件。

如果字符串的条件语句区分大小写，则 "字符串"不等于 "A STRING"。因此，在某些情况下，最好设置变量 $str=TO_LOWER(variable) 或 TO_UPPER(variable)，然后与特定情况进行比较。

条件语句不支持宏。最好将宏（ string ）的结果赋给一个变量，然后在比较中使用该变量。

$fldType = % TO_LOWER ($parameter1)%

$COMMENT = "使用日期和时间字段类型的前 4 个字符"

$fldType4 = % 左 ($fldType, 4)%

%if $fldType4 == "日期"%

约会时间

％万一％

这需要一个值为 "Datetime"、 "DATETIME"或 "Date"的参数，并返回 "Datetime"。

endif 或 endTemplate 宏必须用于表示分支的结束。此外，如果相应的分支正在执行，endTemplate 宏会使模板立即返回。

示例1

%if elemType == "接口"%

;
%别的%
%操作体%
%万一%
在这种情况下：

- 如果接口为 接口″，则返回分号
- 如果体不是 接口″，则调用一个名为 Operation body 的模板

示例2

$bases="ClassBase"
$interfaces=""%
%if $bases !="" 和 $interfaces !=""%
: $bases, $interfaces
%elseIf $bases !=""%
: $基础
%elseIf $interfaces !=""%
: $接口
%万一%
在这种情况下，返回的文本是 ':ClassBase'。

使用布尔值的条件

当使用涉及系统复选框（布尔字段）的条件设置分支时，例如 Attribute.Static (attStatic)，条件语句将写为：
%if attStatic == " T "%
例如：
% 如果 attCollection == " T " 或 attOrderedMultiplicity == " T " %
% 结束模板 %

同步宏

同步宏用于在前向同步期间将新部分插入源代码时为Enterprise Architect提供格式提示。同步宏的值必须在文件模板中设置。

设置同步宏的结构是：

%<名称>=<值>%

其中 <name> 可以是此处列出的宏之一，而 <value> 是用双引号括起来的文字string 。

同步宏

宏名称	描述
synchNewClassNotesSpace	附加到新的类注记的空间。默认值：\n。
synchNewAttributeNotesSpace	附加到新属性注记的空格。默认值：\n。
synchNewOperationNotesSpace	附加到新操作注记的空间。默认值：\n。
synchNewOperationBodySpace	附加到新操作体的空间。默认值：\n。
synchNamespaceBodyIndent	缩进应用于非全局命名空间中的类。默认值：\t。

处理指令 (PI) 宏

PI（处理指令）宏提供了一种方法来定义要在使用模板生成的代码段（代表实体）之间插入的分隔符文本。

设置处理指令的结构是：

%PI=<值>%

在此结构中，<value> 是用双引号括起来的文字string ，具有以下选项：

- "\n" - 换行（默认）
- “ “ - 空间
- "\t" - 制表符
- “““ - 无效的

默认情况下，PI 设置为为每个非空替换生成一个新行 (\n)，可以通过重置 PI 宏来更改该行为。例如，简单 VB 代码中的类的属性声明将生成为单行语句（没有新行）。这些属性来源于模型中的Class-Attribute属性来生成，例如：

Private Const PrintFormat As字符串= "Portrait"

生成此模板的模板以将 PI 设置为空格而不是新行开始：

% PI = " " %

% CONVERT_SCOPE (attScope)%

% 万一 %

% 如果 attConst == " T " %

常量

% 万一 %

在转换它时，attscope 返回 VB 关键字 “Private”，而 attConst 在由单个空格隔开的同一行上返回 “Const”（适合前面的 VB类。属性定义示例）。

或者，在生成类时，您可能需要类声明，注记和类正文都用双线分隔。在这种情况下，%PI 设置为 '/n/n' 以返回双倍行距：

% PI = "\n\n" %

% 类声明 %

% 课堂笔记 %

% 类体 %

PI 特性

- 空行对输出没有影响
- 任何具有产生空结果的宏的行都不会产生 PI 分隔符（空格/新行）
- 最后一个条目不返回 PI；例如，%Classbody% 没有在正文后添加双线

用于可执行状态机的代码生成宏

此处列出的模板可通过代码模板编辑器获得（ 开发 >源代码 > 选项 > 编辑代码模板"功能区选项）；在 语言"字段中选择 "STM_C++_Structured"。

模板的结构如下所示：

StmContextStateMachineEnum

StmStateMachineEnum

StmContextStateEnum

StmAllStateEnum

StmContextTransitionEnum

StmTransitionEnum

StmContextEntryEnum

StmAllEntryEnum

StmContextStateMachineStringToEnum

StmStateMachineStringToEnum

StmContextStateEnumToString

StmStateEnumToString

StmContextTransitionEnumToString

StmTransitionEnumToString

StmContextStateNameToGuid

StmStateNameToGuid

StmContextTransitionNameToGuid

StmTransitionNameToGuid

StmContextDefinition

StmStateMachineEnum

StmAllStateEnum

StmTransitionEnum

StmAllEntryEnum

StmAllRegionVariableInitialize

StmStateWithDeferredEvent

StmDeferredEvent

StmTransitionProcMapping

StmTransitionProc
StmTransitionExit
StmTransitionEntry
StmTargetOutgoingTransition
StmTargetParentSubmachineState
StmStateProcMapping
状态程序
状态条目
StmOutgoingTransition
StmConnectionPointReferenceEntry
StmParameterizedInitial
StmSubMachineInitial
StmRegionInitial
StmRegionDeactive
StmStateExitProc
状态转换
状态事件
StmStateTriggeredTransition
StmStateCompletionTransition
StmStateIncomingTransition
StmStateOutgoingTransition
StmSubmachineStateExitEvent
StmVertexOutgoingTransition
StmConnectionPointReferenceExitEvent
StmStateExitEvent
StmVertexOutgoingTransition
StmAllRegionVariable
StmStateMachineStringToEnum
StmStateMachineRun
StmStateInitialData
StmStateMachineEntry
StmOutgoingTransition
StmStateMachineRunInitial
StmStateMachineInitial
StmStateMachineRuns

StmContextManager

StmSimulationManager
StmContextInstanceDeclaration
StmContextInstance
StmContextVariableRunstate

StmContextInstanceAssociation
StmContextInstanceClear

StmEventProxy
StmSignalEnum
StmContextJoinEventEnum
StmJoinEventEnum
StmEventEnum
信号定义
StmSignalAttributeAssignment
StmSignalAttribute
StmSignalInitialize
StmEventStringToEnum
StmEventEnumToString
StmEventNameToGuid

控制台管理器
StmContextInstanceDeclaration
StmContextInstance
StmContextVariableRunstate
StmContextInstanceAssociation
StmContextInstanceClear

StmStateMachineStrongToEnum

StmInitialForTransition

StmVertextOutgoingTransition

发送事件

Stm广播事件

StmContextRef

信号与事件

宏名称	描述
stmEventEnum	事件名称，前缀为“ENUM_”，全部大写。
StmEventGuid	事件的GUID 。

stmEventName	删除了空格和星号的事件名称。
stmEvent变量	小写前缀 "m_"的事件名称。
stmIsSignalEvent	如果元素是 SignalEvent，则为 "T"。
stmSignalEnum	带有前缀 "信号"的信号名称，全部大写。
stmSignalFirstEvent	事件名称，前缀为 "ENUM_"，全部大写。
stmSignalGuid	信号的GUID 。
stmSignalName	已删除空格和星号的信号名称。
stm信号变量	带有前缀 "m_"的 写信号的名称。
stmTriggerName	转移 属性：简单的触发器。
stmTriggerSpecification	转移 属性：简单的触发器。
stmTriggerType	转移 属性：简单的触发器。

语境

宏名称	描述
stmContextName	删除了空格和星号的类的名称。
stmContextQualName	为其生成代码的类的限定名称。
stmContextVariableName	
stmContextFileName	正在为其生成代码的类的输出文件名。

写物件到状态机初始化

宏名称	描述
stmContextVariableRunstateName	
stmContextVariableRunstat	

eValue	
stmContextHasStatemachine	如果当前上下文有一个或多个状态机，则为' T '。
stmHasHistoryPattern	如果状态机具有历史模式，则为 "T"。
stmHasTerminate模式	如果状态机有终止模式，则为' T '。
stmHasDeferredEventPattern	如果状态机具有延迟事件模式，则为 "T"。
stmHasSubmachine模式	如果状态机具有子机模式，则为 "T"。
stmHasOrthogonalPattern	如果状态机具有正交模式，则为 "T"。

状态机

宏名称	描述
stmStatemachineName	删除了星号和空格的状态机名称。
stmStatemachineEnum	状态机名称加 "ENUM_"加状态机名称大写。
stmStatemachineGuid	状态机元素的GUID 。
stmStateCount	状态机中状态元素的数量。
stmSubmachineInitialCount	子机状态元素中初始元素的数量。
stmStatemachineHasSubmachineState	如果状态机至少有一个状态，则为 "T"。
stmStatemachineInitialCount	状态机中 Initial 元素的数量。

区域

宏名称	描述
stmRegionEnum	状态名称加 "状态"加状态区域名称区域。
stmRegionFQName	状态区域的完全限定名称。
stmRegionName	删除了空格和星号的状态区域的名称。

stm区域变量	状态区域的名称，前缀为 “m_”，小写。
stmRegionFQ变量	状态区域的完全限定名称，前缀为 “m_”，小写。
stmRegionGuid	区域的GUID 。
stmRegionInitial	
stmRegionIsOwnedBySt ate Machine	如果区域为状态机所有，则为 “T ”。

转移

宏名称	描述
stmTransitionEnum	转移 的名字转移 带有前缀 “ENUM_”，加上转移 的名称转移 大写。
stmTransitionGuid	转移 的GUID转移 .
stmTransitionName	转移 的名字转移 删除了空格和星号。
stmTransitionSourceGuid	转移 中源元素的GUID转移 .
stmTransitionTargetGuid	转移 中目标元素的GUID转移 .
stmTransitionVariable	转移 的名字转移 带有前缀 “m_”的 写字母。
stmTransitionSourceVariab le	

stmTransitionTargetVariable	
stmTransitionFQ变量	
stmSourceVertexEnum	转移 的名字转移 's源顶点加上'_ENUM'加上转移 的名字转移 's源顶点大写。
stmTargetVertexEnum	转移 的名字转移 的目标顶点加上'_ENUM'加上转移 的名称转移 的目标顶点大写。
stmSourceIsInitial	如果转移 是' T '转移 's源是首字母。
stmSourceIsState	如果转移 是' T '转移 的源是一种状态。
stmSourceIsEntryPoint	如果转移 是' T '转移 的源是一个入口。
stmSourceIsExitPoint	如果转移 是' T '转移 '源是一个出口。
stmSourceIsFork	如果转移 是' T '转移 's源是一个分叉。
stmSourceIsJoin	如果转移 是' T '转移 '源是一个汇合元素。
stmTargetIsFinalState	如果转移 是' T '转移 的目标是终点状态元素。
stmTargetIsExitPoint	如果转移 是' T '转移 的目标是一个出口元素。

stmTargetIsState	如果转移 是' T '转移 的目标是状态元素。
stmTargetIsChoice	如果转移 是' T '转移 的目标是选择元素。
stmTargetIsJunction	如果转移 是' T '转移 的目标是一个结合点元素。
stmTargetIsEntryPoint	如果转移 是' T '转移 的目标是一个入口元素。
stmTargetIsConnectionPointReference	如果转移 是' T '转移 的目标是一个连接点参考元素。
stmTargetIsFork	如果转移 是' T '转移 的目标是一个分叉元素。
stmTargetIsJoin	如果转移 是' T '转移 的目标是汇合元素。
stmTransitionEffect	影响的转移 .
stmTransitionGuard	转移 的守卫条件转移 .
stmTransitionKind	转移 的类型或种类转移 .
stmTargetInitialTransition	
stmTargetIsSubmachineState	如果转移 是' T '转移 的目标是一个子机状态。
stmSourceStateEnum	转移 的名字转移

	's源状态，前缀'状态'大写。
stmTargetStateEnum	转移 的名字转移 的目标状态，前缀'状态'大写。
stmTargetVertexFQName	转移 的完全限定名称转移 的目标顶点。
stmTargetIsDeepHistory	如果转移 是' T '转移 的目标是深度历史状态。
stmTargetIsShallowHistory	如果转移 是' T '转移 的目标是一个浅历史状态。
stmTargetIsTerminate	如果转移 是' T '转移 的目标是元素终止
stmParentIsStateMachine	如果顶点是入口或出口，或者容器是状态机，则为' T '。
stmSourceParentStateEnum	
stmTargetParentStateEnum	
stmTargetSubmachineEnum	
stmTargetRegionIndex	
stmIsSelfTransition	如果转移 是' T '转移 '源与它的目标相同。
stmHistoryOwningRegionInitialTransition	
stmDefaultHistoryTransition	

顶点和状态

宏名称	描述

stmVertexName	顶点的名称。
状态名称	状态的状态。
stmVertexGuid	顶点的GUID 。
stmVertexFQName	顶点的完全限定名称。
stmStateFQName	状态的完全限定状态。
stmVertexType	顶点的类型； 状态″、 FinalState″、 伪状态″、 ConnectionPointReference″或 ″之一（空）。
stmPseudostateKind	伪状态的种类；“initial″、 deepHistory″、 shallowHistory″、 join″、 fork″、 “junction″、 choice″、 entryPoint″、 exitPoint″或 terminate″之一。
stmPseudostateName	伪状态的名称。
stm伪状态变量	带有前缀 m_″的伪状态的名称，小写。
stmPseudostateStateMachineName	伪状态机的名称。
stmPseudostateStateMachineVariable	小写前缀 m_″的伪状态机的名称。
stmVertexVariable	带有前缀 m_″的顶点的名称，小写。
stmVertexEnum	顶点名称加上 _ENUM″加上大写的顶点名称。
stmStateEnum	状态名称加 状态″加状态名称大写。
stmConnectionPointReferenceStateName	连接点的名称参考。
stmConnectionPointReferenceStateVariable	连接点的名称参考带有前缀 m_″的 写字母。
stmConnectionPointReferenceEntryCount	
stmParameterizedInitialCount	
stmInitialCountForTransition	
状态变量	状态名称，前缀为 m_”，小写。
stmStateEntryBehavior	为状态的 入口″行动操作定义的状态（元素特征窗口上 入口″行动操作的 行为″选项卡上的文本）。

stmStateEntryCode	为状态的 进入”行动操作定义的初始代码（行为的 代码”选项卡上的 进入”行动操作的文本）。
stmStateDoBehavior	为状态的 do”行动定义的行为（元素的特征窗口上 do”行动操作的 行为”选项卡上的文本）。
stmStateDoCode	为状态的 do”行动操作定义的初始代码（行为的 代码”选项卡上的 do”行动操作的文本）。
stmStateExitBehavior	为状态的 退出”行动操作定义的状态（元素特征窗口上 退出”行动操作的 行为”选项卡上的文本）。
stmStateExitCode	为状态的 退出”行动操作定义的初始代码（行为的 代码”选项卡上的 退出”行动操作的文本）。
stmStateSubmachineName	子机之名。
stmStateSubmachine变量	子机名称，前缀为 m_”，小写。
stmStateIsFinal	如果状态是 FinalState，则为 ' T '。
stmStateIsSubmachineState	如果状态是子机状态（'属性'页面|高级|'状态'属性），则为' T '。
stmSubMachineEnum	子机名称后跟“_ENUM”加子机名称大写。
stmStateHasChildrenToJoin	
stmStateIsTransitionTarget	
stmThisIsSource	
stmThisIsSourceState	
stmStateParentIsSubmachine	如果状态的容器是状态机，则为 “T ”。
stmStateContainerMatchTransitionContainer	
stmVertexRegionIndex	
stmStateRegionCount	状态中的区域数。
stmStateInitialCount	状态机中 Initial 元素的数量。
stmVertexContainerVariable	
stmVertexParentEnum	
stmStateHasUnGuardedCompletionTransition	

stmStateEventHasUnGuardedTransition	
stmInitialTransition	

实例关联

宏名称	描述
stmSourceInstanceName	
stmTargetInstanceName	
stmSourceRoleName	
stmTargetRoleName	

EASL 代码生成宏

Enterprise Architect提供了许多Enterprise Architect仿真库 (EASL) 代码生成宏来从行为模型生成代码。这些是：

- EASL_INIT
- EASL_GET
- EASLList 和
- EASL_END

EASL_INIT

EASL_INIT 宏用于初始化 EASL 行为模型。行为模型代码的生成依赖于该模型。

方面	描述
句法	%EASL_INIT(<<GUID>>)% 在哪里： • <<GUID>>是行为模型所有者的物件（通常是类元素）的GUID

EASL_GET

EASL_GET 宏用于检索 EASL object的属性或集合。 EASL 对象以及每个object的属性和集合在*EASL* 集合和*EASL*属性主题中标识。

方面	描述
句法	$result = %EASL_GET(<<Property>>, <<拥有着ID>>, <<Name>>)% 在哪里： • <<Property>> 是 "属性"、 "Collection"、 "At"、 "Count"或 "IndexOf"之一 • <<OwnerID>> 是要为其检索属性/集合的所有者object的 ID • <<Name>> 是正在访问的属性或集合的名称 • $result 是返回值；如果不是有效属性，则为 "" 如果 <<属性>> 是： • "At"，则 <<OwnerID>> 是集合的 ID，<<Name>> 是要检索项目的集合的索引 • "Count"，则<<拥有着ID>>为集合的ID，不使用<<Name>>；它将检索集合中的项目编号 • "IndexOf"，则<<拥有着ID>>是集合的ID，<<Name>>是集合中的项的ID；它将检索集合中项目的索引（ string格式）
示例	$sPropName = %EASL_GET("属性", $context, "名称")%

EASL列表

EASLList 宏用于使用适当的模板呈现 EASL 集合中的每个object 。

方面	描述
句法	$result = %EASLList=<<模板名称>> @separator=<<分隔符>> @indent=<<缩进>> @owner=<<OwnedID>> @collection=<<CollectionName>> @option1=<<OPTION1>> @option2=<<OPTION2>>.....@optionN=<<OPTIONN>>% 在哪里： • <<TemplateName>> 是任何行为模型模板或自定义模板的名称 • <<Separator>> 是列表分隔符（如 \"n"） • <<indent>> 是应用于结果的任何缩进 • <<OwnedID>> 是包含所需集合的object的 ID • <<CollectionName>> 是所需集合的名称 • <<OPTION1>...<<OPTION99>> 是可能在模板上传递的杂项选项；每个选项都作为模板的附加输入参数给出 • $result 是结果值；如果不是有效集合，这是 ""
示例	$sStates = %EASLList="状态" @separator="\n" @indent="\t" @owner=$StateMachineGUID @collection="States" @option=$sOption%

EASL_END

EASL_END 宏用于发布 EASL 行为模型。

方面	描述
句法	%EASL_END%

行为模型模板

- 行动
- 行动分配
- 行动中断
- 行动调用
- 行动创造
- 行动毁灭
- 行动如果
- 行动循环
- 行动不透明
- 行动平行
- 行动提升事件
- 行动引发异常
- 行动开关

- 行为
- 行为体
- 行为声明
- 行为参数
- 调用参数值
- 决策行动
- 决策条件
- 决策逻辑
- 决策表
- 守卫条件
- 属性声明
- 属性注记
- 属性物件
- 状态
- 状态回调
- 状态
- 状态EnumeratedName
- 状态机
- 状态机HistoryVar
- 转移
- 转移
  影响
- 触发器

EASL 系列

本主题列出了每个 EASL 对象的 EASL 集合，由EASL Code Generation Macros代码生成宏检索。

行动

收藏名称	描述
论据	行动的论据。
子动作	行动的子行动。

行为

收藏名称	描述
行动	行为的行动。
节点	行为的节点。
参数	行为的参数。
变量	行为的变量。

分类器

收藏名称	描述
所有状态机	分类器的所有状态机。
异步属性	分类器的异步属性。
异步触发器	分类器的异步触发器。
行为	分类器的行为。
属性	分类器的属性。
定时属性	分类器的定时属性。
定时触发器	分类器的定时触发器。
触发器	分类器的所有触发器。

构建

收藏名称	描述
所有儿童	构建的孩子。
客户端依赖	客户依赖于构建。
立体类型	商业的刻板构建。
供应商依赖	供应商依赖于构建。

节点

收藏名称	描述
传入边缘	节点的传入边。
传出边缘	节点的出边。
子节点	节点的子节点。

状态

收藏名称	描述
行为	状态的 Do 行为。
进入行为	状态的进入行为。
退出行为	状态的退出行为。

状态机

收藏名称	描述
所有最终状态	状态机的最终状态。

全州	状态机内的所有状态，包括子机内的状态。
派生转换	状态机的派生转换以及相关的有效效果。
状态	状态机内的状态。
过渡	状态机内的转换。
顶点	状态机的顶点。

转移

收藏名称	描述
效果	转移 的影响。
警卫	转移 的守卫。
触发器	转移 的触发器。

触发器

收藏名称	描述
触发转换	与简单相关的触发器转换。

顶点

收藏名称	描述
DerivedOutgoingTransitions	遍历伪节点后顶点派生的传出转换。
传入转换	顶点的传入转换。
传出转换	顶点的传出转换。

EASL属性

本主题列出了每个 EASL 对象的 EASL属性，由EASL Code Generation Macros代码生成宏检索。

行动

属性名称	描述
行为	行动的相关行为（调用行为行动或调用行动行动）。
体	行动的身体。
语境	行动的上下文。
守卫条件	行动的守卫。
IsFinal	检查A动作是否为最终行动。
被保护	检查A动作是否为受保护的行动。
首字母	检查A动作是否为初始行动。
种类	行动的那种。
下一个	行动的下一步行动。
节点	图中行动的关联节点。

参数值

属性名称	描述
参数	参数值关联参数的 ID。
价值	参数的默认值。

行为

属性名称	描述
初始动作	行为的初始动作。
是只读的	行为的行为。

isSingleExecution	行为的行为。
种类	那种行为。
返回类型	行为的返回类型。
规格	行为规范。

呼叫事件

属性名称	描述
手术	CallEvent 的操作。

变更事件

属性名称	描述
变化表达	ChangeEvent 的变化表达式。

分类器

属性名称	描述
有行为	检查分类器A具有行为模型（活动和交互）。
语	分类器的语言。
状态机	分类器的状态机。

条件

属性名称	描述
表达	条件的表达。
降低	条件的较低值。

上	条件的上限值。

构建

属性名称	描述
获取标记值	属性的标记值。
应用了刻板印象	检查特定构造A是否应用于属性。
注记	属性的注记。
UML 类型	属性的UML类型。
能见度	属性的可见性。

边缘

属性名称	描述
从	Edge 产生的节点的 ID。
至	Edge 所针对的节点的 ID。

事件对象

属性名称	描述
事件种类	事件物件的事件物件。

实例

属性名称	描述
分类器	实例的分类器。
价值	实例的值。

参数

属性名称	描述
方向	参数的方向。
类型	参数的类型。
价值	参数的值。

原始

属性名称	描述
全名	原始的 FQ 名称。
ID	原始的ID。
名称	原始的名字。
ObjectType	原始的object类型。
家长	原始的原始。

属性对象

属性名称	描述
边界大小	PropertyObject 的绑定大小（如果它是一个集合）。
分类器StereoType	PropertyObject 的分类器的构造型。
IsAsynchProp	检查 PropertyObject 是否A异步属性。
集合	检查 PropertyObject 是否为A 。
有序	检查 PropertyObject 是否是有序A （如果它是一个集合）。
IsTimedProp	检查 PropertyObject 是否A定时属性。
种类	PropertyObject 的种类。
低值	PropertyObject 的较低值（如果它是一个集合）。

类型	PropertyObject 的类型。
上限值	PropertyObject 的上限值（如果它是一个集合）。
价值	PropertyObject 的值。

信号事件

属性名称	描述
信号	SignalEvent 的信号。

状态

属性名称	描述
有子机	检查状态是否为A子机状态。
是最终状态	检查状态是否A最终状态。
子机	获取状态包含的子机ID（如果有）。

状态机

属性名称	描述
HasSubMachineState	检查状态机是否A子机状态。
初始状态	状态机的初始状态。
子机状态	状态机的子机状态。

时间事件

属性名称	描述
什么时候	TimeEvent 的 "when" 属性。

转移

属性名称	描述
有效果	检查过渡是否有效A
是派生的	检查转换是否A派生转换。
是超越	检查转换是否从A状态机（子机状态）超越到另一个状态机。
已触发	检查是否触发A转换。
源	转移 的源。
目标	转移 的目标。

触发器

属性名称	描述
异步目的地状态	简单的触发器目标状态（如果是异步触发器）。
依赖属性	与简单关联的属性的触发器。
事件	触发器的事件。
名称	触发器的名字。
类型	触发器的类型。

顶点

属性名称	描述
是历史	检查顶点是否为A状态。
是伪状态	检查顶点是否为A状态。

伪状态类	Vertex 的伪状态类型。

模板模板调用

使用带参数的函数调用，您可以从其他模板调用模板，无论是标准模板还是项目中创建的用户定义模板。此外，被调用的模板可以返回一个值，并且可以递归调用。

例子

将参数返回给变量A调用语句：

$sSource = %StateEnumeratedName($Source)%

对具有参数的模板A调用语句：

%RuleTask($GUID, $index)%

使用被调用模板中的 $parameter 语句：

$GUID = $parameter1

$index = $parameter2

模板支持递归调用，比如这个对模板RuleTask的递归调用：

$GUID = $parameter1

$index = $parameter2

% PI = "" %

$nul = "初始化条件和动作object "

$count = %BR_GET("RuletCount")%

% 如果 $count == "" 或 $count == $index %

%ComputeRule($GUID)%

\n

% 结束模板 %

%规则（$索引）%

\n

$index = %MATH_ADD($index, " 1 ")%

%RuleTask($GUID, $index)%

MDG开发中的代码模板编辑器

这些主题描述了如何使用代码模板编辑器窗口来创建自定义模板：

- Create Custom Templates
- Customize Base Templates
- Add New Stereotyped Templates

代码编辑器提供公共代码编辑器的功能，包括用于代码生成模板的智能感知代码模板功能。有关智能感知代码和公共代码编辑器的详细信息，请参阅编辑源代码主题

创建自定义模板

Enterprise Architect提供了广泛的模板来定义如何生成代码元素。如果这些不足以满足您的目的——例如，如果您想以Enterprise Architect当前不支持的语言生成代码——您可以创建全新的自定义模板。您还可以将构造型覆盖添加到自定义模板中；例如，您可以在方法注记中列出所有参数及其注记记。

访问

功能区	开发 >源代码 > 选项 > 编辑代码模板 设计>包>变换>变换模板
键盘快捷键	Ctrl+Shift+P（代码生成模板） Ctrl+Alt+H（MDA 转换模板）

使用代码模板编辑器创建自定义模板

节	描述
1	在 语言”字段中，单击下拉箭头并选择适当的编程语言。
2	点击加新自定义模板按钮。 将显示 创建新的自定义模板”对话框。
3	在 模板类型”字段中，单击下拉箭头并选择适当的建模object 。 '<None>' 选项需要特殊处理；它启用了函数宏的定义，该宏实际上并不适用于任何类型，但必须作为函数调用来为传入的每个值定义变量 $parameter1 、$parameter2 等。
4	在 模板名称”字段中，输入适当的名称。 点击确定按钮。
5	在 代码模板编辑器”选项卡上，新模板包含在 模板”列表中， 已修改”字段中的值为 是”。 模板称为 <模板类型>__<模板名称>。 注记模板类型和模板名称之间的双下划线字符。
6	从模板列表中选择模板并编辑模板字段中的内容以满足您的要求。
7	单击保存按钮。 这存储了新模板，现在可以从模板列表中获得使用。如有必要，您还可以向模板添加构造型覆盖。

注记

- 对于自定义语言，您必须定义文件模板，以便它可以调用导入部分、命名空间和类模板，以及您认为适用的任何其他模板

自定义基础模板

Enterprise Architect提供了广泛的模板来定义如何生成代码元素。如果要更改代码元素的生成方式，可以自定义适当的现有系统提供的模板。您的更改可能会影响到模板本身的效果，或者影响到它对其他模板的调用。您还可以将原型覆盖添加到您的自定义模板中；例如，您可以在方法注记中列出所有参数及其注记记。

当您自定义系统提供的（基本）模板时，您有效地创建了优先于原始模板使用的模板副本。所有后续更改都针对该副本，并且隐藏了原始基本模板。如果您随后删除该副本，它就不能再覆盖原件，然后再次使用该原件。

访问

功能区	开发 >源代码 > 选项 > 编辑代码模板
键盘快捷键	Ctrl+Shift+P

自定义基础模板

节	描述
1	在代码模板编辑器的 语言”字段中，单击下拉箭头并选择要为其自定义基本模板的编程语言。
2	在模板列表中，单击基础模板进行编辑。
3	更新模板。
4	单击 保存”按钮以存储您的更改。
5	对您要自定义的每个相关基本模板重复步骤 2 到 4。
6	如果您愿意，可以向任何模板添加一个或多个构造型覆盖。

加新Stereotyped模板

有时，定义一个特定的代码生成模板用于给定原型的元素是很有用的。这使得可以为元素生成不同的代码，具体取决于它们的原型。 Enterprise Architect提供了一些默认模板，这些模板专门用于支持语言中的常用构造型。例如，C# 的 操作体"模板已专门用于属性型，因此它会自动生成其组成的 get"和 set"方法。您可以覆盖默认构造型模板，如覆盖默认模板主题中所述。此外，您可以为自己的构造型定义模板，如此处所述。

访问

功能区	开发 >源代码 > 选项 > 编辑代码模板
键盘快捷键	Ctrl+Shift+P

使用代码模板编辑器添加新的原型模板

节	描述
1	从语言列表中选择适当的语言。
2	从模板列表中选择一个基本模板。
3	点击 加新Stereotyped Override"按钮。 将显示 新模板覆盖"对话框。
4	选择所需的特征和/或类刻板印象。 点击确定按钮。
5	新的构造型模板覆盖显示在构造型覆盖列表中，标记为已修改。
6	在代码模板编辑器中进行所需的修改。
7	单击 保存"按钮将新的原型模板存储在项目文件中。 Enterprise Architect现在可以在为该原型的元素生成代码时使用原型模板。

注记

- 类和特征刻板印象可以结合起来为特征提供进一步的专业化水平；例如，如果当类具有原型属性时应以不同方式生成属性，则应在 新建模板覆盖"对话框中指定属性和 MyStereotype

覆盖默认模板

Enterprise Architect有一组内置或默认的代码生成模板。代码模板编辑器使您能够修改这些默认模板，从而自定义Enterprise Architect生成代码的方式。您可以选择修改任何或所有基本模板以实现所需的编码风格。

您覆盖的任何模板都存储在项目文件中。生成代码时， Enterprise Architect首先检查一个模板是否已被修改，如果是，则使用该模板。否则使用适当的默认模板。

访问

功能区	开发 >源代码 > 选项 > 编辑代码模板
键盘快捷键	Ctrl+Shift+P

参考

使用代码模板编辑器覆盖默认代码生成模板。

生成代码时， Enterprise Architect现在使用覆盖模板而不是默认模板。

字段/按钮	描述
语	从列表中选择适当的语言。
模板	从列表中选择一个基本模板。
构造型覆盖	如果基本模板具有原型覆盖，您可以从列表中选择其中之一。
<其它字段>	进行任何其他所需的修改。
节省	单击此按钮可将修改后的模板版本存储到项目文件中。该模板被标记为已修改。

语法框架

Enterprise Architect为许多流行的编程语言提供逆向工程支持。但是，如果您使用的语言不受支持，您可以使用内置的语法编辑器为其编写自己的语法。然后，您可以将语法合并到MDG 技术中，为您的目标语言提供逆向工程和代码同步支持。

编写语法并将其导入Enterprise Architect的框架是对代码模板框架的直接补充。代码模板用于将模型转换为文本形式，而将文本转换为模型则需要语法。两者都需要将更改同步到您的源文件中。

样本目录中提供了该语言的示例语言源文件和示例语法，您可以从安装目录（默认位置为C:\Program Files\Sparx Systems\EA ）访问该目录。还提供了另外两个语法文件，说明了开发语法的特定方面。

成分

部件	描述
语法句法	语法定义如何将文本分解为结构，这在将代码转换为UML表示时是必需的。在最简单的层面上，语法是用于分解输入以形成结构的指令。 Enterprise Architect使用 Backus-Naur Form (nBNF) 的变体来包含处理指令，其执行以抽象语法树(AST) 的形式从解析结果返回结构化信息，用于生成UML表示。
语法编辑器	语法编辑器是一个内置编辑器，可用于打开、编辑、验证和保存语法文件。
语法调试	您可以使用两个功能调试您创建的语法文件： ● 解析器，它为语法生成 AST ● Profiler，它还解析语法并生成 AST，但它公开了 Profiling 路径以准确显示过程的每个步骤中发生的情况

语法句法

语法定义了如何将文本分解为结构，这正是您将代码转换为UML表示时所需要的。在最简单的层面上，语法只是分解输入以形成结构的指令。 Enterprise Architect使用 Backus-Naur Form (BNF) 的一种变体来表达语法，使其能够将文本转换为UML表示。 Enterprise Architect的语法在纯 BNF 上提供的是添加处理指令，允许从解析结果以抽象语法树(AST) 的形式返回结构化信息。在 AST 完成时， Enterprise Architect将对其进行处理以生成UML模型。

句法

句法	细节
注释	注释具有与许多编程语言相同的形式。 // 可以通过添加两个 /s 来注释行尾。 /* 您可以通过添加 / 后跟 * 来注释多行。 当您添加 * 后跟 / 时，注释结束。 */
指示	说明指定语法如何工作的关键细节。它们通常包含在语法的顶部，类似于大多数编程语言中的函数调用。
规则	规则构成了语法的主体。 A规则可以有一个或多个定义，由管道分隔符 (|) 分隔。 要通过规则，必须通过任何单个完成定义。规则以分号字符 (;) 结束。
定义	A是规则可以采用的路径之一。每个定义都由一个或多个术语组成。
定义列表	A列表对应于一组或多组术语。这些将按顺序进行评估，直到成功。如果没有成功，则包含规则失败。每对定义由 | 分隔。特点。 这是一个简单的规则，包含三个定义： <问候> ::= "你好" | 嗨" | [早上好";
条款	术语可以是对规则、特定值、值范围、子规则或命令A引用。
命令	像指令一样，命令类似于函数调用。它们有两个主要目的： ● 以特定方式处理令牌或 ● 向调用者提供结果

语法说明

说明指定语法如何工作的关键细节。它们通常包含在语法的顶部，类似于大多数编程语言中的函数调用。

指示

操作说明	描述
区分大小写（ ）	这两个指令之一应指定令牌匹配是否需要区分大小写。例如，BASIC 家族的语言不区分大小写，而 C 家族的语言区分大小写。
不区分大小写（ ）	
分隔符（DelimiterRule：表达式）	delimiters 指令告诉词法分析器使用哪个规则来发现分隔符。定界符用于关键字分析，可以定义为可以在语言关键字之前或之后使用的字符。
lex(TokenRule: 表达式)	lex 指令告诉词法分析器用于分析的根规则的名称。
解析（根规则：表达式） 解析（RootRule：表达式，SkipRule：表达式）	parse 指令告诉解析器用于处理的根规则的名称。可选的第二个参数指定一个跳过（或转义）规则，通常用于处理注释。

语法规则

规则是将运行分解为结构的运行。规则由A或多个定义组成，每个定义由一个或多个术语组成。

规则类型

规则	描述
命名规则	A名称，后跟一个定义列表。例如： <规则> ::= <术语1> <术语2> | "-" <term1>;
内联规则	在定义内，括号内定义的规则。它们的行为方式与它们是被术语调用的命名规则完全相同。例如： <规则> ::= (<内联>);
可选规则	在定义内，方括号内定义的规则。即使内容失败，此规则也会成功。例如： <规则> ::= [<内联>];
重复规则	在定义中，一个术语后跟一个加号。此规则匹配内部规则一次或多次。例如： <规则> ::= <内联>+; 规则 ::= (<term1> <term2>)+;
可选的重复规则	在定义中，规则后跟一个星号。此规则与内部规则匹配零次或多次，这意味着即使内部规则从未成功，它也会成功。例如： <规则> ::= <内联>*; 规则 ::= (<term1> <term2>)*;

语法术语

条款确定了代币的消费地点。

团队类型

类型	描述
具体条款	引用的字符串。 例如， 班级″
统一码字	仅词法分析器术语，前缀为 U+0x，后跟A十六进制数。 例如：U+0x1234
范围	仅词法分析器术语，匹配指定A两个字符之间的任何字符。 例如，“a”..“z″或 U+0x1234..U+2345
参考	另一个规则的名称，在尖括号中。如果该规则成功，令牌将匹配。 例如，<anotherRule>
命令	对特定命令A调用。

语法命令

命令，如指令，类似于函数调用。它们有两个主要目的：

- 以特定方式处理令牌或
- 向调用者提供结果

命令

命令	描述
属性（名称：字符串，值：表达式）	在当前 AST 节点上创建一个属性。该属性将使用语法源中指定的名称创建，并将被赋予作为执行 Value 表达式的一部分所消耗的所有标记的值。 此命令生成Enterprise Architect在代码工程中操作的 AST 节点属性。
attributeEx(名称：字符串) attributeEx(名称：字符串，值：字符串)	在当前 AST 节点上创建一个属性，而不消耗任何令牌。该属性将使用与语法源中指定的名称相同的名称创建，并且具有空值或由可选的 Value 参数指定的值。 此命令生成Enterprise Architect在代码工程中操作的 AST 节点属性。
node(名称：字符串,目标：表情)	在当前 AST 节点（ Enterprise Architect在代码工程中操作的节点）下创建一个 AST 节点。将使用语法源中指定的名称创建节点。
令牌（目标：表达式）	在词法分析期间创建一个标记，以便在解析期间进行处理。令牌的值将是执行目标表达式所消耗的所有字符的值。
关键字（ ）	匹配任何用作语法术语的文字string ；也就是说，如果您输入要搜索的显式string ，它将成为关键字。
跳过（目标：表达式） skip(目标：表达式，转义：表达式)	使用输入数据（词法分析时的字符和解析时的标记），直到匹配“目标”表达式。可选的“Escape”表达 可用于处理字符串中的转义引号等实例。
skipBalanced(原点：表情，目标：表情) 目标(Origin: Expression, Target: Expression, Escape: Expression)	消耗输入数据（字符），直到“目标”表达式匹配并且嵌套级别或令牌达到零。如果在此过程中匹配了“Origin”表达 ，则增加了嵌套级别。如果匹配“目标”表达 ，则嵌套“目标”级别。当命令嵌套级别达到零时，成功。可以提供可选的“Escape”表达 。
跳过EOF()	使用所有剩余数据（字符或标记）直到文件结束。
失败（ ）	导致解析器使当前规则失败，包括任何剩余的定义。
警告（ ）	在生成的 AST 中插入警告。
除了（目标：表达式，例外：表达式）	使用与目标表达式匹配的输入数据，但在与异常表达式匹配的数据上失败。这个操作有点类似于跳过命令，但完全相反。
preProcess(目标：表达式)	评估一个表达式并在多个定义中使用该预处理数据。这在表达式解析中最有用，其中将针对多个运算符评估相同的左侧表达式。该命令减少了解析器为实现这一点而必须做的工作。

AST 节点

在定义语法时，您将使用可以在Enterprise Architect的代码工程中识别的 AST 节点和 AST 节点属性，以及由 attribute、attributeEx 和 node 命令返回的 AST 结果。这些库表中标识了节点和属性。在代码工程中，任何其他人都将被忽略。

文件节点

FILE 节点代表一个文件。它没有映射到任何东西，但包含所有必需的信息。

多重性/节点	描述
0..* / 包	请参阅*PACKAGE*节点。
0..* / 类	见*CLASS*节点。
0..* / 进口	表示导入的命名空间/包或等效项的节点。节点的 'NAME' 属性将是导入的命名空间/包的名称或等效名称。
0..* / 评论	作为跳过规则一部分的字段标签将在根级别；代码生成器通过相对于节点的位置查找此类注释。
0.. 1 / INSERT_POSITION	这给出了可以将新类、包和方法实现插入文件的位置。如果没有找到，代码生成器会在代码中找到最后一项后立即自动插入新项。

包节点

PACKAGE 节点对应于文件中的命名空间或等效项。当使用 包per namespace″导入时， Enterprise Architect将直接在导入下创建一个包，并将所有类放在其中。当不导入命名空间时， Enterprise Architect将在这一点下寻找类，但它不会对这个节点做任何事情。

此外，如果您在启用命名空间的情况下生成（请参阅通用语言的代码选项帮助主题），除非它们位于相同的包结构下，否则生成的类将不匹配代码中的类。

包含在节点中：FILE

多重性/节点	描述
1 / 姓名	见*NAME*节点。
0..* / 类	见*CLASS*节点。
0..* / 包	子包节点。
0.. 1 / OPEN_POSITION	给出包体打开的位置。这也可以用作插入位置。
0.. 1 / INSERT_POSITION	给出新类和包可以插入文件的位置。如果没有找到，代码生成器会在代码中找到最后一项后立即自动插入新项。
0.. 1 / 抑制	插入此包时防止缩进。

CLASS/INTERFACE节点

CLASS（或 INTERFACE）节点是代码生成中最重要的节点。它作为类（或接口）对象引入。
参见类*DECLARATION*和类*BODY* 。
包含在节点中：FILE、PACKAGE、类BODY

类声明

包含在节点中：类/接口

多重性/节点	描述
1 / 姓名	见*NAME*节点。
0..* / 家长	请参见*PARENT*节点。
0..* / 标签	见*TAG*节点。
0.. 1 / 描述	见节点。
1 / 姓名	类的名称。如果有一个节点 NAME，那将覆盖这个属性。
0.. 1 / 范围	类的UML范围 - Public、Private、Protected 或包。
0.. 1 / 摘要	如果存在，则表明这是一个抽象类。
0.. 1 / 版本	类的版本。
0.. 1 / 刻板印象	Enterprise Architect应该分配给类的构造型。这不支持多种刻板印象。
0.. 1 / 岛	如果存在，则表明这是一个叶子/最终/密封类，不能被任何子类继承。
0.. 1 / 多重性	如果存在，则表示类的多样性。
0.. 1 / 语言	一般不需要设置。
0.. 1 / 注意	通常不使用，因为它在类上面的评论中得到解决。
0.. 1 / 别名	如果存在，则表示任何标识符的别名，例如命名空间、类或变量。
0..* / 宏	添加Enterprise Architect可用于round宏的编号标记值。

类BODY节点

包含在节点中：类/接口

多重性/节点	描述
0..* / 方法	请参见*METHOD*节点。
0..* / 属性	请参见*ATTRIBUTE*节点。
0..* / 场	见*FIELD*节点。
0..* / 类	见*CLASS*节点。
0..* / 范围	见*SCOPE*节点。
0..* / 属性	该节点表示类体的属性定义。
0..* / 标签	见*TAG*节点。
0..* / 家长	请参见*PARENT*节点。
0.. 1 / OPEN_POSITION	给出类体打开的位置。这也可以用作插入位置。
0.. 1 / INSERT_POSITION	给出新类成员可以插入文件的位置。如果没有找到，代码生成器会在代码中找到最后一项后立即自动插入新项。

SCOPE节点

对于具有指定元素范围的块的类似于 C++ 的语言，这是一个可选特征。语言需要指定一个名称，用于块中所有元素的范围。在所有其他方面，它的行为与类节点相同。

包含在节点中：类BODY

多重性/节点	描述
1 / 姓名	用作范围内包含的所有方法和属性的范围。

METHOD节点

包含在节点中：类BODY、SCOPE

多重性/节点	描述
1 / 方法声明	请参阅方法节点。

方法节点

包含在节点中：方法

多重性/节点	描述
0.. 1 / 类型	见*TYPE*节点。
0..* / 参数	请参见*PARAMETER*节点。
0..* / 标签	请参阅标记节点。
0.. 1 / 描述	见节点。
0.. 1 / 多参数	支持 Delphi 的参数列表样式的声明。这相当于 FIELD。
1 / 姓名	方法的名称。
0.. 1 / 类型	方法的返回类型。
0.. 1 / 范围	方法的UML范围 - Public、Private、Protected 或包。
0.. 1 / 摘要	如果存在，则表明该方法是抽象的。
0.. 1 / 刻板印象	Enterprise Architect应该分配给方法的构造型。这不支持多种刻板印象。
0.. 1 / 静态	如果存在，则表明该方法是静态的。
0.. 1 / 常数或常数	如果存在，则表明该方法是恒定的。
0.. 1 / 纯	如果存在，则表明该方法是纯方法。
0.. 1 / 查询	如果存在，则表明该方法是查询/只读的。
0.. 1 / 阵列	如果存在，则表明方法类型（返回类型）是一个数组。
0.. 1 / 同步	如果存在，则表明该方法是同步方法。
0..* / 宏	方法声明中指定的宏。
0.. 1 / CSHARPIMPLEMENTS	指定 C# 的特殊行为。
0.. 1 / 行为	使用行为提供对 Aspect J的支持。
0.. 1 / 显示行为	提供对 Aspect J的支持，使用行为，并在图表上显示逆向工程行为。

ATTRIBUTE节点

包含在节点中：类BODY、SCOPE

多重性/节点	描述

1 / 类型	见*TYPE*节点。
0..* / 标签	见*TAG*节点。
0.. 1 / 描述	见节点。
1 / 姓名	属性的名称。
0.. 1 / 类型	属性的类型。
0.. 1 / 范围	属性的UML范围 - Public、Private、Protected 或包。
0.. 1 / 默认	属性的默认值。
0.. 1 / 容器或数组	如果存在，则指示属性的容器。
0.. 1 / 遏制	参考或价值。
0.. 1 / 刻板印象	Enterprise Architect应该分配给属性的构造型。这不支持多种刻板印象。
0.. 1 / 静态	如果存在，则表明它是静态属性。
0.. 1 / 常数或常数	如果存在，则表明它是一个常量属性。
0.. 1 / 已订购	如果存在，则表明属性（值）是有序的。
0.. 1 / 低限	如果存在，则表示属性值的下边界。
0.. 1 / 高边界	如果存在，则表示属性值的较高边界。
0.. 1 / 瞬态或易失	如果存在，则表明该属性是瞬态的或易失的。

FIELD节点

A字段对应多个属性声明在一个。任何未在声明符中定义但在字段本身中定义的内容都将为每个声明符设置。字段中支持属性中支持的所有内容。如果没有找到声明符，那么这与属性的工作方式相同。

包含在节点中：类BODY、SCOPE

多重性/节点	描述
0..* / 声明者	请参见*ATTRIBUTE*节点。

PARAMETER节点

包含在节点中：方法声明、模板

多重性/节点	描述

1 / 类型	见*TYPE*节点。
0..* / 标签	见*TAG*节点。
0.. 1 / 描述	见节点。
0.. 1 / 名称	参数的名称。
0.. 1 / 类型	参数的类型。
0.. 1 / 种类	预计进入、退出、退出或返回。
0.. 1 / 默认	参数的默认值。
0.. 1 / 固定	如果存在，则表明该参数是固定的/恒定的。
0.. 1 / 阵列	如果存在，则表明参数类型是一个数组。

NAME节点

包含在节点中：PACKAGE,类DECLARATION

多重性/节点	描述
1 / 姓名	名称部分。
0..* / 限定符	限定符部分。
0..* / 名称部分	使用 NAME 和 QUALIFIER 的替代方法。 A string ，除了最后一个作为限定符之外的所有值。最后一个作为名称。

TYPE节点

包含在节点中：方法声明、属性、参数

多重性/节点	描述
0.. 1 / 模板	模板的整个文本是类型的名称。 仅在 NAME未定义使用。 见*TEMPLATE*节点。
1 / 姓名	名称部分。
0..* / 限定符	限定符部分。
0..* / 名称部分	使用 NAME 和 QUALIFIER 的替代方法。 A string ，除了最后一个作为限定符之外的所有值。最后一个作为名称。

TEMPLATE节点

包含在节点中：TYPE

多重性/节点	描述
0..* / 参数	请参见*PARAMETER*节点。
1 / 姓名	

PARENT节点

包含在节点中：类声明

多重性/节点	描述
0.. 1 / 类型	具有值 Parent、Implements 或 VirtualP。
1 / 姓名	父级的名称部分。
0..* / 限定符	Parent 的限定符部分。
0..* / 名称部分	使用 NAME 和 QUALIFIER 的替代方法。 A string ，除了最后一个作为限定符之外的所有值。最后一个作为名称。
0.. 1 / 实例化	如果存在，则表示模板参数的实例化。

TAG节点

包含在Nodes中：类DECLARATION、Method DECLARATION、ATTRIBUTE、PARAMETER

多重性/节点	描述
1 / 姓名	标记值（标签）的名称。
0..* / 值	标记值的价值。
0.. 1 / 备忘录	如果存在，则表示标注标记值的类型为<memo>。
0.. 1 / NOMEMO	如果存在，则表示标注标记值的类型不是<memo>。
0.. 1 / 组	如果存在，则表示该值为一个标记值组。

描述节点

包含在Nodes中：类DECLARATION、Method DECLARATION、ATTRIBUTE、PARAMETER

多重性/节点	描述
0..* / 值	Enterprise Architect应分配给注记的文本。

编辑语法

如果您需要为以新编程语言导入的代码编写和编辑语法，您可以使用内置的语法编辑器来完成。

访问

功能区	开发 >源代码 > 语法编辑器

创建和编辑语法

字段/按钮	行动
开放语法	显示一个浏览器，您可以通过该浏览器找到并打开包含您要编辑的语法的文件。
最近的	使用此组合框可以快速访问最近使用的语法。
节省	保存当前文件。
另存为	保存当前文件的副本
验证语法	语法验证将对当前运行进行一系列测试以确保其有效性。将显示错误和警告，通知您导致语法不可用的错误以及可能获得意外结果的情况。
帮助	显示此帮助主题。

上下文菜单选项

字段/按钮	行动
打开文件	显示一个浏览器，您可以通过该浏览器找到并打开包含您要编辑的语法的文件。
证实	语法验证将对当前运行进行一系列测试以确保其有效性。将显示错误和警告，通知您导致语法不可用的错误以及可能获得意外结果的情况。
语	语法编辑器默认为正常的 Backus-Naur 形式 (nBNF)。 mBNF 选项也可用。
行号	在语法编辑器中打开或关闭行号。

解析 AST 结果

抽象语法树(AST) 是Enterprise Architect在处理语法时看到的代码。

您在语法编辑器窗口的下半部分解析文本并审阅显示的结果。您可以打开文件或在其中粘贴文本。如果粘贴的文本对应于无法在文件级别显示的内容（例如操作参数），则可以选择替代规则作为起点。然后解析将从该规则开始。

访问

功能区	开发 >源代码 > 语法编辑器 > 语法调试器> AST 结果

工具栏选项

选项	行动
打开文件	打开一个示例输入文件进行测试。
最近的	可以从此组合框中选择最近打开的源文件。
解析	执行解析操作。如果解析成功，“AST 结果”选项卡将包含生成的 AST。
选择规则	此下拉菜单允许您选择替代根规则来处理您的样本源。
帮助	显示此帮助主题。

分析语法分析

当您解析您创建的语法时，它可能会显示您无法立即诊断的错误。为了帮助您解决此类错误，您可以使用Grammar Profiler审阅解析器生成您可以看到的 AST 的过程。

您再次解析语法编辑器窗口下半部分的文本，但这一次树显示了解析器尝试的每条规则、到达的位置以及是否通过。打开文件、粘贴文件和设置启动规则的规则保持不变。

访问

功能区	开发 >源代码 > 语法编辑器 > 语法调试器> Profiler 结果

工具栏选项

选项	行动
打开文件	显示一个浏览器，您可以通过该浏览器找到并打开包含您要编辑的语法的文件。
解析	执行解析操作。如果解析成功，“AST 结果”选项卡将包含生成的 AST， 配置文件结果”选项卡将包含有关解析器通过您的语法所采用的路径的调试信息。在调试新语法时，配置文件数据非常有用。
选择规则	如果您想使用不同的根规则来处理您的样本源，请单击下拉箭头并选择替代规则。
帮助	显示此帮助主题。

注记

- 因为对大文件进行分析可能需要很长时间，所以如果您在开始解析时没有显示该选项卡，则不会填充 配置文件结果”选项卡

宏编辑器

宏编辑器允许用户用关键字和规则列表来补充语法，以在语法分析操作期间排除宏。在为支持宏的语言（如C++）开发语法时，宏定义列表特别有用。它避免了在语法本身中描述这些规则的必要性，并且可以与多种语法一起使用。

此特征可从Enterprise Architect Release 14. 1中获得。

访问

功能区	开发 >源代码 > 语法编辑器 > 宏编辑器

编辑宏

打开文件	打开现有的宏定义列表
最近的	可以从此组合框中选择最近打开的宏定义列表
节省	保存对打开的宏定义列表的更改
另存为	保存现有宏定义列表的副本
证实	验证宏定义列表的语法

示例语法

Enterprise Architect安装程序设置的代码样本目录包含一个示例语法，您可以将其加载到语法编辑器中进行审阅，也可以加载到语法调试器中进行解析和分析。

语法示例包含两个文件：

- test.ssl - 一个简单的示例语言源文件，采用 C 风格，以及
- ssl.nbnf - 简单示例语言的语法

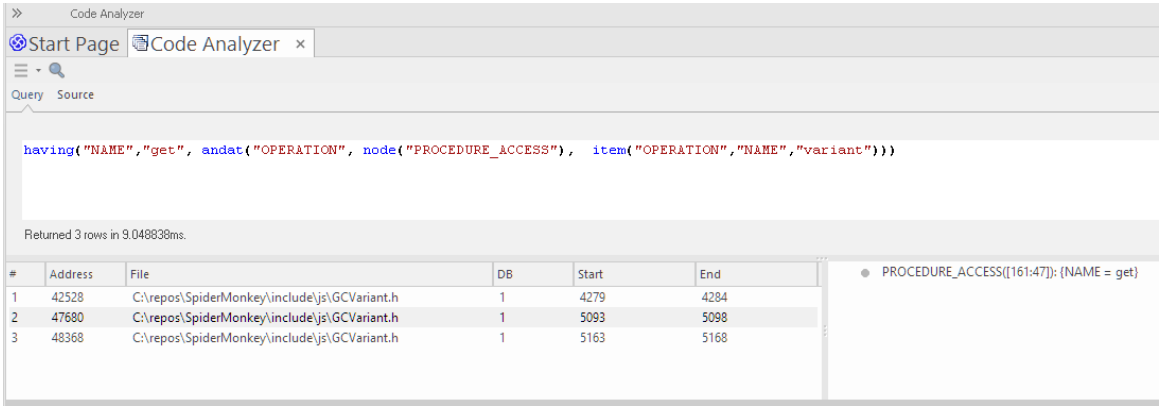
该示例说明：

- 标记化（使用 Lexer）
- 创建一个包
- 创建一个类或接口
- 创建属性
- 创建操作（带参数）
- 导入评论

样本目录还包含您可以检查的其他两个语法文件：

- Expressions Sample.nBNF - 这说明了如何设置和处理表达式解析，详细的注释文本提供解释
- CSV Sample.nBNF - 处理 CSV 文件的示例语法

代码分析器



对于每天处理源代码的人来说，代码分析器是必不可少的工具。

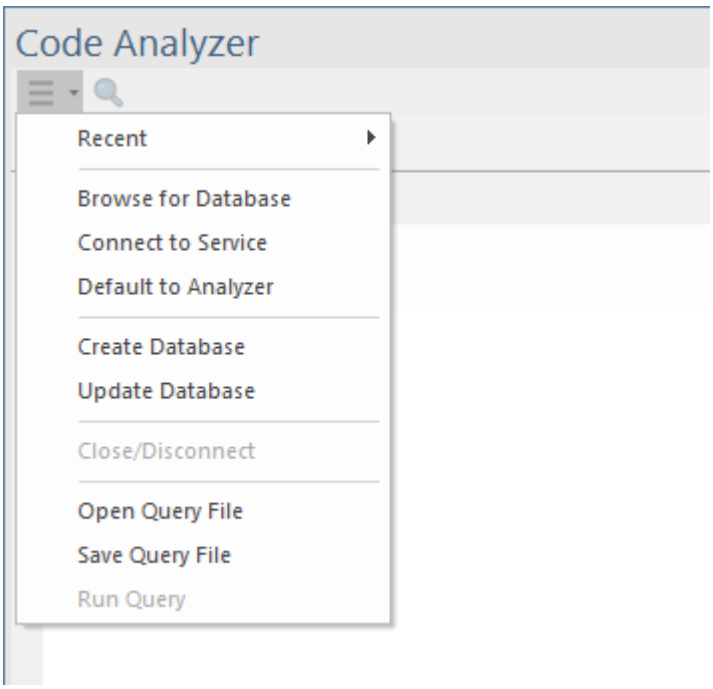
它可以在本地或 Sparx 英特尔云服务上以闪电般的速度对源代码存储库执行非常复杂的查询。查询是使用 Sparx系统开发的高级语言组成的。该语言使用易于学习的小而富有表现力的词汇表，并且允许比传统方法更快地查询代码指标。

访问

功能区	开发 >源代码 > 代码分析器

代码分析器菜单

当您单击窗口左上角的 图标时，将显示代码分析器菜单。



该菜单为与代码分析器的使用相关的活动提供各种命令，包括选择要使用的代码代码矿工数据库、更新代码矿工数据库和打开代码矿工查询文件进行编辑。

此表描述了每个菜单命令。

命令	描述
最近的	显示一个子菜单，提供最近连接到服务和本地数据库文件的列表。
浏览数据库	显示 文件选择器"对话框，允许您浏览机器上的代码矿工数据库。
连接到服务	显示 代码矿工数据库连接"对话框，您可以在其中指定（列表）代码矿工数据库服务的连接详细信息。
默认为分析器	选择此选项会导致代码脚本在启动代码分析器自动连接到为活动执行分析器分析器配置的代码矿工服务。
创建数据库	显示 创建代码矿工数据库"对话框，允许您从文件系统中的源代码存储库创建代码矿工数据库。
更新数据库	显示 代码矿工数据库更新"对话框，该对话框允许您对现有代码矿工数据库执行增量更新，以合并对源代码文件的最新更改。
关闭/断开	关闭或断开与代码矿工数据库库或服务的连接。
打开查询文件	显示 文件打开"对话框，允许您从文件系统中选择 mFQL 查询文件。
保存查询文件	显示 文件保存"对话框，允许您将当前 mFQL 查询保存到命名文件。
运行查询	运行在 查询"选项卡编辑器中输入的整个查询或查询的选定内容。 快捷键 F6。

使用分析器之前

在您可以使用代码分析器，您必须首先创建一个代码矿工数据库或找到代码分析器可以访问的现有数据库。这里总结了创建代码矿工数据库，或者您可以阅读帮助主题创建新代码矿工数据库帮助的详细说明。

根据您将使用的库的位置，您应该：

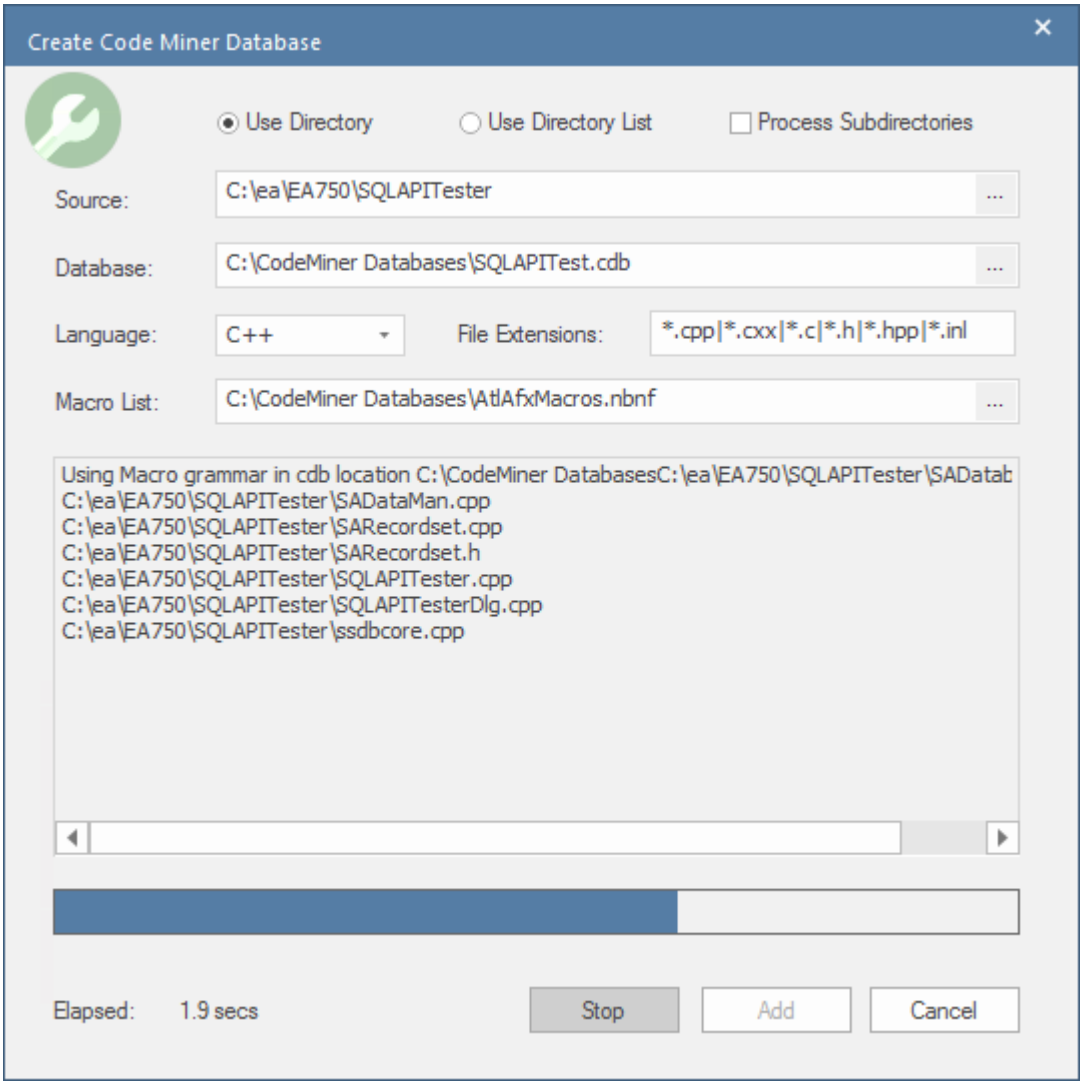
- 选择要使用的代码矿工库文件，或
- 连接到托管代码矿工数据库的服务。

完成这些步骤后，您就可以开始在代码分析器中编写和运行查询了。

创建代码矿工数据库

代码矿工数据库是从源代码存储库构建的。该过程类似于代码编译，使用语言语法分析单个文件。

有两种类型的构建——完整的和增量的。最初的完整构建可能需要一些时间，但随后的增量构建非常快。



使用目录作为输入

您可以选择单个文件夹作为要编译的源代码的根。使用此选项，您可以选择包含子目录

使用目录列表

有时，您想使用多个项目，但并非所有项目都在一个目录下。在这种情况下，您可以创建一个文本文件，列出您要包含的每个文件夹的完整路径，并在 源"字段中指定该文本文件。每个目录路径都应列在单独的行上。

c:\myprojects\project1\tools\scintilla

c:\myprojects\project2\src

d:\mylibs\lib1\src

如果要递归处理目录中的子目录，请在路径前加上感叹号，如下所示：

!d:\mylibs\lib1\src

任何以 # 字符开头的行都被视为注释。

包括闪烁

c:\myprojects\project1\tools\scintilla

语

在此字段中，您指定用于构建此代码矿工数据库的源代码中使用的语言。

可用的语言有：C++、C#、 Java 、XML、MDGTechnology 和 Custom。

宏列表

When the language selected is 'C++', the 'Macro List' selection field is displayed .对于C++来说，信息编译到数据库中的成功和深度与宏的使用有着千丝万缕的联系。此字段可用于选择将用作编译的辅助语法组件的 nBNF 宏文件。

默认情况下，宏文件将默认为Enterprise Architect安装文件夹中的宏文件。您可以自由修改或扩展此文件的内容以满足您的要求 - 例如，当您需要更正编译log文件中报告的错误时。

语法

Sparx Systems为下拉选择列表中列出的所有语言开发了语法； C++、C#、 Java 、XML 以及 MDGTechnology。对于这些语言，使用内置语法文件。

还有一个选项可以选择 自定义"语言。选择 自定义"时，将显示 语法"字段。此字段用于指定包含自定义语言语法的文件。然后代码矿工将使用该语法来解析以该语言编写的源代码。

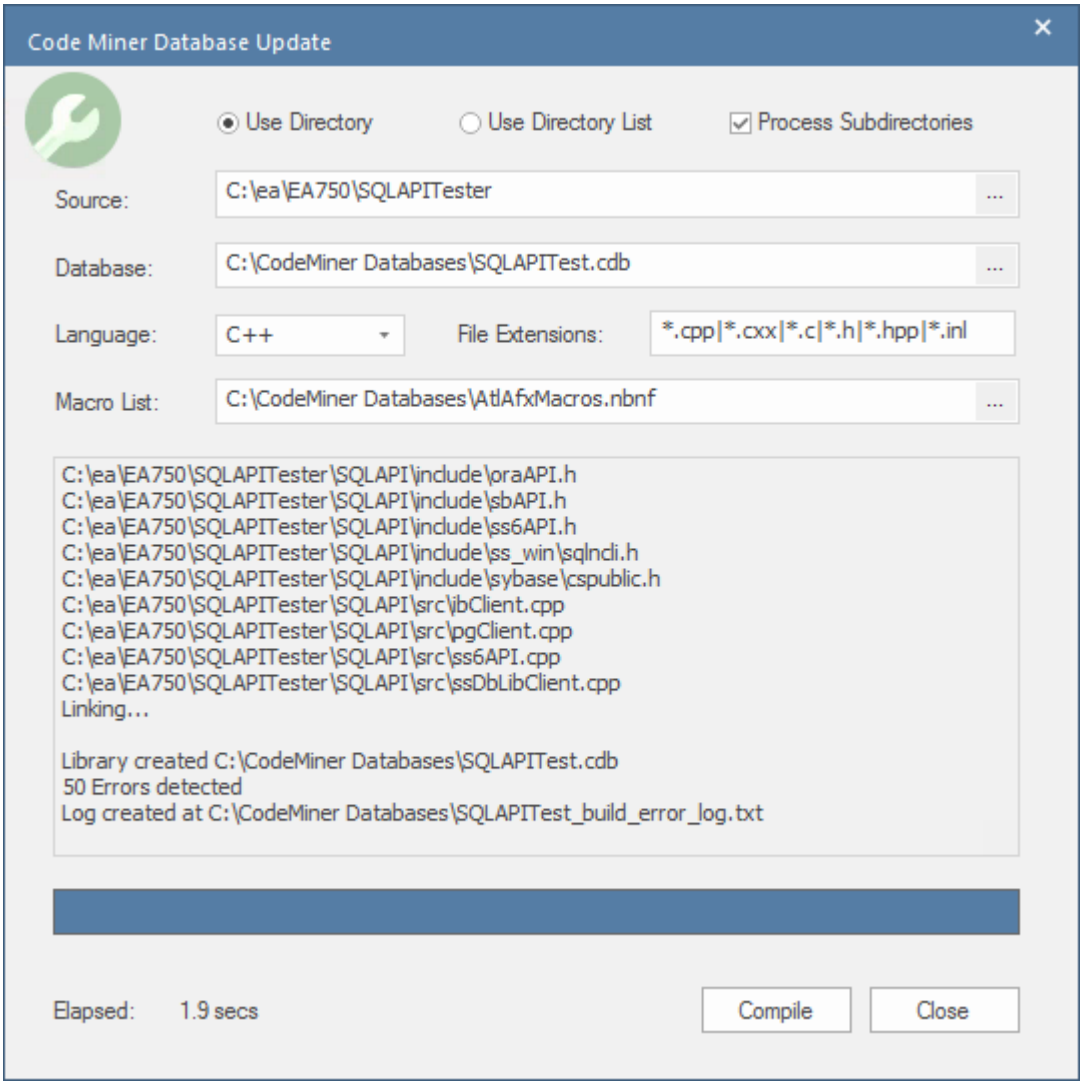
开发自定义语言的用户需要指定该语言的语法规则并将它们保存到 nBNF 文件中。 Enterprise Architect的语法编辑器专为此目的而设计。

The帮助主题语法框架提供了有关编写帮助语法的详细信息。

更新代码矿工数据库

有时，您会想要更新您的代码矿工数据库。通常，当您对源代码进行更改时，以及在更新语法文件或扩展宏文件之后。

更新数据库的过程与创建新数据库非常相似，但速度更快，因为您不是从头开始。只需选择菜单选项 更新数据库"。将显示 代码矿工数据库更新"对话框。输入字段将使用上次构建的值填充。继续 创建代码矿工数据库"。



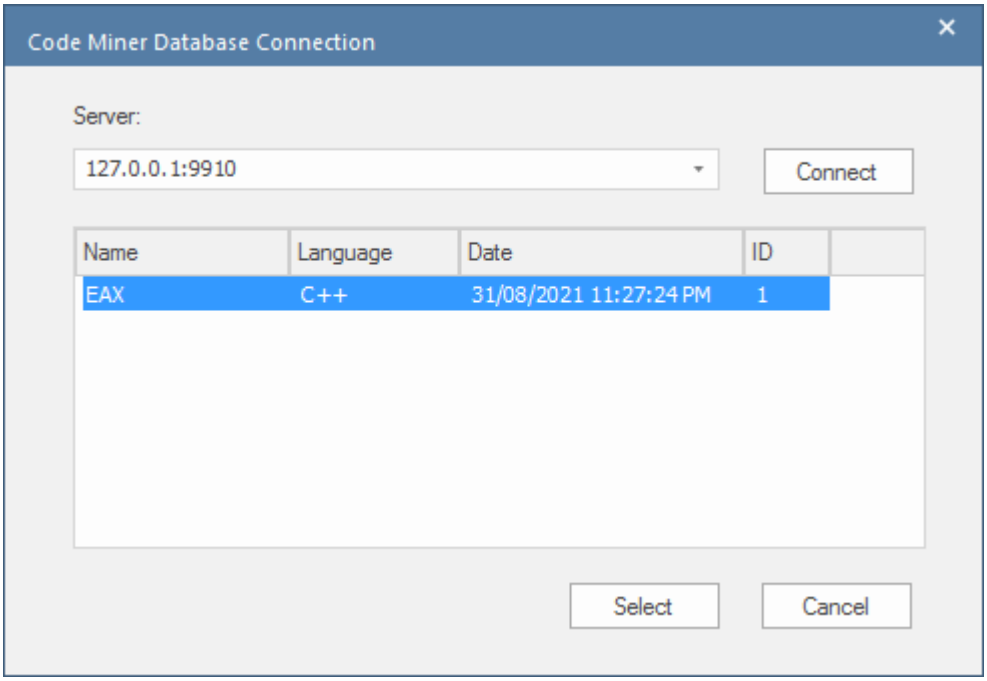
选择一个代码矿工数据库文件

如果您选择为您的代码矿工数据库使用库文件，请选择菜单选项 浏览数据库”。这将显示一个 文件选择器”，您可以在其中浏览并选择一个 *.cdb 文件。

连接到服务

连接到服务时，对话框会列出该服务托管的所有数据库。

您可以选择在列表中选择单个数据库，或者只需单击 选择”按钮，在这种情况下，将在服务列出的所有数据库中执行查询。

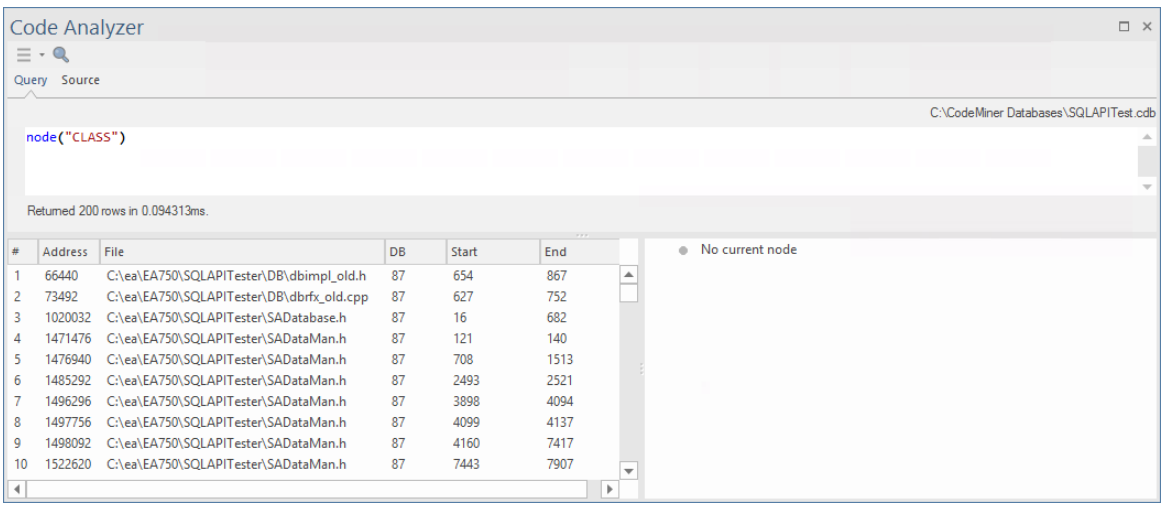


运行查询

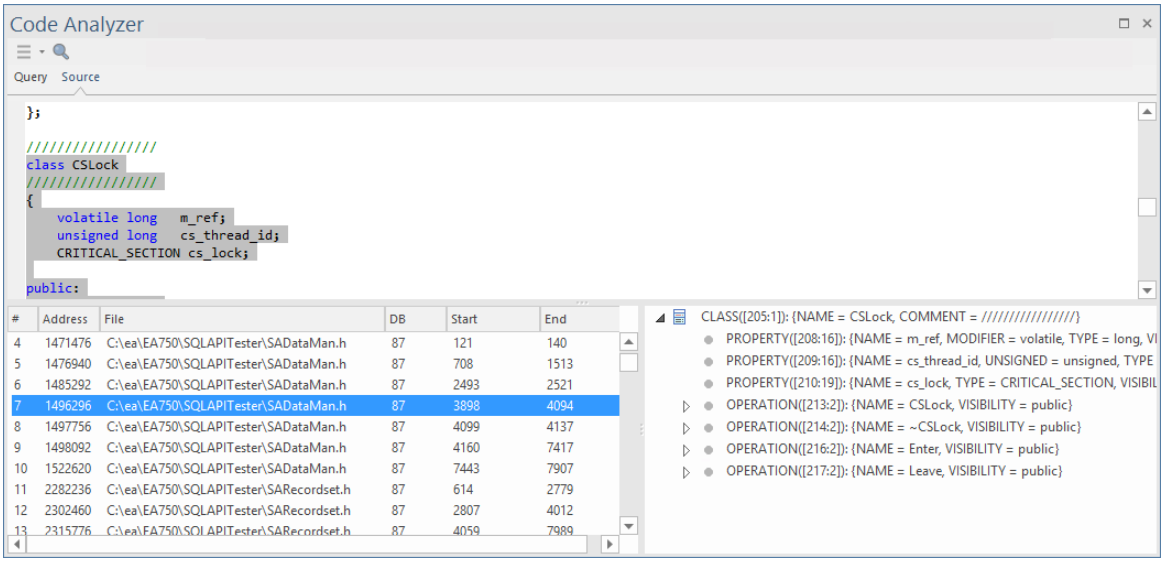
连接到代码矿工数据库后，您就可以开始运行查询了。

要运行查询，请选择代码分析器窗口中的查询选项卡，输入您的查询，然后单击 图标执行查询。

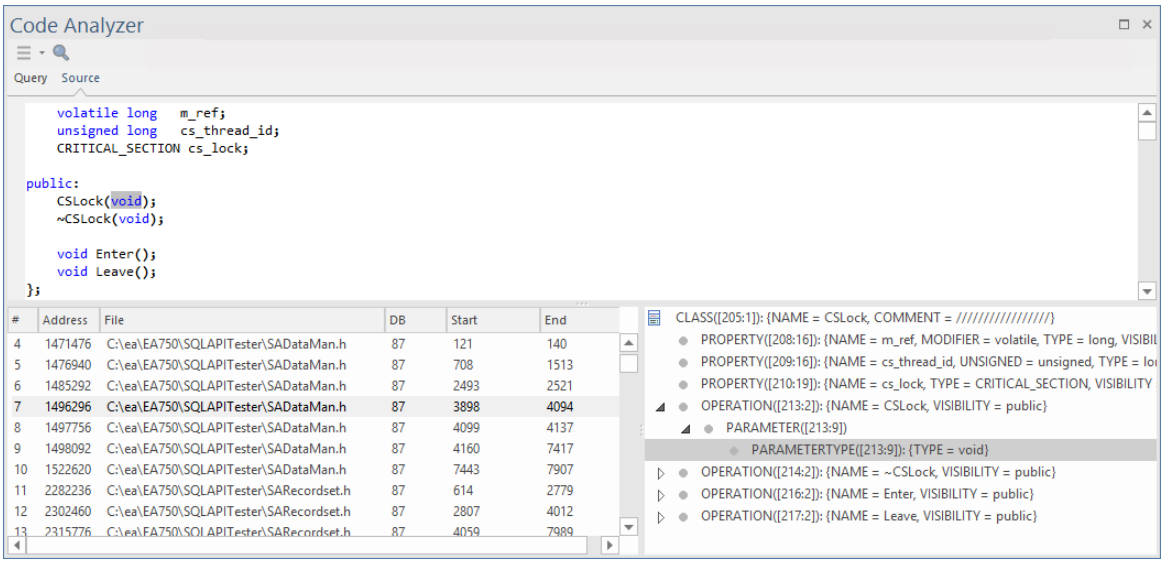
在此示例中，我们运行了一个简单的查询节点（“CLASS”），它将返回在代码矿工数据库中找到的所有 类”节点。



通过在左下方面板中选择一个结果， 源”选项卡被激活并显示 所选节点对应的源代码。该类节点的详细信息显示在右下方面板中。



在右下角面板中选择一个细节项，会缩小源代码中的选择范围，如此处所示。



示例查询-交叉路口

As an example, this mFQL query finds all the classes that have an operation named GetOption.

```
andat( "CLASS", item("OPERATION", "NAME", "GetOption"), node("CLASS"))
```

This clause returns a set of operations for which the 'NAME' value is "GetOption":

```
item("OPERATION", "NAME", "GetOption")
```

This clause returns a set of all Class nodes:

```
node("CLASS")
```

Formal syntax:

```
andat( string:rule, set:left, set:right)
```

'andat' takes the set of operations (left), applies the rule "CLASS" (only include rows that have a CLASS parent), then intersects that set with the set of all known classes (right). If the intersection succeeds, the operation node is added to the result set, otherwise it is excluded.

查询语言 - mFQL

代码分析器使用的查询语言在代码矿工查询语言 *(mFQL)* 帮助分析器帮助有完整描述。

此处还提供A简要说明和一些示例。

mFQL 语言基于集合。每个语句都使用各种类型的集合操作，其中只有少数。

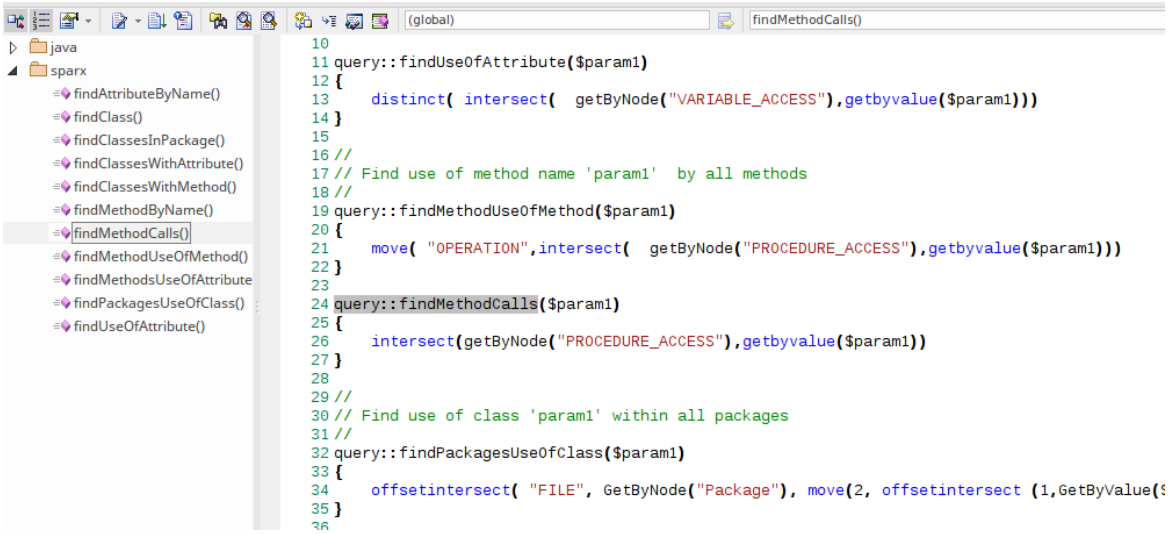
代码矿工框架

代码矿工系统提供对现有源代码中信息的快速和全面的访问。通过解析所有源代码并将生成的抽象语法树存储在读取优化的数据库中，系统以机器完成的格式提供对原始源代码所有方面的完整访问。

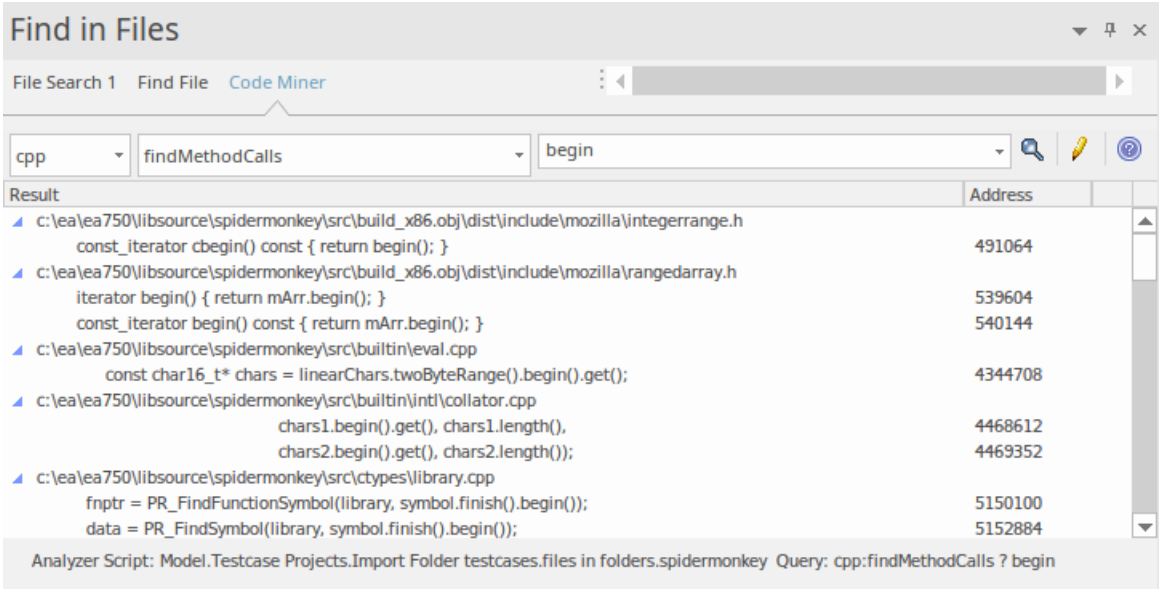
该系统背后的核心目标是及时有效地提供对隐藏在源代码中的数据的访问。为了确保最大性能，同时提供尽可能简单的接口，我们付出了巨大的努力。因此，该系统可用于分析程序结构、计算指标、跟踪关系甚至执行重构。

使用以代码矿工查询语言（mFQL）编写的查询来检索来自代码矿工数据库的信息，该语言是代码矿工自己的语言。该语言本身相当简单，提供少量命令。尽管语言很简单，但它支持任意大小和复杂性的查询。该设计为所有查询（无论大小）提供了极致的性能。

此特征可从Enterprise Architect Release 14. 1中获得。



Enterprise Architect分析器的代码分析器、它的搜索工具和它的代码编辑器的智能感知特征都利用了这些数据库中的信息。

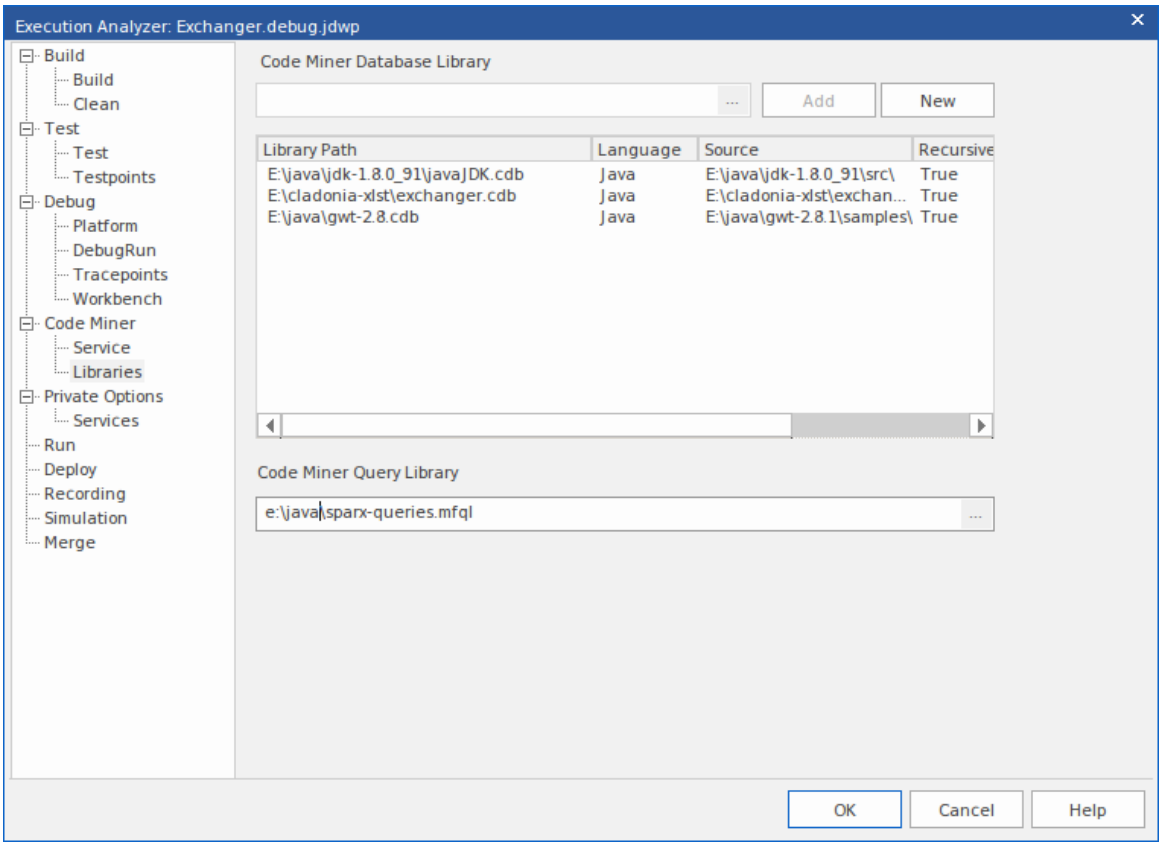


当前激活的分析器脚本以及查询参数，都在搜索工具的 代码矿工”页面底部标出。

代码矿工库

代码矿工库在Enterprise Architect中使用分析器脚本器进行管理。这些库是代码矿工数据库的集合，其中一个通常存在于每个框架或项目中。分析器脚本器允许创建新的数据库，以及添加、更新或删除现有的数据库。这些数据库共同构成了Enterprise Architect特征的代码代码分析器和智能感知代码矿工库。该库可以在本地使用，也可以部署到可以为多个客户端提供服务的服务器位置。您在分析器脚本的 脚本Intel Service"页面上选择要使用的场景。

此特征可从Enterprise Architect Release 14. 1中获得。



访问

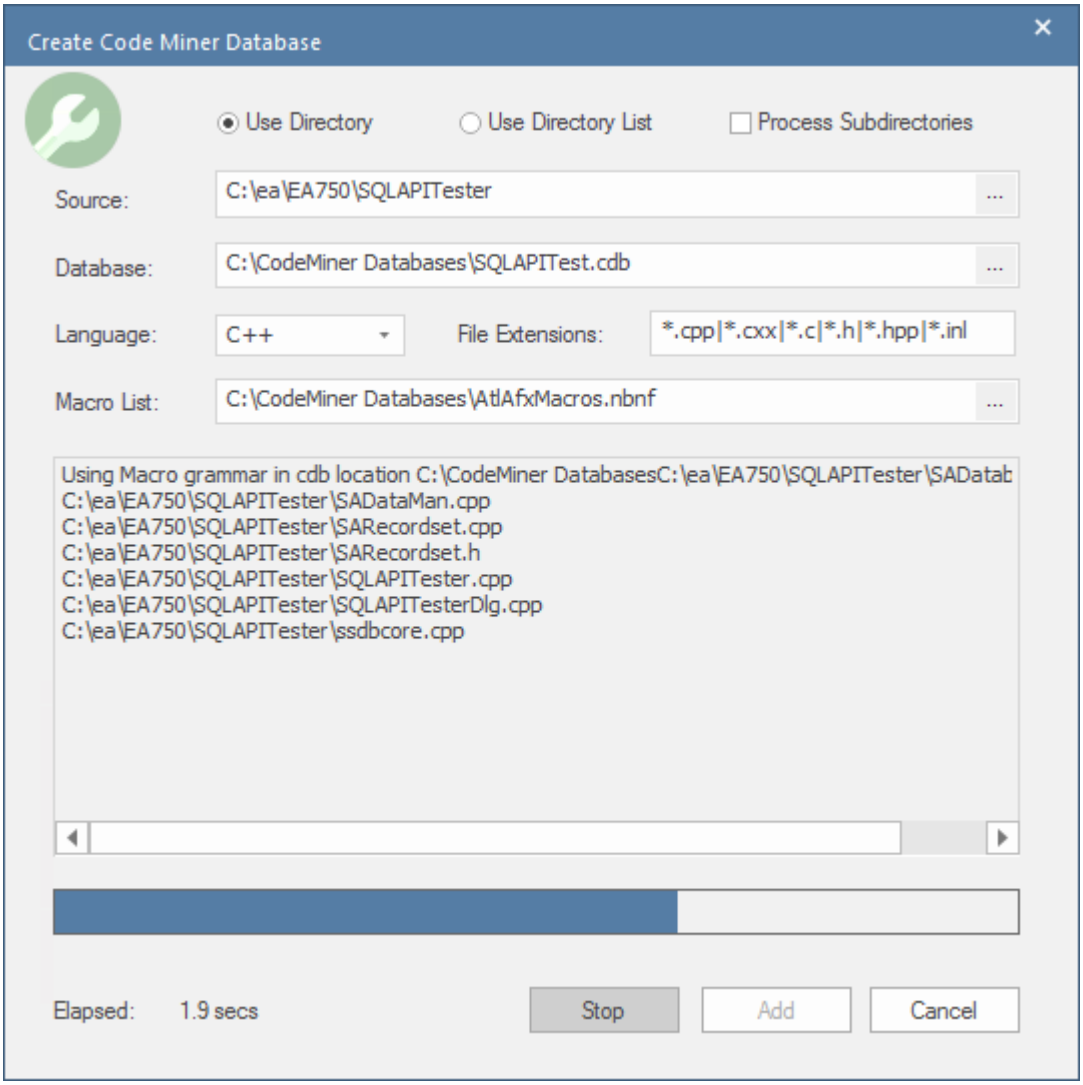
在执行分析器窗口中，找到并双击所需的脚本 - 将显示脚本编辑器对话框。在该对话框中，选择 代码矿工> 库"页面。

功能区	执行 > 工具 >分析器，或 开发>源代码>执行分析器>编辑分析器脚本

创建新数据库

关于'代码矿工|分析器编辑器的脚本页面，使用'New'按钮创建一个新的数据库。

在 创建代码矿工数据库"对话框中，指定包含项目源代码的文件夹，选择编程语言并输入代码矿工数据库的目标路径。当您单击 编译"按钮时，构建的详细信息将显示在log窗口中。



当过程完成时，按 添加”按钮将新创建的数据库添加到库中。

有关创建新数据库的详细信息，请参阅帮助主题创建新的代码矿工数据库。

添加现有数据库

使用数据库路径字段中的 “...”选择按钮选择现有的代码矿工数据库。

（代码矿工数据库具有 .CDB 文件扩展名），然后单击添加按钮。库中列出了有关数据库的详细信息。显示的信息显示了用于构建数据库的编程语言语法。还显示了在构建期间解析的代码库路径以及解析过程是否通过任何子目录递归应用。

更新数据库

有时，当您更新项目的源代码时，您会想要更新从该源代码构建的代码矿工数据库。

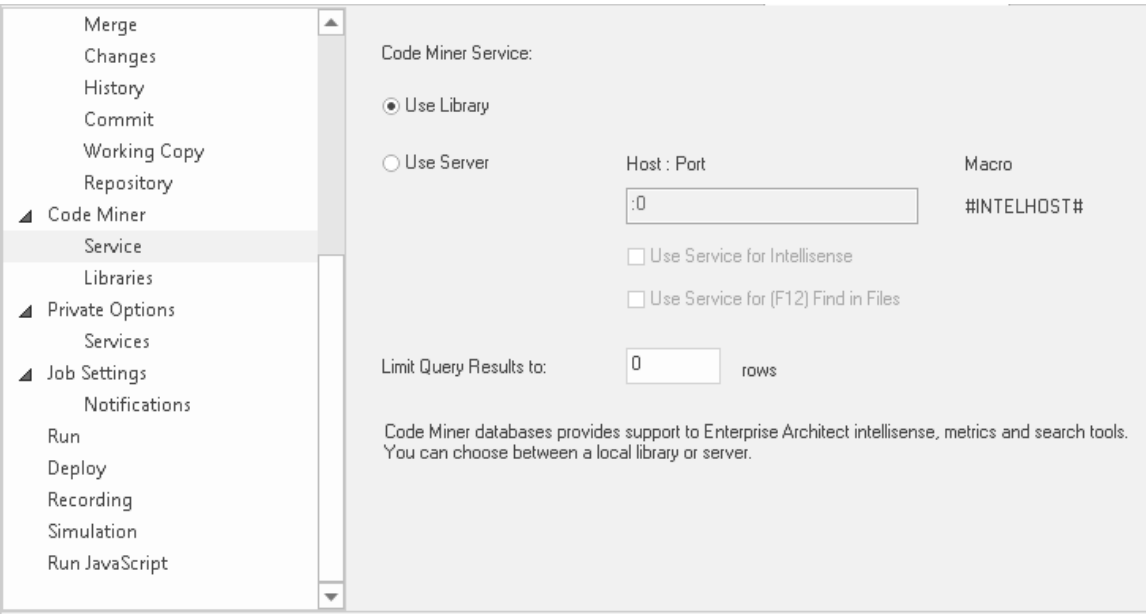
要更新代码矿工数据库，请从列表中选择它，右键单击并从其上下文菜单中选择 更新所选”。将显示A类似于 创建数据库”对话框的对话框。单击 编译”按钮，代码矿工将从更新的代码库中重新创建数据库。

删除数据库

要删除单个代码矿工数据库，请从列表中选择它，然后从其上下文菜单中选择 删除选定项”。

配置Enterprise Architect以使用代码矿工库

在Enterprise Architect分析器中，选择 脚本Intel Service”页面并选择 使用库”。 Enterprise Architect然后从当前活跃的分析器脚本 图书馆”部分列出的数据库中智能感知其信息。



创建新的代码矿工数据库

Enterprise Architect的代码分析器、智能感知特征及其代码编辑器的搜索工具均使用代码矿工数据库。

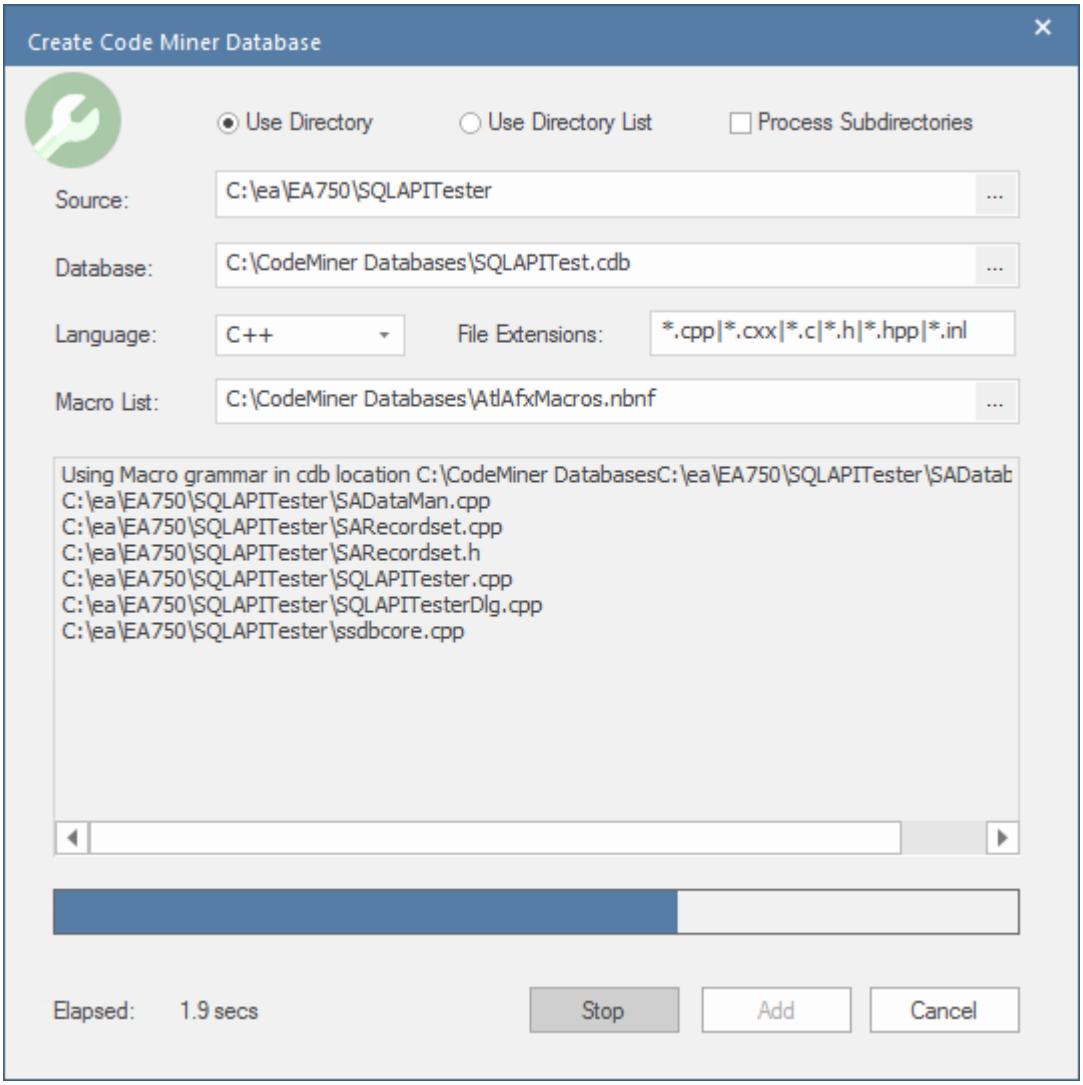
代码矿工数据库是通过A所选语言的语法规则解析源代码文件并将生成的抽象语法树存储在读取优化的数据库中而创建的。一个或多个数据库可以组合成一个代码矿工库。

访问

代码分析器窗口	在代码分析器窗口中，单击工具栏中的菜单按钮 ，然后选择菜单选项 创建数据库”。
执行分析器脚本	打开执行分析器的脚本编辑器窗口，选择页面'代码矿工>库'，然后点击'创建'按钮。

创建代码矿工数据库对话框

创建代码矿工数据库”对话框用 启动解析源代码文件以创建代码矿工数据库的过程。在对话框中，您可以指定进程使用的一系列输入，例如源代码文件夹、语言和宏列表文件，以及输出文件名。对话字段在下表中描述。



字段	描述
使用目录	当所有要处理的源文件都在一个目录下时，选择此选项。 选择此选项时，启用 进程子目录”复选框。
使用目录列表	当您的项目源代码位于多个单独的目录中时，请选择此选项。在这种情况下，您使用 源”字段来指定一个文件，该文件包含一个包含要处理的源代码的目录列表。
进程子目录	此复选框在选择 使用目录”选项时启用。选中后，驻留在指定 源”目录的任何子目录中的源代码文件也将被处理。
源	该字段用于指定包含将被处理以创建代码矿工数据库的源代码文件的目录（或多个目录）。 When the option '使用Directory' is selected, this field is used to specify the根folder in which to search for源code files. When the option '使用Directory List' is selected, this field is used to specify a user created file containing a list of path names to the directories that contain the源files to be processed.单击 ... 按钮打开一个 文件选择器”对话框，允许您浏览并选择扩展名为 .ssdirlist”的文件。有关详细信息，请参阅下面的目录列表文件部分。

数据库	此字段指定将创建的代码矿工数据库文件的完整路径名。该文件使用文件扩展名 ".cdb"。
语	这是一个下拉列表，您可以在其中指定正在处理的源代码文件中使用的语言。 Enterprise Architect为多种语言提供 "内置"支持。 （有用于解析支持的语言的内置语法）。 还有一个选项可以选择 "自定义"语言。如果您选择使用自定义语言，则需要创建自己的语法来支持对该语言的解析。选择 "自定义"选项时，将显示字段 "语法"字段，允许您指定定义自定义语法的文件。
文件扩展名	该字段列出了一些通常与所选语言的源代码文件相关联的文件扩展名。只有文件扩展名与列表中的文件扩展名匹配的文件才会被解析器处理。您可以添加或删除文件扩展名以满足您的需要。
宏列表	When the language selected is 'C++', the 'Macro List' selection field is displayed.宏列表字段允许您指定一个文件，该文件提供解析器在遇到宏时应跳过的宏列表。 对于 C++ 语言，宏给解析器带来了问题，因为它们隐藏了本地语言结构。将宏的名称添加到宏列表文件并更新数据库通常会清除与该宏相关的所有错误。 有关详细信息，请参阅下面的扩展宏列表文件部分。
语法文件	Sparx Systems为下拉选择列表中列出的所有语言开发了语法。 C++、C#、 Java 、XML 以及 MDGTechnology。 还有一个选项可以选择 "自定义"语言。开发自定义语言的用户需要指定该语言的语法规则并将其保存到 nBNF 文件中，以便代码矿工可以正确解析用该语言编写的源代码。 Enterprise Architect的语法编辑器专为此目的而设计。 当您选择 "自定义"作为语言时，您应该指定您为该语言创建的语法文件，以便代码矿工可以正确解析您的源代码。 The帮助主题语法框架提供了有关编写帮助语法的详细信息。
输出窗口	输出窗口显示解析源代码文件的进度。完成后，它还会显示创建的数据库文件和log文件的名称以及遇到的错误数。
编译/停止按钮	"编译"按钮用 启动处理操作。一旦处理开始，此按钮将变为 "停止"按钮，允许用户中止操作。
添加按钮	编译数据库后，可以使用 "添加"按钮将该数据库添加到代码矿工库。 可以将多个数据库添加在一起以构建一个涵盖许多源代码项目的库。 **注记**：从代码分析器窗口打开 "创建代码矿工数据库"对话框时，不显示 "添加"按钮。

目录列表文件

如果您选择指定目录列表文件，则需要使用文件扩展名 ".ssdirlist"创建一个简单的文本文件，其中列出了您希望处理的每个目录的完整路径，每行一个路径。例如：

c:\myprojects\project1\tools\scintilla

c:\myprojects\project2\src

d:\mylibs\lib1\src

如果您希望递归处理列出目录中的子目录，请在该路径前加上感叹号，如下所示：

!d:\mylibs\lib1\src

任何以 # 字符开头的行都被视为注释：

包括闪烁

c:\myprojects\project1\tools\scintilla

扩展宏列表文件

对于 C++ 语言，宏给语法带来了问题，因为它们隐藏了本地语言结构。解析器不能不对宏执行替换，因为它们通常是有条件地定义的，并且解析器不知道架构。宏列表文件提供了解析器在遇到宏时应跳过的宏列表。

当您为 C++源代码存储库构建代码矿工数据库时，您可能会看到列出的错误。发生错误时，使用错误log查找并检查导致错误的代码行。这几乎总是标识导致语法失败的宏。将该名称添加到宏列表并更新数据库通常会清除与该宏相关的所有错误。

例如，错误log显示此错误：

C:\ea\EA750\SQLAPITester\SQLAPI\include\asa\sqlfuncs.h，行：12，列：18，意外符号'，'。

经检查，导致错误的代码行是这样的：

FUNC_INFO(外部, void ， _esqlentry_, sqlstop, （ SQLCA *))

（还有许多其他类似的行使用宏“FUNC_INFO”。）

因此，我们编辑默认宏列表文件“AtxAflMacros.nbnf”，添加以下行：

"FUNC_INFO" "(" skipBalanced("(" , ")") ")" |

这一行指示解析器在遇到宏"FUNC_INFO"时应用函数skipBalanced("(" , ")")，它有两个参数；在这种情况下，它们是左括号和右括号。因此，指示解析器忽略左括号和右括号之间的所有内容。

保存对宏列表文件的更改并重新编译（更新）数据库后，与宏“FUNC_INFO”有关的所有错误都已消除。

了解更多

- Grammar Framework
- Code Analyzer

Code Miner Queries

Code Miner queries are best considered as functions written in the Code Miner NBNF Query Language (mFQL). As such, they have unique names, can be grouped by namespace and can take one or more parameters. Queries are bundled together into one source file. This source file is identified to Enterprise Architect by naming it in your Analyzer Script.

When specified, the queries it contains are available in the Code Miner control. Parameters to these queries can be taken from selected text in a code editor, the model context or typed directly into the search field of the control.

This feature is available from Enterprise Architect Release 14.1.

```
188
189 namespace java
190 {
191 //
192 // Find all references
193 //
194 query::findByName($param1)
195 {
196     distinct(GetByValue( $param1 +))
197 }
198
199 query::findMethodByName($name)
200 {
201     move( 1, "METHOD", intersect( GetByNode("NAME"), GetByValue( $name ) )  )
202 }
203
204 query::findMethodCall($name)
205 {
206     filter( "METHOD_ACCESS", intersect(GetByNode("NAME"), GetByValue( $name )) )
207 }
208
```

This image illustrates an mFQL query from the Sparx Queries file distributed with Enterprise Architect installations. The syntax for composing an mFQL query and the mFQL language itself is described here.

Query Syntax

The syntax for composing mFQL queries is:

namespace

{

query:*name*([$*param1* [, $*param2*]])

{

mfql-expression

}

}

where:

- *namespace* names the collection of queries
- *name* is the 'function' name of the query
- *$param1 and $param2* are placeholders for argument substitutions at runtime
- *mfql-expression* is an mFQL expression

代码矿工查询语言 (mFQL)

The Code Miner system provides fast and comprehensive access to the information in existing source code. By parsing all source code and storing the resulting Abstract Syntax Tree (AST) in a read-optimized database, the system provides complete access to all aspects of the original source code, in a machine understandable format.

The core goal behind the system is to provide access to the data hidden within source code in a timely and effective manner. Great pains have been taken to ensure maximal performance, while providing the simplest interfaces possible. As a result the system can be used to analyze program structure, calculate metrics, trace relationships and even perform refactoring.

mFQL

mFQL is the query language of the Code Miner. The language itself is reasonably simple, providing a small number of commands. Simple as the language is, it supports queries of arbitrary size and complexity. The design provides extreme performance for all queries, great and small.

The language is set-based; it operates primarily on sets of abstract data obtained through discrete vertical indices. For our purposes, a set is an ordered array of numbers, each of which is a pointer to a node in the AST Store. A discrete vertical index provides a mechanism to retrieve sets by discrete value.

The language includes the three basic set-joining operations. These are 'intersect', 'union', and 'except'. The 'except' join is, more precisely, a 'symmetric difference' join. A 'complement' join can be achieved by using a short sub-query; this is detailed in the 'except' join documentation. The 'offsetIntersect' join is also discussed in detail there.

The Code Miner database provides three discrete vertical indices in its AST Store. These indices are 'node name', 'attribute name', and 'attribute value'. Each vertical index can be queried for a discrete value, which will return a set of all nodes where that value is present. The three vertical indices are queried using the functions 'getByNode', 'getByName' and 'getByValue', respectively.

Set 'traversal routines' provide mechanisms to filter sets based on patterns in the AST. The traversal routines are either destructive (move) or non-destructive (filter). Destructive traversals modify the set member values to point to the target node; non-destructive traversals ensure the target node exists. In both cases, nodes that cannot complete the traversal are removed.

Please note that all traversals in mFQL are upwards. Downwards traversals are technically complex, as a node could have any number of child nodes. Conversely, upward traversals are much simpler, with every node having zero or one parent node. For these reasons, downward traversals are not supported in the query language.

Although there are only a small number of operations in mFQL, the language is capable of expressing very finely grained and complex queries. The language is functional in design, and supports arbitrary nesting calls.

mFQL queries execute at lightning speed. The backend database was designed from the ground up for read performance. The query parser was hand optimized. Knowing that it always has pure ordered sets, the low-level code takes several shortcuts to perform joins with minimal work effort.

In order to use nBNF effectively one must possess a working knowledge of the target language, and an intimate knowledge of the grammar used to parse it.

mFQL 语言

This section provides a list of Code Miner NBNF Query Language (mFQL) queries with explanations and comments.

The queries shown here demonstrate different capabilities and different approaches to exploring and extracting data using mFQL and the Code Analyzer in Enterprise Architect. The mFQL queries help make the syntax human-readable and intuitive, and have been extended in Enterprise Architect to include additional functions necessary to do real things with Code Miner databases.

The Query Language

String parameters are indicated by **string**, set parameters are indicated by **set** and number parameters are indicated by **numbers**.

Notes

1. Case sensitivity is defined by the case sensitivity of the language of the source code used to populate the database. If the source language is case sensitive (such as C++) all string literal parameters are case sensitive. If the source language is case insensitive (such as SQL) all string literal parameters are case insensitive.
2. Hierarchical traversals in mFQL are generally upwards. Downwards traversals are not optimal, as a node might have any number of child nodes. Upward traversals are much simpler, with every node having zero or one parent node. Downward-looking queries such as 'children' only query one level down.
3. Synonyms of some keywords are provided to better express a query intent or action in particular circumstances, and to support legacy queries. Synonyms are simple alternatives for the base function keyword. For example, 'type(str)' can be written as 'node(str)' or 'byNode(str)' or 'getByNode(str)'. The current specified version is the preferred one, with the synonyms only intended for use in exceptional circumstances.

陈述	描述
type(value)	type(value) Extracts a set based upon node name. The exact name for a node is defined by the grammar used to parse the original source. In this example, find all nodes within the database of type "CLASS". type("CLASS") *Synonyms:* • node • byNode • getByNode
with(name)	with(name) Searches the database for any element that has a named attribute matching the search string. The value of the attribute is ignored - this is a query for the attribute NAME only. All nodes with one or more attributes of the specified name are returned. If a single node has two attributes of the same name, one instance of that node is returned. This example will find all elements in the database that have an attribute named "Type": with("Type") *Synonyms:* • name

	• byName • getByName
find(value) find([+] value [+ value] [+])	find(value) find([+] value [+ value] [+]) Search the database for any element having an attribute value with the provided search term. The match is case sensitive and must match the whole word. You can extract a set based upon an attribute value; when extracting nodes by attribute value, the values of all attributes for the node are considered. Wildcards allow for specifying a subset of attribute values for a node. Wildcards can be used at either the beginning or end of a value specification: • A leading concatenation symbol allows for any number of attributes preceding the first matched attribute • A trailing concatenation symbol allows for arbitrary trailing attributes In both cases, if the node would match without wildcards, it will match with them – the wildcard specifies any number of leading/trailing attributes, including none. In this example, we retrieve a set of nodes that have their last two attributes being "." and "sun". The leading concatenation symbol specifies that any number of attributes (including none), with any value, can exist before the matched attributes, but none can follow. find(+ "." + "sun") The next example has a trailing wildcard. Any node with "com", "." and "sun" as the first three attributes will be returned. Any number of trailing attributes can exist. find("com" + "." + "sun" +) Both wildcards can be used together. In this example nodes with attributes with the three specified values as names, in order, regardless of leading or trailing attributes, will be returned. find(+ "com" + "." + "sun" +) Example: Find all nodes in the database that have any attribute with a value of "CString": find("CString") Example: Find all nodes in the database with a set of attributes having these values in this order: find("com" + "." + "sun") *Synonyms:* • value • byValue • getByValue
has(name,value)	has(name, value) Finds all elements that have a named attribute with the value supplied. Unlike the intersection of 'find' and 'with', this query will only return rows with an exact name/value pair. has("Type","CString")
having(name, value, set)	having (name, value, set) Finds all elements within the supplied set that have a named attribute with the given value. Similar to 'has' but supplies a predefined input set to search. Whether to use

	'has' or 'having' is generally determined by the kind of query structure being used, its depth and its readability. Example 1: Find all Property elements with a name of "m_strName" that have a Type attribute of CString: having("Type","CString",this("PROPERTY","NAME","m_strName")) Example 2: Extend Example 1 to only include those that store a CString *: having("Reference","*", having("Type","CString",this("PROPERTY","NAME","m_strName")))
this(type,name,value)	this(type, name, value) Function finds one or more elements that have a matching TYPE, and WITH a named attribute having the specified VALUE. Example: Find all operations named "Import Solution": this("OPERATION","NAME","ImportSolution") *Synonyms:* • object • item
like(name,like,set)	like(name, like, set) Finds a set of elements that have an attribute that starts with the search sub-string. Note that this is not a fully wild-carded search but is case sensitive and must be an exact match for the length of the search string. Example: Find all Classes in the database whose NAME attribute starts with "CMapStr": like("NAME","CMapStr",gettype("CLASS"))
and(set1,set2,...)	and(set1, set2, ...) Returns the intersection of nodes between two or more sets. To be included in the final set, an element must exist in ALL the input sets. *Synonyms:* • intersect(set, set,...) • {set, set, ...}
union(set1,set2,...)	union(set1,set2, ...) Returns the distinct union of ALL nodes present in the input sets. *Synonyms:* • or(set, set ...) • [set, set ...]
ancestor(str,set)	ancestor(str, set) ancestor(num, set) ancestor(num, str, set) The ancestor function traverses each node in a set of a number of parent nodes,

	excluding any nodes that fail the traversal. The number of nodes to traverse, the name of the target node for the traversal, or both can be provided as parameters. When the number of nodes is provided, but the target node name is not, any nodes with the specified number of parents will pass the traversal. Any node that runs out of parents will be dropped from the set. When the name of the target is specified, but the number of nodes to traverse is not, any nodes with a parent with a matching name, at any point in the hierarchy, will pass the traversal. Any node with no matching parent is excluded. When both the number of nodes and the target name are provided, only nodes that have a parent node with the specified name, at the specified offset, pass the traversal. All other nodes are removed from the set. In this example the set hasParameter("CString","&",1) is moved up to an ancestor node named "OPERATION". If the move fails the node is dropped from the result. ancestor("OPERATION",hasParameter("CString","&",1)) In this example the set is moved up one rung to its parent. If there is no parent, the node is dropped from the result. ancestor(1,hasParameter("CString","&",1)) In this example the set is moved up three steps to its parent->parent->parent . If there is no such node, the node is dropped from the result. ancestor(3,hasParameter("CString","&",1)) *Synonyms*: • move
filter(str,set)	filter(str, set) filter(num, set) filter(num, str, set) The filter function is the same as the 'ancestor' function, except that it returns nodes from the original child set rather than new ancestor nodes. If a node is unable to pass the specified traversal, it is removed from the set. Nodes that pass the traversal are left in place, unmodified. In this example the set hasParameter("CString","&",1) is tested for an ancestor node named "OPERATION". If the move fails the node is dropped from the result. The result set is a set of parameter types that meet the criteria. filter("OPERATION",hasParameter("CString","&",1))
match(NameA,setA,NameB,setB)	match(NameA,setA, NameB,setB) 'Match' takes two input sets and two attribute names and returns all those in 'setA' that have a matching record in 'setB', as determined by comparing the values of the named attributes 'strA' and 'strB'. That is, a 'setA' row is included if the value of attribute 'strA' in 'setA' exists in 'setB' as the value of an attribute of name 'strB'. 'Match' is useful for finding where one element feature is used in a different context elsewhere in the database. For example, where a unique element name or GUID is referenced by another element. In this example, we match the attribute named 'TYPE' from the right set to the attribute 'NAME' in the left set. The result will be all "CLASS" type objects from the left set with NAME == TYPE(s) as specified in the right set. match("NAME",type("CLASS"),"TYPE",this("PROPERTY","NAME","m_pLink"))

graph(targetType, targetName, linkType, linkName, start)	graph(targetType, targetName, linkType, linkName, start) Find a recursive set of elements that form some kind of graph when linked by attribute pairs, in a manner similar to 'match'. The starter set is queried for all owned instances of the linkType with link Name and these are matched against a new query based on the targetType with targetName. The new set is filtered in a manner similar to 'match', and all elements in the new query that share the same NAME/VALUE pair as from the starter set are kept; all others are discarded. The resultant set is then fed back into the original set as the starter for the next iteration, with the results at each stage being added together to form the final result set. Example: Return the Class hierarchy for a Class named "Car". graph("CLASS","NAME","GENERALIZATION","GENERAL",this("CLASS","NAME","Car"))
prune(set_test,str,set_base)	prune(set_test, str, set_base) prune(set_test, num, set_base) For two sets of nodes, temporarily move one set UP to the named or numeric position in its ancestry and filter out any nodes that do not exist by strict intersection in the TEST set. The first set is the TEST set, the right or last set is the BASE set. The set returned is all the elements in the BASE set that, when moved to the TEST position, matched something in the TEST set. The returned nodes are the original nodes from the BASE set and are not moved up when returned. Example 1 finds the set of parameter types used for operation parameters named "CustomerName" across the whole database. prune(this("PARAMETER","NAME","CustomerName"),"PARAMETER",type("PARAMETERTYPE")) Example 2 finds all Properties of a Class named Customer, assuming the grammar used to compile the database placed the Property definition two hierarchy levels below the Class definition. prune(this("CLASS","NAME","Customer"),2,type("PROPERTY"))
andat(str,test,base)	andat(str, base, test) andat(num, base, test) andat(num, str, base, test) For two sets of nodes, temporarily move one set UP to the named or numeric position in its ancestry and filter out any nodes that do not exist by strict intersection in the TEST set. The first set is the TEST set, the right or last set is the BASE set. The set returned is all the elements in the BASE set that, when moved to the TEST position, matched something in the TEST set. The returned nodes are the original nodes from the BASE set and are not moved up when returned. Similar to 'prune', this query supports additional options and structures the inputs in a different order to facilitate different kinds of stacked searches. The 'andat' function performs both a non-destructive tree traversal and an intersect join in one operation. Each node in the left set is traversed according to parameters provided, then the result of the traversal is intersected with the right set. If the intersect passes, the original node is added to the result set. If the intersect fails, the node is excluded from the result set. The traversal parameters for 'andat' are the same as for 'ancestor' and 'filter'. For more information about the traversal parameters, see the 'ancestor' function.

	Example: For the set of all "PROPERTY" nodes in the database, move them up to a parent node of type CLASS and then intersect the result with the right hand set - in this case a CLASS named CDiagram. All nodes that pass this test are returned as PROPERTY nodes, effectively giving the set of all properties of the Class CDiagram. andat("CLASS",type("PROPERTY"),this("CLASS","NAME","CDiagram")) *Synonyms:* • offsetIntersect • offsetx
unique(left,right) / except(left,right)	unique(left, right) except(left, right) Except joins return sets that contain any nodes from either set that do not appear in both sets. This join is similar to a bitwise XOR operation. In set theory, this type of join is referred to as a 'symmetric difference join'. {1, 2, 3} excepted with {2, 3, 4} results in {1, 4}
omit(left,right) / exclude(left,right)	omit(left, right) exclude(left, right) Exclude joins return a set that contains all nodes from the left set that do not appear in the right set. In set theory, this type of join is referred to as a 'relative complement join'. {1, 2, 3} complemented with {2, 3, 4} results in {1}
differ(name,set,name,set)	differ(name, set, name, set) Return a set of nodes that do not have a matching row in another set, using a NAME/VALUE pair from each set to match on. Example: This more complex example tests the complete set of Generalizations for a Class hierarchy and identifies missing or unresolved Class names in the total inheritance hierarchy. Like the 'match()' function discussed later, this function iterates over attribute name/value pairs as specified in the left and right input sets, but only includes rows in the final set where there is NO match. differ("GENERAL", children("GENERALIZATION", graph("CLASS","NAME","GENERALIZATION","GENERAL", this("CLASS","NAME","CMainFrame"))), "NAME", graph("CLASS","NAME","GENERALIZATION","GENERAL", this("CLASS","NAME","CMainFrame")))
children(type,set)	children(type, set) Return a set of child nodes of a specified type for one or more parents in the source set. For all children regardless of type, use an empty string. For example, in the first query we return ALL first level children of the CMainFrame Class. In the second query we restrict the nodes returned to be of type

	"REGION" only. children("",this("CLASS","NAME","CMainFrame")) children("REGION",this("CLASS","NAME","CMainFrame"))

childcount(num,type,set)	childcount (num,type,set)
	Return nodes that exactly match the number of specified children of a specified type. For example, only return operations that have 5 parameters.
	An example usage is in specifying an exact operation signature, so we check firstly that parameter1 and parameter2 match the type we are querying for, then move those to their operation ancestor and intersect the result with the operation name "GetFromCache" we are interested in. To rule out spurious hits with operations having more than 2 parameters, we explicitly add childcount(2, ...) to ensure we only get operations that have 2 parameters.
	
	childCount(2,"PARAMETER",
	{
	ancestor("OPERATION",hasParameter("CString","&",1)),
	ancestor("OPERATION",hasParameter("CString","&",2)),
	this("OPERATION","NAME","GetFromCache")
	}
	)

byAddress(num)	byAddress(num) The byAddress function is used in applying the results of one query to another. For example, we might have a node of particular interest, and want our query to return only nodes that join (in some way) to the specified node. byAddress(node: number) This example builds a set containing the single node related to the address specified: byAddress(11256)
byPosition(File, Offset)	byPosition(File, Offset) The byPosition function is used to return the inner-most node that covers a certain position in a file. This function is useful for locating a position in the AST based upon a file position.
distinct(set)	distinct(set) The distinct function ensures that a set has no duplicate values. All duplicate values are excluded from the result set.

集提取

These procedures extract sets from discrete vertical indices. There are three indices available, each with a specific extraction function. String literal parameters to these functions could be case sensitive. Case sensitivity is defined by the language of the source code used to populate the database. If the source language is case sensitive (as C++ is) all string literal parameters are case sensitive. If the source language is case insensitive (as SQL is) all string literal parameters are case insensitive.

type

type(value: string)

Extract a set based upon a node name. The exact name for a node is defined by the grammar used to parse the original source. In this example, all nodes with the name "OPERATION" are returned.

type("OPERATION")

–

with

with(value: string)

Extract a set based upon attribute name. All nodes with one or more attributes of the specified name are returned. If a single node has two attributes of the same name, one instance of that node is returned. This example returns all nodes with one or more attributes named "NAMEPART".

with("NAMEPART")

find

find([+] value: string [+ value: string] [+])

Extract a set based upon an attribute value. When extracting nodes by attribute value, the value of all attributes for the node are considered. Wildcards allow for specifying a subset of attribute values for a node.

When a single value is provided, all nodes that have a single attribute with the value specified are returned. If a node has any other attributes, it is excluded. In this example, all nodes with exactly one attribute with the value of 'i' are returned.

find("i")

More than one value can be specified by using a concatenation symbol. When more than one value is specified, the resulting set will contain all nodes that have attributes with exactly the values specified, in the order specified. Any node with extra leading or trailing attributes is excluded. This example retrieves a set of all nodes with a set of three attributes with the values “com”, “.” and “sun”, in that order.

find("com" + "." + "sun")

Wildcards can be used at either the beginning or end of a value specification. A leading concatenation symbol allows for any number of attributes preceding the first matched attribute. A trailing concatenation symbol allows for arbitrary trailing attributes. In both cases, if the node would match without wildcards, it will match with them – the wildcard specifies any number of leading/trailing attributes, including none.

In this example, we retrieve a set of nodes that have their last two attributes being “.” and “sun”. The leading concatenation symbol specifies that any number of attributes (including none), with any value, can exist before the matched attributes, but none can follow.

find(+ “.” + “sun”)

The next example has a trailing wildcard. Any node with attributes “com”, “.” and “sun” as the first three attributes will be returned. Any number of trailing attributes can exist.

find(“com” + “.” + “sun” +)

Both wildcards can be used together. In this example, nodes with attributes named as the three values specified, in order, regardless of leading or trailing attributes, will be returned.

find(+ “com” + “.” + “sun” +)

—

设置遍历

ancestor

ancestor(count: number, source: set)

ancestor(value: string, source: set)

ancestor(count: number, value: string, source: set)

The 'ancestor' function traverses each node in a set up a number of parent nodes, excluding any nodes that fail the traversal. The number of nodes to traverse, the name of the target node for the traversal, or both can be provided as parameters.

- When the number of nodes is provided, but the target node name is not, any nodes with the specified number of parents will pass the traversal; any node that runs out of parents will be dropped from the set
- When the name of the target is specified, but the number of nodes to traverse is not, nodes with a parent with a matching name at any point in the hierarchy will pass the traversal; any node with no matching parent is excluded
- When both the number of nodes and the target name are provided, only nodes that have a parent node with the specified name at the specified offset pass the traversal; all other nodes are removed from the set

It is possible - even likely - that these calls will generate sets having duplicate values. This is by design, as the concrete rules for sets do not define them as being discrete. If (as in most cases) you want your set to be discrete, use the 'distinct' function described in the *The mFQL Language* Help topic.

This sample extracts a set of all nodes named 'OPERATION', then traverses each node up one level to its immediate parent. Any 'OPERATION' node with no parent is excluded.

ancestor(1, getByNode("OPERATION"))

This sample extracts a set of all nodes named 'OPERATION', then traverses each node up to the first 'CLASS' parent node. Any 'OPERATION' node with no 'CLASS' parent is excluded.

ancestor("CLASS", getByNode("OPERATION"))

This sample extracts a set of all nodes named 'OPERATION', then traverses each node up one level to its immediate parent. If the parent node is not a 'CLASS' node, or the node fails to traverse though a lack of parent nodes, it is excluded.

ancestor(1, "CLASS", getByNode("OPERATION"))

–

filter

filter(count: number, source: set)

filter(value: string, source: set)

filter(count: number, value: string, source: set)

The 'filter' function is the same as the 'ancestor' function, except that it does not modify nodes – it is non-destructive. If a node is unable to pass the specified traversal, it is removed from the set. Nodes that pass the traversal are left in place, unmodified.

It is often desirable to filter a set by the current node name. This can be used to ensure that the nodes returned from a 'with' or 'find' call are of a particular node type. This example returns all nodes with an attribute with the value of “CFoo”, where the resulting node is a “TYPE” node.

filter(0, “TYPE”, find(“CFoo”))

For more details on the use of the 'filter' function, see the 'ancestor' function.

—

设置加入

and

and(left: set, right: set)

An 'and' join will return a set containing all nodes that exist in both the left and right set. This join is comparable to a bitwise AND operation. In set theory, this type of join is called an 'intersection'.

{1, 2, 3} intersected with {2, 3, 4} results in {2, 3}

This example returns a set that contains all nodes that have a single attribute with the name of "TYPE" and the value of "int".

```
and(
find("int"),
with("TYPE")
)
```

union

union(left: set, right: set [, right: set])

'Union' joins return a set that includes all nodes found in either the left or the right set. This join is used to combine the results of two or more sub-queries into a single set. A 'union' join is similar to a logical OR operation. In set theory, the 'union' join is known as a union.

The 'union' join is able to operate on more than two sets. The result is a set that contains all nodes from all supplied sets. The 'union' join is the only join able to operate on more than two sets.

The result of a 'union' join is always a discrete set, unless one of the source sets contained duplicates. This means that duplicates in source sets will be preserved, but the 'union' join itself will not generate duplicates.

{1, 2, 3} unioned with {2, 3, 4} results in {1, 2, 3, 4}

This sample creates a set containing all nodes with an attribute named “TYPE” or a single attribute with the value of “int”.

```
union(
find("int"),
with("TYPE")
)
```

except

except(left: set, right: set)

'except' joins return sets that contain any nodes from either set that do not appear in both sets. This join is similar to a bitwise XOR operation. In set theory, this type of join is referred to as a 'symmetric difference' join.

{1, 2, 3} excepted with {2, 3, 4} results in {1, 4}

For more information on the 'symmetric difference' join in set theory, see https://en.wikipedia.org/wiki/Symmetric_difference

This sample returns a set of all nodes with an attribute named "TYPE" but no single attribute with the value of "int", plus all nodes with an attribute with the value of "int" that are not named "TYPE".

```
except(
 find("int"),
 with("TYPE")
)
```

exclude

exclude(left: set, right: set)

'exclude' joins return a set that contains all nodes from the left set that do not appear in the right set. In set theory, this type of join is referred to as a relative complement join.

{1, 2, 3} complemented with {2, 3, 4} results in {1}

This sample returns a set of all nodes with a value of “int” that are not “TYPE” nodes:

```
Exclude(
find(“int”),
with(“TYPE”)
)
```

andat

andat(count: number, left: set, right: set)

andat(value: string, left: set, right: set)

andat(count: number, value: string, left: set, right: set)

The andat function performs both a non-destructive tree traversal and an intersect join in one operation. Each node in the left set is traversed according to parameters provided, then the result of the traversal is intersected with the right set. If the intersect passes, the original node is added to the result set. If the intersect fails, the node is excluded from the result set.

The traversal parameters for andat are the same as for 'ancestor' and 'filter'. For more information about the traversal parameters, see the 'ancestor' function described in the *Set Traversal* Help topic.

This sample takes all “NAME” nodes, traverses them up one parent, and intersects them with a set of all “CLASS” nodes. If a “NAME” node passes both the traversal and intersect join, it is added to the result set. The result is a set of all “NAME” nodes whose immediate parent is a “CLASS” node.

```
andat(1,
type(“NAME”),
type(“CLASS”)
)
```

—

Sparx 英特尔服务

Sparx 英特尔服务计划为开发项目和参与者提供了一种方法，可以深入了解他们正在使用的代码库和软件框架。该服务充当Enterprise Architect客户的提供商，允许访问智能感知代码编辑和搜索工具中的有见地的搜索结果

Sparx Intel 服务是 Sparx卫星服务伞的一部分。该服务可以在运行网络上运行，也可以在云上运行 Microsoft窗口。运行英特尔卫星窗口服务可以作为服务安装或作为独立进程运行。该服务允许多个Enterprise Architect客户端访问和查询来自许多不同软件域和框架的相同信息。

此特征可从Enterprise Architect版本 16.0 获得

Sparx 英特尔服务配置

SparxIntelService.exe 程序为Enterprise Architect运行一项或多项英特尔服务。该程序与Enterprise Architect位于相同的安装文件夹中，它使用一个配置文件来运行可以在本地计算机上运行的服务。

在本主题的示例中，程序将尝试使用文件c:\mystuff\myservices.config。它会寻找一个名为*EA*的服务，如果找到，就启动它I

SparxIntelService.exe监听服务= EA config= c:\mystuff\myservices.config

配置文件格式

The configuration file has this format:

 # comment

 # comment

 # comment

{ # start of service definition

... # list of directives as pairs

} # end of service definition

{ # start of service definition

... # list of directives as pairs

} # end of service definition

Comments are indicated by the # character.

If the config directive is omitted (not recommended), the program will look for a config file of the same name as the program, in the same directory as the program.

In this example the program will attempt to use the file SparxIntelService.config in the same folder:

SparxIntelService.exe listen service:EA

指示	描述
姓名	在命令行上命名服务时，将启动具有匹配名称属性的服务。
状态	当状态=ON时，服务将被启动；否则，将不会启动。
懒加载	当lazyload为 "true" 时，任何代码矿工数据库都将延迟加载，直到向服务发出英特尔请求。
日志级别	将记录的信息级别定义为由 |" 分隔的关键字 {information,warning,error} 的组合。例如： loglevel=信息|警告|错误
日志输出	指定要写入的log文件的完整路径名。例如： 日志输出=c:\logfiles\intel-service-project1。 log
数据库	指定要加载的代码矿工数据库的全路径名。例如： 数据库=c:\intel--service\project1.cdb 允许多个 "数据库" 指令，每个指令指定一个不同的数据库。

允许	标识允许连接到端口上的服务的 IP 地址。例如： 允许=本地主机 允许=127.0.0。 1 allow=172.160.*（当 'network' 允许使用通配符 指令的值为 网络"或 公共"， 但不是 本地"）
网络	允许限制服务连接。 ● 本地 - 服务不会侦听除localhost以外的任何连接 ● 网络 - 当与通配符 允许"指令一起使用时，允许使用允许的 IP 地址通配符的客户端进行连接 ● 公共 - 允许任何连接
节目	当为 真"时，将显示服务的控制台窗口；默认值为 false"。
港口	服务将侦听的端口。

服务配置模板

When choosing the 'Execute > Tools > Services > Code Miner Service > Edit Configuration File' ribbon option you display the Windows 'Save As' browser through which you can choose either the config file to open or where a file should be created.

If no config file is recorded in the registry and you specify a non-existent filename, that file is created, filled with a 'bare bones' configuration skeleton and saved. The selected/new configuration is then shown in the Enterprise Architect default editor.

The 'bare bones' template is shown here.

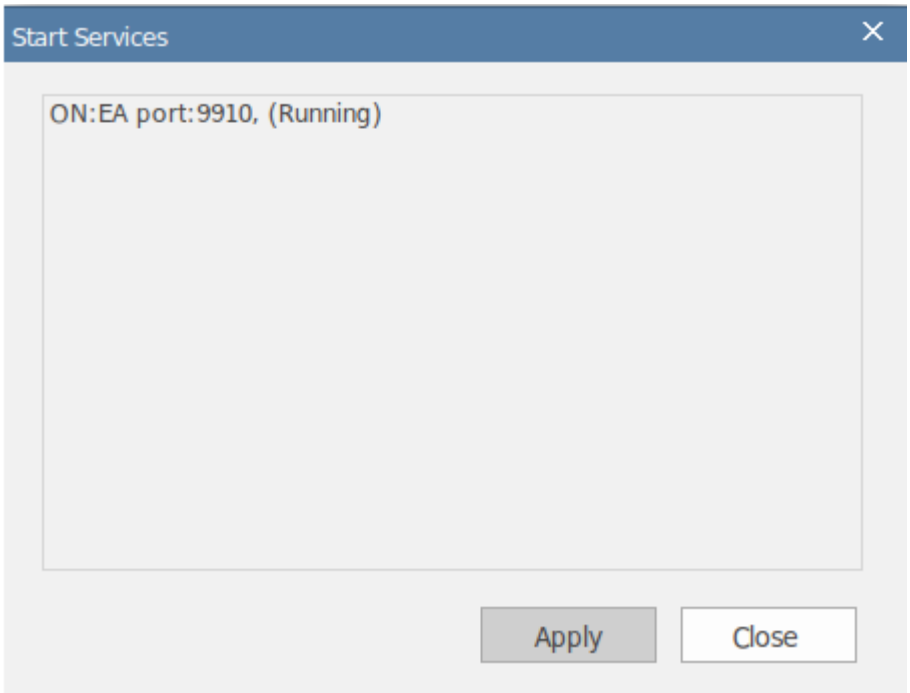
```
#-------------------------------------------------------------------------------
# Sparx Intel Service Configuration File
# -------------------------------------------------------------------------------
# This file is used to describe one or more intel services and the code miner databases that they support
# This file can be used in EA to manage a number of services on the local machine
# -------------------------------------------------------------------------------
# Service Attributes
# -------------------------------------------------------------------------------
# name                    The unique name of the service in this file
# status                  "ON" - service can run, "OFF" service will never run
# lazyload                "true" - databases are loaded n demand, "false" - databases are loaded when service starts
# port                    Unique Port number that service will listen on and EA will connect to
# network                 [optional,default=local] Restricts service to listening to locahost only (local), to a range of addresses (network) or any address (public)
# allow                   Allows a specific IP address or wildcard IP address to connect (if network is NOT local)
#                         (There can be multiple allow directives present)
```

```
# autoupdate          "true" - will detect updates to listed databases and reload them, "false" default, changes are not detected
# show                [optional,default=false] shows the console window for the service
# logoutput           [optional] The path of a log file which service can write to
# loglevel            [optional] The levels of information logged. Combine with '|' character, e.g.: { information|warning|error }
# database            [Required] The full path to a codeminer database which usually has the .cdb file extension
#                         (There can be multiple database directives present)
#
# ---------------------------------------------------------------------------
# Attribute Values
# ---------------------------------------------------------------------------
#
# <string>   - text. (do not include quotes)
# <boolean>  - text, { true, false, ON, OFF }
# <path>     - fully specified file path to codeminer database
# <number>   - digits
# ---------------------------------------------------------------------------
#
{
        name=<string>,
        status=<boolean>,
        lazyload=<boolean>,
        port=<number>,
        allow=<string>,
        allow=<string>,
        network=<string>,
        autoupdate=<string>,
        show=<boolean>,
        logoutput=<string>,
        loglevel=<string>,
        database=<path>,
        database=<path>,
        database=<path>
},
{
        name=Project1,
        status=ON,
        lazyload=TRUE,
        allow=localhost,
        allow=127.0.0.1,
        port=9999,
```

```
        autoupdate=true,
        database=c:\Project1\Project1.cdb
}
```

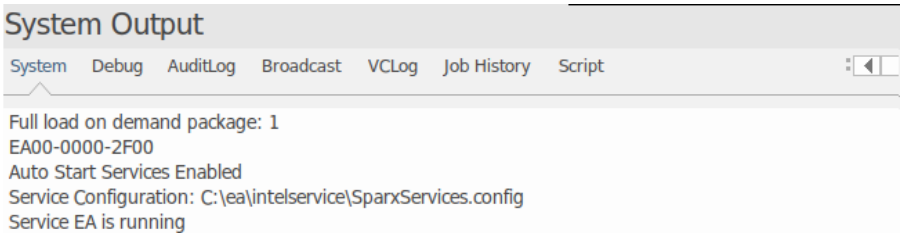
Sparx Intel 服务功能区选项

当服务配置文件存在时，您可以使用代码矿工菜单选项组中 执行 > 工具 > 服务"功能区选项中的许多选项来编辑或执行它。

选项	描述
视图所有服务的状态	（在所有服务类别之上。）此选项显示一个视图，其中列出了在当前配置文件中状态的每个Enterprise Architect服务的状态及其状态。
开始	该选项读取当前的Service配置文件，启动配置为运行的服务，停止运行未配置为运行的运行。 A以下情况下配置服务： 1. 它在配置文件中命名。 2. 它具有属性状态：ON。 
停止所有	此选项停止当前正在运行的所有服务。
编辑配置文件	此选项提示要使用的服务配置文件，然后在Enterprise Architect文本编辑器中打开该文件。系统会记住文件所在的位置。

<table>
<tr><td></td><td>

```
31 # <number>   - digits
32 # ----------------------------------------------------------
33 #
34 {
35     name=project1,
36     status=ON,
37     lazyload=true,
38     port=9910,
39     allow=localhost,
40     network=local,
41     autoupdate=true,
42     show=true,
43     logoutput=c:\My Documents\project1.txt,
44     loglevel=information|warning|error,
45     database=c:\My Documents\project1\project1.cdb
46 }
47
```

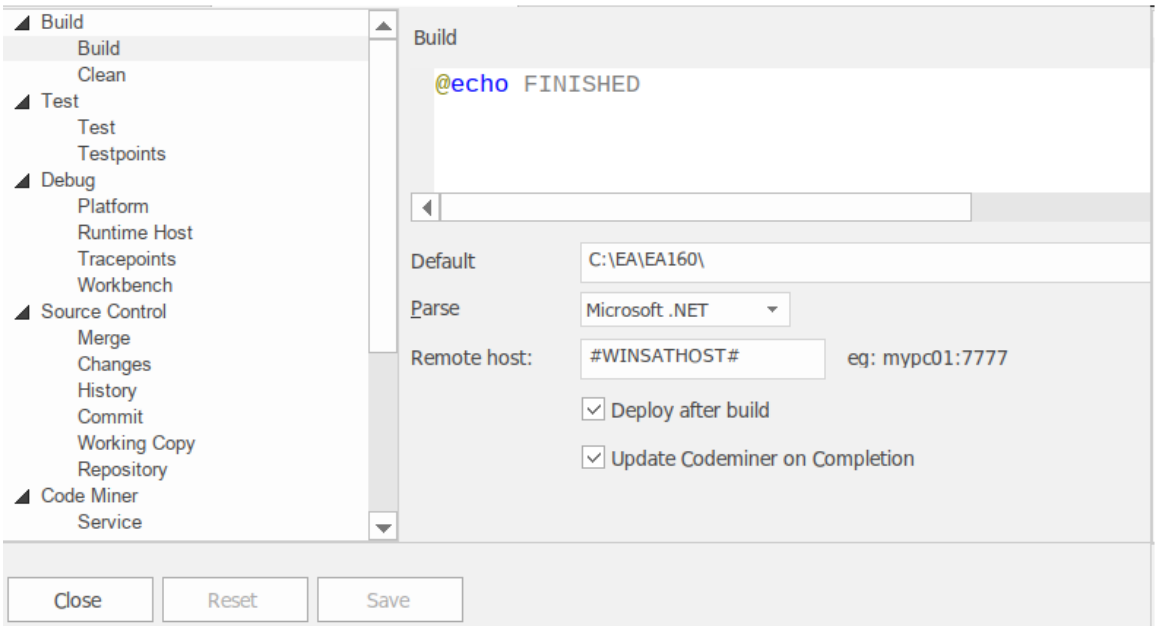
</td></tr>
<tr><td>自动开始使用 EA</td><td>此选项会在模型打开时自动启动具有 状态：ON”属性的服务。
 

当模型打开时，此处记录到系统输出窗口的消息表明该服务已经在运行。</td></tr>
<tr><td>关闭时自动停止</td><td>此选项在Enterprise Architect关闭时自动停止运行服务。</td></tr>
</table>

Sparx Intel 服务自动更新

当你执行分析器脚本编译命令时，一个作业被添加到作业队列中。

如果编译脚本在分析器脚本编辑器中勾选了 "Update分析器on Completion"复选框，则会在作业中添加一个额外的任务来更新脚本中列出的每个 Codeminer 数据库。



这些库可以在代码矿工| 中看到。脚本的库部分。

任务如何运行

代码矿工更新任务运行带有两个参数的程序SSCodeMiner.exe 。

第一个参数指定要在其上执行增量构建的数据库，并具有以下形式：

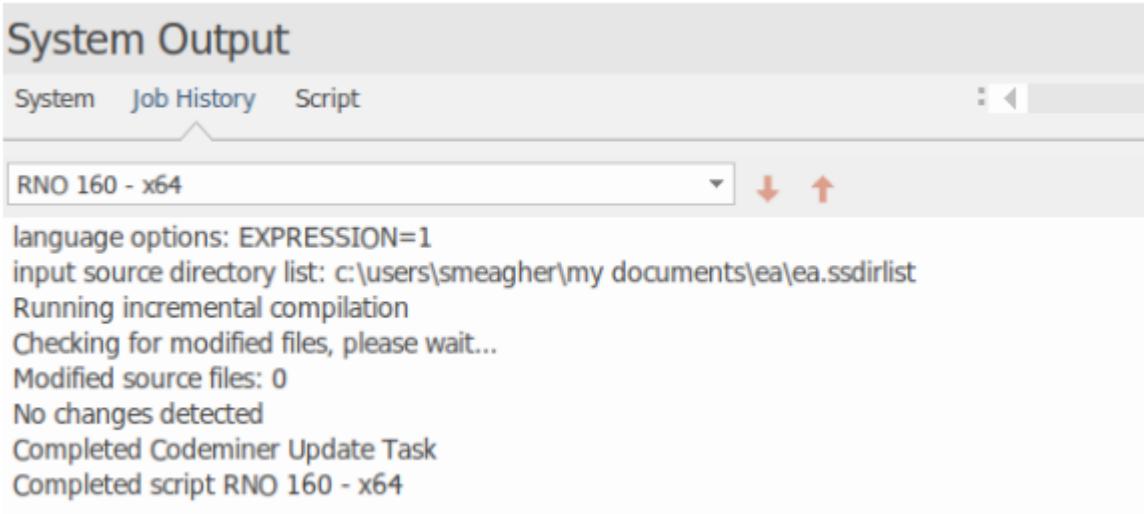
更新="c:\path\ea.cdb"

第二个参数是可选的，指定编译数据库时使用的辅助宏语法文件；它有这种形式：

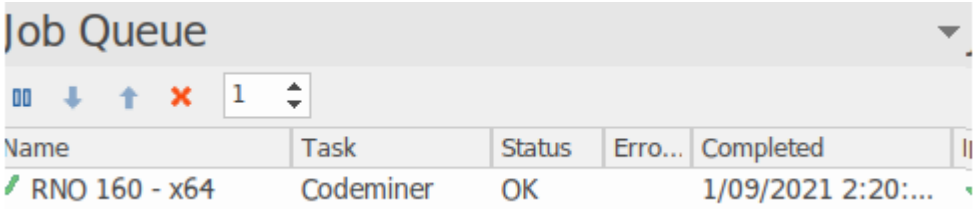
宏 =“c:\ea\ea160\config\CodeMiner\SparxProjectMacros.nbnf”

工作输出

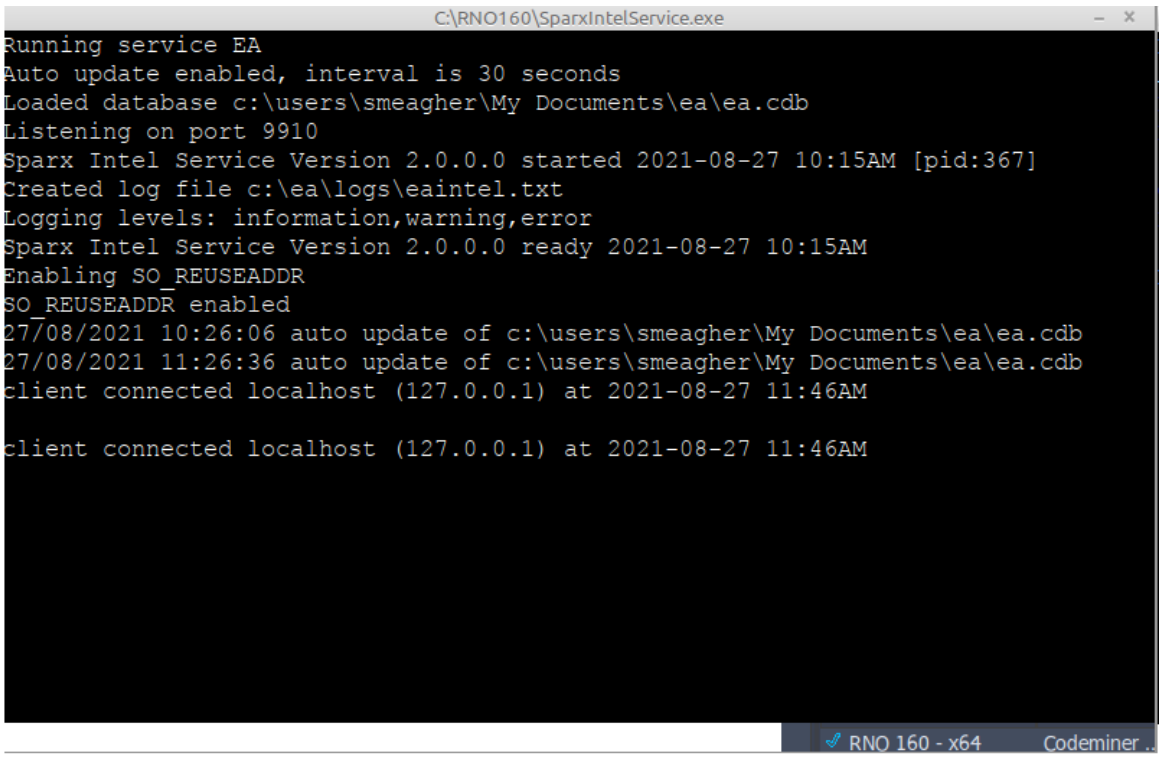
当代码矿工更新任务运行时，捕获的 SSCodeMiner.exe 更新过程的输出将发送到系统输出窗口的 作业历史"选项卡，其格式与手动更新代码矿工数据库时显示的格式相同。 Enterprise Architect 。在这个图中我们可以看到解析器脚本RNO 160分析器已经成功完成。



作业队列"窗口显示作业已完成。运行的最 一项任务是代码矿工运行。



作业历史"选项卡显示没有源代码文件已更改。如果检测到修改 的源代码更改 - 也就是说，代码矿工服务检测到代码矿工的新版本并自动更新它 - 将显示以下信息：



服务配置

服务方案

服务程序的名称是 SparxintelService.exe。

配置文件

该服务由文件 SparxIntelService.config 配置。
该文件必须与服务程序位于同一目录中。
该文件包含许多指令，还列出了要服务的代码矿工数据库。
该文件在服务启动时被读取一次。

指令	描述
港口	服务将侦听的端口号。
允许	命名允许访问的域或 IP 地址：198.* 或 127.0.0。 1
网络	值可以是 ″public″、 ″network″或 ″private″。 ● 当允许指令指定一个或多个单个 IP 地址时使用 ″private” ● 当允许指令指定通配符域时使用 ″network”：198* ● 使用 ″public″允许所有客户端
数据库	命名服务器上代码矿工数据库的完整物理文件路径。

独立运行程序

从普通控制台输入命令：SparxIntelService -listen

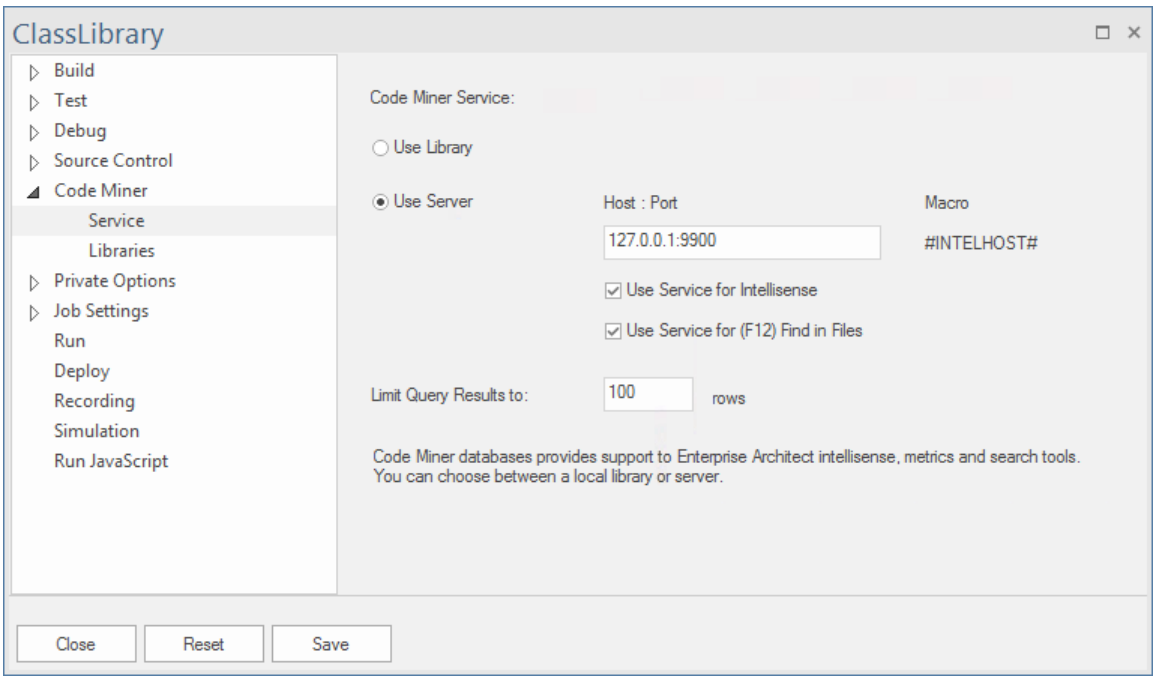
```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\sparxsys>cd C:\servers

C:\servers>SparxIntelService -listen
Listening on port 9000
Loaded database e:\codeminer\jdk1.cdb
Loaded database e:\codeminer\bcgsoft10.cdb
Loaded database e:\codeminer\atlmfc.cdb
```

作为窗口服务安装

从管理控制台输入命令：SparxIntelService -install

客户端配置- 将Enterprise Architect配置为使用代码矿工服务

Enterprise Architect使用称为分析器脚本的组件来配置许多支持系统。这是指定服务器位置的地方。此图显示了脚本的 代码矿工服务"页面。



访问

功能区	开发>源代码>执行分析器>编辑分析器脚本>双击脚本>代码矿工>服务

配置字段

使用服务器	选择此单选按钮以设置要使用的代码矿工服务器。
主机：端口	类型在服务将通过的端口的数量。
为智能感知使用服务	选中该复选框以使用英特尔服务完成智能感知。
使用Service for [F12] Find in Files	如果您想使用服务而不是在文件中查找窗口来运行搜索查询，请选中该复选框，当您按下 F12 时。
将查询结果限制为行	类型在每页显示的查询结果的行数。

节省	单击此按钮以保存您输入的配置详细信息。