Code Template Framework Guide

by SparxSystems Japan

Enterprise Architect 日本語版

コードテンプレートフレームワーク 機能ガイド

振る舞い図からのコード生成編

(2025/01/30 最終更新)

目次

1 はじめに 3
2 振る舞い図からのソースコード生成 3
3 追加されているテンプレートとマクロ 4
4 EASL マクロ 5
4.1 EASL_GET 6
4.2 EASLList 7
4.3 マクロの活用の基本ルール 8
5 テンプレートの呼び出し 8
5.1 引数の渡し方と受け取り方 8
5.2 サンプル 9
6 既定のテンプレート 11
6.1 シーケンス図・アクティビティ図からのコード生成のためのテンプレート 11
6.2 ステートマシン図からのコード生成のためのテンプレート 11
6.3 ビジネスルールからのソースコード生成のためのテンプレート 12
6.4 属性としてのプロパティ 13

Copyright (c) 2010-2025 Sparx Systems Japan

1 はじめに

Enterprise Architect には、コードテンプレートフレームワーク(以下 CTF と表記します)と呼ばれる新しい概念・機能が搭載されています。このドキュメントでは、上位エディションで利用可能な、振る舞い図からのソースコード生成に関するカスタマイズの情報を説明します。

2 振る舞い図からのソースコード生成

ユニファイド版とアルティメット版では、振る舞い図からのソースコード生成が可能です。この場合の振る舞い図とは、以下の 3 つの種類のダイアグラム(図)です。

- ステートマシン図
- シーケンス図
- アクティビティ図

対応する言語は、C 言語, C++, Java, C#, VB.NET です。C 言語は、「オブジェクト指向のサポート」が有効(設定値が true)な場合のみの対応です。

また、ユニファイド版とアルティメット版で利用可能な、ビジネスルールからのソースコードの生成機能でも、このドキュメントで説明する拡張機能を利用しています。この機能は、Java, C#, VB.NET に対応しています。

この 3 つのダイアグラムについて、既定のテンプレートでどのようにソースコードに出力されるか、については、サンプルプロジェクトで実際に生成してご確認ください。手順は以下のとおりです。

1. 「ホーム」リボン内の「ヘルプ」パネルにある「ヘルプ」ボタンを押して「サンプルプロジェクトを開く」を選択します。
2. モデルブラウザで「ソフトウェアエンジニアリング」→「Java モデル(振る舞い定義あり)」と遷移して、「振る舞い図からのソースコード生成」ダイアグラムを開きます。
3. ダイアグラム上で振る舞い図を持つクラスを選択してソースコード生成をします。

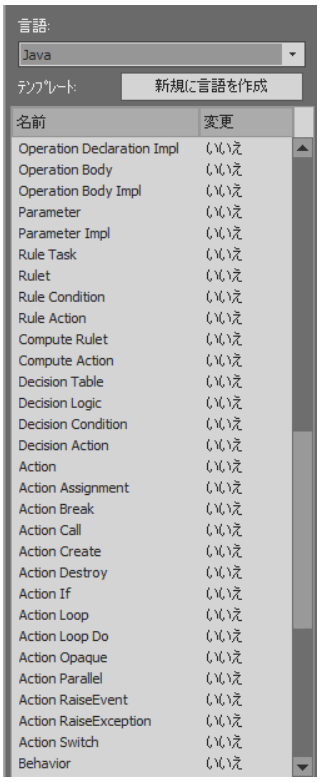
モデルの構造などにつきましては、ヘルプの「振る舞いモデルからのソースコード生成」のページをご参照ください。

Copyright (c) 2010-2025 Sparx Systems Japan

本ドキュメントでは、これらの 3 つの図と 1 つの機能で共通に利用されている、ソースコード生成テンプレートの拡張部分についての概要を説明します。出力される内容をカスタマイズするためには、このドキュメントで説明している基礎知識が必要となります。

3 追加されているテンプレートとマクロ

ユニファイド版あるいはアルティメット版でコード生成テンプレートエディタを開くと、コーポレート版までのエディタよりも多くのテンプレートが表示されます。



多くのテンプレートが追加されていますが、追加されている全てのテンプレートは、コーポレート版までのテンプレートとは異なり、「テンプレートの種類」はありません。つまり、「新規テンプレートの追加」ボタンを押したときに表示される画面で「対象の種類」を「<None>」に設定した場合と同じ種類のテンプレートです。

このドキュメントで説明している範囲について独自に拡張する場合も、「対象の種類」を「<None>」にしたテンプレートを追加することになります。

追加されているテンプレートは、「Class Body」テンプレート(および、言語によっては

Copyright (c) 2010-2025 Sparx Systems Japan

「Class Body Impl」テンプレート)から呼び出されています。呼び出し関係を把握したい場合には、このテンプレートから動作を追跡します。以下は呼び出している部分の抜粋です。

```
$StateMachineGUID = %EASL_GET("Property", classGUID, "StateMachine")%
$sProp = %EASLList="Property" @separator="¥n" @indent="¥t"
@owner=classGUID @collection="Properties"%
$sBehaviors = %EASLList="Behavior" @separator="¥n" @indent="¥t" @owner=
classGUID @collection="Behaviors"%
```

(中略)

```
%if $sProp != ""%
	/* Begin - EA generated code for Parts and Ports */
$sProp
	/* End - EA generated code for Parts and Ports */
%endIf%

%if $sBehaviors != ""%
	/* Begin - EA generated code for  Activities and Interactions */
$sBehaviors
	/* End - EA generated code for  Activities and Interactions */
%endIf%

%if $StateMachineGUID != ""%
	/* Begin - EA generated code for StateMachine */
%StateMachine()%
	/* End - EA generated code for StateMachine */
%endIf%
```

この例では、EASL_GET や EASLList のマクロが、今回説明する機能拡張部分になります。また、%StateMachine()%のように、追加されたテンプレートを直接呼び出している部分も追加されています。

4 EASLマクロ

振る舞い図からのソースコード生成などの機能を実現するために利用可能になっている

Copyright (c) 2010-2025 Sparx Systems Japan

追加機能が「EASL マクロ」です。

この EASL マクロは EASL の名前で始まるマクロであり、いくつかの種類があります。ただし、ほとんどの場合には、次の 2 つのマクロのみを利用します。

4.1 EASL_GET

このマクロは、対象の要素の名前などの属性情報を取得するために利用します。一例として、クラスの名前を取得する場合(クラスのテンプレート内で%className%を実行した場合と同じ結果を得る)には、次のようになります。

```
%EASL_GET("Property", classGUID, "Name")%
```

第 1 引数は、基本的には"Property"を指定します。その他、"Collection", "Count", "At" が指定できますが、これらの値を利用する必要があるのはまれです。このドキュメントでは説明を割愛します。

第 2 引数は、対象の要素の GUID を指定します。ソースコード生成対象のクラスの場合には%classGUID%マクロで取得できますが、多くの場合には EASL_GET マクロを利用して GUID を取得します。例えば、クラスが保持する(クラス要素の子要素として保持する)状態マシン要素の名前を取得する場合には、次のようになります。

```
$StateMachineGUID = %EASL_GET("Property", classGUID, "StateMachine")%
$Name = %EASL_GET("Property", $ StateMachineGUID, "Name")%
```

このようにして EASL_GET マクロで対象要素の GUID を取得して変数に格納し、その変数の値を渡すことで、目的の属性情報を取得します。

第 3 引数は、取得する属性情報の種類です。"Name"であれば要素や接続の名前を、"Notes"であれば要素や接続のノートの内容を取得できます。その他、上記の例のように、指定した属性の GUID を取得するためにも利用できます。

この第 3 引数で指定可能な文字列は、第 2 引数で渡す GUID が指す要素の種類に依存します。指定可能な内容は、ヘルプの「EASL プロパティ」のページをご覧ください。

Copyright (c) 2010-2025 Sparx Systems Japan

4.2 EASLList

先ほどの EASL_GET は、対象の要素や接続についての 1 つの情報を取得するために利用しました。これに対して、EASLList は、対象の要素や接続に対して、複数の要素や接続があるような場合に、それぞれの処理を行うために利用します。つまり、list マクロと同じ位置づけです。

具体的には、例えば以下のような状況で EASLList マクロを利用します。

- 状態マシンが保持する状態要素を取得する
- 状態要素に定義された Do などのアクションを取得する
- 状態要素から出て行く遷移を取得する
- アクティビティ要素から出て行くコントロールフローを取得する
- 要素が持つステレオタイプ(複数)を取得する

次の EASLList マクロは、状態マシン要素が保持する状態要素を取得する処理の定義の例です。

```
%EASLList="StateEnumerate" @separator=",¥n" @indent="" @owner=$StateMachineGUID @collection="AllStates" @option1="S_"%
```

最初の引数は、呼び出すテンプレートの名前です。取得した対象(ここでは、個々の状態要素)に対して、このテンプレートを繰り返し呼び出します。この部分は list マクロと同じ考え方です。

@separator および@indent は、list マクロと同じです。省略することも可能です。

@owner の値は、取得対象となる要素や接続の GUID(が含まれる変数)を指定します。必須です。この例では、事前に状態マシン要素の GUID を EASL_GET マクロで $StateMachineGUID 変数に格納してあります。

@collection は、対象の要素や接続に対して、取得するコレクションの種類を指定します。必須です。設定可能な文字列は、GUID で指定した要素や属性の種類によって変わります。設定可能な値の一覧は、ヘルプファイルの「EASL コレクション」のページをご覧ください。

最後の@option1 は、呼び出すテンプレートに渡す引数です。渡すことのできる引数は 1

Copyright (c) 2010-2025 Sparx Systems Japan

つではなく、@option2, @option3 などと、複数の引数を渡すことができます。呼び出し先のテンプレートで条件分岐する場合などに役立ちます。この詳細は、次の 5 章「テンプレートの呼び出し」をご覧ください。

4.3 マクロの活用の基本ルール

このドキュメントで説明する機能では、基本的にこの 2 つのマクロを組み合わせて利用します。組み合わせとして、「EASLList マクロで、処理対象(複数)を取得する」「EASL_GET マクロで、それぞれの処理対象の属性情報を取得し、ソースコードを構成する」という組み合わせの繰り返しになります。

一例として、ステートマシン図の場合には、以下のような流れになります。

1. EASL_GET で、クラス要素が持つ状態マシン要素を取得
2. EASLList で、状態マシン要素が持つ状態要素(複数)を取得
3. それぞれの状態要素に対して、EASL_GET で詳細情報を取得
4. EASLList で、それぞれの状態要素が関係する遷移(複数)を取得
5. EASL_GET で、それぞれの遷移の情報を取得する
6. ...

5 テンプレートの呼び出し

5.1 引数の渡し方と受け取り方

EASLList マクロでは、最初の引数で呼び出すテンプレートを指定します。また、@option1 などの引数で、文字列情報を渡すことができます。

4.2 章での例を再掲します。

```
%EASLList="StateEnumerate" @separator=",¥n" @indent="" @owner=$StateMachineGUID @collection="AllStates" @option1="S_"%
```

このような呼び出しがある場合には、呼び出し先のテンプレート(この例では StateEnumerate テンプレート)で、以下のようにして値を取得することができます。

$parameter1 = 対象の要素(この例では、個々の状態要素)の GUID
$parameter2 = @option1 として渡された文字列（この場合は"S_"の文字列)
$parameter3 = @option2 として渡された文字列（この場合は指定がないので、空文字列)
(以下同様)

このように、$parameter*で取得できる文字列と、@option*で渡す文字列には、数字の差が 1 つあることに気をつけてください。例えば、上記の呼び出し例では、@option1 に"S_"という文字列を渡していますが、この値は呼び出し先の StateEnumerate テンプレートで $parameter2 として、値を取得することができます。

このようにして、振る舞い図からのソースコード生成機能などでは、テンプレート間でさまざまな文字列をパラメータとして渡しています。そして、渡されたパラメータの内容を元に、if 文で条件分岐するなどの処理を行っています。

この引数を渡す機能は、テンプレートの種類が「<None>」の場合にも利用できます。ただし、この場合には、以下のように記述します。

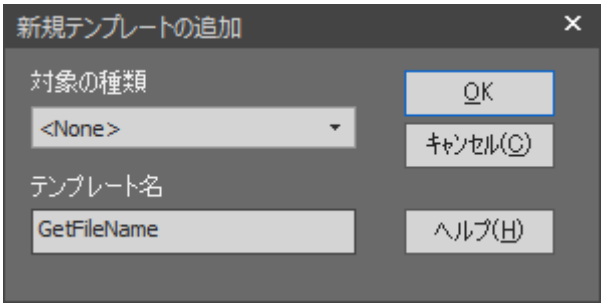
%テンプレート名(引数 1, 引数 2, ...)%

具体的な例は、次のサンプルをご覧下さい。

5.2 サンプル

テンプレートの種類が「<None>」のテンプレートの定義と、引数渡しを活用した例を紹介します。

まず、「新規テンプレートの追加」でテンプレートを追加します。



このテンプレートの中身を以下のように記述します。

Copyright (c) 2010-2025 Sparx Systems Japan

```
$path = $parameter1
$sl = %sl%
$pos = %FIND($path,$sl)%
%if $pos != "-1"%
$pos = %MATH_ADD($pos,"1")%
$path = %MID($path,$pos)%
%GetFileName($path)%
%else%
$path
%endIf%
```

この状況で、

```
$fileName = %GetFileName($relPath)%
```

のようにしてテンプレートを呼び出すと、”path¥to¥files¥filename.h”のような文字列から、”filename.h”のみを取り出すことができます。

なお、上記の内容の処理内容の詳細は以下の通りです。

1 行目:　渡された引数を、変数$path に格納します。
2 行目:　円記号(バックスラッシュ)はエスケープ文字としても使われているので、変数$sl に格納して 3 行目で利用します。
3 行目: 変数内で円記号を探し、結果を変数$pos に格納します。
4 行目: 円記号が存在する場合には、$pos の値は-1 以外になります。
5 行目: 文字列を切り取るため、円記号が存在する位置の次の位置を取得します。CTF では変数は文字列しかありませんので、足し算をする場合には機能マクロ MATH_ADD を利用する必要があります。
6 行目: 円記号の次の位置から末尾までを取得し、変数$path に格納しなおします。
7 行目: このテンプレートを再帰的に呼び出します。
8 行目以降: 円記号が存在しない場合には、引数として渡されてきた内容をそのまま返します。

Copyright (c) 2010-2025 Sparx Systems Japan

6 既定のテンプレート

最後に、Enterprise Architect が既定の状態で保持するテンプレートについて概要を説明します。

独自の出力結果を得るためのカスタマイズの際には、EASL マクロをどのように利用しているのか、既定のテンプレートを参考にすることが有用です。

6.1 シーケンス図・アクティビティ図からのコード生成のためのテンプレート

以下のテンプレートは、シーケンス図とアクティビティ図からのコード生成のために利用しています。

- Action
- Action で始まる多くのテンプレート
- Behavior
- Behavior Body
- Behavior Declaration
- Behavior Parameter
- CallArgument
- Guard

例えば、アクティビティ図のアクション要素については、まず Action テンプレートが呼ばれ、そのアクション要素の種類に応じて、それぞれのテンプレートを呼んでいます。

Behavior で始めるテンプレートは、コード生成の対象のクラス要素が子要素として保持する相互作用要素(シーケンス図)やアクティビティ要素(アクティビティ図)の処理のために利用しています。子要素の数だけこのテンプレートが呼ばれ、それぞれの図の内容を出力します。

6.2 ステートマシン図からのコード生成のためのテンプレート

以下のテンプレートは、ステートマシン図からのコード生成のために利用しています。

Copyright (c) 2010-2025 Sparx Systems Japan

- State
- State CallBack
- State Enumerate
- State EnumaratedName
- StateMachine
- StateMachine HistoryVar
- Transition
- Transition Effect
- Trigger

ステートマシン図からのコード生成の場合には、まず StateMachine テンプレートが呼ばれます。状態要素については State テンプレート・遷移については Transition テンプレート・トリガについては Trigger テンプレートがそれぞれ対応し、コードを生成しています。

なお、既定のテンプレートでは、例えば

```
$COMMAND_TYPE = "CommandType"
```

のように固定文字列を変数に代入し、その値を利用しています。これは、カスタマイズ可能な箇所をテンプレートの先頭付近に集結させるためであり、実際にカスタマイズする場合で、このようなソースコード内に出力される文字列が決まる場合には、こうした固定文字列(自動的に生成される変数やメソッドの名前など)は、テンプレート内に直接記述してしまう方がわかりやすくなります。

6.3 ビジネスルールからのソースコード生成のためのテンプレート

以下のテンプレートは、ビジネスルールからのソースコード生成で利用します。

- Rule Task
- Rulet
- Rule Condition
- Rule Action
- Compute Rulet
- Compute Action

ビジネスルールからのソースコード生成では、処理の条件と内容をテーブルという形で表

Copyright (c) 2010-2025 Sparx Systems Japan

現していますので、いわゆる for 文のような処理を行いテーブルのそれぞれの行の内容を解釈し、ソースコードとして生成しています。そのため、Rule Task テンプレートではカウンターとして利用するための MATH_ADD マクロを利用しています。
(このマクロの結果得られる数値は、テーブル内容を取得するための BR_GET マクロの引数として利用しています。)

6.4 属性としてのプロパティ

クラスのポートやパートは、出力時にはクラスの属性となります。そのために利用しているテンプレートが、Property・Property Declaration・Property Notes です。内容は、属性の出力のためのテンプレート(Attribute で始まるテンプレート)と基本的には同じです。属性として出力するべき情報を EASL_GET マクロで取得する必要がある点が異なります。

例えば、属性の型を取得するためには、Attribute テンプレートでは %attType%で取得できますが、このドキュメントで説明している拡張部分でのテンプレートでは
$TypeID = %EASL_GET("Property", $GUID, "Type")%
のようにしなければなりません。($GUID はポートやパートの要素の GUID)

Copyright (c) 2010-2025 Sparx Systems Japan

○改版履歴

2011/12/08 初版
2013/01/24 第 5 章を大幅加筆。いくつかの内容の補足を追加。
2014/04/22 Enterprise Architect11.0 のリリースに伴い、内容を更新。
2015/06/15 Enterprise Architect12.0 のリリースに伴い、内容を更新。
2018/05/16 Enterprise Architect14.0 のリリースに伴い、内容を更新。
2019/08/22 Enterprise Architect15.0 のリリースに伴い、内容を更新。
2025/01/30 参考情報に関する内容を更新。

Copyright (c) 2010-2025 Sparx Systems Japan