


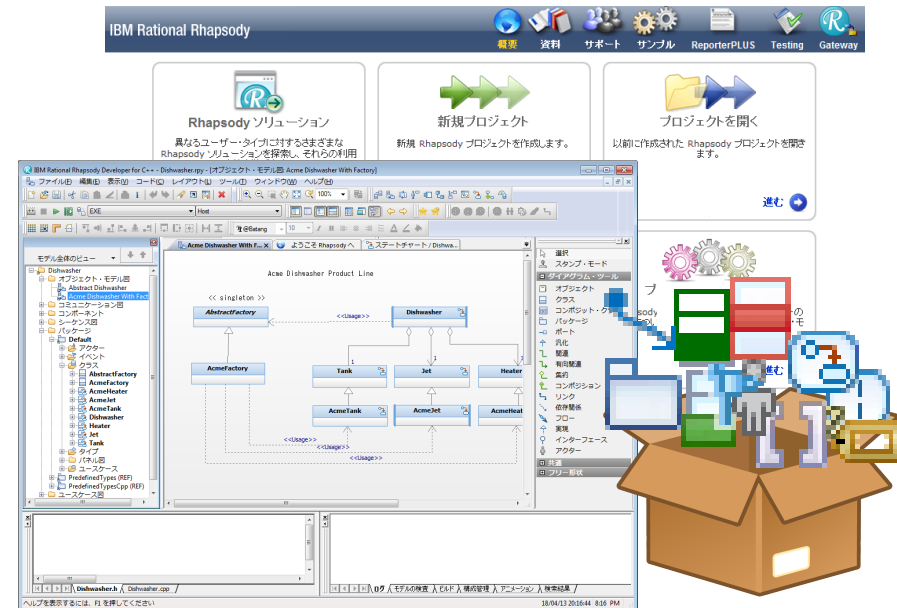
IBM Rational Rhapsody

コード生成に関するカスタマイゼーションの
活用方法 【実践編】



本日のテーマ

-  コード生成におけるカスタマイズ手法
-  プロパティ
-  カスタムCG (Demo)
-  ポストプロセス
-  RulesComposer



コード生成におけるカスタマイズ手法

Lightweight Customization

- コード生成パラメータ設定(プロパティ)
- ステレオタイプベース生成
- スタンダードオペレーション
- ポストプロセス

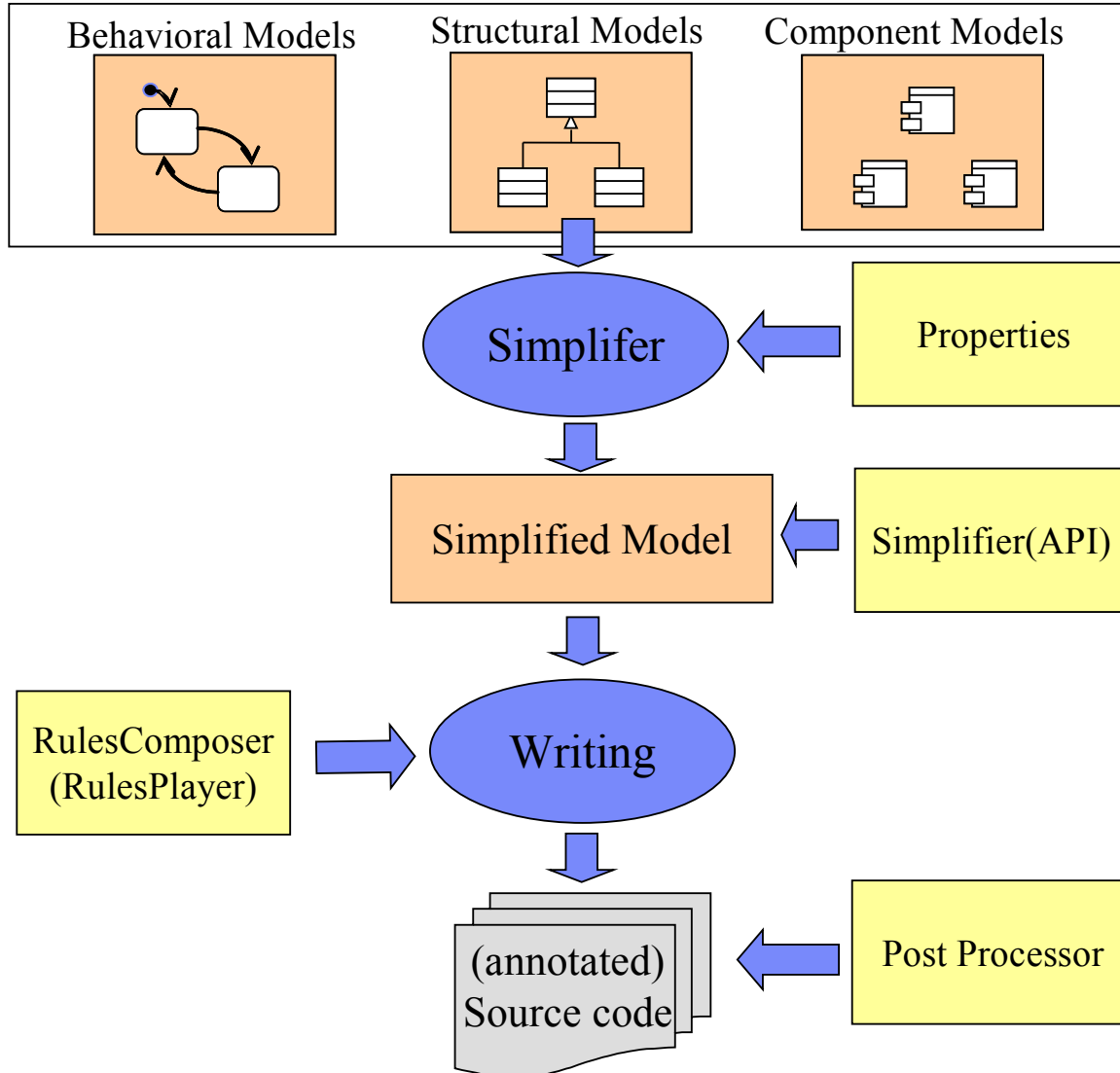
Heavyweight Customization

- カスタムCG (Plugin / Simplifar)
- RulesComposer



コード生成のアーキテクチャ

User Model



プロパティ

プロパティ構成

Subject::Metaclass::Property

クラス: MyClass (Default 内)

一般 説明 属性 操作 ポート フロー・ポート 関係 タグ **プロパティ**

すべて(L)を表示 ▾

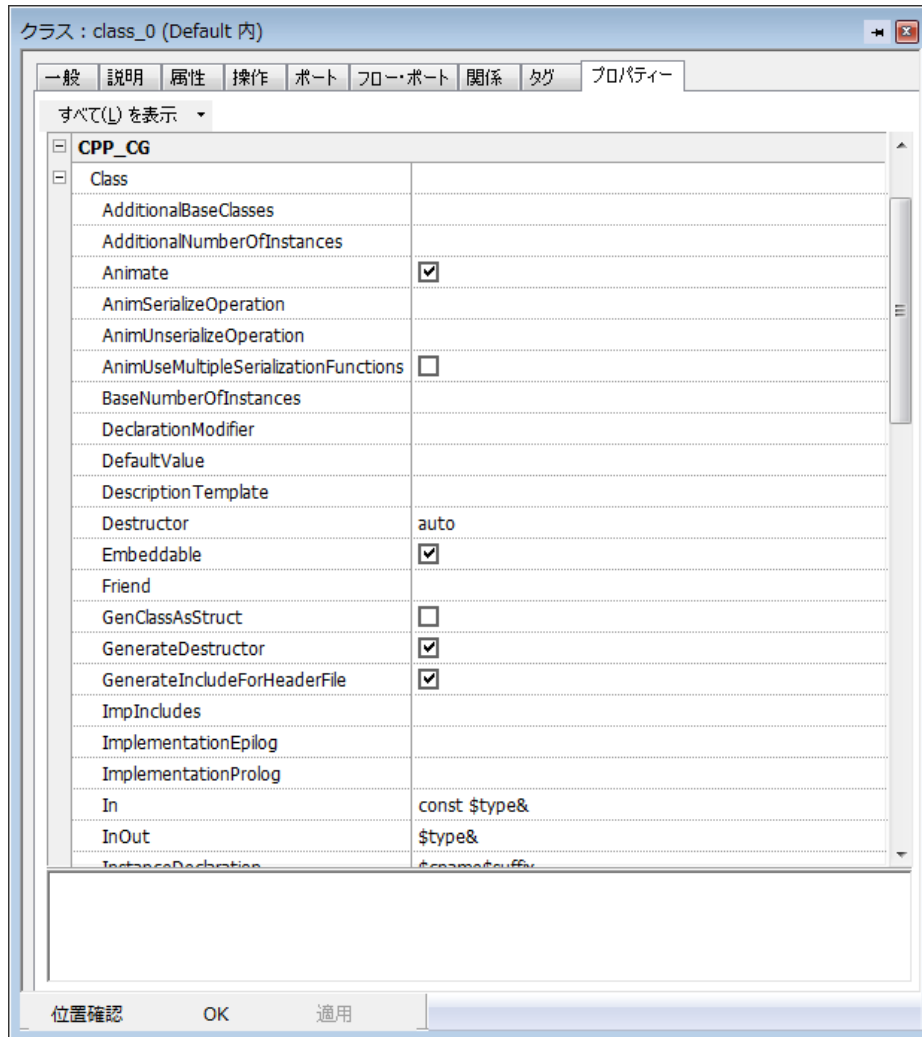
CG		サブジェクト
Class		メタクラス
ActiveMessageQueueSize		プロパティ
ActiveStackSize		
ActiveThreadName		
ActiveThreadPriority		
AdditionalCleanupCode		
AdditionalInitializationCode		
AttributesAutoArrange	<input checked="" type="checkbox"/>	
CallUserInitRelations	<input checked="" type="checkbox"/>	
ComplexityForInlining	3	
Concurrency	sequential	
CreateImplicitDependencies	<input checked="" type="checkbox"/>	
DeleteGlobalInstance	<input type="checkbox"/>	

CG::Class::ComplexityForInlining
 ComplexityForInlining プロパティは、ステートチャートに生成されるコード内の関数呼び出しの代わりに、Rhapsody がインライン化を使用する程度に関する一定程度の制御を提供します。

このプロパティは整数値を取ります。値が高くなれば、Rhapsody はそれだけ多くインライン化を使用します。値をゼロに設定すると、Rhapsody が生成されるコード内の関数呼び出しの代わりにインライン化を使用することはありません。

位置確認 OK 適用

コード生成(CG)プロパティ



CG サブジェクト

すべての言語に共通のコード生成を制御するプロパティが含まれたメタクラスを持ちます。

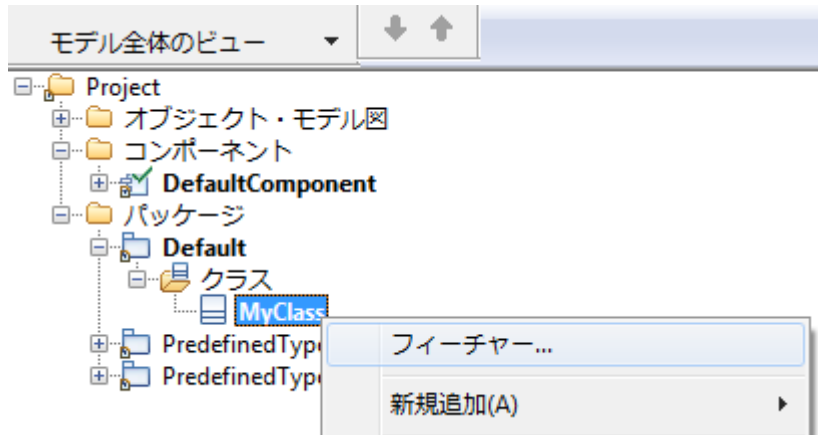
CPP_CG サブジェクト

コード生成用の一般メタクラスに加えて、OS環境を指定するプロパティが含まれたメタクラスを持ちます。

※C言語は、C_CG、Javaは、JAVA_CG

例①

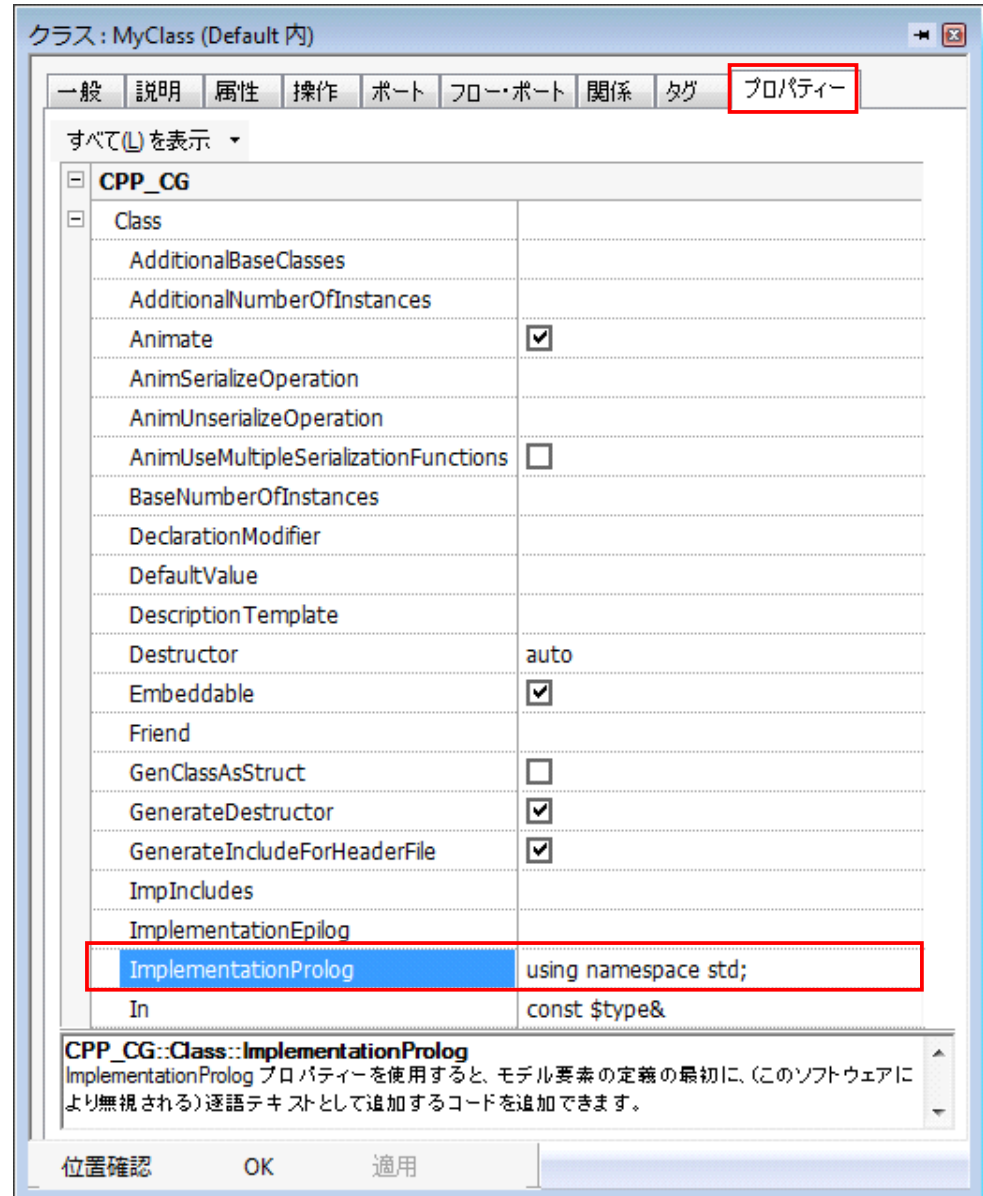
CPP_CG::Class::ImplementationProlog プロパティを使って、標準ライブラリの名前空間を設定します。



プロパティ値:

```
using namespace std;
```

宣言ファイルにテキストを追加する場合は、**CPP_CG::Class::SpecificationProlog** プロパティを使用します。



コード生成結果①

MyClass.cpp (変更前)

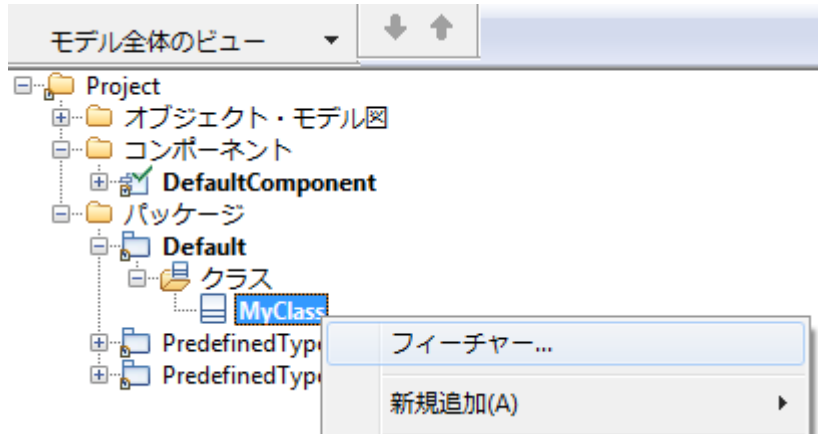
```
///  
//## auto_generated  
#include "MyClass.h"  
///  
//## package Default  
  
//## class MyClass  
MyClass::MyClass () {  
}  
  
MyClass::~MyClass () {  
}
```

MyClass.cpp (変更後)

```
///  
//## auto_generated  
#include "MyClass.h"  
///  
//## package Default  
  
//## class MyClass  
using namespace std;  
  
MyClass::MyClass () {  
}  
  
MyClass::~MyClass () {  
}
```


例②

CPP_CG::Class::ImpIncludes プロパティを使って、
任意のヘッダーファイルを追加します。

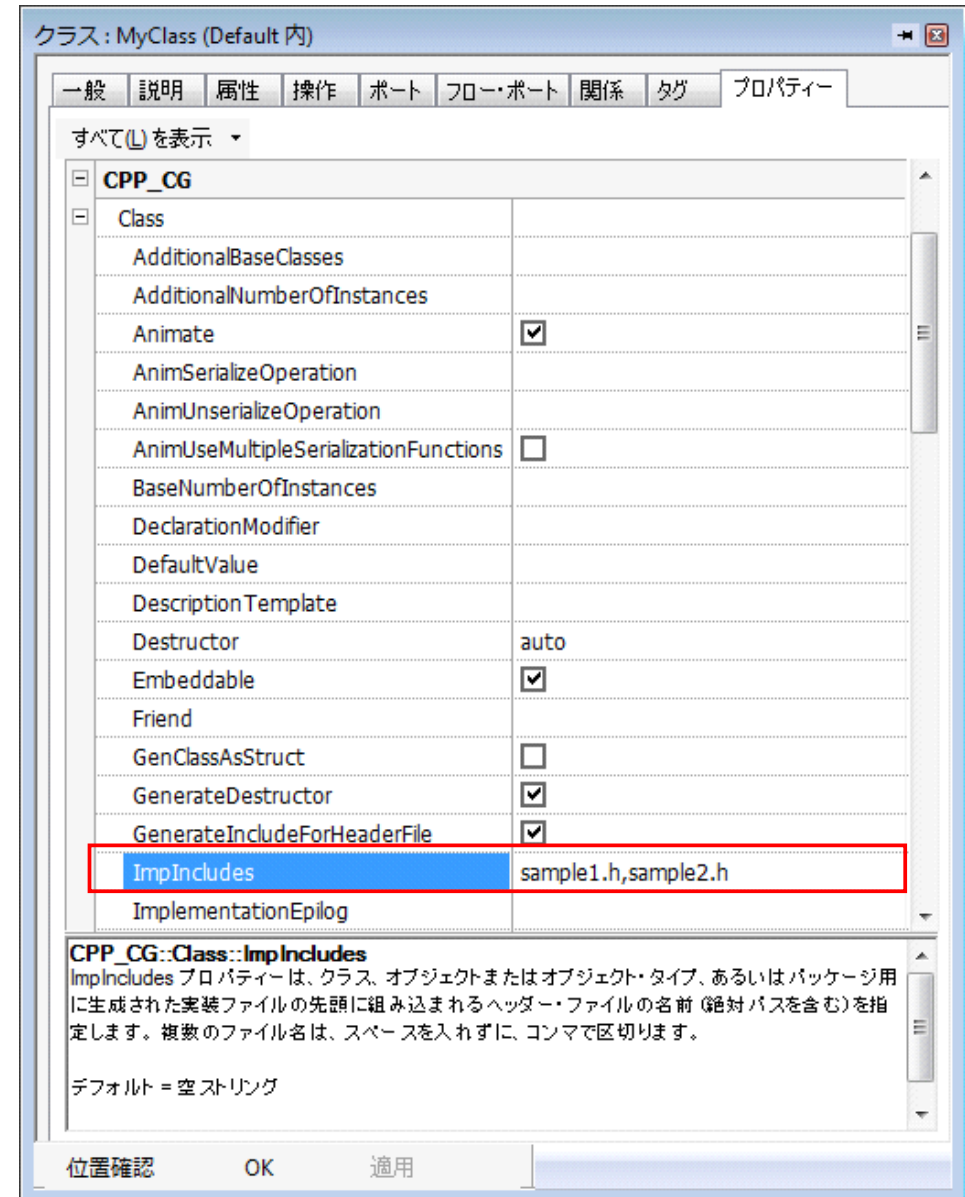


プロパティ値:

sample1.h,sample2.h

※複数指定は、カンマ区切り

宣言ファイルに追加する場合は、
CPP_CG::Class::SpecIncludes
プロパティを使用します。



コード生成結果②

MyClass.cpp (変更前)

```
///  
//## auto_generated  
#include "MyClass.h"  
///  
//## package Default  
  
///  
//## class MyClass  
MyClass::MyClass() {  
}  
  
MyClass::~MyClass() {  
}
```

MyClass.cpp (変更後)

```
///  
//## auto_generated  
#include "MyClass.h"  
//## auto_generated  
#include "sample1.h"  
//## auto_generated  
#include "sample2.h"  
///  
//## package Default  
  
///  
//## class MyClass  
MyClass::MyClass() {  
}  
  
MyClass::~MyClass() {  
}
```

例③

コメント関連プロパティ

```

/*****
Rhapsody      : 8.0.2
Component     : DefaultComponent : DefaultConfig
Model Element : MyClass
Copyright (C) 2013 IBM Australia Ltd. All Rights Reserved.
*****/

#ifndef MyClass_H
#define MyClass_H

///## auto_generated
#include <oxf*oxf.h>
///## package Default

///## class MyClass
// Default::MyClass - This is a sample class
class MyClass {
    /// Constructors and destructors ///

public:

    ///## auto_generated
    MyClass():

    ///## auto_generated
    MyClass():

    /// Operations ///
    // Default::MyClass::MyOperation
    // arg1 - description from arg1
    // arg2 - description from arg2
    ///## operation MyOperation(int, int)
    void MyOperation(int arg1, int arg2);

    /// Additional operations ///

    ///## auto_generated
    int getMyAttribute() const;

    ///## auto_generated
    void setMyAttribute(int p_MyAttribute);

    /// Attributes ///

protected:
    int MyAttribute;    ///## attribute MyAttribute
};
#endif
/*****/
    
```

コード関連プロパティ

```

/*****
Rhapsody      : 8.0
Login         : yukihir
Component     : DefaultComponent
Configuration : DefaultConfig
Model Element : MyClass
Generated Date: 火, 23, 4 2013
File Path     : DefaultComponent\DefaultConfig\MyClass.h
*****/

#ifndef MyClass_H
#define MyClass_H

///## auto_generated
#include <oxf*oxf.h>
///## package Default

///## class MyClass
class MyClass {
public:

    ///## Friend List ///
    friend int t0;
    friend class x;

    /// Constructors and destructors ///

    ///## auto_generated
    MyClass():

    ///## auto_generated
    MyClass():

    /// Operations ///
    ///## operation MyOperation(int, int)
    inline void MyOperation(int arg1,
                             int arg2);

    /// Additional operations ///

    ///## auto_generated
    void setMyAttribute(int p_MyAttribute);

private:
    ///## auto_generated
    int getMyAttribute() const;
    ///## Attributes ///

protected:
    int MyAttribute;    ///## attribute MyAttribute
};

inline void MyClass::MyOperation(int arg1,
                                  int arg2)
{
    ///## operation MyOperation(int, int)
    ///##
}

#endif
/*****/
    
```

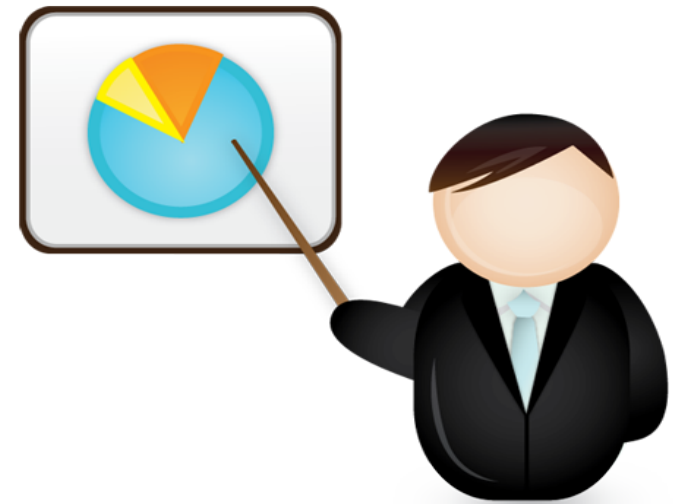
まとめ

特徴

- プロパティ設定は容易に、コードのカスタマイズができます
- モデル全体、または要素単位でのカスタマイズを行うことができます

注意点

- プロパティが豊富に準備されていますが、カスタマイズの範囲には限界があります



カスタムCG

カスタムCGは、プラグインとして Rhapsody に登録する Java アプリケーションです。
カスタムCGを利用して、お客様自身、自由自在にコード生成のカスタマイズができます。

カスタマイズポイント

- ユーザモデルからSimplifiedModelを生成する単純化処理
- SimplifiedModel からソースコードを生成するWriting 処理

モデルの単純化の為の準備

- プラグイン登録
- デバッグ設定 (Eclipse)

API 情報

単純化のメカニズムと構造



プラグイン登録

What

プラグインにて、Rhapsodyにメニュー、ポップアップメニューを追加することができます。
更に(自定義メニューのクリック、コード生成、モデルチェックなどの)イベントハンドラを記述することも可能です。

How

1) .iniファイルに設定する方法

範囲: Rhapsody全体に適用

```
[Plugin]
Plugins=MyPlugin
[MyPlugin]
Name=My Plugin
RhpVersion=8.0.2
JavaMainClass=com.myplugin.PluginMain
JavaClassPath=C:\.....\MyPlugin1.jar;$OMROOT/...../MyPlugin2.jar;
isPlugin=1
IsVisible=1
MenuItem=MyPluginMenu
```

プラグイン基本

2) .hepファイルに設定する方法

範囲: 該当プロジェクトに適用

方法1) 属性 **General:Model:HelpersFile** に.hepファイルを設定します。

方法2) プロファイルと関連付ける(同じディレクトリ、同名) .hep ファイルは自動的にロードされます。他の.hepファイルは属性 **General:Profile:AdditionalHelpersFiles** に設定します。

```
[Helpers]
numberOfElements=2

#REM: Definition of the plug-in
name1=Diagram Formatter
JavaMainClass1=JavaPlugin.PluginMainClass
JavaClassPath1=$OMROOT\..\Samples\JavaAPI Samples\Plug-in
isPlugin1=1
isVisible1=1

#REM: Definition of the pup-up menu that trigger the plugin
name2=Format Diagram
isPluginCommand2=1
command2=Diagram Formatter
applicableTo2=ObjectModelDiagram
isVisible2=1
```

プラグインのメソッド一覧

抽象クラス `com.telelogic.rhapsody.core.RPUserPlugin` のメソッド一覧

```
//プラグインがロードされる時
public abstract void RhpPluginInit(final IRPApplication rhpApp);

//プラグインのツールメニューが選択される時
public void RhpPluginInvokeItem();

//プラグインのポップアップメニューが選択される時
public void OnMenuItemSelect(String menuItem);

//プラグインのトリガーが呼び出される時
public void OnTrigger(String trigger);

//プロジェクトがクローズされる時(trueを返すと、プラグインがアンロードされる)
public boolean RhpPluginCleanup();

//Rhapsodyが終了される時
public void RhpPluginFinalCleanup();
```


API 関連の情報

"So what's the deal with the Java API?" (日本語版)

<http://www-01.ibm.com/support/docview.wss?uid=swg21512252>

メタクラス情報:

<http://www-01.ibm.com/support/docview.wss?uid=swg21573599>

Rhapsody API Javadoc:

保存場所:<インストールフォルダ>%Doc%java_api%index.html

技術文献:



Press Enter to search.

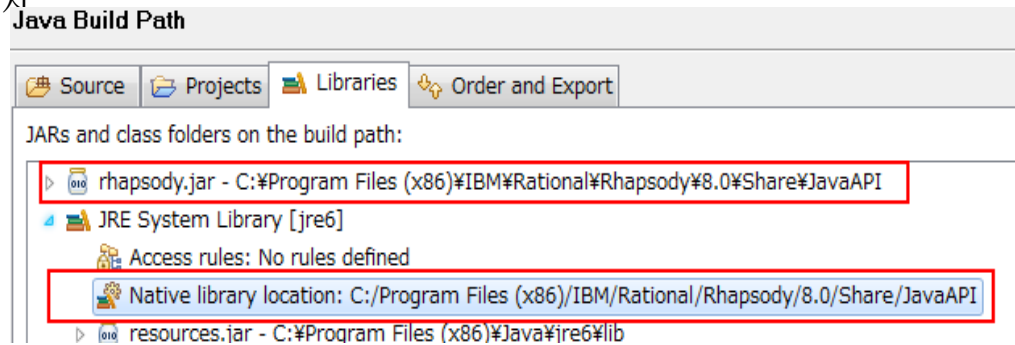
Eclipseでのデバッグ設定

1. Rhapsody.ini の設定

```
[JVM]
JavaLocation=C:%Program Files (x86)%IBM\Rational\Rhapsody%8.0%jre
Options=ClassPath,LibPath,Debug1,Debug2,Debug3
Debug1=-Xnoagent
Debug2=-Xdebug
Debug3=-Xrunjdwpt:transport=dt_socket,address=6743,server=y,suspend=y
ClassPath=!!java.class.path!!C:%Program Files (x86)%IBM\Rational\Rhapsody
```

2. Eclipse の設定

•ライブラリ設定

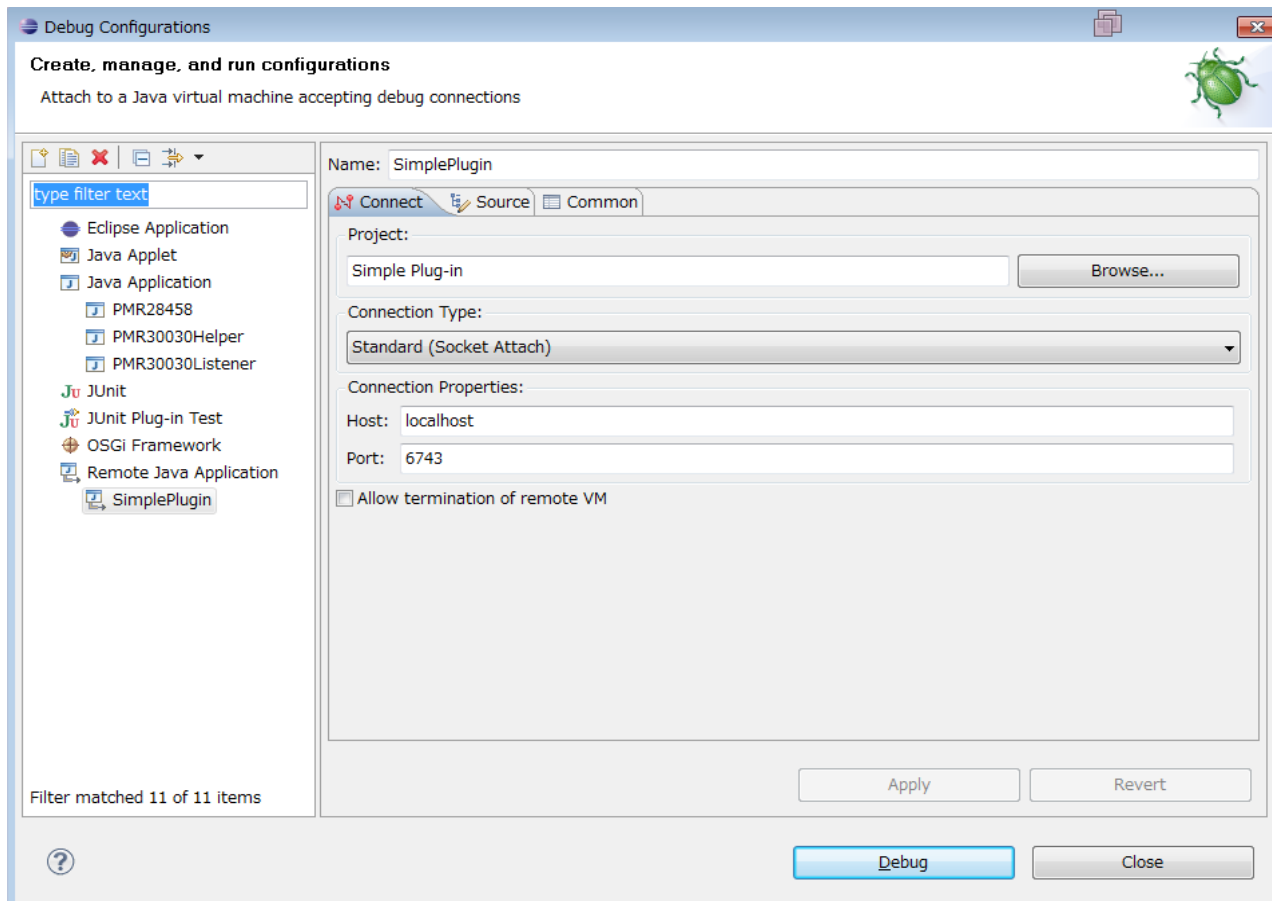


•プラグインに

```
public static void main(String[] args) {
    //create an instance of my plugin
    MyPlugin myPlugin = new MyPlugin ();
    //get Rhapsody application that is currently running
    IRPApplication app = RhapsodyAppServer.getActiveRhapsodyApplication();
    //init the plugin
    myPlugin.RhpPluginInit(app);
    //imitate a call to the plugin
    myPlugin.RhpPluginInvokeItem();
}
```

•リモートデバッグ設定

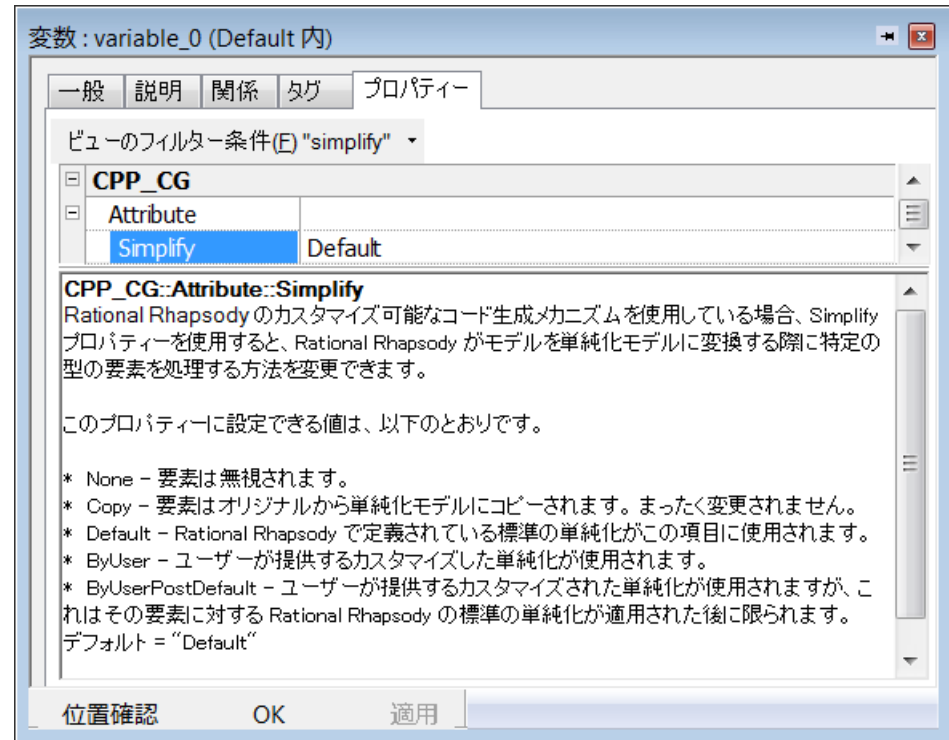
"Run-> Debug Configurations" メニューをクリックして、
Remote Java Application への接続を設定します。



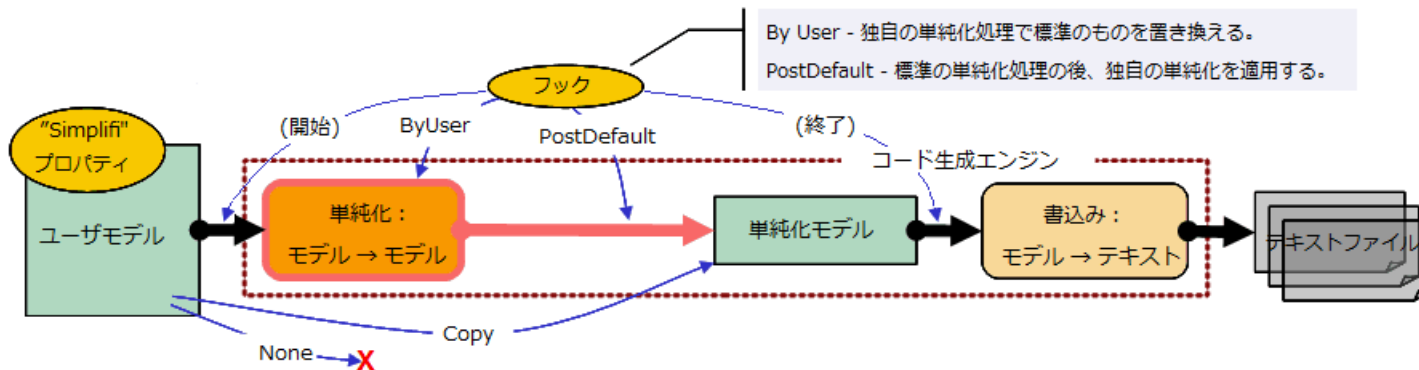
モデルの単純化処理の流れ

Simplify プロパティとは...

モデル要素のタイプごとに指定することができ、ユーザーモデルから単純化モデルへ変換する際に、その処理方法を決定するためのプロパティとなります。単純化を制御するために使われます。

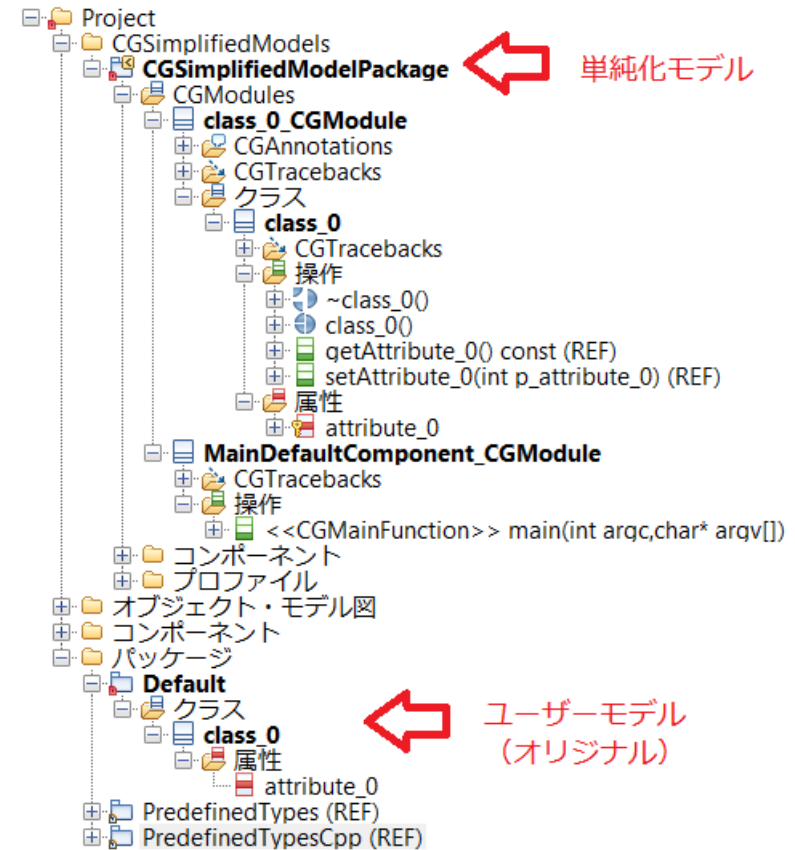
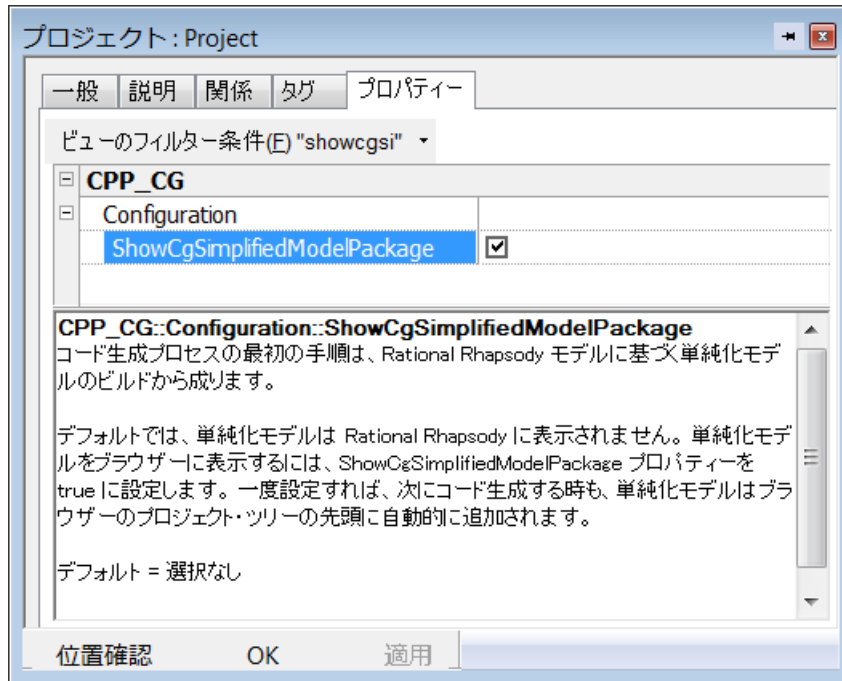


モデルの単純化処理のメカニズム



単純化モデルの構造

ShowCgSimplifiedModelPackageとは...





www.ibm.com/software/rational

CppUserSimplifiersSample の構造解説

AbstractSimplifier を経由して、それぞれの Simplifier が RPCodeGenSimplifier を実装する構造で整理されている。



OperationSimplifier.java

IBM Rational Rhapsody Developer for C++ - UserSimplifiers.rpy - [ようこそ Rhapsody へ]

クラス: Triangle (Shapes 内)

一般	説明	属性	操作	ポート	フローポート
関係		タグ			プロパティ
ローカルにオーバーライド済み(Q)を表示					
CPP.CG			Simplify		ByUser
Attribute			Operation		
			Simplify		ByUserPostDefault

```

double Triangle::calculateArea() {
    //#[ operation calculateArea()
    Reporter::getInstance()->report("Entering operation 'Triangle.calculateArea'.");

    double vecTr = mBase1.x * mBase2.y +
        mBase1.y * mTop.x +
        mTop.y * mBase2.x -
        mBase2.y * mTop.x -
        mBase1.y * mBase2.x -
        mBase1.x * mTop.y;

    double multiplier = (vecTr<0) ? -0.5 : 0.5;

    return vecTr * multiplier;
    //#[
}

double Triangle::calculatePerimeter() {
    //#[ operation calculatePerimeter()
    Reporter::getInstance()->report("Entering operation 'Triangle.calculatePerimeter'.");

    return SUPER::distance(mBase1,mBase2) + SUPER::distance(mBase2,mTop) + SUPER::dista
    //#[
}
    
```

Triangle.h \ Triangle.cpp

ヘルプを表示するには、F1を押してください 2013/04/16 22:24:12 10:24 PM

IBM Rational Rhapsody Developer for C++ - UserSimplifiers.rpy - [ようこそ Rhapsody へ]

クラス: Triangle (Shapes 内)

一般	説明	属性	操作	ポート	フローポート
関係		タグ			プロパティ
ローカルにオーバーライド済み(Q)を表示					
CPP.CG			Simplify		Default
Attribute			Operation		
			Simplify		Default

```

double Triangle::calculateArea() {
    //#[ operation calculateArea()
    double vecTr = mBase1.x * mBase2.y +
        mBase1.y * mTop.x +
        mTop.y * mBase2.x -
        mBase2.y * mTop.x -
        mBase1.y * mBase2.x -
        mBase1.x * mTop.y;

    double multiplier = (vecTr<0) ? -0.5 : 0.5;

    return vecTr * multiplier;
    //#[
}

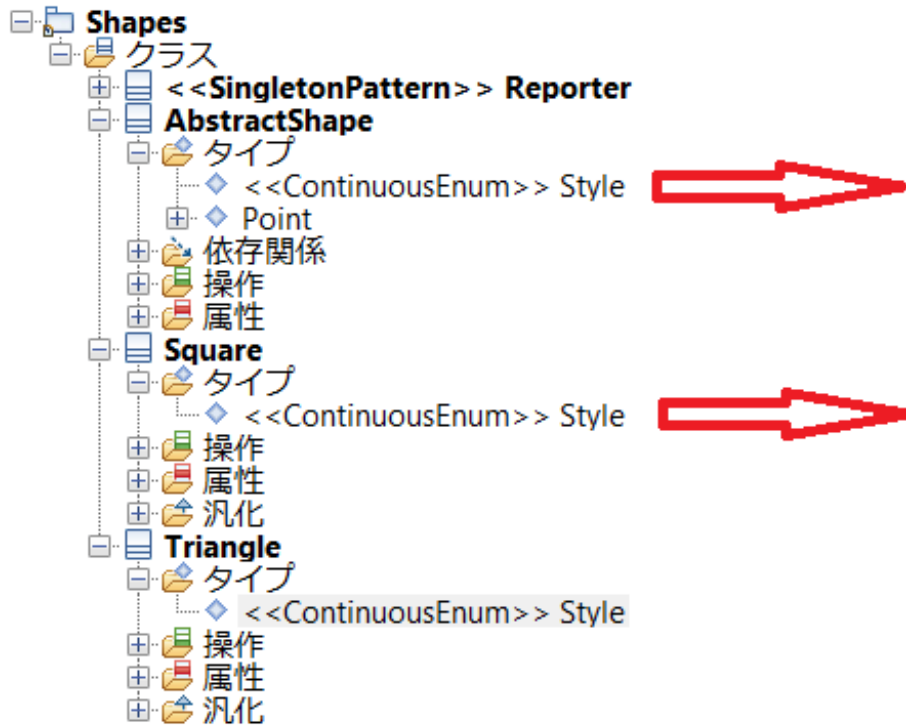
double Triangle::calculatePerimeter() {
    //#[ operation calculatePerimeter()
    return SUPER::distance(mBase1,mBase2) + SUPER::distance(mBase2,mTop) + SUPER::dista
    //#[
}

/*****
    
```

Triangle.h \ Triangle.cpp

ヘルプを表示するには、F1を押してください 2013/04/16 22:24:12 10:24 PM

TypeSimplifier.java

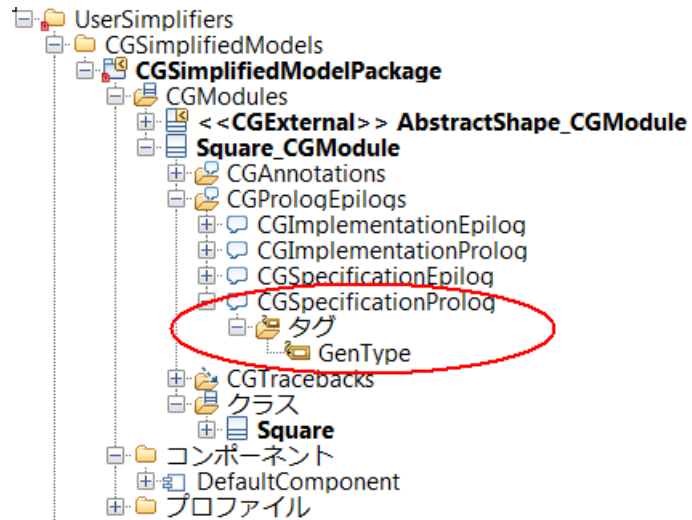


```
enum Style {
    NONE = 0,
    LAST_VALUE = 1
};
```

```
class Square : public AbstractShape {
public :

    ///## type Style
    enum Style {
        CONVEX = SUPER::LAST_VALUE + 0,
        CONCAVE = SUPER::LAST_VALUE + 1,
        LAST_VALUE = SUPER::LAST_VALUE + 2
    };
};
```

GeneralizationSimplifier.java



```

//## class Square

//## ignore
#define SUPER AbstractShape

class Square : public AbstractShape {
public :

//## type Style
enum Style {

```

コード修正:

```

IRPComment specProlog = (IRPComment)simpleOwner.addNewAggr("CGPrologEpilog", "CGSpecificationProlog");
IRPTag my_tag = specProlog.getTag("GenType");
if (my_tag != null)
{
    specProlog.setTagValue(my_tag, "Specification");
    specProlog.addProperty("CG.ModelElement.Kind", "String", "Prolog");
}
specProlog.setSpecification("//## ignore" + "\n" + "#define SUPER " + simpleBaseClassifier.getName());

```



www.ibm.com/software/rational

注意事項

1. 極力、Simplify=ByUserPostDefault (Post Simplify) でカスタマイズを行う。Simplify 段階でカスタマイズする場合には、アノテーションの挿入はユーザーが実施しなければならない。
2. サンプルのように複数の Simplifier を登録する場合、どの順番で Simplifier が行使されるかは決められていない。
3. 複数要素が同一階層にある場合、単純化の順番は不明である。ただし、親の要素はかならず子の要素の前に処理される。
4. Simplify 工程の段階にてオリジナルモデルを更新することは可能だが、その段階ではオリジナルモデルがまだ参照されている状態なので、更新することは推奨されない。なお、単純化処理後でのオリジナルモデルの更新は可能である。

📁 ポストプロセス

- GUI にて簡単に実現できない、軽量型のコードカスタマイゼーション方法の一つです。
- 処理対象は、モデルから生成された任意のソースファイルになります。
- 使い慣れた任意の言語で、編集処理を実装する実行ファイルを作成できます。



実行ファイルの作成

慣れ親しんだ任意のアプリケーション言語、もしくは、スクリプト言語で、実行ファイルを作成します。

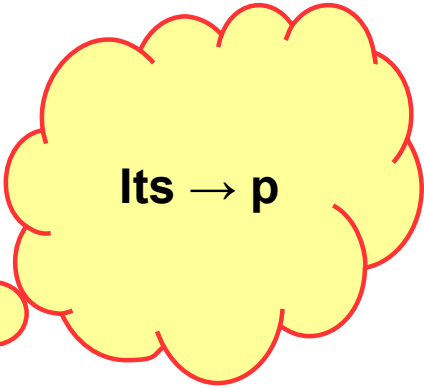
下記のサンプルコードは、ターゲットのソースファイル内で、デフォルトで生成された関連ロール 'its'を、例えば、社内のコーディング規約に準じた、'p'に置き換える C++ のコードになります。

```
static void process (const char* filename)
{
    string lineString, fileString;
    int found;

    ifstream in;
    ofstream out;
    in.open(filename);

    if (!in)
    {
        cout << filename << " does not exists!" << endl;
    }
    else
    {
        while (getline(in,lineString))
        {
            found=lineString.find(" its");
            if(found!=string::npos)
            {
                lineString.replace(found, 3, "p");
            }
            fileString += lineString + '\n';
        }
        in.close();
        out.open(filename);
        out.seekp(ios::beg);
        out << fileString;
        out.close();
    }
}
```

```
found=lineString.find(" its");
if(found!=string::npos)
{
    lineString.replace(found, 3, "p");
}
fileString += lineString + '\n';
```

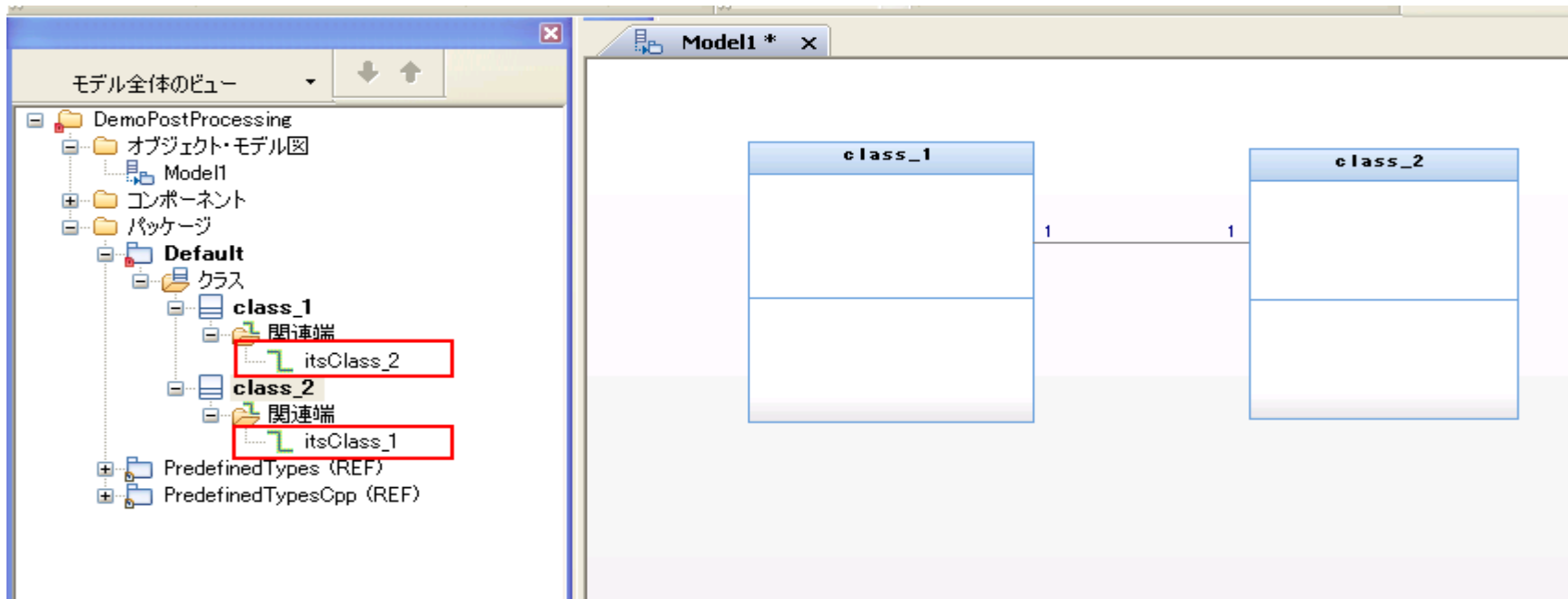


設定方法

CG::File::InvokePostProcessor プロパティに、作成した実行ファイルを実行させる為のパスを設定する必要があります。

CG	
File	
InvokePostProcessor	\$projectPath#target.exe \$file

コード生成を行うことで、下記のモデルに対して、“its” を“p” に置き換える処理を行うポストプロセスが適用されます。



コード生成結果

適用前

```

class_1.h x class_1.cpp

//## class class_1
class class_1 {
    //// Constructors and destructors    ////

public :

    ## auto_generated
    class_1();

    ## auto_generated
    ~class_1();

    //// Additional operations    ////

    ## auto_generated
    class_2* getItsClass_2() const;

    ## auto_generated
    void setItsClass_2(class_2* p_class_2);

protected :

    ## auto_generated
    void cleanUpRelations();

    //// Relations and components    ////

    class_2* itsClass_2;    ## link itsClass_2

    //// Framework operations    ////

public :

    ## auto_generated
    void __setItsClass_2(class_2* p_class_2);
    .....
    
```

適用後

```

class_1.h x class_1.cpp Model1 *

//## class class_1
class class_1 {
    //// Constructors and destructors    ////

public :

    ## auto_generated
    class_1();

    ## auto_generated
    ~class_1();

    //// Additional operations    ////

    ## auto_generated
    class_2* getpClass_2() const;

    ## auto_generated
    void setpClass_2(class_2* p_class_2);

protected :

    ## auto_generated
    void cleanUpRelations();

    //// Relations and components    ////

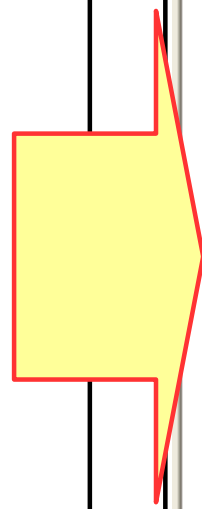
    class_2* pClass_2;    ## link m_itsClass_2

    //// Framework operations    ////

public :

    ## auto_generated
    void __setpClass_2(class_2* p_class_2);

    ## auto_generated
    
```



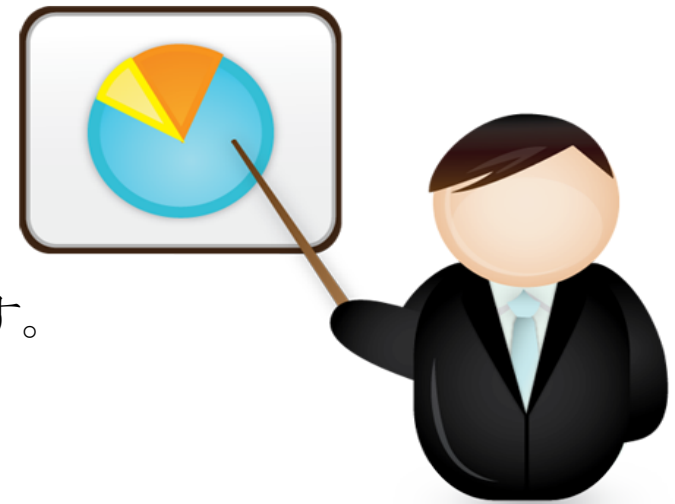
まとめ

特徴

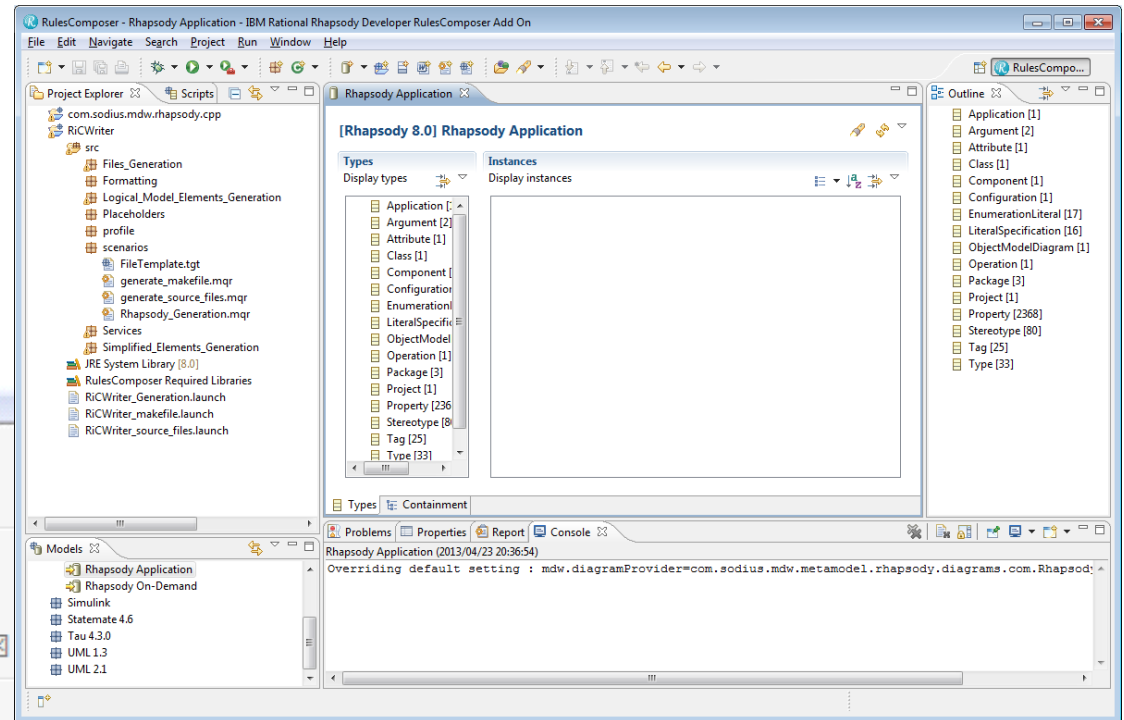
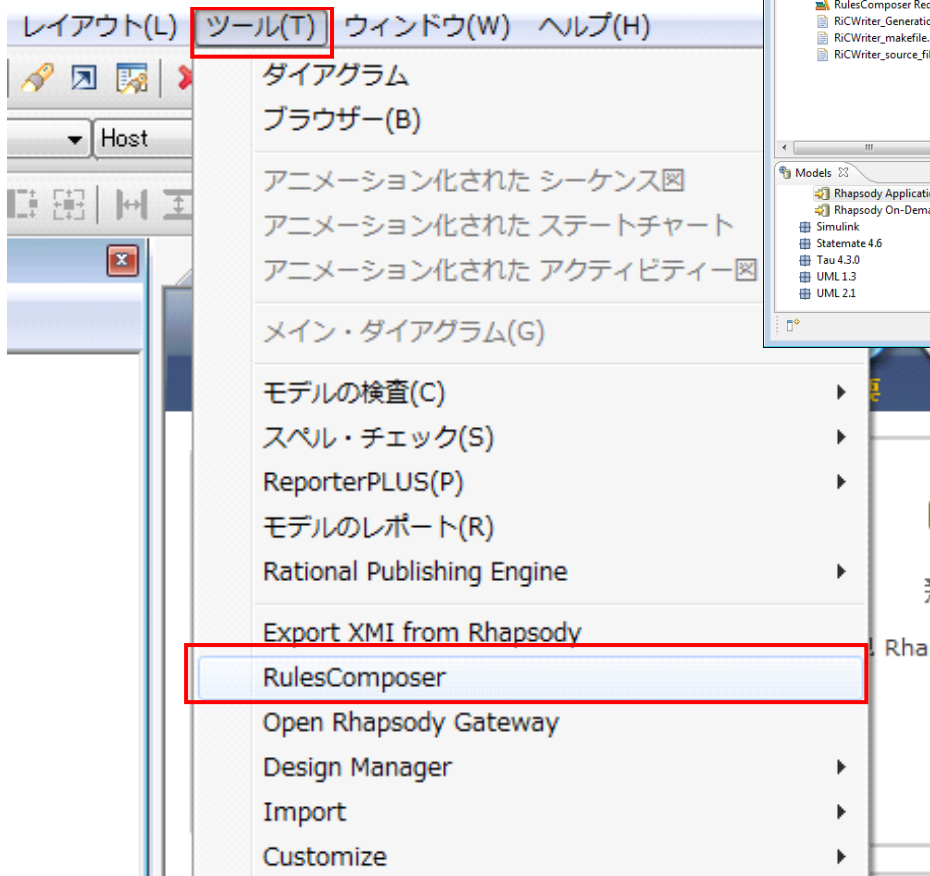
- ユーザー特有のコーディングルールを適用することができ、軽量型のコードカスタマイゼーションを 容易に実現できます。
- ソースファイルのみに対し変更を加え、モデル要素には影響を与えてほしくない場合に利用できます。
- 異なる目的を実現するポストプロセッシングを、個別要素上に適用可能で、ある程度の柔軟性が 提供されます。

注意事項

- 変更内容に次第で、変更後のソースファイルの情報は、モデル内容と一致しない可能性があります。
- モデル要素と関係のない変更は、特別な問題を引き起こすことはありませんが、加えた変更内容及び、変更場所により、ポストプロセッシング処理後の RoundTrip の際に解析エラーが発生する可能性があります。



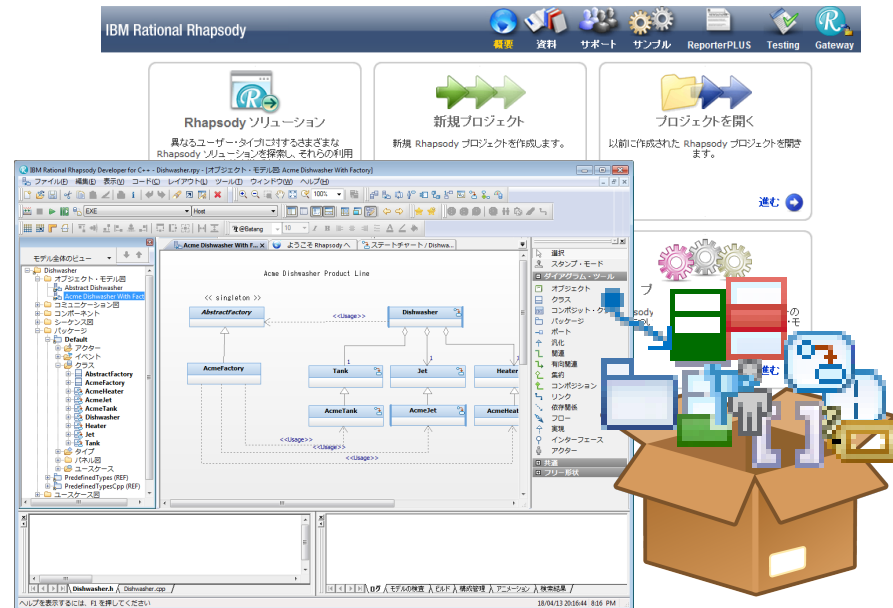
RulesComposer

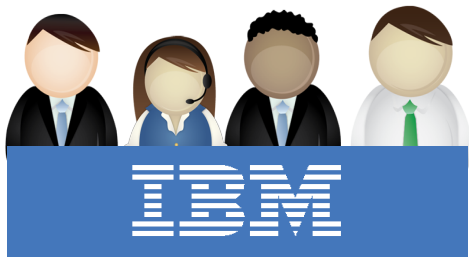


QUESTIONS

本日のテーマ

- コード生成におけるカスタマイズ手法
- プロパティ
- カスタムCG
- ポストプロセス
- RulesComposer





ご意見、ご質問等がありましたら、いつでも
サポートセンターまでご連絡ください！！

© Copyright IBM Corporation 2013. All rights reserved. The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. IBM, the IBM logo, Rational, the Rational logo, Telelogic, the Telelogic logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.

免責事項

当資料は、お客様の問題解決のためのヒントとしてご利用ください。当資料における記載内容は、お客様固有の問題に対し適切であるかどうか、また正確であるかどうかは十分検証されていません。結果についていかなる保証も責任も負いかねますので、あらかじめご了承ください。