系统建模仿真环境
平台层 API 详细说明

科学计算与系统建模仿真项目组

编制	周雯、王越洲	**生效日期**	2023-03-17
审核	王天飞	**批准**	郭俊峰

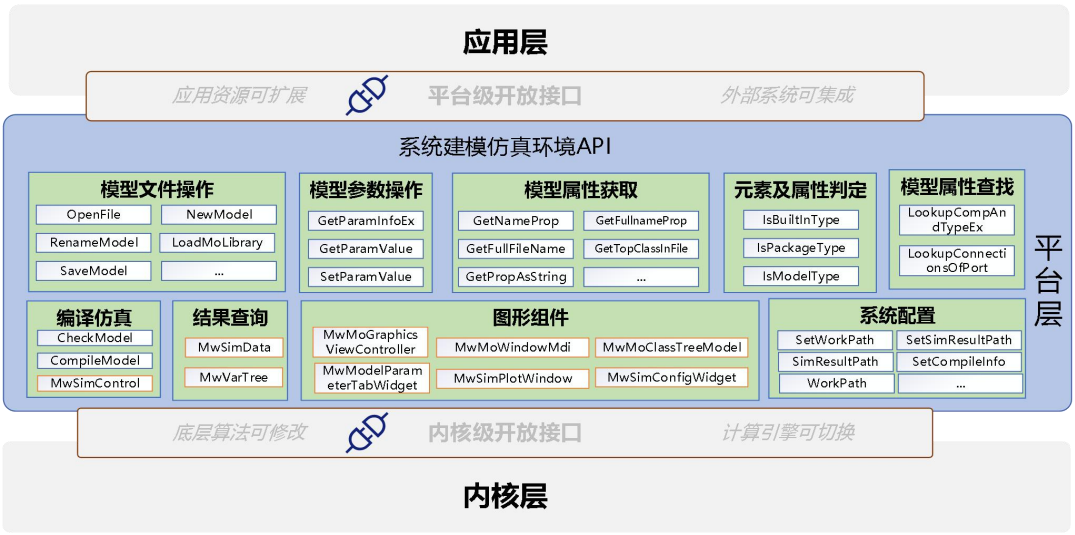
文件变更摘要

日期	版本	变更说明	修订	审核	批准
2023/3/17	V1.0	初始建立	王越洲		
2023/4/02	V1.1	添加 API 整体框架，修改文档目录	王天飞		
2023/5/18	V1.2	增加 API 示例，修改全部 API 结构，增加名词解释章节	周雯		
2023/7/18	V1.3	根据田老师批注进行修改：增加类图，文本细节修改	周雯		

目 录

1. 概述

系统建模仿真环境平台层 API 是 MWORKS.Sysplorer 供开发者和外部系统调用的标准接口。按照工作流分为模型文件操作、模型参数操作、模型属性获取、元素及属性判定、模型属性查找、编译仿真、结果查询、图形组件类和系统配置共 9 类 API。

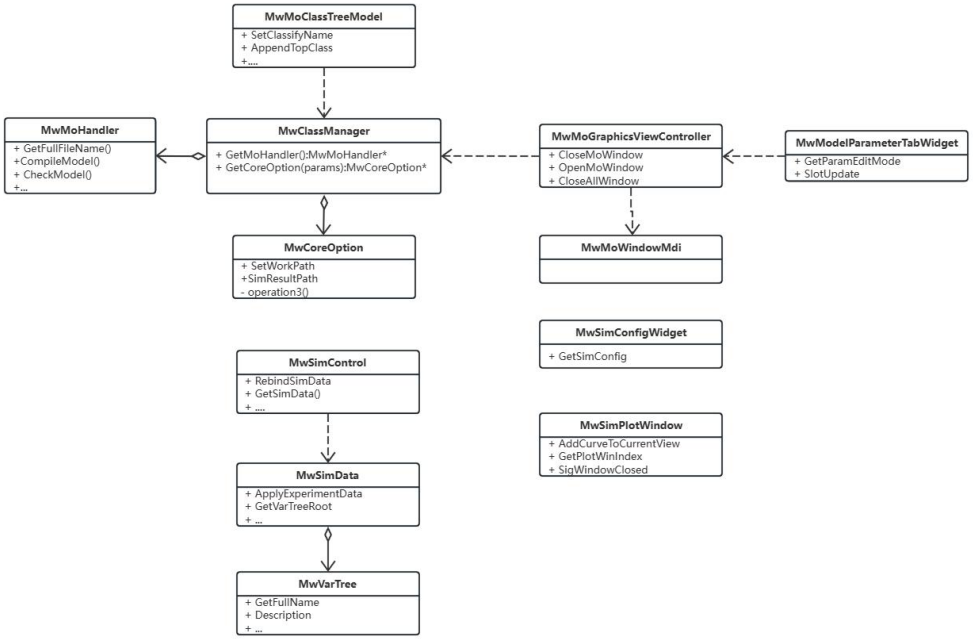


1-1 平台层-系统建模仿真环境接口模块

其中除图形组件模块外，其他八大模块均可在无界面情况下对模型进行操作、编译、仿真、数据后处理。

图形组件模块，为用户提供了模型可视化界面，包括显示模型视图、建模操作、仿真设置界面、后处理数据显示界面、曲线显示界面，用户可结合其他几大模块搭建一个完整的带界面的模型处理软件。

SDK 类结构图如下图所示，其中模型相关操作均在 MwMoHandler 类中、系统环境配置相关接口在 MwCoreOption，这两个类均在 MwClassManager 中获取，其余为面板界面呈现、仿真控制类、仿真面板设置类关系如下。



2. 系统建模仿真环境平台层 API

2.1 模型文件操作

模型文件操作，主要为对模型文件进行新建、打开、加载、重命名、保存、卸载、复制模型等操作。

函数名	简介
OpenFile	打开模型文件(mo,bmf,mef)
NewModel	新建模型
RenameModel	为模型重命名
LoadMoLibrary	加载模型库(mo)
SaveModel	将修改内容保存到模型文件中
UnloadModel	卸载已加载或打开的模型
DuplicateModel	复制模型

2.1.1 OpenFile

打开模型文件

A. 语法

```
/**
 * @brief 打开模型文件(mo,bmf,mef)
 * @param [in] strFile 模型文件路径
 * @return 模型是否打开成功
 */
bool MwMoHandler::OpenFile(const std::wstring& str_file);
```

B. 说明

用于打开模型，包括 mo，bmf，mef 类型的模型，以及加密模型。调用该接口打开模型之前需使用 LoadMoLibrary 加载相关依赖的模型库。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
QString str_file = "C:\\Users\admin\Documents\\MWorks\\PID_Controller.mo"
classMgr->GetMoHandler()->OpenFile(str_file.toStdWString());
```

2.1.2 NewModel

新建模型

A. 语法

```
/*
 * @brief 新建模型
 * @param [in] class_info 新建模型信息
 * @return 新建模型的键值
 */
MWint MwMoHandler::NewModel(const MwNewClassInfo& class_info);
```

B. 说明

用于新建模型文件，可指定模型名称、模型类型、模型存储路径，会自动为模型生成键值(key)。

模型键值见 3.1 章节解释。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
MwNewClassInfo new_info("model");
 new_info.strIdent = "new_model";
 new_info.strDescription = "hello world";
class_info.strMoDir = "D:/";
 bool actual_res = classMgr->GetMoHandler()->NewModel(new_info);
```

2.1.3 RenameModel

模型重命名。

A. 语法

```
/**
 * @brief  模型重命名，包括指定的引用点。
 * @param [in]cls_key         模型键值
 * @param [in]new_ident       新的模型名称
 * @param [in]ref_info        引用点集合，空集合则只重命名类型本身
 * @return 是否重命名成功
 */
bool MwMoHandler::RenameModel(MWint cls_key,
const std::string& new_ident,
const std::list<MwTypeRefInfo>& ref_info = std::list<MwTypeRefInfo>());
```

B. 说明

模型重命名，包括指定的引用点。通过模型的 key 对模型本身进行重命名，此处需要注意当重命名后模型原来的 key 将无效，需要获取新的 key，可以使用模型名获取，即调用 GetKeyByTypeName。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
std::string new_ident = "Model1"
bool res
= classMgr->GetMoHandler()->RenameModel(cls_key,new_ident);
```

2.1.4 LoadMoLibrary

打开模型库。

A. 语法

```
/*
 * @brief 打开模型库
 * @param [in] strLibName 模型库名称
 * @param [in] strLibVersion 模型库版本
 * @return 模型库是否打开成功
 */
bool MwMoHandler::LoadMoLibrary(
const std::string& str_lib_name,
const std::string& str_lib_version);
```

B. 说明

一般用于加载带版本号的模型库，例如 Modelica 3.2.1，也可加载用户自己的模型库文件，可将模型库放在 bin/Library 目录下，并按照内置库模型库的放置规则进行相关配置，即可直接使用该接口加载内置模型库。

内置模型库放置规则:(1)内置模型库放在软件安装包或者 SDK 安装包下 bin 目录中的 Library 文件夹下(2)模型库文件夹命名为名称+空格+版本号(3)模型库文件夹下放置名为 MWorksCustomLibrary.ini 内容为空的文件，该文件标志为可加载的内置库文件夹。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
bool is_success =
classMgr->GetMoHandler()->LoadMoLibrary("Modelica", "3.2.1");
```

2.1.5 SaveModel

将模型保存到文件中。

A. 语法

```
/*
 * @brief 将模型保存到文件中
 * @param [in] model_name 模型名
 * @param [in] save_encoding 采用编码方式(-1-默认(同原来模型编码方
```

```
式),0-ANSI,1-UNI16_LE,2-UNI16_BE,3-UTF_8)
 *  @param [in]  ignore_encoding_lost 是否忽略编码转换时字符丢失，默认是false
 *  @note  根据输入模型查找到模型所在文件，然后保存文件中的唯一顶层模型
 */
MWstatus MwMoHandler::SaveModel(
const std::string& model_name,
bool ignore_encoding_lost=false,
int save_encoding=-1);
```

B. 说明

保存模型，对模型进行相关修改后，如设置模型参数，需调用该接口对模型进行保存。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
ClassMgrPtr->GetMoHandler()->SaveModel(ClassMgrPtr->GetMoHandler()->GetFull
nameProp(top_class_key)) == MWStat_Ok)
```

2.1.6 UnloadModel

卸载模型

A. 语法

```
/*
 *  @brief 卸载模型
 *  @param [in]  model_name  模型名称
 *  @return 模型库是否卸载成功
 */
bool MwMoHandler::UnloadModel(const std::string& model_name);
```

B. 说明

将模型从内存中卸载，不会删除本地文件，若移除的模型处于 package 中整个 package 模型将被卸载。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
std::string mo_name = ClassMgrPtr->GetMoHandler()->GetFullnameProp(mo_key);
 ClassMgrPtr->GetMoHandler()->UnloadModel(mo_name);
```

2.1.7 DuplicateModel

模型复制

A. 语法

```
/**
 * @brief 模型复制
 * @param [in]cls_key              模型键值
 * @param [in/out]new_ident        复制后模型的新名字
 * @param [in]new_par_key          复制后模型的新父类键值
 * @param [in/out]storage_loc      复制后模型的存储位置，不指定则不自动设置新模型文件路径
 * @return new_cls_key        复制后模型的键值
 * @note 复制后的模型文件名跟模型名一致，调用者负责文件可覆盖性检查并提醒用户
 */
MWint MwMoHandler::DuplicateModel(
MWint cls_key,
const std::string& new_ident,
 MWint new_par_key,
const std::wstring& storage_loc = std::wstring());
```

B. 说明

复制模型，并可输入复制后的名称，若复制后的模型文件名跟模型名一致调用者负责文件可覆盖性检查并提醒用户。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
MwMoHandlerPtr->DuplicateModel(cls_key, ident.toStdString(), parentKey);
```

2.2 模型参数操作

模型参数操作主要为获取模型参数相关信息和值，并可修改模型相关参数值。

函数名	简介
GetParamInfoEx	获取参数信息
GetParamValue	获取模型参数值
SetParamValue	设置模型参数值

2.2.1 GetParamInfoEx

获取指定组件中的参数列表

A. 语法

```
/**
 * @brief  获取指定实例层次中的参数列表
 * @param  [in]cls_key     主模型key
 * @param  [in]name_nodes  组件名字列表，前缀节点必须是非内置类型
 * @param  [out]param_info 参数信息列表
 * @return 是否成功获取
 * @note   参量和重声明都看做参数，因此包含参量和重声明
 */
bool MwMoHandler::GetParamInfoEx(MWint cls_key, const MwStrList& name_nodes,
std::vector<MwParamInfo>* param_info);
```

B. 说明

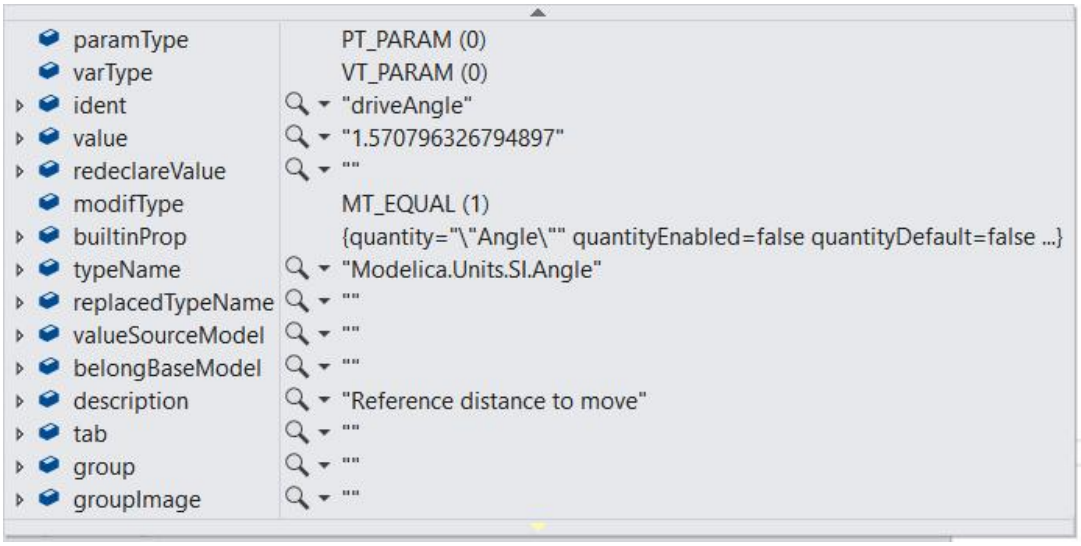
获取模型中的参数信息，传入主模型 key 并且 name_nodes 为空时可获取该模型下所有参数信息；当传入主模型 key 并且在 name_nodes 传入组件名称可获取该模型下对应组件下的所有参数。

C. 示例

(1) 获取模型 PID_Controller 的参数信息

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
MWint model_key
=classMgr->GetMoHandler()->GetKeyByTypeName("Modelica.Blocks.Examples.PID_C
ontroller");
MwStrList name_nodes;
std::vector<MwParamInfo> cur_param_info;
classMgr->GetMoHandler()->GetParamInfoEx(model_key,              name_nodes,
&cur_param_info);
```

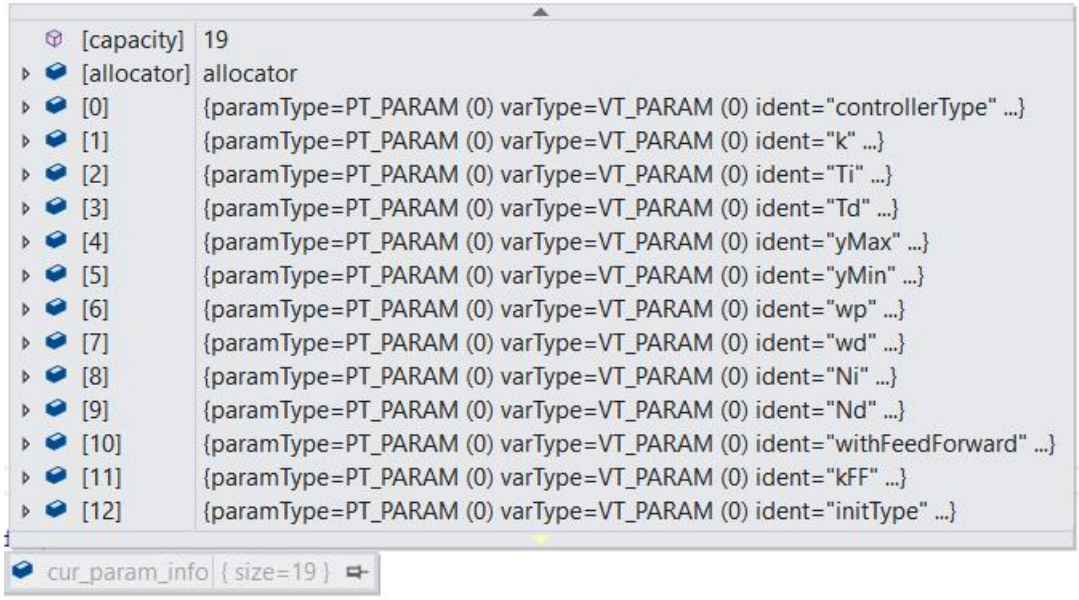
示例结果如下：



(2) 获取 PID_Controller 下组件名为”PI”的参数信息

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
MWint
model_key=classMgr->GetMoHandler()->GetKeyByTypeName("Modelica.Blocks.Examples.PID_Controller");
MwStrList name_nodes;
std::vector<MwParamInfo> cur_param_info;
name_nodes.push_back("PI");
ClassManagerPtr->GetMoHandler()->GetParamInfoEx(model_key,       name_nodes,
&cur_param_info);
```

示例结果如下：



2.2.2 GetParamValue

获取指定参数的值

A. 语法

```
/**
 * @brief  获取指定参数的值
 * @param  [in]cls_key     主模型key
 * @param  [in]name_nodes  参数全名节点列表
 * @return 字符串格式的变型值
 */
std::string  MwMoHandler::GetParamValue(MWint  cls_key,  const  MwStrList&
name_nodes);
```

B. 说明

获取指定参数值，传入主模型 key 并且 name_nodes 传入该模型下的参数名时，可获取到该参数的值；传入主模型 key 并且 name_nodes 传入该模型下的组件名和参数名时，可获取到该组件下对应的参数名的值；

C. 示例

(1) 获取模型下的参数值

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
QString model_name = "Modelica.Blocks.Examples.PID_Controller";
```

```
MWint model_key
=classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
MwStrList name_nodes;
name_nodes.push_back("driveAngle");
std::string value = classMgr->GetMoHandler()->GetParamValue(model_key,
name_nodes);
```

value = “1.570796326794897”

(2) 获取模型下 PI 组件的参数 k 的值

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
QString model_name = "Modelica.Blocks.Examples.PID_Controller";
MWint model_key
=classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
MwStrList name_nodes;
name_nodes.push_back("PI");
name_nodes.push_back("k");
std::string value = classMgr->GetMoHandler()->GetParamValue(model_key,
name_nodes);
```

value = “100”

2.2.3 SetParamValue

设置指定参数值

A. 语法

```
/**
 * @brief  设置指定参数的值
 * @param  [in]cls_key     主模型key
 * @param  [in]name_nodes  参数全名节点列表
 * @param  [in]val         参数的值
 * @return 字符串格式的变型值
 */
bool MwMoHandler::SetParamValue(
MWint cls_key,
const MwStrList& name_nodes,
const std::string& val);
```

B. 说明

修改参数值，传入主模型 key 并且 name_nodes 传入该模型下的参数名时，

可修改到该参数的值；传入主模型 key 并且 name_nodes 传入该模型下的组件名和参数名时，可修改到该组件下对应的参数名的值；修改后注意调用 SaveModel 函数将修改参数保存到模型中。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
QString model_name = "Modelica.Blocks.Examples.PID_Controller";
MWint model_key
= classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
MwStrList name_nodes;
name_nodes.push_back("PI");
name_nodes.push_back("k");
bool value = classMgr->GetMoHandler()->SetParamValue(model_key, name_nodes,
"200");
classMgr->GetMoHandler() -> SaveModel(model_name);
```

2.3 模型属性获取

对模型内部属性信息进行获取如获取模型的键值、名称、全名、模型的父类，以及模型中的注解等属性获取。

函数名	简介
GetKeyByTypeName	根据模型的名称获取模型键值
GetNestedCompKeys	获取模型下的所有组件键值
GetFullnameProp	获取元素的全名
GetNameProp	获取元素的名称
GetFullFileName	获取模型所在文件路径
GetPropAsString	获取元素描述
GetTopClassInFile	获取文件中的顶层类键值
GetTopClassInFileByKey	获取顶层父类的键值

2.3.1 GetKeyByTypeName

通过模型的类型全名获取模型键值

A. 语法

```
/**
 * @brief  通过模型的类型全名查找模型键值
 * @param  [in]type_name                类型全名
 * @return 获取到的key
 */
MWint MwMoHandler::GetKeyByTypeName(const std::string& type_name);
```

B. 说明

通过全名获取到对应的键值，若传入的为模型的全名则查找模型键值；传入模型下对应的组件全名，可查看到该组件的键值。

模型的全名和组件全名查看 3.1 章节。

C. 示例

(1) 获取模型的键值

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
QString model_name = "Modelica.Blocks.Examples.PID_Controller";
MWint model_key
= classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
```

(2) 获取模型下组件名为“PI”的键值

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
QString model_name = "Modelica.Blocks.Examples.PID_Controller.PI";
 MWint                                model_key                                =
classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
```

2.3.2 GetNestedCompKeys

获取模型下的所有组件键值

A. 语法

```
/**
```

```
 * @brief  获取模型下的所有组件键值
 * @param  [in]cls_key                模型key
 * @param  [out]vec_keys              组件key列表
 * @param  [in]include_extends        是否包含内部组件，默认不包括
 * @return 获取到的全名
 */
void MwMoHandler::GetNestedCompKeys(
MWint cls_key,
std::vector<MWint> &vec_keys,
bool include_extends = false);
```

B. 说明

获取模型下的所有组件的键值。

组件键值解释查看 3.1 章节。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
QString model_name = "Modelica.Blocks.Examples.PID_Controller";
MWint model_key
= classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
std::vector<MWint> comp_list;
classMgr->GetMoHandler()->GetNestedCompKeys(model_key, comp_list);
```

2.3.3 GetFullnameProp

获取元素的全名属性

A. 语法

```
/**
 * @brief  通过元素的key获取全名
 * @param  [in]key                模型key
 * @return 获取到的全名
 */
std::string MwMoHandler::GetFullnameProp (MWint key);
```

B. 说明

通过模型的 key 可获取到模型的全名。通过组件的 key 也可获取到组件的全

名。

模型全名和组件全名查看 3.1 章节解释。

C. 示例

获取模型的全名

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
QString model_name = "Modelica.Blocks.Examples.PID_Controller";
 MWint                                model_key                                =
classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
classMgr->GetMoHandler()->GetFullnameProp(model_key);
```

```
Modelica.Blocks.Examples.PID_Controller
```

2.3.4 GetNameProp

获取元素的名字属性

A. 语法

```
/**
 *@brief 获取元素的名字属性
 *@param [in] key 元素的键值
 *@return 名字
 */
std::string MwMoHandler::GetNameProp(MWint key);
```

B. 说明

获取元素的简名。若传入模型 key 则获取到模型的简名，若传入组件 key 则获取到组件的简名(名称)。

模型简名和模型简明查看 3.1 章节解释。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
QString model_name = "Modelica.Blocks.Examples.PID_Controller";
 MWint                                model_key                                =
classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
std::string name = classMgr->GetMoHandler()->GetNameProp(model_key);
```

```
PID_Controller
```

2.3.5 GetFullFileName

模型所在文件的路径全名

A. 语法

```
/**
 *@brief 获取模型所在文件的路径全名
 *@param [in] class_key      模型的键值
 *@param [out] full_file_name 路径全名
 */
MWstatus MwMoHandler::GetFullFileName(
MWint class_key,
std::wstring* full_file_name);
```

B. 说明

模型所在文件的路径全名。

可查看 3.1 章节对模型路径的解释进行查看。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
QString model_name = "Modelica.Blocks.Examples.PID_Controller";
MWint model_key
= classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
classMgr->GetMoHandler()->GetFullFileName (model_key) ;
```

```
D:\TYSVN\MworksSdk\MWORKS SDK 2023a(vs2017)_5.2.1.7575_Setup\MWORKS SDK
2023a(vs2017)\bin\win_msvc2017x64\Library\Modelica
3.2.1\Modelica\Blocks\package.mo
```

2.3.6 GetPropAsString

获取指定名字的属性，属性值以字符串的形式返回

A. 语法

```
/**
 *@brief 获取指定名字的属性
 *@param [in] key       元素的键值
 *@param [out]  name    获取的名称
```

```
 */
std::string MwMoHandler::GetPropAsString(MWint key, std::string& name);
```

B. 说明

获取指定名字的属性

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
QString model_name = "Modelica.Blocks.Examples.PID_Controller";
MWint model_key
= classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
classMgr->GetMoHandler()->GetPropAsString (model_key, out_name) ;
```

2.3.7 GetTopClassInFile

获取文件中的顶层类

A. 语法

```
/**
 * @brief 获取文件中的顶层类
 * @param [in]  strFile  模型文件路径
 * @return 文件中的顶层类键值
 */
MWint MwMoHandler::GetTopClassInFile(const std::wstring &file);
```

B. 说明

获取模型所在顶层类的 key 值。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
QString model_name = "Modelica.Blocks.Examples.PID_Controller";
MWint model_key
= classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
MWint key =  classMgr->GetMoHandler()->GetTopClassInFile(full_file_name);
```

2.3.8 GetTopClassInFileByKey

获取与给定类型处于同一文件的顶层父

A. 语法

```
/**
 * @brief 获取与给定类型处于同一文件的顶层父
 * @param [in] key 模型键值
 * @return 父类键值
 */
MWint MwMoHandler::GetTopClassInFileByKey(MWint key);
```

B. 说明

获取同一文件中的顶层父类的模型 key。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
QString model_name = "Modelica.Blocks.Examples.PID_Controller";
MWint model_key
= classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
MWint key = classMgr->GetMoHandler()->GetTopClassInFileByKey(model_key);
```

2.4 元素及属性判定

元素及属性判定为对模型的类型进行判定，从而确定模型的类型。

函数名	简介
IsBuiltInType	判断是否是内置类型
IsPackageType	判断是否是 package 类型
IsModelType	判断是否是 model 类型

2.4.1 IsBuiltInType

判断是否是内置类型

A. 语法

```
/**
```

```
 * @brief 判断模型是否是内置类型
 * @param [in]class_key  模型键值
 * @return 判断结果
 */
MWbool MwMoHandler::IsBuiltInType(MWint class_key)const;
```

B. 说明

判断是否是内置类型

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
QString model_name = "Modelica.Blocks.Examples.PID_Controller";
MWint model_key
= classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
bool res = classMgr->GetMoHandler()->IsBuiltInType(model_key);
```

2.4.2 IsPackageType

判断是否是 package 类型

A. 语法

```
/**
 * @brief 判断模型是否是package类型
 * @param [in]key  模型键值
 * @return 判断结果
 */
MWbool MwMoHandler::IsPackageType(MWint key)const;
```

B. 说明

判断是否是 package 类型

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
QString model_name = "Modelica.Blocks.Examples.PID_Controller";
 MWint model_key
= classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
bool res = classMgr->GetMoHandler()->IsPackageType(model_key)
```

2.4.3 IsModelType

判断是否是 model 类型

A. 语法

```
/**
 * @brief 判断模型是否是model类型
 * @param [in]key  模型键值
 * @return 判断结果
 */
MWbool MwMoHandler::IsModelType(MWint key)const;
```

B. 说明

判断是否是 model 类型

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
QString model_name = "Modelica.Blocks.Examples.PID_Controller";
MWint model_key
= classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
bool res = classMgr->GetMoHandler()->IsModelType(model_key);
```

2.5 模型属性查找

模型属性查找为在模型中查找相关属性，如查找模型中所有组件键值的列表等。

函数名	简介
LookupCompAndTypeEx	通过组件的全名查找组件类型 key
LookupConnectionsOfPort	查找端口连线

2.5.1 LookupCompAndTypeEx

获取组件及其类型键值（应用重声明）

A. 语法

```
/**
```

```
 * @brief  通过组件的全名查找组件类型键值（应用重声明）
 * @param  [in]main_key            主模型
 * @param  [in]compo_name          组件全名
 * @param  [out]class_key          组件类型键值
 * @return 查找到的key
 */
MWint MwMoHandler::LookupCompAndTypeEx(
MWint main_key,
const std::list<std::string>& compo_name,
MWint* class_key);
```

B. 说明

通过传入模型 key，在 compo_name 中传入组件名称,将获取到该组件对应的类型 key。

例如 Real a；a 为组件名，Real 为类型名，都有对应的键值。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
String model_name = "Modelica.Blocks.Examples.PID_Controller";
MWint model_key
= classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
MWint type_key = 0;
MwStrList name_nodes;
name_nodes.push_back("PI");
classMgr->GetMoHandler()->LookupCompAndTypeEx(model_key,        name_nodes,
&type_key);
```

2.5.2 LookupConnectionsOfPort

获得指定端口相连的连接线

A. 语法

```
/**
 * @brief 获得指定端口相连的连接线
 * @param  [in]key                主模型
 * @param  [in]port_key            端口的键值
 * @return 查找到该端口所有的连线key列表
 */
```

```
std::vector<MWint> MwMoHandler::LookupConnectionsOfPort(
MWint key,
const std::string& port_key);
```

B. 说明

获得指定连接器相连的连接线。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
String model_name = "Modelica.Blocks.Examples.PID_Controller";
MWint model_key
= classMgr->GetMoHandler()->GetKeyByTypeName(model_name.toStdString());
auto vecList = classMgr->GetMoHandler()->LookupConnectionsOfPort(model_key,
"PI");
```

2.6 编译仿真类

对模型实现检查模型文本、编译模型操作。

函数名	简介
CheckModel	检查模型文本
CompileModel	编译模型

2.6.1 CheckModel

检查模型

A. 语法

```
/**
 * @brief  检查模型
 * @param  [in]model_name 模型名
 * @return 模型检查结果
```

```
 */
bool MwMoHandler::CheckModel(const std::string& model_name);
```

B. 说明

检查模型文本是否存在错误，与 Sysplorer 上的仿真-检查按钮一致。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
bool checkSuccess
= ClassMgrPtr->GetMoHandler()->CheckModel(modelName.toStdString());
```

2.6.2 CompileModel

编译模型，生成求解器

A. 语法

```
/**
 * @brief  编译模型，生成求解器
 * @param  [in] model_name 模型全名
 * @param  [in] sim_inst_path 求解器实例的目录
 * @return 模型编译结果
 */
bool MwMoHandler::CompileModel(const std::string& model_name, const
std::wstring& sim_inst_path);
```

B. 说明

编译模型，生成求解器。

在对 MwSimData 初始化和仿真之前需先调用该接口，对模型进行编译。

C. 示例

```
MwClassManager* classMgr = new MwClassManager();
classMgr->Initialize();
bool compileSuccess =
ClassMgrPtr->GetMoHandler()->CompileModel(modelName.toStdString(),
simInstDir.toStdWString());
```

2.6.3 MwSimControl

仿真控制类，负责控制仿真的开始、暂停、继续、单步执行、停止等。

函数名	简介
RebindSimData	绑定仿真数据
GetSimData	获取仿真数据
IsSimIdle	判断仿真是否空闲
IsSimPaused	判断仿真是否暂停
IsSimRunning	判断仿真是否正在运行中
IsSimPausing	判断仿真是否正在暂停中
IsSimStarting	判断仿真是否正在启动中
IsSimLocked	判断仿真是否占用
StartSimulate	开始仿真
PauseSimulate	暂停仿真
ResumeSimulate	继续仿真
SimulateNextStep	仿真执行一步
StopSimulate	停止仿真
KillSimulate	强行停止仿真
SigBeforeSimStart	仿真启动信号
SigSimPaused	仿真暂停信号
SigSimStopped	仿真停止信号
SigSimResumed	仿真继续信号
SigSimTime	仿真时间点信号
SigSimLog	仿真日志信号

函数名	简介
SigSimStep	仿真单步信号

2.6.3.1 公共类型

```
enum SimScale
{
Sim_Single,
Sim_Batch
}
```

功能	仿真规模	
参数	Sim_Single	仿真规模为单次仿真
	Sim_Batch	仿真规模为批量仿真

```
enum SimMode
{
Sim_ContinueMode,
Sim_RealtimeMode,
Sim_InteractiveStepMode,
Sim_InteractiveRealTimeMode,
Sim_SavetoDBModel
}
```

功能	仿真模式	
参数	Sim_ContinueMode	持续仿真模式
	Sim_RealtimeMode	实时仿真模式
	Sim_InteractiveStepMode	交互单步仿真模式
	Sim_InteractiveRealTimeMode	交互实时仿真模式
	Sim_SavetoDBModel	通信仿真模式

```
enum ExitCode
{
EXIT_NORMAL,
KERNEL_EXIT_FATAL = 1,
KERNEL_EXIT_ALLOC_FAILED,
```

```
KERNEL_EXIT_INST_FAILED,
KERNEL_EXIT_INIT_FAILED,
KERNEL_EXIT_STEP_FAILED,
SOLVER_EXIT_INALID_CMD_LINE = 101,
SOLVER_EXIT_LOAD_SHARED_FAILED,
SOLVER_EXIT_CREATE_RESULT_FAILED,
SOLVER_EXIT_NO_CONFIG_FILE,
SOLVER_EXIT_INST_FAILED,
EXIT_IOTDB_FAIL,
EXIT_IOTDB_TIMES_FAIL,
EXIT_UNKOWN_ERROR = 999
}
```

功能	仿真退出代码	
参数	EXIT_NORMAL	正常退出
	KERNEL_EXIT_FATAL	严重错误而退出
	KERNEL_EXIT_ALLOC_FAILED	内存分配失败而退出
	KERNEL_EXIT_INST_FAILED	实例化失败而退出
	KERNEL_EXIT_INIT_FAILED	初始化失败而退出
	KERNEL_EXIT_STEP_FAILED	单步计算失败而退出
	SOLVER_EXIT_INALID_CMD_LINE	无效的命令行
	SOLVER_EXIT_LOAD_SHARED_FAILED	加载模型求解器共享库失败
	SOLVER_EXIT_CREATE_RESULT_FAILED	创建结果文件失败
	SOLVER_EXIT_NO_CONFIG_FILE	XML 配置文件不存在
	SOLVER_EXIT_INST_FAILED	求解器实例化失败
	EXIT_IOTDB_FAIL	启动 IoTDB 失败
	EXIT_IOTDB_TIMES_FAIL	插入 IoTDB 时间序列出错
	EXIT_UNKOWN_ERROR	未知错误

2.6.3.2 公共函数

2.6.3.2.1 RebindSimData

去除已绑定的 SimData，并与新 SimData 建立单向引用

A. 语法

```
/**
 * @brief  去除已绑定的SimData，并与新SimData建立单向引用
 * @param  [in] data 新绑定的SimData
*/
void RebindSimData(MwSimData *data)
```

B. 说明

去除已绑定的 SimData，并与新 SimData 建立单向引用。

MwSimControl 用于控制仿真，一般情况只需要 new 出一个该对象即可，若想对多个仿真数据进行保存，可通过该接口切换绑定的仿真数据。

C. 示例

在调用 RebindSimData 之前先调用 GetSimData 获取一次 MwSimControl 当前绑定的 MwSimData，然后创建一个新的 MwSimData 并调用 RebindSimData 接口将其绑定到 MwSimControl 上，再调用 GetSimData 接口查看当前 MwSimControl 绑定的 MwSimData。

```
MwSimControl *sim_ctrl = new MwSimControl;
MwSimData *before_data = sim_ctrl->GetSimData();
MwSimData       *sim_data       =       new       MwSimData(L"PID_Controller",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller/Result.msr",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller");
sim_ctrl->RebindSimData(sim_data);
MwSimData *after_data = sim_ctrl->GetSimData();
```

before_data 为空指针，sim_data 和 after_data 指向的为同一个 MwSimDta 对象。

before_data	0x0000000000000000 <NULL>	MwSimData *
sim_data	0x000001a35e8ea690 {simInstId=3452816845 hasInitialized=false originModel="" ...}	MwSimData *
after_data	0x000001a35e8ea690 {simInstId=3452816845 hasInitialized=false originModel="" ...}	MwSimData *

2.6.3.2.2 GetSimData

获取当前绑定的 SimData

A. 语法

```
MwSimData* GetSimData()
```

B. 说明

获取当前绑定的 SimData

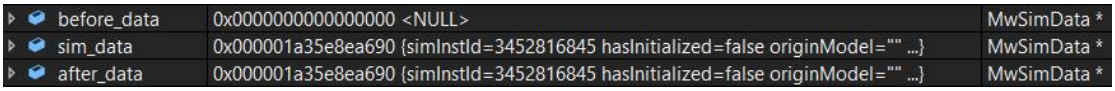
C. 示例

在调用 RebindSimData 之前先调用 GetSimData 获取一次 MwSimControl 当前绑定的 MwSimData，然后创建一个新的 MwSimData 并调用 RebindSimData 接口将其绑定到 MwSimControl 上，再调用 GetSimData 接口查看当前 MwSimControl 绑定的 MwSimData。

```
MwSimControl *sim_ctrl = new MwSimControl;
MwSimData       *sim_data       =       new       MwSimData(L"PID_Controller",
```

```
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller/Result.msr",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller");
sim_ctrl->RebindSimData(sim_data);
MwSimData *after_data = sim_ctrl->GetSimData();
```

- before_data 为空指针，sim_data 和 after_data 指向的为同一个 MwSimDta 对象，符合预期结果。



2.6.3.2.3 IsSimIdle

判断仿真是否空闲

A. 语法

```
bool IsSimIdle()
```

B. 说明

在开始仿真前调用 MwSimControl 的 IsSimIdel 接口获取当前仿真是否空闲

C. 示例

在开始仿真前调用 MwSimControl 的 IsSimIdel 接口获取当前仿真是否空闲，开始仿真后再次调用 IsSimIdle 接口查看仿真是否仍空闲。

```
MwSimControl *sim_ctrl = new MwSimControl;
wClassManager *class_mgr = new MwClassManager;
class_mgr->Initialize();
MwSimControl *sim_ctrl = new MwSimControl;
MwSimData       *sim_data       =       new       MwSimData(L"PID_Controller",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller/Result.msr",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller");
sim_data->InitializeSimInst();
sim_ctrl->RebindSimData(sim_data);
bool is_sim_idle_1 = sim_ctrl->IsSimIdle();
sim_ctrl->StartSimulate(MwSimControl::Sim_RealtimeMode);
sim_ctrl->PauseSimulate();
bool is_sim_idle_2 = sim_ctrl->IsSimIdle();
```

- is_sim_idle_1 为真，is_sim_idle_2 为假，符合预期结果。

2.6.3.2.4 IsSimPaused

判断仿真是否已经暂停

A. 语法

```
bool IsSimPaused()
```

B. 说明

判断仿真是否暂停，而 IsSimPausing 则是判断仿真是否正在暂停过程中。

2.6.3.2.5 IsSimRunning

判断仿真是否正在运行

A. 语法

```
bool IsSimRunning()
```

B. 说明

判断仿真是否正在运行

C. 示例

```
MwSimControl *sim_ctrl = new MwSimControl;
MwClassManager *class_mgr = new MwClassManager;
class_mgr->Initialize();
MwSimControl *sim_ctrl = new MwSimControl;
MwSimData *sim_data = new MwSimData(L"PID_Controller",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller/Result.msr",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller");
sim_data->InitializeSimInst();
sim_ctrl->RebindSimData(sim_data);
bool is_sim_running_1 = sim_ctrl->IsSimRunning();
sim_ctrl->StartSimulate(MwSimControl::Sim_RealtimeMode);
bool is_sim_running_2 = sim_ctrl->IsSimRunning();
```

is_sim_ running _1 为假，is_sim_ running _2 为真，符合预期结果。

is_sim_running_1	false	bool
is_sim_running_2	true	bool

2.6.3.2.6 IsSimPausing

判断仿真是否正在暂停

A. 语法

```
bool IsSimPausing()
```

B. 说明

判断仿真是否正在暂停

C. 示例

```
MwClassManager *class_mgr = new MwClassManager;
class_mgr->Initialize();
MwSimControl *sim_ctrl = new MwSimControl;
MwSimData       *sim_data       =       new       MwSimData(L"PID_Controller",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller/Result.msr",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller");
sim_data->InitializeSimInst();
sim_ctrl->RebindSimData(sim_data);
sim_ctrl->StartSimulate(MwSimControl::Sim_RealtimeMode);
sim_ctrl->PauseSimulate();
bool is_sim_paused_2 = sim_ctrl->IsSimPausing();
```

is_sim_paused_1 为假，is_sim_paused_2 为真，符合预期结果。

is_sim_paused_1	false	bool
is_sim_paused_2	true	bool

2.6.3.2.7 IsSimStarting

判断仿真是否正在启动

A. 语法

```
bool IsSimStarting()
```

B. 说明

判断仿真是否正在启动

2.6.3.2.8 IsSimLocked

判断仿真是否占用

A. 语法

```
bool IsSimLocked()
```

B. 说明

判断仿真是否占用

2.6.3.2.9 StartSimulate

开始仿真

A. 语法

```
bool StartSimulate(
SimMode mode = Sim_ContinueMode,
SimScale scale = Sim_Single)
```

B. 说明

开始仿真

C. 示例

```
MwSimControl *sim_ctrl = new MwSimControl;
MwClassManager *class_mgr = new MwClassManager;
class_mgr->Initialize();
MwSimControl *sim_ctrl = new MwSimControl;
MwSimData       *sim_data       =       new       MwSimData(L"PID_Controller",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller/Result.msr",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller");
sim_data->InitializeSimInst();
sim_ctrl->RebindSimData(sim_data);
MwSimData *before_data = sim_ctrl->GetSimData();
MwSimData       *sim_data       =       new       MwSimData(L"PID_Controller",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller/Result.msr",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller");
sim_ctrl->RebindSimData(sim_data);
MwSimData *after_data = sim_ctrl->GetSimData()
```

is_sim_idle_1 为真，is_sim_idle_2 为假，符合预期结果。

is_sim_idle_1	true	bool
is_sim_idle_2	false	bool

2.6.3.2.10 PauseSimulate

暂停仿真

A. 语法

```
bool PauseSimulate()
```

B. 说明

暂停仿真

C. 示例

```
MwSimControl *sim_ctrl = new MwSimControl;
MwClassManager *class_mgr = new MwClassManager;
class_mgr->Initialize();
MwSimControl *sim_ctrl = new MwSimControl;
MwSimData *sim_data = new MwSimData(L"PID_Controller",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller/Result.msr",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller");
sim_data->InitializeSimInst();
sim_ctrl->RebindSimData(sim_data);
bool is_sim_paused_1 = sim_ctrl->IsSimPausing();
sim_ctrl->StartSimulate(MwSimControl::Sim_RealtimeMode);
sim_ctrl->PauseSimulate();
bool is_sim_paused_2 = sim_ctrl->IsSimPausing();
```

is_sim_paused_1 为假，is_sim_paused_2 为真

2.6.3.2.11 ResumeSimulate

继续仿真

A. 语法

```
bool ResumeSimulate()
```

B. 说明

继续仿真，应在暂停后使用该接口继续仿真。

2.6.3.2.12 SimulateNextStep

仿真执行一步

A. 语法

```
bool SimulateNextStep()
```

B. 说明

仿真执行一步，在交互仿真模式中进行使用。

2.6.3.2.13 StopSimulate

停止仿真

A. 语法

```
void StopSimulate()
```

B. 说明

异步停止仿真

2.6.3.2.14 KillSimulate

强行停止仿真。

A. 语法

```
void KillSimulate()
```

B. 说明

停止仿真

2.6.3.3 信号

SDK 架构代码中对程序执行一定状态所发送的 QT 信号，开发者可通过信号槽获取信号执行自己的代码逻辑

2.6.3.3.1 SigBeforeSimStart

仿真启动前信号

A. 语法

```
void SigBeforeSimStart()
```

B. 说明

仿真启动前信号。

2.6.3.3.2 SigSimPaused

仿真暂停信号

A. 语法

```
void SigSimPaused()
```

B. 说明

仿真暂停信号

2.6.3.3.3 SigSimStopped

仿真停止信号

A. 语法

```
void SigSimStopped(int exit_code)
```

B. 说明

仿真停止信号

2.6.3.3.4 SigSimResumed

仿真继续信号

A. 语法

```
void SigSimResumed()
```

B. 说明

仿真继续信号

2.6.3.3.5 SigSimTime

仿真时间点信号

A. 语法

```
void SigSimTime(double cur_time)
```

B. 说明

仿真时间点信号

2.6.3.3.6 SigSimLog

仿真日志信号

A. 语法

```
void SigSimLog(const QString &log)
```

B. 说明

仿真日志信号

2.6.3.3.7 SigSimStep

仿真单步信号

A. 语法

```
void SigSimLog(double cur_time)
```

B. 说明

仿真单步信号

2.7 结果数据查询类

2.7.1 MwSimData

模型仿真结果类，包含模型所有变量的仿真结果数据。

函数名	简介
ApplyExperimentData	应用仿真设置
GetVarTreeRoot	获取仿真后数据根节点
InitializeSimInst	初始化仿真实例
GetVarData	读取结果变量

2.7.1.1 公共类型

```
Enum SimResultStatus
{
Sim_Empty,
Sim_Succeeded,
Sim_Stopped,
Sim_Failed,
Sim_Writing,
Sim_Paused
}
```

功能	仿真仿真结果状态

参数	Sim_Empty 仿真实例为空 Sim_Succeeded 仿真成功 Sim_Stopped 仿真停止 Sim_Failed 仿真失败 Sim_Writing 仿真进行中 Sim_Paused 仿真暂停

2.7.1.2 公共函数

2.7.1.2.1 ApplyExperimentData

应用仿真设置，可设置仿真开始时间、停止时间、仿真步长、算法、积分算法步长。

A. 语法

```
/**
 * @brief  应用仿真设置
 * @param  [in]exp_data仿真设置数据
 */
void ApplyExperimentData(MwExperimentData *exp_data)
```

B. 说明

应用仿真设置

C. 示例

```
MwSimData *sim_data = new MwSimData(L"PID_Controller",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller/Result.msr",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller");
sim_data->InitializeSimInst();
MwExperimentData *exp_data = new MwExperimentData;
exp_data->stopTime = 10;
exp_data->intervalLength = 0.01;
exp_data->numberOfIntervals = 0;
exp_data->algorithm = "Radau5";
exp_data->tolerance = 0.0002;
sim_data->ApplyExperimentData(exp_data);
```

2.7.1.2.2 GetVarTreeRoot

获取结果变量树的根节点

A. 语法

```
popro::MwVarTree* GetVarTreeRoot()
```

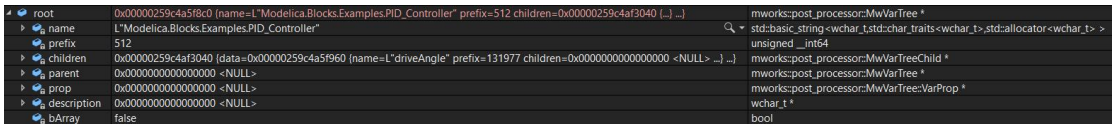
B. 说明

获取结果变量树的根节点

C. 示例

```
MwSimData *sim_data = new MwSimData(L"PID_Controller",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller/Result.msr",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller");
sim_data->InitializeSimInst();
popro::MwVarTree *root = sim_data->GetVarTreeRoot();
```

获取到结果变量树的根节点信息。



2.7.1.2.3 InitializeSimInst

初始化仿真实例

A. 语法

```
bool InitializeSimInst()
```

B. 说明

初始化仿真实例

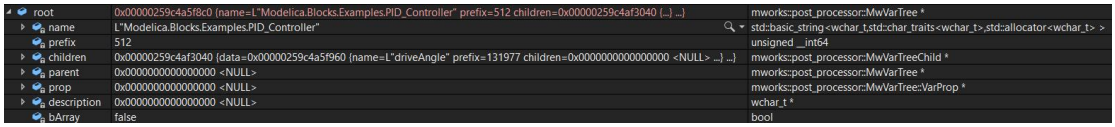
C. 示例

用 MwSimData 的 InitializeSimInst 接口进行初始化，然后调用 GetVarTreeRoot 接口，获取结果变量树的根节点。

```
MwSimData *sim_data = new MwSimData(L"PID_Controller",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller/Result.msr",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller");
```

```
sim_data->InitializeSimInst();
popro::MwVarTree *root = sim_data->GetVarTreeRoot();
```

获取到结果变量树的根节点信息，符合预期结果。



2.7.1.2.4 GetVarData

读取结果变量

A. 语法

```
/**
 * @brief  获取变量结果
 * @param  [in]var_name仿真设置数据
 * @param  [out]times 时间点数组
 * @param  [out]values 对应时间点的值
 */
bool GetVarData(
const std::wstring &var_name,
std::vector<Mwdouble> &times,
std::vector<Mwdouble> &values)
```

B. 说明

读取结果变量

C. 示例

```
MwSimData *sim_data = new MwSimData(L"PID_Controller",
L"C:/Users/TR/Documents/MWORKS/WorkSpace/PID_Controller/Result.msr");
sim_data->InitializeSimInst();
std::vector<MWdouble> times, values;
sim_data->GetVarData(L"PI.u_s", times, values);
```

获取到变量“PI.u_s”的仿真结果数据点集，

名称	值	类型	名称	值	类型
times	{ size=509 }	std::vector	values	{ size=509 }	std::vecto
[capacity]	509	__int64	[capacity]	509	__int64
[allocator]	allocator	std::_Comp	[allocator]	allocator	std::_Com
[0]	0.0000000000000000	double	[0]	0.0000000000000000	double
[1]	0.0080000000000000002	double	[1]	0.0000000000000000	double
[2]	0.016000000000000000	double	[2]	0.0000000000000000	double
[3]	0.024000000000000000	double	[3]	0.0000000000000000	double
[4]	0.032000000000000001	double	[4]	0.0000000000000000	double
[5]	0.040000000000000001	double	[5]	0.0000000000000000	double
[6]	0.048000000000000001	double	[6]	0.0000000000000000	double
[7]	0.056000000000000001	double	[7]	0.0000000000000000	double
[8]	0.064000000000000001	double	[8]	0.0000000000000000	double
[9]	0.072000000000000008	double	[9]	0.0000000000000000	double
[10]	0.080000000000000002	double	[10]	0.0000000000000000	double
[11]	0.087999999999999995	double	[11]	0.0000000000000000	double
[12]	0.096000000000000002	double	[12]	0.0000000000000000	double
[13]	0.10400000000000001	double	[13]	0.0000000000000000	double
[14]	0.11200000000000000	double	[14]	0.0000000000000000	double
[15]	0.12000000000000000	double	[15]	0.0000000000000000	double
[16]	0.12800000000000000	double	[16]	0.0000000000000000	double
[17]	0.13600000000000001	double	[17]	0.0000000000000000	double
[18]	0.14400000000000002	double	[18]	0.0000000000000000	double
[19]	0.15200000000000000	double	[19]	0.0000000000000000	double
[20]	0.16000000000000000	double	[20]	0.0000000000000000	double
[21]	0.16800000000000001	double	[21]	0.0000000000000000	double
[22]	0.17599999999999999	double	[22]	0.0000000000000000	double
[23]	0.18400000000000000	double	[23]	0.0000000000000000	double
[24]	0.19200000000000000	double	[24]	0.0000000000000000	double
[25]	0.20000000000000001	double	[25]	0.0000000000000000	double
[26]	0.20800000000000002	double	[26]	0.0000000000000000	double
[27]	0.21600000000000000	double	[27]	0.0000000000000000	double
[28]	0.22400000000000000	double	[28]	0.0000000000000000	double
[29]	0.23200000000000001	double	[29]	0.0000000000000000	double
[30]	0.23999999999999999	double	[30]	0.0000000000000000	double
[31]	0.24800000000000000	double	[31]	0.0000000000000000	double

2.7.2 MwVarTree

结果变量树节点类，用于获取仿真结果变量树结构，一般不主动创建该对象实例。

函数名	简介
Name	变量名字
GetFullName	获取变量全名
Parent	获取父节点
Description	获取描述
Unit	单位
Children	孩子节点

2.7.2.1 公共函数

2.7.2.1.1 Name

获取变量名字

A. 语法

```
const std::wstring& Name()
```

B. 说明

获取变量名字

C. 示例

(1) 模型树的根节点的 Name

```
popro::MwVarTree *var_tree = sim_data->GetVarTreeRoot();
std::wstring name = var_tree->Name();
```

name = Examples.Controller

(2) 获取模型中组件的 Name

```
std::vector<popro::MwVarTree*> vec_children;
   var_tree->Children(&vec_children);
std::wstring pi_name = vec_children[1]->Name();
```

name = PI

(3) 获取模型中组件下变量的 Name

```
std::vector<popro::MwVarTree*> pi_vec_children;
   vec_children[1]->Children(&pi_vec_children);
std::wstring um_name = pi_vec_children[1]->Name();
```

name =u_m

2.7.2.1.2 GetFullName

获取变量全名

A. 语法

```
std::wstring GetFullName()
```

B. 说明

获取变量全名

C. 示例

(1) 获取模型的根节点的 FullName

```
ppro::MwVarTree *var_tree = sim_data->GetVarTreeRoot();
std::wstring full_name = var_tree->GetFullName();
```

(2) 获取模型中组件 PI 的 FullName

```
std::vector<popro::MwVarTree*> vec_children;
   var_tree->Children(&vec_children);
std::wstring pi_full_name = vec_children[1]->GetFullName();
```

(3) 获取模型中组件 PI 下变量 u_m 的 FullName

```
std::vector<popro::MwVarTree*> pi_vec_children;
   vec_children[1]->Children(&pi_vec_children);
std::wstring um_full_name = pi_vec_children[1]->GetFullName();
```

2.7.2.1.3 Parent

获取父节点

A. 语法

```
MwVarTree* Parent()
```

B. 说明

获取父节点

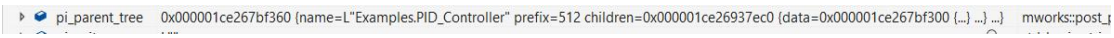
C. 示例

(1) 模型的根节点的 Parent

```
popro::MwVarTree *var_tree = sim_data->GetVarTreeRoot();
auto  parent_tree = var_tree->Parent();
```

(2) 获取模型中组件 PI 的 Parent

```
std::vector<popro::MwVarTree*> vec_children;
   var_tree->Children(&vec_children);
auto  pi_parent_tree = vec_children[1]->Parent();
```



(3) 获取模型中组件 PI 下变量 u_m 的 Parent

```
std::vector<popro::MwVarTree*> pi_vec_children;
   vec_children[1]->Children(&pi_vec_children);
auto um_parent_tree = pi_vec_children[1]->Parent();
```

2.7.2.1.4 Description

获取变量描述

A. 语法

```
const MWwchar* Description()
```

B. 说明

获取变量描述

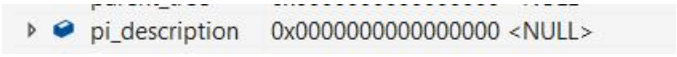
C. 示例

(1) 获取模型的根节点的 Description

```
popro::MwVarTree *var_tree = sim_data->GetVarTreeRoot();
auto description = var_tree->Description();
```

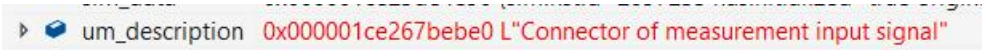
(2) 获取模型中组件 PI 的 Description

```
std::vector<popro::MwVarTree*> vec_children;
   var_tree->Children(&vec_children);
auto pi_description = vec_children[1]->Description();
```



(3) 获取模型中组件 PI 下变量 u_m 的 Description

```
std::vector<popro::MwVarTree*> pi_vec_children;
   vec_children[1]->Children(&pi_vec_children);
auto um_description = pi_vec_children[1]->Description();
```



2.7.2.1.5 Unit

获取变量单位

A. 语法

```
std::wstring Unit()
```

B. 说明

获取变量单位

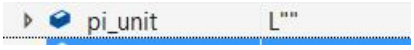
C. 示例

(1) 获取模型的根节点的 Unit

```
popro::MwVarTree *var_tree = sim_data->GetVarTreeRoot();
std::wstring unit = var_tree->Unit();
```

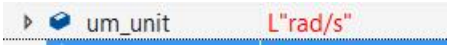
(2) 获取模型中组件 PI 的 Unit

```
std::vector<popro::MwVarTree*> vec_children;
   var_tree->Children(&vec_children);
std::wstring pi_unit = vec_children[1]->Unit();
```



(3) 获取模型中组件 PI 下变量 u_m 的 Unit

```
std::vector<popro::MwVarTree*> pi_vec_children;
   vec_children[1]->Children(&pi_vec_children);
std::wstring um_unit = pi_vec_children[1]->Unit();
```



2.7.2.1.6 Children

获取当前变量节点的所有孩子

A. 语法

```
void Children(std::vector<MwVarTree*>* vec_children)
```

B. 说明

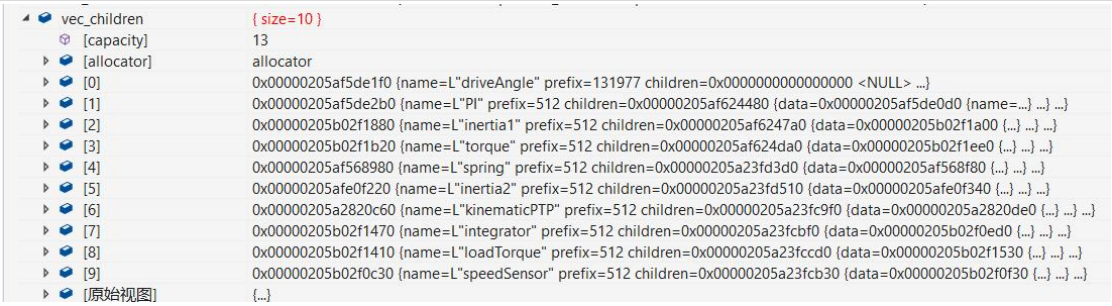
获取当前变量节点的所有孩子

C. 示例

(1) 获取模型的根节点的 Children

```
popro::MwVarTree *var_tree = sim_data->GetVarTreeRoot();

std::vector<popro::MwVarTree*> vec_children;
```

```
    var_tree->Children(&vec_children);
```

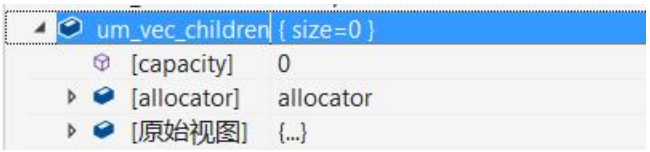


(2) 获取模型中组件 PI 的 Children

```
popro::MwVarTree *var_tree = sim_data->GetVarTreeRoot();
std::vector<popro::MwVarTree*> pi_vec_children;
vec_children[1]->Children(&pi_vec_children);
```

(3) 获取模型中组件 PI 下变量 u_m 的 Children

```
popro::MwVarTree *var_tree = sim_data->GetVarTreeRoot();
std::vector<popro::MwVarTree*> um_vec_children;
   pi_vec_children[1]->Children(&um_vec_children);
```



2.8 图形组件类

2.8.1 MwMoGraphicsViewController

模型视图管理类，负责显示模型的图标视图，组件视图和文本视图，并提供模型编辑功能。

函数名	简介

函数名	简介
CloseMoWindow	关闭模型窗口
OpenMoWindow	打开模型窗口
CloseCurrentWindow	关闭当前窗口
CloseAllWindow	关闭所有窗口
SetMdiInterface	设置视图接口
SetClassDirty	设置脏标
GetCurrentClassKey	获取当前模型 key
SaveCurrentWindow	保存当前模型
SigUpdate	模型视图更新信号
SigClassDirtyChanged	脏标变化信号
SigAppendClass	添加模型信号
SigRemoveClass	移除模型信号
SigReplaceClass	替换模型信号

2.8.1.1 公共类型

```
enum MwModelicaLayer
{
MLAYER_ICON,
MLAYER_DIAGRAM,
MLAYER_TEXT,
MLAYER_DOCUMENT,
MLAYER_UNKNOW
}
```

功能	模型视图层次

参数	MLAYER_ICON 模型图标图层 MLAYER_DIAGRAM 模型组件图层 MLAYER_TEXT 模型文本图层 MLAYER_DOCUMENT 模型帮助图层 MLAYER_UNKNOW

```
enum mogv::MoGvEvent::Type
{
None = 0,
SelectChanged,
ComponentLevelChanged,
ViewChanged,
MoWindowChanged,
ActivateWindowChanged,
ModelDataChanged,
ModelAppended,
ModelRemoved
}
```

功能	模型视图活动
参数	None 不合法的活动 SelectChanged 组件选择变化 ComponentLevelChanged 组件层次变化 ViewChanged 视图变化 MoWindowChanged 模型窗口切换 ActivateWindowChanged 活动窗口变化 ModelDataChanged 模型数据变化 ModelAppended 模型打开或新建 ModelRemoved 卸载顶层模型或删除嵌套模型

2.8.1.2 公共函数

2.8.1.2.1 CloseMoWindow

关闭模型窗口

A. 语法

```
/**
```

```
 * @brief 关闭模型窗口
 * @param [in] main_key 模型key
 */
void CloseMoWindow(MWint main_key)
```

B. 说明

关闭模型窗口

C. 示例

```
MwClassManager *class_manager = new MwClassManager;
class_manager->Initialize();
class_manager->GetMoHandler()->LoadMoLibrary("Modelica", "3.2.1");
class_manager->GetMoHandler()->OpenFile(L"C:/Users/TR/Documents/MWORKS/WorkS
pace/PID_Controller.mo");
MWint mo_key
=
class_manager->GetMoHandler()->GetTopClassInFile(L"C:/Users/TR/Documents/MWO
RKS/WorkSpace/PID_Controller.mo");

MwMoGraphicsViewController *mo_controller
= new MwMoGraphicsViewController(class_manager);
MwMoWindowMdi *mo_mdi = new MwMoWindowMdi(mo_controller);
mo_controller->SetMdiInterface(mo_mdi);

mo_controller->OpenMoWindow(mo_key);

main_win.setCentralWidget(mo_mdi);
main_win.showMaximized();
mo_controller->CloseMoWindow(mo_key);
```

2.8.1.2.2 OpenMoWindow

打开模型窗口

A. 语法

```
/**
 * @brief 打开模型窗口
 * @param[in] key                   需要打开的模型key
 * @param[in] in_new_tab            是否在新窗口中打开模型
 * @param[in] layer                 打开模型的图层
```

```
 */
void OpenMoWindow(MWint key, bool in_new_tab = false, MwModelicaLayer layer =
MLAYER_UNKNOW)
```

B. 说明

打开模型窗口

C. 示例

可参加 CloseMoWindow 接口中如何打开窗口。

2.8.1.2.3 CloseCurrentWindow

关闭当前模型窗口

A. 语法

```
void CloseCurrentWindow()
```

B. 说明

关闭当前模型窗口

C. 示例

```
MwClassManager *class_manager = new MwClassManager;
class_manager->Initialize();
class_manager->GetMoHandler()->LoadMoLibrary("Modelica", "3.2.1");
class_manager->GetMoHandler()->OpenFile(L"C:/Users/TR/Documents/MWORKS/WorkS
pace/PID_Controller.mo");
MWint mo_key
=
class_manager->GetMoHandler()->GetTopClassInFile(L"C:/Users/TR/Documents/MWO
RKS/WorkSpace/PID_Controller.mo");

MwMoGraphicsViewController *mo_controller
= new MwMoGraphicsViewController(class_manager);
MwMoWindowMdi *mo_mdi = new MwMoWindowMdi(mo_controller);
mo_controller->SetMdiInterface(mo_mdi);

mo_controller->OpenMoWindow(mo_key);

main_win.setCentralWidget(mo_mdi);
main_win.showMaximized();
```

```
mo_controller->CloseCurrentWindow();
```

2.8.1.2.4 CloseAllWindow

关闭所有模型窗口

A. 语法

```
void CloseAllWindow()
```

B. 说明

关闭所有模型窗口

C. 示例

```
MwClassManager *class_manager = new MwClassManager;
class_manager->Initialize();
class_manager->GetMoHandler()->LoadMoLibrary("Modelica", "3.2.1");
class_manager->GetMoHandler()->OpenFile(L"C:/Users/TR/Documents/MWORKS/WorkS
pace/PID_Controller.mo");
MWint mo_key
=
class_manager->GetMoHandler()->GetTopClassInFile(L"C:/Users/TR/Documents/MWO
RKS/WorkSpace/PID_Controller.mo");

MwMoGraphicsViewController *mo_controller
= new MwMoGraphicsViewController(class_manager);
MwMoWindowMdi *mo_mdi = new MwMoWindowMdi(mo_controller);
mo_controller->SetMdiInterface(mo_mdi);

mo_controller->OpenMoWindow(mo_key);

main_win.setCentralWidget(mo_mdi);
main_win.showMaximized();
mo_controller->CloseAllWindow();
```

2.8.1.2.5 SetMdiInterface

设置视图接口类

A. 语法

```
/**
 * @brief 设置视图接口类
 * @param [in]mdi 视图接口
 */
void SetMdiInterface(MwMdiInterface *mdi)
```

B. 说明

设置视图接口类，创建模型控制器后需调用该接口设置一个模型视图。

C. 示例

参数上述示例中设置该接口。

2.8.1.2.6 SetClassDirty

设置模型脏标志

A. 语法

```
/**
 * @brief 设置模型脏标志
 * @param [in]class_key 模型key
 * @param [in]dirty     是否设置脏标
 */
void SetClassDirty(MWint class_key, bool dirty)
```

B. 说明

设置模型脏标志，当中央视图的模型发送改变后，可调用该接口将模型树设置为脏标。当模型进行修改后，模型树上应对应有所修改。

C. 示例

```
MwClassManager *class_manager = new MwClassManager;
class_manager->Initialize();
class_manager->GetMoHandler()->LoadMoLibrary("Modelica", "3.2.1");
class_manager->GetMoHandler()->OpenFile(L"C:/Users/TR/Documents/MWORKS/WorkS
pace/PID_Controller.mo");
MWint mo_key
=
class_manager->GetMoHandler()->GetTopClassInFile(L"C:/Users/TR/Documents/MWO
```

```
RKS/WorkSpace/PID_Controller.mo");

MwMoGraphicsViewController *mo_controller
= new MwMoGraphicsViewController(class_manager);
MwMoWindowMdi *mo_mdi = new MwMoWindowMdi(mo_controller);
mo_controller->SetMdiInterface(mo_mdi);

mo_controller->OpenMoWindow(mo_key);

main_win.setCentralWidget(mo_mdi);
main_win.showMaximized();
mo_controller->SetClassDirty(mo_key,true);
```

2.8.1.2.7 GetCurrentClassKey

获取当前窗口的模型 key

A. 语法

```
MWint GetCurrentClassKey()
```

B. 说明

获取当前窗口的模型 key

C. 示例

```
MwClassManager *class_manager = new MwClassManager;
class_manager->Initialize();
class_manager->GetMoHandler()->LoadMoLibrary("Modelica", "3.2.1");
class_manager->GetMoHandler()->OpenFile(L"C:/Users/TR/Documents/MWORKS/WorkS
pace/PID_Controller.mo");
MWint mo_key
=
class_manager->GetMoHandler()->GetTopClassInFile(L"C:/Users/TR/Documents/MWO
RKS/WorkSpace/PID_Controller.mo");

MwMoGraphicsViewController *mo_controller
= new MwMoGraphicsViewController(class_manager);
MwMoWindowMdi *mo_mdi = new MwMoWindowMdi(mo_controller);
mo_controller->SetMdiInterface(mo_mdi);
```

```
mo_controller->OpenMoWindow(mo_key);

main_win.setCentralWidget(mo_mdi);
main_win.showMaximized();
MWint cur_key = mo_controller->GetCurrentClassKey();
```

2.8.1.2.8 SaveCurrentWindow

保存当前窗口的模型

A. 语法

```
/**
 * @brief 保存当前模型
 * @param [in] close_window 是否关闭窗口
 * @param [in] show_error_dlg 是否弹出错误提示对话框（如果模型保存失败）
 */
bool SaveCurrentWindow(
bool close_window = false,
bool show_error_dlg = true)
```

B. 说明

保存当前窗口的模型

C. 示例

参考上述示例。

2.8.1.3 信号

2.8.1.3.1 SigUpdate

模型视图更新信号，报告模型视图中发生的行为以供其他面板刷新

A. 语法

```
/*
**  @brief 模型视图更新信号，报告模型视图中发生的行为以供其他面板刷新
*   @param [in]mo_event 模型视图行为类型
*   @param [in]class_key_list 模型视图的行为操作对象key列表
**  @note触发UpdateType类型的信号，激活连接信号的面板同步刷新
    */
void SigUpdate(mogv::MoGvEvent mo_event, Qlist<MWint> class_key_list)
```

B. 说明

模型视图更新信号，报告模型视图中发生的行为以供其他面板刷新

2.8.1.3.2 SigClassDirtyChanged

模型脏标志变化信号

A. 语法

```
void SigClassDirtyChanged(MWint class_key)
```

B. 说明

模型脏标志变化信号

2.8.1.3.3 SigAppendClass

添加模型信号

A. 语法

```
void SigAppendClass(MWint key)
```

B. 说明

添加模型信号

2.8.1.3.4 SigRemoveClass

移除模型信号

A. 语法

```
void SigRemoveClass(MWint key)
```

B. 说明

内核移除模型信号

2.8.1.3.5 SigReplaceClass

内核替换模型信号

A. 语法

```
void SigReplaceClass(MWint key)
```

B. 说明

替换模型信号

2.8.2 MwMoWindowMdi

中央视图控件，负责管理模型视图窗口，创建后设置到 MwMoGraphicsViewController 中。该类只负责提供视图，用户无需对该窗口有其他操作，所有对窗口操作可通过 MwMoGraphicsViewController 完成。

2.8.3 MwMoClassTreeModel

模型树数据类，用于将内核的模型数据操作同步到界面模型中，利用 QT 所提供的 MVD 设计模式，提供了一个现成的 Model 自动管理数据，用户只需实现自己的 View 视图用于展示数据并将 Model 设置到 View 视图中。

函数名	简介
SetClassifyName	设置分类
AppendTopClass	增加顶层模型
GetTopItems	获取所有顶层节点
InsertClass	插入模型
RemoveClass	移除模型

2.8.3.1 公共函数

2.8.3.1.1 SetClassifyName

设置顶层分类节点

A. 语法

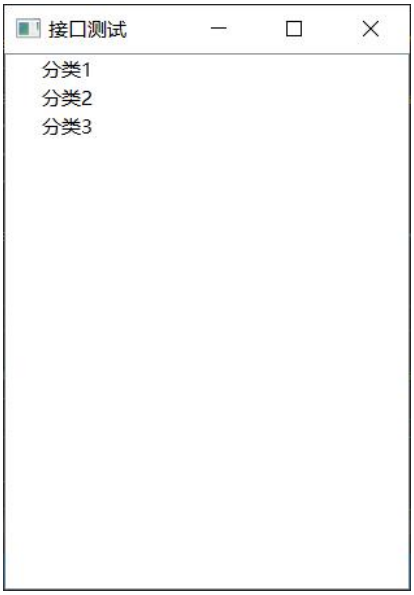
```
/**
 * @brief  设置分类
 * @param[in] name_list 分类名称列表
 */
void SetClassifyName(const QstringList &name_list)
```

B. 说明

设置顶层分类节点，为模型树设置分类，默认分类是“模型库”和“用户模型”。

C. 示例

```
QMainWindow main_win;

MwClassManager *class_manager = new MwClassManager;
class_manager->Initialize();
MwMoClassTreeModel *tree_model = new MwMoClassTreeModel(class_manager);
QTreeView *tree_view = new QTreeView;
tree_view->setModel(tree_model);

main_win.setCentralWidget(tree_view);
main_win.showMaximized();
tree_model->SetClassifyName(QStringList() << QStringLiteral(" 分 类 1") <<
QStringLiteral("分类 2") << QStringLiteral("分类 3"));
```



2.8.3.1.2 AppendTopClass

添加顶层模型

A. 语法

```
/**
```

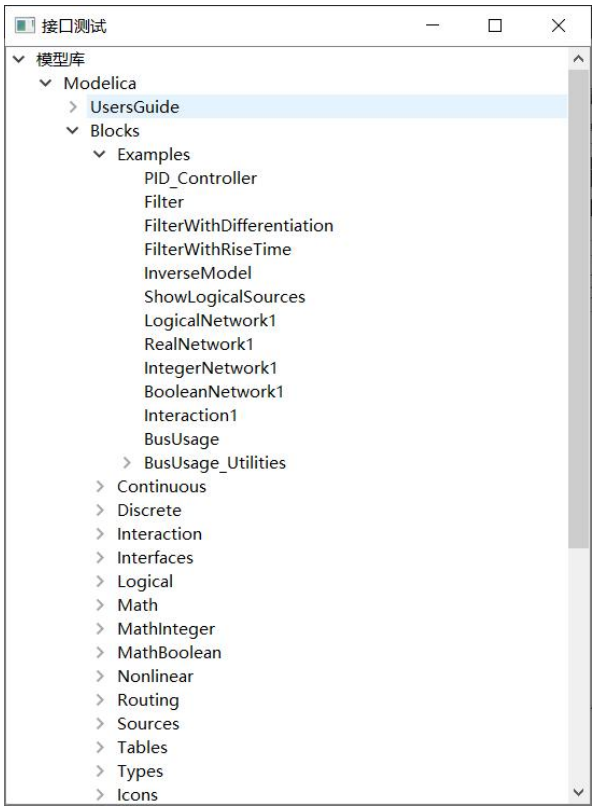
```
 * @brief  添加顶层模型
 * @param [in]key 模型key
 * @param [in]classify_name 分类名
 */
void AppendTopClass(MWint key, const Qstring &classify_name)
```

B. 说明

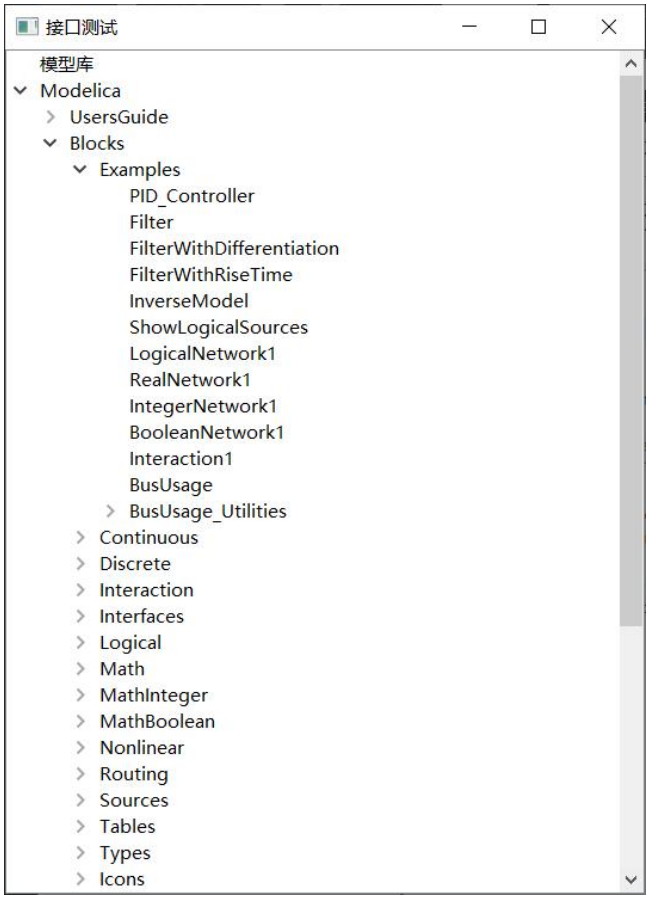
添加顶层模型

C. 示例

```
QMainWindow main_win;

MwClassManager *class_manager = new MwClassManager;
class_manager->Initialize();
class_manager->GetMoHandler()->LoadMoLibrary("Modelica", "3.2.1");
MWint lib_key = class_manager->GetMoHandler()->GetKeyByTypeName("Modelica");

MwMoClassTreeModel *tree_model = new MwMoClassTreeModel(class_manager);
QTreeView *tree_view = new QTreeView;
tree_view->setModel(tree_model);
tree_model->SetClassifyName(QStringList() << QStringLiteral("模型库"));

main_win.setCentralWidget(tree_view);
main_win.showMaximized();
tree_model->AppendTopClass(lib_key, QStringLiteral("模型库"));
```




```
tree_model->AppendTopClass(lib_key, QStringLiteral("用户模型"));
```



2.8.3.1.3 GetTopItems

获取所有顶层节点

A. 语法

```
Qlist<QStandardItem*> GetTopItems()
```

B. 说明

获取所有顶层节点

C. 示例

```
QList<QStandardItem*> lst_item = tree_model->GetTopItems();
QStringList lst_item_name;
for (auto item : lst_item)
{
    qDebug() << item->data(NameRole).value<QString>();
}
```

```
"模型库"
"用户模型"
"Modelica"
```

2.8.3.1.4 InsertClass

在指定节点下插入模型

A. 语法

```
/**
 * @brief  插入模型
 * @param [in]key 模型key
 * @param [in]parent_item 父节点
 */
void InsertClass(MWint key, QstandardItem *parent_item)
```

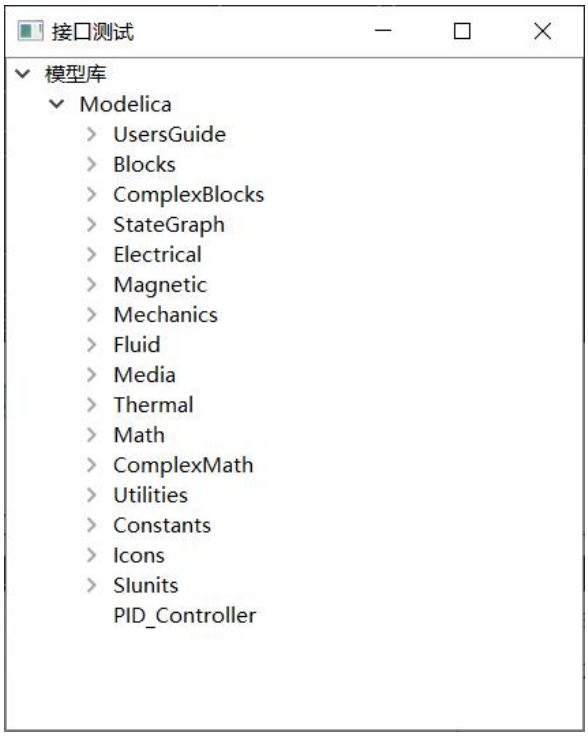
B. 说明

在指定节点下插入模型，插入节点时，模型树会自动刷新。

C. 示例

调用 MwMoClassTreeModel 的 InsertClass 接口，参数 1 设置为 PID_Controller 模型键值，参数 2 设置为 Modelica 模型树顶层节点。

```
tree_model->InsertClass(mo_key, lib_item);
```



2.8.3.1.5 RemoveClass

移除模型

A. 语法

```
void RemoveClass(MWint key)
```

B. 说明

移除模型，模型视图中心和模型浏览树会自动进行移除刷新。

C. 示例

参数模型。

```
tree_model->RemoveClass(lib_key);
```

2.8.4 MwModelParameterTabWidget

模型参数面板类，能够显示选中模型或组件的参数，并且支持对各种类型的参数进行编辑，能够与中央视图进行联动。

函数名	简介
GetParamEditMode	获取参数编辑模式

函数名	简介
SlotUpdate	更新面板

2.8.4.1 公共类型

```
enum MwParamEditMode
{
PEM_Panel,
PEM_Dialog
}
```

功能	参数编辑模式	
参数	PEM_Panel PEM_Dialog	面板模式 对话框模式

2.8.4.2 公共函数

2.8.4.2.1 GetParamEditMode

获取参数编辑模式

A. 语法

```
MwParamEditMode GetParamEditMode()
```

B. 说明

获取参数编辑模式

C. 示例

```
QMainWindow main_win;

MwClassManager *class_manager = new MwClassManager;
class_manager->Initialize();
MwMoGraphicsViewController           *mo_controller           =           new
MwMoGraphicsViewController(class_manager);
MwModelParameterTabWidget             *param_wgt             =             new
```

```
MwModelParameterTabWidget(mo_controller, PEM_Dialog);

main_win.setCentralWidget(param_wgt);
main_win.showMaximized();
MwParamEditMode edit_mode = param_wgt->GetParamEditMode();
```

2.8.4.3 公共槽函数

2.8.4.3.1 SlotUpdate

响应模型视图模块更新信号，刷新参数面板

A. 语法

```
void SlotUpdate(mogv::MoGvEvent mo_event, Qlist<MWint> ckass_key_list)
```

B. 说明

响应模型视图模块更新信号，刷新参数面板

2.8.5 MwSimPlotWindow

仿真曲线视图类，用于显示仿真变量曲线。

函数名	简介
GetPlotWinIndex	获取曲线窗口唯一 ID
AddCurveToCurrentView	添加变量到曲线图
SigWindowClosed	窗口关闭信号

2.8.5.1 公共函数

2.8.5.1.1 GetPlotWinIndex

获取曲线窗口编号

A. 语法

```
size_t GetPlotWinIndex()
```

B. 说明

获取曲线窗口编号，每个曲线窗口存在唯一 ID，通过获取 ID 号可定位到曲线窗口。

C. 示例

```
MwClassManager *class_manager = new MwClassManager;
class_manager->Initialize();

MwSimPlotWindow *plot_win = new MwSimPlotWindow(2, true, class_manager);
size_t win_index = plot_win->GetPlotWinIndex();
```

win_index	2	unsigned __int64

2.8.5.1.2 AddCurveToCurrentView

添加变量曲线到当前视图中

A. 语法*

```
/**
* @brief  增加变量曲线到当前视图
* @param [in]curve_name 曲线名称
* @param [in]data 曲线数据
* @return 返回曲线
*/
MwPlotCurve* AddCurveToCurrentView(
const QString &curve_name,
MwAbstractData *data)
```

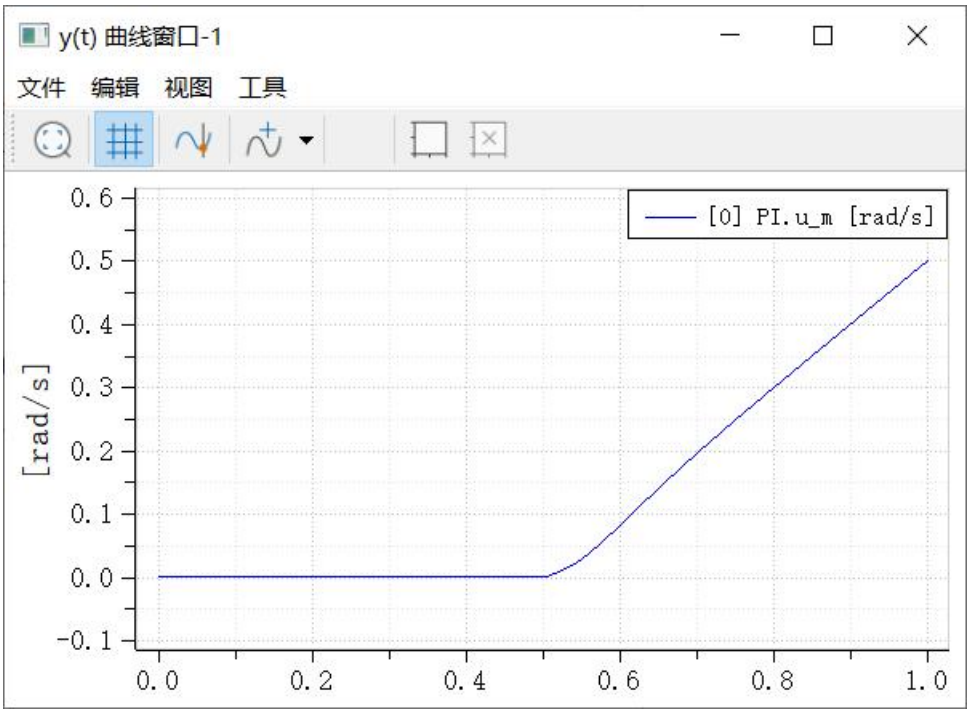
B. 说明

添加变量曲线到当前视图中，通过传入变量的名称和当前结果数据，即可创建变量曲线。

C. 示例

调用 MwSimPlotWindow 的 AddCurveToCurrentView 接口，参数 1 设置为 PI.u_m，参数 2 设置为(MwAbstractData*)sim_data

```
plot_win->AddCurveToCurrentView("PI.u_m", (MwAbstractData*)sim_data);
```



2.8.5.2 信号

2.8.5.2.1 SigWindowClosed

窗口关闭信号

A. 语法

```
void SigWindowClosed(MwSimPlotWindow *plot_win)
```

B. 说明

窗口关闭信号

2.8.6 MwSimConfigWidget

模型仿真设置控件，用于显示和修改模型仿真设置。

函数名	简介
GetSimConfig	获取仿真设置

模块描述：模型仿真设置控件，用于显示和修改模型仿真设置。

模块使用：

```
MwExperimentData exp_data;
MwSimConfigWidget *sim_config_wgt = new MwSimConfigWidget(exp_data, &main_win);
```

2.8.6.1 公共类型

```
struct MwExperimentData
{
   MWdouble startTime;
   MWdouble stopTime;
   MWdouble intervalLength;
   MWcint numberOfIntervals;
   std::string algorithm;
   MWdouble tolerance;
   MWdouble fixedOrInitStepSize;
   std::vector<std::pair<MWdouble, MWdouble>> pieceWiseStep;
}
```

功能	仿真配置	
参数	startTime	仿真开始时间
	stopTime	仿真终止时间
	intervalLength	仿真输出步长
	numberOfIntervals	仿真输出步数
	algorithm	积分算法
	tolerance	精度
	fixedOrInitStepSize	初始积分步长
	pieceWiseStep	分段固定积分步长数组

2.8.6.2 公共函数

2.8.6.2.1 GetSimConfig

获取界面上的仿真设置

A. 语法

```
/**
 * @brief  获取仿真设置
 */
```

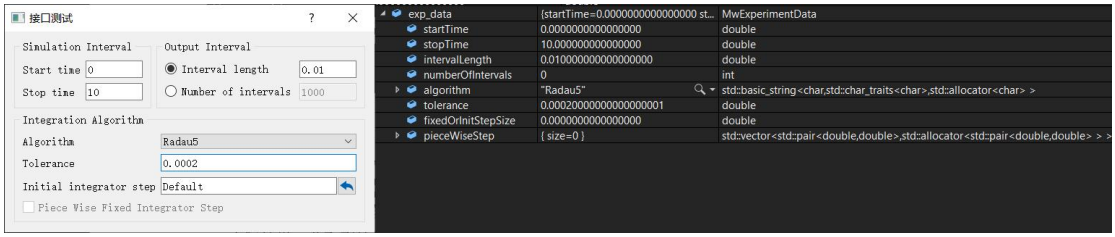
```
void GetSimConfig(MwExperimentData *sim_config)
```

B. 说明

获取界面上得仿真设置

C. 示例

```
MwSimConfigWidget *sim_config_wgt = new MwSimConfigWidget(new
MwExperimentData);
QDialog dlg;
QVBoxLayout *layout = new QVBoxLayout;
layout->addWidget(sim_config_wgt);
dlg.setLayout(layout);
sim_config_wgt->GetSimConfig(&exp_data);
```



2.9 系统配置

软件或模型进行相关系统环境设置，如可设置系统工作的路径、设置编译平台的信息。

函数名	简介
SetWorkPath	设置软件工作路径
WorkPath	获取软件工作路径
SetSimResultPath	设置仿真结果路径
SimResultPath	获取仿真结果路径
SetCompileInfo	设置编译器信息
GetCompileInfo	获取编译器信息
SwitchSolverPlatform	切换求解器平台

2.9.1 SetWorkPath

设置工作路径

A. 语法

```
/**
 * @brief 设置工作目录
 * @param [in]工作路径
 */
void SetWorkPath(const std::wstring &work_path);
```

B. 说明

设置工作路径，软件工作路径

C. 示例

```
std::wstring path = classMgr->GetCoreOption->WorkPath();
classMgr->GetCoreOption->SetWorkPath(new_path);
std::wstring res_path = classMgr->GetCoreOption->WorkPath();
```

2.9.2 WorkPath

获取工作路径

A. 语法

```
/**
 * @brief 获取工作目录
 */
std::wstring WorkPath();
```

B. 说明

获取工作路径，运行软件的路径

2.9.3 SetSimResultPath

设置仿真结果路径

A. 语法

```
/**
 * @brief 设置仿真结果目录
 * @param [in]仿真目录路径
 */
void SetSimResultPath(const std::wstring &simulation_path);
```

B. 说明

设置仿真结果路径，用于仿真时存储仿真相关结果。

C. 示例

```
std::wstring path = classMgr->GetCoreOption->SimResultPath();
classMgr->GetCoreOption->SetSimResultPath(new_path);
std::wstring res_path = classMgr->GetCoreOption->SimResultPath();
```

2.9.4 SimResultPath

获取仿真结果路径

A. 语法

```
/**
 * @brief 获取仿真结果目录
 */
std::wstring SimResultPath();
```

B. 说明

获取仿真结果路径

2.9.5 SetCompileInfo

编译器设置

A. 语法

```
/**
 * @brief 编译器设置
 * @para [in/out]cmpl_type 编译器类型
 * @para [in/out]cmpl_name 编译器名字
 * @para [in/out]cmpl_path_x86 32位编译器路径
 * @para [in/out]cmpl_path_x64 64位编译器路径
```

```
 * @para [in/out]platform  编译时的平台
 */
void SetCompileInfo(
const std::string &cmpl_type,
const std::wstring &cmpl_name,
const std::wstring &cmpl_path_x86,
const std::wstring &cmpl_path_x64,
MwCPUArch platform);
```

B. 说明

编译器设置，用户可设置本机上的编译器路径和位数，设置后默认使用该编译器进行编译。

C. 示例

```
classMgr->GetCoreOption->SetCompileInfo("vc",L"Microsoft Visual Studio 2017",L"D:/VS2017/VC",L"D:/VS2017/VC",MwCoreOption::x86_64);
```

2.9.6 GetCompileInfo

获取编译器设置

A. 语法

```
/**
 * @brief 编译器设置
 * @para [in/out]cmpl_type 编译器类型
 * @para [in/out]cmpl_name 编译器名字
 * @para [in/out]cmpl_path_x86 32位编译器路径
 * @para [in/out]cmpl_path_x64 64位编译器路径
 * @para [in/out]platform  编译时的平台
 */
void GetCompileInfo(
std::string *cmpl_type,
std::wstring *cmpl_name,
std::wstring *cmpl_path_x86,
std::wstring *cmpl_path_x64,
MwCPUArch  *platform);
```

B. 说明

获取编译器设置

C. 示例

```
classMgr->GetCoreOption->SetCompileInfo(cmpl_type,cmpl_name,cmpl_path_x86,cm
pl_path_x64,platform);
```

type = vc,

name = Microsoft Visual Studio 2017

path_x86 = D:/VS2017/VC

path_x64 = D:/VS2017/VC

platform = MwCoreOption::x86_64

2.9.7 SwitchSolverPlatform

切换求解器平台

A. 语法

```
/**
 * @brief 切换求解器平台
 * @param[in] 求解平台
 */
bool SwitchSolverPlatform(MwCPUArch platform);
```

B. 说明

切换求解器平台位数，可根据需求将求解器切换为64位和32位下进行仿真。

C. 示例

```
classMgr->GetCoreOption->SwitchSolverPlatform(MwCoreOption::x86_64);
```

3. 名词解释

为了帮助用户更好的使用接口，并理解接口中出现的相关名词，本章节用于解释接口描述或者说明中出现的名词。

3.1 模型结构解释

以 Modelica 标准库中的“PID_Controller”模型来对模型的 key、模型简名、模型全名、模型路径、模型描述等名词进行解释。

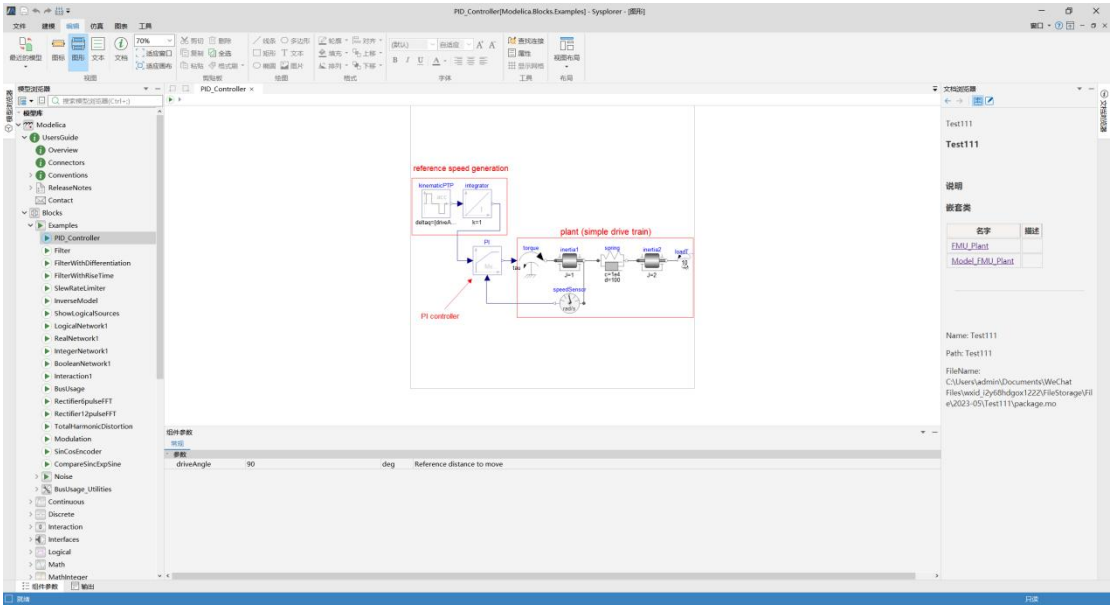


图 3-1 示例模型

- 模型 key：每个模型都会对应一个 MWint 类型的键值(简称 key)，用于标识模型，作为模型的 ID；
  - 软件每次启动后若对模型进行了卸载重新打开、模型编辑、模型修改等操作都会导致 key 发生改变，可用 key 来对模型进行定位、获取属性等相关操作；软件重新启动后模型 key 与上次软件打开的模型 key 不一致。
  - PID_Controller 本模型有一个对应的唯一模型 key，也称主模型 key；
  - 模型 key 可通过 GetKeyByTypeName 接口对传入模型全名进行查询。
- 组件 key：模型下会存在组件，每个组件名称都存在一个唯一 key；
  - 例如，PID_Controller 模型下有个名称为“PI”的组件，该组件会存在一个组件 key。
  - 组件 key 可通过传入组件全名 GetKeyByTypeName 接口进行查询
- 组件类型 key：组件是一个模型的实例，而模型是一个组件的类型，所以每个组件会对应一个类型，该类型也会存在一个 key。
  - 例如名称为“PI”的组件其类型为“Modelica.Blocks.Continuous.LimPID”该类型会对应一个 key，该 key 为 PI 组件的类型 key。

- 组件类型 key 可通过传入组件对应的类型全名 GetKeyByTypeName 接口进行查询
- 通过组件 key 查询到的模型相关属性为该组件所在模型 key 下的属性，通过类型 key 查询到的相关属性为该类型对应的模型的相关属性。

- 模型描述：对模型使用的描述；
  - 模型描述可通过 GetPropAsString 接口并传入模型 key 进行查询。
- 组件描述：模型下组件对应的描述；
  - 组件描述可通过 GetPropAsString 接口并传入组件 key 进行查询
- 模型全名：由包含模型的所有外层类（依次由外至内）及模型本身名称组合而成，各名称之间用“.”分割。
  - 例如本模型中模型全名为：Modelica.Blocks.Examples.PID_Controller；
  - 模型全名可通过 GetFullnameProp 接口来查询。
- 组件全名：模型下的组件的全名，
  - 例如组件 PI 的全名为 Modelica.Blocks.Examples.PID_Controller .PI；
  - 组件全名可通过 GetFullnameProp 接口来查询
- 组件简明(组件名称)：模型下的组件名称。
  - 例如 PI；
  - 组件名称可通过 GetNameProp 接口来查询。
- 模型简名(模型名称)：只有模型名称，称为简名。
  - 例如 PID_Controller ；
  - 模型名称可通过 GetNameProp 接口来查询。
- 组件列表 key：模型下有多个组件，每个组件都有一个唯一 key，所有的 key 形成一个列表称为组件 key，通过组件列表 key 可循环操作组件并获取相关属性。
  - 组件列表 key 可通过 LookupCompAndTypeEx 来查询；
- 模型路径：保存模型的文件；
  - 模型路径可通过 GetFullFileName 接口来查询；

Sysplorer 界面中查看以上解释：

在“PID_Controller”模型中点击空白处，右键点击“属性”，弹出下列属性框：

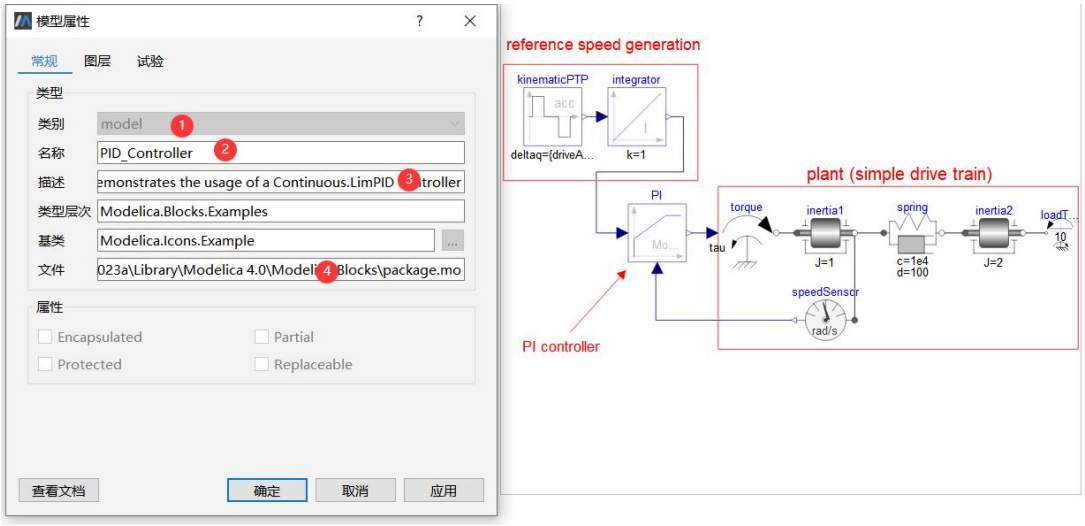


图 3-2 模型属性

(1) 模型类别

(2) 模型简明

(3) 模型描述

(4) 模型路径

选择对应的组件，右键点击“属性”，弹出下列属性框：

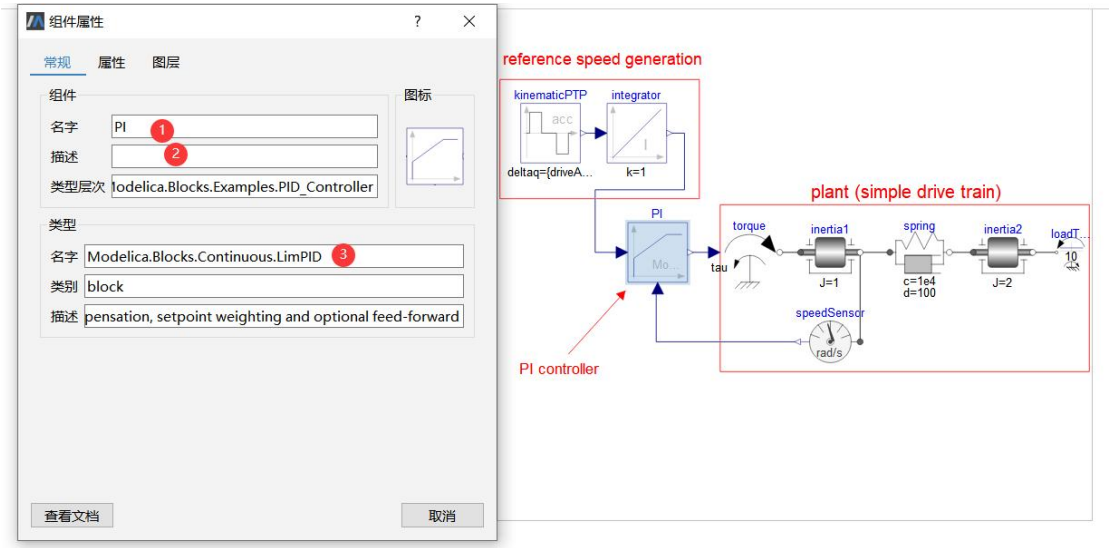


图 3-3 组件属性

(1) 组件简名；该组件会有一个组件 key；

(2) 组件描述；

(3) 组件类型全名；该模型会对应一个组件类型 key；

点击“组件浏览器”窗口，可查看到该模型下的所有组件，这些组件会有一个组件列表 key：

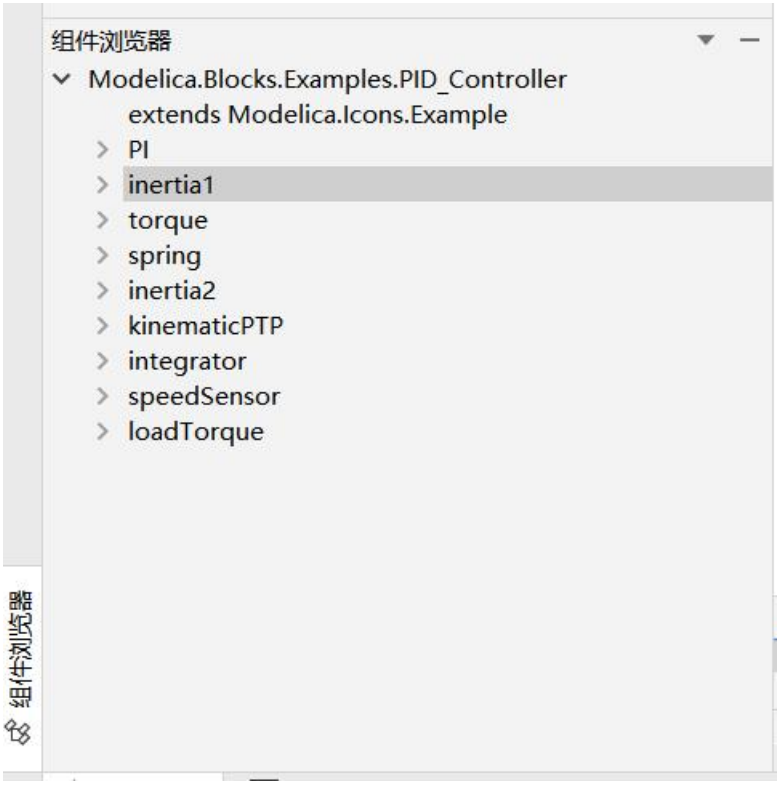

组件浏览器
Modelica.Blocks.Examples.PID_Controller
extends Modelica.Icons.Example
PI
inertia1
torque
spring
inertia2
kinematicPTP
integrator
speedSensor
loadTorque
组件浏览器

图 3-4 组件列表