# Anforderungen für den Wrapper Generator

Der Wrapper Generator für die XML-States generiert für jeden XML State eine Basisklasse, von der der XML State erbt. Diese Basisklasse stellt Zugriffsfunktionen auf die Input- Local- und Output Parameter des States zur Verfügung. Die generierten Dateien sollen im jeweiligen ArmarX-Projekt unter [project]/build/[pathOfState]/[statetname].generated.h abgelegt werden.

Der Wrapper Generator wird an zwei unterschiedlichen Stellen in ArmarX aufgerufen: Im Editor und während des Build-Vorgangs.

## Generierung zur Editor-Laufzeit

Wenn eine Statechart Group im Editor gespeichert wird, oder die Generierung der Basisklassen explizit durch den Benutzer angefordert wird, müssen die Basisklassen neu generiert werden. Dies ist notwendig, damit der Statechart Editor parallel zur Statechart Implementierung in QTCreator genutzt werden kann. Somit werden die Zugriffsfunktionen von QTCreator in der Autovervollständigung angezeigt.

## Generierung zur Compile Zeit

Da die generierten Basisklassen nicht eingecheckt werden sollen, müssen diese beim Bauen des Projekts immer automatisch erzeugt werden. Hierzu ruft CMake das speziell dafür erstellte Geneator-Executable mit entsprechenden Parametern auf. Die Benötigten Parameter richten sich nach den benötigten Informationen für die Generierung, siehe unten.

## Notwendige Informationen für den Generator

Der Generator muss für jedes verwendete Variant 3 Informationen bestimmen. Diese sind an unterschiedlichen Stellen im generierten Code notwendig:

|  |  |
| --- | --- |
| Information | Notwendig für |
| Daten Typ Name des Variants | 1. Template Parameter von “getInput“, “getLocal”, “getOutput” Funktionen der State Klasse. 2. Parameter Typ der generierten set-Funktionen |
| Rückgabe Typ Name des Variants | Rückgabe Typ der generierten get Funktionen. |
| Factory Include für das Variant | Muss als #include angegeben werden. |

Diese Informationen müssen anhand des Base-Typs des Variants ermittelt werden. Der Base-Type ist der abgespeicherte Datentyp für Inputs/Outputs in der State-Definition. Der Base-Type entspricht dem String, mit dem das Variant registriert wurde. [„Variant::addTypeName(…)“]

### Beispiel std::string

|  |  |
| --- | --- |
| Base-Typ | ::armarx::StringVariantData |
| Daten Typ Name | std::string |
| Rückgabe Typ Name | std::string |
| Factory Include | #include <Core/observers/ObserverObjectFactories.h> |

### Beispiel MatrixFloat

|  |  |
| --- | --- |
| Base-Typ | ::armarx::MatrixFloatBase |
| Daten Typ Name | ::armarx::MatrixFloat |
| Rückgabe Typ Name | ::armarx::MatrixFloatPtr |
| Factory Include | #include <Core/util/variants/eigen3/Eigen3VariantObjectFactories.h> |

## Speicherort der Variant Informationen

In jedem ArmarX Projekt wird einen VariantInfo.xml abgelegt. Diese Datei befindet sich immer an derselben Stelle innerhalb eines Projekts, damit sie leicht gefunden werden kann. Die Datei ist wie folgt aufgebaut:

<?xml version="1.0" encoding="utf-8"?>

<VariantInfo>

<Lib name="[LibName]">

<VariantFactory include="[IncludePath]" />

<VariantFactory … />

<Variant baseType="[Base-Typ]" dataType="[Daten-Typ]" humanName="[Anzeige im Editor]" basic="[true oder Attribut weglassen]" />

<Variant … />

</Lib>

</VariantInfo>

## Weitere Abhängigkeiten

* Die VariantInfo.xml Dateien müssen sowohl vom Editor als auch vom Code Generator gefunden und ausgelesen werden.
* Der Editor muss den Code Generator aufrufen. Entweder
  + Direkt, so dass die Dateien im richtigen build Verzeichnis erstellt werden
  + Indirekt, indem der Editor CMake im entsprechenden build Verzeichnis ausführt und CMake wiederum dann die Code Generierung auslöst

## Allgemeine Bemerkungen zum Editor

* Der Editor muss beim Laden einer Statechart Group prüfen, ob alle notwendigen VariantInfo.xml Dateien gefunden wurden und ob alle notwendigen Variant Libraries geladen wurden.
* In der Auswahlliste der Variants sollte angezeigt werden, zu welcher Library ein Variant gehört. Eventuell kann man die Variants nach Libraries gruppiert werden.
* Aus der Auswahl der Variants ergeben sich Abhängigkeiten der Statechart Group gegen die entsprechenden Variant Libraries. Diese Abhängigkeiten müssen bislang in die CMakeLists.txt der Statechart Group eingetragen werden. Eventuell ist es möglich diese Abhängigkeiten automatisch in die CMakeLists.txt einzutragen.

# Anhang: Beispiel für generierte Basisklasse

#ifndef ARMARX\_MYGROUP\_MYSTATEWRAPPER\_H

#define ARMARX\_MYGROUP\_MYSTATEWRAPPER\_H

#include <Core/statechart/xmlstates/XMLState.h>

#include <Core/observers/ObserverObjectFactories.h>

#include <Core/util/variants/eigen3/Eigen3VariantObjectFactories.h>

namespace armarx

{

namespace MyGroup

{

class MyStateWrapper

: public XMLStateTemplate < MyStateWrapper >,

public XMLStateFactoryBase

{

protected:

class MyStateIn

{

private:

MyStateWrapper \*parent;

public:

MyStateIn(MyStateWrapper \*parent)

: parent(parent)

{

}

public:

std::string getInput1()

{

return parent->getInput< std::string>("Input1");

}

::armarx::MatrixFloatPtr getInput2()

{

return parent->getInput< ::armarx::MatrixFloat>("Input2");

}

}; // class MyStateIn

class MyStateLocal

{

private:

MyStateWrapper \*parent;

public:

MyStateLocal(MyStateWrapper \*parent)

: parent(parent)

{

}

public:

std::string getLocal1()

{

return parent->getLocal< std::string>("Local1");

}

void setLocal1(const std::string & value)

{

parent->setLocal("Local1", value);

}

std::string getLocal2()

{

return parent->getLocal< std::string>("Local2");

}

void setLocal2(const std::string & value)

{

parent->setLocal("Local2", value);

}

}; // class MyStateLocal

class MyStateOut

{

private:

MyStateWrapper \*parent;

public:

MyStateOut(MyStateWrapper \*parent)

: parent(parent)

{

}

public:

std::string getOutput1()

{

return parent->getOutput< std::string>("Output1");

}

void setOutput1(const std::string & value)

{

parent->setOutput("Output1", value);

}

std::string getOutput2()

{

return parent->getOutput< std::string>("Output2");

}

void setOutput2(const std::string & value)

{

parent->setOutput("Output2", value);

}

}; // class MyStateOut

protected:

MyStateIn in;

MyStateLocal local;

MyStateOut out;

public:

MyStateWrapper(XMLStateConstructorParams stateData)

: XMLStateTemplate < MyStateWrapper > (stateData),

in(MyStateIn(this)),

local(MyStateLocal(this)),

out(MyStateOut(this))

{

}

public:

static std::string GetName()

{

return "MyState";

}

}; // class MyStateWrapper

} // namespace MyGroup

} // namespace armarx

#endif // ARMARX\_MYGROUP\_MYSTATEWRAPPER\_H