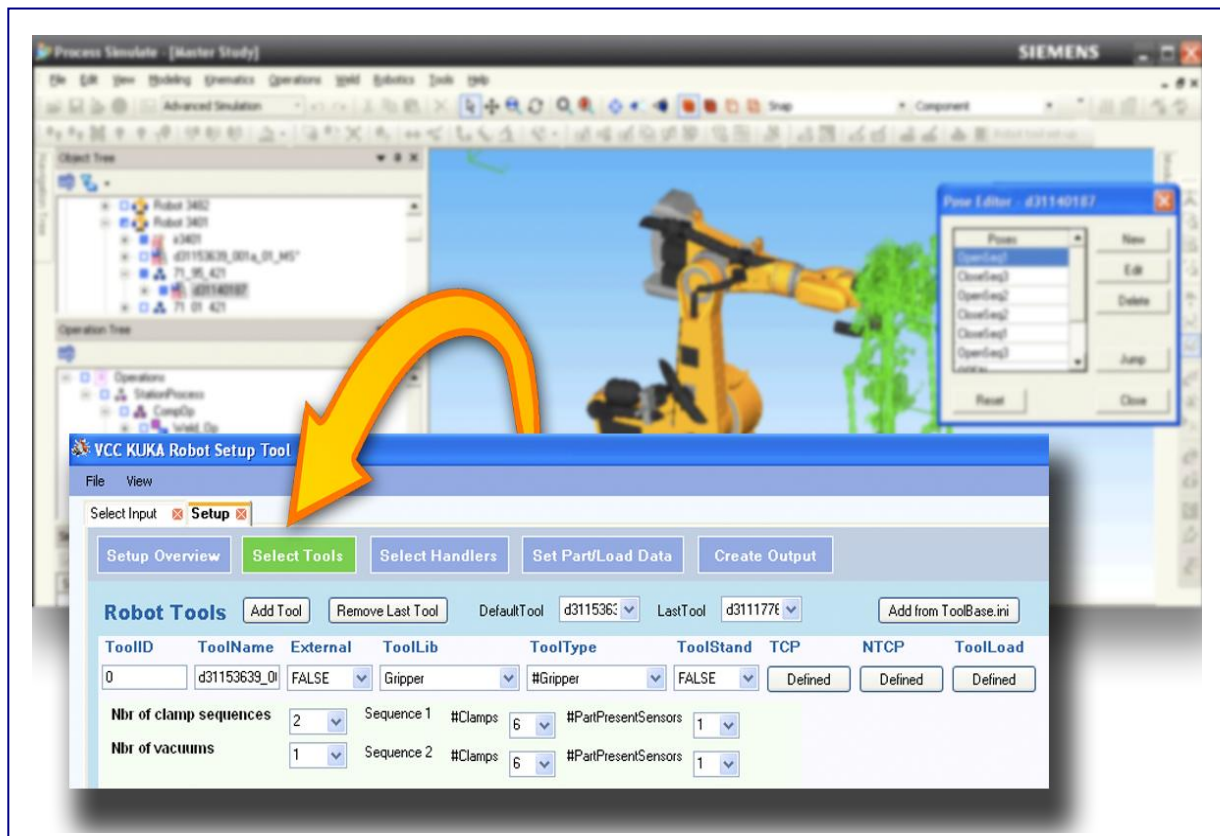


# CHALMERS



## Automatic Generation of Control Code for Robot Function Packages

Design of software for robot setup description generation

*Master of Science Thesis*

DANIEL WAHLBERG

YIXIAN ZHANG

Department of Signals and Systems

*Division of Automatic Control, Automation and Mechatronics*

CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden, 2010

Report No. EX093/2010



# Automatic Generation of Control Code for Robot Function

Design of software for robot setup description generation

DANIEL WAHLBERG, Systems, Control and Mchatronics

YIXIAN ZHANG, Software Engineering and Technology

Examiner and Chalmers Supervisor:

Petter Falkman, petter.falkman@chalmers.se

Automation Research Group

Chalmers University of Technology

41296 Göteborg, Sweden

Volvo Car Corporation Supervisor:

Tord Nordin, tnordin@volvocars.se

Facilities, Tooling and Equipment

Volvo Car Corporation

Dept. 81331 FTE, TAÖ40

40531, Göteborg, Sweden

Department of Signals and Systems

*Division of Automatic Control, Automation and Mechatronics*

CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2010

Automatic Generation of Control Code for Robot Function Packages  
Design of software for robot setup description generation

Master's Thesis in the Master's programme in Systems, Control and  
Mechatronics & Software Engineering and Technology

DANIEL WAHLBERG, [wahlberd@student.chalmers.se](mailto:wahlberd@student.chalmers.se)

YIXIAN ZHANG, [yixian@student.chalmers.se](mailto:yixian@student.chalmers.se)

© DANIEL WAHLBERG & YIXIAN ZHANG, 2010

Report No. EX093/2010

Department of Signals and Systems

Division of Automatic Control, Automation and Mechatronics

CHALMERS UNIVERSITY OF TECHNOLOGY

SE-412 96 Göteborg

Sweden

Telephone: + 46 (0)31-772 1000

Automatic Generation of Control Code for Robot Function Packages  
Design of software for robot setup description generation

DANIEL WAHLBERG

YIXIAN ZHANG

Department of Signals and Systems

Division of Automatic Control, Automation and Mechatronics

Chalmers University of Technology

Summary

Much effort has been put in developing efficient automation procedures for manufacturing industries. This has however been mostly focused on the final implementation steps. To further decrease ramp-up time and improve production commissioning there is a need to examine the possibilities of applying more automation strategies in the planning stage of plant adjustments. This thesis presents a method to transfer robot setup data from a simulation tool to a robot controller setup tool. The AutomationML specification has been implemented to provide the bridge between the two environments. A new setup tool has been developed during automatic generation of robot setup descriptions. These contain control code files that make up complete robot function packages that can be downloaded to a robot controller.

*Keywords: Automation, Software, Simulation, AutomationML, Setup tool, Robot control, virtual commissioning, Robot program development, Volvo Cars, Chalmers*

## **Acknowledgements**

In particular, we acknowledge Petter Falkman, who provided us with the project, recommended the right people to us, and patiently guided us through the process and provided encouragement, ensuring that the process kept on going during the whole project.

A special acknowledgement goes to our supervisor Tord Nordin at Volvo Cars, who has been helpful and supportive towards our work. He has provided us with required knowledge, materials and in-time feedback. As a continual project, Ruben Pabello, Sathyamyla KanthabhabhaJeya, and Fei Zhennan answered a lot of questions.

Of course, we have appreciated the help and patience from Stefan Axelsson, who solved quite a lot of small questions and issues that showed up every now and then in VCC. Special thanks also go out to Mathias Sundbäck and Fredrik Westman who helped us in the verification part.

# Contents

<b>1. Introduction</b>	<b>1</b>
<b>2. Goals</b>	<b>3</b>
2.1. Subgoals . . . . .	3
<b>3. Background</b>	<b>5</b>
3.1. Developing the robot commissioning process at Volvo Cars . . . . .	5
3.2. Current setup description generation . . . . .	7
3.2.1. The setup description . . . . .	7
3.2.2. Automatic setup description generation . . . . .	8
3.2.3. The current Setup program . . . . .	10
3.2.4. The Robot function library . . . . .	11
3.3. AutomationML . . . . .	13
3.3.1. The AutomationML structure . . . . .	14
<b>4. Method</b>	<b>19</b>
4.1. Rebuild of Setup program . . . . .	19
4.2. Choice of software development environment . . . . .	19
4.3. Developing a software model . . . . .	20
4.3.1. General model . . . . .	21
4.3.2. Robot function library connection . . . . .	22
4.3.3. Setup Data Model . . . . .	22
4.4. AutomationML Interface . . . . .	23
4.5. Setup description generation . . . . .	24
4.6. Adjustment of Process Simulate add-on . . . . .	24
4.7. Verification methods . . . . .	24
4.7.1. Verification of generated setup description . . . . .	25
4.7.2. Verification of data consistency . . . . .	25
4.7.3. Verification of add-on change . . . . .	26
<b>5. Results</b>	<b>27</b>
5.1. Software model . . . . .	27
5.1.1. Robot function library connection . . . . .	30

5.1.2.	Setup Data Model . . . . .	32
5.1.3.	GUI design . . . . .	32
5.1.4.	Setup description generation . . . . .	34
5.2.	Process Simulate connection . . . . .	34
5.3.	AutomationML . . . . .	34
5.3.1.	The AMLEngine Extension . . . . .	34
5.4.	Verification . . . . .	35
5.4.1.	Verification of generated setup description . . . . .	37
5.4.2.	Verification of data consistency . . . . .	40
5.4.3.	Verification of Process Simulate connection . . . . .	42
<b>6.</b>	<b>Conclusion</b>	<b>43</b>
	<b>References</b>	<b>45</b>
<b>A.</b>	<b>Setup Program Extension Guide</b>	<b>47</b>
<b>B.</b>	<b>Setup Description Generation Overview</b>	<b>49</b>



# 1. Introduction

Since IT technology has been widely used in every area of industry, off-line programming of industrial robots has largely been used by production companies. It provides an interface between simulation and the plant. This greatly reduces the need to stop production when making changes without rebuilding factory cells.

Volvo Car Corporation (VCC) has been using this procedure since early 1990s. However, when planning for new installations in the production line (commissioning) the same degree of automation is not established today. The simulation of a new robot station results in a specification for the robot configuration, covering for example Tool Center Points, mounted tools, external tools, tool changes and media (air, water). This specification is to large extent dealt with manually, and used as input to a VCC setup software which assembles needed function packages and control code from a library. The outputted result is downloaded to the robot controller and a description is passed to a line builder to make adjustments to the plant.

In industry, the virtual robotic environments are commonly used to develop, prototype and simulate control strategies and algorithms for single or multi-robot systems. The concept of the Digital Factory (Kuhn 2006) has been a topic for discussions on how to further develop these procedure. This is discussed in (Schleipen and Drath 2009) and also touched upon in (Ranky 2004).

The robots in the production line need to be reconfigured with new control software, apart from change of their production programs. The main idea of this project is to automate data exchange between the simulation tool and the robot controller during a station build-up. Today robot setup data is transferred through manually created documents between the production planning stages. Through the work here presented an automatic interface is instead established. This has been subject to two master theses works.

The previous thesis deals with the extraction of data from the simulation tool through development of an API and an add-on software (Kanthabhabhajeya and Pabello Rodriguez 2010).

This thesis covers the process of developing a software tool with a GUI for defining a setup description. As input this software can use the extracted simulation tool data. Its output is function packages and code files that can be downloaded to a robot controller.

For the data exchange between the tools the AutomationML data format has been examined and implemented. This format has been suggested for exchange of manufacturing industry engineering data in (Luder, Hundt and Keibel 2010), (Rossdeutscher, Zuern and Berger 2010) and (Schleipen and Drath 2009). This work provides a real implementation of some of these concepts.

## 2. Goals

Our main goal is a connection of automatic data exchange between simulation system and robot controller.

It is desired to establish a less manual link between simulation and setup generation for new robot cells, since it would reduce the lead time in the commissioning phase. This can be achieved by introducing data exchange between the production simulation tool and the setup generation tool. Two essential questions in this process are:

1. How to extract the needed information from the simulation software?
2. How to transform this information into a corresponding setup?

The first question has been subjected to a thesis work carried out by R. Pabello and S. KanthabhabhaJeya. The second question is the starting-point for this thesis. It is also requested to examine the possibility of using Automation Markup Language (AutomationML, or AML) as the data exchange format to bridge the gap between the simulation and the setup software. AutomationML is a new standard for describing automation processes, incorporating the XML-structures of CAEX, PLC Open and Collada.

This project involves developing a new software program used for defining a robot controller setup. It will be used to automatically generate control code for the robot controller. There is currently a similar tool in use at VCC, but it does not provide all the necessary functionality for todays requirements.

### 2.1. Subgoals

A number of subgoals are also defined for this new Setup program. It should:

- maintain the functionality of the old Setup program
- handle manual and automatic input
- be flexible to future changes in input
- be well documented



## 3. Background

This chapter covers some key concepts related to this report. Section 3.1 gives an overview of the current robot commissioning process at VCC. In section 3.2 the robot controller setup description is presented. Section 3.3 provides an introduction to the AutomationML standard (Alonso-Garcia and Drath 2010).

### 3.1. Developing the robot commissioning process at Volvo Cars

The schema in figure 3.1 provides a general view of the connection between simulation and robot controller implementation at VCC today.

The simulation model is developed according to provided requirements. When a working simulation model has been defined the specifications for each robot station is denoted manually into a document. This document is passed by the engineer at the simulation department to a colleague at the manufacturing department. A Robot *Setup program* is then used for input of the specification. The Setup program assembles a *setup description* (see section 3.2.1), which is downloaded to a robot controller. This project limits to setup description generation for KUKA robot controllers.

The Setup program interacts with user input and a *Robot function library* (see section 3.2.4) to fetch and adjust files within the library according to setup. The files chosen from the library are only the ones needed for a particular setup.

This schema should be extended by introducing an automatic data exchange link between the simulation tool and the Setup program (see figure 3.2). The simulation tool used is Tecnomatix Process Simulate, provided by Siemens. It allows for simulation data to be extracted through the addition of a customized plug-in program, a so called add-on.

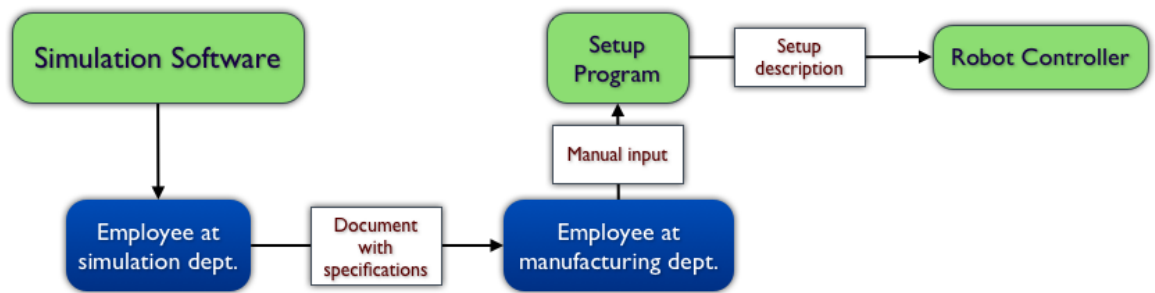


Figure 3.1.: Current workflow from simulation to robot controller at VCC

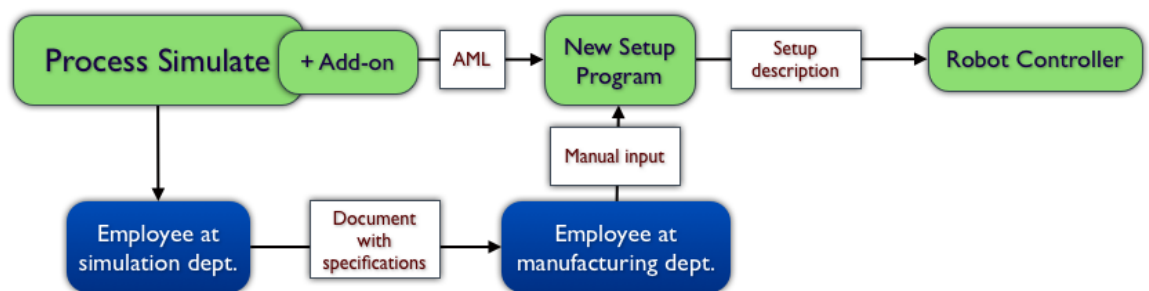


Figure 3.2.: New workflow from simulation to robot controller

The add-on is an application called Robot tool set-up generation. This was developed in the first master thesis work associated with this project (Kanthabhabhajeya and Pabello Rodriguez 2010). By including the add-on in the environment another button is included. This is enabled when a robot object is chosen, and the add-on GUI is activated when this is clicked, as shown below in figure 3.3. The add-on will, when Continue is clicked, generate an AutomationML document (see section 3.3) containing extracted robot setup data in a structured way.

Another interface for the AutomationML document will be included in the new Setup program. This will parse the data from the document to be used by the program. Thus the manual paper interface can be removed, providing both a faster and more secure data transfer. Ideally this automatic data exchange would include all necessary setup data obtainable from the simulation environment. For the scope of this project this has been restricted to contain gripper tool configuration data. This means that the manual link will still be used for other setup data.

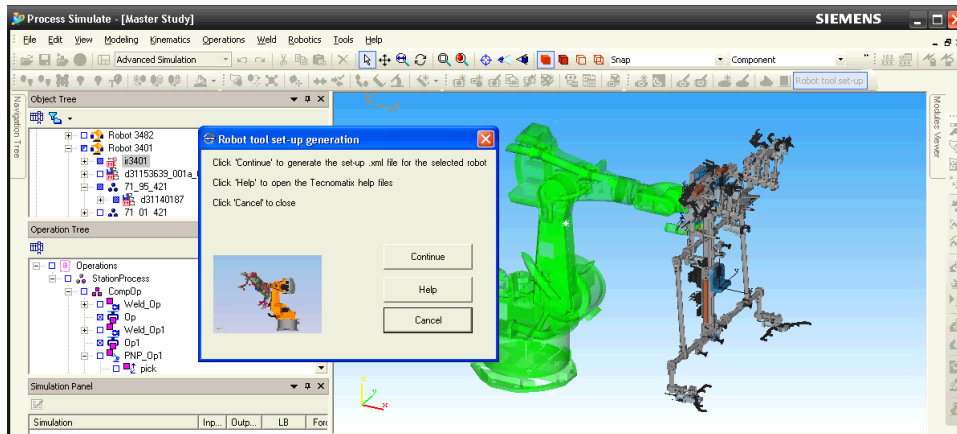


Figure 3.3.: The Process Simulate add-on

## 3.2. Current setup description generation

This section gives an overview of the setup description and how it is generated by the Setup program. The Robot function library is also presented together with an explanation of how its content is used as a base in the Setup description generation.

### 3.2.1. The setup description

The setup description is a collection of setup files that can be downloaded to a robot controller. It serves the controller with necessary parameters and functions. This includes:

- Definitions of available tools
- Definitions of available loads
- Definitions of available parts
- Configuration of IO-ports
- General handler function libraries included in every setup description
- Handler function libraries for the specified tool types
- Handler function libraries for communication interfaces
- Handler function libraries for media control, such as air or water supervision
- Icons, menus and options to the teach pendant user interface

It needs to be pointed out that the setup description does not contain the program executed by the robot controller when used in production. Rather it provides the necessary data and functionality for such a program to be executed as intended when downloaded to the controller.

### 3.2.2. Automatic setup description generation

Although a robot setup is particular for a certain robot, different setup descriptions have many components in common. Some are used in all setup descriptions, other are specific to for example certain tools. With this in mind VCC have created file libraries and template files arranged in logical directory structures. A Setup program has then been developed to help with the generation of setup descriptions. Different data sources are used in this generation:

- The *Default setup folder* contains files used as template files in the setup description generation.
- The Robot function library contain control code and initialization files, both general and setup specific.
- The file and user input to the Setup program provide the necessary information on what files to include and what parameter values to change in the template files.

A schema of the setup description generation procedure is presented in figure 3.4. An input file containing an initial specification is imported to the Setup program. The available options in the Setup program adapts to the structure of the Robot function library. By input to different forms the user can make changes and additions to the specification. The updated specification is then processed to generate a setup description.

A file system view of the components is given in figure 3.5.



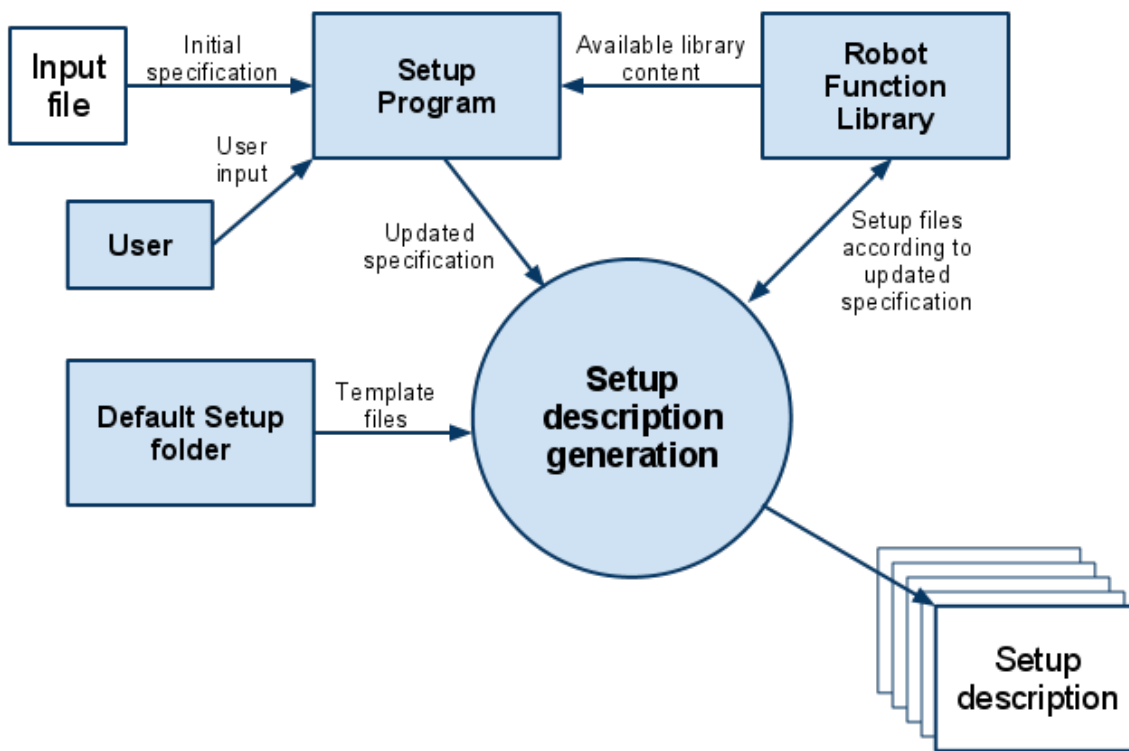


Figure 3.4.: Schema of the setup description generation

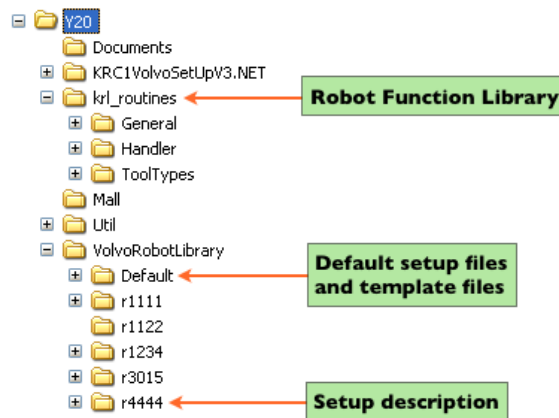


Figure 3.5.: Robot setup file system overview

### 3.2.3. The current Setup program

As input to the Setup program is a file called ToolBase.ini which provide specifications for:

- Tool data
- Part data
- Load data

This file is parsed within the Setup program and its data is presented in different forms. The Setup program is divided into a few steps where the user does different choices from available options. These options reflect the content of the Robot function library (see section 3.2.4). Thus the inputted specification can be extended and changed manually. When all the steps has been gone through an output directory is chosen. A setup description is then generated by assembling setup files according to the updated specification. This procedure is illustrated by figure 3.6.

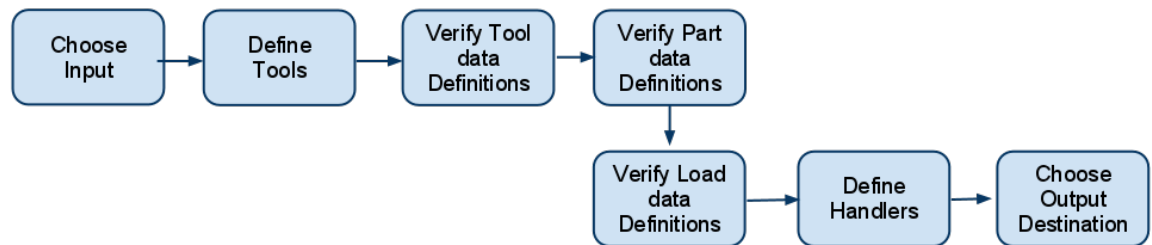


Figure 3.6.: Current Setup program steps

### 3.2.4. The Robot function library

The Robot function library, *krl\_routines*, used by VCC is divided into three different categories, represented by three folders (see figure 3.7):

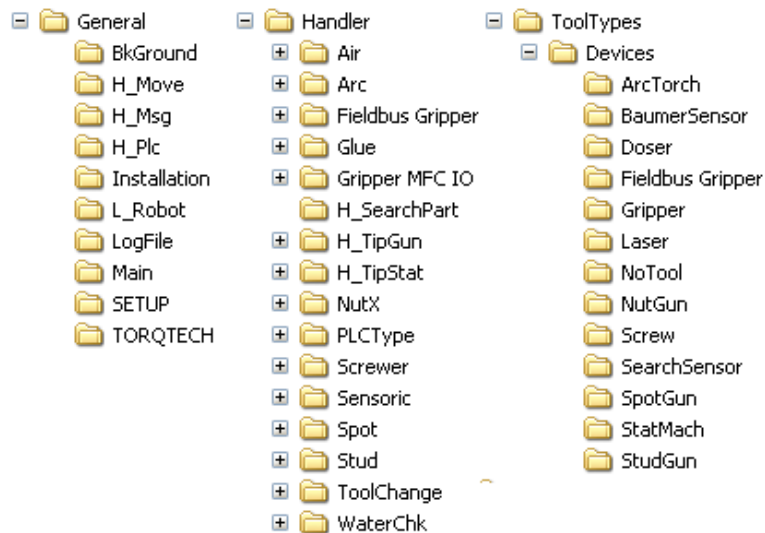


Figure 3.7.: The Robot function library folders

- The folder *General* contain General handler function libraries with source and initialization files that is included in every setup description.
- The folder *Handler* contain Optional handler function libraries that can be chosen through the Setup program.
- The folder *ToolTypes* contain function libraries for specific tools, as well as so called tool fork files which handles branching in execution depending on what tool is currently held by the robot.

Since the Robot function library will change over time, the Setup program need to adapt to the current content of the library. The available options in the program forms thus reflect the library folder structure. This is illustrated by two examples from the program, namely the tool form (figure 3.8) and the optional handler form (figure 3.9).

ToolId	External	ToolName	ToolLib	ToolType	ToolStand
0		NoTool			
1	TRUE	Tool1	Gripper	#Gripper	FALSE
2	FALSE	Tool2	SpotGun	#Gun	TRUE
3		Tool3			
4		Tool4			
5		Tool5			
6		Tool6			
7		Tool7			
8		Tool8			
9		Tool9			
10		Tool10			
11		Tool11			
12		Tool12			

DefaultTool= 2 The tool the robot starts with

LastTool= 2 Last defined tool (all tools)

next cancel

Figure 3.8.: Form for robot tools definition in current Setup program

Available optional handlers and possible values according to folder structure

Gripper MFC IO

NotUsed  
1Check  
d1Sequ  
d2Seq2Out  
d2Seq3Inp  
d2SeqBCD  
d2Sequ  
d3Seq3Inp  
d3Sequ  
d4Sequ  
NoSequ

next cancel

Figure 3.9.: Form for optional handlers in current Setup program

### 3.3. AutomationML

This chapter cover the basics of the AutomationML standard.

AutomationML (Automation Markup Language, also abbreviated AML) is a standard that is being developed with the goal to provide a data exchange format gluing together different engineering steps in the Digital Factory (Kuhn 2006). This is achieved by providing a top level XML architecture, which in turn incorporates other now established standards. The AutomationML standard is based upon CAEX (Computer Aided Engineering Exchange) which is a schema designed to describe hierarchical object structures. By including definitions for references AutomationML can be used to reflect different type of engineering information. Table 3.1 presents the standards that are supported by AutomationML today and what data they can represent.

Table 3.1.: Standards supported by AutomationML

Standard	Data
CAEX (IEC 62424)	Hierarchical structures, such as plant topology. Describes the attributes of different objects, and their relations.
Collada	Kinematics, graphics and geometry descriptions.
PLCopen XML	Logical behaviour, operational sequences and IO-connections.

The engineering tools associated with specific project tasks are often sophisticated. They however tend to work independently, without proper interfaces towards each other. The information from one tool is therefore transferred by manually assembled documents to be used as more or less manual input to the next tool. AutomationML is intended to solve this issue, allowing for faster and secure information transfers between tools. How this can be further conceptualized to provide a base for data and knowledge exchange between various engineering disciplines is discussed in (Schleipen and Bader 2010).

For AutomationML to work as a data exchange format the tools need to include proper interfaces providing export and import functionality. This is illustrated by figure 3.10. One important aspect pointed out is that AutomationML requires a global unique identifier (GUID) to be assigned to each object in the hierarchy. This way a specific object can be tracked from one tool to another.

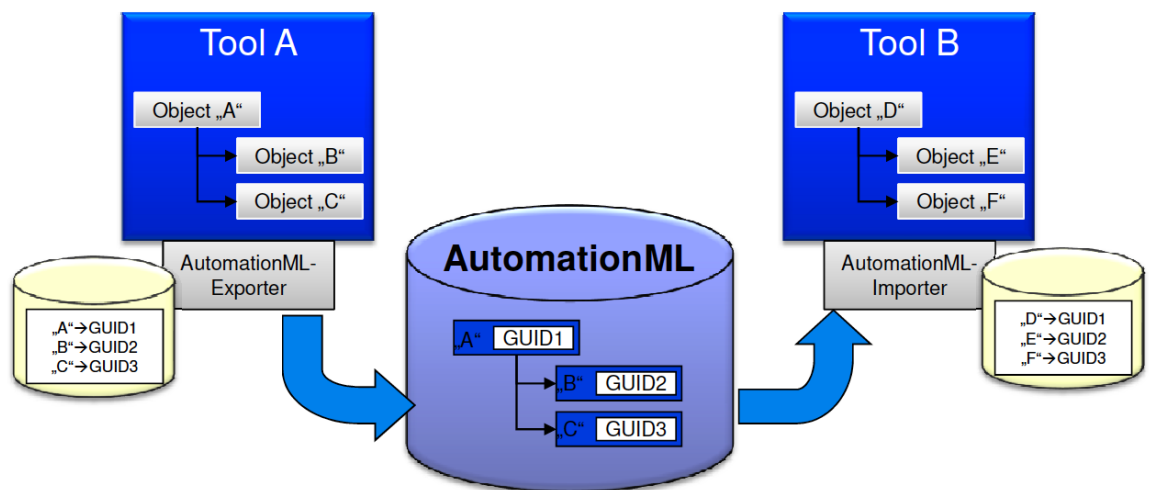


Figure 3.10.: Using AutomationML as data exchange between engineering tools

### 3.3.1. The AutomationML structure

The CAEX structure used in AutomationML will here be further explained. As mentioned CAEX provides a hierarchical representation of different objects. An example would be a production cell, with its different components (as depicted in figure 3.11). The top component Project is the name for the *InstanceHierarchy*. All its children object instances referred to as *InternalElements*. These can hold Attributes describing the properties of the object.

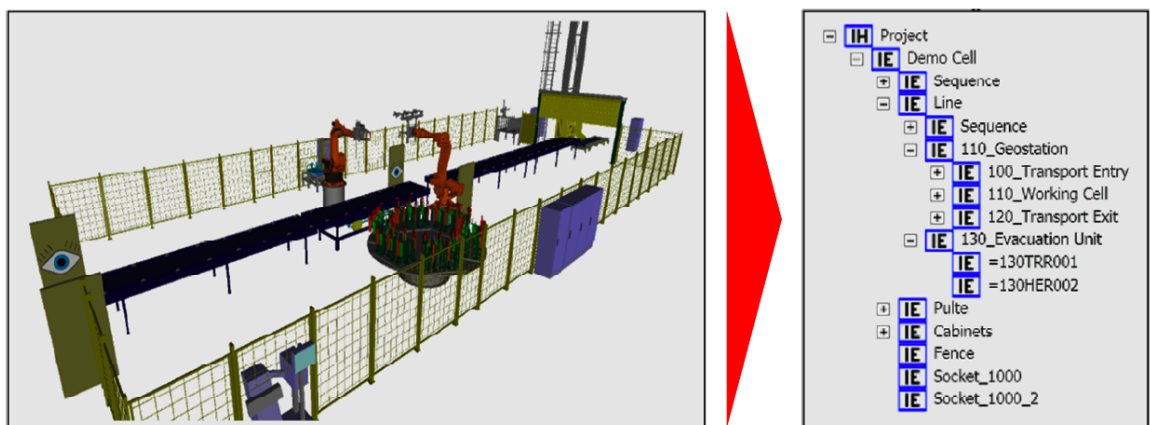


Figure 3.11.: Production line represented in AutomationML by InternalElements (IE) in an InstanceHierarchy (IH)

As mentioned an Internal Element is an object instance. There is thus an object concept included in the CAEX model. This is based upon the entity *RoleClass*. According to the AutomationML specification, every AutomationML object instance in the InstanceHierarchy should be refer to a specified RoleClass. The RoleClass provides an abstract view on the functionality of the object, and can be used for automatic semantic classification of the object when an AutomationML document is imported in a tool.

Each AutomationML RoleClass is derived from a base role class called *AutomationMLBaseRole*. The specification also defines a *RoleClassLibrary* with seven other base role classes. These are shown in figure 3.12.

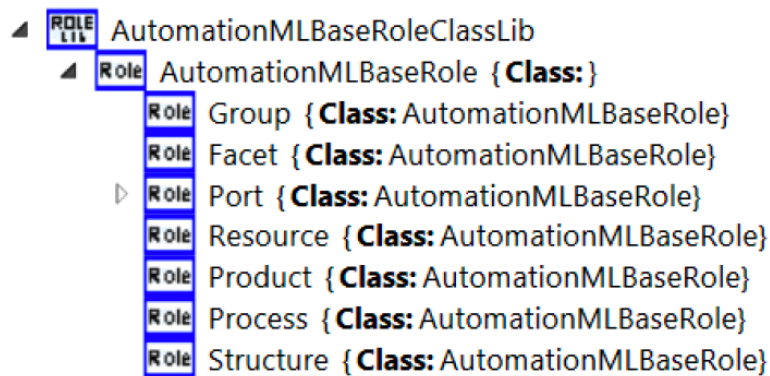


Figure 3.12.: AutomationML BaseRoleClass library

The second part of the AutomationML specification (which only exists as a white paper version at the time of writing this paper) is called AutomationML Libraries. It further develops the Role-Class concept by introducing additional RoleClassLibraries suitable for different applications. Of interest for this project is the Manufacturing Industry RoleClassLibrary found in figure 3.13. The guideline is that an InternalElement should refer to one of this RoleClasses. In case one RoleClass is too specific its parent is instead chosen.

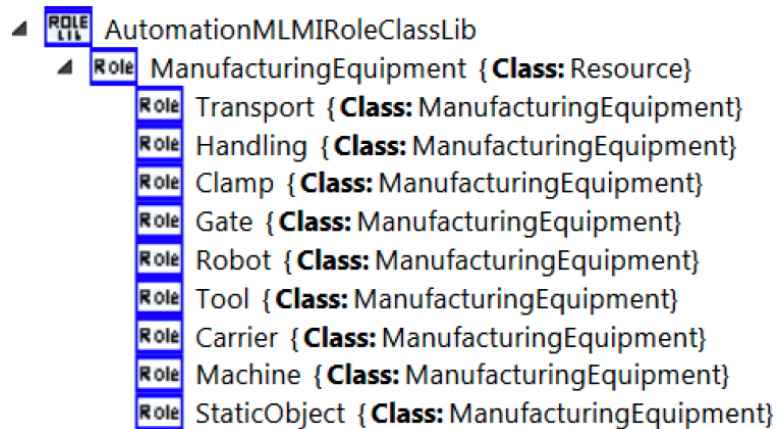


Figure 3.13.: AutomationML Manufacturing Industry RoleClass library

In addition to the RoleClasses AutomationML also allows for user defined *SystemUnitClasses*. These are user/vendor specific classes, for example different models of an object, and can support one or more RoleClasses. Attributes with optional default values and constraints can also be defined for these SystemUnitClasses. A SystemUnitClass is used as a template for an InternalElement. The RoleClass - SystemUnitClass - Object instance relations are visualised by figure 3.14 and figure 3.15.

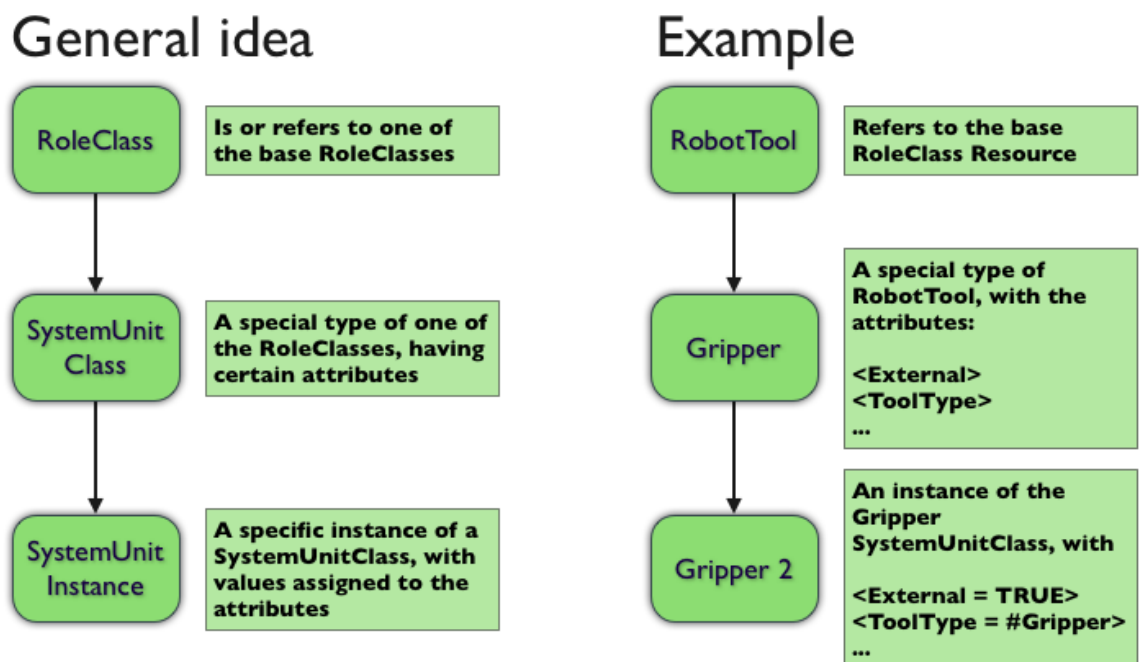


Figure 3.14.: AutomationML object concept



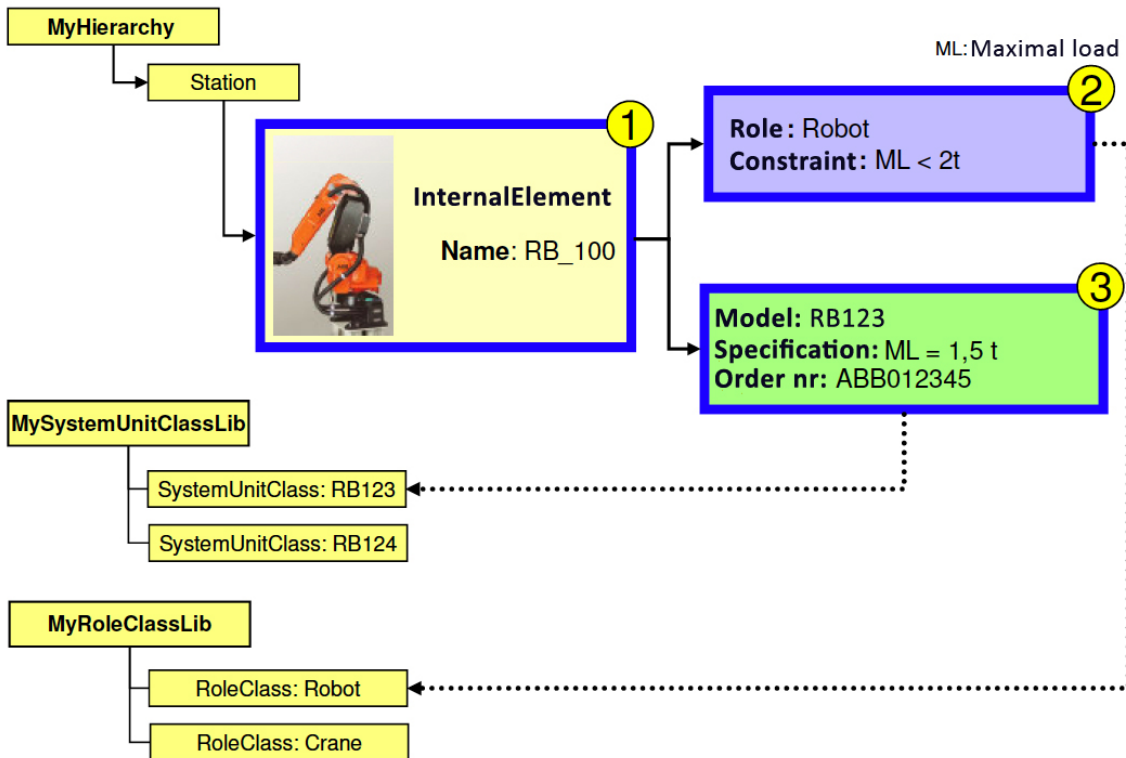


Figure 3.15.: AutomationML RoleClass - SystemUnitClass - InternalElement example

The aspects of AutomationML here presented are the essential ones in the scope of this project. AutomationML supports a range of other concepts as well. ExternalInterfaces provide connection point between objects. InternalLinks are used to establish such connections. The referencing of other documents, for example COLLADA and PLCopen XML documents, have also just been touched upon briefly. For descriptions of these and other concepts we refer to the AutomationML specification (Alonso-Garcia and Drath 2010). A good introduction is also provided by (Drath, Luder, Peschke and Hundt 2008).



## 4. Method

This chapter describes the steps taken in order to reach the final solution.

### 4.1. Rebuild of Setup program

The Robot function library has incorporated new functionality, so the Setup program need to be updated to meet the new requirements. The current Setup program is developed in Visual Basic, and its source code is not compilable in the new Visual Studio environment. To rebuild the Setup program from scratch can provide full control of software and outputs. Helpful documentation, lacking for the current Setup program, can then also be created. In addition, Visual Basic is old according to the software development maintenance.

### 4.2. Choice of software development environment

When choosing the software development environment (SDE) for the new Setup program we consider the following:

- In which operating system is the application going to be used?
- Which SDE:s are preferred at VCC?
- For which SDE:s are useful resources available?
- Which do we have previous knowledge of?
- Which would we find useful to learn more of?

The operating system used for most engineering tools at VCC is Microsoft Windows. Applications are commonly developed both in JAVA SDE:s and Microsoft Visual Studio according to software designers at VCC. The Process Simulate add-on, together with an accompanying API, has been developed in C# in Visual Studio 2005 edition. A C# class library for developing AML related applications, called the *AMLEngine* is provided by the AutomationML Group. This can be used in Visual Studio 2008 edition.

We have more experiences of developing in JAVA SDE:s, but find that development in C# would better suit the project. Since both are objected oriented languages with many similarities our

choice is using Visual C# 2008 edition. Learning this SDE during the process is also a good educational experience.

### 4.3. Developing a software model

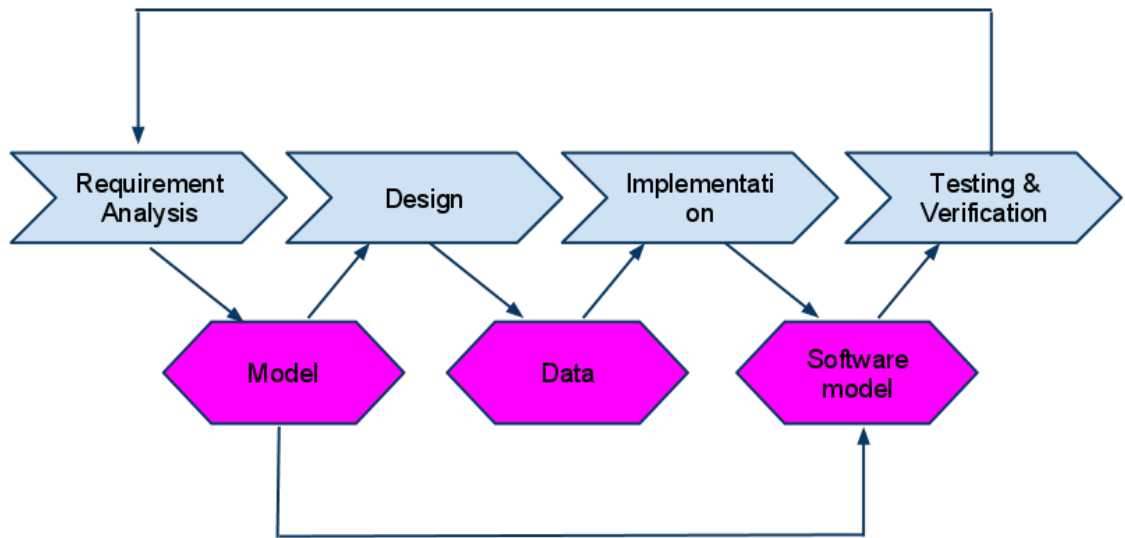


Figure 4.1.: Software development according to the waterfall model

The standard process of software development through waterfall model structure (figure 4.1) is used as a guideline. Considering the requirements from Volvo perspective, the software needs to fulfill all the functionalities of the old setup program. It is required to handle import and export of AML documents. The software should be highly flexible to allow for future extension when updates are needed.

A general working model (see section 4.3.1) is developed according to the requirements. With this as base software functionalities are designed and implemented. This is mostly done through a bottom to top manner, in which the lowest level algorithms first are implemented and then used when creating higher level algorithms. Finally every functionality is combined to form a complete software.

Each part of the software is tested for different input followed by verification of outputs. At last, verification of the data exchange link to the Process Simulate add-on is performed. Details on the verification methods are provided in section 4.7.

### 4.3.1. General model

A general model is developed according to the requirements. This is presented as a schema showing necessary functionalities and connections in figure 4.2.

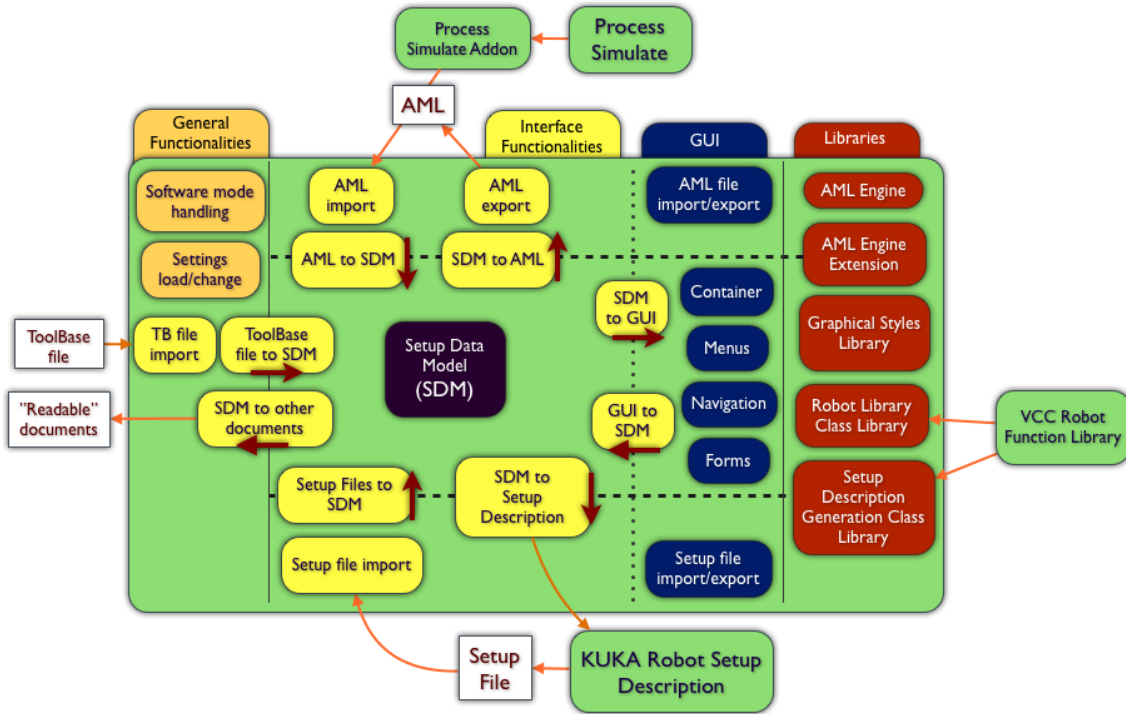


Figure 4.2.: General software model

The core of the software model is the *Setup Data Model* (see section 4.3.3), which is used as an internal format to store setup data. The following interfaces should be provided for the Setup Data Model:

- To and from AML documents (the new setup data files)
- To and from the GUI
- From the file ToolBase.ini (which provide Tool, Load and Part data)
- From the file SetUp.dat (the setup data file used by the old setup program)
- To a generated setup description (that can be downloaded to a robot controller)
- To other documents which can be easily read by a person

The GUI should allow for manipulation of these parts of the setup:

- Tool definitions
- Handler choices
- Load definitions
- Part definitions

This should be kept within a containing environment providing easy navigation and usability. To enable simple use of different robot libraries and output paths it is desirable to implement configurable program settings. Some suitable class libraries are also suggested:

- The AML engine provided by the AutomationML group
- An extension library to add VCC specific AML functionality
- A Graphical styles library, with templates that can be easily used by the GUI objects
- A Robot library class library, which provide a connection to the Robot function library
- A class library which contain algorithms for generating the Setup description.

#### **4.3.2. Robot function library connection**

For presenting the available tools and handlers the Setup program need to fetch this information from the Robot function library. Different versions of the library might be used and they will also undergo changes which should be reflected within the Setup program. Thus the software needs an import functionality that loads the necessary information from the Robot function library into classes. These classes can then be used as reference in different execution steps.

#### **4.3.3. Setup Data Model**

The setup description generated by the Setup program is an assembly of different function files which are adjusted to the setup specification. The specification contain information for the setup description components given in table 4.1. In order for the program to be able to operate on this data it needs to be represented by suitable class objects. This is referred to as the *Setup Data Model* (SDM).

The Setup Data Model is used as an intermediate data format. Each supported input file is parsed into the SDM, which is loaded into the GUI components. Changes can be applied by the user, and when generating output the GUI form object values are first stored back into SDM. The possibly changed SDM is then used as input for generation of a setup description or an AutomationML file. Figure 4.3 visualizes this in the form of a Data Flow Diagram.

Table 4.1.: Necessary Setup description components

Setup description component	Description
Robot data	Name and ID of robot, and some other parameters
Tools	Tools available for the robot along with necessary parameters
Tool configuration	Specifies Default tool, Last tool and Last tool with tool stand
Handling packages	Packages providing functionality for different tool types, communication interfaces, media control etc
Load data	Loads available for the robot along with necessary parameters
Part data	Parts available for the robot along with necessary parameters
Output paths	Directions on where the different part of the setup description should be outputted

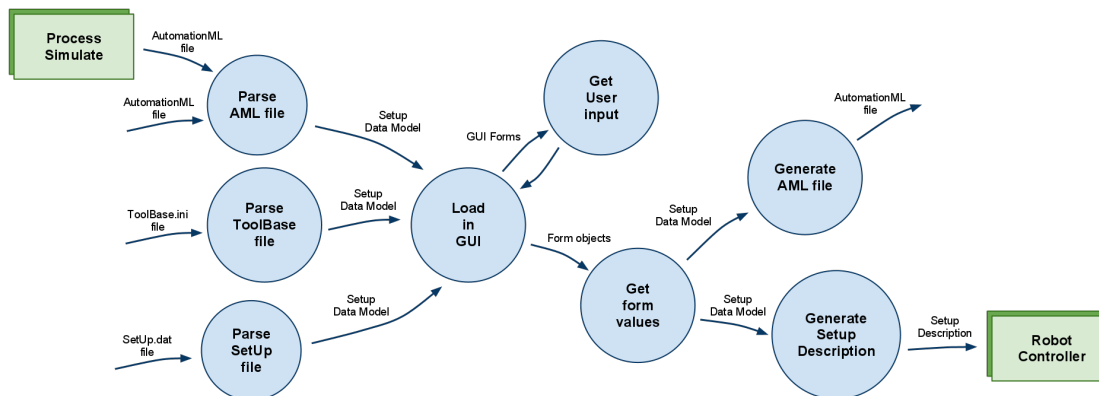


Figure 4.3.: Setup program data flow diagram

## 4.4. AutomationML Interface

As a base for the AutomationML interface is used the AutomationML C# class libraries provided by the AutomationML website. This is referred to as the *AMLEngine*. To better suit the needs for VCC in general and our solution in particular a new AML class library providing additional functionality is developed. This is called the *AMLEngine Extension*.

The Setup program deals with AML documents in three ways:

- As input retrieving the Process Simulate setup information
- As output storing the complete setup information
- As input retrieving the complete setup

The AMLEngine together with the new AMLEngine Extension provide necessary functionality for simple creation of AML documents. All content of the Setup Data Model needs a corresponding AutomationML representation. The concept of RoleClass - SystemUnitClass - InternalElement as described in section 3.3.1 will be used.

For enabling import of AML documents a parser needs to be built. This should be enough flexible to allow different layouts of the InstanceHierarchy holding the object instances. Objects will be identified by their roles. The imported data should be easily translated into the Setup Data Model.

## **4.5. Setup description generation**

In order to correctly implement all steps in the setup description generation all necessary algorithms are identified through:

- Inspection of the old Setup program source code
- Discussion with the VCC supervisor
- Comparison of the output from the new and old Setup program (see section 4.7.1)

## **4.6. Adjustment of Process Simulate add-on**

The Process Simulate add-on created in the preceding master thesis project produces an output that does not completely correspond to our AutomationML parser. For example, the parser is built upon the RoleClass concept (section 3.3.1), which was not used in the add-on AML output. To resolve this issue, the add-on need to be updated to meet the requirements for the parser. This is done through modification of the add-on source code.

## **4.7. Verification methods**

To assure that the correct information is imported, withheld and exported through the different interfaces it should be verified that:



- The Setup program generates a correct setup description.
- Data is consistent if not deliberately changed.
- Data is not affected by the changes to the Process Simulate add-on.

#### 4.7.1. Verification of generated setup description

It is vital that the setup description generated by our software is completely correct. Otherwise it might cause serious problems in the robot controller. This is verified by comparing the generated setup descriptions to:

- Setup descriptions generated for the same parameters in the old Setup program.
- A setup description provided from VCC, including the additions implemented.

In order to compare the output we use a software called *Beyond Compare*, which provides methods and GUI to easily check that:

- The content in two folders is the same. All files should be included, and no additional files are allowed.
- The content of two files is the same.

The final verification of the setup description would be to do a real download to a robot controller. The resources and time is however not enough in the project scope for performing this step. Since the Setup program is going to be put to use at VCC, this will be done quite soon after the writing of this report.

#### 4.7.2. Verification of data consistency

Data imported to the new setup program should be maintained in a correct ways. That means:

- No essential data should be missed when imported. No data should be unintentionally changed.
- No data should be unintentionally added.
- No data should be missed when outputted.

This can also be checked with *Beyond Compare*, by assuring that:

- An imported AML document is sufficiently equal to an exported, when the setup is not changed through the GUI.
- An imported SetUp.dat-file is sufficiently equal to an exported, when the setup is not changed through the GUI.

#### **4.7.3. Verification of add-on change**

The changes done in the source code for the Process Simulate add-on should not affect the data stored in the outputted AML document, but only the document syntax and XML structure. This can be verified by comparing the output from the unchanged add-on with the output from the updated.

## 5. Results

This chapter provides the result of the project, divided into different sections.

### 5.1. Software model

To fulfill the General model presented in section 4.3.1 and the requirements of a modular, flexible solution the software model here presented was developed. All entities shown in figure 5.1 are class libraries with their own namespaces. A separation has been done between external existing libraries, external new libraries, data model, GUI, other IO interfaces, and other general libraries. A short description of the libraries functionalities are hereby given:

- External existing class libraries

**AMLEngine** Provided by the AutomationML Group

- External new class libraries

**VCC\_AMLEngine\_Extension** Providing additional functionality for the AMLEngine and VCC specific AML document generation and parsing. This might be useful for other applications dealing with AML as well.

- Data model

**SetupDataModel** The internal data model gluing together the software. Also see section 5.1.2 for more details.

- IO interfaces

**SetupFileImport** Parsers for reading data from the files SetUp.dat and ToolBase.ini into the Data Model.

**AML\_Interface** Parsing of AML documents into the Data Model, and methods for creating AML documents from the Data Model. Uses both the AMLEngine, and the VCC\_AMLEngine\_Extension.

SetupGeneration - Assembly of Setup description according to the Data Model.

- General functionalities

**Settings** Allows (and demands) of the user to specify paths to the Robot function library, the AutomationML Standard Library, the Default folder (containing templates) and (optional) standard output paths. The settings are stored to a file called Settings.txt in the application folder. This class library holds both objects for holding the settings data, and the GUI to adjust it.

**GraphicalStyles** Provide templates that can be used by GUI components.

- GUI structure

**Main, MainForm** Start of execution, and the overall GUI container.

**InputSelection** Choice of input source. Contain methods that connect to the import functionality of the IO interfaces.

**SetupGUI** Holds the form objects for a loaded (or empty) setup. Also contain methods that connect to the output functionalities of the IO interfaces.

**SetupOverview** Shows a grouped list of all setup data.

**ToolSelection** Allows for definition of tools.

**PartAndLoadData** Allows for definition of Part and Load data.

**HandlerSelection** Allows for definition of optional Handlers.

**CoordSelection** Allows for definition of objects with three or six degrees of freedom, such as Tool Center Points, or inertia. Used by ToolSelection and LoadSelection.

**LoadSelection** Allows for definition of loads. Used by ToolSelection and PartAndLoadData.

**OutputSelection** Allows for definition of output paths, and hold GUI components to call the output methods in SetupGUI.

The general model also includes functionality for creation of other readable documents. This functionality has been left out because of time limitations, but will be easily implemented if needed, since all necessary data is conveniently provided through the Setup Data Model.

The modular solution for the Setup program provide flexibility for future extensions, which was one of the subgoals. Appendix A contain a guide on how to extend the functionality of the Setup program.

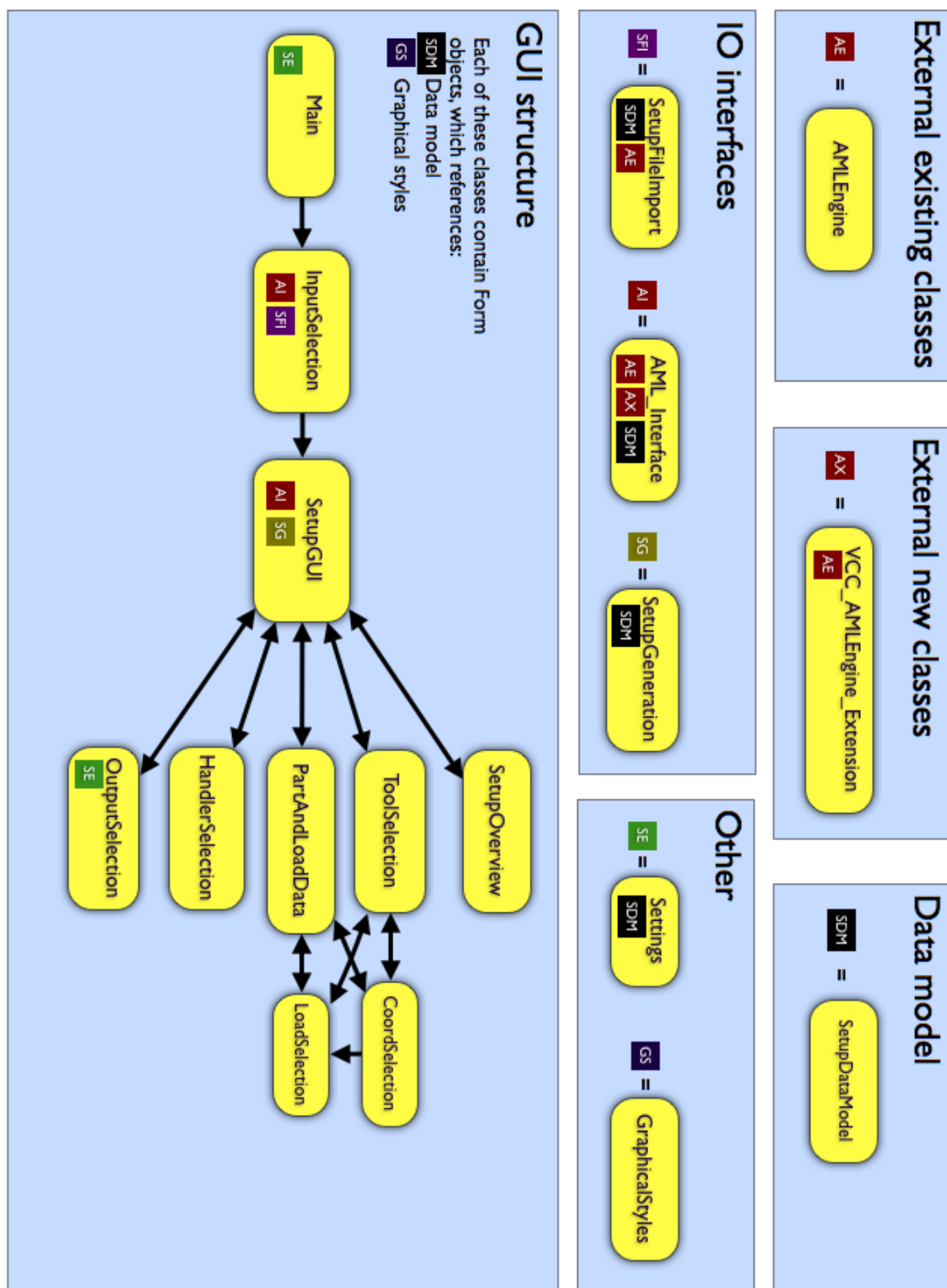


Figure 5.1.: Setup program class library overview

### 5.1.1. Robot function library connection

Since the information from the Robot function library should not be changed once loaded to the software, the content is loaded into a static class, named *RobLib* (see figure 5.2). This class is then used as a source for information about the Robot function library throughout the software. It provides the following information through different public fields:

- The names and paths of the directories within *krl\_routines/General*, *krl\_routines/Handler* and *krl\_routines/ToolTypes*
- Handler objects according to *krl\_routines/Handler* (see figure XX)
- ToolLib objects according to *krl\_routines/ToolTypes* (see figure XX)
- The ToolTypes associated with different Tool Libraries
- Which Tool Libraries should be regarded as Special Tool Libraries

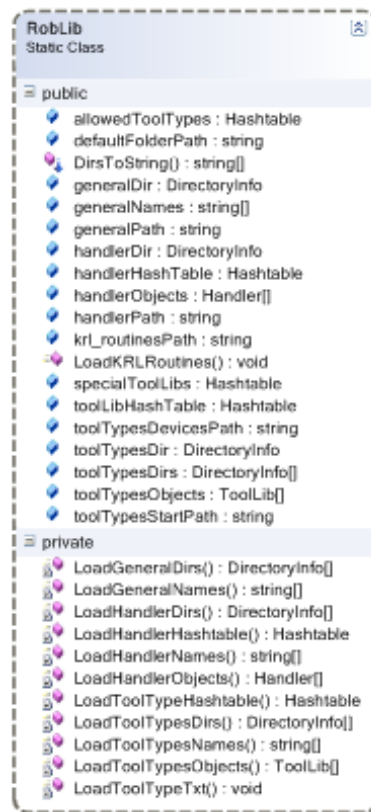


Figure 5.2.: The Robot function library class



### 5.1.2. Setup Data Model

The Setup Data Model (SDM) is designed to be a structural representation of the components of a robot setup. This is achieved by letting the components be represented by class objects with suitable inheritances and associations. All components are included in the instance of the object type Setup. The Setup class hold fields for the types Tool, Handler, ToolConfiguration, RobotData, PartData, LoadData and OutputPaths, as can be seen in the class model diagram in figure 5.3.

### 5.1.3. GUI design

We build the GUI according to the data model used in our model (figure XX above). We set the background color to blue in order to give a equable environment for customers. By creating five buttons and the same functional tool lists, we can choose among the required input formats.

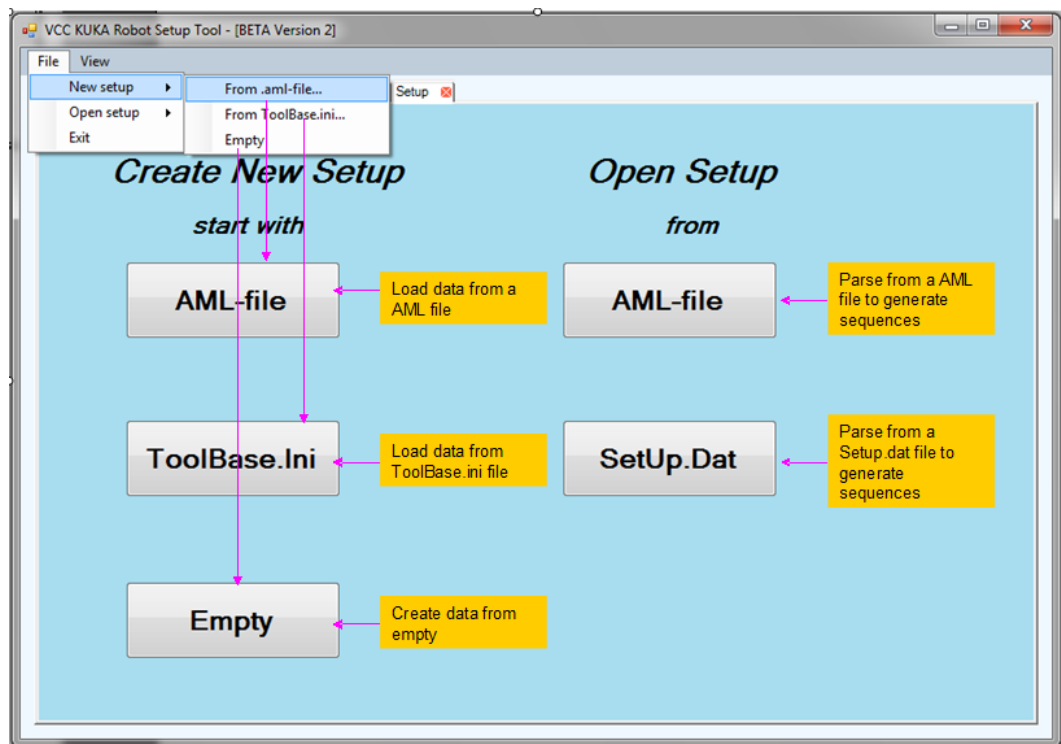


Figure 5.4.: GUI - Input selection

All GUI components such as the comboboxes and buttons are chosen according to the functionalities. For example, SetupForm is constructed and added to GUI by methods in InputSelectionForm (figure 5.4). In the ToolSelectionForm (figure 5.5, we get the Add Tool button to add any tool or delete tool by using Remove Tool. Considering the large number of different graphic



components to adjust, we choose a distinct class UseGraphicalStyles to set the styles and fonts of each graphic component. In this way we define the style formally.

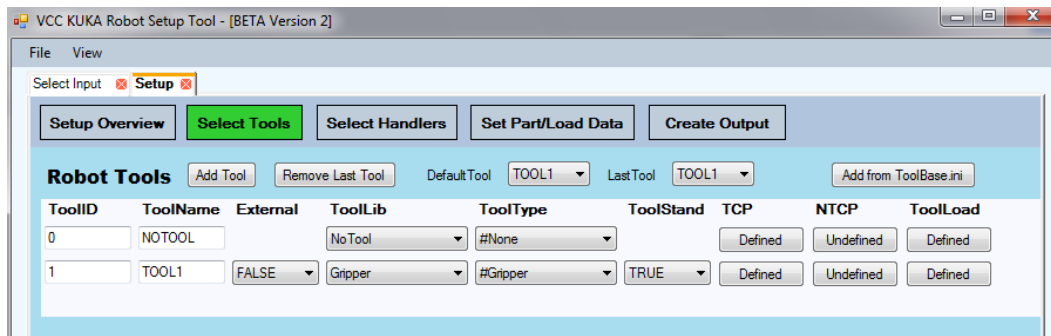


Figure 5.5.: GUI - Tool selection

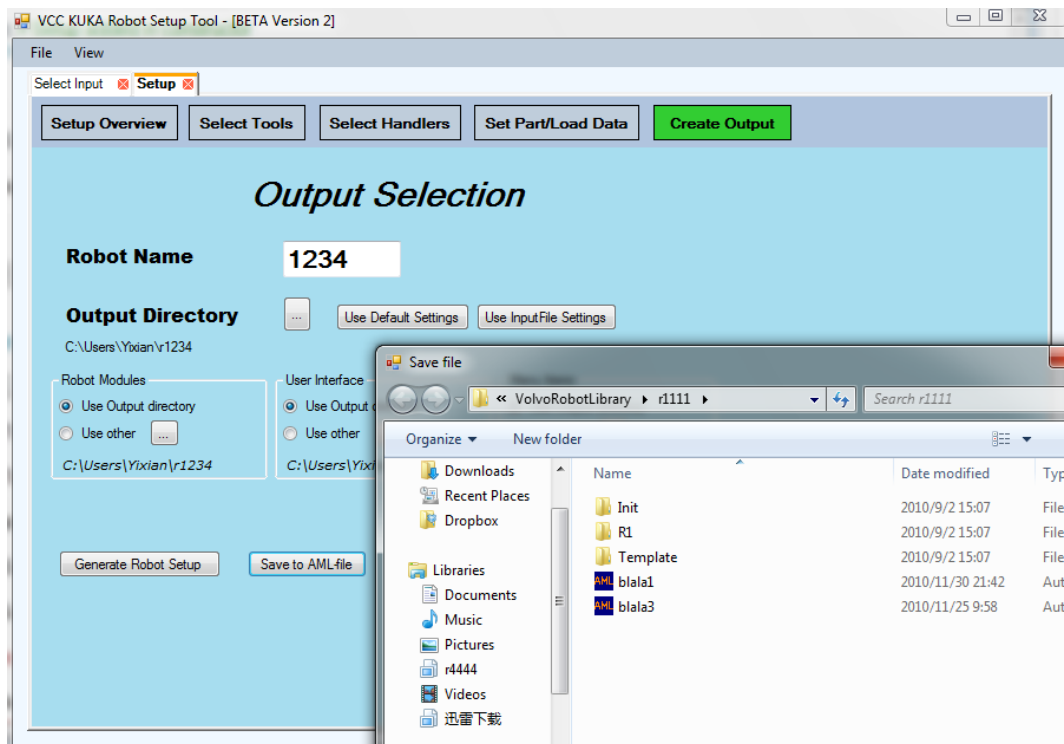


Figure 5.6.: GUI - Output selection

For creating an AML output file, we set the output directory first, and save it to the folder we choose as shown in the figure 5.6.

#### 5.1.4. Setup description generation

All steps in the setup description are included in the class library SetupGeneration. They are executed sequentially from the method CreateSetup. In order to not lose data if the generation process for some reason fails the current output directory is first copied to a backup folder. This is deleted when the new setup description has been correctly assembled. An in depth guide to all steps are provided in Appendix B.

### 5.2. Process Simulate connection

The first attempt to update the Process Simulate add-on was to rewrite the AML document generation using our new developed VCC\_AMLEngine.Extension class library. This solution could however not be implemented because the needed class libraries could not be used in Visual Studio 2005.

Instead the AML document was created in the same manner as it was originally, by a combination of hard coding and a few help methods. This solution is not as neat from a source code maintenance point of view, but at least generates an AML document according to our updated requirements.

### 5.3. AutomationML

Three aspects of the AutomationML related results will here be presented. First is an example of the new AMLEngine Extension. This is followed by a presentation of the AutomationML representation of the Setup Data Model. The third aspect is an overview of the import functionality, the AutomationML document parser.

#### 5.3.1. The AMLEngine Extension

When doing tests of the AMLEngine class library provided at the AutomationML website there were some programming steps in the document creation that appeared non-intuitive when compared to the rest. An example of that is the recurring task of creating an InternalElement using a SystemUnitClass as template. The code for doing this using the AMLEngine looks like this:

```
// Create instance of SystemUnitClass
Altova.Xml.TypeBase instance = systemUnitClass.CreateClassInstance();
// Cast instance of SystemUnitClass to InternalElement
InternalElementType internalElement = (InternalElementType)instance;
// Assign a name to the new InternalElement
internalElement.Name.Value = "internalElementName";
```

Especially the first operation involves general XML operations, which draws attention from the AutomationML structure we want to work with. A cast is then necessary to get to type we actually want. This lead us to develop the AMLEngine Extension - a class library built upon the AMLEngine, which add additional functionality and cover some of the flaws. The same code is then used within a function, which can be called by a more simple (and intuitive) command:

```
InternalElementType internalElement =  
    systemUnitClass.CreateInternalElement("internalElementName");
```

Other functionality implemented in the AMLEngine Extension is:

- Creation of Attributes with defined fields in one step
- Fetching of objects (Attributes, RoleClasses, SystemUnitClasses) by their names, instead of by uninformative indexes.
- Appending elements after others in the hierarchy (instead of inserting them on top, which is default in the AMLEngine).

Apart from this general functionality the AMLEngine Extension also contain methods for creation of AutomationML data specific to VCC and our solution. A straightforward example of this is shown in figure 5.7. This shows the method for creating a Robot object. Role, SystemUnit and InternalElement are all created in one go. Types and methods specific to the AMLEngine is marked **bold**. Methods from the AMLEngine Extension is marked ***bold italic***. The included comments (preceded by dual slash //) should be sufficient to grasp the meaning of the code. The resulting AutomationML output is showed in figure 5.8.

## 5.4. Verification

Three verification steps are provided in this section:

- Verification of generated setup description
- Verification of data consistency
- Verification of Process Simulate connection

```

        public static InternalElementType Robot(String robotName, String robotID, String robotType,
String robotController, RoleClassLibType roleClassLib, SystemUnitClassLibType systemUnitClassLib,
InstanceHierarchyType instanceHierarchy)
        {
            // --- ROLECLASS --- //
            // Create reference to existing RoleClass
            String roleClassRef = "AutomationMLMIRoleClassLib/ManufacturingEquipment/Robot";

            // --- SYSTEMUNITCLASS --- //
            // Create new systemUnitClass to systemUnitClassLib
            SystemUnitFamilyType systemUnitClass = systemUnitClassLib.By_name("Robot");
            if (systemUnitClass == null) // Create new systemunitclass if not existing
            {
                systemUnitClass = systemUnitClassLib.New_SystemUnitClass("Robot");
                // Add attributes
                systemUnitClass.New_Attribute("Name", "xs:string");
                systemUnitClass.New_Attribute("RobotID", "xs:string");
                systemUnitClass.New_Attribute("RobotType", "xs:string");
                systemUnitClass.New_Attribute("RobotController", "xs:string");
                // Add supported roleclass
                systemUnitClass.New_SupportedRoleClass(roleClassRef);
            }

            // --- INTERNAL ELEMENT --- //
            InternalElementType internalElement = systemUnitClass.CreateInternalElement("Robot");
            // Set Attributes
            internalElement.Attribute.By_name("Name").Value = robotName;
            internalElement.Attribute.By_name("RobotID").Value = robotID;
            internalElement.Attribute.By_name("RobotType").Value = robotType;
            internalElement.Attribute.By_name("RobotController").Value = robotController;
            // Set RoleRequirement
            internalElement.New_RoleRequirements().RefBaseRoleClassPath.Value = roleClassRef;
            // Add to InstanceHierarchy
            instanceHierarchy.Append_InternalElement(internalElement);

            // Return the InternalElement node to this robot
            return internalElement;
        }
    }

```

Figure 5.7.: Creating AML data with the AML Engine and the AML Engine Extension

```

<InstanceHierarchy Name="MyInstanceHierarchy">
  <InternalElement Name="Robot" RefBaseSystemUnitPath="RobotSystemUnitClassLib/Robot"
ID="{a5c94adf-0b29-4121-9171-3e397eea63c1}">
    <Attribute Name="Name" AttributeDataType="xs:string">
      <Value>ir3401</Value>
    </Attribute>
    <Attribute Name="RobotID" AttributeDataType="xs:string">
      <Value>PP-NV_TEST1-11-2-2010-8-45-37-903580-921911</Value>
    </Attribute>
    <Attribute Name="RobotType" AttributeDataType="xs:string">
      <Value>
        </Value>
    </Attribute>
    <Attribute Name="RobotController" AttributeDataType="xs:string">
      <Value>Tecnomatix.Engineering.TxRobotRRSController</Value>
    </Attribute>
    <SupportedRoleClass RefRoleClassPath="AutomationMLMIRoleClassLib/ManufacturingEquipment/Robot" />
    <RoleRequirements RefBaseRoleClassPath="AutomationMLMIRoleClassLib/ManufacturingEquipment/Robot" />
  </InternalElement>
</InstanceHierarchy>
<SystemUnitClassLib Name="MySystemUnitClassLib">
  <SystemUnitClass Name="Robot">
    <Attribute Name="Name" AttributeDataType="xs:string" />
    <Attribute Name="RobotID" AttributeDataType="xs:string" />
    <Attribute Name="RobotType" AttributeDataType="xs:string" />
    <Attribute Name="RobotController" AttributeDataType="xs:string" />
    <SupportedRoleClass RefRoleClassPath="AutomationMLMIRoleClassLib/ManufacturingEquipment/Robot" />
  </SystemUnitClass>
</SystemUnitClassLib>

```

Figure 5.8.: Output generated from the method in figure 5.7

### 5.4.1. Verification of generated setup description

To verify that the generated setup description corresponded to one generated with the old setup program the outputs were compared using the software Beyond Compare. This was an iterative process where new aspects of the setup description generation were discovered and adjusted to. Figure 5.9 shows an example where some files included in the old setup description are not included in the new.

Name	Size	Modified
Screwdriver	7 075	2010-11-26 15:17:07
Sensor	21 081	2010-11-26 15:17:07
Spot	290 666	2010-11-26 15:17:08
Stud	114 464	2010-11-26 15:17:09
diSequ.dat	25 318	2010-11-26 15:17:27
diSequ.src	656	2010-06-21 16:06:07
dGrpTC.dat	1 395	2010-11-26 15:17:06
dGrpTC.src	4 094	2010-06-21 16:06:02
dMultiTS.dat	1 579	2010-11-26 15:17:09
dMultiTS.src	4 126	2010-06-21 16:05:54
DoOrder.dat	846	2010-11-26 15:17:09
DoOrder.src	1 148	2010-06-21 16:05:54
htoolChg.dat	878	2010-11-26 15:16:57
htoolChg.src	2 718	2010-06-21 16:06:06
JobSel.src	1 148	2010-11-26 15:17:09
Main.dat	2 867	2010-06-21 16:05:54
Main.src	577	2010-06-21 16:06:06
SetUp.dat	498	2010-11-26 15:16:57
ToolLock.dat	554	2010-06-21 16:06:06
ToolLock.sub	4 569	2010-11-26 15:17:28
ToolBase.ini	925	2010-11-26 15:17:09
vssver.scc	5 808	2010-06-21 16:05:54

Name	Size	Modified
Screwdriver	7 069	2010-11-26 16:59:13
Sensor	21 077	2010-11-26 16:59:13
Spot	290 666	2010-11-26 16:59:13
Stud	114 460	2010-11-26 16:59:13
diSequ.dat	25 411	2010-11-26 16:59:14
DoOrder.dat	876	2010-06-21 16:06:06
DoOrder.src	2 718	2010-06-21 16:06:06
JobSel.src	577	2010-06-21 16:06:06
Main.dat	496	2010-06-21 16:06:06
Main.src	554	2010-06-21 16:06:06
SetUp.dat	4 776	2010-11-26 16:59:14
ToolBase.ini	2 331	2010-06-21 16:05:51
vssver.scc	80	2010-06-21 16:05:51

Figure 5.9.: Output generated from the original Setup program to the left, and from the new to the right. Some files are not included.

The files assembled within the setup description generation were also checked through the software. Information missed out, as shown in figure 5.10, made necessary additional changes to the setup description generation algorithms.

File	Content
Left	<pre>;FOLD MAP Globals ;***** ;MAP globals DECL BOOL MapStopRequest=FALSE DECL BOOL MapStopRespond=FALSE ;ENDFOLD  ;FOLD PLC Signals Background SIGNAL I_ProcessStop \$IN[75] ;Process Stop SIGNAL O_BiSafe \$OUT[3] ;BiSafe SIGNAL O_DrivesOn \$OUT[41] ;Drives is on SIGNAL O_StopRequest2Plc \$OUT[42] ;Stop request to PLC ;ENDFOLD  ;FOLD PLC Variables STRUC AlarmSignalStruc INT Number,BOOL ErrorStat,CHAR Info[40] ;ENDFOLD  ;FOLD KUKA Standard Calibration Variables ;variables for KUKA standard calibration STRUC EX_AX_DATA_T FRAME ROOT,ESYS EX_KIN,FRAME OFFSET</pre>
Right	<pre>;FOLD MAP Globals ;***** ;MAP globals DECL BOOL MapStopRequest=FALSE DECL BOOL MapStopRespond=FALSE ;ENDFOLD  ;FOLD PLC Signals Background  ;FOLD PLC Variables  ;FOLD KUKA Standard Calibration Variables ;variables for KUKA standard calibration STRUC EX_AX_DATA_T FRAME ROOT,ESYS EX_KIN,FRAME OFFSET</pre>

Figure 5.10.: File generated by the original Setup program to the left, and by the new to the right. Some lines are not included.

The final comparison contained all available ToolTypes and all different Handlers. This is not a probable real scenario, but assures that each possible scenario should be correct. The result is given in figure 5.11, figure 5.12 and figure 5.13.

Init	14 277	2010-11-26 15:17:28	Init	12 921	2010-11-26 18:45:55
MenuKeyUser.ini	14 277	2010-11-26 15:17:28	MenuKeyUser.ini	12 921	2010-11-26 18:45:55
R1	1 053 212	2010-11-26 15:17:28	R1	1 053 237	2010-11-26 18:45:55
Arc	78 076	2010-11-26 15:16:58	Arc	78 054	2010-11-26 18:45:49
Glue	153 658	2010-11-26 15:17:05	Glue	153 642	2010-11-26 18:45:49
Gripper	57 493	2010-11-26 15:17:06	Gripper	57 479	2010-11-26 18:45:49
IRobot	88 820	2010-11-26 15:17:27	IRobot	88 692	2010-11-26 18:45:55
ITools	112 580	2010-11-26 15:16:57	ITools	112 569	2010-11-26 18:45:49
Media	14 735	2010-11-26 15:17:10	Media	14 729	2010-11-26 18:45:50
Nut	42 262	2010-11-26 15:17:07	Nut	42 250	2010-11-26 18:45:50
PLC	11 774	2010-11-26 15:17:07	PLC	11 770	2010-11-26 18:45:50
Screwdriver	7 875	2010-11-26 15:17:07	Screwdriver	7 869	2010-11-26 18:45:50
Sensor	21 081	2010-11-26 15:17:07	Sensor	21 077	2010-11-26 18:45:50
Spot	290 690	2010-11-26 15:17:08	Spot	290 666	2010-11-26 18:45:50
Stud	114 464	2010-11-26 15:17:09	Stud	114 450	2010-11-26 18:45:50

Figure 5.11.: Comparison containing all ToolTypes and Handlers

SpotsAtLastSmall=SpotCnt	SpotsAtLastSmall=SpotCnt
FacesDone=FaceCnt-FacesAtLastChange	FacesDone=FaceCnt-FacesAtLastChange
RobotInService[1]=TRUE	RobotInService[10]=TRUE
CASE #BadCutting	CASE #BadCutting
SpotsAtLastSmall=SpotCnt-SpotsPerSmall	SpotsAtLastSmall=SpotCnt-SpotsPerSmall
FlangeAtLastRight=FlangeAtLastRight-Flange	FlangeAtLastRight=FlangeAtLastRight-Flange

Figure 5.12.: ToolType file comparison. Incorrect output from original Setup program, to the left.

VSearchOK2= SearchOK2, 2010, INLINEFORM	VSearchOK2= SearchOK2, 2010, INLINEFORM
VSearchOK3= SearchOK3, 2010, INLINEFORM	VSearchOK3= SearchOK3, 2010, INLINEFORM
VSearchOK4= SearchOK4, 2010, INLINEFORM	VSearchOK4= SearchOK4, 2010, INLINEFORM
VSearchOK5= SearchOK5, 2010, INLINEFORM	VSearchOK5= SearchOK5, 2010, INLINEFORM
VolvoAnySearch= Search, , , , POP	VolvoAnySearch= Search, , , , POP
{VolvoBar}	
VolvoAnySearchTech= VSearchCtrl, VSe	
{mTechnologie}	
VolvoAnySearch	
{VOLVOkeys}	
VSINIT= Init, 2010, INLINEFORM, KUKAT	VSINIT= Init, 2010, INLINEFORM, KUKAT
VSEXPTEP= Ptp, 2010, INLINEFORM, KUKATP	VSEXPTEP= Ptp, 2010, INLINEFORM, KUKATP

Figure 5.13.: Teach Pendant menu file. Incorrect output from original Setup program, to the left.

Every Handler library is correctly copied. The ToolType files in lTools are equal, with the difference that the old setup tool outputs incorrect files when the tool id number reaches 10 and above (see figure 5.12). It is not likely that any VCC robot setup will contain that many tools,

and that is the probable reason that this wrong output has not been discovered. The menu-file for the teachpendant, MenueKeyUser.ini was not either correctly assembled by the old setup program for some ini-files (see figure 5.13). Both these issues is corrected in the new setup program.

■ \$config.dat	25 318	2010-11-26 15:17:27	≈	■ \$config.dat	25 411	2010-11-26 18:45:55
■ BkGround.sub	656	2010-06-21 16:06:07	≈	■ BkGround.sub	656	2010-06-21 16:06:07
■ d1Sequ.dat	1 395	2010-11-26 15:17:06	≈	■ d1Sequ.dat	1 393	2010-06-21 16:06:02
■ d1Sequ.src	4 094	2010-06-21 16:06:02	≈	■ d1Sequ.src	4 094	2010-06-21 16:06:02
■ dGrpTC.dat	1 579	2010-11-26 15:17:09	≈	■ dGrpTC.dat	1 577	2010-06-21 16:05:54
■ dGrpTC.src	4 126	2010-06-21 16:05:54	≈	■ dGrpTC.src	4 126	2010-06-21 16:05:54
■ dMultiTS.dat	846	2010-11-26 15:17:09	≈	■ dMultiTS.dat	844	2010-06-21 16:05:54
■ dMultiTS.src	1 148	2010-06-21 16:05:54	≈	■ dMultiTS.src	1 148	2010-06-21 16:05:54
■ DoOrder.dat	878	2010-11-26 15:16:57	≈	■ DoOrder.dat	876	2010-06-21 16:06:06
■ DoOrder.src	2 718	2010-06-21 16:06:06	≈	■ DoOrder.src	2 718	2010-06-21 16:06:06
■ hToolChg.dat	1 148	2010-11-26 15:17:09	≈	■ hToolChg.dat	1 146	2010-06-21 16:05:54
■ hToolChg.src	2 867	2010-06-21 16:05:54	≈	■ hToolChg.src	2 867	2010-06-21 16:05:54
■ JobSel.src	577	2010-06-21 16:06:06	≈	■ JobSel.src	577	2010-06-21 16:06:06
■ Main.dat	498	2010-11-26 15:16:57	≈	■ Main.dat	496	2010-06-21 16:06:06
■ Main.src	554	2010-06-21 16:06:06	≈	■ Main.src	554	2010-06-21 16:06:06
■ SetUp.Dat	4 569	2010-11-26 15:17:28	≈	■ SetUp.dat	4 776	2010-11-26 18:45:55
■ ToolLock.dat	925	2010-11-26 15:17:09	≈	■ ToolLock.dat	923	2010-06-21 16:05:54
■ ToolLock.sub	5 808	2010-06-21 16:05:54	≈	■ ToolLock.sub	5 808	2010-06-21 16:05:54

Figure 5.14.: Full comparison. Original Setup program to the left, new Setup program to the right.

All files are correctly copied to the R1 folder (figure 5.14). The assembled file \$config.dat differs only in that the new output replaces xxyy with the robot number, which should be more correct. The SetUp.Dat-file is equal apart from the time-stamp and output folder specified.

Template	146 396	2010-11-26 15:17:28	≈	Template	146 627	2010-11-26 18:45:55
■ VolvoTech.kfd	37 919	2010-11-26 15:17:28	≈	■ VolvoTech.kfd	37 913	2010-11-26 18:45:55
■ hMove.kfd	22 220	2010-11-26 15:16:56	≈	■ hMove.kfd	22 180	2010-11-26 18:45:49
■ hSearch.kfd	8 894	2010-11-26 15:17:06	≈	■ hSearch.kfd	8 892	2010-11-26 18:45:49
■ hSpot.kfd	8 785	2010-11-26 15:17:08	≈	■ hSpot.kfd	8 785	2010-11-26 18:45:50
■ hNutx.kfd	4 921	2010-11-26 15:17:07	≈	■ hNutx.kfd	5 429	2010-11-26 18:45:50
■ hUniGrip.kfd	2 830	2010-11-26 15:17:05	≈	■ hUniGrip.kfd	2 828	2010-11-26 18:45:49
■ Collision.kfd	2 292	2010-11-26 15:16:57	≈	■ Collision.kfd	2 290	2010-11-26 18:45:49
■ dSensor.kfd	2 213	2010-11-26 15:17:07	≈	■ dSensor.kfd	2 211	2010-11-26 18:45:50
■ ToolBase.ini	1 740	2010-06-21 16:05:52	≈	■ ToolBase.ini	1 740	2010-06-21 16:05:52

Figure 5.15.: Full comparison. Template files folder.

A few template files differ in what default values are assigned in certain definitions (figure 5.15). The new setup program output values according to directions gotten from discussions with our supervisor at VCC.

Another comparison was made for a Fieldbus gripper setup. This type of setup cannot be produced by the old setup program. Provided to us was instead a partly manually created setup description. The result is presented in figure 5.16.

Init	5 150	2010-11-02 11:53:49		Init	5 146	2010-11-2	
■ MenueKeyUser.ini	5 150	2010-09-27 09:50:24	≈	■ MenueKeyUser.ini	5 146	2010-11-2	
R1	200 427	2010-11-02 11:53:49		R1	200 704	2010-11-2	
Gripper	35 041	2010-11-02 11:53:49		Gripper	35 033	2010-11-2	
IRobot	97 135	2010-11-02 11:53:49		IRobot	97 063	2010-11-2	
ITools	27 715	2010-11-02 11:53:49		ITools	27 735	2010-11-2	
Media	2 800	2010-11-02 11:53:49		Media	2 798	2010-11-2	
PLC	12 101	2010-11-02 11:53:49		PLC	12 097	2010-11-2	
Utilities	5 287	2010-11-02 11:53:49		Utilities	5 285	2010-11-2	
\$config.dat	10 348	2010-09-27 09:50:24	≈	\$config.dat	10 381	2010-11-2	
BkGround.sub	656	2009-02-13 14:27:16	≈	BkGround.sub	656	2009-02-1	
DoOrder.dat	1 037	2010-09-27 09:50:18	≈	DoOrder.dat	1 035	2009-02-1	
DoOrder.src	2 515	2009-02-16 16:34:22	≈	DoOrder.src	2 515	2009-02-1	
JobSel.src	2 661	2009-02-13 14:27:18	≈	JobSel.src	2 661	2009-02-1	
Main.dat	590	2010-09-27 09:50:18	≈	Main.dat	588	2009-02-1	
Main.src	644	2009-02-13 14:27:20	≈	Main.src	644	2009-02-1	
SetUp.Dat	1 897	2010-09-27 09:50:24	≈	SetUp.dat	2 213	2010-11-2	
Template	44 991	2010-11-10 17:00:25		Template	44 888	2010-11-2	
Collision.kfd	2 292	2010-09-27 09:50:18	≈	Collision.kfd	2 290	2010-11-2	
expl.ico	768	2010-09-27 09:50:18	=	expl.ico	766	2009-02-1	
GripperClose.ico	768	2010-09-27 09:50:18	=	GripperClose.ico	766	2009-02-1	
GripperClosed.ico	768	2010-09-27 09:50:18	=	GripperClosed.ico	766	2009-02-1	
GripperOpen.ico	768	2010-09-27 09:50:18	=	GripperOpen.ico	766	2009-02-1	
GripperOpend.ico	768	2010-09-27 09:50:18	=	GripperOpend.ico	766	2009-02-1	
GripperTogg.ico	768	2010-09-27 09:50:18	=	GripperTogg.ico	766	2009-02-1	
hMove.kfd	21 817	2010-09-27 09:50:18	≈	hMove.kfd	21 815	2010-11-2	
hUniGrip.kfd	2 262	2010-09-27 09:50:18	≈	hUniGrip.kfd	2 260	2010-11-2	

Figure 5.16.: Comparison for Fielbus Gripper setup. Original Setup program output to the left, new Setup program output to the right.

In this case the files \$config.dat, MenueKeyUser.ini and hMove.kfd dont show any differences when compared. For the ToolType files in ITools the new setup program add the tool ID to some places where it is missed out in the manually created setup description. Another differing file is SetUp.dat. Apart from the time-stamp and output path differences the new setup program also adds Fieldbus gripper specific info to this file.

The setup description generated by the new setup program is thus equal or improved compared to the old.

#### 5.4.2. Verification of data consistency

Our first test of data consistency is to verify that all data stored in the AML documents are correctly imported, kept, and exported. A large test containing all ToolTypes and Handlers is used. The steps executed are:

- Create an AML document (doc1)
- Import doc1 to the Setup Data Model (SDM)
- Load the SDM into the GUI



- Save the GUI data back to SDM
- Create a new AML document (doc2)

If doc1 and doc2 are sufficiently equal according to our requirements, that means that our data has been kept consistent. The documents are compared using the text comparator in Beyond Compare. A small part of the result is provided in figure 5.17.

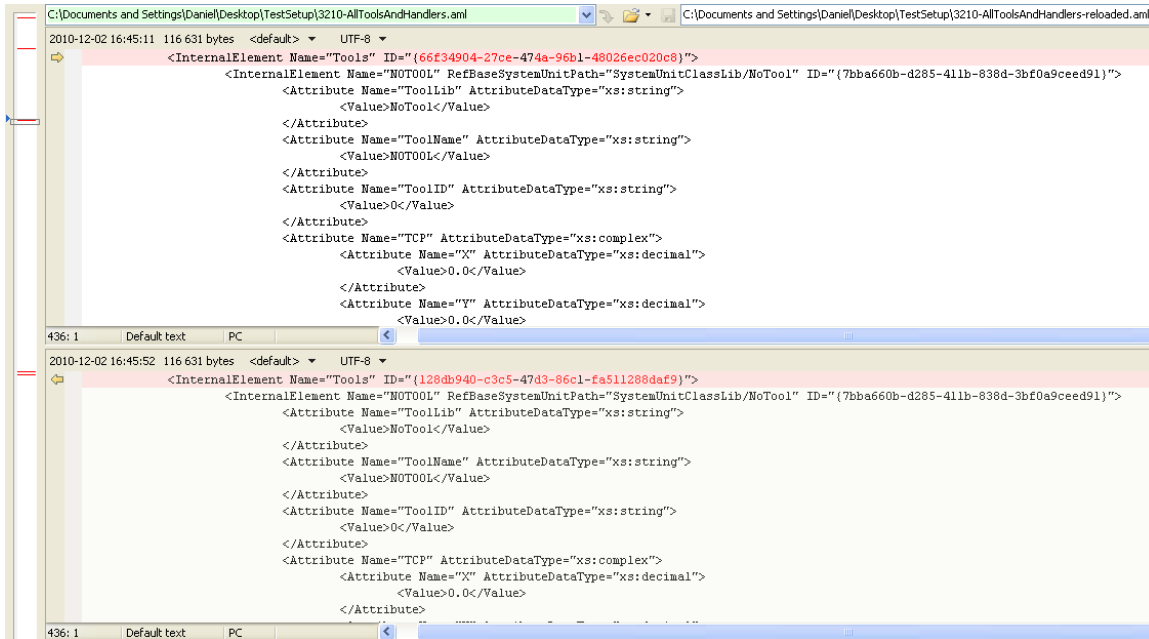


Figure 5.17.: AML document comparison

The only difference in the documents is the ID numbers for some InternalElements. This is in order, since they are solely used to form collections of other InternalElements holding the essential data. These inner InternalElements do keep their ID through the process, which means that the same ID is kept for the same Tool, Handler, Load and Part. This is essential to provide trackability of the specific objects. For example, in figure 5.17 the InternalElement Tools (which holds the Role Collection) differs in ID, but NoTool (of role Tool) have the same ID. With this we can conclude that the AML documents are sufficiently equal, and that data consistency is granted.

A second test of data consistency is to do the same experiment on the old setup program data storage - the file Setup.dat. In this comparison the only difference (see figure 5.18) is the time for creating the file, which of course is alright.

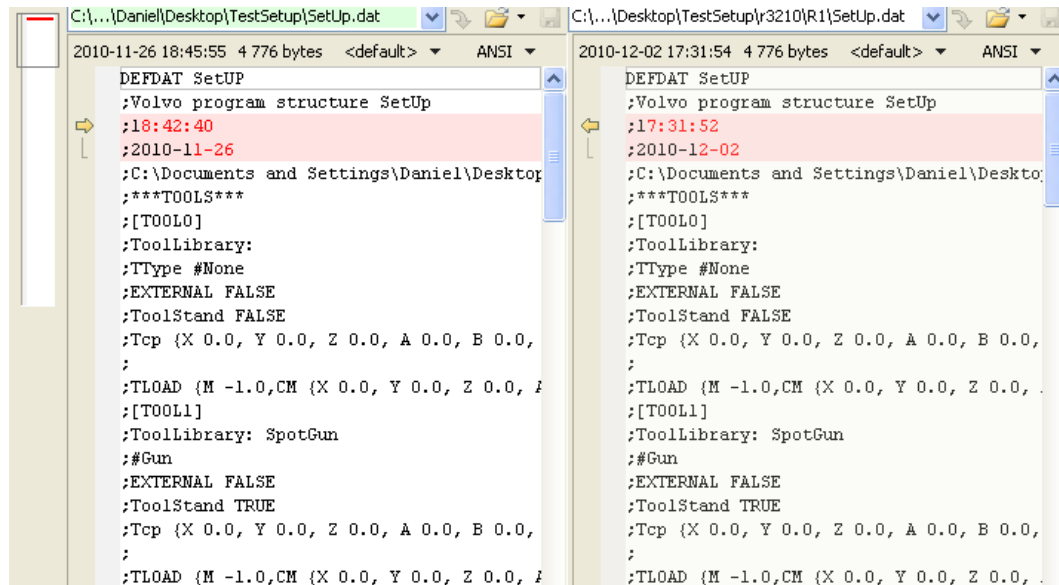


Figure 5.18.: Comparison of SetUp.dat files

### 5.4.3. Verification of Process Simulate connection

The essential part in the update of the Process Simulate add-on is that all data stored in the outputted AML document is the same. This was also the result we received when comparing an output from the original add-on with an output from the updated. The same Process Simulate project was used in both cases.

When using the new AML output as input to our Setup program all information fetched from Process Simulate is also correctly transferred. The requested data exchange connection is therefore obtained.

## 6. Conclusion

For this work, a working data exchange link was established between the simulation environment and a new robot controller setup program. Robot setup data was extracted from Process Simulate through an add-on, and outputted to a document following the AutomationML specification. Through the construction of a parser this data could be imported into the Setup program, which allows for adjustments through a GUI. Setup descriptions was generated by the Setup program according to the setup specification. These contains control code files for complete robot function packages that can be directly downloaded to a robot controller. Verification of the steps in this process assure data consistency and correct interface functionalities.

The setup data retrieved from the simulation model is partial. A natural future step would be to extend the data extraction API to include other robot tool configurations. This is supported by the AutomationML document, and to large extent prepared for in the AML document parsing. Other parts of the setup data need to be provided by a new common framework or database. This would hold information about available resources and keep a common naming standard, so that for example the different handling packages could be correctly associated to the robot controller setup at an earlier stage in the planning. The `VCC_AMLEngine.Extension` class library could be used to help this implementation, but that requires upgrade of the simulation tool API to at least Visual Studio 2008 version.



# References

- Alonso-Garcia, A. and Drath, R. (2010). Automationml whitepaper, web resource: [http://www.automationml.org/images/download/tecDoc/automationml whitepaper part 1 - automationml architecture v 1.4.pdf](http://www.automationml.org/images/download/tecDoc/automationml%20whitepaper%20part%201%20-%20automationml%20architecture%20v%201.4.pdf). available 2010-12-09.
- Drath, R., Luder, A., Peschke, J. and Hundt, L. (2008). Automationml - the glue for seamless automation engineering, *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, pp. 616 –623.
- Kanthabhabhajeya, S. and Pabello Rodriguez, R. E. (2010). *Automatic generation of control code for robot function packages. generation of robot set-up descriptions based on data in the simulation*, Master's thesis, Chalmers tekniska hskola.
- Kuhn, W. (2006). Digital factory - simulation enhancing the product and production engineering process, *Simulation Conference, 2006. WSC 06. Proceedings of the Winter*, pp. 1899 – 1906.
- Luder, A., Hundt, L. and Keibel, A. (2010). Description of manufacturing processes using automationml, *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pp. 1 –8.
- Ranky, P. G. (2004). Automotive robotics, *Industrial Robot: An International Journal*, Vol. 31:3, pp. 252–257.
- Rossdeutscher, M., Zuern, M. and Berger, U. (2010). Virtual robot program development for assembly processes using rigid-body simulation, *Computer Supported Cooperative Work in Design (CSCWD), 2010 14th International Conference on*, pp. 417 –422.
- Schleipen, M. and Bader, T. (2010). A concept for interactive assistant systems for multi-user engineering based on automationml.
- Schleipen, M. and Drath, R. (2009). Three-view-concept for modeling process or manufacturing plants with automationml, *Emerging Technologies Factory Automation, 2009. ETFA 2009. IEEE Conference on*, pp. 1 –4.



## **A. Setup Program Extension Guide**

# VCC Kuka Robot Setup Tool

## Developer's extension guide

### Overview

This guide provides information on how to extend the functionality of the VCC KUKA Robot Setup Tool. It will focus on the addition of a Special Tool with specific attributes. In order to achieve this additions to the Setup Data Model, the GUI, the AML interface and the Setup Generation will be needed. This guide can thus be used also to get a general understanding of the architecture of the VCC KUKA Robot Setup Tool, and how the different software modules interact.

### The TestTool

We will create the new Special Tool *TestTool*. "Special Tool" refers to a tool which have certain characteristics that can't be found in the standard tool. It still inherits all the base characteristics of a standard tool. *The TestTool will not be of actual use, but is only provided as help for future extensions.*

To keep it simple our *TestTool* will just have two additional attributes:

- The attribute *Power* represent the power consumption of the tool
- The attribute *Color* is one of three available colors: red, green, blue.

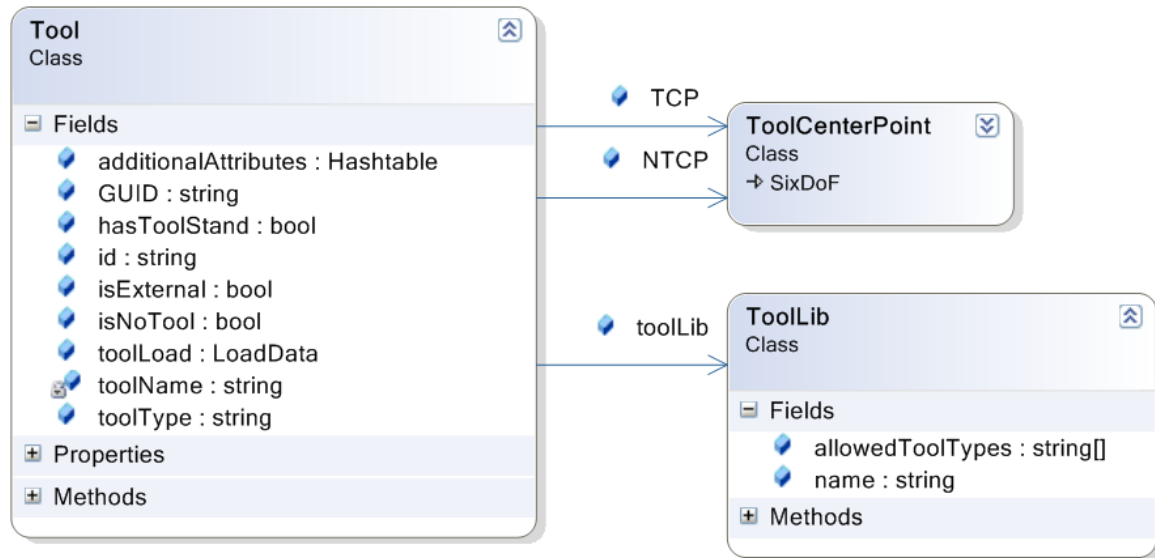
This will be implemented in a number of steps. The order of executing these steps are not absolute, though they have been found to enable fast addition of new functionality during the development process.

### STEP 1 - Setup Data Model

The *Setup Data Model* provide classes to create objects representing all different parts of a robot controller setup. This class library is the core of the software.

The type *Tool* is used to represent a tool in the setup. Its Class Diagram looks like this:





We now want to create the class `TestTool` as a child to this class:

- Open *SeputDataModel.cs*
- Expand the region *Special Tools*
- Insert the following code:

```

public class TestTool : Tool // Inherit from Tool
{
    public int power;        // New field
    public string color;     // New field

    public TestTool()
        : base() // Parent constructor: Tool()
    {
        power = null;       // Default value
        color = "red";      // Default value
    }
}
  
```

In addition to the first constructor, we'd like to add a constructor for import of AutomationML data. The AML document parser in *VCC\_AMLEngine\_Extension* puts imported data in a Hashtable structure that provide a convenient way to fetch the correct data for each attribute. The Hashtable (called *valueTable* in the constructor) for *Tool* looks like this:

valueTable	Count = 9
["NTCP"]	{System.Collections.Hashtable}
["ToolType"]	"#Gun"
["ToolStand"]	"True"
["Load"]	{System.Collections.Hashtable}
["ToolLib"]	"SpotGun"
["TCP"]	{System.Collections.Hashtable}
["ToolID"]	"1"
["External"]	"False"
["ToolName"]	"Tool1"
Raw View	

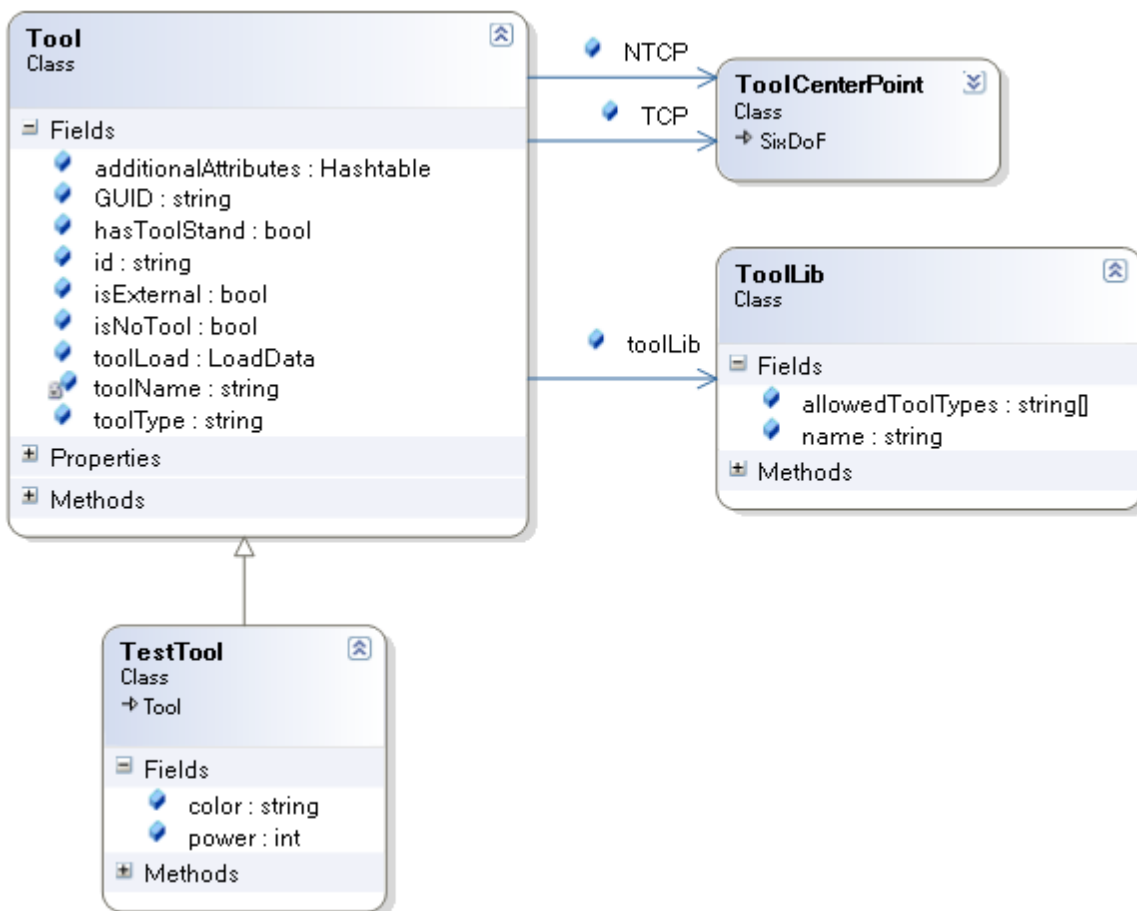
valueTable	Count = 1
["Attributes"]	{System.Collections.Hashtable}
Key	Value
Count = 6	
["X"]	"0.0"
["Y"]	"0.0"
["Z"]	"0.0"
["A"]	"0.0"
["B"]	"0.0"
["C"]	"0.0"
Raw View	

Some of the keys provide string values, while others hold other Hashtables. An example of a TCP hashtable is also shown. A constructor using the hashtable would then first be used by the *Tool* constructor to fetch values for the standard keys, and then by the Special Tool to fetch values for the special tool keys. The *TestTool* hashtable constructor thus becomes:

```
public TestTool(Hashtable valueTable)
    : base(valueTable, new String[2] { "Power", "Color" }) // Call parent constructor
    // and ignore these two values
{
    if (valueTable.Contains("Power")) // If attribute Power present
        power = Convert.ToInt16(valueTable["Power"]); // Cast value to int
    if (valueTable.Contains("Color")) // If attribute Color present
        color = (string)valueTable["Color"]; // Cast value to string
}
```

*NOTE: The string with attributes particular to the TestTool must be passed to the parent constructor. Otherwise it would treat them as Additional attributes (which are attributes not specifically defined in a Tool class).*

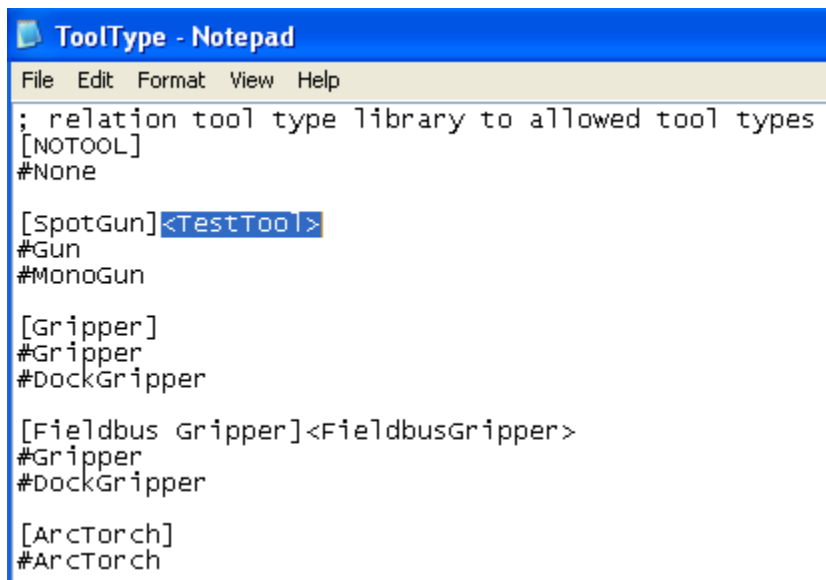
Here is the updated Class Diagram:



## STEP 2 - krl\_routines Connection

Since all tools available in the Setup Program reflects the *krl\_routines* library, we need a way to point out which tool library should be represented by our Special Tool. The tool libraries are connected to the different tool types as specified in the file *ToolType.txt*, which is located in *krl\_routines/ToolTypes*. By adding an extra tag "<SpecialTool>" to a ToolLib the Setup program will identify this ToolLib as a Special Tool. In this way it is even possible to assign the same Special Tool Class to more than one tool library.

For our example we associate the Special tool *TestTool* with the Tool library *SpotGun*:



```
ToolType - Notepad
File Edit Format View Help
; relation tool type library to allowed tool types
[NOTOOL]
#None

[SpotGun]<TestTool>
#Gun
#MonoGun

[Gripper]
#Gripper
#DockGripper

[Fieldbus Gripper]<FieldbusGripper>
#Gripper
#DockGripper

[ArcTorch]
#ArcTorch
```

## STEP 3 - GUI

The form for defining robot tools is found in the class library *ToolSelectionForm.cs*. The complete tool definition form is an instance of the class *ToolSelectionForm*. Each tool on this is of type *ToolForm*. This will for a standard tool occupy one row in the Tool selection form. To allow for additional form components for a SpecialTool the ToolForm can hold a *SpecialToolPanel*. This will be put on the row below the standard tool values for a Special Tool.

So - in order to enable the GUI form elements for our TestTool we need to create a new class, inheriting from *SpecialToolPanel*:

```

public class TestToolPanel : SpecialToolPanel // Inherit from SpecialToolPanel
{
    private TestTool attachedTestTool; // TestTool added in constructor
    private TextBox powerTextBox; // TextBox for input of Power attribute
    private ComboBox colorComboBox; // ComboBox for input of Color attribute

    public TestToolPanel(TestTool testTool)
    {
        this.attachedTestTool = testTool; // Associated TestTool to panel
        AddGUIComponents(); // Add GUI objects

        // Set values according to inputed tool
        if (testTool.power >= 0)
            powerTextBox.Text = testTool.power.ToString();
        if (testTool.color != null)
            colorComboBox.SelectedItem = testTool.color;
        else
            colorComboBox.SelectedIndex = 0;
    }

    private void AddGUIComponents()...

    public override Tool SaveToTool() // Override method in SpecialToolPanel
    {
        // to enable saving of Special Tool data
        // Update the attachedTestTool fields
        attachedTestTool.power = Convert.ToInt16(powerTextBox.Text);
        attachedTestTool.color = colorComboBox.SelectedItem.ToString();
        return attachedTestTool;
    }
}

```

---

The constructor creates the GUI, and if data is held in the inputed TestTool object, it is assigned to the corresponding form elements.

To be able to retrieve new input made by the user the method *SaveToTool* must be included. This updates the attached TestTool object and returns it.

The *AddGUIComponents* method is used to create the GUI and for our example it looks like this:

```

private void AddGUIComponents()
{
    // Create Control objects
    Label powerLabel = new Label();
    powerLabel.Text = "Power";
    powerLabel.AutoSize = true;
    Styles.MakeAttribute(powerLabel); // Add GraphicalStyles to label
    powerTextBox = new TextBox();
    Label colorLabel = new Label();
    colorLabel.Text = "Color";
    colorLabel.AutoSize = true;
    Styles.MakeAttribute(colorLabel);
    colorComboBox = new ComboBox();
    colorComboBox.AutoSize = true;
    String[] colorOptions = new String[3] { "red", "green", "blue" };
    colorComboBox.Items.AddRange(colorOptions);

    // Create a TableLayoutPanel
    TableLayoutPanel tlPanel = new TableLayoutPanel();
    tlPanel.AutoSize = true;
    tlPanel.Location = new Point(2, 2); // Position the TableLayoutPanel
    tlPanel.RowCount = 2;
    tlPanel.ColumnCount = 2;
    tlPanel.RowStyles.Add(new RowStyle(SizeType.AutoSize));
    tlPanel.RowStyles.Add(new RowStyle(SizeType.AutoSize));
    tlPanel.ColumnStyles.Add(new ColumnStyle(SizeType.AutoSize));
    tlPanel.ColumnStyles.Add(new ColumnStyle(SizeType.AutoSize));

    // Add control objects to TableLayoutPanel
    tlPanel.Controls.AddRange(new Control[] { powerLabel, powerTextBox, colorLabel, colorCom

    // Add TableLayoutPanel to this SpecialToolPanel
    this.Controls.Add(tlPanel);
}

```

One additional thing needs to be added for this *additionalPanel* to show up. In the class *ToolForm*, open region *ToolType/ToolLib variations*. Here you find a method *SetAdditionalPanel*. This method checks if the chosen tool is a Special Tool, and, in case that's true, creates an instance of its SpecialToolPanel to the *additionalPanel* field:

CONSTRUCTOR

TOOLTYPE/TOOLLIB VARIATIONS

ADDITIONAL ATTRIBUTES

SAVE TOOLFORM

EVENT LISTENERS



```
private void SetAdditionalPanel(Tool tool)
{
    if (tool is FieldbusGripper)
    {
        if (((FieldbusGripper)tool).isDefined)
            additionalPanel = new GripperConfigPanel((FieldbusGripper)tool);
        else
            additionalPanel = new GripperConfigPanel(new FieldbusGripper());
    }
    else if (tool is TestTool)
        additionalPanel = new TestToolPanel((TestTool)tool);
    else
        additionalPanel = null;
}
```

Here is added an if-statement to check whether the tool is of type TestTool.

The GUI part is now finished, and can be viewed and tested by running the software. The new additional panel will show up when choosing the SpotGun toollib, as specified in STEP 2. This is our result:

ToolID	ToolName	External	ToolLib
1	SpotGun1	FALSE	SpotGun

**Power** 1600

**Color** green

## STEP 4 - AML output

We dealt with the AML input interface in STEP 1. In this step we will add code to enable generation of TestTool data to an AML document.

- Open *AML\_Interface.cs*
- Expand the region *Create AutomationML documents from SetupDataModel objects*

- Expand method *AML\_From\_SpecialTool*

In this method add a new if-statement for our TestTool:

```
private static void AML_From_SpecialTool(Tool tool, InternalElementType internalElement)
{
    if (tool is FieldbusGripper)
        AML_From_FieldbusGripper((FieldbusGripper)tool, internalElement);
    else if(tool is TestTool)
        AML_From_TestTool((TestTool)tool, internalElement);
}
```

Here the internalElement represents the parent node in the AML structure. In this case it's the Tool node. Next create the method *AML\_From\_TestTool*, in which the new attributes are added to the Tool object:

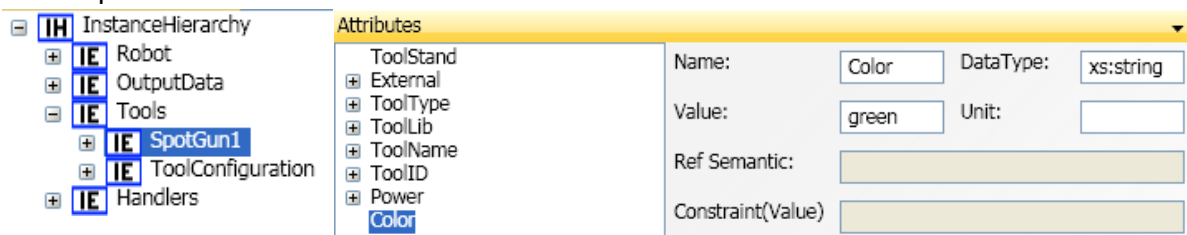
```
private static void AML_From_TestTool(TestTool tool, InternalElementType internalElement)
{
    internalElement.New_Attribute("Power", "xs:int", tool.power.ToString());
    internalElement.New_Attribute("Color", "xs:string", tool.color);
}
```

*NOTE: The VCC\_AMLEngine\_Extension allows for this simple creation of attributes (in which you directly set (Name, DataType, Value)).*

The resulting AML output for the tool is then:

```
<InternalElement Name="SpotGun1" RefBaseSystemUnitPath="SystemUnitClassLib/SpotGun" ID="{7cb99573-537d-476b-bb19-2ec5
  <Attribute Name="ToolStand" AttributeDataType="xs:boolean">
    <Value>True</Value>
  </Attribute>
  <Attribute Name="External" AttributeDataType="xs:boolean">
    <Value>False</Value>
  </Attribute>
  <Attribute Name="ToolType" AttributeDataType="xs:string">
    <Value>#Gun</Value>
  </Attribute>
  <Attribute Name="ToolLib" AttributeDataType="xs:string">
    <Value>SpotGun</Value>
  </Attribute>
  <Attribute Name="ToolName" AttributeDataType="xs:string">
    <Value>SpotGun1</Value>
  </Attribute>
  <Attribute Name="ToolID" AttributeDataType="xs:string">
    <Value>1</Value>
  </Attribute>
  <Attribute Name="Power" AttributeDataType="xs:int">
    <Value>1600</Value>
  </Attribute>
  <Attribute Name="Color" AttributeDataType="xs:string">
    <Value>green</Value>
  </Attribute>
  <SupportedRoleClass RefRoleClassPath="AutomationMLMIRoleClassLib/ManufacturingEquipment/Tool" />
  <RoleRequirements RefBaseRoleClassPath="AutomationMLMIRoleClassLib/ManufacturingEquipment/Tool" />
</InternalElement>
```

and as presented in the AutomationML Editor:



## STEP 5 - Setup Overview

The next step is to include the new data in the Setup Overview tab:

- Open *SetupOverviewForm.cs*
- Expand the region *Constructor* in the class *SetupOverviewForm*
- Expand the method *AddSpecialTool*
- Insert code for *SpecialTool*, as below:

```
private Panel AddSpecialTool(Tool tool)
{
    if (tool is FieldbusGripper)
        return ((FieldbusGripper)tool).ToPanel();
    else if (tool is TestTool)
        return ((TestTool)tool).ToPanel();
    else
        return null;
}
```

The method *ToPanel()* is not defined within the *TestTool* class. We instead add a method extension:

- Expand the region *Special Tools Data Overview* in the class *MethodExtensions*
- Add the following method extension:

```
public static Panel ToPanel(this TestTool testTool)
{
    Panel p = TemplatePanel(); // Panel used for SpecialTool values
    String s = "Power: " + testTool.power.ToString() + nl; // Add power + newline
    s += "Color: " + testTool.color; // Add color
    p.Controls[0].Text = s; // Add the string to the panel textbox
    return p; // Return this panel
}
```

**NOTE:** The method extension syntax is: *MethodName(this Type name)*. The method *MethodName* will then be available for objects of *Type*.

Now the new attributes are visible in the Setup Overview:



Tools	
ToolID	1
ToolName	SpotGun1
ToolLib	SpotGun
ToolType	#Gun
External	False
ToolStand	True
Additional attributes	
Special Tool attributes	Power: 1600 Color: green

## STEP 6 - Setup Generation

The final step is to adjust *SetupGeneration.cs*, so that the new additions correctly affects the outputted Setup description. Since this step might look quite different for new functionalities, just a few guidelines are here given.

When adjusting the Setup Generation to different SpecialTools you will probably find the method *CopyFilesFromToolTypeDevices* suitable. In this you can add calls to new methods that you design for the special tool you're using. The code snippet below shows how the special method *SetLToolRSequences* is called for the SpecialTool FieldbusGripper.

```
for(int j=0;j< toolTypeDirs.Length;j++)
{
    if (toolTypeDirs[j].Name == toolLibName)
    {
        WriteToolIDToToolTypeFile(toolTypeDirs[j].FullName + "\\LTool.src", targetPatl
        WriteToolIDToToolTypeFile(toolTypeDirs[j].FullName + "\\LTool.dat", targetPatl
        WriteToolIDToToolTypeFile(toolTypeDirs[j].FullName + "\\LToolR.src", targetPatl
        // Call specialized methods for treating SpecialTools
        if (setup.tools[i] is FieldbusGripper)
            SetLToolRSequences((FieldbusGripper) setup.tools[i]);
        else // Or call standard method
            WriteToolIDToToolTypeFile(toolTypeDirs[j].FullName + "\\LToolR.dat", targe
    }
}
```

If you would like to update the *SetUp.dat* output as well, this is done in the method *SpecialToolToSetupDat*.

## THAT'S IT!

If you would also like to make additions to the *SetUp.dat* and *ToolBase.ini*-file that is also possible through *SetupFileImport.cs*. This should however not be necessary when using the

AutomationML document for data storage.



## **B. Setup Description Generation Overview**

# VCC KUKA Robot Setup Assembly

## Overview

### Version I

This document is a description of how a Setup for a Kuka robot is assembled from the files in *krl\_routines* depending on the specification given in the *VCC KUKA Robot Setup Tool*. The procedure can be divided into a number of steps. The steps are connected to one or more methods in the class library *SetupGeneration*. These are here shown as *method*

## The steps

1. Create directory, backup content
2. Copy Default folder content
3. Initialize Setup.dat
4. General handler setup
5. ToolType files setup
6. Optional handler setup
7. Handler to Setup.dat
8. Adjust robot depending files
9. Divide ini-string content depending on chapters
10. Check Double IO
11. Parse new ini-strings
12. Assemble standard files
13. Delete backup folder

# VCC KUKA Robot Setup Assembly

## I. Create directory, backup content

*CopyAll(mainDirectory, backupDirectory)  
DeleteFolderContent(mainDirectory, true, true)*

- ▶ Choose a name for the new robot directory, e.g. a number between 0001 and 9999
- ▶ If necessary create new directory in *VolvoRobotLibrary/*
- ▶ A robot with number 4444 gets for example the directory *VolvoRobotLibrary/r4444/*
- ▶ If the directory exists, first backup its content into folder */backup*
- ▶ Delete all other content in output directory, in order to create the setup from scratch, not keeping unnecessary files

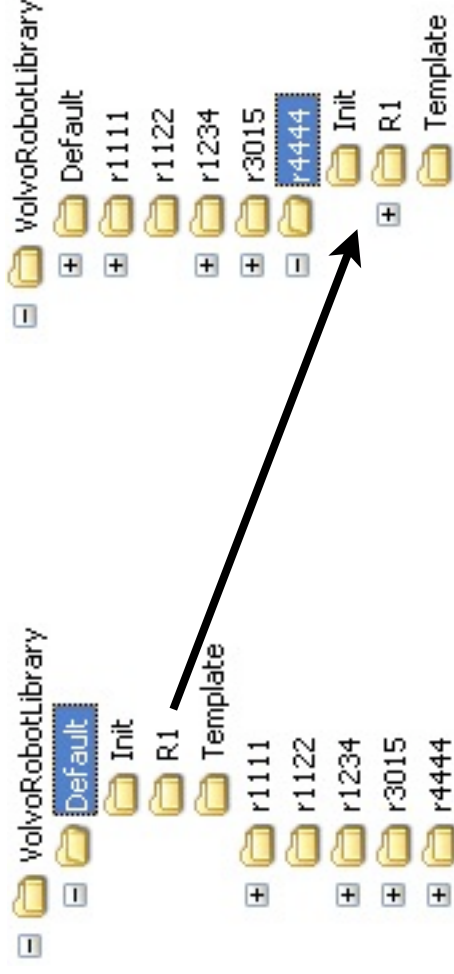


# VCC KUKA Robot Setup Assembly

## 2. Copy Default folder content

***CopyDefaultFolder()***

- Copy the content from the *VolvoRobotLibrary/Default/* folder to the new robot folder.





# VCC KUKA Robot Setup Assembly

## 3. Initialize Setup file

**InitializeSetupDat()**

**AddToolsToSetupDat()**

- Add first lines to *Setup.dat*:

```
DEFDAT Setup
;Volvo program structure Setup
;10:39:17
;2009-06-01
;c:\_Y352 KUKA Reuse\KUKA VSS\Y20\VolvoRobotLibrary\r4444
```

**Your path**

- followed by specified Tool configuration:

```
***TOOLS***
;[TOOL0]
;ToolLibrary:
;TType #None
;EXTERNAL FALSE
;ToolStand FALSE
;Tcp {X 0.0, Y 0.0, Z 0.0, A 0.0, B 0.0, C 0.0}
;
;TLOAD {M -1.0,CM {X 0.0,Y 0.0,Z 0.0,A 0.0,B 0.0,C 0.0}},J
{X 0.0,Y 0.0,Z 0.0}}
;[TOOL1]
;ToolLibrary: StudGun
;#StudGun
;External FALSE
;ToolStand FALSE
;Tcp {X 0.0, Y 0.0, Z 0.0, A 0.0, B 0.0, C 0.0}
;
;TLOAD {M -1.0,CM {X 0.0,Y 0.0,Z 0.0,A 0.0,B 0.0,C 0.0}},J
{X 0.0,Y 0.0,Z 0.0}}
```

**Values fetched from  
the Tool configuration  
input form**



# VCC KUKA Robot Setup Assembly

## 3. Initialize Setup file

AddPartAndLoadDataToSetUpDat()

AddToolConfigToSetupDat()

- Add Part and Load data into *Setup.dat*:

```
***PARTDATA***  
;ROB  
;PartRef #World  
;PartFrame {X 0.0, Y 0.0, Z 0.0, A 0.0, B 0.0, C 0.0}  
;PartLOAD {M -1.0,CM {X 0.0,Y 0.0,Z 0.0,A 0.0,B 0.0,C  
0.0},J {X 0.0,Y 0.0,Z 0.0}}  
;P28Carl  
;PartRef #World  
;PartFrame {X 0.0, Y 0.0, Z 0.0, A 0.0, B 0.0, C 0.0}  
;PartLOAD {M -1.0,CM {X 0.0,Y 0.0,Z 0.0,A 0.0,B 0.0,C  
0.0},J {X 0.0,Y 0.0,Z 0.0}}
```

- and

```
***LOADDATA***  
;GunLoad1  
;{M -1.0,CM {X 0.0,Y 0.0,Z 0.0,A 0.0,B 0.0,C 0.0},J {X  
0.0,Y 0.0,Z 0.0}}  
;GripperLoad1  
;{M -1.0,CM {X 0.0,Y 0.0,Z 0.0,A 0.0,B 0.0,C 0.0},J {X  
0.0,Y 0.0,Z 0.0}}  
;PartLoad1  
;{M -1.0,CM {X 0.0,Y 0.0,Z 0.0,A 0.0,B 0.0,C 0.0},J {X  
0.0,Y 0.0,Z 0.0}}
```

- and from Tool configuration form

```
;DefaultTool= 1  
;LastTool= 1  
;LastToolStand= 1
```

Values fetched from  
the Tool configuration  
input form

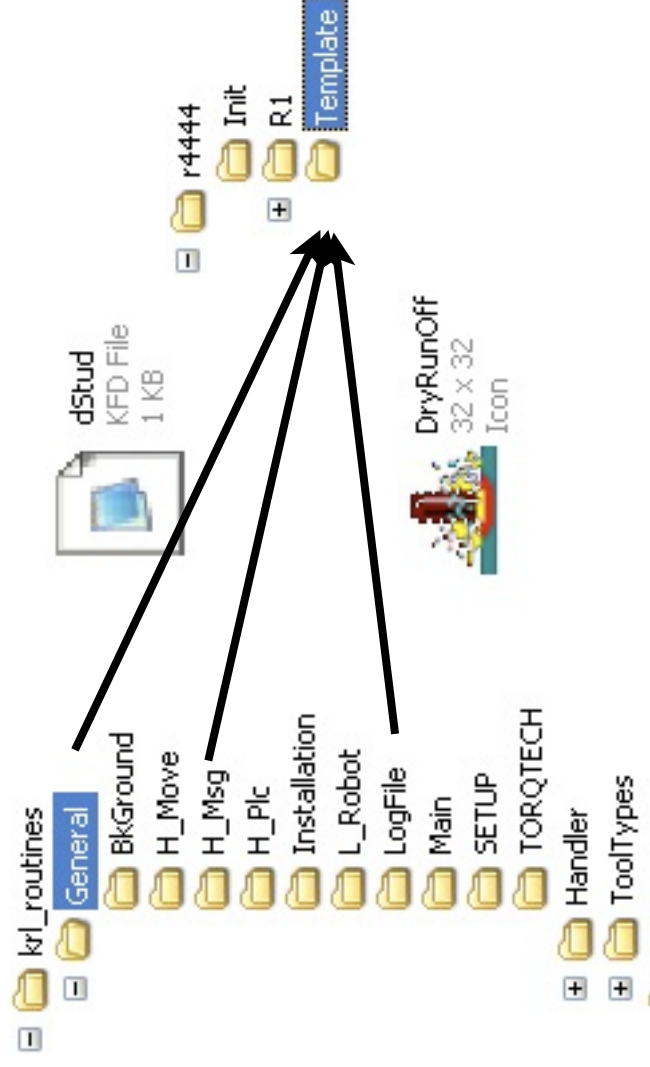
# VCC KUKA Robot Setup Assembly

## 4. General handler setup

`CopyFilesFromGeneralDirectory()`

`UpdateKfdFile()`

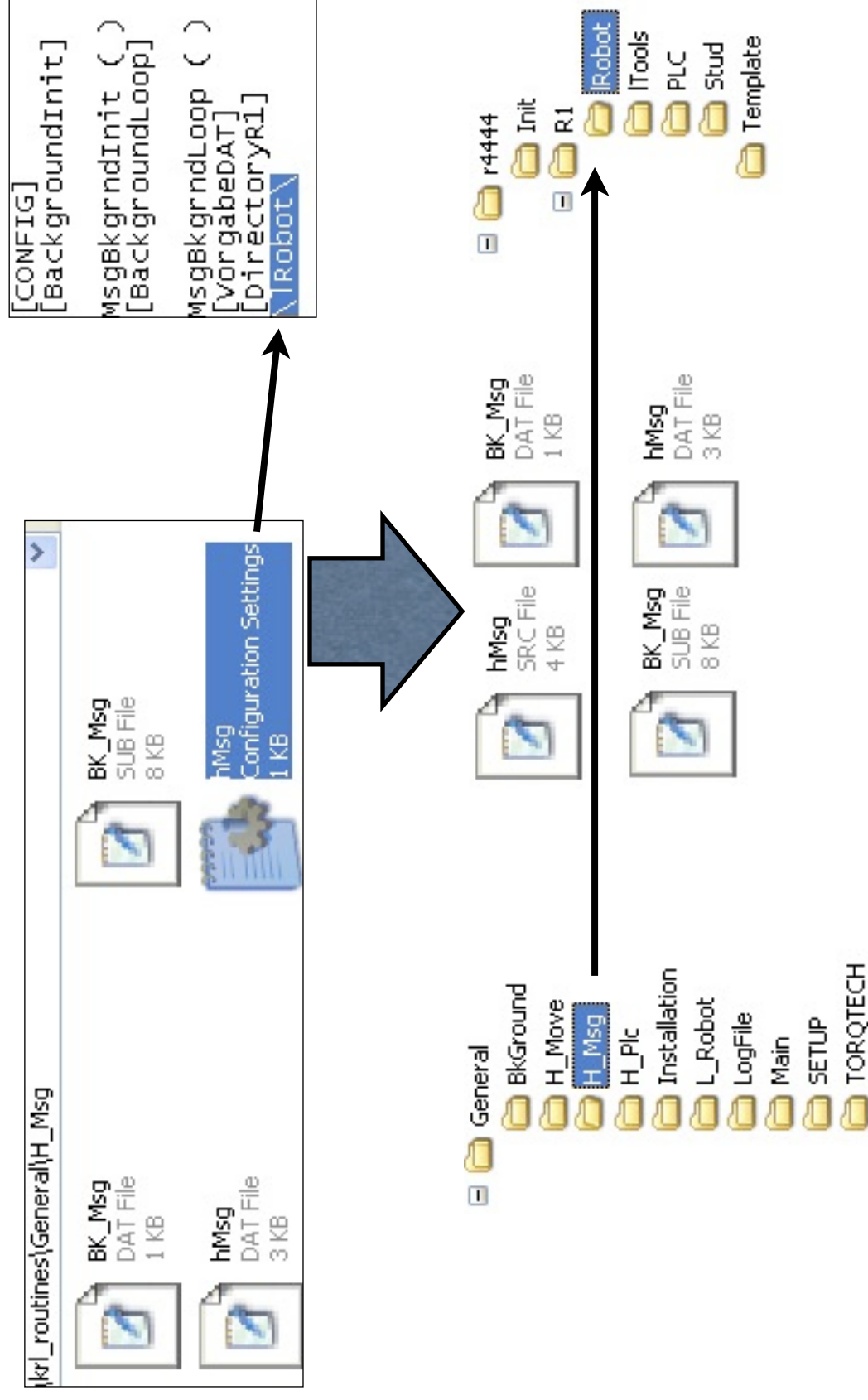
- ▶ Copy *.kfd*, *.ico* and *.bmp*-files from *krl\_routines/General/...* into *Template*
- ▶ Update *kfd*-files if needed



# VCC KUKA Robot Setup Assembly

## 4. General handler setup, cont.

- Copy *.src*, *.dat* and *.sub-files* from *krl\_routines/General/...* into directory given in General Handler ini-files under chapter *[DirectoryR1]*. e.g.:



- Also collect the content of all the ini-files in a common ini-string. This string is later on parsed (see 9), and its content then divided depending on chapters.



# VCC KUKA Robot Setup Assembly

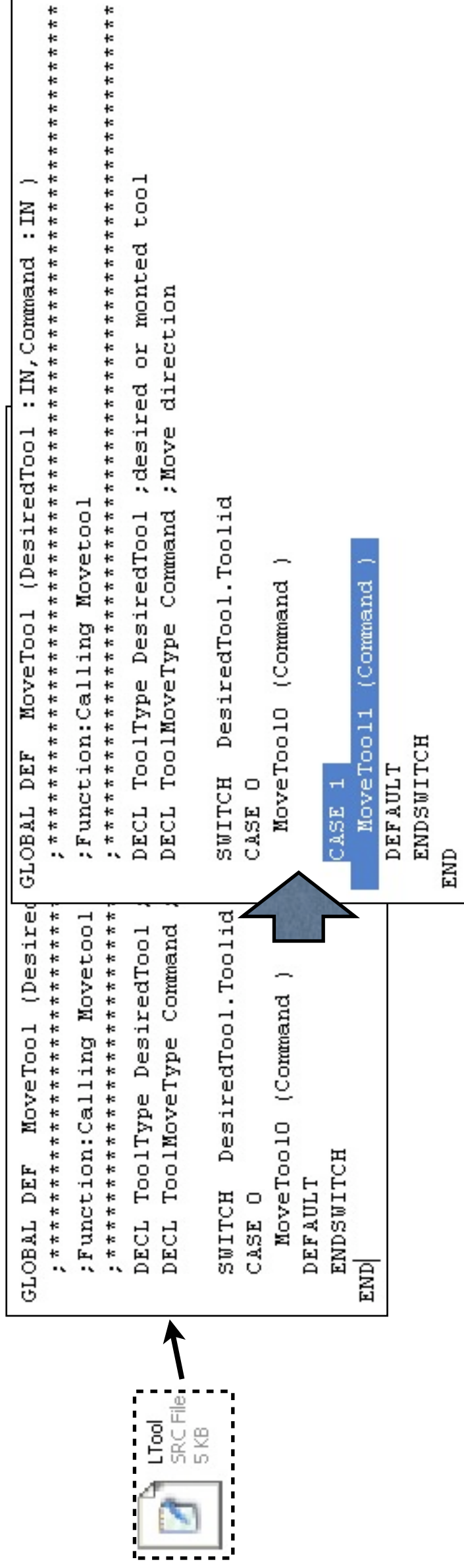
## 5. ToolType files setup

### AddForkFiles()

- Copy files *LTool.src* and *LTool.dat* from folder *ToolTypes/* to *R1/LTools/*



- In *LTool.src* add "cases" in every defined function for each tool in the setup

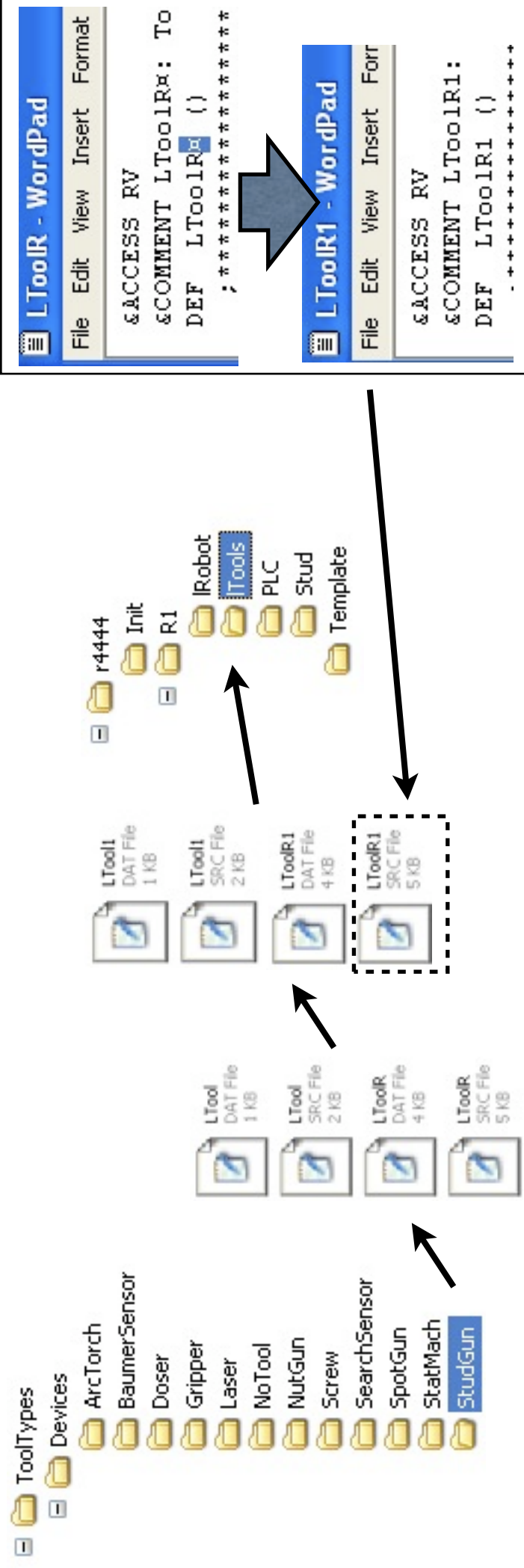


# VCC KUKA Robot Setup Assembly

## 5. ToolType files setup, cont.

### CopyFilesFromToolTypeDevices()

- For each tool used in the setup copy all *.src*, *.dat* and *.sub-files* from corresponding folder in *ToolTypes/Devices/* into *R1/LTools/*
  - Rename the files with the tool number. e.g. *LToolR2.dat*, *LTool3.src*
  - In the files, exchange all "sun"-symbols with the tool number



- For special tool files, perform certain adjustments (see method)



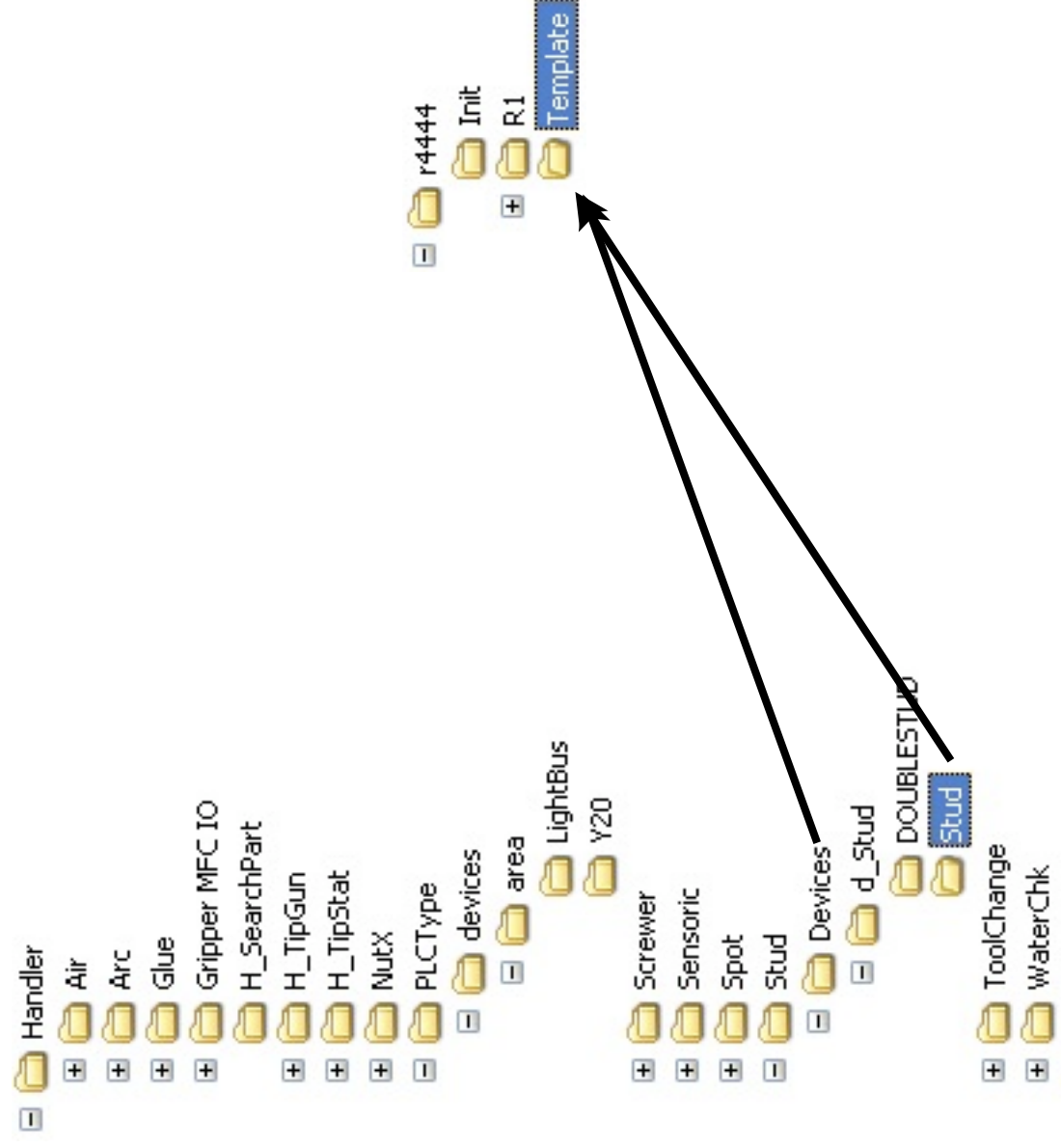
# VCC KUKA Robot Setup Assembly

## 6. Optional handler setup

`CopyFilesFromOptionalHandlerDirectory()`

`UpdateKfdFile()`

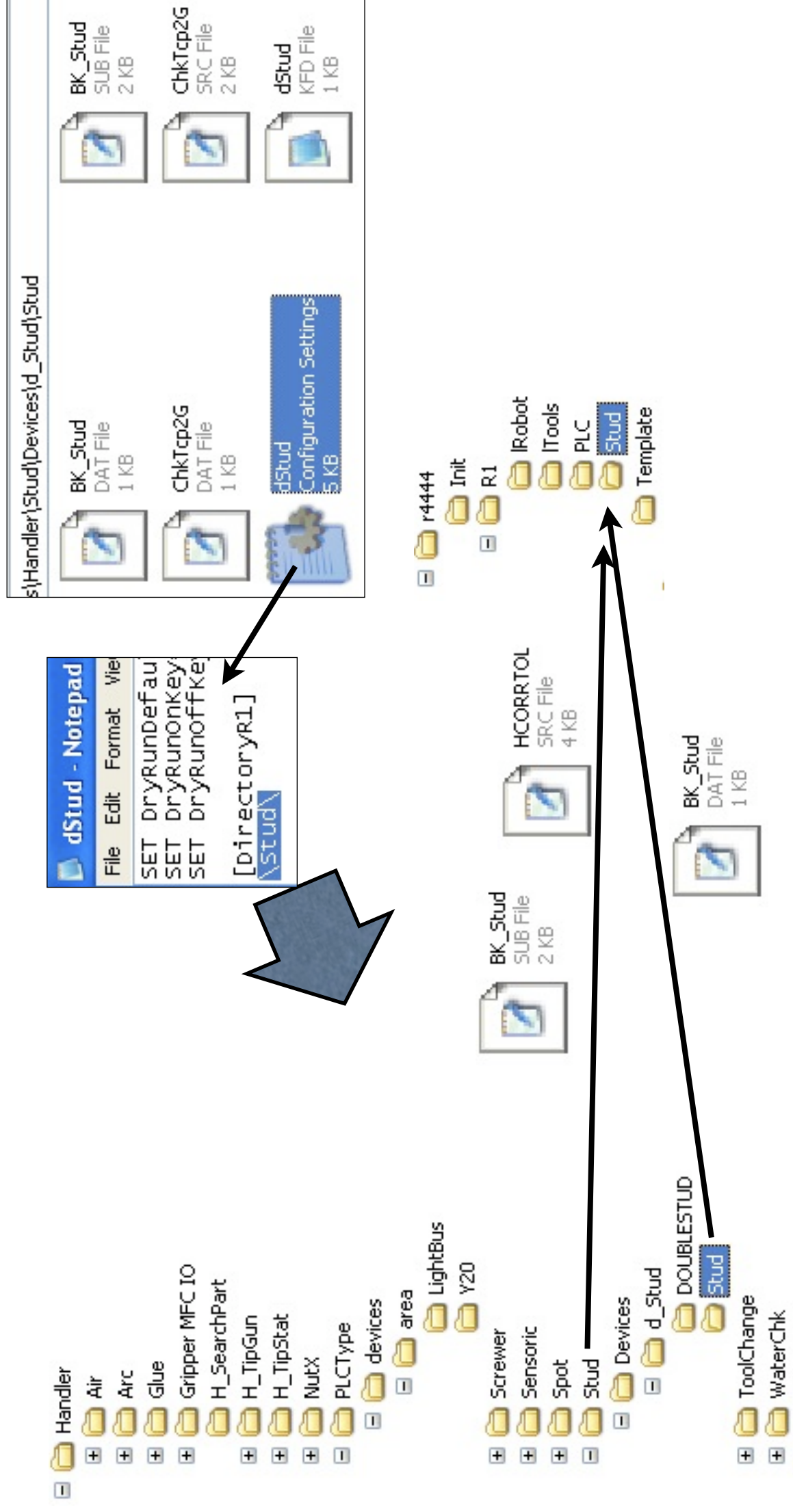
- ▶ Copy *.kfd*, *.ico* and *.bmp*-files from chosen handler folders, and device folders in *krl\_routines/Handler/...* into *Template*
- ▶ Update *kfd*-files if needed



# VCC KUKA Robot Setup Assembly

## 6. Optional handler setup, cont.

- Copy *.src*, *.dat* and *.sub-files* from chosen *Handler* and *devices*-folders into directory given in Handler ini-files under chapter *[DirectoryR1]*. e.g.:



- Also collect the content of all the ini-files in the common ini-string. This string is later on parsed (see 9), and its content then divided depending on chapters.

# VCC KUKA Robot Setup Assembly

## 7. Handler to setup file

AddGeneralHandlerToSetupDat()

AddOptionalHandlerToSetupDat()

- Add to *Setup.dat* the general handler folder names

```
***GENERAL HANDLER***  
; [BkGround]  
; [H_Move]  
; [H_Msg]  
; [H_PlC]  
; [Installation]  
; [LogFile]  
; [L_Robot]  
; [Main]  
; [SETUP]  
; [TORQTECH]
```



- Add to *Setup.dat* the chosen optional handlers

```
***OPTIONAL HANDLER***  
; [PLCType]  
; Y20  
; [Stud]  
; Stud
```

Fetches from  
Setup.Handlers

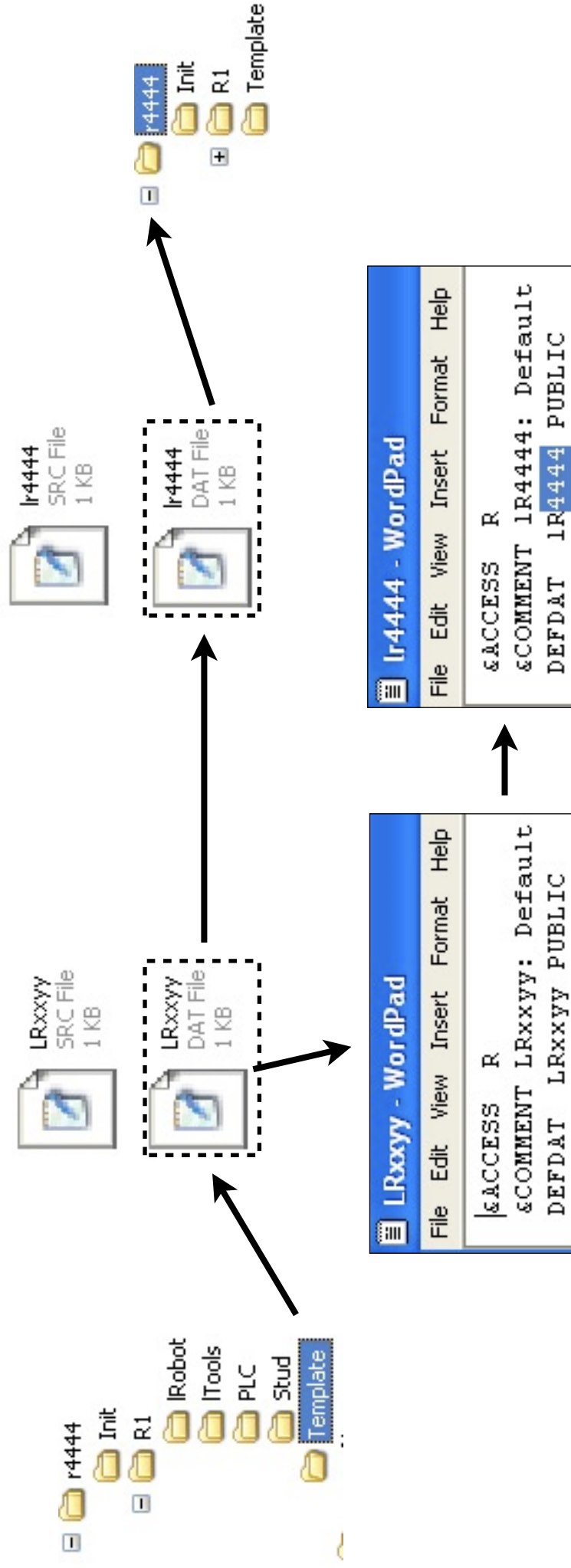


# VCC KUKA Robot Setup Assembly

## 8. Adjust robot depending files

### *CopyFilesFromTemplateFolder()*

- ▶ Copy *LRxxyy.src*, and *LRxxyy.dat* from chosen Template folder into rXXYY-folder.
- ▶ Rename the files according to robot number.
- ▶ Exchange "xxyy" with robot number in the files.



# VCC KUKA Robot Setup Assembly

## 9. Divide ini-string content according to chapters

**ParseIniString()**

- Parse the ini-string, which is assembled by ini-files from General and Optional Handler folders (see 5 and 7). The chapter headers in the ini-string tells where to put the information. An example:

**[CONFIG]**

```
;FOLD General Variables
DECL LOAD TLOAD
DECL LOAD PARTLOAD
; Variable decleration for approximation
DECL INT EXACT=0
DECL INT FINE=10
DECL INT Coarse=100
;ENDFOLD
```

Add this to a string called **strConfig**

**[INIT]**

```
MsgInit ( )
MovInitAll ( )
IniTool (ToolOnRobot ())
lRxxyy ( )
InitPlc ( )
```

Add this to a string called **strInit**

**[VolvoTechKfd]**

**{StatKeyScript}**

```
Decl STATKEYBAR TopStatKeybar
```

Add this to the subchapter **{StatKeyScript}** in a string called **strVolvoTechKfd**

**[CONFIG]**

**[BackgroundInit]**

**[BackgroundLoop]**

Add *nothing* to **strConfig**, **strBackgroundInit** and **strBackgroundLoop**

**[MenuKUKAIni]**

**{VOLVOKeys}**

```
VPTP= Ptp, 2010, INLINEFORM, KUKATPUSER;Mov;P
VLIN= Lin, 2010, INLINEFORM, KUKATPUSER;Mov;L
```

Add this to the subchapters **{VOLVOKeys}** and **{VOLVOBar}** respectively in a string

**{VOLVOBar}**

```
VolvoMoveextTech= VEXTPTP,VEXTLIN,VEXTCIRC,VExtCheckPos
```

called **strMenuKukalni**

# VCC KUKA Robot Setup Assembly

## 10. Check Double IO

CheckDoubleIO()

AddToolInfoToConfigDat()

- Parse the string *strConfig* (see step 9), and raise a warning for every double Input/Output signal that is found.

```
;FOLD Media Signals
SIGNAL I_AirPress $IN[4] ;Air Pressure Input
SIGNAL I_AirPres2nd $IN[6] ;2nd Air Pressure Input
;ENDFOLD

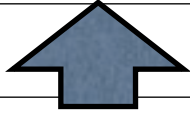
;FOLD Media Signals
; I/O Signals
SIGNAL I_WaterFlow $IN[5] ;Water Flow Input
SIGNAL I_WaterPrs $IN[6] ;Water Pressure Input
SIGNAL O_CoolWater $OUT[1] ;Water Cooling Output
;ENDFOLD
```

Raise a warning since \$IN[6] is present in both.  
Allow the double to be ignored.

- Add to *strConfig* the Tool, Part and Load definitions.

```
-----
; TOOL definition Rxyy
-----
DECL TOOLType NOTOOL={TOOLId 0, Tcp {X 0.0, Y 0.0, Z
0.0, A 0.0, B 0.0, C 0.0}, EXTERNAL FALSE,ToolStand
FALSE, TType #None, TLOAD {M -1.0,CM {X 0.0,Y 0.0,Z
0.0,A 0.0,B 0.0,C 0.0},J {X 0.0,Y 0.0,Z 0.0}}
DECL TOOLType TOOL1={TOOLId 1, Tcp {X 0.0, Y 0.0, Z
0.0, A 0.0, B 0.0, C 0.0}, EXTERNAL FALSE,ToolStand
FALSE, TType #Gun, TLOAD {M -1.0,CM {X 0.0,Y 0.0,Z
0.0,A 0.0,B 0.0,C 0.0},J {X 0.0,Y 0.0,Z 0.0}}
DECL TOOLType TOOL2={TOOLId 2, Tcp {X 0.0, Y 0.0, Z
0.0, A 0.0, B 0.0, C 0.0}, EXTERNAL FALSE,ToolStand
FALSE, TType #Gun, TLOAD {M -1.0,CM {X 0.0,Y 0.0,Z
0.0,A 0.0,B 0.0,C 0.0},J {X 0.0,Y 0.0,Z 0.0}}
DECL TOOLType TOOL3={TOOLId 3, Tcp {X 0.0, Y 0.0, Z
0.0, A 0.0, B 0.0, C 0.0}, EXTERNAL FALSE,ToolStand
FALSE, TType #Gun, TLOAD {M -1.0,CM {X 0.0,Y 0.0,Z
0.0,A 0.0,B 0.0,C 0.0},J {X 0.0,Y 0.0,Z 0.0}}

```



```
-----
; TOOL definition Rxyy
-----
DECL TOOLType NOTOOL={TOOLId 0, Tcp {X 0.0, Y 0.0, Z
0.0, A 0.0, B 0.0, C 0.0}, EXTERNAL FALSE,ToolStand
FALSE, TType #None, TLOAD {M -1.0,CM {X 0.0,Y 0.0,Z
0.0,A 0.0,B 0.0,C 0.0},J {X 0.0,Y 0.0,Z 0.0}}
DECL TOOLType TOOL1={TOOLId 1, Tcp {X 0.0, Y 0.0, Z
0.0, A 0.0, B 0.0, C 0.0}, External FALSE,ToolStand
FALSE, TType #StudGun, TLOAD {M -1.0,CM {X 0.0,Y
0.0,Z 0.0,A 0.0,B 0.0,C 0.0},J {X 0.0,Y 0.0,Z 0.0}}
DECL TOOLType TOOL2={TOOLId 2, Tcp {X 0.0, Y 0.0, Z
0.0, A 0.0, B 0.0, C 0.0}, EXTERNAL FALSE,ToolStand
FALSE, TType #Undefined, TLOAD {M -1.0,CM {X 0.0,Y
0.0,Z 0.0,A 0.0,B 0.0,C 0.0},J {X 0.0,Y 0.0,Z 0.0}}
DECL TOOLType TOOL3={TOOLId 3, Tcp {X 0.0, Y 0.0, Z
0.0, A 0.0, B 0.0, C 0.0}, EXTERNAL FALSE,ToolStand
FALSE, TType #Undefined, TLOAD {M -1.0,CM {X 0.0,Y
0.0,Z 0.0,A 0.0,B 0.0,C 0.0},J {X 0.0,Y 0.0,Z 0.0}}

```

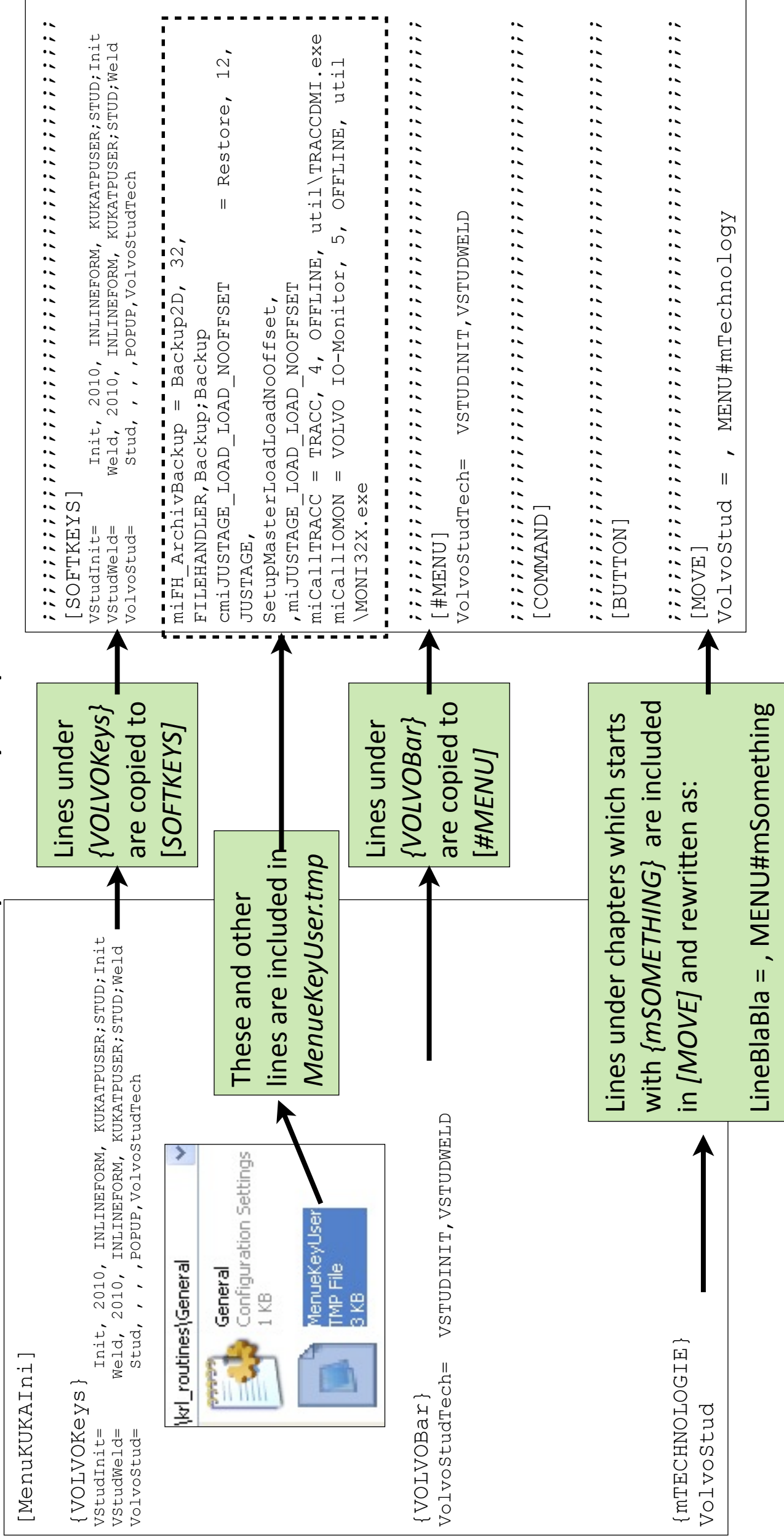


# VCC KUKA Robot Setup Assembly

## II. Parse new ini-strings

### ParseMenueKeyUser()

- ▶ To create the *MenueKeyUser.ini*, which is a setup document for the teach pendant, parse *strMenuKukaIni* (created in step 9) into the template file *MenueKeyUser.tmp* found in the *General* directory. The different chapters are handled a little bit differently, but they map to the final file like this:



# VCC KUKA Robot Setup Assembly

## 11. Parse new ini-strings, cont.

### ParseVolvoTechKfd()

- ▶ Similarly to *strMenuKukalni*, divide *strVolvoTechKfd* into the chapters:
  - ▶ {STATKEYSCRIPT}
  - ▶ {STATKEYBAR}\*\*
  - ▶ {STATKEYDECL}\*
  - ▶ {STATKEYSET}\*
    - \* Just copy the rows under this into strings *strScript*, *strDeclare* and *strSet*
    - \*\* Create with each row under this string *strBar*:  
*SET TopStatKeybar={;STATKEY[1] row1, STATKEY[2] row2....}*
- ▶ Create the new *strVolvoTechKfd* like this:  
*strVolvoTechKfd = strScript + strDecl + strBar + strSet;*

# VCC KUKA Robot Setup Assembly

## I2. Assemble standard files

### AssembleStandardFiles()

- ▶ With the strings created in step 9, and the adjusted *strConfig* from step 10, add start and finish statement and create the following files:

```
"DEFDAT $CONFIG PUBLIC"  
strConfig  
"ENDDAT"
```

into R1\config.dat

```
"DEF BkGdInit () "  
strBackGroundInit  
"END"
```

into R1\IRobot\BkGdInit.sub

```
"DEF BkGdLoop () "  
strBackGroundLoop  
"END"  
into R1\IRobot\BkGdLoop.sub
```

Within the ini-string (*strIni*) 'IRxxyy' is changed to 'IR4444' etc

```
"DEF INIT () "  
strInit  
"END"  
into R1\IRobot\init.src
```

```
"DEFDAT Setup"  
strSetup  
"ENDDAT"  
into SetUp.dat
```

This for example  
results in the final  
file ***BkGdLoop.sub***

```
BkGdLoop - WordPad  
File Edit View Insert Format  
DEF BkGdLoop ()  
MsgBkgrndLoop ()  
PlcBkGrndLoop ()  
StudBkGrndLoop ()  
END
```

And this results in  
***init.src***

```
init - WordPad  
File Edit View Insert Format Help  
DEF INIT ()  
MsgInit ()  
MovInitAll ()  
IniTool (ToolOnRobot ())  
IR4444 ()  
InitPlc ()  
StudInit (#None)  
END
```



# VCC KUKA Robot Setup Assembly

## I3. Delete backup folder

*DeleteFolder(backupDirectory, false, false)*

- ▶ If all steps in this procedure has been correctly executed the folder *backup* is deleted.
- ▶ If problems have occurred the setup generation will abort. The backup folder can than be used to restore the old setup.