# DaVinci Developer Adaptive Automation-Interface

Development Documentation of the AutomationInterface (AI)

| | |
|---|---|
| Authors | DaVinci Developer Adaptive Team |

# Contents

# 1. Introduction

## 1.1 General

The users of the DaVinci Developer Adaptive (DvDevA) can create scripts, which will be executed inside the product:

- Create projects
- Manipulate the data model with an access to the whole AUTOSAR model
- Get validation results & generate code
- Executed repetitive tasks with code, without user interaction
- Undo/Redo support
- More

The DvDevA consists of an Eclipse Plugin for development support and a command line tool for non GUI usage. Automation scripts can be executed in the UI and CLI.

The scripts are written by the *user* with the DvDevA AutomationInterface.

## 1.2 Facts

**Installation** The DvDevA can execute customer defined scripts out of the box. No additional scripting language installation is required by the customer.

**Languages** The scripts are written in Groovy or Java. See 3.2 on page 17 for details.

**Documentation** The AutomationInterface provides a comprehensive documentation:

- This document
- Javadoc HTML pages as class reference
- Script samples and templates
- API documentation inside of an IDE

**Code Completion** When creating a script project, code completion for Groovy and Java will be offered. To use the code completion it is necessary to install the IntelliJ IDEA.[1]

[1]See chapter 7 on page 144 for details.

# 2. Getting started with Script Development

## 2.1 General

This chapter gives a short introduction of how to get started with script file or script project creation.

## 2.2 Automation Script Development Types

The DvDevA supports two types of automation scripts:

- Script files (`.dvgroovy` files)
- Script projects (`.jar` files)

**Script File** The script file provides the **simplest way** to implement an automation script. The preferred way should always be to create a script project 2.4.

To create a script file proceed with chapter 2.5 on page 15.

**Script Project** The script project provides IDE support for:

- Code completion
- Syntax highlighting
- API Documentation
- Debug support
- Build support

It is the **recommended way to develop** scripts, containing more tasks or multiple classes.

To create a script project proceed with chapter 2.4.

## 2.3 Script Samples

In the folder **AutomationInterface/_doc/templates/ScriptSamples**, you may find *.dvgroovy files, which can be used as templates for different use cases, e.g., creating a project, creating elements, and reading and writing model data. In 2.4 on the next page shows you how to create a script project step by step.

## 2.4 Script Project

The script project is the **preferred** way to develop an automation script, if the content is more than one simple task.

A script project is a normal IDE project (IntelliJ IDEA recommended), with compile bindings to the DvDevA AutomationInterface. It is also called "Automation Script Project" throughout this document.

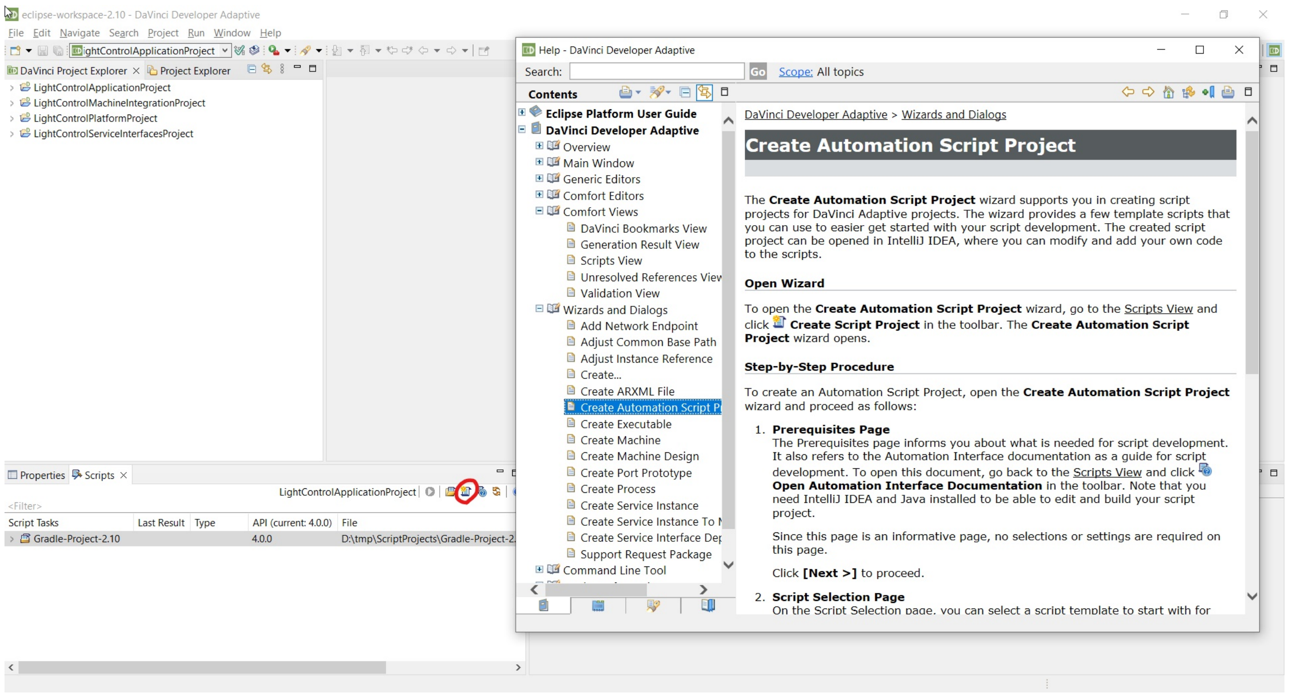
The DvDevA will load a script project as a single `.jar` file. So the script project must be built and packaged into a `.jar` file before it can be executed by the DvDevA.

**Prerequisites** Before you start, **please make sure** that the following items are available on your system:

- **Java JDK**: For the development with the IntelliJ IDEA a "Java SE Development Kit 17" (JDK 17) is required[1].
  Please install the JDK 17 as described in chapter 2.4.2 on page 13.
- **IDE**: For the script project development the recommended IDE is IntelliJ IDEA[2].
  Please install IntelliJ IDEA as described in chapter 2.4.3 on page 13.
- **Build system**: To build the script project the build system Gradle is required. See chapter 2.4.4 on page 14 for installation instructions.

Note: The Adaptive AUTOSAR Standard is currently in a stabilization phase, which means the standard is not yet finalized. Because our model APIs gets directly generated from the Adaptive AUTOSAR Standard, the model APIs are marked as beta. Means, to use those APIs it is necessary to activate the beta API usage 7.9.3.5 on page 160.

**Project Creation By Wizard** Open the help in DvDevA and you can find a section called *Create Automation Script Project*, which helps you create script project step by step.


eclipse-workspace-2.10 - DaVinci Developer Adaptive
Help - DaVinci Developer Adaptive
Search:
Scope: All topics
Contents
Eclipse Platform User Guide
DaVinci Developer Adaptive
Wizards and Dialogs
Create Automation Script Project
DaVinci Project Explorer
Scripts

DaVinci Developer Adaptive > Wizards and Dialogs

**Create Automation Script Project**

The **Create Automation Script Project** wizard supports you in creating script projects for DaVinci Adaptive projects. The wizard provides a few template scripts that you can use to easier get started with your script development. The created script project can be opened in IntelliJ IDEA, where you can modify and add your own code to the scripts.

**Open Wizard**

To open the **Create Automation Script Project** wizard, go to the Scripts View and click **Create Script Project** in the toolbar. The **Create Automation Script Project** wizard opens.

**Step-by-Step Procedure**

To create an Automation Script Project, open the **Create Automation Script Project** wizard and proceed as follows:

1. **Prerequisites Page**
   The Prerequisites page informs you about what is needed for script development. It also refers to the Automation Interface documentation as a guide for script development. To open this document, go back to the Scripts View and click **Open Automation Interface Documentation** in the toolbar. Note that you need IntelliJ IDEA and Java installed to be able to edit and build your script project.

   Since this page is an informative page, no selections or settings are required on this page.

   Click **[Next >]** to proceed.
2. **Script Selection Page**
   On the Script Selection page, you can select a script template to start with for

Figure 2.1: Create Automation Script Project Wizard

[1] `https://www.oracle.com/java/technologies/downloads/` [2023-01-04]

[2] `https://www.jetbrains.com/idea/download/` [2023-01-04]

**Project Creation Manually** Use the project template to create an automation script project. The template for the script project can be found 6.1 on page 142. Copy it to the directory of your choice.

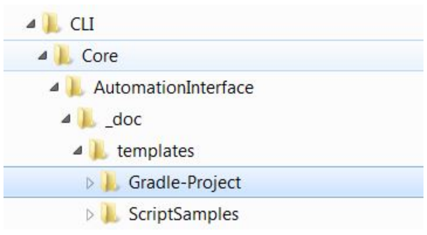


Figure 2.2: Script Project Template

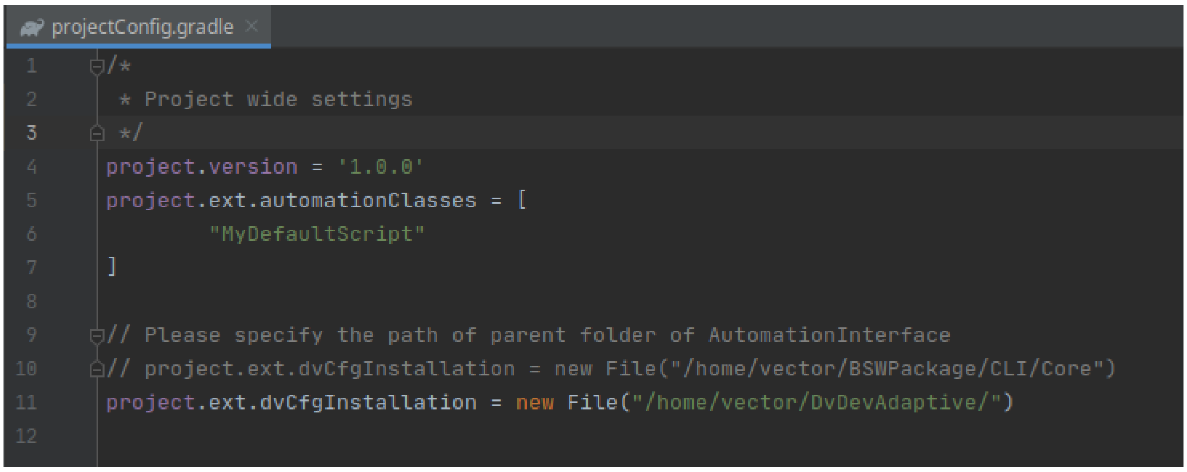
The project creation will be done via Gradle. To create a full working automation script project there are a few steps to be done. The following steps needs to be done inside the template script folder structure.

- **Navigate to projectConfig.gradle file**
  - Adjust installation Path 2.3 on the next page
- **Open the command line**
  - Type in *gradlew tasks -Dorg.gradle.java.home="<PATH_TO_JDK_17>"*[3] - This will download and install Gradle
  - Type in *gradlew idea -Dorg.gradle.java.home="<PATH_TO_JDK_17>"* - This will create the script project 2.4 on the following page
- **Start the IDE**
  - Opening the .ipr file will start IntelliJ and load the script project

Optional

- **Gradle Distribution URL**
  - File: /gradle/wrapper/gradle-wrapper.properties
  - **Gradle Default**
    - This will download the required Gradle build system from Gradle Inc. directly. To use this option you need **internet access**.
  - **Custom URL**
    - Specify an URL to your own Gradle distribution.
      To setup your own Gradle build system see 2.4.4 on page 14.

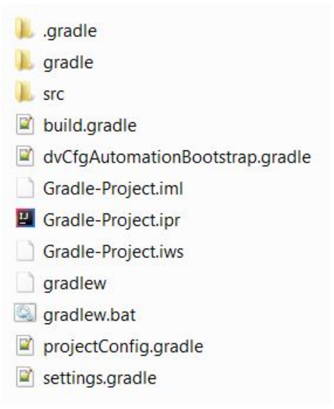
[3]Argument `-Dorg.gradle.java.home` is always needed when executing `gradlew` if you do NOT have Java 17 as the default Java version. If you have Java 17 as the default Java, you can skip this argument.



projectConfig.gradle

```
/*
 * Project wide settings
 */
project.version = '1.0.0'
project.ext.automationClasses = [
        "MyDefaultScript"
]

// Please specify the path of parent folder of AutomationInterface
// project.ext.dvCfgInstallation = new File("/home/vector/BSWPackage/CLI/Core")
project.ext.dvCfgInstallation = new File("/home/vector/DvDevAdaptive/")
```

Figure 2.3: Installation Path



```
.gradle
gradle
src
build.gradle
dvCfgAutomationBootstrap.gradle
Gradle-Project.iml
Gradle-Project.ipr
Gradle-Project.iws
gradlew
gradlew.bat
projectConfig.gradle
settings.gradle
```

Figure 2.4: Generated Script Project

You can now modify the implementation according to your needs. For the AutomationInterface API Reference see chapter 4 on page 20. To edit and rebuild the project use IntelliJ IDEA.

After each build copy the generated jar file into your DvDevA project script folder.

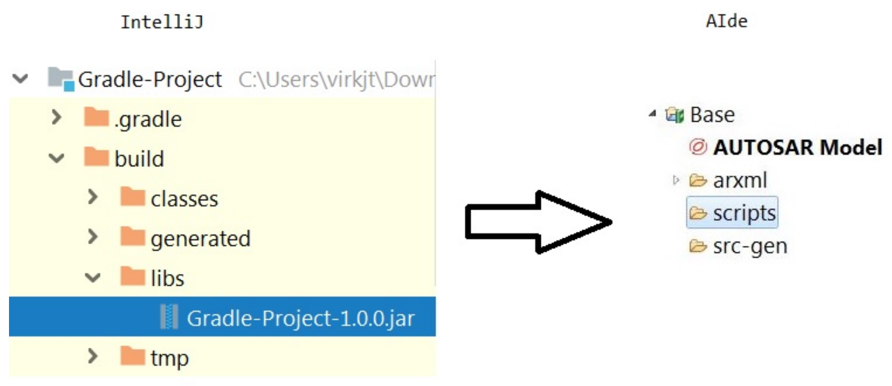


Figure 2.5: Generated Script Project

**IntelliJ IDEA Usage** Ensure that the Gradle JVM and the Project SDK are set in the IntelliJ IDEA Settings. For details see 2.4.3.

Having modified and saved `MyScript.groovy` in the IntelliJ IDEA editor you can build the project by pressing the **Run Button provided in the toolbar**. The functionality of this Run Button is determined by the option selected in the Menu beneath this button. In this menu **<Project-Name> `build` shall be selected**.

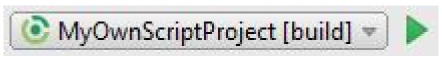


Figure 2.6: Project Build

For more information to IntelliJ IDEA usage please see chapter 7.5 on page 145. If you have trouble with IntelliJ, see 7.5.5 on page 148.

**DaVinci Developer Adaptive views** The View **Script Tasks** provides an overview over the scripts and tasks contained in the project. See 7.12 on page 150.

### 2.4.1 Script Project Development

For more details to the development of a script project see chapter 7 on page 144.

### 2.4.2 Java JDK Setup

Install a JDK 17 on your system. The Java JDK website provides download versions for different systems. Download an appropriate version.

The DvDevA only supports 64 Bits, so make sure you get the x64 version.

The JDK is needed for the Java Compiler for IntelliJ IDEA and Gradle.

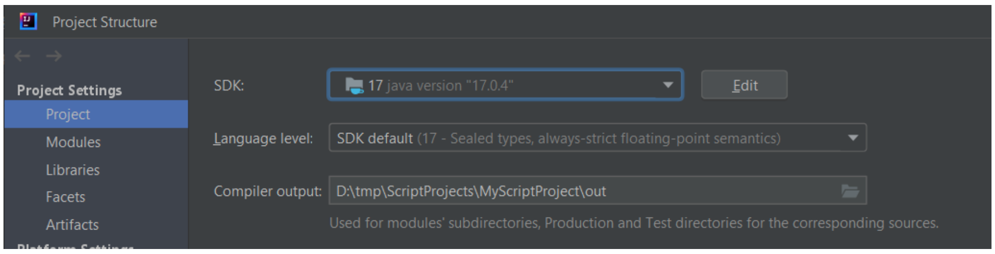
### 2.4.3 IntelliJ IDEA Setup

Install IntelliJ IDEA on your system. The IntelliJ IDEA website provides download versions for different applications. Download a version that supports Java and Groovy and that is in the list of supported versions (see list 7.5.1 on page 145).

Code completion and compilation additionally require that the Project SDK is set. Therefore open the File -> **Project Structure** Dialog in IntelliJ IDEA and switch to the settings dialog for

**Project**. If not already available set an appropriate option for the **Project SDK**. Please set the value to a valid Java JDK (see 2.4.2 on the previous page).

**NOTE:** Do not select a JRE.



SDK: 17 java version "17.0.4"

Language level: SDK default (17 - Sealed types, always-strict floating-point semantics)

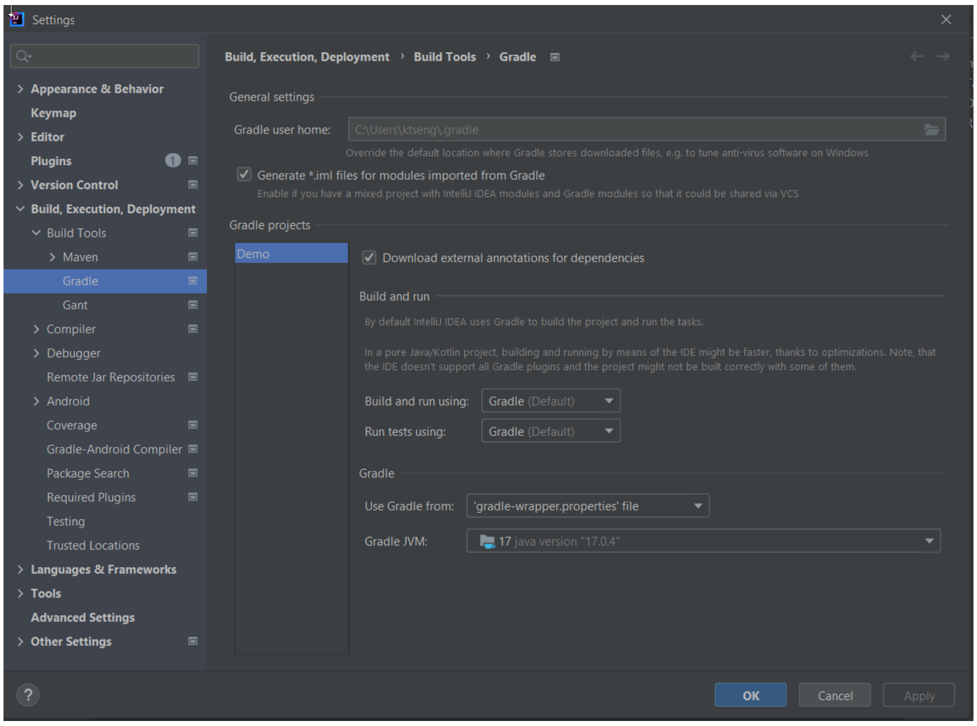
Compiler output: D:\tmp\ScriptProjects\MyScriptProject\out

Used for modules' subdirectories, Production and Test directories for the corresponding sources.

Figure 2.7: Project SDK Setting

To enable building of projects ensure that the Gradle JVM is set. Therefore open the File -> **Settings** Dialog in IntelliJ IDEA and find the settings dialog for **Gradle**. If not already available set an appropriate option for the **Gradle JVM**. Please set the value to the same Java JDK as the Project SDK above. **Do not** select a JRE.

If you do not have the Gradle settings, please make sure that the Gradle plugin inside of IntelliJ IDEA is installed. Open the File -> **Settings** Dialog then Plugins and select the Gradle plugin.



General settings

Gradle user home: C:\Users\ktseng\.gradle

Override the default location where Gradle stores downloaded files, e.g. to tune anti-virus software on Windows

[✓] Generate *.iml files for modules imported from Gradle

Enable if you have a mixed project with IntelliJ IDEA modules and Gradle modules so that it could be shared via VCS

Gradle projects

Demo

[✓] Download external annotations for dependencies

Build and run

By default IntelliJ IDEA uses Gradle to build the project and run the tasks.

In a pure Java/Kotlin project, building and running by means of the IDE might be faster, thanks to optimizations. Note, that the IDE doesn't support all Gradle plugins and the project might not be built correctly with some of them.

Build and run using: Gradle (Default)

Run tests using: Gradle (Default)

Gradle

Use Gradle from: 'gradle-wrapper.properties' file

Gradle JVM: 17 java version "17.0.4"

Figure 2.8: Gradle JVM Setting

### 2.4.4 Gradle Setup

If your system has internet access you can use the default Gradle Build System provided by the DvDevA. In this case you **do not** have to install Gradle.

If you want to use your own Gradle Build System install it on your system. The Gradle website provides the required download version for the Gradle Build System. Please **download the version 7.5.1**. See chapter 7.9 on page 152 for more details to the Build System.

## 2.5 Script File

The script file is the simplest way to implement an automation script. It could be sufficient for **small** tasks and if the developer does not need support by the tool during implementing the script and if debugging is not required.

**Creation** Inside the AutomationInterface folder you find examples of Automation script files see screenshot 2.9. Copy the example, e.g. `...ScriptSamples/SimpleScriptAdaptive.dvgroovy` to your project script folder inside DvDevA, see screenshot 2.10.

Rename the script file and open it in any text editor. In case of `SimpleScriptAdaptive.dvgroovy` it consists of several tasks. One of the tasks will print a "HelloApplication" string to the console.

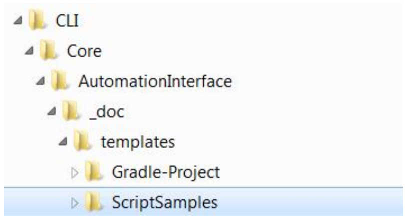


Figure 2.9: Script Samples location

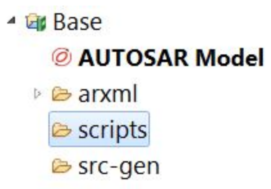
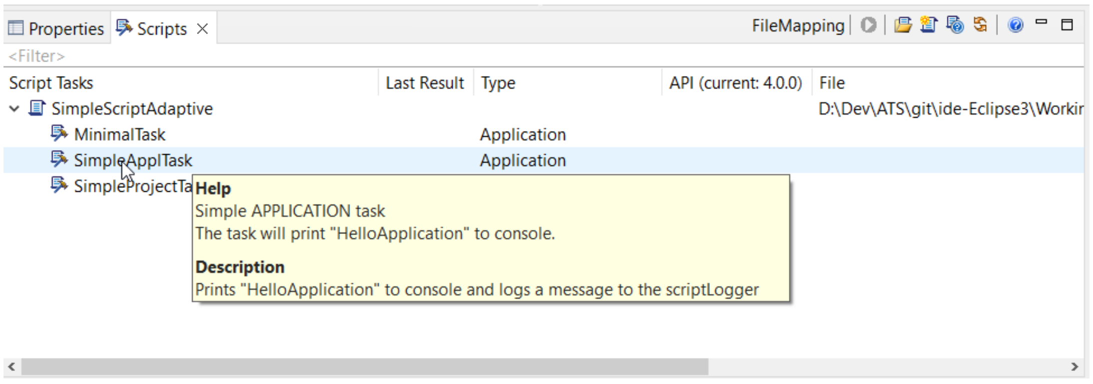


Figure 2.10: Project Script Folder

Open the DvDevA and if not yet visible open the view "Scripts" via the Window -> Show View menu.

Switch to the **Scripts** View. It provides an overview over the tasks contained in your script.



| Script Tasks | Last Result | Type | API (current: 4.0.0) | File |
|---|---|---|---|---|
| SimpleScriptAdaptive | | | | D:\Dev\ATS\git\ide-Eclipse3\Workir |
| MinimalTask | | Application | | |
| SimpleAppITask | | Application | | |
| SimpleProjectTa | | | | |

**Help**
Simple APPLICATION task
The task will print "HelloApplication" to console.

**Description**
Prints "HelloApplication" to console and logs a message to the scriptLogger

Figure 2.11: Scripts View

**Execute** the SimpleAppTask by double-click, context menu or Execute Button of the Scripts View and check that "HelloApplication" is printed in the console.

You can modify the implementation according to your needs. For the AutomationInterface API Reference see chapter 4 on page 20. It is sufficient to edit and save the modifications in your editor. The file is automatically reloaded by the DvDevA and can be executed immediately.

# 3. AutomationInterface Architecture

## 3.1 Components

The DvDevA consists of three components:

- Core components
- AutomationInterface (AI) - also called Automation API
- Scripting engine

The other part is the script provided by the user.

The scripting engine will load the script, and the script uses the AutomationInterface to perform tasks. The AutomationInterface will translate the requests from the script into core components calls.

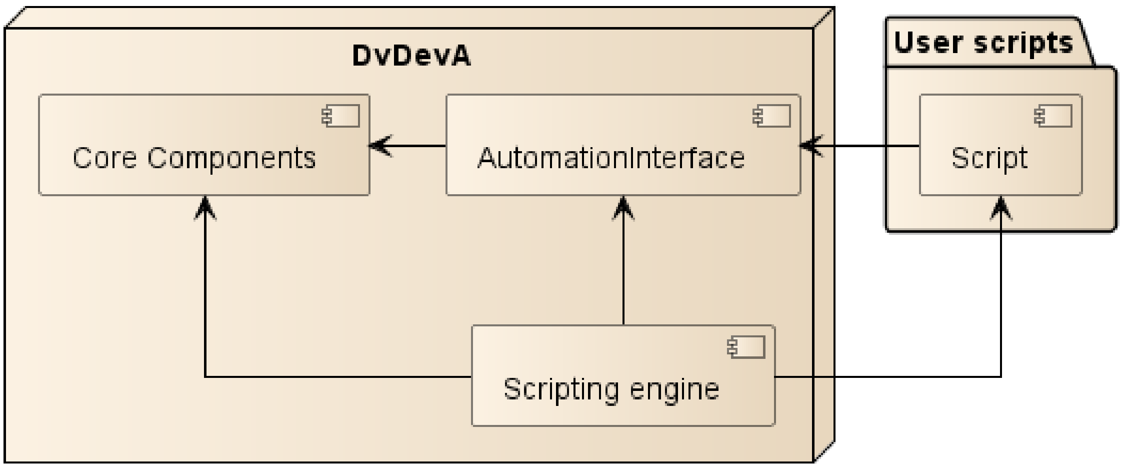


Figure 3.1: DvDevA components and interaction with scripts

The separation of the AutomationInterface and the core components has multiple benefits:

- Stable API for script writers
  - Including checks, that the API will not break in following releases
- Well defined and documented API
- Abstraction from the internal heavy lifting
  - This ease the usage for the user, because the automation interfaces are tailored to the use cases.

**PublishedApi** All AutomationInterface classes are marked with a special annotation to **highlight** the fact that it is part of the published API. The annotation is called `@PublishedApi`.

So every class marked with `@PublishedApi` can be used by the client code. But if a class is **not** marked with `@PublishedApi` or is marked with `@Deprecated` it should not be used by any client code, nor shall a client call methods via reflection or other runtime techniques.

You should **not** access DvDevA private or package private classes, methods or fields.

**PublishedBetaApi** The `PublishedBeta` annotations signifies that a public API (public class, interface, method or field) annotated with `PublishedApi` is subject to incompatible changes, or even removal, in a future release. An API bearing this annotation is exempt from any compatibility guarantees made by its containing library. Note that the presence of this annotation implies nothing about the quality or performance of the API in question, only the fact that it is not "API-frozen".

It is generally safe for **clients** to depend on beta APIs, at the cost of some extra work during upgrades. Note that the client which uses this API must upgrade to each DaVinci Product release, to guarantee, that the used API is still available.

## 3.2 Languages

The DvDevA provides out of the box language support for:

- Java[1]
- Groovy[2]

The recommended scripting language is **Groovy** which shall be preferred by all users.

### 3.2.1 Why Groovy

**Flat Learning Curve** Groovy is concise, readable with an expressive syntax and is easy to learn for Java developers[3].

- Groovy syntax is 95%-compatible with Java[4]
- Any Java developer will be able to code in Groovy without having to know nor understand the subtleties of this language

This is very important for teams where there's not much time for learning a new language.

**Domain-Specific Languages (DSL)** Groovy has a flexible and malleable syntax, advanced integration and customization mechanisms, to integrate readable business rules in your applications.

The DSL features of Groovy are extensively used in DaVinci Automation API to provide simple and expressive syntax.

**Powerful Features** The Groovy language supports Closures, builders, runtime & compile-time meta-programming, functional programming, type inference, and static compilation.

**Website** The website of Groovy is http://groovy-lang.org. It provides a good documentation and starting guides for the Groovy language.

[1]`http://http://www.java.com` [2016-05-09]

[2]`http://groovy-lang.org` [2016-05-09]

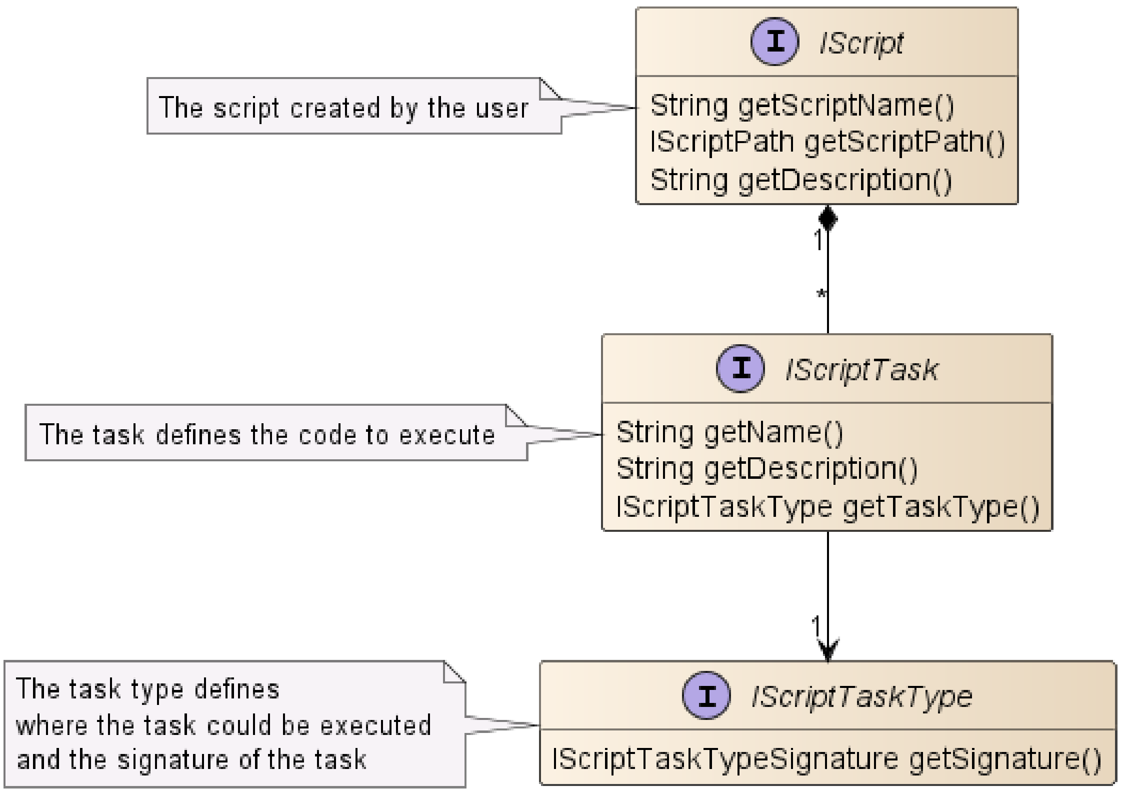
[3]Copied from `http://groovy-lang.org` [2016-05-09]

[4]Copied from `http://melix.github.io/blog/2010/07/27/experience_feedback_on_groovy.html` [2016-05-09]

**Groovy Book** The book **"Groovy in Action, Second Edition"**[5] provides a comprehensive guide to Groovy programming language. It is written by the developers of Groovy.

## 3.3 Script Structure

A script always contains one or more script tasks. A script is represented by an instance of `IScript`, the contained tasks are instances of `IScriptTask`.



The script created by the user
I IScript
String getScriptName()
IScriptPath getScriptPath()
String getDescription()
1
*
The task defines the code to execute
I IScriptTask
String getName()
String getDescription()
IScriptTaskType getTaskType()
1
The task type defines where the task could be executed and the signature of the task
I IScriptTaskType
IScriptTaskTypeSignature getSignature()

Figure 3.2: Structure of scripts and script tasks

You create the `IScript` and `IScriptTask` instance with the API described in chapter 4.2 on page 21.

The script task type (`IScriptTaskType`) defines where the task could be executed. It also defines the signature of the task's `code {}` block. See chapter 4.3 on page 25 for the available script task types.

### 3.3.1 Scripts

Script contain the tasks to execute and are loaded from the script folder in the DvDevA.

The DvDevA supports two types of automation scripts:

- Script files (`.dvgroovy` files)
- Script projects (`.jar` files)

For details to the script project, see chapter 7 on page 144.

[5] Groovy in Action, Second Edition by Dierk König, Paul King, Guillaume Laforge, Hamlet D'Arcy, Cédric Champeau, Erik Pragt, and Jon Skeet June 2015 ISBN 9781935182443 `https://www.manning.com/books/groovy-in-action-second-edition` [2016-05-09]

### 3.3.2 Script Tasks

Script tasks are the executable units of scripts (specified by the `IScriptTaskType`). Every script task has a `code {}` block, which contains the logic to execute.

## 3.4 Script loading

All scripts contained in the script folder are automatically loaded by the DvDevA. If new scripts are added to the script folder these scripts are automatically loaded.

**Note:** A jar file of a script project *should be updated by the Gradle build system*, not by hand. Because the Java VM is holding a lock to the file. If you try to replace the file in the explorer you will get an error message.

### 3.4.1 Internal Script Reload Behavior

Your script can be loaded and unloaded automatically multiple times during the execution of the DvDevA. More precise, when a script is currently not used and there are memory constraints your script will be automatically unloaded.

If the script will be executed again, it is automatically reloaded and then executed. So it is possible that the script initialization code is called multiple times in the DvDevA lifecycle. But this is no issue, because the script and the tasks **shall not** have any internal state during initialization.

**Memory Leak Prevention** The feature above is implemented to prevent leaking memory from an automation script into the DvDevA memory. So when the memory run low, all unused scripts are unloaded, which will also free leaked memory of scripts.

But this **does not** mean that is impossible to construct memory leaks from an automation script. E.g. Open file handles without closing them will still cause a memory leak.

## 3.5 Script editing

The DvDevA does not contain any editing support for scripts, like:

- Script editor
- Debugger
- REPL (Read-Eval-Print-Loop)

These tasks are delegated to other development tools:

- IntelliJ IDEA (recommended)
- EclipseIDE
- Notepad++

See chapter 7 on page 144 for script development and debugging with IntelliJ IDEA.

# 4. AutomationInterface API Reference

## 4.1 Introduction

This chapter contains the description of the DvDevA AutomationInterface. The figure 4.1 shows the APIs and the containment structure of the different APIs.

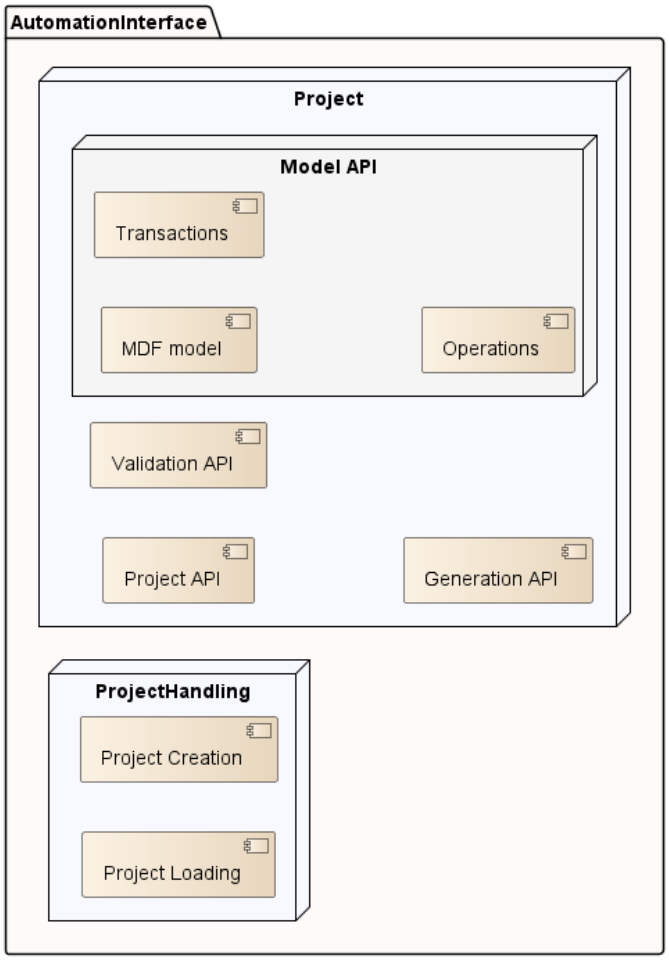


Figure 4.1: The API overview and containment structure

The components have an hierarchical order, where and when the components are usable. When a component is contained in another the component is only usable, when the other is active.

Usage examples:

- The Generation API is only usable inside of a loaded Project
- The Project creation API is only usable outside of a loaded Project

## 4.2 Script Creation

This section lists the APIs to create, execute and query information for script tasks. The sections document the following aspects:

- Script task creation
- Description and help texts
- Task executable query

### 4.2.1 Script Task Creation

To create a script task you have to call one of the `scriptTask()` methods. The last parameter of the `scriptTask` methods can be used to set additional options of the task. Every script task needs one `IScriptTaskType`. See chapter 4.3 on page 25 for all available task types.

The `code{ }` block is **required** for every `IScriptTask`. The block contains the code, which is executed when the task is executed.

**Script Task with default Type** The method `scriptTask()` will create an script task for the default `IScriptTaskType`.

Default: `DV_PROJECT`.

```
scriptTask("TaskName"){
    code{
        // Task execution code here
    }
}
```

Listing 4.1: Task creation with default type

**Script Task with Task Type** You could also define the used `IScriptTaskType` at the `script-Task()` methods.

The methods

- `scriptTask(String, IApplicationScriptTaskType, Closure)`
- `scriptTask(String, IProjectScriptTaskType, Closure)`

will create an script task for passed `IScriptTaskType`. The two methods differentiate, if a project is required or not. See chapter for all available task types 4.3 on page 25

```
scriptTask("TaskName", DV_APPLICATION){
    code{
        // Task execution code here
    }
}
```

Listing 4.2: Task creation with TaskType Application

```
scriptTask("TaskName", DV_PROJECT){
    code{
        // Task execution code here
    }
}
```

Listing 4.3: Task creation with TaskType Project

**Multiple Tasks in one Script** It is also possible to define multiple tasks in one script.

```
scriptTask("TaskName"){
    code{ }
}

scriptTask("SecondTask"){
    code{ }
}
```

Listing 4.4: Define two tasks is one script

#### 4.2.1.1 Script Creation with IDE Code Completion Support

The IDE could not know which API is available inside of a script file. So a glue code is needed to tell the IDE, what API is callable inside of a script file.

The `ScriptApi.daVinci()` method enables the IDE code completion support in a script file. You have to write the `daVinci{ }` block and inside of the block the code completion is available. The following sample shows the glue code for the IDE:

```
import static com.vector.cfg.automation.api.ScriptApi.*

//daVinci enables the IDE code completion support
daVinci{

    // Normal script code here
    scriptTask("TaskName"){
        code{
             // Script task execution code here
        }
    }
}
```

Listing 4.5: Script creation with IDE support

The `daVinci{}` block is only required for code completion support in the IDE. It has no effect during runtime, so the `daVinci{}` is optional in script files (`.dvgroovy`)

#### 4.2.1.2 Script Task isExecutableIf

You can set an `isExecutableIf` handler, which is called before the `IScriptTask` is executed. The code can evaluate, if the `IScriptTask` shall be executable. If the handler returns `true`, the code of the `IScriptTask` is executable, otherwise `false`. See class `IExecutableTaskEvaluator` for details.

The `Closure` isExecutable has to return a `boolean`. The passed arguments to the closure are the same as the `code{ }` block arguments.

Inside of the `Closure` a property `notExecutableReasons` is available to set reasons why it is not executable. It is highly recommended to set reasons, when the `Closure` returns `false`.

```
scriptTask("TaskName"){

    isExecutableIf{ taskArgument ->
        // Decide, if the task shall be executable
        if(taskArgument == "CorrectArgument"){
            return true
        }
        notExecutableReasons.addReason "The argument is not 'CorrectArgument'"
        return false
    }

    code{ taskArgument ->
        // Task execution code here
    }
}
```

Listing 4.6: Task with isExecutableIf

### 4.2.2 Description and Help

**Script Description** The script can have an optional description text. The description shall list what this script contains. The method `scriptDescription(String)` sets the description of the script.

The description shall be a short overview. The `String` can be multiline.

```
// You can set a description for the whole script
scriptDescription "The Script has a description"

scriptTask("Task"){
    code{}
}
```

Listing 4.7: Script with description

**Task Description** A script task can have an optional description text. The description shall help the user of the script task to understand what the task does. The method `taskDescription(String)` sets the description of the script task.

The description shall be a short overview. The `String` can be multiline.

```
scriptTask("TaskName"){
    taskDescription "The description of the task"

    code{ }
}
```

Listing 4.8: Task with description

**Task Help** A script task can also have an optional help text. The help text shall describe in detail what the task does and when it could be executed. The method `taskHelp(String)` sets the help of the script task.

The help shall be elaborate text about what the task does and how to use it. The `String` can be multiline.

The help text is automatically expanded with the help for user defined script task arguments, see `IScriptTaskBuilder.newUserDefinedArgument(String, Class, String)`.

```
scriptTask("TaskName"){
  taskDescription "The short description of the task"
  taskHelp """
          The long help text
          of the script with multiple lines

          And paragraphs ...
       """
  // The three """ are needed, if you want to write a multiline string

    code{ }
}
```

Listing 4.9: Task with description and help text

## 4.3 Script Task Types

The `IScriptTaskType` instances define where a script task is executed in the DvDevA. The types also define the arguments passed to the script task execution and what return type an execution has.

Every script task needs an `IScriptTaskType`. The type is set during creation of the script tasks.

**Interfaces** All task types implement the interface `IScriptTaskType`. The following figure show the type and the defined sub types:

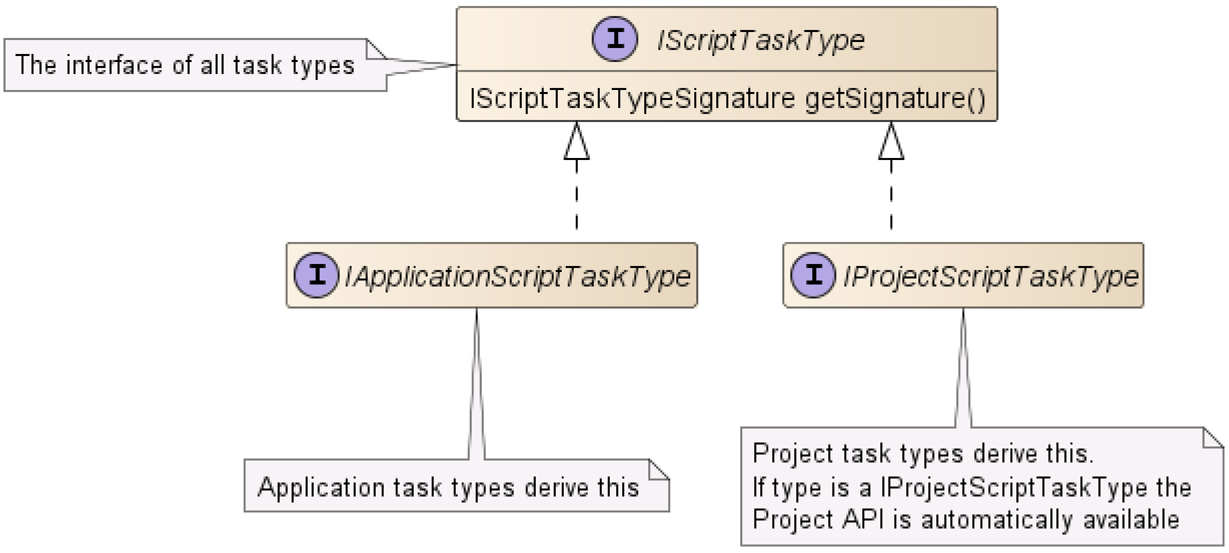

The interface of all task types
IScriptTaskType
IScriptTaskTypeSignature getSignature()
IApplicationScriptTaskType
IProjectScriptTaskType
Application task types derive this
Project task types derive this.
If type is a IProjectScriptTaskType the
Project API is automatically available

Figure 4.2: IScriptTaskType interfaces

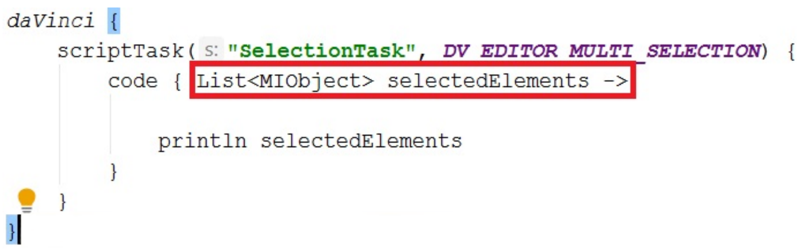
### 4.3.1 Available Types

The class `IScriptTaskTypeApi` defines all available `IScriptTaskTypes` in the DvDevA. All task types start with the prefix `DV_`.

`None` at parameters and return types mean, that any arguments could be passed and return to or from the task. Normally it will be nothing. The arguments are used, when the task is called in unit tests for example.

ScriptTaskType input parameters can get accessed via script by adding them to the code {} closure. See example:

4.3



```
daVinci {
    scriptTask( s: "SelectionTask", DV_EDITOR_MULTI_SELECTION) {
        code { List<MIObject> selectedElements ->

            println selectedElements
        }
    }
}
```

Figure 4.3: ScriptTaskType input parameters in code closure

#### 4.3.1.1 Application Types

**Application** The type `DV_APPLICATION` is for application wide script tasks. A task could create/open/close/update projects. Use this type, if you need full control over the project handling, or you want to handle multiple project at once.

| | |
|---|---|
| **Name** | Application |
| **Code identifier** | DV_APPLICATION |
| **Task type interface** | `IApplicationScriptTaskType` |
| **Parameters** | None |
| **Return type** | None |
| **Execution** | Standalone |

#### 4.3.1.2 Project Types

**Project** The type `DV_PROJECT` is for project script tasks. A task could access the currently loaded project. Manipulate the data, generate and save the project. This is the default type, if no other type is specified.

| | |
|---|---|
| **Name** | Project |
| **Code identifier** | DV_PROJECT |
| **Task type interface** | `IProjectScriptTaskType` |
| **Parameters** | None |
| **Return type** | None |
| **Execution** | Standalone |

#### 4.3.1.3 UI Types

**Editor selection** The type `DV_EDITOR_SELECTION` allows the script task to access the currently selected element of an editor. The task is executed in context of the selection and is not callable by the user without an active selection.

| | |
|---|---|
| **Name** | Editor selection |
| **Code identifier** | DV_EDITOR_SELECTION |
| **Task type interface** | `IProjectScriptTaskType` |
| **Parameters** | `MIObject selectedElement` |
| **Return type** | `Void` |
| **Execution** | In context menu of an editor selection |

**Editor multiple selections** The type `DV_EDITOR_MULTI_SELECTION` allows the script task to access the currently selected elements of an editor.
The task is executed in context of the selection and is not callable by the user without an active selection.
The type is also usable when the `DV_EDITOR_SELECTION` apply.

| | |
|---|---|
| **Name** | Editor multiple selections |
| **Code identifier** | DV_EDITOR_MULTI_SELECTION |
| **Task type interface** | `IProjectScriptTaskType` |
| **Parameters** | `List<MIObject> selectedElements` |
| **Return type** | `Void` |
| **Execution** | In context menu of an editor selection |

Those ScriptTaskTypes can be executed via selecting Configuration Elements in the editor. See example:

4.4

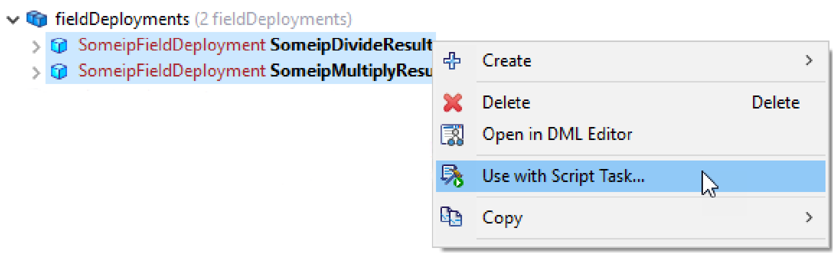

fieldDeployments (2 fieldDeployments)
SomeipFieldDeployment SomeipDivideResult
SomeipFieldDeployment SomeipMultiplyResu
Create
Delete
Delete
Open in DML Editor
Use with Script Task...
Copy

Figure 4.4: Access Editor Selection

#### 4.3.1.4 Generation Types

**Generation Process Start** The type `DV_ON_GENERATION_START` defines that the script task is automatically executed when **validation**(including **full validation**) or **generation** is started. The parameter **phasesToExecute** will be [`EGenerationPhaseType.VALIDATION`] for validation, and [`EGenerationPhaseType.VALIDATION`, `EGenerationPhaseType.GENERATION`] for generation.

| | |
|---|---|
| **Name** | Generation Process Start |
| **Task type interface** | `IProjectScriptTaskType` |
| **Code identifier** | DV_ON_GENERATION_START |
| **Parameters** | `List<EGenerationPhaseType> phasesToExecute` |
| | `List<IGenerator> generators` |
| **Return type** | `Void` |
| **Execution** | Automatically before GenerationProcess |

See chapter 4.8.2 on page 108 for usage samples.

**Generation Process End** The type `DV_ON_GENERATION_END` defines that the script task is automatically executed when **validation**(including **full validation**) or **generation** has finished.

| | |
|---|---|
| **Name** | Generation Process End |
| **Code identifier** | DV_ON_GENERATION_END |
| **Task type interface** | `IProjectScriptTaskType` |
| **Parameters** | `EGenerationProcessResult processResult` |
| | `List<IGenerator> generators` |
| **Return type** | `Void` |
| **Execution** | Automatically after GenerationProcess |

See chapter 4.8.2 on page 108 for usage samples.

#### 4.3.1.5 Model Extension Migration Types

**Mex Migration** The type `DV_MEX_MIGRATION` is for model extension migration. A task would get a `IMexExtension` in parameters to access and manipulate the corresponding model extension data. Script will be executed in a transaction automatically, so there is no need to create one if you want to change the model.

The task name of script should follow the naming convention:
<EXTENSION-MODEL-NAME>_<TARGET-VERSION>, e.g. "PrioLimitModel_2.0.0".

- EXTENSION-MODEL-NAME is the shortname of the SdgDef which defines the extension model to be migrated.

- TARGET-VERSION is the target version of the extension model, this migration script migrates to. After migration, the affected Sdgs will have this version.

| | |
|---|---|
| **Name** | Mex Migration |
| **Code identifier** | DV_MEX_MIGRATION |
| **Task type interface** | `IProjectScriptTaskType` |
| **Parameters** | `IMexExtension mexExtension` |
| **Return type** | None |
| **Execution** | Automatically during mex migration |

## 4.4 Script Task Execution

This section lists the APIs to execute and query information for script tasks. The sections document the following aspects:

- Script task execution
- Logging API
- Path resolution
- Error handling
- User defined classes and methods
- User defined script task arguments

### 4.4.1 Execution Context

Every `IScriptTask` could be be executed, and retrieve passed arguments and other context information. This execution information of a script task is tracked by the `IScriptExecutionContext`.

The `IScriptExecutionContext` holds the context of the execution:

- The script task arguments
- The current running script task
- The current active script logger
- The active project, if existing
- The script temp folder
- The script task user defined arguments

The `IScriptExecutionContext` is also the entry point into every automation API, and provide access to the different API classes. The classes are describes in their own chapters like `IProjectHandlingApiEntryPoint`.

The context is immediately active, when the code block of an `IScriptTask` is called.

**Groovy Code** The client sample illustrates the seamless usage of the `IScriptExecutionContext` class in Groovy:

```
scriptTask("taskName", DV_APPLICATION){
  code{  // The IScriptExecutionContext is automatically active here
     // Call methods of the IScriptExecutionContext
     def logger = scriptLogger
     def temp = paths.tempFolder

     // Use an automation API
  }
}
```

Listing 4.10: Access automation API in Groovy clients by the IScriptExecutionContext

In Groovy the `IScriptExecutionContext` is automatically activated inside of the `code{}` block.

**Java Code** For java clients the method `IScriptExecutionContext.getInstance(Class)` provides access to the API classes, which are seamlessly available for the groovy clients:

```
// Java code
// Retrieve automation API in Java
IProjectHandlingApi projectHandlingApi = IScriptExecutionContext.getInterfaces(
    IProjectHandlingApi.class);
IProject project = projectHandlingApi.getActiveProject();
project.getProjectFilePath();

// In groovy code it would be:
activeProject{
     getProjectFilePath()
}
```

Listing 4.11: Access to automation API in Java clients by the IScriptExecutionContext

In Java code the context is always the first parameter passed to every task code (see `IScriptTaskCode`).

#### 4.4.1.1 Code Block Arguments

The code block can have arguments passed into the script task execution. The arguments passed into the `code{ }` block are defined by the `IScriptTaskType` of the script task. See chapter 4.3 on page 25 for the list of arguments (including types) passed by each individual task type.

```
scriptTask("Task"){
  code{ arg1, arg2, ... -> // arguments here defined by the IScriptTaskType

  }
}

scriptTask("Task2"){
  // Or you could specify the type of the arguments for code completion
  code{ String arg1, List<Double> arg2 ->
  }
}
```

Listing 4.12: Script task code block arguments

The arguments can also retrieved with `IScriptExecutionContext.getScriptTaskArguments()`.

### 4.4.2 Task Execution Sequence

The figure 4.5 on the following page shows the overview sequence when a script task gets executed by the user and the interaction with the `IScriptExecutionContext`. Note that the context gets created each time the task is executed.

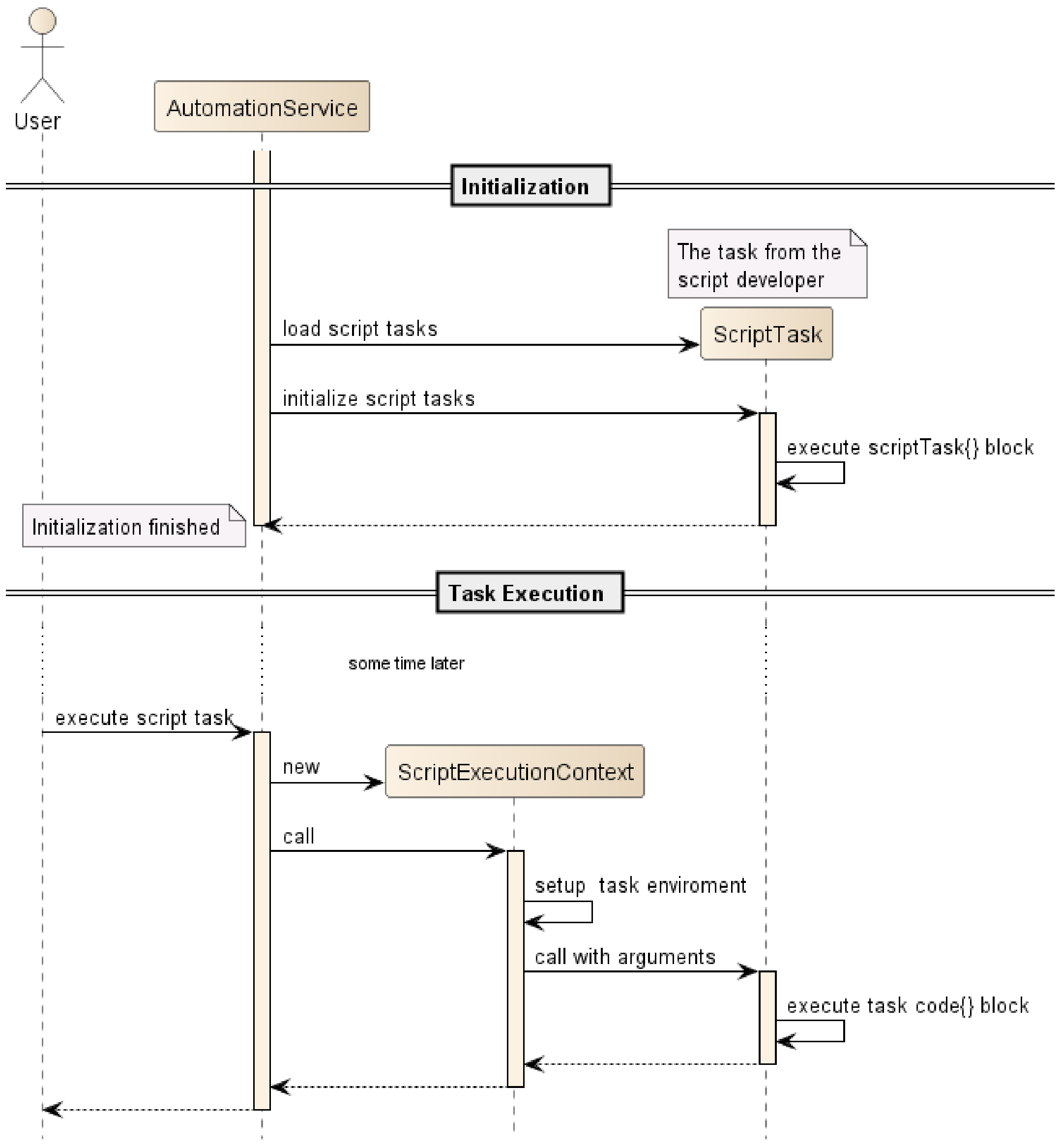


Figure 4.5: Script Task Execution Sequence

### 4.4.3 Script Path API during Execution

Script tasks could resolve relative and absolute file system paths with the `IAutomationPath-sApi`.

As entry point call `paths` in a `code{ }` block (see `IScriptExecutionContext.getPaths()`).

There are multiple ways to resolve relative paths:

- by Script folder
- by Temp folder
- by BSW Package folder
- by Project folder
- by any parent folder

#### 4.4.3.1 Path Resolution by Parent Folder

The `resolvePath(Path parent, Object path)` method resolves a file path relative to supplied parent folder.

This method converts the supplied path based on its type:

- A `CharSequence`, including `String` or `GString`. Interpreted relative to the parent directory. A string that starts with `file:` is treated as a file URL.
- A `File`: If the file is an absolute file, it is returned as is. Otherwise, the file's path is interpreted relative to the parent directory.
- A `Path`: If the path is an absolute path, it is returned as is. Otherwise, the path is interpreted relative to the parent directory.
- A `URI` or `URL`: The URL's path is interpreted as the file path. Currently, only `file:` URLs are supported.
- A `IHasURI`: The returned URI is interpreted as defined above.
- A `Closure`: The closure's return value is resolved recursively.
- A `Callable`: The callable's return value is resolved recursively.
- A `Supplier`: The supplier's return value is resolved recursively.
- A `Provider`: The provider's return value is resolved recursively.

The return type is `java.nio.file.Path`.

```
scriptTask("TaskName"){
    code{
       // Method resolvePath(Path, Object) resolves a path relative to the
           supplied folder
       Path parentFolder = Paths.get('.')
       Path p = paths.resolvePath(parentFolder, "MyFile.txt")

      /* The resolvePath(Path, Object) method will resolve
       * relative and absolute paths to a java.nio.file.Path object.
       */
    }
}
```

Listing 4.13: Resolves a path with the resolvePath() method

#### 4.4.3.2 Path Resolution

The `resolvePath(Object)` method resolves the `Object` to a file path. Relative paths are preserved, so relative paths are not converted into absolute paths.

This method converts the supplied path same as the `resolvePath(Path, Object)` method. The return type is `java.nio.file.Path`. See 4.4.3.1. But it does **NOT** convert relative paths into absolute.

```
scriptTask("TaskName"){
    code{
       // Method resolvePath() resolves a path and preserve relative paths
       Path p = paths.resolvePath("MyFile.txt")

      /* The resolvePath() method will resolve
       * relative and absolute paths to a java.nio.file.Path object.
       * Is also preserves relative paths.
       */
    }
}
```

Listing 4.14: Resolves a path with the resolvePath() method

#### 4.4.3.3 Script Folder Path Resolution

The `resolveScriptPath(Object)` method resolves a file path relative to the script directory of the executed `IScript`.

This method converts the supplied path same as the `resolvePath(Path, Object)` method. The return type is `java.nio.file.Path`. See 4.4.3.1 on the preceding page.

```
scriptTask("TaskName"){
    code{
       // Method resolveScriptPath() resolves a path relative to the script folder
       Path p = paths.resolveScriptPath("MyFile.txt")

      /* The resolveScriptPath() method will resolve
       * relative and absolute paths to a java.nio.file.Path object.
       */
    }
}
```

Listing 4.15: Resolves a path with the resolveScriptPath() method

#### 4.4.3.4 Project Folder Path Resolution

The `resolveProjectPath(Object)` method resolves a file path relative to the project directory (see `getDpaProjectFolder()`) of the current active project.

This method converts the supplied path same as the `resolvePath(Path, Object)` method. The return type is `java.nio.file.Path`. See 4.4.3.1 on the previous page.

There must be an active project to use this method. See chapter 4.5.2 on page 50 for details about active projects.

```
scriptTask("TaskName"){
    code{
       // Method resolveProjectPath() resolves a path relative active project
           folder
       Path p = paths.resolveProjectPath("MyFile.txt")

      /* The resolveProjectPath() method will resolve
       * relative and absolute paths to a java.nio.file.Path object.
       */
    }
}
```

Listing 4.16: Resolves a path with the resolveProjectPath() method

### 4.4.3.5 BSW Package Folder Path Resolution

The `resolveBSWPackagePath(Object)` method resolves a file path relative to the BSW Package directory (see `getBswPackageRootFolder()`).

This method converts the supplied path same as the `resolvePath(Path, Object)` method. The return type is `java.nio.file.Path`. See 4.4.3.1 on page 32.

```
scriptTask("TaskName"){
    code{
       // Method resolveBSWPackagePath() resolves a path relative BSW Package
          folder
       Path p = paths.resolveBSWPackagePath("MyFile.txt")

      /* The resolveBSWPackagePath() method will resolve
       * relative and absolute paths to a java.nio.file.Path object.
       */
    }
}
```

Listing 4.17: Resolves a path with the resolveBSWPackagePath() method

```
scriptTask("TaskName"){
    code{
       // The property bswPackageRootFolder is the folder of the used BSW Package
       Path folder = paths.bswPackageRootFolder
    }
}
```

Listing 4.18: Get the BSW Package folder path

### 4.4.3.6 Temp Folder Path Resolution

The `resolveTempPath(Object)` method resolves a file path relative to the script temp directory of the executed `IScript`. A new temporary folder is created for each `IScriptTask` execution.

This method converts the supplied path same as the `resolvePath(Path, Object)` method. The return type is `java.nio.file.Path`. See 4.4.3.1 on page 32.

```
scriptTask("TaskName"){
    code{
       // Method resolveTempPath() resolves a path relative to the temp folder
       Path p = paths.resolveTempPath("MyFile.txt")

      /* The resolveTempPath() method will resolve
       * relative and absolute paths to a java.nio.file.Path object.
       */
    }
}
```

Listing 4.19: Resolves a path with the resolveTempPath() method

## 4.4.4 Script logging API

The script task execution (`IScriptExecutionContext`) provides a script logger to log events during an execution. The method `getScriptLogger()` returns the logger. The logger can be used to log:

- Errors

- Warnings
- Debug messages
- More...

You shall **always prefer** the usage of the **logger** before using the `println()` of `stdout or stderr`.

In any code block without direct access to the script API, you can write the following code to access the logger: `ScriptApi.scriptLogger`

```
scriptTask("TaskName"){
    code{
        // Use the scriptLogger to log messages
        scriptLogger.info  "My script is running"
        scriptLogger.warn  "My Warning"
        scriptLogger.error "My Error"
        scriptLogger.debug "My debug message"
        scriptLogger.trace "My trace message"

        // Also log an Exception as second argument
        scriptLogger.error("My Error", new RuntimeException("MyException"))
        scriptLogger.fatal("My Fatal Error", new RuntimeException("MyException"))
    }
}
```

Listing 4.20: Usage of the script logger

The `ILogger` also provides a formatting syntax for the format String. The syntax is `{IndexNumber}` and the index of arguments after the format `String`.

It is also possible to use the Groovy `GString` syntax for formatting.

```
scriptTask("TaskName"){
    code{ argument ->
        // Use the format methods to insert data
        scriptLogger.infoFormat("My script {0} with:{1}", scriptTask, argument)
    }
}
```

Listing 4.21: Usage of the script logger with message formatting

```
scriptTask("TaskName"){
    code{ argument ->
        // Use the Groovy GString syntax to insert data
        scriptLogger.info "My script $scriptTask with: $argument"
    }
}
```

Listing 4.22: Usage of the script logger with Groovy GString message formatting

### 4.4.5 User Interactions and Inputs

The UserInteraction and UserInput API provides methods to directly communicate with the user via MessageBoxes, Input dialogs or report progress of long running operations.

You should use the API only if you want to communicate directly with the user, because some API calls may block and wait for user interaction. So you should not use the API for batch jobs.

#### 4.4.5.1 User Interaction

The User Interaction API provides methods to display messages to the user directly. In UI mode the DvDevA will prompt a message box an will block until the user has acknowledged the message. In console (non UI) mode, the message is logged to the console in a user logger.

The user logger will display error, warnings and infos by default. The logger name will not be displayed.

The user interaction is good to display information where the user has to respond to immediately. Please use the feature sparingly, because users do not like to acknowledge multiple messages for a single script task execution.

The code block `userInteractions{}` provides the API inside of the block. The following methods can be used:

- `errorToUser()`
- `warnToUser()`
- `infoToUser()`
- `messageToUser(ELogLevel, Object)`

The severity (error, warning, info) will change the display (icons, text) of the message box. No other semantic is applied by the severity.

```
scriptTask("TaskName", DV_APPLICATION){
    code{

        userInteractions{
            warnToUser("Warning displayed to the user as message box")
        }

        // You could also write
        userInteractions.errorToUser("Error message for the user")
    }
}
```

Listing 4.23: UserInteraction from a script

#### 4.4.5.2 Progress Indication

If you perform long running operations in a script task, you should display some progress to the user, otherwise the user may cancel the whole execution. The progress API will display the progress of the currently running script task by the information provided by the script code.

The method `progress(String, Closure)` displays the passed message in progress information dialog and executed the code block. So the message is displayed until the code block has finished.

```
userInteractions.progress("The text for the user"){
  // Here the code of the long running operation
}
```

Listing 4.24: Display progress to the user

You could also nest multiple `progress()` calls. When a progress block is left, the parent progress text will be displayed again.

```
userInteractions{
  progress("The text for the user"){
    // Here the code of the long running operation
    progress("Inner operation"){
      // Here code of inner operation
    }
  }
  progress("Second operation"){
    //Code of the second operation
  }
}
```

Listing 4.25: Display progress to the user nested

The method `progress(String, int, Closure)` updates the progress information for the user with the message, during the code is running with work ticks.

It also indicates progress in the progress bar, but you have to set the total amount of work. The total work will be taken from the parent and sets the remaining work for the code block.

The root script task always starts with totalWork of `1000` ticks, so you have to consume `1000` ticks to fill the progress bar.

```
userInteractions{
  progress("The text for the user", 1000){
    worked(100)
    progress("Inner operation", 400){
        //100 ticks
        worked(200)
        //300 ticks
    }
    // half reached - 500 ticks
    progress("Inner operation", 200){
        worked(100)
        // 600 ticks reached
    }
    // 700 ticks reached
  }
  // All 1000 ticks done, the progress bar is now full!
}
```

Listing 4.26: Display progress to the user with progress bar work

**Eclipse API** You can also use the underlying Eclipse API to fine grain control the progress bar and information data. To do this use the `getProgressMonitor()` method to retrieve the Eclipse `SubMonitor`. See also the Eclipse API `SubMonitor.setWorkRemaining(int)` to scale your own work to different values (also more than `1000` ticks).

### 4.4.6 Script Error Handling

#### 4.4.6.1 Script Exceptions

All exceptions thrown by any script task execution are sub types of `ScriptingException`.

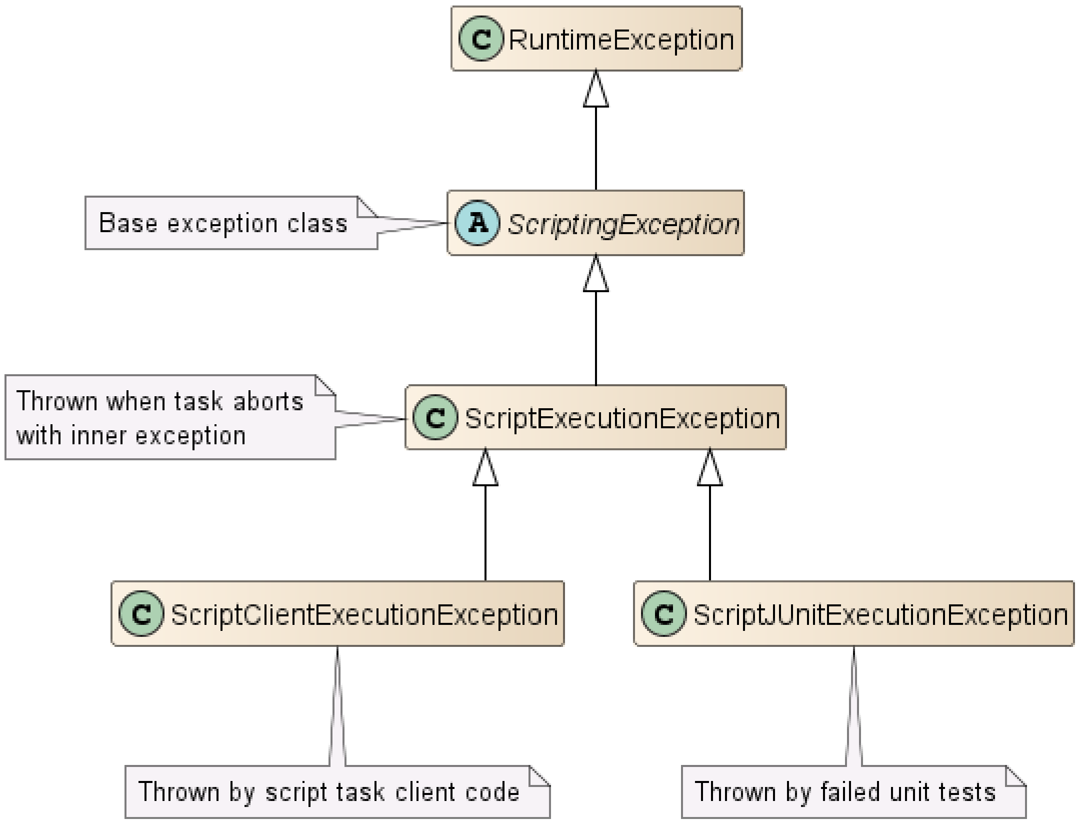

RuntimeException
Base exception class
ScriptingException
Thrown when task aborts with inner exception
ScriptExecutionException
ScriptClientExecutionException
ScriptJUnitExecutionException
Thrown by script task client code
Thrown by failed unit tests

Figure 4.6: ScriptingException and sub types

#### 4.4.6.2 Script Task Abortion by Exception

The script task can throw an `ScriptClientExecutionException` to abort the execution of an `IScriptTask`, and display a meaningful message to the user.

```
scriptTask("TaskName"){

    code{
        // Stop the execution and display a message to the user
        throw new ScriptClientExecutionException("Message to the User")
    }
}
```

Listing 4.27: Stop script task execution by throwing an ScriptClientExecutionException

**Exception with Console Return Code** An `ScriptClientExecutionException` with an return code of type `Integer` will also abort the execution of the `IScriptTask`.

But it *also changes the return code* of the console application, if the `IScriptTask` was executed in the console application. This could be used when the DvDevA is called for other scripts or batch files.

```
scriptTask("TaskName"){

    code{
        // The return code will be returned by the command line process
        def returnCode = 50
        throw new ScriptClientExecutionException(returnCode, "Message to the User"
            )
    }
}
```

Listing 4.28: Changing the return code of the console application by throwing an ScriptClientExecutionException

**Reserved Return Codes** The returns codes `0-20` are reversed for internal use of the DvDevA, and are not allowed to be used by a client script. Also negative returns codes are not permitted.

#### 4.4.6.3 Unhandled Exceptions from Tasks

When a script task execution throws any type of `Exception` (more precise `Throwable`) the script task is marked as failed and the `Exception` is reported to the user.

### 4.4.7 User defined Classes and Methods

You can define your own methods and classes in a script file. The methods a called like any other method.

```
scriptTask("Task"){
    code{
        userMethod()
    }
}

def userMethod(){
    return "UserString"
}
```

Listing 4.29: Using your own defined method

Classes can be used like any other class. It is also possible to define multiple classes in the script file.

```
scriptTask("Task"){
    code{
        new UserClass().userMethod()
    }
}

class UserClass{
    def userMethod(){
        return "ReturnValue"
    }
}
```

Listing 4.30: Using your own defined class

You can also create classes in different files, but then you have to write imports in your script like in normal Groovy or Java code.

The script should be structured as any other development project, so if the script file gets too big, please refactor the parts into multiple classes and so on.

**daVinci Block** The classes and methods must be outside of the `daVinci{ }` block.

```
import static com.vector.cfg.automation.api.ScriptApi.*
daVinci{
    scriptTask("Task"){
        code{}
    }
}

def userMethod(){}

class UserClass{}
```

Listing 4.31: Using your own defined method with a daVinci block

**Code Completion** Note that the code completion for the Automation API will not work automatically in own defined classes and methods. You have to open for example a `scriptCode{}` block. The chapter4.4.8 on the next page describes how to use the Automation API for your own defined classes and methods.

### 4.4.8 Usage of Automation API in own defined Classes and Methods

In your own methods and classes the automation API is not automatically available differently as inside of the script task `code{}` block. But it is often the case, that methods need access to the automation API.

The class `ScriptApi` **provides static methods as entry points** into the automation API. The static methods either return the API objects, or you could pass a `Closure`, which will activate the API inside of the `Closure`.

#### 4.4.8.1 Access the Automation API like the Script code{} Block

The `ScriptApi.scriptCode(Closure)` method provides access to all automation APIs the same way as inside of the normal script `code{}` block.

This is useful, when you want to call script code API inside of your own methods and classes.

```
def yourMethod(){
    // Needs access to an automation API
    ScriptApi.scriptCode{
        // API is now available
        projects.isProjectActive()
    }
}
```

Listing 4.32: ScriptApi.scriptCode{} usage in own method

The `ScriptApi.scriptCode()` method can be used to call API in Java style.

```
def yourMethod(){
    // Needs access to an automation API
    ScriptApi.scriptCode().projects.isProjectActive()
}
```

Listing 4.33: ScriptApi.scriptCode() usage in own method

Java note: The `ScriptApi.scriptCode()` returns the `IScriptExecutionContext`.

#### 4.4.8.2 Access the Project API of the current active Project

The `ScriptApi.activeProject()` method provides access to the project automation API of the currently active project. This is useful, when you want to call project API inside of your own methods and classes.

```
def yourMethod(){
    // Needs access to an automation API
    ScriptApi.activeProject{
        // Project API is now available
        transaction{
            // Now model modifications are allowed
        }
    }
}
```

Listing 4.34: ScriptApi.activeProject{} usage in own method

The `ScriptApi.activeProject()` method returns the current active `IProject`.

```
def yourMethod(){
    // Needs access to an automation API
    IProject theActiveProject = ScriptApi.activeProject()
}
```

Listing 4.35: ScriptApi.activeProject() usage in own method

#### 4.4.8.3 Inject a service

In order to get a service instance inside a user defined method it has to be injected via the `getInstance(Class)` function from either `ScriptApi.activeProject()` or `ScriptApi.scriptCode()`:

```
def myMethodUsingServices() {
    myServiceFromScript = scriptCode().getInstance(MyService1.class)
    myServiceFromProject = activeProject().getInstance(MyService2.class)
    // E.g. get IReferrableAccess service from project
    IReferrableAccess referrableAccess = activeProject().getInstance(
        IReferrableAccess.class)
    MIReferrable package = referrableAccess.getReferrableByPath("/vector")
}
```

Listing 4.36: Injecting a service from a user defined method

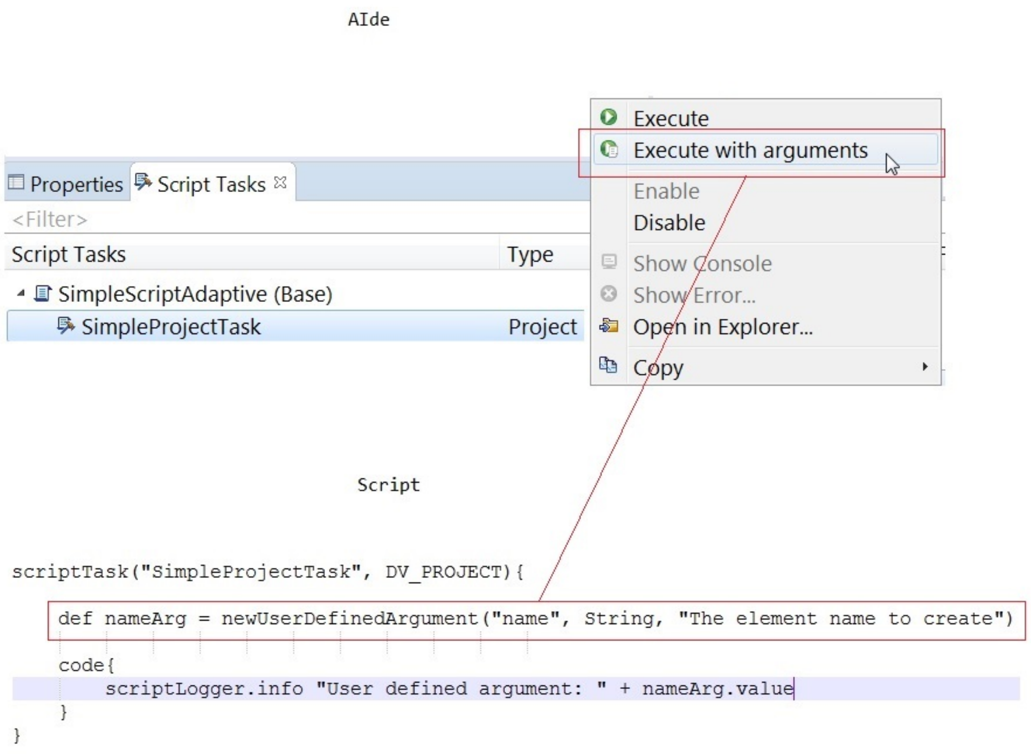
### 4.4.9 User defined Script Task Arguments

A script task can create `IScriptTaskUserDefinedArgument`, which can be set by the user to pass user defined arguments to the script task execution. The arguments can be passed via commandline or gui, see 4.7 on the following page . An argument can be optional or required. The arguments are type safe and checked before the task is executed. An argument can be specified with a value and also without one.
Example: "–name Blubb" or "-s"

Possible valueTypes are:

- `String`
- `Boolean`
- `Void`: For parameter where only the existence is relevant.
- `File`: The existence of the file is not checked by default. See argument validators.
- `Path`: Same as `File`
- `Integer`
- `Long`
- `Double`

The help text is automatically expanded with the help for user defined script task arguments.



```
scriptTask("SimpleProjectTask", DV_PROJECT){
    def nameArg = newUserDefinedArgument("name", String, "The element name to create")
    code{
        scriptLogger.info "User defined argument: " + nameArg.value
    }
}
```

Figure 4.7: GUI - User defined Arguments

```
/*
Cmd Script Call Syntax
--scriptTask "MyScript:TaskName" --taskArgs "TaskName" "-p"
*/
scriptTask("TaskName"){
  def procArg = newUserDefinedArgument("p", Void, "Enables the processing of ...")
  code{
      if(procArg.hasValue){
        scriptLogger.info  "The argument -p was defined"
      }
  }
}
```

Listing 4.37: Script task UserDefined argument with no value

```
/*
Cmd Script Call Syntax
--scriptTask "MyScript:TaskName" --taskArgs "TaskName" "--count 25 --name Test"
*/
scriptTask("TaskName"){
  def countArg = newUserDefinedArgument("count", Integer,
                                        "The amount of elements to create")

  def nameArg = newUserDefinedArgument("name", String,
                                       "The element name to create")
  code{
      // NOTE: The value can only be retrieved within the code closure
      int count = countArg.value
      String name = nameArg.value

      scriptLogger.info  "The arguments --name and --count were $name, $count"
  }
}
```

Listing 4.38: Define and use script task user defined arguments from commandline

```
/*
Cmd Script Call Syntax
--scriptTask "MyScript:TaskName" --taskArgs "TaskName" "-p"
*/
scriptTask("TaskName"){
  //  User Defined Argument with the default value 25.0
  def procArg = newUserDefinedArgument("p", Double, 25.0, "Help text ...")
  code{
      double value = procArg.value
      scriptLogger.info  "The argument -p was $value"
  }
}
```

Listing 4.39: Script task UserDefined argument with default value

```
/*
Cmd Script Call Syntax
--scriptTask "MyScript:TaskName" --taskArgs "TaskName" "--multiArg "argValueOne"
    --multiArg "argValueTwo""
*/
scriptTask("TaskName"){
  def multiArg = newUserDefinedArgument("multiArg", String, "Help text ...")

  code{

      List<String> values = multiArg.values  // Call values instead of value
      scriptLogger.info  "The argument --multiArg  had values: $values"
  }
}
```

Listing 4.40: Script task UserDefined argument with multiple values

#### 4.4.9.1 User defined Argument Validators

You could also specify a validator for the argument to check for special conditions, like the file must exist. This is helpful to provide a quick feedback to the user, if the task would be executable. Simply add the validator at the end of the `newUserDefinedArgument()` call. The validator code is called when the input is checked.

There are also default validators available, like:

- Constraints.IS_EXISTING_FOLDER
- Constraints.IS_EXISTING_FILE
- Constraints.IS_VALID_AUTOSAR_SHORT_NAME

Please see chapter 4.11.1 on page 122 for more available validators.

```
/*
Cmd Script Call Syntax
--scriptTask "MyScript:TaskName" --taskArgs "TaskName" "-p InvalidAsrShortName"
*/
import com.vector.cfg.business.Constraints

scriptTask("TaskName"){
  def contArg = newUserDefinedArgument( "p", String,
                                        "Help text ...",
                                        Constraints.
                                            IS_VALID_AUTOSAR_SHORT_NAME_PATH )
  code{

      String value = contArg.value
      scriptLogger.info  "The argument -p was $value"
  }
}
```

Listing 4.41: Script task UserDefined argument with predefined validator

Or you implement your own validation logic, by passing a `Closure`, which throws an exception, if the value is invalid.

```
/*
Cmd Script Call Syntax
--scriptTask "MyScript:TaskName" --taskArgs "TaskName" "-p"
*/
scriptTask("TaskName"){

  //  User Defined Argument with the validator code as parameter
  def procArg = newUserDefinedArgument( "p", Integer, 20, "Help text ...",
                { value ->
                    if( value % 2){
                        throw new IllegalArgumentException("The value has to be
                            even.")
                    }
                } )

  code{
  }
}
```

Listing 4.42: Script task UserDefined argument with own validator

#### 4.4.9.2 Call Script Task with Task Arguments from Commandline

The commandline option `taskArgs` is used to specify the arguments passed to a script task to execute:

```
--taskArgs <TASK_ARGS>   Passes arguments to the specified script tasks.
```

The arguments have the following syntax:

```
Syntax: --taskArgs "<TaskName>" "<ArgName ArgValue>"
                   |             |________________||
                   |________________________________|
Examples:
    Single Argument without value:
        E.g. --taskArgs "MyTask" "-s"
    Single Argument with value:
        E.g. --taskArgs "MyTask" "--count 25"
    Multiple Arguments with and without value:
        E.g. --taskArgs "MyTask" "-s --count 25"
```

If only one task is executed, the `"<TaskName>"` can be skipped.

If there are multiple tasks apply for the following syntax:

```
Syntax: --taskArgs "<TaskName>" "<ArgName ArgValue>"
                   "<TaskName2>" "<ArgName ArgValue>"

   E.g. --taskArgs "MyTask" "-s --projectCfg MyFile.cfg"
                   "Task2" "-d --saveTo saveFile.txt"
```

Note: The newlines in the listing are only for visualization.

If the task name is not unique, your can specify the full qualified name with script name

```
  --taskArgs "MyScript:MyTask" "--projectCfg MyFile.cfg"
```

Arguments with spaces inside the script task argument could be quoted with `""`

```
  --taskArgs "MyScript:MyTask" "--projectCfg \"Path to File\MyFile.cfg\""
```

The task help of a task will print the possible arguments of a script task.

```
  --scriptTaskHelp taskName
```

### 4.4.10 Stateful Script Tasks

Script tasks normally have no state or cached data, but it can be useful to cache data during an execution, or over multiple task executions. The `IScriptExecutionContext` provides two methods to save and restore data for that purpose:

- `getExecutionData()` - caches data during one task execution
- `getSessionData()` - caches data over multiple task executions

**Execution Data** Caches data during a single script task execution, which allows to save calculated values or services needed in multiple parts of the task, without recalculating or creating it.

NOTE: When the task is executed again the `executionData` will be empty.

```
scriptTask("TaskName"){
  code{
      // Cache a value for the execution
      executionData.myCacheValue = 500

      def val = executionData.myCacheValue // Retrieve the value anywhere
      scriptLogger.info  "The cached value is $val"

      // Or access it from any place with ScriptApi.scriptCode like:
      def sameValue = ScriptApi.scriptCode.executionData.myCacheValue
  }
}
```

Listing 4.43: executionData - Cache and retrieve data during one script task execution

**Session Data** Caches data over multiple task executions, which allows to implement a stateful task, by saving and retrieving any data calculated by the task itself.

**Caution:** The data is saved globally so the usage of the `sessionData` can lead to memory leaks or `OutOfMemoryErrors`. You have to take care not to store too much memory in the `sessionData`.

The DvDevA will also free the `sessionData`, when the system run low on free memory. So you have to deal with the fact, that the `sessionData` was freed, when the script task getting executed again. But the data is not deallocated during a running execution.

```
scriptTask("TaskName"){
  // Setup - set the value the first time, this is only executed once (during
      initialization)
  sessionData.myExecutionCount = 1

  code{
      // Retrieve the value
      def executionCount = sessionData.myExecutionCount

      scriptLogger.info  "The task was executed $executionCount times"

      // Update the value
      sessionData.myExecutionCount = executionCount + 1
  }
}
```

Listing 4.44: sessionData - Cache and retrieve data over multiple script task executions

**API usage** Both methods `executionData` and `sessionData` return the same API of type `IScriptTaskUserData`.

The `IScriptTaskUserData` provides methods to retrieve and store properties by a key (like a `Map`). The retrieval and store methods are `Object` based, so any `Object` can be a key. The exception are `Class` instances (like `String.class`, which required that the value is an instance of the `Class`).

On retrieval if a property does not exist an `UnknownPropertyException` is thrown. Properties can be set multiple times and will override the old value. The keys of the properties used to retrieve and store data are compared with `Object.equals(Object)` for equality.

The listing below describes the usage of the API:

```
scriptTask("TaskName"){
  code{
      def val
      // The sessionData and executionData have the same API

      // You have multiple ways to set a value
      executionData.myCacheId = "VALUE"
      executionData.set("myCacheId", "VALUE")
      executionData["myCacheId"] =  "VALUE"
      // Or with classes for a service locator pattern
      executionData.set(Integer.class, 50)  // Possible for any Class
      executionData[Integer] = 50

      // There are the same ways to retrieve the values
      val = executionData.myCacheId
      val = executionData.get("myCacheId")
      val = executionData["myCacheId"]
      // Or with classes for a service locator pattern
      val  = executionData.get(Integer.class)
      val  = executionData[Integer]

      // You can also ask if the property exists
      boolean exists = executionData.has("myCacheId")
  }
}
```

Listing 4.45: sessionData and executionData syntax samples

### 4.4.11 ScriptAccess - Calling ScriptTasks

Sometimes it can be helpful to call other script tasks from inside your task. The `scripts{}` block or `getScripts()` method provides API to retrieve existing `IScripts` and call other `IScriptTasks` from your running `IScriptTask`.

**Note:** If you **just want to reuse code** of your own scripts in an automation script project, create a normal method containing the code and call it, instead of calling the task. The method is typesafe, has code completion support and is **much faster** than calling a script task.

**Calling script tasks** To call a task you need the name of the task and the `IScriptTaskType`. The `IScriptTaskType` determines the argument types and the return type of the script task. Then you can use `scripts.callScriptTask(String, Object...)` to call the script.

You could also use `callScriptTaskWithUserArgs(String, String, Object...)`, if you want to pass user defined arguments.

```
scriptTask("TaskName"){
    code{
        scripts.callScriptTask("OtherTask")
    }
}

scriptTask("OtherTask"){
    code{
        //Other task code
    }
}
```

Listing 4.46: Call another script task from a script task

**Calling script tasks with task arguments** If the `IScriptTaskType` requires task arguments, you have to pass the arguments to the `callScriptTask()` methods. The return value of the method is the returned value of the called script task.

```
scriptTask("TaskName", DV_PROJECT){
    code{
        def arg1 = "First argument"
        def arg2 = 5
        def result = scripts.callScriptTask("OtherTask", arg1, arg2)
        // Result contains the calculated value of OtherTask
    }
}

scriptTask("OtherTask"){
    code{arg1, arg2 ->
        return arg1 + arg2
    }
}
```

Listing 4.47: Call another script task with arguments

## 4.5 Project Handling

Project handling comprises creating new projects, opening existing projects or accessing the currently active project.

`IProjectHandlingApi` provides methods to access to the active project, for creating new projects and for opening existing projects.

`getProjects()` allows accessing the `IProjectHandlingApi` like a property.

```
scriptTask('taskName') {
  code {
    // IProjectHandlingApi is available as "projects" property
    def projectHandlingApi = projects
  }
}
```

Listing 4.48: Accessing IProjectHandlingApi as a property

`projects(Closure)` allows accessing the `IProjectHandlingApi` in a scope-like way.

```
scriptTask('taskName') {
  code {
    projects {
      // IProjectHandlingApi is available inside this Closure
    }
  }
}
```

Listing 4.49: Accessing IProjectHandlingApi in a scope-like way

### 4.5.1 Projects

Projects in the AutomationInterface are represented by `IProject` instances. These instances can be created by:

- Creating a new project
- Loading an existing project

You can only access `IProject` instances by using a `Closure` block at `IProjectHandlingApi` or `IProjectRef` class. This shall prevent memory leaks, by not closing open projects.

### 4.5.2 Accessing the active Project

The `IProjectHandlingApi` provides access to the active project. The active project is either (in descending order):

- The last `IProject` instance activated with a `Closure` block
  - Stack-based - so multiple opened projects are possible and the last (inner) `Closure` block is used.
- The passed project to a project task
- Or the loaded project in the current DvDevA in an application task

The figure 4.8 on the following page describes the behavior to search for the active project of a script task.

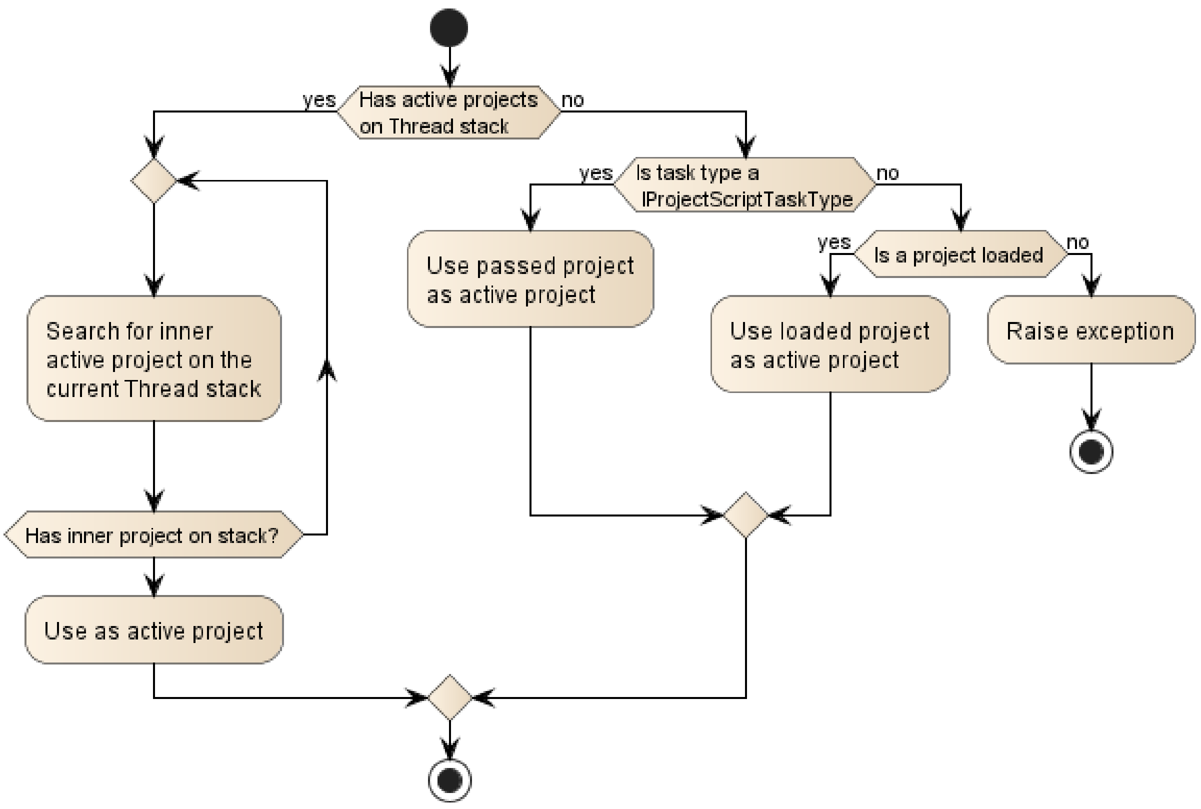

yes
Has active projects on Thread stack
no
yes
Is task type a IProjectScriptTaskType
no
Use passed project as active project
yes
Is a project loaded
no
Search for inner active project on the current Thread stack
Use loaded project as active project
Raise exception
Has inner project on stack?
Use as active project

Figure 4.8: Search for active project in getActiveProject()

It is possible that there is no active project, e.g. no project was loaded.

You can switch the active project, by calling the `with(Closure)` method on an `IProject` instance.

```
// Retrieve theProject from other API like load a project
IProject theProject = ...;
theProject.with {
    // Now theProject is the new active project inside of this closure
}
```

Listing 4.50: Switch the active project

To access the active project you can use the `activeProject(Closure)` and `getActiveProject()` methods.

```
scriptTask('taskName') {
  code {
    if (projects.projectActive) {
      // active IProject is available as "activeProject" property
      scriptLogger.info "Active project: ${projects.activeProject.projectName}"
      projects.activeProject {
        // active IProject is available inside this Closure
        scriptLogger.info "Active project: ${projectName}"
      }
    } else {
      scriptLogger.info 'No project active'
    }
  }
}
```

Listing 4.51: Accessing the active IProject

`isProjectActive()` returns `true` if and only if there is an active `IProject`. If `isProjectActive()` returns `true` it is safe to call `getActiveProject()`.

`getActiveProject()` allows accessing the active `IProject` like a property.

`activeProject(Closure)` allows accessing the active `IProject` in a scope-like way. This will enable the project specific API inside of the `Closure`.

### 4.5.3 Creating a new Adaptive Project

The method `createProject(Closure)` and the `ICreateProjectApi` are deprecated and will be removed in an upcoming release. Please use the method `createAdaptiveProject(Closure)` instead.

The method `createAdaptiveProject(Closure)` creates a new adaptive project. Inside the closure the `ICreateAdaptiveProjectApi` is available.

If a project with the same name already exists, it will not create a new project but update `ICreateAdaptiveProjectApi.filesToImport(java.util.List)` and `ICreateAdaptiveProjectApi.referencedProjects(java.util.List)` in the existing one instead. Be aware that any files or directories specified in `ICreateAdaptiveProjectApi.filesToImport(java.util.List)` will be ignored if they are already present in the created project.

If projectLocation is specified, which must be an absolute path, the project will be created in that location. If not specified, the project will be created in the default location, which is `<WORKSPACE_PATH>/<PROJECT_NAME>`. It's important to note that two projects with the same name cannot be created in different locations. If a project with the same name already exists in a different location, specifying a new projectLocation will result in an error.

The new project is not opened and usable until `IProjectRef.openProject(Closure)` is called on the returned `IProjectRef`.

Note: `createAdaptiveProject(Closure)` only works in the GUI.

#### 4.5.3.1 Example

```
import java.nio.file.Path
import java.nio.file.Paths
import com.google.common.collect.Lists

scriptTask('taskName', DV_APPLICATION) {
  code {

    def newProject = projects.createAdaptiveProject {

        name 'NewProject'
        filesToImport = importFileList  // List<Path>
        referencedProjects = referencedProjectList     // List<Path>
        projectLocation = Paths.get("D:/path/to/project/location/NewProject")

    } // createAdaptiveProject

  } // code
} // scriptTask
```

Listing 4.52: Creates a new project

#### 4.5.3.2 Project Settings

**Name** Sets the name of the new project with `name(String)` or `setName(String)`.

**FilesToImport** Set the files or folders to import via `filesToImport(List)` or `setFilesToImport(List)`. Add a list of arxml files or folders to import. If the imported file already exists in the project, it will be skipped. If the imported folder already exists in the project, it will overwrite the existing one.

**ReferencedProjects** Sets the referenced projects to link via `referencedProjects(List)` or `setReferencedProjects(List)`. Add a list of projects to reference from the project.

**ProjectLocation** Sets the location of project via `projectLocation(Path)` or `setProjectLocation(Path)`, which must be an absolute folder path. All the project files will be created directly in this folder.

### 4.5.4 Duplicating a Adaptive Project

The method `duplicateAdaptiveProject(Closure)` duplicates an adaptive project. Inside the closure the `IDuplicateAdaptiveProjectApi` is available.

In GUI mode, if a project with the same name already exists in the workspace, it throws `IllegalArgumentException` and does not duplicate the project.

The **projectLocation** and **name** are mandatory but **newProjectLocation** is optional. If **newProjectLocation** is not provided the project will be duplicated to the default location, which is `<WORKSPACE_PATH>/<PROJECT_NAME>`.

The duplicated project is not opened and usable until `IProjectRef.openProject(Closure)` is called on the returned `IProjectRef`.

#### 4.5.4.1 Example

```
import java.nio.file.Path
import com.vector.cfg.automation.scripting.api.project.IProjectRef

scriptTask('taskName', DV_APPLICATION) {
    code {projectFolderPath, newProjectFolderPath ->
            IProjectRef duplicatedProject = projects.duplicateAdaptiveProject() {
                projectLocation = projectFolderPath // Path (mandatory)
                name "newProjectName" // String (mandatory)
                newProjectLocation = newProjectFolderPath // Path (optional)

            } //duplicateAdaptiveProject
    } //code
} //scriptTask
```

Listing 4.53: Duplicating an Adaptive Project

#### 4.5.4.2 Project Settings

**Name (Mandatory)** Sets the name of the new duplicated project with `name(String)` or `setName(String)`.

**projectLocation (Mandatory)** Sets the location of project via `projectLocation(Path)` or `setProjectLocation(Path)`, which must be an absolute folder path. All the folders and files under this folder will be duplicated recursively to the newProjectLocation.

**newProjectLocation (Optional)** Sets the location of project via `newProjectLocation(Path)` or `setNewProjectLocation(Path)`, which must be an absolute folder path. If this is not provided, the project will be duplicated to the default location which is the `<WORKSPACE_PATH>/<PROJECT_NAME>`.

### 4.5.5 Change project references

The method `getProjectReferences(Path, boolean)` returns a list of the project references of the given project file. Transitive dependencies are not included if the boolean argument `recursively` is false.

The method `setProjectReferences(Path, List)` clears the existing project references of the given project file and set it to the given list.

The method `addProjectReferences(Path, List)` adds the given list to the project references of the given project file. Existing project references in the list will be ignored.

The method `removeProjectReferences(Path, List)` removes the given list from the project references of the given project file. Non-existent project references will be ignored.

```
/*
    ProjectA already depends on A,
    ProjectB already depends on B,
    ProjectC already depends on C.
*/
assert projects.getProjectReferences(ProjectC, false).size() == 1 //[C]

// Adding C is ignored because such a reference already exists
projects.addProjectReferences(ProjectC, [B, C])
assert projects.getProjectReferences(ProjectC, false).size() == 2 //[B, C]

// Removing A is ignored because such a reference does not exist
projects.removeProjectReferences(ProjectC, [A, B])
assert projects.getProjectReferences(ProjectC, false).size() == 1 //[C]

// Overwriting project references of ProjectC to be ProjectA and ProjectB
projects.setProjectReferences(ProjectC, [ProjectA, ProjectB])
assert projects.getProjectReferences(ProjectC, false).size() == 2 //[ProjectA,
    ProjectB]
assert projects.getProjectReferences(ProjectC, false).contains(ProjectA)
assert projects.getProjectReferences(ProjectC, false).contains(ProjectB)
// Getting project references recursively
assert projects.getProjectReferences(ProjectC, true).size() == 4 //[A, B, ProjectA
    , ProjectB]
assert projects.getProjectReferences(ProjectC, true).contains(A)
assert projects.getProjectReferences(ProjectC, true).contains(B)
assert projects.getProjectReferences(ProjectC, true).contains(ProjectA)
assert projects.getProjectReferences(ProjectC, true).contains(ProjectB)
```

Listing 4.54: Change Project References

### 4.5.6 Opening an existing Project

You can open an existing DaVinci project with the automation interface.

The method `openProject(Object, Closure)` opens the project at the given project file location, delegates the given code to the opened `IProject`.

The project is automatically closed after leaving the `Closure` code of the `openProject(Object, Closure)` method.

The `Object` given as project file is converted to `Path` using `ScriptConverters.TO_SCRIPT_PATH` 4.11.2 on page 123

```
scriptTask('taskName', DV_APPLICATION) {
  code {


    // Path pathToProjectFile = Paths.get(".../TestProject/.adpa/adpa.adpa")
    projects.openProject(pathToProjectFile) {

        // the opened IProject is available inside this Closure
        scriptLogger.info 'Project loaded and ready'
    }
  }
}
```

Listing 4.55: Opening a project from .dpa file

#### 4.5.6.1 Parameterized Project Load

You can also configure how a Dpa project is loaded, e.g. by disabling the generators.

The method `parameterizeProjectLoad(Closure)` returns a handle on the project specified by the given `Closure`. Using the `IOpenProjectFileApi`, the `Closure` may further customize the project's opening procedure.

The project is not opened until `openProject()` is called on the returned `IProjectRef`.

```
scriptTask('taskName', DV_APPLICATION) {
  code {
    def project = projects.parameterizeProjectLoad {


      // Path pathToProjectFile = Paths.get(".../TestProject/.adpa/adpa.adpa")
      projectFile pathToProjectFile

      // prevent activation of generators and validation
      loadGenerators false
      enableValidation false
    }

    project.openProject {
        // the opened IProject is available inside this Closure
        scriptLogger.info 'Project loaded and ready'
    }
  }
}
```

Listing 4.56: Parameterizing the project open procedure

`IOpenProjectApi` contains the methods for parameterizing the process of opening a project.

**ADPA File** The method `setProjectFile(Object)` sets the .adpa file of the project to be opened. The value given here is converted to `Path` using `ScriptConverters.TO_SCRIPT_PATH`, see 4.11.2 on page 123.

**Generators** Using `setLoadGenerators(boolean)` specifies whether or not to activate generators (including their validations) for the opened project.

**Validation** `setEnableValidation(boolean)` specifies whether or not to activate validation for the opened project.

#### 4.5.6.2 Open Project Details

`IProjectRef` is a handle on a project not yet loaded but ready to be opened. This could be used to open the project.

`IProjectRef` instances can be obtained from form the following methods:

- `IProjectHandlingApi.createProject(Closure)` 4.5.3 on page 53
- `IProjectHandlingApi.parameterizeProjectLoad(Closure)` 4.5.6 on the preceding page

The `IProject` is not really opened until `IProjectRef.openProject(Closure)` is called. Here, the project is opened and the given `Closure` is executed on the opened project. When `IProjectRef.openProject(Closure)` returns the project has already been closed.

**Advanced Open Project Use Cases** The method `IProjectRef.advanced()` provides methods for advanced usages of `IProject` instances. For example you can open a project which will not be closed when the open stack frame is left. This can be helpful for unit tests.

- `IProjectRefAdvancedUsage.openProject()`: Open the project and return the `IProject` as reference, but you have to manually close the project.

The `IProjectRefAdvancedUsage` API this only for special use cases, with have very narrow scope. If you are not sure that you need it don't use it.

### 4.5.7 Saving a Project

`IProject.saveProject()` saves the current state including all model changes of the project to disc.

```
import com.vector.cfg.model.mdf.ar4x.swcomponenttemplate.datatype.datatypes.
    MIApplicationPrimitiveDataType;
import com.vector.cfg.model.mdf.model.autosar.base.MIARPackage;

scriptTask('taskName', DV_APPLICATION) {
  code {


    // Path pathToProjectFile = Paths.get(".../TestProject/.adpa/adpa.adpa")
    def project = projects.openProject(pathToProjectFile) {

      // modify the opened project
      transaction {

           def referrableAccess = ScriptApi.activeProject.getInstance(
               IReferrableAccess)
           def myPkg = (MIARPackage) referrableAccess.getReferrableByPath("/vector
               ")
           def dstFile = project.getArxmlFile(Paths.get("MachineManifest.arxml"))

           MIApplicationPrimitiveDataType dataType2 = myPkg.element.byNameOrCreate
               (MIApplicationPrimitiveDataType.class, "NewElement", dstFile)
           dataType2.setCategory("VALUE")
      }

      // save the modified project
      saveProject()
    }
  }
}
```

Listing 4.57: Opening, modifying and saving a project

### 4.5.8 Opening AUTOSAR Files as Project

Sometimes it could be helpful to load AUTOSAR `arxml` files instead of a full-fledged DvDevA project. For example to modify the content of a file for test cases with the AutomationInterface, instead of using an XML editor.

You could load multiple `arxml` files into a temporary project, which allowed to read and write the loaded file content with the normal model APIs.

**Caution:** Some APIs and services may not be available for this type of project, like:

- Validation: The validation is disabled by default
- Generation: The generators are not loaded by default

The method `parameterizeArxmlFileLoad(Closure)` allows to load multiple arxml files into a temporary project. The given `Closure` is used to customize the project's opening procedure by the `IOpenArxmlFilesProjectApi`.

The arxml file project is not opened until `openProject()` is called on the returned `IProjectRef`.

```
scriptTask('taskName', DV_APPLICATION) {
  code {
    def project = projects.parameterizeArxmlFileLoad {
      // Add here your arxml files to load
      arxmlFiles(arxmlFilesToLoad)
      rawAutosarDataMode = true
    }
    project.openProject {
        scriptLogger.info 'Project loaded and ready'
    }
  }
}
```

Listing 4.58: Opening Arxml files as project

**Arxml Files** Add `arxml` files to load with the method `arxmlFiles(Collection)`. Multiple files and method calls are allowed. The given values are converted to `Path` instances using `ScriptConverters.TO_SCRIPT_PATH` 4.11.2 on page 123.

**Raw AUTOSAR Data Mode** the method `setRawAutosarDataMode(boolean)` specifies whether or not to use the raw ATUOSAR data model.

**Currently only this mode is supported!** You have to set `rawAutosarDataMode = true`.

Note: In raw mode most of the provided services and APIs will disabled, see below for details.

## 4.6 IArxmlFile

`IArxmlFile` represents an arxml file instance. APIs that manipulate model locations can use it as an argument. It has the following methods.

- `isReadOnly()` Checks if this is read-only. An arxml file contained in a locked project is also considered to be read-only.
- `isContainedInBSWPackage()` Checks if this is contained in BSW Package.
- `getAbsolutePath()` Gets the absolute path.
- `getRelativePath()` Gets the relative path to the ARXML folder of the current active project, which is specified in .adpa file.
- `delete()` Deletes the representing arxml file. Model located in it will also be deleted. Use this method instead of `IArxmlFile.absolutePath.toFile().delete()`.

See Splitting Model Objects 4.7.3.10 on page 77, Moving Model Objects 4.7.3.11 on page 77 and Copying Model Objects 4.7.3.12 on page 78 for related APIs.

```
scriptTask("TaskName"){
    code{
        assert lockedProject.getArxmlFiles().every { it.isReadOnly() } == true
    }
}
```

Listing 4.59: isReadOnly() returns true because project is locked

```
scriptTask("TaskName"){
    code{
        assert project.getArxmlFile(Paths.get("readOnly.arxml")).isReadOnly() ==
            true
    }
}
```

Listing 4.60: isReadOnly() returns true because file is read-only

```
def arxmlFile = project.getOrCreateArxmlFile(Paths.get("fileToBeDeleted.arxml"))

def packageElement = mdfModel(AsrPath.create("/package")) // Exists in multiple
    files
def packageElement2 = mdfModel(AsrPath.create("/packageToBeDeleted")) // Exists
    only in fileToBeDeleted.arxml

transaction {
    arxmlFile.delete()

    assert arxmlFile.absolutePath.toFile().exists() == false
    assert packageElement.isDeleted() == false
    assert packageElement2.isDeleted() == true
}
```

Listing 4.61: delete() deletes both file and model in the file

`MIObject`.getArxmlFiles() gets the arxml files where this `MIObject` instance is located.

```
MIObject obj = mdfModel(AsrPath.create("/package/Executable"))
List<IArxmlFile> = obj.getArxmlFiles()
```

Listing 4.62: Gets arxml files where MIObject is located

`IProject` provides the following methods to get or create arxml files.

- `getArxmlFolder()` Gets the absolute path of ARXML folder.
- `getArxmlFiles()` Gets all the loaded arxml files, including those located in BSW Package. Files are sorted by absolute path in alphabetical order.
- `getArxmlFile(Path)` Gets the arxml file with relative path to the ARXML folder. Returns `null` if the file cannot be found.
- `getOrCreateArxmlFile(Path)` Gets the arxml file with relative path to the ARXML folder if it exists. Creates it and returns it if the file cannot be found. Intermediate folders will be created automatically. Throws IllegalArgumentException if failed to construct a valid path. Throws UnsupportedOperationException if file creation failed.

```
scriptTask("TaskName"){
    code{
        def existFile = project.getArxmlFile(Paths.get("New_ARXML_1.arxml"))
        def nonExistFile = project.getArxmlFile(Paths.get("New_ARXML_1_not_exists.
            arxml"))
        assert existFile != null
        assert nonExistFile == null
    }
}
```

Listing 4.63: getArxmlFile(Path) returns the file only if it exists

```
scriptTask("TaskName"){
    code{
        def originProjectFileSize = project.getArxmlFiles().size()
        def existFile = project.getOrCreateArxmlFile(Paths.get("New_ARXML_1.arxml"
            ))
        def newFile = project.getOrCreateArxmlFile(Paths.get("newfolder/
            New_ARXML_1.arxml")) // newfolder is created automatically even though
             it did not exist
        assert existFile != null
        assert newFile != null
        assert originProjectFileSize + 1 == project.getArxmlFiles().size()
    }
}
```

Listing 4.64: getOrCreateArxmlFile(Path) gets or creates file

## 4.7 Model

### 4.7.1 Introduction

The model API provides the means to retrieve AUTOSAR model content and to modify AUTOSAR data.

In this chapter you'll first find a brief introduction into the model handling. Here you also find some simple cut-and-paste examples which allow starting easily with low effort. Subsequent sections describe more and more details which you can read if required.

Chapter 5 on page 135 may additionally be useful to understand detailed concepts and as a reference to handle special use cases.

### 4.7.2 Getting Started

The adaptive model API basically provides access layer for:

- The **MDF model** is the low level AUTOSAR model.
  It stores all data read from AUTOSAR XML files. Its structure is based on the AUTOSAR MetaModel which can be found for example on the AUTOSAR website. In 5.1 on page 135 you find detailed information about this model.

### 4.7.2.1 Read the SystemDescription

This section contains only one example for reading the SystemDescription by means of the MDF model. See chapter 4.7.3.1 on page 67 for more details.

```
// Required imports
import com.vector.cfg.model.mdf.ar4x.adaptiveplatform.applicationdesign.
    portinterface.*
import com.vector.cfg.model.mdf.ar4x.swcomponenttemplate.datatype.datatypes.*

scriptTask("mdfModel", DV_PROJECT){
    code {
        // Create a type-safe AUTOSAR path
        def asrPath =
            AsrPath.create("/ServiceInterfaces/MyServiceInterface_Dummy",
                MIServiceInterface)

        // Enter the MDF model tree starting at the object with this path
        mdfModel(asrPath) { MIServiceInterface serviceInterface ->

            // Traverse down to the fields (List<MIField>)
            // Execute the following for ALL elements of this list
            serviceInterface.field { MIField field ->

                // access the name of the field
                def name = field.name

                // access the data type reference
                def dataTypeRef = field.type
                assert dataTypeRef != null

                // access the data type reference target
                def dataType = dataTypeRef.refTarget
                assert dataType != null
            }
        }
    }
}
```

Listing 4.65: Read system description starting with an AUTOSAR path in closure

The same sample as above, but in property access style instead of closures:

```
// Create a type-safe AUTOSAR path
def asrPath =
    AsrPath.create("/ServiceInterfaces/MyServiceInterface_Dummy",
        MIServiceInterface)

def serviceInterface = mdfModel(asrPath)
def fieldsParam = serviceInterface.field

// Execute the following for ALL fields
fieldsParam.each{ field ->

    // access the name of the field
    def name = field.name

    // access the data type reference
    def dataTypeRef = field.type
    assert dataTypeRef != null

    // access the data type reference target
    def dataType = dataTypeRef.refTarget
    assert dataType != null
}
```

Listing 4.66: Read system description starting with an AUTOSAR path in property style

#### 4.7.2.2 Write the SystemDescription

Writing the system description looks quite similar to the reading, but you have to use methods like (see chapter 4.7.3.3 on page 70 for more details):

- `get<Element>OrCreate()` or `<element>OrCreate`
- `createAndAdd()`
- `byNameOrCreate()`

You have to open a transaction before you can modify the MDF model, see chapter 4.7.4 on page 81 for details.

The following samples show the different types of write API:

```
transaction{
    // The asrPath points to an MIField of an MIServiceInterface
    mdfModel(asrPathField) { field ->
        field.hasGetter = true
    }
}
```

Listing 4.67: Changing a simple property of an MIField

```
transaction{
    // The asrPath points to an MIServiceInterface
    mdfModel(asrPathServiceInterface) {
        int count = 0
        assert adminData == null
        adminDataOrCreate {
            count++
        }
        assert count == 1
        assert adminData != null
    }
}
```

Listing 4.68: Creating non-existing member by navigating into its content with OrCreate()

```
transaction{
    // The asrPath points to an MIServiceInterface
    mdfModel(asrPathServiceInterface) {
        introductionOrCreate(MIBlockLevelContent) { docuBlock ->
            assert docuBlock instanceof MIBlockLevelContent
        }
    }
}
```

Listing 4.69: Creating child member by navigating into its content with OrCreate() with type

```
transaction{
    // The asrPath points to an MIServiceInterface
    mdfModel(asrPathServiceInterface) {
        assert method.empty

        method.createAndAdd(MIClientServerOperation) {
            name = "NewMethod"
        }

        assert method.first.name == "NewMethod"
    }
}
```

Listing 4.70: Creating new members of child lists with createAndAdd() by type

```
transaction{
    // The path points to an MIServiceInterface
    mdfModel(asrPathServiceInterface) { serviceInterface ->
        def fieldList = serviceInterface.field

        def field = fieldList.byNameOrCreate("BoolField")
        field.name = "NewName"

        def field2 = fieldList.byNameOrCreate("NewName")

        assert field == field2
    }
}
```

Listing 4.71: Updating existing members of child lists with byNameOrCreate() by type

### 4.7.3 MDF Model in AutomationInterface

The MDF model implements the raw AUTOSAR data model and is based on the AUTOSAR meta-model. For details about the MDF model, see chapter 5.1 on page 135.

For more details concerning the methods mentioned in this chapter, you should also read the JavaDoc sections in the described interfaces and classes, see 4.8.1.1 on page 103.

Due to the unstable AUTOSAR Adaptive Standard, it is necessary to activate "allowBetaApiUsage" in the build.gradle file for the full support of the MDF model.

For more details concerning the Beta APIs, see 7.9.3.5 on page 160.

**CAUTION:** By activating the Beta API, you lose the guarantee that this script will work in future releases.

#### 4.7.3.1 Reading the MDF Model

The `mdfModel()` methods provide entry points to start navigation through the MDF model. Client code can use the `Closure` overloads to navigate into the content of the found MDF objects. Inside the called closure the related MDF object is available as closure parameter.

The following types of entry points are provided here:

- `mdfModel(TypedAsrPath)` searches an object with the specified AUTOSAR path
- `mdfModel(Class)` searches all objects with the specified model type (meta class)
- `mdfModel(String)` searches for model elements with by different properties, see 4.7.3.2 on page 70 for details.
- `mdfModel(MIObject, String)` searches for model elements by giving a root element and a relative path, see 4.7.3.2 on page 70 for details.

When a closure is being used, the object found by `mdfModel()` is provided as parameter when this closure is called:

```
code {
    // Create a type-safe AUTOSAR path for an MIServiceInterface
    def asrPath =
        AsrPath.create("/ServiceInterfaces/MyServiceInterface_Dummy",
            MIServiceInterface)

    // Use the Java-Style syntax
    def fieldsMdf = mdfModel(asrPath).field

    // Or use the Closure syntax to navigate

    // Enter the MDF model tree starting at the object with this path
    mdfModel(asrPath) {
        // Parameter type is MIServiceInterface:
        serviceInterface ->

        // Traverse down to the fields
        serviceInterface.field {MIField field ->
            println "Do something ..."
        }
    }

   saveProject()
}
```

Listing 4.72: Navigate into an MDF object starting with an AUTOSAR path

The `mdfModel()` method itself returns the found object too. Retrieving the objects member (as property) is then possible directly using the returned object.

Naming of the interface classes to create the type safe AUTOSAR path is described in chapter 5.1 on page 135.

An alternative is using a closure to navigate into the MDF object and access its member there:

```
// Get an MDF object and get its members directly
def obj = mdfModel(asrPath)     // Type MIServiceInterface
def fields = obj.field  // Type List<MIField>

// Get an MDF object and get its members using a closure
def fields2
def obj2 = mdfModel(asrPath) {
    fields2 = field
}

// The results are the same
assert obj == obj2
assert fields == fields2
```

Listing 4.73: Find an MDF object and retrieve some content data

Closures can be nested to navigate deeply into the MDF model tree:

```
mdfModel(asrPath) {
    int count = 0

    field {
        // Execute the following for ALL elements of this List
        it.type {
            // Execute for the type reference
            def dataType = it.refTarget
            println "Do something ..."
            count++
        }
    }
    assert count >= 1
}
```

Listing 4.74: Navigating deeply into an MDF object with nested closures

When a member doesn't exist during navigation into a deep MDF model tree, the specified closure is not called:

```
mdfModel(asrPath) {
    int count = 0
    assert adminData == null
    adminData {
        count++
    }
    assert count == 0
}
```

Listing 4.75: Ignoring non-existing member closures

**Retrieving a Child by Shortname** The shortname can be used to retrieve children from a MDF model object. The shortname can be used at the object with `childByName()` or at the child list with `byName()`.

**childByName** The `childByName(MIARObject, String, Closure)` method calls the passed Closure, if the request child exists. And returns the child `MIReferrable` below the specified object which has this relative AUTOSAR path (not starting with '/').

```
MIField field1 = ...
serviceInterface.childByName("FirstField"){ child->
     //Do something
}
```

Listing 4.76: Get a MIReferrable child object by name

### Lists containing Referrables

- The method `byName(String)` retrieves the child with the shortname, or `null`, if no child exists with this shortname.
- The method `byName(String, Closure)` retrieves the child with the shortname, or `null`, if no child exists with this shortname. Then the closure is executed with the child as closure parameter, if the child is not `null`. The child is finally returned.
- The method `byName(Class, String)` retrieves the child with the shortname and type, or `null`, if no child exists with this shortname.

- The method `byName(Class, String, Closure)` retrieves the child with the shortname and type, or `null`, if no child exists with this shortname. Then the closure is executed with the child as closure parameter, if the child is not `null`. The child is finally returned.
- The method `getAt(String)` all members with this relative AUTOSAR path. Groovy also allows to write `list["ShortnameToSearchFor"]`.

```
// The asrPath points to an MIServiceInterface
def serviceInterface = mdfModel(asrPath)

// byName() with shortname
def field1 = serviceInterface.field.byName("BoolField")
assert field1.name == "BoolField"

// byName() with type and shortname
def data2 = serviceInterface.field.byName(MIField, "BoolField")

// getAt() with shortname
def data3 = serviceInterface.field["BoolField"]
```

Listing 4.77: Retrieve child from list with byName()

#### 4.7.3.2 Reading the MDF Model by String

The method `mdfModel(String)` searches for model elements by multiple ways at once. The method evaluates the specified property in the following order, it will continue, if nothing was found:

- AUTOSAR path, see `mdfModel(AsrPath)`.
- Example: `mdfModel(AsrPath.create('/AUTOSAR/StdTypes/ApplicationExe/RootSwComponentProtot`
- Shortname of an `MIARElement` if the path does not contain any '/'.
- Example: `ApplicationExe`

**Relative search - `mdfModel(MIObject, String)`** Retrieves model elements based on the root element. The system navigates relative to the model element based on the root element. The relative path may not start with an '/'. In case of a variant project the collection may have more than one entry.

```
// Required imports
import com.vector.cfg.model.access.AsrPath

scriptTask("mdfModel", DV_PROJECT){
    code {
        def asrPath = AsrPath.create('/ServiceInterfaces')
        def root = mdfModel(asrPath)
        def srvIfcDummy = mdfModel(root,"MyServiceInterface_Dummy")
        println srvIfcDummy
    }
}
```

Listing 4.78: Read elements with a relative path using the mdfModel

#### 4.7.3.3 Writing the MDF Model

Writing to the MDF model can be done with the same `mdfModel(AsrPath)` API, but you have to call specific methods to modify the model objects. The methods are devided in the following use cases:

- Change a simple property like `Strings`
- Change or create a single child relateion (0:1)
- Create a new child for a child list (0:*)
- Update an existing child from a child list (0:*)

You have to open a transaction before you can modify the MDF model, see chapter 4.7.4 on page 81 for details about transactions.

#### 4.7.3.4 Simple Property Changes

The properties of MDF model object simply be changed by with the setter method of the model object. Simple setter exist for example for the types:

- `String`
- `Enums`
- `Integer`
- `Double`

```
transaction{
    // The asrPath points to an MIField of an MIServiceInterface
    mdfModel(asrPathField) { field ->
        field.hasGetter = true
    }
}
```

Listing 4.79: Changing a simple property of an MIField

#### 4.7.3.5 Creating single Child Members (0:1)

For single child members (0:1), the automation API provides an additional method for the getter `get<Element>OrCreate()` for convenient child object creation. The methods will create the element, instead of returning `null`.

```
transaction{
    // The asrPath points to an MIServiceInterface
    mdfModel(asrPathServiceInterface) {
        int count = 0
        assert adminData == null
        adminDataOrCreate {
            count++
        }
        assert count == 1
        assert adminData != null
    }
}
```

Listing 4.80: Creating non-existing member by navigating into its content with OrCreate()

If the compile time child type is not instantiable, you have to provide the concrete type by `get<Element>OrCreate(Class childType)`.

```
transaction{
    // The asrPath points to an MIServiceInterface
    mdfModel(asrPathServiceInterface) {
        introductionOrCreate(MIBlockLevelContent) { docuBlock ->
            assert docuBlock instanceof MIBlockLevelContent
        }
    }
}
```

Listing 4.81: Creating child member by navigating into its content with OrCreate() with type

It is possible to decide where you want to create the child object by `get<Element>OrCreate(IArxmlFile arxmlFile)` or `get<Element>OrCreate(Class childType, IArxmlFile arxmlFile)`.

```
transaction {
    // The asrPath points to an MIMachine
    mdfModel(asrPathMachine) {
        IArxmlFile file1 = project.getOrCreateArxmlFile(Path.of("New_ARXML_1.arxml
            "))
        defaultApplicationTimeoutOrCreate(file1) { timeout ->
            assert timeout.arxmlFiles.getFirst() == file1
        }
    }
}
```

Listing 4.82: Creating child member by navigating into its content with OrCreate() with file

```
transaction {
    // The asrPath points to an MIServiceInterface
    mdfModel(asrPathServiceInterface) {
        IArxmlFile file1 = project.getOrCreateArxmlFile(Path.of("New_ARXML_1.arxml
            "))
        introductionOrCreate(MIBlockLevelContent, file1) { docuBlock ->
            assert docuBlock instanceof MIBlockLevelContent
            assert docuBlock.arxmlFiles.getFirst() == file1
        }
    }
}
```

Listing 4.83: Creating child member by navigating into its content with OrCreate() with type and file

#### 4.7.3.6 Creating and adding Child List Members (0:*)

For child list members, the automation API provides many `createAndAdd()` methods for convenient child object creation. These method will always create the element, regardless if the same element (e.g. same ShortName) already exists.

If you want to update element see the chapter 4.7.3.7 on page 75.

```
transaction{
    // The asrPath points to an MIServiceInterface
    mdfModel(asrPathServiceInterface) {
        assert method.empty

        method.createAndAdd(MIClientServerOperation) {
            name = "NewMethod"
        }

        assert method.first.name == "NewMethod"
    }
}
```

Listing 4.84: Creating new members of child lists with createAndAdd() by type

These methods are available — but be aware that not all of these methods are available for all child lists. Adding parameters, for example, is only permitted in the parameter child list of an `MIContainer` instance.

### All Lists:

- The method `createAndAdd()` creates a new MDF object of the lists content type and appends it to this list. If the type is not instantiable the method will throw a `ModelException`. If this parent object is located in multiple files a `UnsupportedOperationException` will be thrown, as creating new elements in multiple files is not supported. The new object is finally returned.
- The method `createAndAdd(IArxmlFile)` creates a new MDF object of the lists content type and appends it to this list. If the type is not instantiable the method will throw a `ModelException`. The new object is added to the passed file. Finally the new object is returned.
- The method `createAndAdd(Closure)` creates a new MDF object of the lists content type and appends it to this list. If the type is not instantiable the method will throw a `ModelException`. If the current object is located in multiple files a `UnsupportedOperationException` will be thrown, as creating new elements in multiple files is not supported. Then the closure is executed with the new object as closure parameter. The new object is finally returned.
- The method `createAndAdd(IArxmlFile, Closure)` creates a new MDF object of the lists content type and appends it to this list. If the type is not instantiable the method will throw a `ModelException`. The new object is added to the passed file. Then the closure is executed with the new object as closure parameter. Finally the new object is returned.
- The method `createAndAdd(Class)` creates a new MDF object of the specified type and appends it to this list. The new object is finally returned.
- The method `createAndAdd(Class, IArxmlFile)` creates a new MDF object of the specified type and appends it to this list. The new object is added to the passed file. Finally the new object is returned.
- The method `createAndAdd(Class, Closure)` creates a new MDF object of the specified type and appends it to this list. Then the closure is executed with the new object as closure parameter. The new object is finally returned.
- The method `createAndAdd(Class, IArxmlFile, Closure)` creates a new MDF object of the specified type and appends it to this list. The new object is added to the passed file.

Then the closure is executed with the new object as closure parameter. Finally the new object is returned.

- The method `createAndAdd(Class, Integer)` creates a new MDF object of the specified type and inserts it to this list at the specified index position. The new object is finally returned.
- The method `createAndAdd(Class, Integer, IArxmlFile)` creates a new MDF object of the specified type and inserts it to this list at the specified index position. The new object is added to the passed file. Finally the new object is returned.
- The method `createAndAdd(Class, Integer, Closure)` creates a new MDF object of the specified type and inserts it to this list at the specified index position. Then the closure is executed with the new object as closure parameter. The new object is finally returned.
- The method `createAndAdd(Class, Integer, Closure, IArxmlFile)` creates a new MDF object of the specified type and inserts it to this list at the specified index position. The new object is added to the passed file. Then the closure is executed with the new object as closure parameter. Finally the new object is returned.

### Lists containing Referrables

- The method `createAndAdd(String)` creates a new child with the specified shortname and appends it to this list. The new object is finally returned. The used type is the lists content type. If the type is not instantiable the method will throw a `ModelException`.
- The method `createAndAdd(String, IArxmlFile)` creates a new child with the specified shortname in the given file and appends it to this list. The new object is finally returned. The used type is the lists content type. If the type is not instantiable the method will thrown a `ModelException`.
- The method `createAndAdd(String, Closure)` creates a new `MIReferrable` with the specified shortname and appends it to this list. Then the closure is executed with the new object as closure parameter. The new object is finally returned. The used type is the lists content type. If the type is not instantiable the method will throw a `ModelException`.
- The method `createAndAdd(String, IArxmlFile, Closure)` creates a new `MIReferrable` with the specified shortname in the given file and appends it to this list. Then the closure is executed with the new object as closure parameter. The new object is finally returned. The used type is the lists content type. If the type is not instantiable the method will throw a `ModelException`.
- The method `createAndAdd(Class, String)` creates a new `MIReferrable` with the specified type and shortname and appends it to this list. The new object is finally returned.
- The method `createAndAdd(Class, String, IArxmlFile)` creates a new `MIReferrable` with the specified type and shortname in the given file and appends it to this list. The new object is finally returned.
- The method `createAndAdd(Class, String, Closure)` creates a new `MIReferrable` with the specified type and shortname and appends it to this list. Then the closure is executed with the new object as closure parameter. The new object is finally returned.
- The method `createAndAdd(Class, String, IArxmlFile, Closure)` creates a new `MIReferrable` with the specified type and shortname in the given file and appends it to this list. Then the closure is executed with the new object as closure parameter. The new object is finally returned.

- The method `createAndAdd(Class, String, Integer)` creates a new `MIReferrable` with the specified type and shortname and inserts it to this list at the specified index position. The new object is finally returned.
- The method `createAndAdd(Class, String, Integer, IArxmlFile)` creates a new `MIReferrable` with the specified type and shortname in the given file and inserts it to this list at the specified index position. The new object is finally returned.
- The method `createAndAdd(Class, String, Integer, Closure)` creates a new `MIReferrable` with the specified type and shortname and inserts it to this list at the specified index position. Then the closure is executed with the new object as closure parameter. The new object is finally returned.
- The method `createAndAdd(Class, String, Integer, IArxmlFile, Closure)` creates a new `MIReferrable` with the specified type and shortname in the given file and inserts it to this list at the specified index position. Then the closure is executed with the new object as closure parameter. The new object is finally returned.

#### 4.7.3.7 Updating existing Elements

For child list members, the automation API provides many `byNameOrCreate()` methods for convenient child object update and creation on demand. These method will create the element if it does not exists, or return the existing element.

```
transaction{
    // The path points to an MIServiceInterface
    mdfModel(asrPathServiceInterface) { serviceInterface ->
        def fieldList = serviceInterface.field

        def field = fieldList.byNameOrCreate("BoolField")
        field.name = "NewName"

        def field2 = fieldList.byNameOrCreate("NewName")

        assert field == field2
    }
}
```

Listing 4.85: Updating existing members of child lists with byNameOrCreate() by type

These methods are available — but be aware that not all of these methods are available for all child lists. Updating container, for example, is only permitted in the parameter child list of an `MIContainer` instance.

#### Lists containing Referrables

- The method `byNameOrCreate(String)` retrieves the child with the passed shortname, or creates the child, if it does not exist. The shortname is automatically set before returning the new child.
- The method `byNameOrCreate(String, IArxmlFile)` retrieves the child with the passed shortname, or creates the child in the passed file, if it does not exist. The shortname is automatically set before returning the new child.
- The method `byNameOrCreate(TypedDefRef, String, Closure)` retrieves the child with the passed shortname, or creates the child, if it does not exist. The shortname is automatically set before returning the new child. Then the closure is executed with the child as closure parameter. The child is finally returned.

- The method `byNameOrCreate(TypedDefRef, String, IArxmlFile, Closure)` retrieves the child with the passed shortname, or creates the child in the passed file, if it does not exist. The shortname is automatically set before returning the new child. Then the closure is executed with the child as closure parameter. The child is finally returned.
- The method `byNameOrCreate(Class, String)` retrieves the child with the passed type and shortname, or creates the child, if it does not exist. The shortname is automatically set before returning the new child.
- The method `byNameOrCreate(Class, String, IArxmlFile)` retrieves the child with the passed type and shortname, or creates the child in the passed file, if it does not exist. The shortname is automatically set before returning the new child.
- The method `byNameOrCreate(Class, String,Closure)` retrieves the child with the passed type and shortname, or creates the child, if it does not exist. The shortname is automatically set before returning the new child. Then the closure is executed with the child as closure parameter. The child is finally returned.
- The method `byNameOrCreate(Class, String, IArxmlFile, Closure)` retrieves the child with the passed type and shortname, or creates the child in the passed file, if it does not exist. The shortname is automatically set before returning the new child. Then the closure is executed with the child as closure parameter. The child is finally returned.

#### 4.7.3.8 Deleting Model Objects

The method `delete(MIObject)` deletes the specified object from the model. This method must be called inside a transaction because it changes the model content.

```
// MIParameterValue param = ...

transaction {
    assert !param.isDeleted()
    param.delete()
    assert param.isDeleted()
}
```

Listing 4.86: Delete a parameter instance

The method `moRemove()` does the same as `delete()`. For details about model object deletion and access to deleted objects, read section 5.1.5.4 on page 139.

**IsDeleted** The `isDeleted(MIObject)` method returns `true` if the specified object has been deleted (removed) from the MDF model, or is invisible in the current active `IModelView`.

```
MIObject obj = ...
if (!obj.isDeleted()) {
    work with obj ...
}
```

Listing 4.87: Check is a model instance is deleted

Note: The return value is dependent on the current active thread and the current active `IModelView` in this thread!

The method `moIsRemoved()` does the same as `isDeleted()`.

#### 4.7.3.9 Deleting a Model Object from a file

`MIObject`.delete(IArxmlFile) deletes a `MIObject` from a file. If `MIObject` is located in multiple arxml files, only a partial object located in `modelSrcFile` will be deleted. Throws UnsupportedOperationException if project is read-only or `modelSrcFile` is read-only. Throws IllegalArgumentException if the `MIObject` is not contained in `modelSrcFile`. **An IllegalStateException will also be thrown if this is not done within a transaction**.

```
def executable = mdfModel(AsrPath.create("/package/Executable")) // Exists in
    New_ARXML_1.armxl and New_ARXML_3.arxml
def originalExecutableFilesSize = executable.getArxmlFiles().size()
def srcFile = project.getOrCreateArxmlFile(Paths.get("New_ARXML_1.arxml"))

transaction {
    executable.delete(srcFile)

    assert executable.getArxmlFiles().size() == originalExecutableFilesSize - 1
}
```

Listing 4.88: Deletes a MIObject from one file

#### 4.7.3.10 Splitting Model Objects

`MIObject`.isSplittable() checks if a `MIObject` can be split between multiple files.

```
scriptTask("TaskName"){
    code{
        def executable = mdfModel(AsrPath.create("/package/Executable"))
        assert executable.isSplittable() == true
        assert executable.adminData.isSplittable() == false
    }
}
```

Listing 4.89: isSplittable() returns true if MIObject can be split

`MIObject`.splitToFiles(Collection<IArxmlFile>, boolean, IArxmlFile) splits a `MIObject` to multiple files. Can either be used to split only the current object or recursively to split all object under the current one as well. If `modelSrcFile` is provided, only objects located in `modelSrcFile` will be split. Throws UnsupportedOperationException if the `MIObject` is not splittable, project is read-only or any of `dstFiles` is read-only. Throws IllegalArgumentException if the `MIObject` is not contained in `modelSrcFile`. **An IllegalStateException will also be thrown if this is done outside of a transaction.**

```
IArxmlFile dstFile1 = project.getOrCreateArxmlFile(Paths.get("New_ARXML_1.arxml"))
IArxmlFile dstFile2 = project.getOrCreateArxmlFile(Paths.get("New_ARXML_2.arxml"))
MIObject obj = mdfModel(AsrPath.create("/package/Executable"))
IArxmlFile modelSrcFile = obj.getArxmlFiles()[0]
obj.splitToFiles([dstFile1, dstFile2], true, modelSrcFile) // Only model in
    modelSrcFile will be split
```

Listing 4.90: Splits MIObject into multiple arxml files recursively with specified source file

#### 4.7.3.11 Moving Model Objects

`MIObject`.moveToFile(IArxmlFile, IArxmlFile) moves a `MIObject` to a destination file. If `modelSrcFile` is provided, only a partial object located in `modelSrcFile` will be moved. Throws UnsupportedOperationException if project is read-only or `dstFile` is read-only. Throws IllegalArgumentException if the `MIObject` is not contained in `modelSrcFile`. **An IllegalStateException will also be thrown if this is done outside of a transaction.**

```
def executable = mdfModel(AsrPath.create("/package/Executable")) // Exists in
    New_ARXML_1.armxl and New_ARXML_3.arxml
def srcFile = project.getOrCreateArxmlFile(Paths.get("New_ARXML_1.arxml"))
def originalExecutableFilesSize = executable.getArxmlFiles().size()
def dstFile = project.getOrCreateArxmlFile(Paths.get("subfolder/New_ARXML_1.arxml"
    ))

transaction {
    executable.moveToFile(dstFile, srcFile)

    assert originalExecutableFilesSize == executable.getArxmlFiles().size() // Now
        exists in subfolder/New_ARXML_1.armxl and New_ARXML_3.arxml
}
```

Listing 4.91: Moves a MIObject from one file to another

#### 4.7.3.12 Copying Model Objects

`MIObject`.copyTo(MIObject, IArxmlFile) copies a `MIObject` to another `MIObject` in a destination file. **This method always creates a transaction implicitly. An IllegalStateException will be thrown if this is done within an explicitly opened transaction.** Throws UnsupportedOperationException if element cannot be copied to `target`, project is read-only or `dstFile` is read-only.

```
def executable = mdfModel(AsrPath.create("/package2/Executable2"))
def packageElement = mdfModel(AsrPath.create("/package3"))
def dstFile = project.getOrCreateArxmlFile(Paths.get("subfolder/New_ARXML_1.arxml"
    ))

executable.copyTo(packageElement, dstFile)
def copiedExecutable = mdfModel(AsrPath.create("/package3/Executable2"))
assert copiedExecutable != null
assert copiedExecutable.getArxmlFiles()[0] == project.getArxmlFile(Paths.get("
    subfolder/New_ARXML_1.arxml"))
```

Listing 4.92: Copies one MIObject to another MIObject

```
scriptTask("TaskName"){
    code{
        transaction { // this will cause IllegalStateException
            def executable = mdfModel(AsrPath.create("/package2/Executable2"))
            def packageElement = mdfModel(AsrPath.create("/package3"))

            executable.copyTo(packageElement)
        }
    }
}
```

Listing 4.93: Copies with a explicit transaction throws IllegalStateException

#### 4.7.3.13 Duplicating Model Objects

The `duplicate(MIObject)` method copies (clones) a complete MDF model sub-tree and adds it as child below the same parent.

- The source object must have a parent. The clone will be added to the same MDF feature below the same parent then

- AUTOSAR UUIDs will not be cloned. The clone will contain new UUIDs to guarantee unambiguousness

This method can clone any model sub-tree, also see `IOperations.deepClone(MIObject, MIObject)` for details.

Note: This operation must be executed inside of a transaction.

```
transaction {
    MIMachine machine = mdfModel(AsrPath.create("/package/Machine"))
    MIMachineDesign machineDesign = mdfModel(AsrPath.create("/package/
        MachineDesign"))

    // machine contains reference to machineDesign
    assert machine.machineDesign.refTarget == machineDesign
    assert machineDesign.machineDesignARRefRefTargetsOwner.size() == 1

    machine.duplicate() // "/package/Machine_001"
    machineDesign.duplicate() // "/package/MachineDesign_001"

    MIMachine machine001 = mdfModel(AsrPath.create("/package/Machine_001"))
    MIMachineDesign machineDesign001 = mdfModel(AsrPath.create("/package/
        MachineDesign_001"))

    // machine001 also contains reference to machineDesign after duplication
    assert machine001.machineDesign.refTarget == machineDesign
    assert machineDesign.machineDesignARRefRefTargetsOwner.size() == 2
    assert machineDesign.machineDesignARRefRefTargetsOwner.contains(machine001.
        machineDesign)

    // machineDesign001 should not have any reference to it after duplication
    assert machineDesign001.machineDesignARRefRefTargetsOwner.size() == 0
}
```

Listing 4.94: Duplicates a MIMachine and a MIMachineDesign

#### 4.7.3.14 Special properties and extensions

**asrPath** The `getAsrPath(MIReferrable)` method returns the AUTOSAR path of the specified object.

```
MIServiceInterface serviceInterface = ...
AsrPath path = serviceInterface.asrPath
```

Listing 4.95: Get the AsrPath of an MIReferrable instance

See chapter 5.2.2 on page 140 for more details about AsrPaths.

**asrObjectLink** The `getAsrObjectLink(MIARObject)` method returns the `AsrObjectLink` of the specified object.

```
MISdServerConfig sdServerConfig = ...
AsrObjectLink link = sdServerConfig.asrObjectLink
```

Listing 4.96: Get the AsrObjectLink of an AUTOSAR model instance

See chapter 5.2.3 on page 140 for more details about AsrObjectLinks.

**ceState** The CeState is an object which aggregates states of a related MDF object. The `getCeState(MIObject)` method returns the CeState of the specified model object.

```
def serviceInterface = mdfModel('/AUTOSAR/srvIfcDummy')[0]
println serviceInterface.ceState
```

Listing 4.97: Get the CeState of a Service Interface instance

See chapter 5.2.4 on page 141 for more details about the CeState.

#### 4.7.3.15 Reverse Reference Resolution

You can resolve all references in the MDF model in the reverse direction, so you can start at a reference target and navigate to all references which point to the reference target.

**systemDescriptionObjectsPointingToMe** The method `getSystemDescriptionObjectsPointingToMe()` returns all objects located in the system description which are parent objects of references pointing to the specified target. It returns an empty collection if the object is invisible or removed.

```
List<MIObject> references =
        systemDescElement.systemDescriptionObjectsPointingToMe
```

Listing 4.98: systemDescriptionObjectsPointingToMe sample

#### 4.7.3.16 AUTOSAR Root Object

The `getAUTOSAR()` method returns the AUTOSAR root object (the root object of the MDF model tree of AUTOSAR data).

```
MIAUTOSAR root = AUTOSAR
```

Listing 4.99: Get the AUTOSAR root object

### 4.7.4 Transactions

Model changes must always be executed within a transaction. The automation API provides some simple means to execute transactions.

For details about transactions read 5.1.5 on page 138.

```
scriptTask("TaskName", DV_PROJECT){
    code {
        transaction{
            // Your transaction code here
        }
    }
}
```

Listing 4.100: Execute a transaction

```
scriptTask("TaskName", DV_PROJECT){
    code {
        transaction("Transaction name") {
            // The transactionName property is available inside a transaction
            String name = transactionName
        }
    }
}
```

Listing 4.101: Execute a transaction with a name

```
import com.vector.cfg.model.uow.TransactionException
scriptTask("TaskName", DV_PROJECT){
    code {
      try {
        transaction("Transaction") {
            // Any exception occurs
            throw new RuntimeException()
        }
      } catch (TransactionException ex) {
        assert ex.getMessage() == "Failed executing transaction 'Transaction'"
      }
    }
}
```

Listing 4.102: Handle a TransactionException

The transaction name has no additional semantic. It is only be used for logging and to improve error messages.

**Nested Transactions** If you open a transaction inside of a transaction the inner transaction is ignored and it is as no transaction call was done. So be aware that nested transactions are no real transaction, which leads to the fact the these nested transactions can not be undone.

If you want to know whether a transaction is already running, see the transactions API below.

#### 4.7.4.1 Transactions API

The Transactions API with the keyword `transactions` provides access to running transactions or the transaction history.

You can use method `isTransactionRunning()` to check if a transaction is currently running. The method returns `true`, if a transaction is running in the current `Thread`.

```
scriptTask("TaskName", DV_PROJECT){
    code {
        // Switch to the transactions API
        transactions{

            //Check if a transaction is running
            assert isTransactionRunning() == false

            // Open a transaction
            transaction{
                // Now a transaction is running
                assert isTransactionRunning() == true
            }
        }
        // Or the short form
        transactions.isTransactionRunning()
    }
}
```

Listing 4.103: Check if a transaction is running

**TransactionHistory** The transaction history API provides some methods to handle transaction undo and redo. This way, complex model changes can be reverted quite easily.

- The `undo()` method executes an undo of the last transaction. If the last transaction frame cannot be undone or if the undo stack is empty this method returns without any changes.
- The `undoAll()` method executes undo until the transaction stack is empty or an undoable transaction frame appears on the stack.
- The `redo()` method executes an redo of the last undone transaction. If the last undone transaction frame cannot be redone or if the redo stack is empty this method returns without any changes.
- The `canUndo()` method returns `true` if the undo stack is not empty and the next undo frame can be undone. This method changes nothing but you can call it to find out if the next `undo()` call would actually undo something.
- The `canRedo()` method returns `true` if the redo stack is not empty and the next redo frame can be redone. This method changes nothing but you can call it to find out if the next `redo()` call would actually redo something.
- The `clearUndoRedoHistory()` method clears the undo/redo history. After this method was called, all previous undo and redo information are lost and can not be restored.

```
scriptTask("TaskName", DV_PROJECT){
    code {
        transaction("TransactionName") {
            // Your transaction code here
        }

        transactions{
            assert transactionHistory.canUndo()

            transactionHistory.undo()

            assert !transactionHistory.canUndo()
        }
    }
}
```

Listing 4.104: Undo a transaction with the transactionHistory

```
scriptTask("TaskName", DV_PROJECT){
    code {
        transaction("TransactionName") {
            // Your transaction code here
        }

        transactions{
            transactionHistory.undo()

            assert transactionHistory.canRedo()

            transactionHistory.redo()

            assert !transactionHistory.canRedo()
        }
    }
}
```

Listing 4.105: Redo a transaction with the transactionHistory

```
scriptTask("TaskName", DV_PROJECT){
    code {
        transaction("Transaction1") {
            // Your transaction code here
        }

        transaction("Transaction2") {
            // Your transaction code here
        }

        transactions{

            assert transactionHistory.canUndo()

            transactionHistory.undo()

            assert transactionHistory.canRedo()

            assert transactionHistory.canUndo()

            transactionHistory.clearUndoRedoHistory()

            assert !transactionHistory.canRedo()

            assert !transactionHistory.canUndo()
        }
    }
}
```

Listing 4.106: Clear the undo/redo history with the transactionHistory

#### 4.7.4.2 Operations

The model operations implement convenient means to execute complex model changes like AUTOSAR module activation or cloning complete model sub-trees. The operations API is available inside of a transaction with the keyword `operation`. The class `IOperations` defines the available methods.

- The `deepClone(MIObject, MIObject)` operation copies (clones) a complete MDF model sub-tree and adds it as child below the specified parent.
  - The source object must have a parent. The clone will be added to the same MDF feature below the destination parent then
  - AUTOSAR UUIDs will not be cloned. The clone will contain new UUIDs to guarantee unambiguousness
- The method `createModelObject(Class)` creates a new element of the passed modelClass (meta class). The modelObject must be added to the whole AUTOSAR model, before finishing the transaction.
- The method `createUniqueMappedAutosarPackage()` can be used to create new `MIARPackages` in new arxml files. It creates an new instance of the specified AUTOSAR package and adds it to the model tree. All non-existing parent packages will be created too.

  The new package (including new created parent packages) will be mapped uniquely to the specified location (Path and AUTOSAR version).

### 4.7.5 Additional Model API

#### 4.7.5.1 User Annotations

In DvDevA the user can add AUTOSAR annotations to configuration elements. You can create, modify, read and delete these annotations.

All sub types of `MIHasAnnotation` elements support annotations like:

- `MIIdentifiables`
- `MISwDataDefPropsContents`
- `MISwSystemconstValues`

Although annotations are stored in the data model, their `changeable` state is independent of the configuration element `changeable` state. Annotations can be added/changed/deleted on every existing configuration element, except the project was opened in read-only mode.

The `IUserAnnotation` interface provide methods like:

- `getLabel()` - Returns the label of the annotation, like `getName()` of a container
- `setLabel()` - Changes the label
- `getText()` - Returns the text of the annotation.
- `setText()` - Changes the text
- `isChangeable()` - Returns `true`, if the annotation is changeable
- `delete()` - Deletes the annotation

**Access User Annotations** The `getUserAnnotations(MIARObject)` method returns the `IUser-Annotations` for the model element. The returned list provides additional methods defined in `IUserAnnotationList`.

```
// We already have the MIPortInterface "configElement" or any other model element
def myServiceInterface = configElement

def annos = myServiceInterface.userAnnotations // Retrieve the list of annos
def anno = annos.byLabel("MyLabel")      // Select the annotation with "MyLabel"
def text = anno.text

// Or short
text = myServiceInterface.userAnnotations["MyLabel"].text
```

Listing 4.107: Get a UserAnnotation of a CE

**Creation and Modification of User Annotations** You can create new User Annotations with the methods:

- `createAndAdd(label)`
- `byLabelOrCreate(label)`

```
transaction{
    // We already have the configuration element "configElement"
    def anno = configElement.userAnnotations.createAndAdd("MyAnno")
    anno.text = "My Text"
}
```

Listing 4.108: Create a new UserAnnotation

```
transaction{
    // We already have the container "configElement"
    def anno = configElement.userAnnotations.byLabelOrCreate("MyAnno")
    anno.text = "My Text"
}
```

Listing 4.109: Create or get the existing UserAnnotation by label name

**Notes** The `IUserAnnotationList` is updated, when the underlying model changes.

The `IUserAnnotationList` is read only list and does not permit any modify operations defined in `java.util.List`, but certain operations like `createAndAdd(String)` will affect the list content. If you delete a contained `IUserAnnotation` the list will not be updated.

### 4.7.6 Model Extension API

#### 4.7.6.1 Strict Mode

Model Extension API is running in Strict Mode by default, which means the corresponding definition must exist when accessing the data of model extension. It prevents user reading or writing data that does not comply with the model extension definition. Model Extension API is **NOT** running in Strict Mode only for IProjectScriptTaskType `DV_MEX_MIGRATION`.

#### 4.7.6.2 Groovy API

For any **MIHasAdminData** instance, use the following methods to access its model extension, which is stored in *ADMIN-DATA*.

- mexExtension(MIHasAdminData, String)
- mexExtensionOrCreate(MIHasAdminData, String)
- mexExtensionOrCreate(MIHasAdminData, String, IArxmlFile)

`mexExtension(MIHasAdminData, String)` method returns the extension `IMexExtension` with specified name below the specified `MIHasAdminData` object. Returns null if it does not exist.

```
MIExecutable executable = ...
IMexExtension extension = executable.mexExtension("SdgClassName")
```

Listing 4.110: Get a IMexExtension object by extension name

`mexExtensionOrCreate(MIHasAdminData, String, IArxmlFile)` method returns the extension `IMexExtension` with specified name below the specified `MIHasAdminData` object. Creates a new one in the given arxml file and returns it if not already exists.

```
MIExecutable executable = ...
IArxmlFile arxmlFile = project.getOrCreateArxmlFile(Path.of("New_ARXML.arxml"))
IMexExtension extension = executable.mexExtensionOrCreate("SdgClassName",
    arxmlFile)
```

Listing 4.111: Create a IMexExtension object by extension name in the arxml file

#### 4.7.6.3 Model Extension

`IMexExtension` represents a model extension instance. Use `getName()` to get the name of sdg class, which it implements. Use `getVersion()` and `setVersion(String)` to read and write version of model extension. Use `getParent()` to get the `MIHasAdminData` object, which holds the model extension data.

- `delete()` Deletes itself. Throws InvalidObjectException if any of its ancestor models has been removed.
- `getName()` Gets the name.
- `getParent()` Gets the parent, a `MIHasAdminData`.
- `getVersion()` Gets the version. If multiple versions are found, the lowest version is returned. Returns Null if it does not exist any.
- `setVersion(String)` Sets the version. Throws IllegalArgumentException if argument is null or invalid.

Use the following methods to traverse deeper in the `IMexExtension` instance.

- `allAttributes(String)` Gets all attributes in a `List` by given attribute name. Returns an empty `List` if not found.
- `attribute(String)` Gets the first attribute by given attribute name. Returns null if not found.
- `attributeOrCreate(String)` Gets the first attribute by given attribute name. Creates a new attribute and returns it if not found.
- `attributeCreateAndAdd(String)` Creates a new attribute by given attribute name and append it. Returns the created attribute.
- `allReferences(String)` Gets all references in a `List` by given reference name. Returns an empty `List` if not found.
- `reference(String)` Gets the first reference by given reference name. Returns null if not found.
- `referenceOrCreate(String)` Gets the first reference by given reference name. Creates a new reference and returns it if not found.
- `referenceCreateAndAdd(String)` Creates a new reference by given reference name and append it. Returns the created attribute.
- `allChildObjects(String)` Gets all child objects in a `List` by given child relation name. Returns an empty `List` if not found.
- `childObject(String)` Gets the first child object by given child relation name. Returns null if not found.
- `childObjectCreateAndAdd(String, String)` Creates a child object by given child relation name and sdg class name. Appends the created child object and return it.
- `childObjectCreateAndAdd(String)` Creates a child object by given child relation name. Appends the created child object and return it. Only works in Strict Mode.
- `childObjectWithNameCreateAndAdd(String, String)` Creates a child object by given child relation name and short name of child object. Appends the created child object and return it. Only works in Strict Mode.

```
scriptTask('taskName') {
  code {
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    assert ext.parent == executable
  }
}
```

Listing 4.112: parent of IMexExtension

```
scriptTask('taskName') {
  code {
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    transaction {
        ext.setVersion("1.0.3")
    }
    assert ext.version == "1.0.3"
    assert ext.versionOrDefault == com.vector.cfg.util.version.Version.valueOf("
       1.0.3")
  }
}
```

Listing 4.113: setVersion(String) and getVersion() of IMexExtension

```
scriptTask('taskName') {
  code {param1 ->
    def attributeName = "myIntAttribute"
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    assert ext.allAttributes(attributeName).size() > 0
  }
}
```

Listing 4.114: get all attributes of IMexExtension

```
scriptTask('taskName') {
  code {param1 ->
    def referenceName = param1
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    ext.reference(referenceName) // throws IllegalArgumentException
  }
}
```

Listing 4.115: get a non-existing reference of IMexExtension in Strict Mode

```
scriptTask('taskName', DV_MEX_MIGRATION) {
  code {ext ->
    def referenceName = "myForeignReference_NotExists"
    assert ext.reference(referenceName) == null
  }
}
```

Listing 4.116: get a non-existing reference of IMexExtension in Non-Strict Mode

```
scriptTask('taskName') {
  code {
    def childRelationName = "childRelation"
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    assert ext.childObject(childRelationName) != null
  }
}
```

Listing 4.117: get an existing child object of IMexExtension

```
scriptTask('taskName') {
  code {
    def childRelationName = "childRelation"
    def childClassName = "ExtensionClassCaptionTrue"
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    transaction {
        def originalSize = ext.allChildObjects(childRelationName).size()
        def object = ext.childObjectCreateAndAdd(childRelationName, childClassName
            )
        def objects = ext.allChildObjects(childRelationName)
        def size = objects.size()
        assert originalSize + 1 == size

    }
  }
}
```

Listing 4.118: create and add a child object of IMexExtension

```
scriptTask('taskName') {
  code {
    def childRelationName = "childRelation"
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    transaction {
        def originalSize = ext.allChildObjects(childRelationName).size()
        def object = ext.childObjectWithNameCreateAndAdd(childRelationName, "
            ExtensionClass_500")
        def objects = ext.allChildObjects(childRelationName)
        def size = objects.size()
        def name = object.name
        assert originalSize + 1 == size
        assert name == "ExtensionClass_500"

    }
  }
}
```

Listing 4.119: create and add a child object of IMexExtension

### 4.7.6.4 Primitive Attribute

`IMexAttribute` represents a primitive type attribute in model extension. Use `getValue()` and `setValue(String)` to read and write attribute. Use `getParent()` to get its parent.

- `delete()` Deletes itself. Throws InvalidObjectException if any of its ancestor models has been removed.
- `getParent()` Gets the parent, a `IMexExtension` or `IMexObject` instance.
- `getValue()` Gets the value in `String` type.
- `setValue(String)` Sets the value. Throws IllegalArgumentException if newValue is null.

Following utility methods are provided for type-safe value access.

- `getIntValue()` Returns `BigInteger`. Throws NumberFormatException if failed to convert value to a `BigInteger`.
- `getFloatValue()` Returns `BigDecimal`. Throws NumberFormatException if is not a valid representation of a BigDecimal according to the AUTOSAR rules.
- `getBoolValue()` Returns `Boolean`.

```
scriptTask('taskName') {
  code {
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    def attr = ext.attribute("myIntAttribute")
    def valueOrigin = attr.getIntValue()
    transaction {
        attr.setValue("10")
        def value1 = attr.getIntValue()
        assert value1 == 10
        attr.setValue(valueOrigin.toString())
        def value2 = attr.intValue
    }
  }
}
```

Listing 4.120: setValue(String) and getIntValue() of IMexAttribute

```
scriptTask('taskName') {
  code {param1 ->
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    def attr = ext.attribute(param1)
    transaction {
        attr.setValue("0xff")
        def HEX_0xff = attr.getFloatValue()
        assert HEX_0xff == 0xff

        attr.setValue("0b11")
        def BINARY_0b11 = attr.getFloatValue()
        assert BINARY_0b11 == 0b11

        attr.setValue("077")
        def OCTAL_077 = attr.getFloatValue()
        assert OCTAL_077 == OCTAL_077

        attr.setValue("-.0")
        def NEGATIVE_ZERO = attr.getFloatValue()
        assert NEGATIVE_ZERO == 0

        attr.setValue("-INF")
        def NEGATIVE_INFINITY = attr.getFloatValue()
        assert NEGATIVE_INFINITY == com.vector.cfg.util.math.MBigDecimal.
            NEGATIVE_INFINITY
        assert com.vector.cfg.util.math.MBigDecimal.isInfinite(NEGATIVE_INFINITY)

        attr.setValue("NaN")
        def NOT_A_NUMBER = attr.getFloatValue()
        assert NOT_A_NUMBER == com.vector.cfg.util.math.MBigDecimal.NaN
        assert com.vector.cfg.util.math.MBigDecimal.isNaN(NOT_A_NUMBER)

        attr.setValue("123.123")
        def FLOAT_1 = attr.getFloatValue()
        attr.setValue("-123.123")
        def FLOAT_2 = attr.getFloatValue()
        assert FLOAT_1 + FLOAT_2 == 0
        assert FLOAT_1 - FLOAT_2 == 246.246
    }
  }
}
```

Listing 4.121: getFloatValue() of IMexAttribute

#### 4.7.6.5 Reference Attribute

`IMexReference` represents a reference type attribute in model extension. Use `getValue()` and `setValue(String)` to read and write reference. Use `getParent()` to get its parent.

- `delete()` Deletes itself. Throws InvalidObjectException if any of its ancestor models has been removed.
- `getParent()` Gets the parent, a `IMexExtension` or `IMexObject` instance.
- `getValue()` Gets the value in `String` type.
- `setValue(String)` Sets the value. Throws IllegalArgumentException if newValue is null or empty. *
- `getAsrPath()` Gets the AUTOSAR path `AsrPath` it references to.

- `setValue(AsrPath)` Sets the value. Throws IllegalArgumentException if newValue is not a valid AUTOSAR path.
- `getRefTarget()` Gets the `MIReferrable` this refers to.
- `setReferenceTarget(IMexObject)` Sets the reference target. Throws IllegalArgumentException if mexObject is null or is not referrable.
- `setReferenceTarget(MIReferrable)` Sets the reference target. Throws IllegalArgumentException if referrable is null.
- `getRefTargets()` Gets a `List` of `MIReferrable`s this refers to. Returns an empty `List` if not found.
- `getMexRefTarget()` Gets the `IMexObject` this refers to. Returns null if the target cannot be converted to a `IMexObject`.
- `getMexRefTargets()` Gets a `List` of `IMexObjects` this refers to. Returns an empty `List` if not found.

```
scriptTask('taskName') {
  code {param1, param2 ->
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    def foreignRef = ext.reference(param1)
    def ref = ext.reference(param2)
    def foreignRefAsrPath = foreignRef.asrPath
    def refAsrPath = ref.asrPath
    assert foreignRefAsrPath instanceof AsrPath
    assert refAsrPath.getAutosarPathString() instanceof String
  }
}
```

Listing 4.122: asrPath of IMexReference

```
scriptTask('taskName') {
  code {
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    def foreignRef = ext.reference("myForeignReference")
    def ref = ext.reference("myReference")
    def foreignRefTarget = foreignRef.refTarget
    def refTarget = ref.refTarget
    assert foreignRefTarget instanceof com.vector.cfg.model.mdf.commoncore.autosar
        .MIReferrable
    assert refTarget instanceof com.vector.cfg.model.mdf.commoncore.autosar.
        MIReferrable
  }
}
```

Listing 4.123: refTarget of IMexReference

```
scriptTask('taskName') {
  code {
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    def foreignRef = ext.reference("myForeignReference")
    assert foreignRef.asrPath.autosarPathString == "/MyPackage/MyServiceInterface"
    transaction {
        foreignRef.setReferenceTarget(mdfModel("/MyPackage/MyServiceInterface_001"
            ))
        assert foreignRef.asrPath.autosarPathString == "/MyPackage/
            MyServiceInterface_001"
    }
  }
}
```

Listing 4.124: setRefTarget(MIReferrable) of IMexReference

```
scriptTask('taskName') {
  code {
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    def foreignRef = ext.reference("myForeignReference")
    def ref = ext.reference("myReference")
    def foreignMexRefTarget = foreignRef.mexRefTarget
    def mexRefTarget = ref.mexRefTarget
    assert foreignMexRefTarget == null
    assert mexRefTarget instanceof com.vector.cfg.model.asr.extension.instance.
        access.IMexObject
  }
}
```

Listing 4.125: mexRefTarget of IMexReference

```
scriptTask('taskName') {
  code {
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    def ref = ext.reference("myReference")
    assert ref.asrPath.autosarPathString == "/MyPackage/MyExecutable/
        ExtensionClass_100"
    def mexObject200 = ext.allChildObjects("childRelation").get(1)
    transaction {
        ref.setReferenceTarget(mexObject200)
        assert ref.asrPath.autosarPathString == "/MyPackage/MyExecutable/
            ExtensionClass_200"
    }
  }
}
```

Listing 4.126: setRefTarget(IMexObject) of IMexReference

### 4.7.6.6 Child Relation Attribute

`IMexObject` represents a child relation attribute in model extension. Use `getName()` and `setName(String)` to read and write its name. Use `getAsrPath()` to get its AUTOSAR path. Use `getParent()` to get its parent.

**Note: All `IMexObject` belongs to the same extended class instance share the same**

**namespace, even if they locate in different nested levels.**

```
{@code
...
<EXECUTABLE UUID="...">
  <SHORT-NAME>MyExecutable</SHORT-NAME>
  <ADMIN-DATA>
    <SDGS>
      <SDG GID="DvMex:MyExtension">
        <SDG GID="childRelation">
          <SDG GID="ExtensionClass1">
            <SDG GID="childRelation">
              <SDG GID="ExtensionClass2">
                <!-- AUTOSAR path: ../MyExecutable/InnerChild -->
                <SDG-CAPTION>
                  <SHORT-NAME>InnerChild</SHORT-NAME>
                </SDG-CAPTION>
              </SDG>
            </SDG>
          </SDG>
        </SDG>
      </SDG>
    </SDGS>
  </ADMIN-DATA>
</EXECUTABLE>
...
}
```

- `delete()` Deletes itself. Throws InvalidObjectException if any of its ancestor models has been removed.
- `getAsrPath()` Gets the AUTOSAR path `AsrPath`. Returns null if `isReferrable` is false.
- `getAllReferencesPointingTo()` Gets all the references `MIARRef` pointing to this. Returns an empty `List` if not found. Throws IllegalArgumentException if `isReferrable` is false.
- `getAllReferencesPointingToSorted()` Gets all the sorted references `MIARRef` pointing to this. Returns an empty `List` if not found. Throws IllegalArgumentException if `isReferrable` is false.
- `getAllMexReferencesPointingTo()` Gets all the MEX references `IMexReference` pointing to this. Returns an empty `List` if not found. Throws IllegalArgumentException if `isReferrable` is false.
- `getAllMexReferencesPointingToSorted()` Gets all the sorted MEX references `IMexReference` pointing to this. Returns an empty `List` if not found. Throws IllegalArgumentException if `isReferrable` is false.
- `getName()` Gets the name.
- `setName(String)` Sets the name.
- `getParent()` Gets the parent, a `IMexExtension` or `IMexObject` instance.
- `isReferrable()` Gets if this is referrable. Returns true, if this has `MISdgCaption` in its `MISdg`.

Use the following methods to traverse deeper in the `IMexObject` instance.

- `allAttributes(String)` Gets all attributes in a `List` by given attribute name. Returns an empty `List` if not found.
- `attribute(String)` Gets the first attribute by given attribute name. Returns null if not found.
- `attributeOrCreate(String)` Gets the first attribute by given attribute name. Creates a new attribute and returns it if not found.
- `attributeCreateAndAdd(String)` Creates a new attribute by given attribute name and append it. Returns the created attribute.
- `allReferences(String)` Gets all references in a `List` by given reference name. Returns an empty `List` if not found.
- `reference(String)` Gets the first reference by given reference name. Returns null if not found.
- `referenceOrCreate(String)` Gets the first reference by given reference name. Creates a new reference and returns it if not found.
- `referenceCreateAndAdd(String)` Creates a new reference by given reference name and append it. Returns the created attribute.
- `allChildObjects(String)` Gets all child objects in a `List` by given child relation name. Returns an empty `List` if not found.
- `childObject(String)` Gets the first child object by given child relation name. Returns null if not found.
- `childObjectCreateAndAdd(String, String)` Creates a child object by given child relation name and sdg class name. Appends the created child object and return it.
- `childObjectCreateAndAdd(String)` Creates a child object by given child relation name. Appends the created child object and return it. Only works in Strict Mode.
- `childObjectWithNameCreateAndAdd(String, String)` Creates a child object by given child relation name and short name of child object. Appends the created child object and return it. Only works in Strict Mode.

```
scriptTask('taskName') {
  code {
    def childRelationName = "childRelation"
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    def mexObject200 = ext.allChildObjects(childRelationName).get(1)
    def mexObject210 = mexObject200.childObject(childRelationName)
    def mexObject211 = mexObject210.childObject(childRelationName)
    def mexObject200Parent = mexObject200.parent
    def mexObject210Parent = mexObject210.parent
    def mexObject211Parent = mexObject211.parent
    assert mexObject200Parent == ext
    assert mexObject210Parent == mexObject200
    assert mexObject211Parent == mexObject210
  }
}
```

Listing 4.127: parent of IMexObject

```
scriptTask('taskName') {
  code {
    def childRelationName = "childRelation"
    def childRelationName2 = "childRelation2"
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    def mexObject100 = ext.allChildObjects(childRelationName).get(0)
    def mexObject200 = ext.allChildObjects(childRelationName).get(1)
    def mexObject300 = ext.allChildObjects(childRelationName).get(2)
    def mexObject400 = ext.allChildObjects(childRelationName2).get(0) // No
        MISdgCaption in it
    def mexObject410 = mexObject400.childObject(childRelationName2) // No
        MISdgCaption in it
    def mexObject411 = mexObject410.childObject(childRelationName)
    def mexObject100AsrPath = mexObject100.asrPath
    def mexObject200AsrPath = mexObject200.asrPath
    def mexObject300AsrPath = mexObject300.asrPath
    def mexObject400AsrPath = mexObject400.asrPath
    def mexObject410AsrPath = mexObject410.asrPath
    def mexObject411AsrPath = mexObject411.asrPath
    assert mexObject100AsrPath.autosarPathString == "/MyPackage/MyExecutable/
        ExtensionClass_100"
    assert mexObject200AsrPath.autosarPathString == "/MyPackage/MyExecutable/
        ExtensionClass_200"
    assert mexObject300AsrPath.autosarPathString == "/MyPackage/MyExecutable/
        ExtensionClass_300"
    assert mexObject400AsrPath == null
    assert mexObject410AsrPath == null
    assert mexObject411AsrPath.autosarPathString == "/MyPackage/MyExecutable/
        ExtensionClass_411"
  }
}
```

Listing 4.128: asrPath of IMexObject

```
scriptTask('taskName') {
  code {
    def childRelationName = "childRelation"
    def childRelationName2 = "childRelation2"
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    def mexObject400 = ext.allChildObjects(childRelationName2).get(0) // No
        MISdgCaption in it
    def mexObject410 = mexObject400.childObject(childRelationName2) // No
        MISdgCaption in it
    def mexObject411 = mexObject410.childObject(childRelationName)
    def allReferencesPointingToSorted = mexObject411.
        getAllReferencesPointingToSorted()
    def ref_411_1 = allReferencesPointingToSorted.get(0)
    def ref_411_2 = allReferencesPointingToSorted.get(1)
    assert allReferencesPointingToSorted.size() == 2
    assert ref_411_1 instanceof com.vector.cfg.model.mdf.commoncore.autosar.
        MIARRef
    assert ref_411_2 instanceof com.vector.cfg.model.mdf.commoncore.autosar.
        MIARRef
  }
}
```

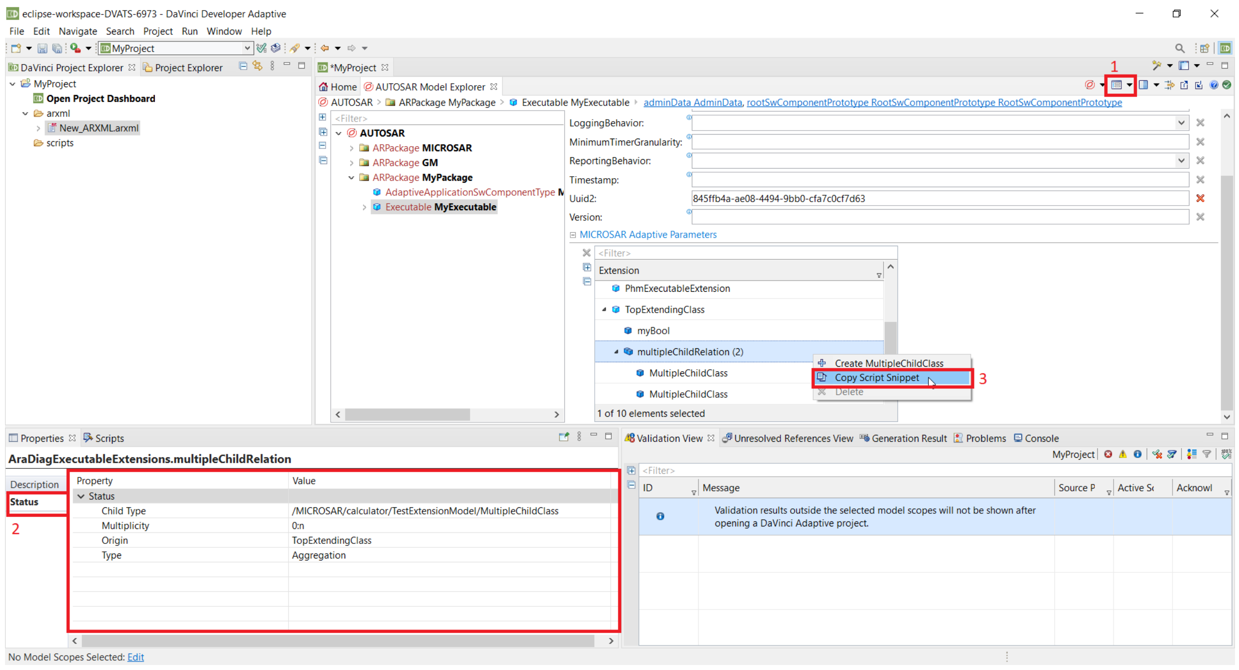
Listing 4.129: allReferencesPointingToSorted of IMexObject

```
scriptTask('taskName') {
  code {
    def childRelationName = "childRelation"
    def childRelationName2 = "childRelation2"
    def asrPath = AsrPath.create("/MyPackage/MyExecutable")
    def executable =  mdfModel(asrPath)
    def ext = executable.mexExtension("MyExecutableExtension")
    def mexObject400 = ext.allChildObjects(childRelationName2).get(0) // No
       MISdgCaption in it
    def mexObject410 = mexObject400.childObject(childRelationName2) // No
       MISdgCaption in it
    def mexObject411 = mexObject410.childObject(childRelationName)
    def allMexReferencesPointingTo = mexObject411.getAllMexReferencesPointingTo()
    def mexRef_411_1 = allMexReferencesPointingTo.get(0)
    def mexRef_411_2 = allMexReferencesPointingTo.get(1)
    assert allMexReferencesPointingTo.size() == 2
    assert mexRef_411_1 instanceof com.vector.cfg.model.asr.extension.instance.
       access.IMexReference
    assert mexRef_411_2 instanceof com.vector.cfg.model.asr.extension.instance.
       access.IMexReference
    assert mexRef_411_1.mexRefTarget == mexObject411
    assert mexRef_411_2.mexRefTarget == mexObject411
  }
}
```

Listing 4.130: allMexReferencesPointingTo of IMexObject

### 4.7.6.7 Find More Info of Model Extension

In the **Form View**(point 1) of **AUTOSAR Model Explorer**, you can find a tree structure of model extension, which belongs to the selected MIHasAdminData; in our example, a MIExecutable. In the **Status tab** of **Properties View**(point 2), it shows properties of selected tree node, e.g. multiplicity, type and origin. In the **context menu**(point 3) of a tree node, you can even copy the script snippet for accessing it.



| | |
|---|---|
| LoggingBehavior: | |
| MinimumTimerGranularity: | |
| ReportingBehavior: | |
| Timestamp: | |
| Uuid2: | 845ffb4a-ae08-4494-9bb0-cfa7c0cf7d63 |
| Version: | |

AraDiagExecutableExtensions.multipleChildRelation

| Property | Value |
|---|---|
| Status | |
| Child Type | /MICROSAR/calculator/TestExtensionModel/MultipleChildClass |
| Multiplicity | 0:n |
| Origin | TopExtendingClass |
| Type | Aggregation |

| ID | Message | Source P | Active S | Acknowl |
|---|---|---|---|---|
| | Validation results outside the selected model scopes will not be shown after opening a DaVinci Adaptive project. | | | |

Figure 4.9: Copy Script Snippet of Model Extension Data

### 4.7.7 Wizard API

Some elements are hard to create with consistent configuring of all related references. Wizard API provides an easier way to configure important attributes and references in one go.

#### 4.7.7.1 Create a MIARPackage

The method `createPackage(Closure)` creates a `MIARPackage`. Inside the closure the `ICreatePackageApi` is available.

```
transaction {
    def pkg = createPackage {
        arPackage = mdfModel(AsrPath.create("/package")) // MIARPackage (optional)
        name = "nestedPackage" // String (mandatory)
        file = project.getOrCreateArxmlFile(Path.of("newArxml1.arxml")) //
            IArxmlFile (mandatory)
    }
}
```

Listing 4.131: Create a MIARPackage

```
transaction {
    def pkg = createPackage {
        name = "rootPackage1"
        file = project.getOrCreateArxmlFile(Path.of("newArxml1.arxml"))
    }
}
```

Listing 4.132: Create a MIARPackage under AUTOSAR root object

##### Settings

**arPackage (Optional)** Sets the parent package of the `MIARPackage` to create. `MIARPackage` will be created under AUTOSAR root object if arPackage is null.

**name (Mandatory)** Sets the shortname of the element to create. Cannot be null. It must comply with the pattern '**[a-zA-Z][a-zA-Z0-9_]***'.

**file (Mandatory)** Sets the arxml file to store the newly created element. Cannot be null.

#### 4.7.7.2 Create a MIServiceInterfaceDeployment

The method `createServiceInterfaceDeployment(Closure)` creates a `MIServiceInterfaceDeployment`. Inside the closure the `ICreateServiceInterfaceDeploymentApi` is available.

```
transaction {
    def arxmlFile = project.getOrCreateArxmlFile(Path.of("newArxml1.arxml"))
    def newPackage = createPackage {
        name = "newPackage"
        file = arxmlFile
    }
    def serviceInterface1 = newPackage.element.createAndAdd(MIServiceInterface, "
        serviceInterface1", arxmlFile)
    def serviceInterface2 = newPackage.element.createAndAdd(MIServiceInterface, "
        serviceInterface2", arxmlFile)

    def someipServiceInterfaceDeployments = createServiceInterfaceDeployment {
        arPackage = newPackage // MIARPackage (mandatory)
        name = "ServiceInterfaceDeployment" // String (mandatory)
        file = project.getOrCreateArxmlFile(Path.of("newArxml2.arxml")) //
            IArxmlFile (mandatory)
        serviceInterfaces = [serviceInterface1, serviceInterface2] // List<
            MIServiceInterface> (mandatory)
        type = ICreateServiceInterfaceDeploymentApi.SOMEIP // String (mandatory)
        transportLayerProtocol = MITransportLayerProtocolEnum.TCP //
            MITransportLayerProtocolEnum (optional)
    }
}
```

Listing 4.133: Create a list of MIServiceInterfaceDeployments

### Settings

**arPackage (Mandatory)** Sets the AUTOSAR package of the element to create. Cannot be null.

**name (Mandatory)** Sets the shortname of the element to create. Cannot be null. It must comply with the pattern '**[a-zA-Z][a-zA-Z0-9_]***'.

**file (Mandatory)** Sets the arxml file to store the newly created element. Cannot be null.

**serviceInterfaces (Mandatory)** Sets the service interfaces of the service interface deployments to reference to. Cannot be null or empty.

**type (Mandatory)** Sets the type of the service interface deployments to create.
Supported value:

- `ICreateServiceInterfaceDeploymentApi.SOMEIP`
- "IpcBinding" if the corresponding MEX definition can be found

**transportLayerProtocol (Optional)** Sets the transport layer protocol of the service interface deployment to create. Only applicable when type is `ICreateServiceInterfaceDeploymentApi.SOMEIP`.

## 4.8 Generation

### 4.8.1 Code Generation

The block **generation** encapsulates all settings and commands which are related to code generation:

The basic structure is the following:

```
generation{
    settings{
        // Settings like the selection of generators or scopes for execution are
           done here
        selectGeneratorByDefRef("/ADAPTIVE/MICROSAR/amsr_logapi_config")
        selectScope("/OneCompany/AppExample/LightControl/Deployment/
           LightControlSwCluster")
    }
    // The execution of the generation or validation can be started here
    generate()
}
```

Listing 4.134: Basic structure of generation

#### 4.8.1.1 Generation Settings

The class `IGenerationSettingsApi` encapsulates all settings which belong to a generation process. E.g.

- Select the generators to execute
- Select the scopes to execute
- Set the output folder of generated files
- Select to create a generation report

**Generation with No Settings** If no settings are provided, all the available generators and scopes will be selected by default.

```
scriptTask("generate_with_no_settings"){
    code{
        generation{
            // All the available generators and scopes are selected if no setting
               are provided
            generate()
        }
    }
}
```

Listing 4.135: Generate with no settings

There are two ways to open a settings block:

- **settings**
  - This keyword creates empty settings. E.g. **no generator or scope** is selected by default.
- **settingsFromProject**

  – This keyword takes the project settings as a template. E.g. **generators** from the project settings are initially activated and can optionally be refined by explicit selections. **No scope** is selected by default.

**Generation with Project Settings** The following snippet executes a validation with the project settings, generator selection is read from the .adpa file.

```
scriptTask("validate_with_default_settings"){
    code{
        generation{
            settings{
                // A new settings closure has no scope selected by default
                selectAllScopes()

                // Select generators according to .adpa file
                loadProjectSettings()
            }
            validate()
        }
    }
}
```

Listing 4.136: Validate with project settings

To execute a generation with the project settings the following snippet can be used. The validation is executed implicitly before the generation because of AUTOSAR requirements.

```
scriptTask("generate_with_project_settings"){
    code{
        generation{
            settings{
                selectAllScopes()
                loadProjectSettings()
            }
            generate() // validate() is also executed in this call
        }
    retValue
    }
}
```

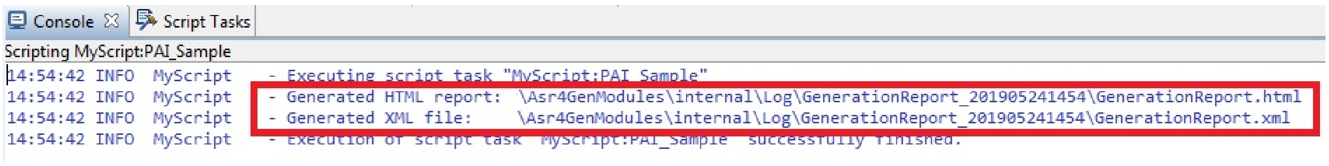
Listing 4.137: Generate with project settings

**Set output folder** To customize an output folder of generated files the following snippet can be used. If no **genDataDir** is provided, the default output folder is `<PROJECT_FOLDER>/src-gen`.

```
scriptTask("generate_with_genDataDir"){
    code{
        generation{
            settings {
                // A new settings closure has no scope selected by default
                selectAllScopes()

                // A new settings closure has no generator selected by default
                selectAll()
                genDataDir = project.projectFilePath.parent.parent.resolve("new-
                    src-gen")
            }

            generate()
        }
    }
}
```

Listing 4.138: Generate with a customized output folder

**Generation with Reporting Settings** `IGenerationReportApi` is the entry point for generation report settings. When the settings are set and generation has been finished, the report output path can be seen in the "Console View".



```
14:54:42 INFO  MyScript   - Executing script task "MyScript:PAI_Sample"
14:54:42 INFO  MyScript   - Generated HTML report:  \Asr4GenModules\internal\Log\GenerationReport_201905241454\GenerationReport.html
14:54:42 INFO  MyScript   - Generated XML file:     \Asr4GenModules\internal\Log\GenerationReport_201905241454\GenerationReport.xml
14:54:42 INFO  MyScript   - Execution of script task "MyScript:PAI_Sample" successfully finished.
```

Figure 4.10: Report Output Path

The following snippet sets the report settings and executes a generation.

```
scriptTask("generate_components_with_report"){
    code{
        generation{
            settings {

                selectGeneratorsByDefRef("/ADAPTIVE/MICROSAR/amsr_applicationbase"
                    )
                selectGeneratorsByDefRef("/ADAPTIVE/MICROSAR/amsr_logapi_config")

                // Open the report closure to get access to the report settings
                report {
                      createHtmlReport true
                      createXmlFile true
                }
            }

            // After generation the output paths can be found in the console view
            generate()
        }
    }
}
```

Listing 4.139: Generation of components with a result report

**createXmlFile** `setCreateXmlFile(Boolean)` defines if XML report file should be generated.

**createHtmlReport** `setCreateHtmlReport(Boolean)` defines if HTML report should be generated.

**Select one generator** This sample selects one specific generator and starts the generation.

```
scriptTask("generate_by_selecting_one"){
    code{
        generation{
            settings{
                selectAllScopes()
                selectGeneratorsByDefRef("/ADAPTIVE/MICROSAR/amsr_logapi_config")
            }
            generate()
        }
    }
}
```

Listing 4.140: Generate with one generator

```
scriptTask("generate_by_deselecting_one"){
    code{
        generation{
            settingsFromProject{
                // "/ADAPTIVE/MICROSAR/amsr_applicationbase" and
                // "/ADAPTIVE/MICROSAR/amsr_logapi_config" are selected in project
                     settings

                deselectGenerators("/ADAPTIVE/MICROSAR/amsr_applicationbase")
            }
            generate() // generate with only "/ADAPTIVE/MICROSAR/
                amsr_logapi_config"
        }
    }
}
```

Listing 4.141: Generate with a modified project settings

**Select multiple generators** To select more than one generator the following snippet can be used.

```
scriptTask("generate_with_two_generators"){
    code{
        generation{
            settings{
                selectAllScopes()
                selectGeneratorsByDefRef("/ADAPTIVE/MICROSAR/amsr_applicationbase"
                    , "/ADAPTIVE/MICROSAR/amsr_logapi_config")
            }
            generate()
        }
    }
}
```

Listing 4.142: Generate with two generators

Instead of selecting the generator directly by its `DefRef`, there is also the possibility to fetch the generator object and select this object for execution.

```
scriptTask("generate_with_two_generator"){
    code{
        generation{
            settings{
                selectAllScopes()
                def gen1 = generatorByDefRef("/ADAPTIVE/MICROSAR/
                    amsr_applicationbase")
                def gen2 = generatorByDefRef("/ADAPTIVE/MICROSAR/
                    amsr_logapi_config")
                selectGenerators([gen1, gen2])
            }
            generate()
        }
    }
}
```

Listing 4.143: Get and generate with multiple generators

**Select scopes** The following snippets show how to de/select scopes for a validation/generation.

```
generation{
    settings {
        // A new settings closure has no generator selected by default
        selectAll()

        selectScope("/OneCompany/AppExample/LightControl/Deployment/
            CentralExecMachine")
        selectScopes([
            "/OneCompany/AppExample/LightControl/HeadlightDesign/
                HeadlightExecutable",
            "/OneCompany/AppExample/LightControl/LightCoordinatorDesign/
                LightCoordinatorExecutable"]
        )
        selectScopes(
            "/OneCompany/AppExample/LightControl/Deployment/LightControlSwCluster"
                ,
            "/OneCompany/AppExample/LightControl/Deployment/PlatformSwCluster"
        )
    }
    validate() // Five scopes are selected
}
```

Listing 4.144: Validate with scopes

```
scriptTask("generate_with_deselecting_scopes"){
    code{
        generation{
            settings {
                // A new settings closure has no generator selected by default
                selectAll()

                // A new settings closure has no scope selected by default
                selectAllScopes()

                def allScopes = getAllScopes()

                // Two scopes are deselected
                deselectScopes(scopes[0], scopes[1])
            }
            generate()
        }
    }
}
```

Listing 4.145: Deselect scopes

If a scope cannot be found, **ScopeNotFoundException** is thrown during validation/generation.

```
import com.vector.cfg.gen.core.genusage.groovy.exceptions.ScopeNotFoundException

scriptTask("generate_with_wrong_scopes"){
    code{
        generation{
            settings {
                // A new settings closure has no generator selected by default
                selectAll()

                selectScope("/Something/NotExists")

                // Use isValidScope(String) to check if it is a valid scope
                assert isValidScope("/Something/NotExists") == false
            }

            try {
                validate() // Throws ScopeNotFoundException
            } catch (ScopeNotFoundException exception) {
                scriptLogger.info exception
            }
        }
    }
}
```

Listing 4.146: Generate with wrong scopes

#### 4.8.1.2 Evaluate generation or validation results

Each validation and generation process has an overall result which states if the execution has been successful or not. Additionally to the overall state, the state of one specific generator and scope can also be of interest. To provide a possibility to access this information all methods for `validate` and `generate` return an `IGenerationResultModel`.

```
scriptTask("generate_and_evaluate_result"){
    code{
        generation{
           def result = generate()
           println "Overall result : " + result.result
           println "Duration       : " + result.formattedDuration

           // Access results of each combination of generator and scope
           result.generationResultRoot.allGeneratorAndStepElements.each {
              println "Generator name : " + it.name
              println "Scope          : " + it.scope
              println "Result         : " + it.currentState
           }

        }
    }
}
```

Listing 4.147: Evaluate the generation result

### 4.8.2 Validation/Generation Task Types

There are two types of `IScriptTaskType` for the validation/generation process:

- Validation/Generation Process Start: `DV_ON_GENERATION_START`
- Validation/Generation Process End: `DV_ON_GENERATION_END`

The general description of the type is in chapter 4.3.1.4 on page 27. The following code samples show the usage of these task types:

**Validation/Generation Process Start** Samples for the `DV_ON_GENERATION_START` type:

```
scriptTask("GenStartTask", DV_ON_GENERATION_START){
    taskDescription "The task is automatically executed at validation/generation
       start"

    code{ phasesToExecute, generators ->

        scriptLogger.info "Phases are: $phasesToExecute"
        scriptLogger.info "Generators to execute are: $generators"

        // Execute code before the validation/generation will start
    }
}
```

Listing 4.148: Hook into the GenerationProcess at the start with script task

```
import com.vector.cfg.gen.core.moduleinterface.EGenerationPhaseType
import com.vector.cfg.gen.core.moduleinterface.IGenFolders

scriptTask("GetOutputFolderTask", DV_ON_GENERATION_START) {
    code { phasesToExecute, generators ->
        if (phasesToExecute.contains(EGenerationPhaseType.GENERATION)) {
            // Only executed for generation, not for validation.
            IGenFolders genFolders = project.getInstance(IGenFolders.class)
            File outputFolder = genFolders.getCodeOutputFolder()
            scriptLogger.info "outputFolder: $outputFolder"
        }
    }
}
```

Listing 4.149: Get generation output folder when generation starts

**Validation/Generation Process End** A sample for the `DV_ON_GENERATION_END` type:

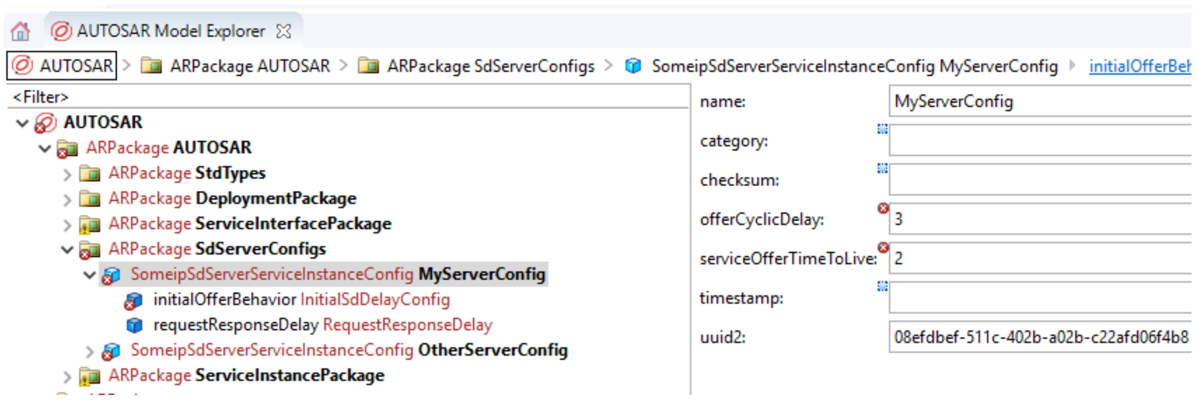
```
scriptTask("GenEndTask", DV_ON_GENERATION_END){
    taskDescription "The task is automatically executed at validation/generation
       end"

    code{ processResult, generators ->

        scriptLogger.info "Process result was: $processResult"
        scriptLogger.info "Executed Generators: $generators"

        // Execute code after the validation/generation process was finished
    }
}
```

Listing 4.150: Hook into the GenerationProcess at the end with script task

## 4.9 Validation

### 4.9.1 Introduction

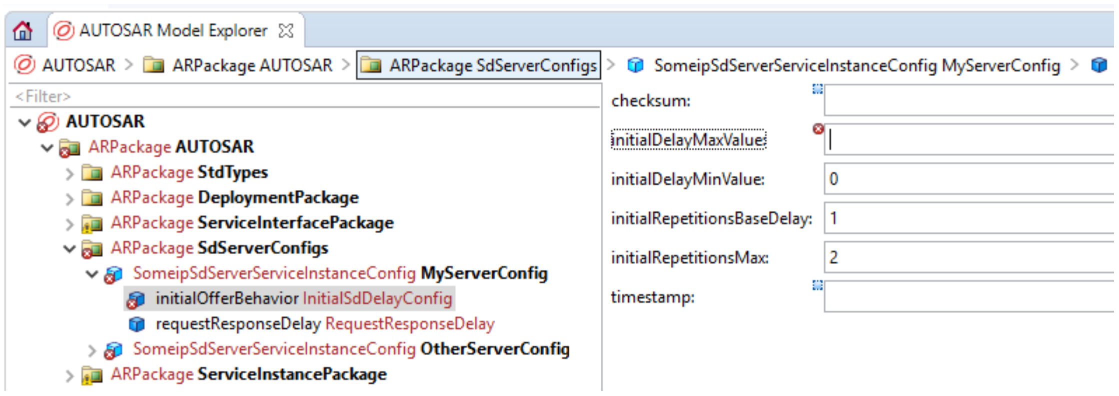
All examples in this chapter are based on the situation of the figures 4.11 and 4.12. The data is not from a real project, but just for the examples. There is a SomeipSdServerInstanceConfig whose offerCyclicDelay and serviceOfferTimeToLive values are validated. If the offerCyclicDelay value is greater than the serviceOfferTimeToLive value, the validation will report it by a validation result with the ID `APCOM70016`. Another validation result with the ID `APCOM70011` will be reported if the initialDelayMaxValue at its initialOfferBehavior is not set. This value is mandatory.


AUTOSAR Model Explorer
AUTOSAR > ARPackage AUTOSAR > ARPackage SdServerConfigs > SomeipSdServerServiceInstanceConfig MyServerConfig > initialOfferBe

<Filter>
- AUTOSAR
  - ARPackage **AUTOSAR**
    - ARPackage **StdTypes**
    - ARPackage **DeploymentPackage**
    - ARPackage **ServiceInterfacePackage**
    - ARPackage **SdServerConfigs**
      - SomeipSdServerServiceInstanceConfig **MyServerConfig**
        - initialOfferBehavior InitialSdDelayConfig
        - requestResponseDelay RequestResponseDelay
      - SomeipSdServerServiceInstanceConfig **OtherServerConfig**
    - ARPackage **ServiceInstancePackage**

| | |
|---|---|
| name: | MyServerConfig |
| category: | |
| checksum: | |
| offerCyclicDelay: | 3 |
| serviceOfferTimeToLive: | 2 |
| timestamp: | |
| uuid2: | 08efdbef-511c-402b-a02b-c22afd06f4b8 |

Figure 4.11: example situation with the GUI


AUTOSAR Model Explorer
AUTOSAR > ARPackage AUTOSAR > ARPackage SdServerConfigs > SomeipSdServerServiceInstanceConfig MyServerConfig >

<Filter>
- AUTOSAR
  - ARPackage **AUTOSAR**
    - ARPackage **StdTypes**
    - ARPackage **DeploymentPackage**
    - ARPackage **ServiceInterfacePackage**
    - ARPackage **SdServerConfigs**
      - SomeipSdServerServiceInstanceConfig **MyServerConfig**
        - initialOfferBehavior InitialSdDelayConfig
        - requestResponseDelay RequestResponseDelay
      - SomeipSdServerServiceInstanceConfig **OtherServerConfig**
    - ARPackage **ServiceInstancePackage**

| | |
|---|---|
| checksum: | |
| initialDelayMaxValue: | |
| initialDelayMinValue: | 0 |
| initialRepetitionsBaseDelay: | 1 |
| initialRepetitionsMax: | 2 |
| timestamp: | |

Figure 4.12: example situation with the GUI

---

Remark:

The validation-results to solve are identified via their ID. This ID is case-sensitive: that means it has to be used exactly as seen in the problems view or the tooltip of the error decoration. Validation-Result-IDs of functional clusters are usually in capital letters (e.g. `APCOM70016`), while the prefix `AP` represents validation results of the AUTOSAR adaptive platform. Other validation-results may use validation-IDs in camel-case style (e.g. `Ats00024`).

---

### 4.9.2 Access Validation-Results

A `validation{}` block gives access to the validation API of the consistency component. That means accessing the validation-results which are shown in the GUI in the problems view and the tooltip when hovering over the error decorations in the AUTOSAR Model Explorer.

getValidationResults() waits for background-validation-idle and returns all validation-results of any kind. The returned collection has no deterministic order, especially it is not the same order as in the GUI.

```
scriptTask("CheckValidationResults_filterByOriginId", DV_PROJECT){
    code{
        validation{
            fullValidation()

            // access all validation-results
            def allResults = validationResults
            assert allResults.size() > 1

            // filter based on methods of IValidationResultUI e.g. isId()
            def apCom00012Results = validationResults.filter{it.isId("AR-APCOM",
                12)}
            assert apCom00012Results.size() == 1
        }

        // alternative access to validation-results without a validation block
        assert validation.validationResults.size() > 1
    }
}
```

Listing 4.151: Access all validation-results and filter them by ID

### 4.9.3 Full Validation

In the validation{} block, a full validation can be requested with the fullValidation() method.

fullValidation() starts the full background validation, all un-validated MDF objects will be validated during the full validation. Note: if the full validation has already been started, this method will do nothing and return directly.

### 4.9.4 Model Transaction and Validation-Result Invalidation

Before we continue in this chapter with solving validation-results, the following information is import to know:

**Relation to model transactions:**

Solving validation-results with solving-actions always creates a transaction implicitly. An IllegalStateException will be thrown if this is done within an explicitly opened transaction.

**Invalidation of validation-results:**

Any model modification may invalidate any validation-result. In that case, the responsible validator creates a new validation-result if the inconsistency still exists. Whether this happens for a particular modification/validation-result depends on the validator implementation and is not visible to the user/client.

Trying to solve an invalidated validation-result will throw an IllegalStateException. Therefore it is not safe to solve a particular ISolvingActionUI that was fetched before the last transaction. Instead, please fetch a solving-action after the last transaction, or use the method ISolver.solve(Closure) which is the most preferred way of solving validation-results with solving-actions.

See chapter 4.9.5.1 on the following page for details.

### 4.9.5 Solve Validation-Results with Solving-Actions

A single validation-result can be solved by calling `solve()` on one of its solving-actions.

```
scriptTask("SolveSingleResultWithSolvingAction", DV_PROJECT){
    code{
        validation{
            fullValidation()
            // Find the result with a result ID. Here the "MANIFEST" is an example
                , should be replaced with an actual ID.
            def results = validationResults.filter{it.isId("MANIFEST", 1)}
            assert results.size() == 4

            // Take first (any) validation-result and filter its solving-actions
                based on methods of ISolvingActionUI
            results.first.solvingActions.filter{
                it.description.contains("Set value to")

            }.single.solve() // reduce the collection to a single ISolvingActionUI
                 and call solve()

            assert validationResults.filter{it.isId("MANIFEST", 1)}.size() == 3
            // One MANIFEST1 validation-result solved
        }
    }
}
```

Listing 4.152: Solve a single validation-result with a particular solving-action

#### 4.9.5.1 Solver API

`getSolver()` gives access to the `ISolver` API, which has advanced methods for bulk solutions.

`ISolver.solve(Closure)` allows to solve multiple validation-results within one transaction. You should always use this method to solve multiple validation-results at once instead of calling `ISolvingActionUI.solve()` in a loop. This is very important, because solving one validation-result, may cause invalidation of another one. And calling `ISolvingActionUI.solve()` of an invalidated validation-result throws an `IllegalStateException`. Also, invalidated validation-results may get recalculated and you would miss the recalculated validation-results with the loop approach. But with `ISolver.solve(Closure)` you can solve invalidated->recalculated results as well as results which didn't exist at the time of the call (but have been caused by solving some other validation-result).

`ISolver.solve(Closure)` first waits for background-validation-idle in order to have reproducible results.

The closure may contain multiple statements like:

```
result{specify result predicate}.withAction{select solving action}
```

All statements together will be used as a mapper from any solvable validation-result to a particular solving-action. The order of these statements does not affect the solving action execution order. The statement order might only be relevant if multiple statements match on a particular result, but would select a different solving-action. In that case, the first statement that successfully selects a solving-action wins.

```
scriptTask("SolveMultipleResults", DV_PROJECT){
  code{
    validation{
      fullValidation()
      assert validationResults.size() == 4
      solver.solve{
        // Call result() and pass a closure that works as filter
        // based on methods of IValidationResultUI.
        result{
          isId("MANIFEST", 1)
        }.withAction{
          containsString("Set value to")
        }

        // On the return value, call withAction() and pass a closure that
        // selects a solving-action based on methods
        // of IValidationResultForSolvingActionSelect

        // multiple result() calls can be placed in one solve() call.
        result{isId("COM", 34)}.withAction{containsString("recalculate")}
      }

      assert validationResults.size() == 0
      // Two MANIFEST1 and zero COM34 (didn't exist) results solved.
}}}
```

Listing 4.153: Fast solve multiple results within one transaction

**Solve all PreferredSolvingActions** `ISolver.solveAllWithPreferredSolvingAction()` solves all validation-results with its preferred solving- action of each validation-result (solving-action return by `IValidationResultUI.getPreferredSolvingAction()`). Validation-results without a preferred solving-action are skipped.

This method first waits for background-validation-idle in order to have reproducible results.

```
scriptTask("SolveAllWithPreferred", DV_PROJECT){
  code{
    validation{
      fullValidation()
      assert validationResults.size() == 4

      solver.solveAllWithPreferredSolvingAction()

      assert validationResults.size() == 1

      // this would do the same
      transactions.transactionHistory.undo()
      assert validationResults.size() == 4

      solver.solve{
        result{true}.withAction{preferred}
      }

      assert validationResults.size() == 1
}}}
```

Listing 4.154: Solve all validation-results with its preferred solving-action (if available)

### 4.9.6 Advanced Topics

### 4.9.6.1 Access Validation-Results of a Model Object

You can retrieve validation-results also from any model object (MDF).

`MIObject.getValidationResults()` returns the validation-results of an `MIObject`. These are those results for which `IValidationResultUI.matchErroneousCE(MIObject)` returns true.

```
scriptTask("CheckValidationResultsOfObject", DV_PROJECT){
    code{
        validation{
            fullValidation()
        }
        // a SomeipSdServerServiceInstanceConfig
        def serviceInstanceConfig = mdfModel(AsrPath.create("/AUTOSAR/
            SdServerConfigs/MyServerConfig"))
        // the request response delay child
        def requestResponseDelayParam = serviceInstanceConfig.requestResponseDelay

        // one result exists for the service instance config (offer cyclic delay >
             serviceOfferTimeToLive)
        // the request response delay of it has no results
        assert serviceInstanceConfig.validationResults.size() == 1
        assert requestResponseDelayParam.validationResults.size() == 0
    }
}
```

Listing 4.155: Access all validation-results of a particular object

`MIObject.getValidationResultsRecursive()` returns the validation-results of an `MIObject` and all its children. So this will return all results of the whole subtree, like an editor displays results at parent objects.

### 4.9.6.2 Filter Validation-Results using an ID Constant

Groovy allows you to spread list elements as method arguments using the spread operator. This allows you to define constants for the `isId(String,int)` method.

```
scriptTask("FilterResultsUsingAnIdConstant2", DV_PROJECT){
    code{
        validation{
            fullValidation()
            def apCom00012Const = ["AR-APCOM",12]

            assert validationResults.size() > 1
            assert validationResults.filter{it.isId(*apCom00012Const)}.size() == 1
        }
    }
}
```

Listing 4.156: Filter validation-results using an ID constant

### 4.9.6.3 Identification of a Particular Solving-Action

A so called solving-action-group-ID identifies a solving-action uniquely within one validation-result. In other words, two solving-actions, which do semantically the same, from two validation-results of the same result-ID (origin + number), belong to the same solving-action-group. This semantical group may have an optional solving-action-group-ID, that can be used for solving-action identification within one validation-result.

Keep in mind that the solving-action-group-ID is only unique within one validation-result-ID, and that the group-ID assignment is optional for a validator implementation.

In order to find out the solving-action-group-IDs, press CTRL+SHIFT+F9 with a selected validation-result to copy detailed information about that result including solving-action-group-IDs (if assigned) to the clipboard.

If group-IDs are assigned, it is much safer to use these for solving-action identification than description-text matching, because a description-text may change.

```
final int SA_GROUP_ID = 1

scriptTask("SolveMultipleResultsByGroupId", DV_PROJECT){
  code{
    validation{
      fullValidation()
      assert validationResults.size() == 4

      solver.solve{
        result{isId("MANIFEST", 1)}
          .withAction{
            byGroupId(SA_GROUP_ID)
          }
          // instead of .withAction{containsString("Set value to")}
      }

      assert validationResults.size() == 0
      // Two MANIFEST1 validation-results solved.
    }
  }
}
```

Listing 4.157: Fast solve multiple validation-results within one transaction using a solving-action-group-ID

#### 4.9.6.4 Validation-Result Description as MixedText

`IValidationResultUI.getDescription()` returns an `IMixedText` that describes the inconsistency.

`IMixedText` is a construct that represents a text, whereby parts of that text can also hold the object which they represent. This allows a consumer e.g. a GUI to make the object-parts of the text clickable and to reformat these object-parts as wanted.
Consumers which don't need these advanced features can just call `IMixedText.toString()` which returns a default format of the text.

#### 4.9.6.5 Further IValidationResultUI Methods

The following listing gives an overview of other "properties" of an IValidatonResultUI.

```
scriptTask("IValidationResultUIApiOverview", DV_PROJECT){
  code{
    validation{
      fullValidation()
      def r = validationResults.filter{it.isId("AR-APCOM", 12)}.first
      assert r.id.origin == "AR-APCOM"
      assert r.id.id == 12
      assert r.description.toString().contains("serviceOfferTimeToLive must be >=
          offerCyclicDelay")
      assert r.severity == EValidationSeverityType.WARNING // since ValidationLib
          0.7 all error are reduced to warning
      assert r.solvingActions.size() == 2

      // this result has no preferred-solving-action
      assert r.preferredSolvingAction == null

      // results with lower severity than ERROR can be acknowledged
      assert r.acknowledgement.isPresent() == false

      // if the cause was an exception, r.cause.get() returns it
      assert r.cause.isPresent() == false

      // on-demand results are visualized with a gear-wheel icon
      assert r.isOnDemandResult() == false
    }
  }
}
```

Listing 4.158: IValidationResultUI overview

#### 4.9.6.6 Erroneous CEs of a Validation-Result

`IValidationResultUI.getErroneousCEs()` returns a collection of `IDescriptor`, each describing a CE that gets an error annotation in the GUI.

To check for a certain model element is affected by the result please use the methods, which return `true`, if a model is affected by the validation-result:

- `IValidationResultUI.matchErroneousCE(MIObject)`
- `IValidationResultUI.matchErroneousCE(IHasModelObject)`
- `IValidationResultUI.matchErroneousCE(MIHasDefinition, DefRef)`

```
import com.vector.cfg.model.cedescriptor.aspect.*
scriptTask("IValidationResultUIErroneousCEs", DV_PROJECT){
    code{
        validation{
            fullValidation()
            def result = validationResults.filter{it.isId("AR-APCOM", 12)}.first

            // Retrieve the model element to check
            def modelElement // = retrieveElement ...

            // Check if the model object is affected by the validation-result
            assert result.matchErroneousCE(modelElement)
        }
    }
}
```

Listing 4.159: CE is affected by (matches) an IValidationResultUI

**Advanced Descriptor Details** An `IDescriptor` is a construct that can be used to "point to" some location in the model. A descriptor can have several kinds of aspects to describe where it points to. Aspect kinds are e.g. `IMdfObjectAspect`, `IDefRefAspect`, `IMdfMetaClassAspect`, `IMdfFeatureAspect`.

`getAspect(Class)` gets a particular aspect if available, otherwise null.

A descriptor has a parent descriptor. This allows to describe a hierarchy.
E.g. if you want to express that something with definition X is missing as a child of the existing MDF object Y. In this example you have a descriptor with an `IDefRefAspect` containing the definition X. This descriptor that has a parent descriptor with an `IMdfObjectAspect` containing the object Y.

The term descriptor refers to a descriptor together with its parent-descriptor hierarchy.

```
import com.vector.cfg.model.cedescriptor.aspect.*
import com.vector.cfg.model.mdf.ar4x.systemtemplate.fibex.fibex4ethernet.
   ethernettopology.MIInitialSdDelayConfig
import com.vector.cfg.model.mdf.model.autosar.base.MIARPackage

scriptTask("IValidationResultUIErroneousCEs", DV_PROJECT){
    code{
        validation{
            fullValidation()
            def result = validationResults.filter{it.isId("AR-APCOM", 15)}.first
            def descriptor = result.erroneousCEs.first // this result in this
                example has one erroneous-CE descriptor
            def featureAspect = descriptor.getAspect(IMdfFeatureAspect.class)
            assert featureAspect != null; // this descriptor in this example has
                an IMdfFeatureAspect
            assert featureAspect.feature.equals(MIInitialSdDelayConfig.
                INITIAL_DELAY_MAX_VALUE)
            def objectAspect = descriptor.getAspect(IMdfObjectAspect.class)
            assert objectAspect != null // this descriptor in this example has an
                IMdfObjectAspect
            // An IMdfObjectAspect would be unavailable for a descriptor
                describing that something is missing

            // Dealing with descriptors is universal, but needs more code. Using
                these method might fit your needs.
            assert result.matchErroneousCE(objectAspect.getObject())
        }
    }
}
```

Listing 4.160: Advanced use case - Retrieve Erroneous CEs with descriptors of an IValidationResultUI

### 4.9.6.7 Examine Solving-Action Execution

The easiest and most reliable option for verifying solving-action execution is to check the presence of validation-results afterwards.

This is also the feedback strategy of the GUI. After multiple solving-actions have been solved, the GUI does not show the execution result of each individual solving-action, but just the remaining validation-results after the operation. Only if a single solving-action is to be solved, and that fails, the GUI shows the message of that failure including the reason.

The following describes further options of examination:

`ISolvingActionUI.solve()` returns an `ISolvingActionExecutionResult`. An `ISolvingActionExecutionResult` represents the result of one solving action execution. Use `isOk()` to find out if it was successful. Call `getUserMessage()` to get the failure reason.

`ISolver.solve(Closure)` returns an `ISolvingActionSummaryResult`. An `ISolvingActionSummaryResult` represents the execution of multiple results. `ISolvingActionSummaryResult.isOk()` returns true if `getExecutionResult()` is `EExecutionResult.SUCCESSFUL` or `EExecutionResult.WARNING`, this is if at least one sub-result was ok.

Call `getSubResults()` to get a list of `ISolvingActionExecutionResults`.

```
import com.vector.cfg.util.activity.execresult.EExecutionResult

scriptTask("SolvingReturnValue", DV_PROJECT){
    code{
        validation{
            fullValidation()
            assert validationResults.size() == 4
            // In this example, three validation-results have a preferred solving
                action.
            // One of the three cannot be solved because a parameter is user-
                defined.
            def summaryResult = solver.solveAllWithPreferredSolvingAction()
            assert validationResults.size() == 2 // Two have been solved, one with
                a preferred solving-action is left.
            assert summaryResult.executionResult == EExecutionResult.WARNING

            // DemoAsserts is just for this example to show what kind of sub-
                results the summaryResult contains.
            DemoAsserts.summaryResultContainsASubResultWith("OK",summaryResult)
            //two such sub-results for the validation-results with preferred-
                solving-action that could be solved

            DemoAsserts.summaryResultContainsASubResultWith(["invalid modification"
                ,"not changeable","Reason","is user-defined"],summaryResult)
            // such a sub-result for the failed preferred solving action due to the
                 user-defined parameter

            DemoAsserts.summaryResultContainsASubResultWith("Maximum solving
                attempts reached for the validation-result of the following solving
                -action",summaryResult)
            // Cfg5 takes multiple attempts to solve a result because other changes
                 may eliminate a blocking reason, but stops after an execution
                limit is reached.
        }
    }
}
```

Listing 4.161: Examine an ISolvingActionSummaryResult

**Reporting ValidationResult with a ResultSink** This sample shows how to report ValidationResult to a ResultSink.

```
scriptTask("ScriptTaskCreationResult" /* Insert with task type providing
    resultSink */ ){
  code{
    validation{
      fullValidation()
      resultCreation{
        // The ValidationResultId group multiple results
        def valId = createValidationResultIdForScriptTask(
                /* ID */ 1234,
                /* Description */ "Summary of the ValidationResultId",
                /* Severity */ EValidationSeverityType.ERROR)
        // Create a new resultBuilder
        def builder = newResultBuilder(valId, "Description of the Result")

        // You can add multiple elements as error objects to mark them
        builder.addErrorObject(mdfModel("/com/ComExecutable").single)
        // Add more calls when needed

        // Create the result from the builder
        def valResult = builder.buildResult()

        // You need to report the result to a resultSink
        // You have to get the sink from the context, e.g. script task args
        // a sample line would be
        resultSink.reportValidationResult(valResult)
      }
    }
}}
```

Listing 4.162: Create a ValidationResult

#### 4.9.6.8 Clear the on-demand ValidationResult

With the method `clearOnDemandValidationResults()`, you can clear all OnDemand results that are currently existing in the Consistency.

## 4.10 Persistency

The `persistency` API provides methods which allow to import and export model data from and to files. The files are normally in the AUTOSAR `.arxml` format.

### 4.10.1 Model Export

The `modelExport` allows to export MDF model data into `.arxml` files.

To access the export functionality use one of the `getModelExport()` or `modelExport(Closure)` methods.

```
// You can access the API in every active project
def exportApi = persistency.modelExport

//Or you use a closure
persistency.modelExport {
}
```

Listing 4.163: Accessing the model export persistency API

#### 4.10.1.1 Advanced Exports

The advanced export use case provides access to multiple `IModelExporter` for special export use cases like export model without elements in BSW Package.

Normally you would retrieve an `IModelExporter` by its ID via `getExporter(String)`. On this exporter you can call `IModelExporter.export(Object)` to export the model.

You can retrieve a list of supported exporters from `getAvailableExporter()`. The list can differ from data loaded in your project.

```
scriptTask('taskName') {
    code {
        def tempExportFolder = paths.resolveTempPath(".")

         // Export with an exporter in multiple lines
        persistency.modelExport{
            withExporterArgs(modelTree: "--element /AUTOSAR"){
                getExporter("modelTree").get().export(tempExportFolder)
            }
        }
    }
}
```

Listing 4.164: Export the project with an exporter into a folder

```
scriptTask('taskName') {
    code {
        def everythingFile = paths.resolveTempPath("./everything.arxml")

        def path = paths.bswPackageRootFolder
        def withoutBSWPackageFile = paths.resolveTempPath("./withoutBSWPackage.
            arxml")

        persistency.modelExport{
            // Export with an exporter that exports everything
            getExporter("everything").get().exportToFile(everythingFile)

            // Export with an exporter that excludes elements in BSW Package
            withExporterArgs(excludePath: "--path " + path){
                getExporter("excludePath").get().exportToFile(
                    withoutBSWPackageFile)
            }

        }
    }
}
```

Listing 4.165: Export the project model with/without elements in BSW Package into a file

```
scriptTask('taskName') {
    code {
        def tempExportFolder = paths.resolveTempPath(".")

        def fileList
        //Switch to the persistency export API
        persistency.modelExport{
            // The getAvailableExporter() returns all exporters in the system
            def exporterList = getAvailableExporter()

            withExporterArgs(modelTree: "--element /AUTOSAR"){
                // Select an exporter by its ID
                def exporterOpt = getExporter("modelTree")

                exporterOpt.ifPresent { exporter ->
                    // Export into folder, when exporter exists
                    fileList = exporter.export(tempExportFolder)
                }
            }
        }
    }
}
```

Listing 4.166: Export the project with an exporter and checks

**Export an Model Tree** The method `exportModelTreeToFile(Object, MIObject)` exports the specified model object and the subtree into a single file of type `Path` specified by the user.

```
scriptTask('taskName') {
    code {
        def destinationFile // Define the file to export into...
        MIARPackage autosarPkg = mdfModel(AsrPath.create("/AUTOSAR"))

        persistency.modelExport{
            exportModelTreeToFile(destinationFile, autosarPkg)
        }
    }
}
```

Listing 4.167: Export an AUTOSAR package into a file

The method `exportModelTree(Object, MIObject)` exports the specified model object and the subtree into a single file of type `Path`.

```
scriptTask('taskName') {
    code {
        def exportFolder = paths.resolveTempPath(".")
        MIARPackage autosarPkg = mdfModel(AsrPath.create("/AUTOSAR"))

        def resultFile = persistency.modelExport.exportModelTree(exportFolder,
           autosarPkg)
    }
}
```

Listing 4.168: Export an AUTOSAR package into a folder

**Export an Model Tree including all referenced Elements** You could also export model trees including all referenced elements with the exporter `modelTreeClosure`:

```
scriptTask('taskName') {
    code {
        def exportFolder = paths.resolveTempPath(".")
        def srvIfcDummyElmt = mdfModel(AsrPath.create("/AUTOSAR/srvIfcDummy"))
        def autosarPkg = mdfModel(AsrPath.create("/AUTOSAR"))

        persistency.modelExport["modelTreeClosure"].export(exportFolder,
           autosarPkg, srvIfcDummyElmt)
    }
}
```

Listing 4.169: Exports two elements and all references elements

**Usage of Exporter Arguments** You can use `withExporterArgs(Map, Closure)` to specify exporter arguments like in the command line with `-exporterArgs` argument. The key is the exporter ID, the value are the arguments to the exporter. See command line help for details.

```
persistency.modelExport{
    // Specify the arguments with exporterId: "arguments"
    withExporterArgs(modelTree: "--element /AUTOSAR"){
        // Call any export code with the active arguments.
        getExporter("modelTree").get().exportToFile(destinationFile)
    }
}
```

Listing 4.170: Use exporter arguments like in the commandline

## 4.11 Utilities

### 4.11.1 Constraints

`Constraints` provides general purpose constraints for checking given parameter values throughout the automation interface. These constraints are referenced from the AutomationInterface documentation wherever they apply. The AutomationInterface takes a fail fast approach verifying provided parameter values as early as possible and throwing appropriate exceptions if values violate the corresponding constraints.

The following constraints are provided:

**IS_NOT_NULL** Ensures that the given `Object` is not `null`.

**IS_NON_EMPTY_STRING** Ensures that the given `String` is not empty.

**IS_VALID_FILE_NAME** Ensures that the given `String` can be used as a file name.

**IS_VALID_PROJECT_NAME** Ensures that the given `String` can be used as a name for a project. A valid project name starts with a letter [a-zA-Z] contains otherwise only characters matching [a-zA-Z0-9_-.] and is at most 128 characters long.

**IS_NON_EMPTY_ITERABLE** Ensures that the given `Iterable` is not empty.

**IS_VALID_AUTOSAR_SHORT_NAME** Ensures that the given `String` conforms to the syntactical requirements for AUTOSAR short names.

**IS_VALID_AUTOSAR_SHORT_NAME_PATH** Ensures that the given `String` conforms to the syntactical requirements for AUTOSAR short name paths.

**IS_WRITABLE** Ensures that the file or folder represented by the given `Path` exists and can be written to.

**IS_READABLE** Ensures that the file or folder represented by the given `Path` exists and can be read.

**IS_EXISTING_FOLDER** Ensures that the given `Path` points to an existing folder.

**IS_EXISTING_FILE** Ensures that the given `Path` points to an existing file.

**IS_CREATABLE_FOLDER** Ensures that the given `Path` either points to an existing folder which can be written to or points to a location at which a corresponding folder could be created.

**IS_ARXML_FILE** Ensures that the given `Path` points to an .arxml file.

**IS_SYSTEM_DESCRIPTION_FILE** Ensures that the given `Path` points to a system description input file (.arxml, .dbc, .ldf, .xml or .vsde file).

### 4.11.2 Converters

General purpose converters (`java.util.Functions`) for performing value conversions throughout the automation interface are provided. These converters are referenced from the AutomationInterface documentation wherever they apply. The AutomationInterface is typed strongly. In some cases, however, e.g. when specifying file locations, it is desirable to allow for a range of possibly parameter types. This is achieved by accepting parameters of type `Object` and converting the given parameters to the desired type.

The following converters are provided:

**ScriptConverters.TO_PATH** Attempts to convert arbitrary `Objects` to `Paths` using `IAutomationPathsApi.resolvePath(Object)` 4.4.3.2 on page 32.

**ScriptConverters.TO_SCRIPT_PATH** Attempts to convert arbitrary `Objects` to `Paths` using `IAutomationPathsApi.resolveScriptPath(Object)` 4.4.3.3 on page 33.

**ScriptConverters.TO_VERSION** Attempts to convert arbitrary `Objects` to `IVersions`. The following conversions are implemented:

- For `null` or `IVersion` arguments the given argument is returned. No conversion is applied.
- `Strings` are converted using `Version.valueOf(String)`.
- `Numbers` are converted by converting the `int` obtained from `Number.intValue()` using `Version.valueOf(int)`.
- All other `Objects` are converted by converting the `String` obtained from `Object.toString()`.

**ScriptConverters.TO_BIG_INTEGER** Attempts to convert arbitrary `Objects` to `BigIntegers`. The following conversions are implemented:

- For `null` or `BigInteger` arguments the given argument is returned. No conversion is applied.
- `Integers`, `Longs`, `Shorts` and `Bytes` are converted using `BigInteger.valueOf(long)`.
- All other types of objects are interpreted as `Strings` (`Object.toString()`) and passed to `BigInteger.BigInteger(String)`.

**ScriptConverters.TO_BIG_DECIMAL** Attempts to convert arbitrary `Objects` to `BigDecimals`. The following conversions are implemented:

- For `null` or `BigDecimal` arguments the given argument is returned. No conversion is applied.
- `Floats` and `Doubles`, are converted using `BigDecimal.valueOf(double)`.
- `Integers`, `Longs`, `Shorts` and `Bytes` are converted using `BigDecimal.valueOf(long)`.
- All other types of objects are interpreted as `Strings` (`Object.toString()`) and passed to `BigDecimal.BigDecimal(String)`.

**ModelConverters.TO_MDF** Attempts to convert arbitrary `Objects` to `MDFObjects`. The following conversions are implemented:

- For `null` or `MDFObject` arguments the given argument is returned. No conversion is applied.
- `IHasModelObjects` are converted using their `getMdfModelElement()` method.
- `IViewedModelObjects` are converted using their `getMdfObject()` method.

- For all other `Objects` `ClassCastExceptions` are thrown.

For thrown `Exceptions` see the used functions described above.

## 4.12 Advanced Topics

This chapter contains advanced use cases and classes for special tasks. For a normal script these items are not relevant.

### 4.12.1 Java Development

It is also possible to write automation scripts in plain Java code, but this is not recommended. There are some items in the API, which need a different usage in Java code.

This chapter describes the differences in the Automation API when used from Java code.

#### 4.12.1.1 Script Task Creation in Java Code

Java code could not use the Groovy syntax to provide script tasks. So another way is needed for this. The `IScriptFactory` interface provides the entry point that Java code could provide script tasks. The `createScript(IScriptCreationApi)` method is called when the script is loaded.

This interface is **not** necessary for Groovy clients.

```
public class MyScriptFactoryAsJavaCode implements IScriptFactory {
    @Override
    public void createScript(IScriptCreationApi creation) {
        creation.scriptTask("TaskFromFactory", IScriptTaskTypeApi.DV_APPLICATION,
                (taskBuilder) -> {
                    taskBuilder.code(
                            (scriptExecutionContext, taskArgs) -> {
                                // Your script task code here
                                return null;
                            });
                });

        creation.scriptTask("Task2", IScriptTaskTypeApi.DV_PROJECT,
                (taskBuilder) -> {
                    taskBuilder.code(
                            (scriptExecutionContext, taskArgs) -> {
                                // Your script task code for Task2  here
                                return null;
                            });
                });
    }
}
```

Listing 4.171: Java code usage of the IScriptFactory to contribute script tasks

You should try to use Groovy when possible, because it is more concise than the Java code, without any difference at script task creation and execution.

#### 4.12.1.2 Java Code accessing Groovy API

Most of the Automation API is usable from both languages Java and Groovy, but some methods are written for Groovy clients. To use it from Java you have to write some glue code.

Differences are:

- Accessing Properties
- Using API entry points.
- Creating Closures

**Accessing Properties** Properties are not supported by Java so you have to use the getter/setter methods instead.

**API Entry Points** Most of the Automation API is added to the object by so called DynamicObjects. This is not available in Java, so you have to call `IScriptExecutionContext.getInstance(Class)` instead. So if you want to access The AUTOSAR root object you have to write:

```
//Java code:
IScriptExecutionContext scriptCtx = ...;
MIAUTOSAR root = scriptCtx.getInstance(IModelApiEntryPoint.class).getAUTOSAR();

//Instead of Groovy code:
scriptTask("SimpleProjectTask", DV_PROJECT) {
    code {
        MIAUTOSAR root = AUTOSAR
    }
}
```

Listing 4.172: Accessing AUTOSAR root object in Java code

**Creating Closure instances from Java lambdas** The class `Closures` provides API to create `Closure` instances from Java `FunctionalInterfaces`.

The `from()` methods could be used to call Groovy API from Java classes, which only accepts `Closure` instances.

Sample:

```
 Closure<?> c = Closures.from((param) -> {
   // Java lambda
 });
```

Listing 4.173: Java Closure creation sample

**Creating Closure Instances from Java Methods** You could also create arbitrary `Closures` from any Java method with the class `MethodClosure`. This is describe in:
http://melix.github.io/blog/2010/04/19/coding_a_groovy_closure_in.html[1]

### 4.12.1.3 Java Code in dvgroovy Scripts

It is not possible to write Java classes when using the `.dvgroovy` script file. You have to create an automation script project, see chapter 7 on page 144.

## 4.12.2 Unit testing API

The Automation Interface provides an connector to execute unit tests as script task. This is helpful, if you want to write tests for:

- Generators (see chapter 4.13 on page 130 for details)
- Validations
- ...

Normally a script task executes it's code block, but the unit test task will execute all contained unit tests instead.

[1]Last accessed 2016-05-24

#### 4.12.2.1 JUnit5 Integration

The AutomationInterface can execute JUnit 5 test cases.

For this you have to add a `JUnit5` dependency in your `build.gradle` file:

```
dependencies{
    compileOnly("org.junit.jupiter:junit-jupiter-api:5.9.0")
}
```

Listing 4.174: Additional JUnit5 dependency in Gradle

**Execution of JUnit Test Classes** A simple unit test class will look like:

```
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;

public class ScriptJUnitTest {

    @Test
    void testYourLogic() {
        System.out.println("JUnit test has been executed");
        // Write your test code here
    }

    @Test
    @Disabled
    void testIgnoreTest() {
        System.out.println("Should not be executed!");
    }
}
```

Listing 4.175: Run all JUnit tests from one class

You can access the Automation API with the `ScriptApi` class. See chapter 4.4.8 on page 41 for details.

#### 4.12.2.2 Execution of Spock Tests

The AutomationInterface can also execute `Spock` tests. Please note that for `Spock` tests, you need a Automation Scripting Project and simple Script File is not sufficient.

See:

- Homepage: https://github.com/spockframework/spock[2]
- Documentation: http://spockframework.org/spock/docs/2.3/index.html[3]

It is also possible to group multiple `Spock` test into a JUnit Test Suite.

Usage sample:

[2]Last accessed 2023-02-27

[3]Last accessed 2023-02-27

```
import spock.lang.Specification

class ScriptSpockTest extends Specification {

    def "Simple Spock test"() {
        when:
        //Add your test logic here
        println "Spock test executed!"
        def myExpectedString = "Expected"

        then:
        myExpectedString == "Expected"
    }
}
```

Listing 4.176: Run unit test with the Spock framework

You can access the Automation API with the `ScriptApi` class. See chapter 4.4.8 on page 41 for details.

You have to add a `Spock` dependency in your `build.gradle` file:

```
dependencies {
  compileOnly("org.spockframework:spock-core:2.3-groovy-4.0"){
    exclude group: 'org.codehaus.groovy', module: 'groovy-all'
  }
}
```

Listing 4.177: Additional Spock dependency in Gradle

Note: after the change you have to call Gradle to update the IntelliJ IDEA project.
`gradlew idea -Dorg.gradle.java.home="<PATH_TO_JDK_17>"`[4]

### 4.12.2.3 Activation of Unit Tests in a Script

Spock tests and JUnit tests can be activated in a script to be an executable script task. The argument is the taskName for the unit tests.

```
project.ext.automationClasses = [
        "sample.MyScript",
        "sample.ActivateMyUnitTests", // This is the script that activates the
            unit tests
]
```

Listing 4.178: The projectConfig.gradle file content for activating unit tests

```
// In sample.ActivateMyUnitTests:
unitTestTask("MyUnitTest") // This activates unit tests.
```

Listing 4.179: Add a UnitTest task with name MyUnitTest

### 4.12.2.4 Model TestInfrastructure

The `ITransactionUndoAllExtension` class provides a `JUnit Extension`, which will undo all transaction after the `JUnit` test was executed. This can be used as static or instance field to undo for the whole test class or for each test case.

[4] Argument `-Dorg.gradle.java.home` is always needed when executing `gradlew` if you do NOT have Java 17 as the default Java version. If you have Java 17 as the default Java, you can skip this argument.

```
public class ProjectLoadTest {
    @RegisterExtension
    static ITransactionUndoAllExtension extension = ITransactionUndoAllExtension.
        forActiveProject(); // Undo changes after ALL test cases.
```

Listing 4.180: Usage of the ITransactionUndoAllExtension in a JUnit test

```
public class ProjectLoadTest {
    @RegisterExtension
    ITransactionUndoAllExtension extension = ITransactionUndoAllExtension.
        forActiveProject(); // Undo changes after EACH test case.
```

Listing 4.181: Usage of the ITransactionUndoAllExtension in a JUnit test

```
public class TransactionUndoAllExtensionTest {
    @RegisterExtension
    public ITransactionUndoAllExtension rule = ITransactionUndoAllExtension.
        forActiveProject();

    @Test
    public void testcase() {
        //After this test case all transactions are reverted by the
            TransactionUndoAllExtension
    }
```

Listing 4.182: Usage of the ITransactionUndoAllExtension in a JUnit test

## 4.13 Generator Testing

You can test Generators with an Automation Interface ScriptProject. There are two categories of generator tests, which can be performed:

- Black-box tests
- White-box tests

Both test categories are achieved by unit tests written in the ScriptProject, see chapter 4.12.2 on page 126 for how to write unit tests with the Automation Interface.

**Black-box tests** The Black-box tests test the functionality without knowing the internal structure of the generator. You can achieve this by using the Code Generation (see chapter 4.8.1 on page 101) and verify that the generated code is correct in your unit test.

### 4.13.1 White-box tests

The white-box tests test the functionality with knowledge about the internal structure of the generator. This allows you to test single classes and internal algorithms especially error cases of the generator.

You need the license option .MD for the white-box tests development and execution.

This is not available for script files, you need a script project.

**Technical description**: It will allow you to access and use all classes of the Generator including package-private classes and methods. The generator is loaded into your script during unit test execution.

#### 4.13.1.1 Step-by-step

You need to do the following steps:

1. Add the generator to your script project as compile dependency, see 4.13.1.2
   - Make sure the the generator package (field `generatorPackageName`) in the `build.gradle` file from you generator is full qualified
     - E.g. `generatorPackageName = "com.vector.cfg.gen.amsr_modelleddatatypes_api"`
2. Create your unit tests for your generator, see 4.12.2 on page 126
   - If you need a GeneratorPackage, see 4.13.1.3 on the next page
   - If you use transactions, see 4.12.2.4 on page 128
   - If you want to load projects, see 4.5.6 on page 56
3. Create a TestSuite, which references your unit test(s), see 4.12.2.1 on page 127. This is mandatory for generator white-box tests.
4. Execute your tests inside of a running DvDevA

#### 4.13.1.2 Compilation

To enable the access to the generator classes you have to specify the generator under test in your `build.gradle` file of your script project.

The `generatorTests` allows to configure generators under test. You can add multiple generators with `generator(<PATH-TO-GENERATOR-JAR-FILE>)`.

```
dvCfgAutomation {
  generatorTests{
     generator("<PATH-TO-GENERATOR-JAR-FILE>")
  }
}
```

Listing 4.183: DvDevA build Gradle DSL API - generatorTests

The specified generators are automatically added as compile dependency to your script project.

**Note:** You have to update the IntelliJ IDEA project, see 7.9.3.2 on page 155.

If you want to access GenDevKit classes or you reference GenDevKit classes indirectly, you have to specify a GenDevKit version:

```
dvCfgAutomation {
  generatorTests{
     generator("<PATH-TO-GENERATOR>")
     genDevKit("<VERSION>") //E.g. 21.00.10
  }
}
```

Listing 4.184: DvDevA build Gradle DSL API - generatorTests - genDevKit

The `genDevKit` method accepts the version as `String` or as path to the GenDevKit folder.

#### 4.13.1.3 GeneratorTestingApi

The `IGeneratorTestingApi` provides methods to test code with white box tests which require a real `IGeneratorPackage` or an `IGeneratorResultSink` etc. of the generator under test.

You can use the `IGeneratorTestingApi` with the call: `ScriptApi.scriptCode.genTesting`

The `createGenerator(Class)` method creates an `IGeneratorTestingAccess` for the passed class of the `IGeneratorPackage` under test. The method will use the currently active project for the generator. This will always create a new generator instance.

You can specify the used project with the method `createDeployablePackage(IProjectContext, Class)`.

The `withGenerator(Class, Closure)` method creates an `IGeneratorTestingAccess` and automatically activates it inside of the `Closure` code block.

**IGeneratorTestingAccess** The `IGeneratorTestingAccess` holds one created `IGeneratorPackage` under test and provides the following accessors:

- `getGeneratorPackage()`
- `getGenerationProcessor()`
- `getResultSink()`

The `IGeneratorPackage` needs to be activated before it is used in the test case. The best way to do it is with an JUnit `Extension`, see listing below. You could also use the `with(Closure)` method. Or you do it manually with `activate()` and `deactivate()`, but you have to use a `try/finally` construct.

**Examples** The following sample use the API to execute a `Spock` test:

```
@Ignore("DO NOT UN-IGNORE this test class. This is the input for the test
    GeneratorTestingSpockTest.")
class SpockGeneratorTest extends Specification {

    @Shared
    IGeneratorTestingAccess<TestGeneratorPackage> genUnderTest = ScriptApi.
        scriptCode.
    //We test the TestGeneratorPackage class
    genTesting.createGenerator(TestGeneratorPackage)

    def "GeneratorName" (){
        when:
        def gen = genUnderTest.generatorPackage

        then:
        gen.getGeneratorName() == "TestGenerator"
    }
```

Listing 4.185: Spock test using a GeneratorPackage

The following sample use the API to execute a `JUnit` test:

```
@Disabled("DO NOT UN-DISABLE this test class. This is the input for the test
    GeneratorTestingJunitTest.")
public class JunitGeneratorTest {

    public static IGeneratorTestingAccess<TestGeneratorPackage> genUnderTest =
        ScriptApi.scriptCode.
    //We test the TestGeneratorPackage class
    genTesting.createGenerator(TestGeneratorPackage)

    @Test
    public void testGeneratorName(){
        def gen = genUnderTest.generatorPackage
        assert gen.getGeneratorName() == "TestGenerator"
    }
```

Listing 4.186: JUnit test using a GeneratorPackage

#### 4.13.1.4 Test OuputFile Generation

You can also test the generation of `IOutputFile` classes with the `IGeneratorTestingApi`.

**Examples** The following sample use the API to execute a `Spock` test:

```
def "Content of TestOutputFile" (){
    when: "Generate the output file"
    def gen = genUnderTest.generatorPackage

    def dataRep = new TestDataRep(gen)
    def content = new TestOutputFile(gen, dataRep).generateFileContent()

    then:
    content == "TestFileContent" + NL
}
```

Listing 4.187: Spock test case which generates an OutputFile and verifies the content

The following sample use the API to execute a `JUnit` test:

```
@Test
public void testContentOfTestOutputFile(){
    def gen = genUnderTest.generatorPackage

    def dataRep = new TestDataRep(gen)
    def content = new TestOutputFile(gen, dataRep).generateFileContent()

    assert content == "TestFileContent" + NL
}
```

Listing 4.188: JUnit test case which generates an OutputFile and verifies the content

#### 4.13.1.5 Test GeneratorResults and ValidationResults

The `IGeneratorTestingAccess` provides a special `IGenTestingGeneratorResultSink` for testing purpose. You can retrieve it with `IGeneratorTestingAccess.getResultSink()`. The `IGenTestingGeneratorResultSink` can be used to call code under test which requires an `IGeneratorResultSink`.

The `IGenTestingGeneratorResultSink` provides access to assert that a certain code has created the expected `IValidationResult` objects. You can use the access API for `IValidationResultUI` objects provided by the validation , see chapter 4.9.2 on page 109 for details.

**Examples** The following sample use the result sink to execute a `Spock` test:

```
def "DataRep creates Error ValidationResult" (){
    when: "Execute the code under test"
    def gen = genUnderTest.generatorPackage
    def dataRep = new TestDataRep(gen)
    dataRep.createData()

    then: "Retrieve the validation result created by the code under test"
    def results =  genUnderTest.resultSink.
            validationResults.filter{ it.isId("TestGen", 123) }
    //Notice the id number doesn't contain any leading zero, since a number with
       leading zero is interpreted as an octal number.

    results.size() == 1
    results.first.description.toString() == "DescriptionText"
}
```

Listing 4.189: Spock test case which verifies a ValidationResult

The following sample use the result sink to execute a `JUnit` test:

```
@Test
public void dataRepCreatesErrorValidationResult(){
    def gen = genUnderTest.generatorPackage
    //Execute the code under test
    def dataRep = new TestDataRep(gen)
    dataRep.createData()

    //Retrieve the validation result created by the code under test
    def results =  genUnderTest.resultSink.
            validationResults.filter{ it.isId("TestGen", 123) }
    //Notice the id number doesn't contain any leading zero, since a number with
       leading zero is interpreted as an octal number.

    assert results.size() == 1
    assert results.first.description.toString() == "DescriptionText"
}
```

Listing 4.190: JUnit test case which verifies a ValidationResult

# 5. Data models in detail

This chapter describes several details and concepts of the involved data models. Be aware that the information here is focused on the Java API. In most cases it is more convenient using the Groovy APIs described in 4.7 on page 63. So, whenever possible use the Groovy API and read this chapter only to get background information when required.

## 5.1 MDF model - the raw AUTOSAR data

The MDF model is being used to store the AUTOSAR model loaded from several ARXML files. It consists of Java interfaces and classes which are generated from the AUTOSAR meta-model.

### 5.1.1 Naming

The MDF interfaces have the prefix `MI` followed by the AUTOSAR meta-model name of the class they represent. For example, the MDF interface related to the meta-model class `ARPackage` (AUTOSAR package in the top-level structure of the meta-model) is `MIARPackage`. The AUTOSAR meta model can be found for example on the AUTOSAR website.

### 5.1.2 The models inheritance hierarchy

The MDF model therefore implements (nearly) the same inheritance hierarchy and associations as defined by the AUTOSAR model. These interfaces provide access to the data stored in the model.

See figure 5.1 on the following page shows the (simplified) inheritance hierarchy of the implementation data type `MIImplementationDataType`. What we can see in this example:

- An `MIImplementationDataType` is an `MIAutosarDataType`, which is an `MIARElement`, which is an `MIIdentifiable` which again is a `MIReferrable`. The `MIReferrable` is the type which holds the shortname (getName()). All types which inherit from the `MIReferrable` have a shortname (MIARPackage, `MIARElement`, ...)
- A `MIImplementationDataType` is also a `MIHasSymbolProps`. This is an artificial base class (not part of the AUTOSAR meta-model) which provides all features of types which have symbol properties. The `MIAtomicSwComponentType` therefore has the same base type
- All `MIIdentifiables` can hold ADMIN-DATA and ANNOTATIONs
- All MDF objects in the AUTOSAR model tree inherit from `MIObject` which is again an `MIObject`

#### 5.1.2.1 MIObject and MDFObject

The `MIObject` is the base interface for all AUTOSAR model objects in the DvDevA data model. It extends `MDFObject` which is the base interface of all model objects. Your client code shall always use `MIObject`, when AUTOSAR model objects are used, instead of `MDFObject`.

The figure 5.2 on page 137 describes the class hierarchy of the `MIObject`.

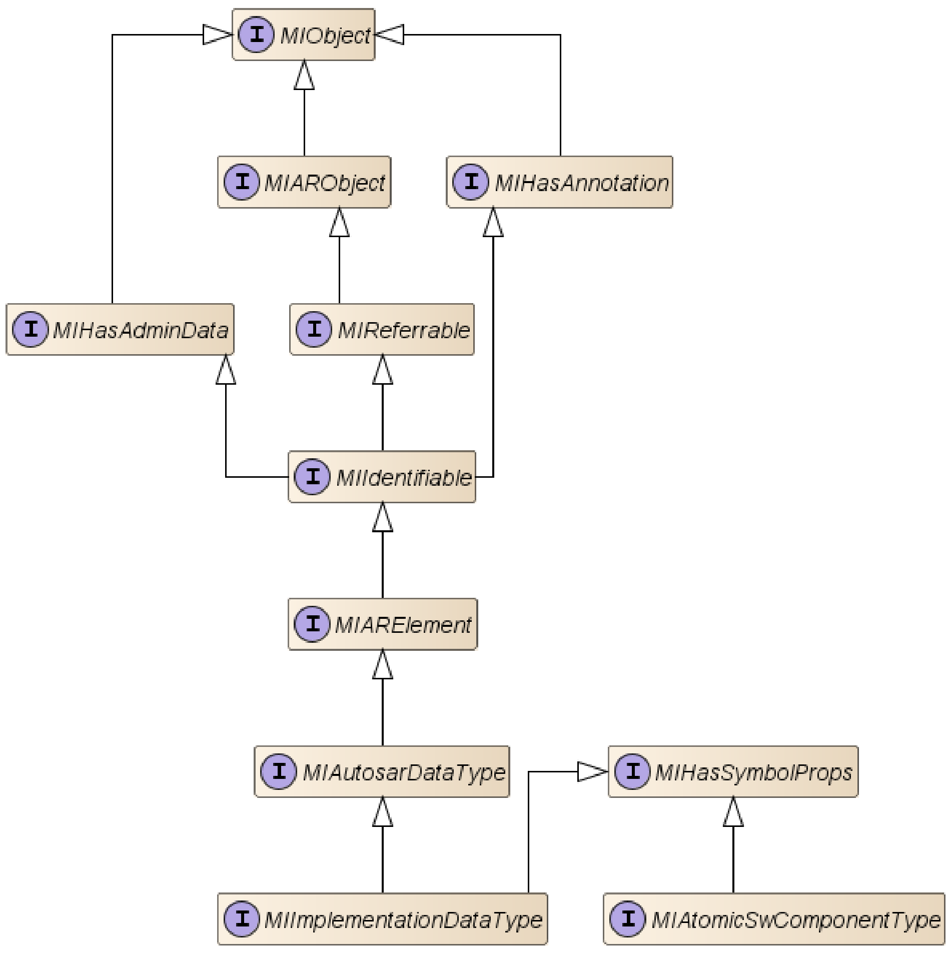


Figure 5.1: Implementation data type inheritance

### 5.1.3 The models containment tree

The root node of the AUTOSAR model is `MIAUTOSAR`. Starting at this object the complete model tree can be traversed. `MIAUTOSAR.getSubPackage()` for example returns a list of `MIARPackage` objects which again have child objects and so on.

Figure 5.3 on the following page shows a simple example of an MDF object containment hierarchy. This example contains two AUTOSAR packages with implementation data types below.

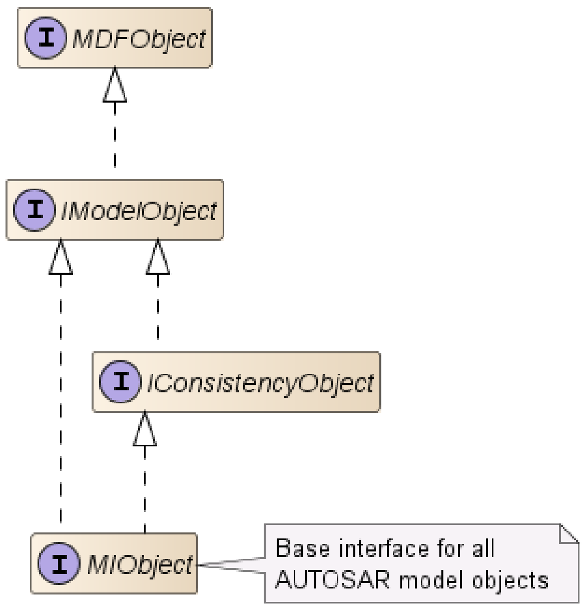


Figure 5.2: MIObject class hierarchy and base interfaces

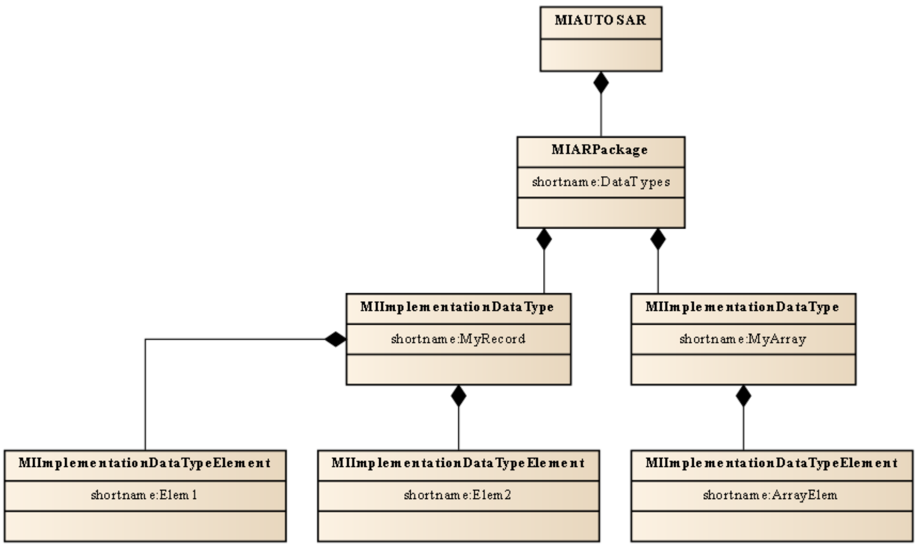


Figure 5.3: Autosar package containment

In general, objects which have child objects provide methods to retrieve them.

- `MIAUTOSAR.getSubPackage()` for example returns a list of child packages

### 5.1.4 Order of child objects

Child object lists in the MDF model have the same order as the data specified in the ARXML files. So, loading model objects from AXRML doesn't change the order.

All AUTOSAR reference objects in the MDF model have the base interface `MIARRef`.

Figure 5.4 shows this type hierarchy for the autosar type reference of a variable data prototype.

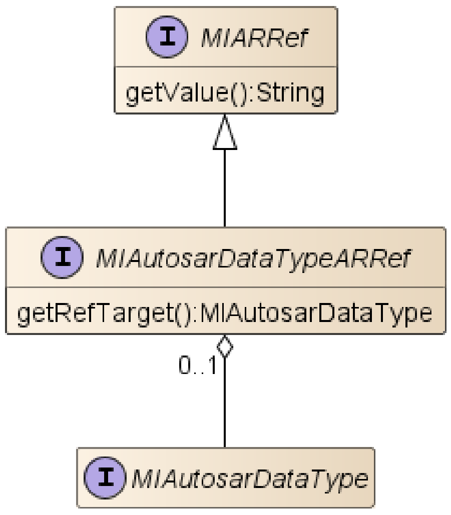


Figure 5.4: The autosar data type reference

In ARXML, such a reference can be specified as:

```
<TYPE-TREF DEST="IMPLEMENTATION-DATA-TYPE">
    /DataTypes/MyImplDataType
</TYPE-TREF>
```

- `MIARRef.getValue()` returns the AUTOSAR path of the object, the reference points to (as specified in the ARXML file). In the example above `"/DataTypes/MyImplDataType"` would be this value
- `MIAutosarDataTypeARRef.getRefTarget()` on the other hand returns the referenced MDF object if it exists. This method is located in a specific, typesafe (according to the type it points to) reference interface which extends `MIARRef`. So, if an object with the AUTOSAR path `"/DataTypes/MyImplDataType"` exists in the model, this method will return it

### 5.1.5 Model changes

#### 5.1.5.1 Transactions

The MDF model provides model change transactions for grouping several model changes into one atomic change.

A solving action, for example, is being executed within a transaction for being able to change model content. Validation and generator developers don't need to care for transactions. The tools framework mechanisms guarantee that their code is being executed in a transaction were required.

The tool guarantees that model changes cannot be executed outside of transactions. So, for example, during validation of model content the model cannot be changed. A model change here would lead to a runtime exception.

#### 5.1.5.2 Undo/redo

On basis of model change transactions, MDF provides means to undo and redo all changes made within one transaction. The tools GUI allows the user to execute undo/redo on this granularity.

#### 5.1.5.3 Event handling

MDF also supports model change events. All changes made in the model are reported by this asynchronous event mechanism. Validations, for example, detect this way which areas of the model need to be re-validated. The GUI listens to events to update its editors and views when model content changes.

#### 5.1.5.4 Deleting model objects

Model objects must be deleted via the `IModelOperationsPublished.deleteFromModel(MDFObject)`. service API.
Note: The delete operation requires a `transaction`.

Use `deleteFromModel(MIObject)` to delete specified object from the model. All associations of the object are deleted. The connection to its parent object gets deleted.
Inside the transaction closure, the system checks via CeState 5.2.4 on page 141 whether the object is deletable or not.

#### 5.1.5.5 Access to deleted objects

All subsequent access to content of deleted objects throws a runtime exception. Reading the shortname of an `MIIdendifiable`, for example.

#### 5.1.5.6 Set-methods

Model interfaces provide get-methods to read model content. MDF also offers set-methods for fields and child objects with multiplicity `0..1` or `1..1`.

These set-methods can be used to change model content.

- `MIARRef.getValue()` for example returns a references AUTOSAR path
- `MIARRef.setValue(String newValue)` sets a new path

#### 5.1.5.7 Changing child list content

MDF doesn't offer set-methods for fields and child objects with multiplicity `0..*` or `1..*`.

Changing child lists means changing the list itself.

- To add a new object to a child list, client code must use the lists add() method. The added object is being appended at the end of the list

- Removing child list objects is a side-effect of deleting this object. The delete operation removes it from the list automatically

#### 5.1.5.8 Change restrictions

The tools transaction handling implements some model consistency checks to avoid model changes which shall be avoided. Such changes are, for example:

- Creating duplicate shortnames below one parent object When client code tries to change the model this way, the related model change transaction is being canceled and the model changes are reverted (unconditional undo of the transaction). A special case here are solving actions. When a solving action inconsistently changes the model, only the changes made by this solving action are reverted (partial transaction undo of one solving action execution).

## 5.2 Model Utility Classes

### 5.2.1 AutosarUtil

The class `AutosarUtil` is a static utility class. It methods are not directly related to the MDF model but are useful when client code deals with AUTOSAR paths and shortnames on string basis.

Some of these methods are

- `isValidShortname(String)`: Checks if this shortname is valid according the rules, the AUTOSAR standard defines (character set for example)
- `getLastShortname(String)`: Returns the last shortname of the specified AUTOSAR path
- `getFirstShortname(String)`: Returns the first shortname of the specified AUTOSAR path
- `getAllShortnames(String)`: Returns all shortnames of the specified AUTOSAR path

### 5.2.2 AsrPath

The `AsrPath` class represents an AUTOSAR path without a connection to any model.

`AsrPaths` are constant; their values cannot be changed after they are created. This class is immutable!

### 5.2.3 AsrObjectLink

This class implements an immutable identifier for AUTOSAR objects.

An `AsrObjectLink` can be created for each object in the MDF AUTOSAR model tree. The main use case of object links is to identify an object unambiguously at a specific point in time for logging reasons. Additionally and under specific conditions it is also possible to find the related MDF object using its AsrObjectLink instance. But this search-by-link cannot be guaranteed after model changes (details and restrictions below).

#### 5.2.3.1 Restrictions of object links

- They are immutable and will therefore become invalid when the model changes
- So they don't guarantee that the related MDF object can be retrieved after the model has been changed. Search-by-link may even find another object or throw an exception in this case

### 5.2.4 CeState

The `CeState` is an object which allows to retrieve different states of a configuration entity.

The most important APIs for generator and script code are:

- `ICeStatePublished`

# 6. AutomationInterface Content

## 6.1 Introduction

This chapter describes the content of the DvDevA AutomationInterface.

## 6.2 Folder Structure

The AutomationInterface consists of the following files and folders:

- **Core**
  - **AutomationInterface**
    - **__doc** (find more details to its content in chapter 6.3)
      - **DVATS__AutomationInterfaceDocumentation.pdf:** this document
      - **javadoc:** Javadoc HTML pages
      - **templates:** script file and script project templates for a simple start of script development 6.1
    - **buildLibs:** AutomationInterface Gradle Plugin to provide the build logic to build script projects, see also 7.9 on page 152
    - **libs:** compile bindings to Groovy and to the DvDevA AutomationInterface, used by IntelliJ IDEA and Gradle
    - **licenses:** the licenses of the used open source libraries

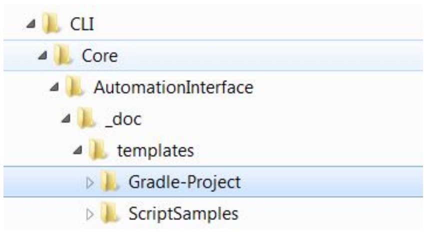


Figure 6.1: Script Project Template

## 6.3 Script Development Help

The help for the AutomationInterface script development is distributed among the following sources:

- DVATS__AutomationInterfaceDocumentation.pdf (this document)
- Javadoc HTML Pages
- Script Templates

### 6.3.1 AutomationInterfaceDocumentation PDF

You find this document as described in chapter 6.2 on the preceding page. It provides a good overview of architecture, available APIs and gives an introduction of how to get started in script development. The focus of the document is to provide an overview and not to be complete in API description. To get a complete and detailed description of APIs and methods use the Javadoc HTML Pages as described in 6.3.2.

### 6.3.2 Javadoc HTML Pages

You find this documentation as described in chapter 6.2 on the preceding page. Open the file `index.html` to access the complete DvDevA AutomationInterface API reference. It contains descriptions of all classes and methods that are part of the AutomationInterface.

The Javadoc is also accessible at your source code in the IDE for script development.

### 6.3.3 Script Templates

You find the Script Templates as described in chapter 6.2 on the previous page. You may copy them for a quick startup in script development.

## 6.4 Libs and BuildLibs

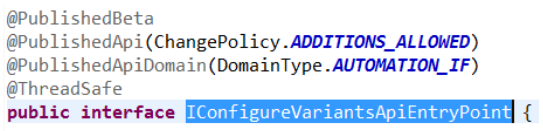
The AutomationInterface contains libraries to build projects, see **buildLibs** in 6.2 on the preceding page . And it contains other libraries which are described in **libs** in 6.2 on the previous page.

## 6.5 Beta API Usage

The beta annotation is exempt from any compatibility guarantees made by its containing library. Note that the presence of this annotation implies nothing about the quality or performance of the API in question, only the fact that it is not "API-frozen". For more details how to use Beta APIs in a script project, see 7.9.3.5 on page 160.

NOTE: Clients which uses this API must upgrade to each product release, to guarantee, that the used API is still available.

See below how beta annotation looks like.



```
@PublishedBeta
@PublishedApi(ChangePolicy.ADDITIONS_ALLOWED)
@PublishedApiDomain(DomainType.AUTOMATION_IF)
@ThreadSafe
public interface IConfigureVariantsApiEntryPoint {
```

Figure 6.2: Beta API Annotation

# 7. Automation Script Project

## 7.1 Introduction

An automation script project is a normal Java/Groovy development project, where the built artifact is a single `.jar` file. The jar file is created by the build system, see chapter 7.9 on page 152.

It is the recommended way to develop scripts, containing more tasks or multiple classes.

The project provides IDE support for:

- Code completion
- Syntax highlighting
- API Documentation
- Debug support
- Build support

The recommended IDE is IntelliJ IDEA.

## 7.2 Automation Script Project Creation

To create a new script project please follow the instructions in chapter 2.4 on page 9.

## 7.3 Project File Content

An automation project will at least contain the following files and folders:

- Folders
  - `.gradle` - Gradle temp folder - **DO NOT** commit it into a version control system
  - `build` - Gradle build folder - **DO NOT** commit it into a version control system
  - `gradle` - Gradle bootstrap folder - Please commit it into your version control system
  - `src` - Source folder containing your Groovy, Java sources and resource files
- Files
  - Gradle files - see 7.9.2 on page 152 for details
    - `gradlew.bat`
    - `build.gradle`
    - `settings.gradle`
    - `projectConfig.gradle`
    - `dvCfgAutomationBootstrap.gradle`
  - IntelliJ Project files (optional) - **DO NOT** commit it into a version control system

   ∗ `ProjectName.iws`
   ∗ `ProjectName.iml`
   ∗ `ProjectName.ipr`

The IntelliJ Project files (`*.iws, *.iml, *.ipr`) can be recreated with the command in the windows command shell (`cmd.exe`): `gradlew idea -Dorg.gradle.java.home="<PATH_TO_JDK_17>"`[1]

## 7.4 Deployment of the Jar File

To deploy your automation script project you only need to deploy the built jar file located in `<ProjectDir>/build/libs/<ProjectName>-<Version>.jar`. All other files in your automation script project are **not required** for the script **execution**.

If it is required to use an automation project script in the DvDevA, copy the jar file into script folder of the adaptive project.

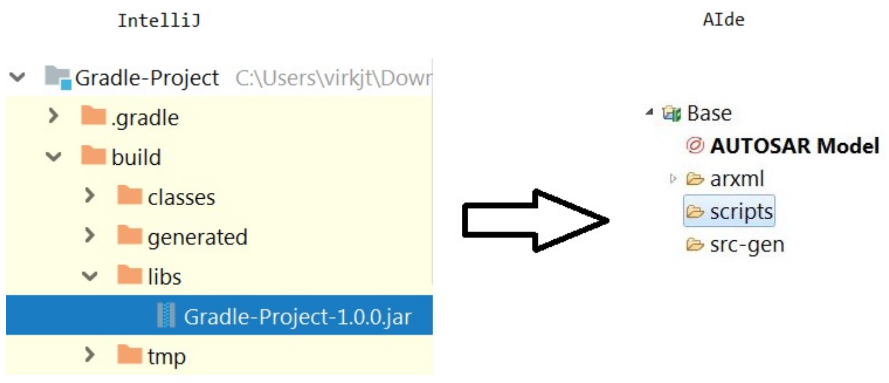


Figure 7.1: Generated Script Project

## 7.5 IntelliJ IDEA Usage

### 7.5.1 Supported versions

The supported IntelliJ IDEA versions are:

- 2021.x
- 2022.x

Please use one of the versions above which are tested with the DvDevA. With other versions, there could be problems with the editing, code completion and so on. The support for the IntelliJ IDEA versions prior to 2016.3 was removed. Please update your version.

The free **Community edition** is **fully sufficient**, but you could also use the *Ultimate edition.*

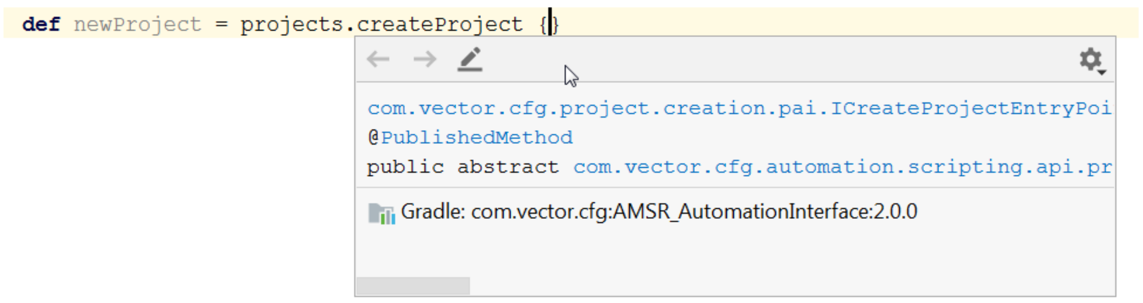
### 7.5.2 Show API Specifications (JavaDoc)

In newer IntelliJ versions the automatically download of the source files and their respective javadocs has been disabled. In order to benefit from the API-Specifications during coding it

[1]Argument `-Dorg.gradle.java.home` is always needed when executing `gradlew` if you do NOT have Java 17 as the default Java version. If you have Java 17 as the default Java, you can skip this argument.

is necessary to download the source files. This has to be done manually.

If source files are not yet downloaded, it looks like 7.2.



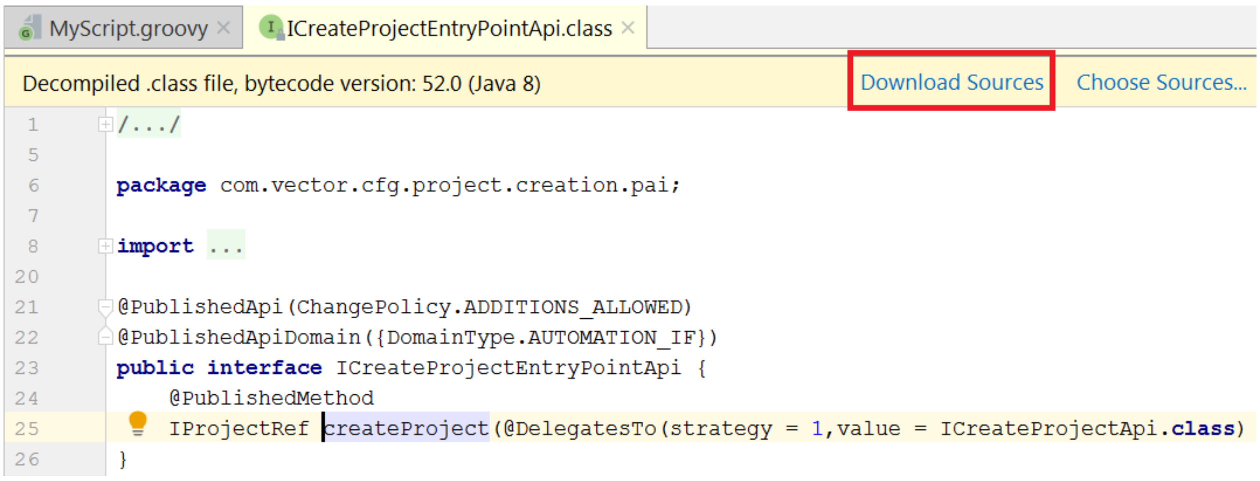
```
def newProject = projects.createProject {}
```

```
com.vector.cfg.project.creation.pai.ICreateProjectEntryPoi
@PublishedMethod
public abstract com.vector.cfg.automation.scripting.api.pr
```

Gradle: com.vector.cfg:AMSR_AutomationInterface:2.0.0

Figure 7.2: No JavaDoc

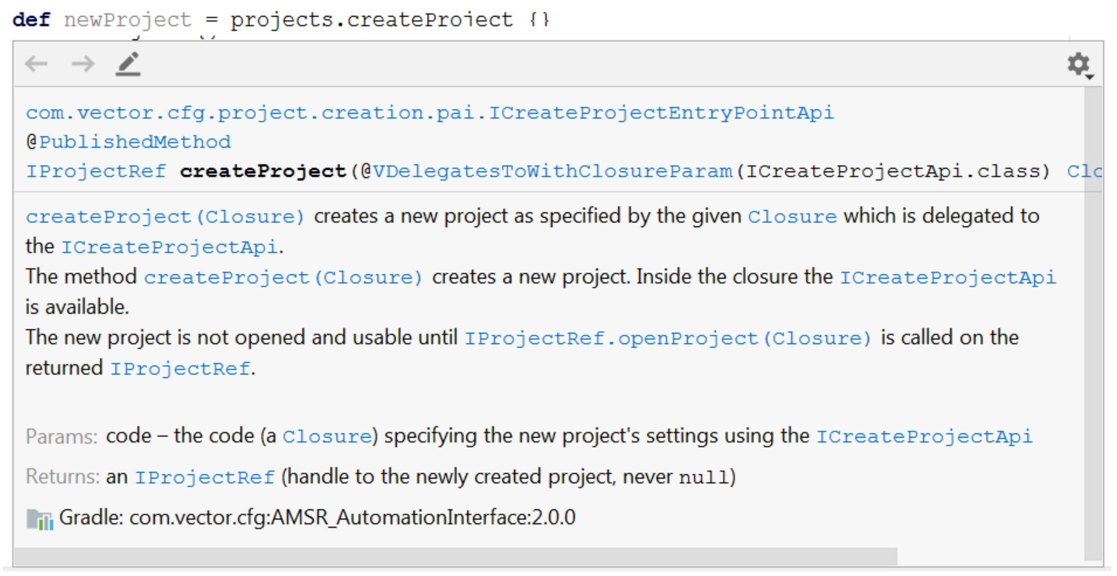
To download source files enter with F3 the API and click on "Download Sources" 7.3.



Decompiled .class file, bytecode version: 52.0 (Java 8)

```
/.../

package com.vector.cfg.project.creation.pai;

import ...

@PublishedApi(ChangePolicy.ADDITIONS_ALLOWED)
@PublishedApiDomain({DomainType.AUTOMATION_IF})
public interface ICreateProjectEntryPointApi {
    @PublishedMethod
    IProjectRef createProject(@DelegatesTo(strategy = 1,value = ICreateProjectApi.class)
}
```

Figure 7.3: No JavaDoc



```
def newProject = projects.createProject {}
```

```
com.vector.cfg.project.creation.pai.ICreateProjectEntryPointApi
@PublishedMethod
IProjectRef createProject(@VDelegatesToWithClosureParam(ICreateProjectApi.class) Clo
```

`createProject(Closure)` creates a new project as specified by the given `Closure` which is delegated to the `ICreateProjectApi`.
The method `createProject(Closure)` creates a new project. Inside the closure the `ICreateProjectApi` is available.
The new project is not opened and usable until `IProjectRef.openProject(Closure)` is called on the returned `IProjectRef`.

Params: code – the code (a `Closure`) specifying the new project's settings using the `ICreateProjectApi`

Returns: an `IProjectRef` (handle to the newly created project, never `null`)

Gradle: com.vector.cfg:AMSR_AutomationInterface:2.0.0

Figure 7.4: JavaDoc

### 7.5.3 Building Projects

**Project Build** The standard way to build projects is to choose the option **<ProjectName>** `build` in the Run Menu in the toolbar and to press the Run Button beneath that menu.

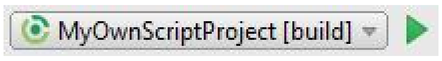


Figure 7.5: Project Build

**Project Continuous Build** A further option is provided for the case you prefer an automatic project building each time you save your implementation. If you choose the menu option **<ProjectName> continuous** `build` in the toolbar the Run Button has to be pressed only one time to start the continuous building. Hence forward each saving of your implementation triggers an automatic building of the script project.

**But be aware that the continuous build option is available for .java and .groovy files only.** In case of changes in e.g. .gradle files you still have to press the Run Button in order to build the project.

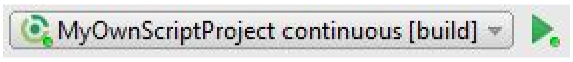


Figure 7.6: Project Continuous Build

The Continuous Build process can be stopped with the Stop Button in the Run View.

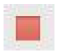
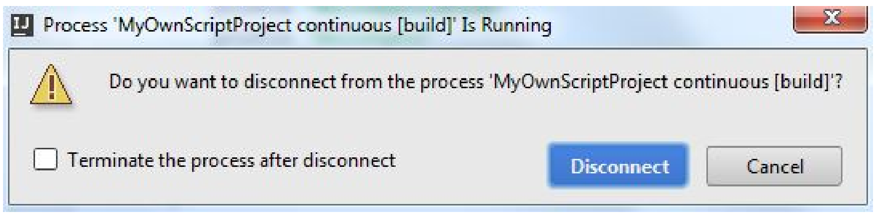


Figure 7.7: Stop Continuous Build

If you want to exit the IntelliJ IDEA while the Continuous Build process is still running, you will be asked to disconnect from it. Having disconnected you are allowed to exit the IDE.



Process 'MyOwnScriptProject continuous [build]' Is Running

Do you want to disconnect from the process 'MyOwnScriptProject continuous [build]'?

Figure 7.8: Disconnect from Continuous Build Process

### 7.5.4 Debugging with IntelliJ

**Be aware that only script projects and not script files are debuggable.**

To enable debugging you must start DvDevA application with the `-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=8000` option as described in 7.8 on page 151.
In the IntelliJ IDEA choose the option **<ProjectName>** `debug` in the Run Menu located in the toolbar. Pressing the Debug Button starts a debug session.

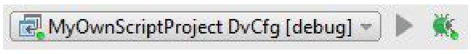


Figure 7.9: Project Debug

Set your breakpoints in IntelliJ IDEA and execute the task. To stop the debug session press the Stop Button in the Debugger View.

If you want to exit the IntelliJ IDEA while the Debug process is still running, you will be asked to disconnect from it. Having disconnected you are allowed to exit the IDE.

### 7.5.5 Troubleshooting

**Code completion, Compilation** If the code completion or compilation does not work, please verify that the Java JDK settings in the IntelliJ IDEA are correct. You have to set the Project JDK and the Gradle JDK setting. See 2.4.3 on page 13.
If code completion for adaptive model elements does not work, check the beta API switch 7.9.3.5 on page 160.

**Gradle build, build button** If the Gradle build does nothing after start or the build button is grayed, please verify that the Java JDK settings in the IntelliJ IDEA are correct. You have to set the Gradle JDK setting. See 2.4.3 on page 13.

If the build button is marked with an error, please make sure that the Gradle plugin inside of IntelliJ IDEA is installed. Open File->Settings...->Plugins and select the Gradle plugin.

**IntelliJ Build** You shall not use the IntelliJ menu "Build" or the context menu entries "Make Project", "Make Module", "Rebuild Project" or "Compile". The project shall be build with Gradle

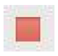
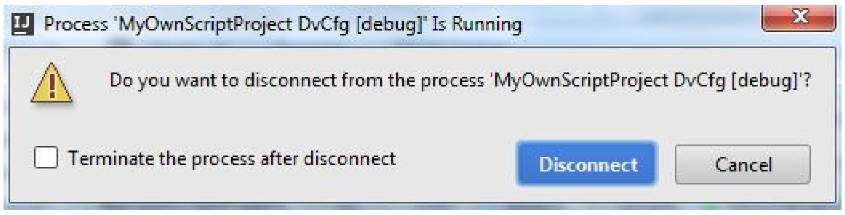


Figure 7.10: Stop Debug Session



Process 'MyOwnScriptProject DvCfg [debug]' Is Running

Do you want to disconnect from the process 'MyOwnScriptProject DvCfg [debug]'?

☐ Terminate the process after disconnect

Figure 7.11: Disconnect from Debug Process

not with IntelliJ IDEA. So you have to select one of the Run Configuration (Run menu) to build the project as described in chapter 7.5 on page 145.

**Groovy SDK not configured** If you get the message 'Groovy SDK is not configured for ...' in IntelliJ IDEA you probably have to migrate your project as described in chapter 7.7 on the next page.

**No JavaDoc Shown** If you don't see a javadoc description for the APIs. See chapter 7.5.2 on page 145.

**Compile errors - Could not find com.vector.cfg:DVCfgAutomationInterface** If you get compile errors inside of the IntelliJ IDEA, after updating the DvDevA or moving projects.

Please execute the **Project Migration to newer DaVinci Developer Adaptive Version** step, see 7.7 on the next page.

**Download of Gradle Distribution Error** If you get an error when you start the `gradlew` like:

```
Downloading
http://<SomeRemoteServer>/gradle-7.5.1-bin.zip

Exception in thread "main" java.io.FileNotFoundException:
http://<SomeRemoteServer>/gradle-7.5.1-bin.zip
at sun.net.www.protocol.http.HttpURLConnection.getInputStream0(HttpURLConnection.java:1836)
```

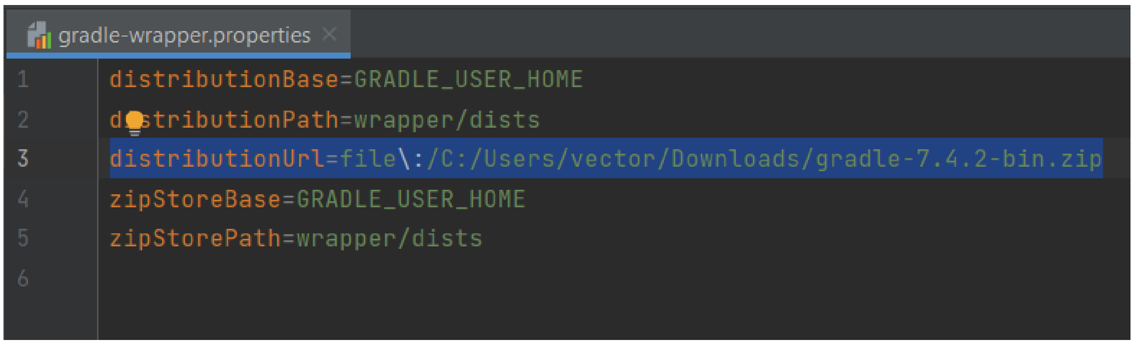
The problem is you can't connect to the server, where the Gradle installation is located[2]. To change the location 7.12 on the following page, you have to open the file `<ProjectRoot>/gradle/wrapper/gradle-wrapper.properties` and change the line `distributionUrl=`.

You have multiple options for the content of the `distributionUrl`:

- Change the URL to the Gradle default (needs internet access):
  - `https\://services.gradle.org/distributions/gradle-7.5.1-bin.zip`
- Change the URL to a Server location of your choice. E.g inside your company.
- Download Gradle manually and change the URL to a local file system location like:

[2]The vector internal server `http://vistrcfgci1.vi.vector.int` is not accessible from outside of the vector network and shall only be used by internal projects. If you have a project with the internal server and your are not inside the network, please change it to another location.

   – `file\:/C:/YourFolder/gradle-7.5.1-bin.zip`



gradle-wrapper.properties

```
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
distributionUrl=file\:/C:/Users/vector/Downloads/gradle-7.4.2-bin.zip
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
```

Figure 7.12: File: gradle-wrapper.properties

**Caution:** You have to escape a `:` with `\:` so an HTTP address would start with `http\://` and the local filesystem would start with `file\:/`.

So the default line in the file ‘gradle-wrapper.properties‘ for the default Gradle server would be:
`distributionUrl=https\://services.gradle.org/distributions/gradle-7.5.1-bin.zip`

## 7.6 Project Usage in different DvDevA Versions

You can execute the script tasks of a script project in different versions of the DvDevA as long as the following conditions are met:

- Compile your script project always against the **oldest** DvDevA version
  - **Request:** Script needs to run in DvDevA_v1 and DvDevA_v2
  - **Required:** Compile script against DvDevA_v1 (oldest)
- The DvDevA version span **must not** contain a breaking change. These changes are documented in the chapter 8 on page 161. Normally the versions have **NO** breaking changes!

## 7.7 Project Migration to newer DvDevA Version/BSW Package

If you update your DvDevA version in your BSW Package or if you copy the project into another BSW Package, you should execute the Gradle task to update the compile dependencies.

Steps to execute:

1. Close IntelliJ IDEA.
2. Make sure distributionUrl in `gradle/wrapper/gradle-wrapper.properties` is poniting to gradle-7.4.2-bin.zip
3. Update the DvDevA in your BSW Package / Change the path to the DvDevA in `projectConfig.gradle`
4. Update the IntelliJ IDEA project, see 7.9.3.2 on page 155

This will update the compile time dependencies of your Automation Script Project according to the new DvDevA version.

After this, please read the Changes (see chapter 8 on page 161) in the new release and update your script, if something of interest has changed.

### 7.7.1 Migration to DvDevA2.10

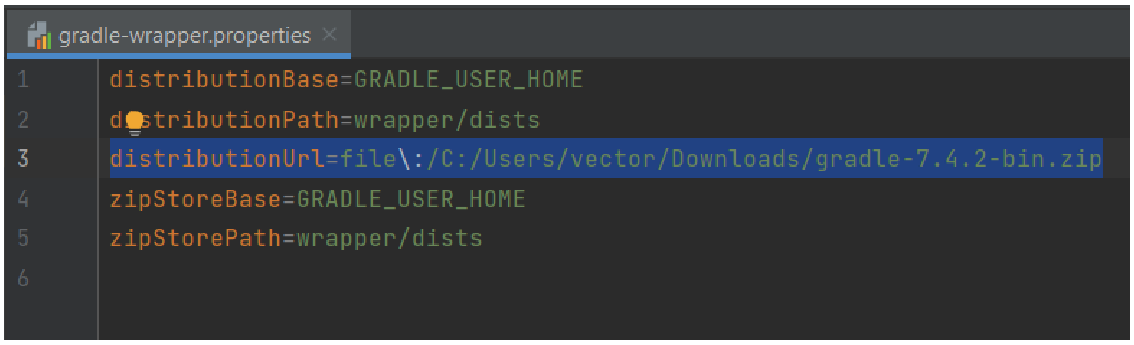
In DvDevA2.10, we have a breaking change and bump the API version to 4.0.0. This indicates that your script projects built on API 3.x.x need to be migrated before you can execute them in DvDevA2.10 or later version.

#### 7.7.1.1 Requirements

- You must have **JDK 17** installed in order to build the script project.
- Download gradle-7.4.2-bin.zip
- IntelliJ IDEA.

#### 7.7.1.2 Step by step

1. In `<ProjectRoot>/gradle/wrapper/gradle-wrapper.properties`, change the value of `distributionUrl` pointing to a gradle-7.4.2-bin.zip, it could be either a local or remote URL.
   E.g. `https\://services.gradle.org/distributions/gradle-7.4.2-bin.zip`
   or `file\:/home/vector/Downloads/gradle-7.4.2-bin.zip`
   or `file\:/C:/Users/vector/Downloads/gradle-7.4.2-bin.zip`



```
gradle-wrapper.properties
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
distributionUrl=file\:/C:/Users/vector/Downloads/gradle-7.4.2-bin.zip
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
```

Figure 7.13: Set distributionUrl

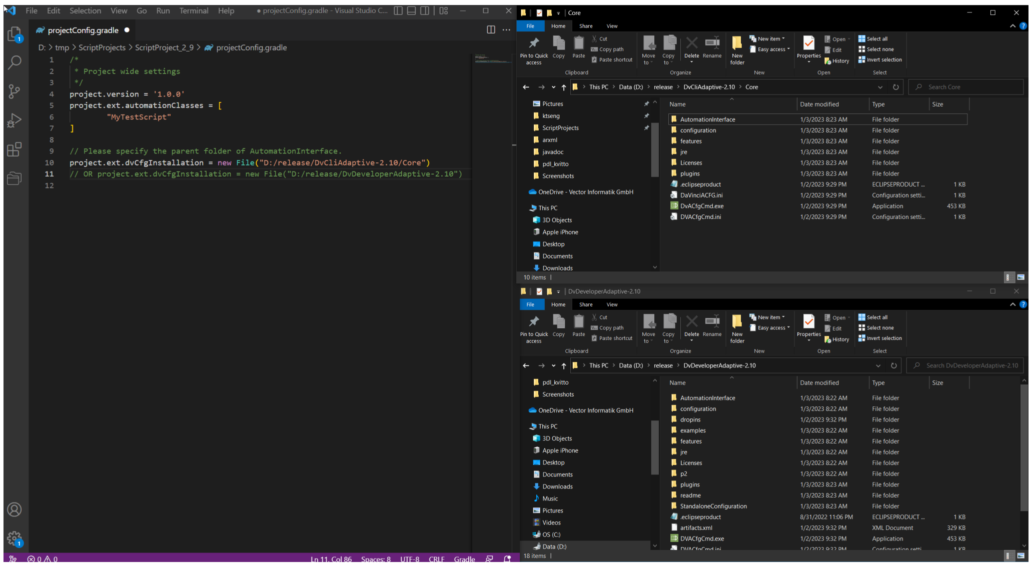
2. Make sure `project.ext.dvCfgInstallation` in `<ProjectRoot>/projectConfig.gradle` point to the parent folder of AutomationInterface in DvDevA2.10.
   E.g. `project.ext.dvCfgInstallation = new File("D:/release/DvCliAdaptive-2.10/Core")`
3. Open the command line at project root folder and run command
   `gradlew idea -Dorg.gradle.java.home="<PATH_TO_JDK_17>"`.[3]
4. Open IntelliJ IDEA press **Reload All Gradle Projects** in gradle tool window. This will change files `<Project>.ipr` and `<Project>.main.iml` to setup the correct dependencies.
5. Build the script project.

## 7.8 Debugging Script Project

**Be aware that only script projects and not script files are debuggable.**

To debug a script project, any java debugger could be used. Simply add the `-vmargs -agentlib:jdwp=transpor`
parameters when starting the DvDevA application.

[3]Argument `-Dorg.gradle.java.home` is always needed when executing `gradlew` if you do NOT have Java 17 as the default Java version. If you have Java 17 as the default Java, you can skip this argument.



D: > tmp > ScriptProjects > ScriptProject_2_9 > projectConfig.gradle

```
/*
 * Project wide settings
 */
project.version = '1.0.0'
project.ext.automationClasses = [
        "MyTestScript"
]

// Please specify the parent folder of AutomationInterface.
project.ext.dvCfgInstallation = new File("D:/release/DvCliAdaptive-2.10/Core")
// OR project.ext.dvCfgInstallation = new File("D:/release/DvDeveloperAdaptive-2.10")
```

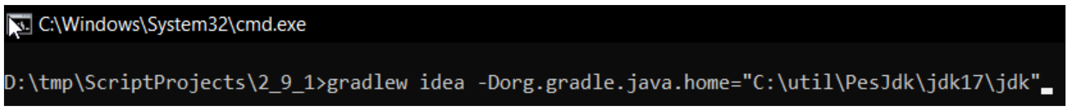
This PC > Data (D:) > release > DvCliAdaptive-2.10 > Core

| Name | Date modified | Type | Size |
|---|---|---|---|
| AutomationInterface | 1/3/2023 8:23 AM | File folder | |
| configuration | 1/3/2023 8:23 AM | File folder | |
| features | 1/3/2023 8:23 AM | File folder | |
| jre | 1/3/2023 8:23 AM | File folder | |
| Licenses | 1/3/2023 8:23 AM | File folder | |
| plugins | 1/3/2023 8:23 AM | File folder | |
| .eclipseproduct | 1/2/2023 9:29 PM | ECLIPSEPRODUCT ... | 1 KB |
| DaVinciACFG.ini | 1/2/2023 9:29 PM | Configuration setti... | 1 KB |
| DvACfgCmd.exe | 1/2/2023 9:29 PM | Application | 453 KB |
| DVACfgCmd.ini | 1/2/2023 9:29 PM | Configuration setti... | 1 KB |

This PC > Data (D:) > release > DvDeveloperAdaptive-2.10

| Name | Date modified | Type | Size |
|---|---|---|---|
| AutomationInterface | 1/3/2023 8:22 AM | File folder | |
| configuration | 1/3/2023 8:22 AM | File folder | |
| dropins | 1/2/2023 9:32 PM | File folder | |
| examples | 1/3/2023 8:22 AM | File folder | |
| features | 1/3/2023 8:22 AM | File folder | |
| jre | 1/3/2023 8:22 AM | File folder | |
| Licenses | 1/3/2023 8:22 AM | File folder | |
| p2 | 1/3/2023 8:22 AM | File folder | |
| plugins | 1/3/2023 8:23 AM | File folder | |
| readme | 1/3/2023 8:23 AM | File folder | |
| StandaloneConfiguration | 1/3/2023 8:23 AM | File folder | |
| .eclipseproduct | 8/31/2022 11:06 PM | ECLIPSEPRODUCT ... | 1 KB |
| artifacts.xml | 1/2/2023 9:32 PM | XML Document | 329 KB |
| DVACfgCmd.exe | 1/2/2023 9:32 PM | Application | 453 KB |

Figure 7.14: Set dvCfgInstallation



C:\Windows\System32\cmd.exe

```
D:\tmp\ScriptProjects\2_9_1>gradlew idea -Dorg.gradle.java.home="C:\util\PesJdk\jdk17\jdk"
```

Figure 7.15: Run gradlew idea with JDK 17

Following an example batch file (start_IDE.bat) to start the DvDevA application in debug mode. Do not add any line feed.

```
start eclipse.exe -data ./ws -vmargs
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=8000
```

Default debugger at port `8000`. This port can be changed.

## 7.9 Build System

The build system uses Gradle[4] to build a single Jar file. It also setups the dependencies to the DvDevA and create the IntelliJ IDEA project.

To setup the Gradle installation, see chapter 2.4.4 on page 14.

### 7.9.1 Jar Creation and Output Location

The call to `gradlew build -Dorg.gradle.java.home="<PATH_TO_JDK_17>"`[5] in the root directory of your automation script project will create the jar file. The jar file is then located in:

- `<ProjectRoot>\build\libs\<ProjectName>-<ProjectVersion>.jar`

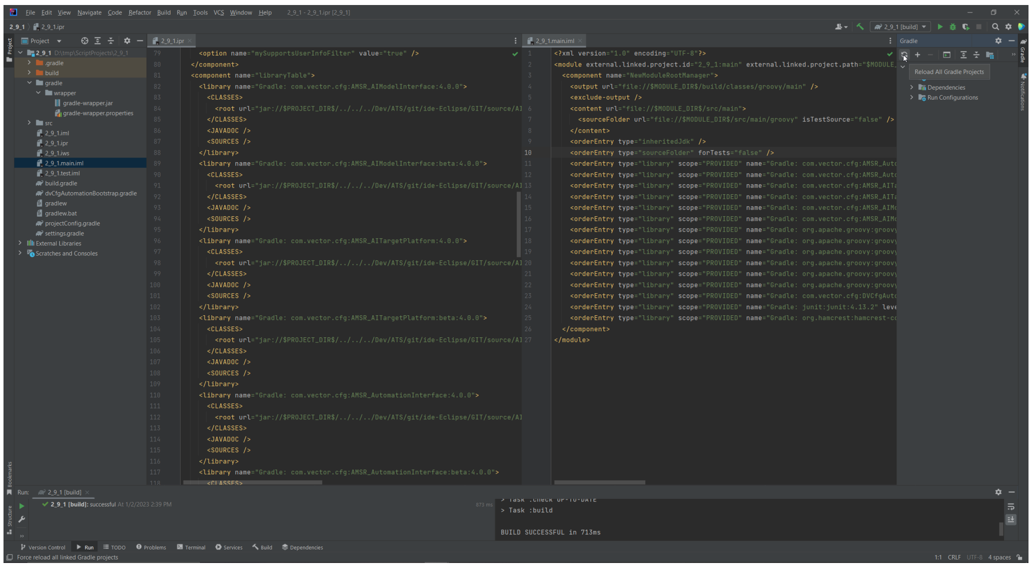
### 7.9.2 Gradle File Structure

The default automation project contains the following Gradle build files:

- `gradlew.bat`

[4] `https://gradle.org/` [2018-11-27]

[5] Argument `-Dorg.gradle.java.home` is always needed when executing `gradlew` if you do NOT have Java 17 as the default Java version. If you have Java 17 as the default Java, you can skip this argument.



```
    <option name="mySupportsUserInfoFilter" value="true" />
  </component>
  <component name="libraryTable">
    <library name="Gradle: com.vector.cfg:AMSR_AIModelInterface:4.0.0">
      <CLASSES>
        <root url="jar://$PROJECT_DIR$/../../../Dev/ATS/git/ide-Eclipse/GIT/source/AI
      </CLASSES>
      <JAVADOC />
      <SOURCES />
    </library>
    <library name="Gradle: com.vector.cfg:AMSR_AIModelInterface:beta:4.0.0">
      <CLASSES>
        <root url="jar://$PROJECT_DIR$/../../../Dev/ATS/git/ide-Eclipse/GIT/source/AI
      </CLASSES>
      <JAVADOC />
      <SOURCES />
    </library>
    <library name="Gradle: com.vector.cfg:AMSR_AITargetPlatform:4.0.0">
      <CLASSES>
        <root url="jar://$PROJECT_DIR$/../../../Dev/ATS/git/ide-Eclipse/GIT/source/AI
      </CLASSES>
      <JAVADOC />
      <SOURCES />
    </library>
    <library name="Gradle: com.vector.cfg:AMSR_AITargetPlatform:beta:4.0.0">
      <CLASSES>
        <root url="jar://$PROJECT_DIR$/../../../Dev/ATS/git/ide-Eclipse/GIT/source/AI
      </CLASSES>
      <JAVADOC />
      <SOURCES />
    </library>
    <library name="Gradle: com.vector.cfg:AMSR_AutomationInterface:4.0.0">
      <CLASSES>
        <root url="jar://$PROJECT_DIR$/../../../Dev/ATS/git/ide-Eclipse/GIT/source/AI
      </CLASSES>
      <JAVADOC />
      <SOURCES />
    </library>
    <library name="Gradle: com.vector.cfg:AMSR_AutomationInterface:beta:4.0.0">
      <CLASSES>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<module external.linked.project.id="2_9_1:main" external.linked.project.path="$MODULE_
  <component name="NewModuleRootManager">
    <output url="file://$MODULE_DIR$/build/classes/groovy/main" />
    <exclude-output />
    <content url="file://$MODULE_DIR$/src/main">
      <sourceFolder url="file://$MODULE_DIR$/src/main/groovy" isTestSource="false" />
    </content>
    <orderEntry type="inheritedJdk" />
    <orderEntry type="sourceFolder" forTests="false" />
    <orderEntry type="library" scope="PROVIDED" name="Gradle: com.vector.cfg:AMSR_Auto
    <orderEntry type="library" scope="PROVIDED" name="Gradle: com.vector.cfg:AMSR_Auto
    <orderEntry type="library" scope="PROVIDED" name="Gradle: com.vector.cfg:AMSR_AITa
    <orderEntry type="library" scope="PROVIDED" name="Gradle: com.vector.cfg:AMSR_AITa
    <orderEntry type="library" scope="PROVIDED" name="Gradle: com.vector.cfg:AMSR_AIMo
    <orderEntry type="library" scope="PROVIDED" name="Gradle: com.vector.cfg:AMSR_AIMo
    <orderEntry type="library" scope="PROVIDED" name="Gradle: org.apache.groovy:groovy
    <orderEntry type="library" scope="PROVIDED" name="Gradle: org.apache.groovy:groovy
    <orderEntry type="library" scope="PROVIDED" name="Gradle: org.apache.groovy:groovy
    <orderEntry type="library" scope="PROVIDED" name="Gradle: org.apache.groovy:groovy
    <orderEntry type="library" scope="PROVIDED" name="Gradle: org.apache.groovy:groovy
    <orderEntry type="library" scope="PROVIDED" name="Gradle: org.apache.groovy:groovy
    <orderEntry type="library" scope="PROVIDED" name="Gradle: com.vector.cfg:DVCfgAuto
    <orderEntry type="library" scope="PROVIDED" name="Gradle: junit:junit:4.13.2" leve
    <orderEntry type="library" scope="PROVIDED" name="Gradle: org.hamcrest:hamcrest-co
  </component>
</module>
```

```
2_9_1 [build]: successful At 1/2/2023 2:39 PM    873 ms
```

```
> Task :build

BUILD SUCCESSFUL in 713ms
```

Figure 7.16: Reload All Gradle Projects in IntelliJ IDEA

  - Gradle batch file to start Gradle (Gradle Wrapper[6])

- `build.gradle`
  - General build file - You can modify it to adapt the build to your needs
- `settings.gradle`
  - General build project settings - See Gradle documentation[7]
- `projectConfig.gradle`
  - Contains automation project specific settings - You can modify it to adapt the build to your needs
- `dvCfgAutomationBootstrap.gradle`
  - This is the internal bootstrap file. **DO NOT** change the file content.

#### 7.9.2.1 projectConfig.gradle File settings

The file contains three essential parts of the build:

- Names of the scripts to load (automationClasses)
- The path to the DvDevA installation (dvCfgInstallation)
- Project version (version)

**automationClasses** You **have to add your classes** to the list of `automationClasses` to make them loadable.

The syntax of `automationClasses` is a list of `Strings`, of all classes as full qualified Class names. Syntax: `"javaPkg.subPkg.ClassName"`

[6]`https://docs.gradle.org/current/userguide/gradle_wrapper.html` [2018-11-27]

[7]`https://docs.gradle.org/current/dsl/org.gradle.api.initialization.Settings.html` [2018-11-27]

```
// The property project.ext.automationClasses defines the classes to load
project.ext.automationClasses = [
        "sample.MyScript",
        "otherPkg.MyOtherScript",
        "javapkg.ClassName"
]
```

Listing 7.1: The automationClasses list in projectConfig.gradle

**dvCfgInstallation** The `dvCfgInstallation` defines the path to the DvDevA installation in your BSW Package 2.3 on page 12. The installation is needed to retrieve the build dependencies and the generated model.

You could also evaluate SystemEnv variables, other project properties or Gradle settings to define the path dependent of the development machine, instead of encoding an absolute path. This will help, when the project is committed to a version control system. But this is project dependent and out of scope of the provided template project.

```
// Use a System environment variable as path to the DvDevA
project.ext.dvCfgInstallation = new File(System.getenv('YOUR_ENV_VARIABLE'))
```

Listing 7.2: The dvCfgInstallation with an System env in projectConfig.gradle

**version** The `project.version` defines the version of your Automation project, e.g. defines the version suffix of the jar file.

### 7.9.3 Advanced Build Topics

#### 7.9.3.1 Usage of external Libraries (Jars) in the AutomationProject

You could reference external libraries (Jar files) in your AutomationProject. But you have to configure the libraries in the Gradle build files. **DO NOT** add a dependency in IntelliJ, this will not work.

The preferred way is the use a library from any Maven repository like MavenCentral or JCenter. This will also handle versions, and transitive dependencies automatically.

Otherwise you could download the jar file an place it in your project[8], but this is **NOT** recommended.

The referenced libraries will be automatically bundled into your Automation project, see chapter 7.9.3.5 on page 159 `includeDependenciesIntoJar`for details.

**How to add a Library?** We assume we have a jar from a Maven repository like `Apache Commons IO` (the identifier would be `'commons-io:commons-io:2.5'`, See MavenCentral).

- Open your `build.gradle`
- Add the code for the dependency

```
dependencies {
    // Change the identifier to your library to use
    compile 'commons-io:commons-io:2.5'
    // You could add multiple libraries with additional compile lines
}
```

[8]See Gradle online documentation, how to add local jar files to the build dependencies

- Optional: if you are behind a proxy or firewall:
  - You must either set proxy options for gradle [9]
  - **Prefered way**: use a Maven repository inside your network: To set a repository, add before the dependencies block:

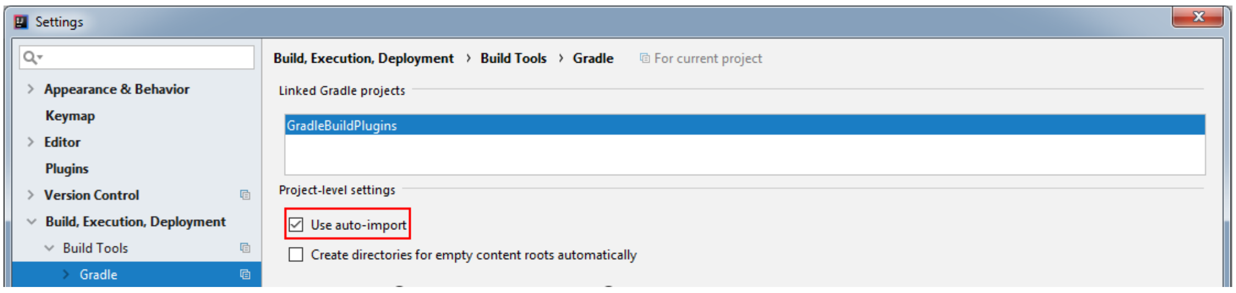
```
repositories {
    // URL to your repository
    // The URL below is the Vector internal network server
    // Please change the URL to your server
    maven { url 'https://vistrpesart1.vi.vector.int/artifactory/pes-davinciall-
        maven' }
    // Or reference MavenCentral server
    mavenCentral()
}
```

- Update the IntelliJ IDEA project, see 7.9.3.2

Now your project has access to the specified library.

#### 7.9.3.2 Update IntelliJ IDEA project

If you have changed dependencies or versions of any library, you have to update your IDEA project. The best way to do this is to select the Gradle auto-import feature in the IntelliJ IDEA, see figure 7.17.



☑ Use auto-import
☐ Create directories for empty content roots automatically

Figure 7.17: Activate the auto-import

Or you can manually execute the following steps:

1. Close IntelliJ IDEA.
2. Open a command shell (`cmd.exe`) at your project folder
   - Folder containing the `gradlew.bat`
3. Type `gradlew idea -Dorg.gradle.java.home="<PATH_TO_JDK_17>"`[10] and press enter
4. Wait until the task has finished
5. Open IntelliJ IDEA

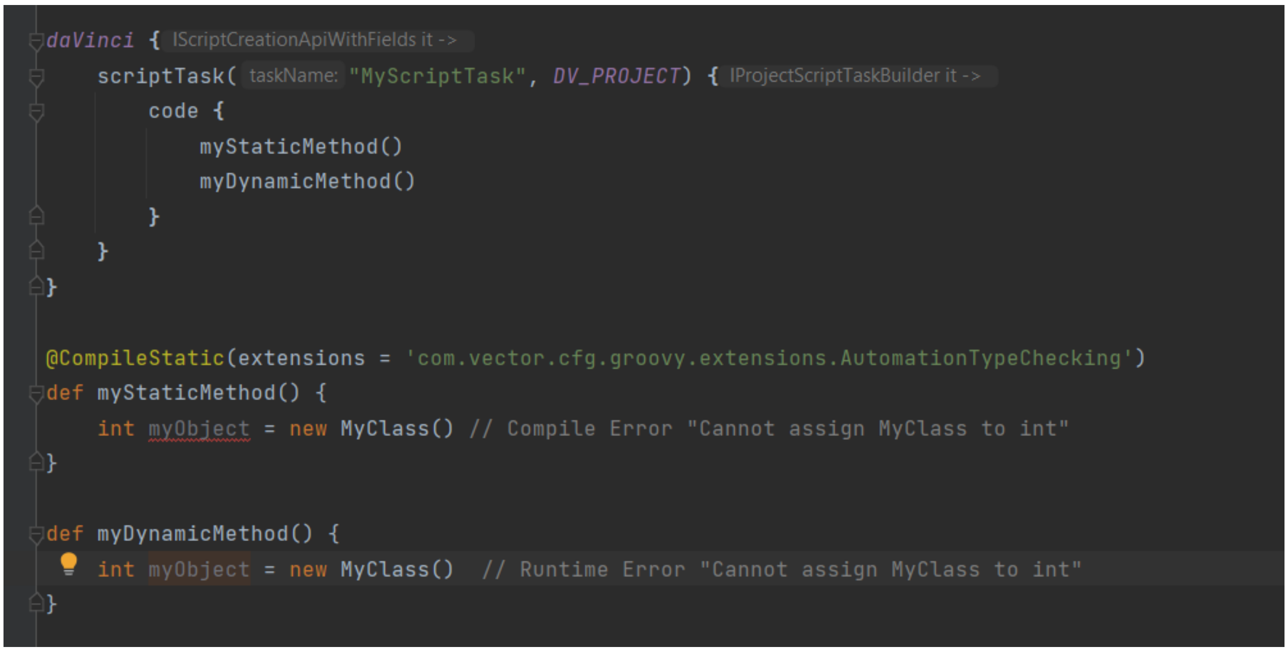
#### 7.9.3.3 Static Compilation of Groovy Code

The AutomationInterface contains a Groovy compiler extension. In some use cases performance has priority and therefore it is possible to use Groovy with static compilation.

[9] Gradle and Java online documentation for details how to set proxy settings

[10] Argument `-Dorg.gradle.java.home` is always needed when executing `gradlew` if you do NOT have Java 17 as the default Java version. If you have Java 17 as the default Java, you can skip this argument.

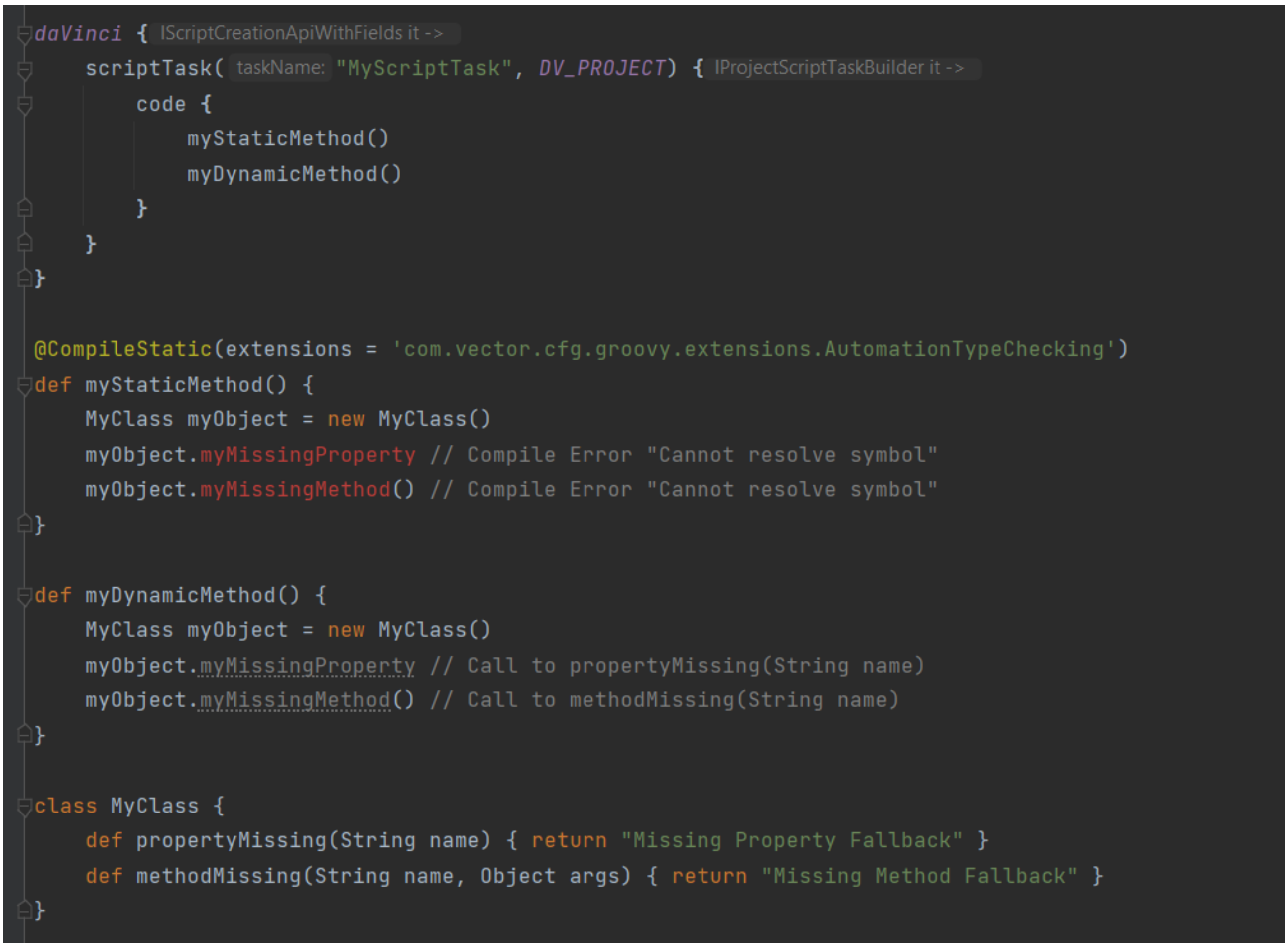
Groovy is a dynamic JVM language using dynamic dispatch for its method calls. Dynamic dispatch in Groovy is approximately three times slower compared to a normal Java method call. Groovy has added the static compilation feature via @CompileStatic annotation, which allows to compile most of Groovy method calls into direct JVM bytecode method calls, thus avoiding all the dynamic dispatch overhead.[11])



```
daVinci { IScriptCreationApiWithFields it ->
    scriptTask( taskName: "MyScriptTask", DV_PROJECT) { IProjectScriptTaskBuilder it ->
        code {
            myStaticMethod()
            myDynamicMethod()
        }
    }
}

@CompileStatic(extensions = 'com.vector.cfg.groovy.extensions.AutomationTypeChecking')
def myStaticMethod() {
    int myObject = new MyClass() // Compile Error "Cannot assign MyClass to int"
}

def myDynamicMethod() {
    int myObject = new MyClass()  // Runtime Error "Cannot assign MyClass to int"
}
```

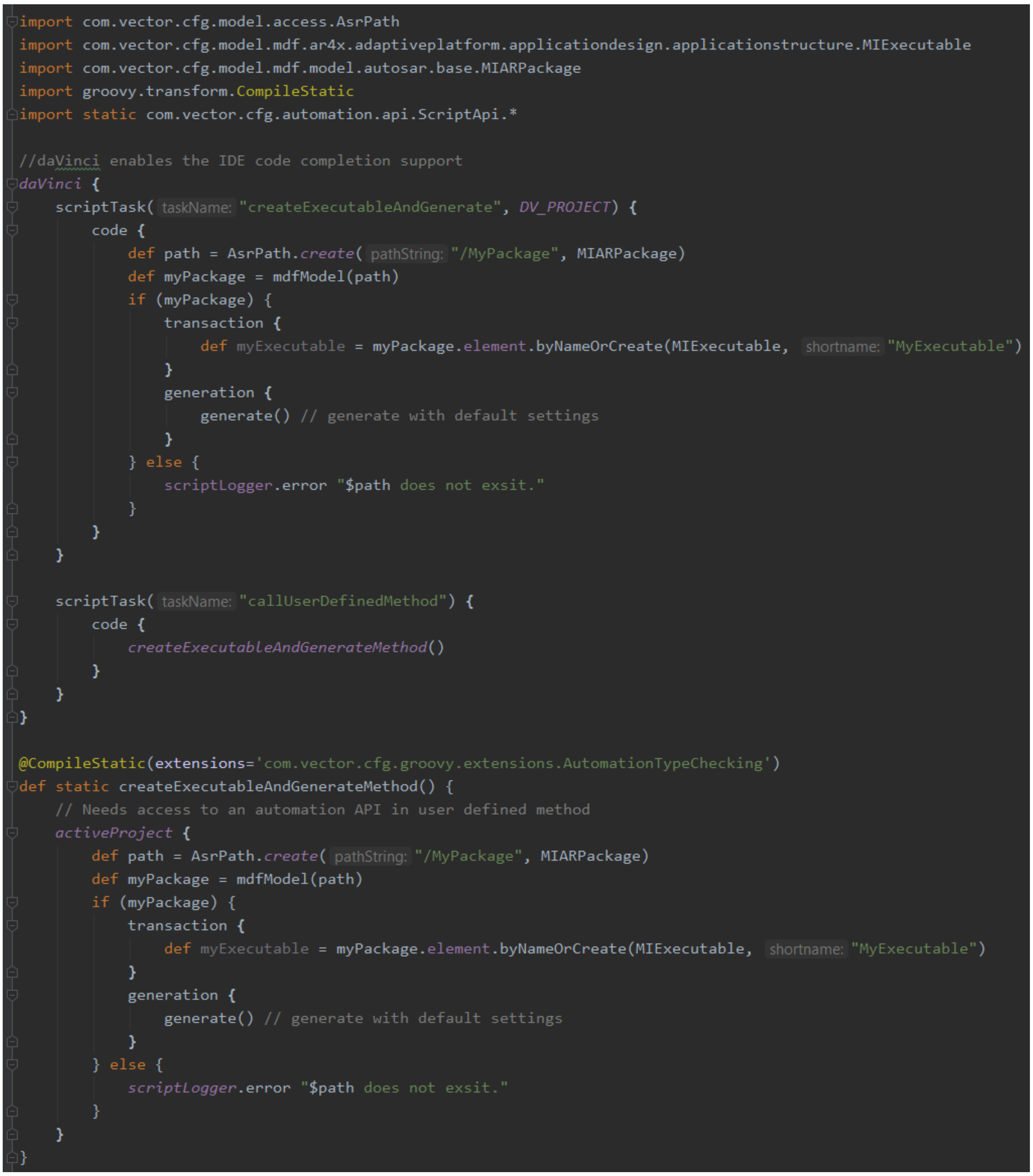
Figure 7.18: Type checking with @CompileStatic

[11]`http://java-performance.info/static-code-compilation-groovy-2-0/` [2018-11-29]



```
daVinci { IScriptCreationApiWithFields it ->
    scriptTask( taskName: "MyScriptTask", DV_PROJECT) { IProjectScriptTaskBuilder it ->
        code {
            myStaticMethod()
            myDynamicMethod()
        }
    }
}

@CompileStatic(extensions = 'com.vector.cfg.groovy.extensions.AutomationTypeChecking')
def myStaticMethod() {
    MyClass myObject = new MyClass()
    myObject.myMissingProperty // Compile Error "Cannot resolve symbol"
    myObject.myMissingMethod() // Compile Error "Cannot resolve symbol"
}

def myDynamicMethod() {
    MyClass myObject = new MyClass()
    myObject.myMissingProperty // Call to propertyMissing(String name)
    myObject.myMissingMethod() // Call to methodMissing(String name)
}

class MyClass {
    def propertyMissing(String name) { return "Missing Property Fallback" }
    def methodMissing(String name, Object args) { return "Missing Method Fallback" }
}
```

Figure 7.19: Dynamic features with @CompileStatic



```
import com.vector.cfg.model.access.AsrPath
import com.vector.cfg.model.mdf.ar4x.adaptiveplatform.applicationdesign.applicationstructure.MIExecutable
import com.vector.cfg.model.mdf.model.autosar.base.MIARPackage
import groovy.transform.CompileStatic
import static com.vector.cfg.automation.api.ScriptApi.*

//daVinci enables the IDE code completion support
daVinci {
    scriptTask( taskName: "createExecutableAndGenerate", DV_PROJECT) {
        code {
            def path = AsrPath.create( pathString: "/MyPackage", MIARPackage)
            def myPackage = mdfModel(path)
            if (myPackage) {
                transaction {
                    def myExecutable = myPackage.element.byNameOrCreate(MIExecutable,  shortname: "MyExecutable")
                }
                generation {
                    generate() // generate with default settings
                }
            } else {
                scriptLogger.error "$path does not exsit."
            }
        }
    }

    scriptTask( taskName: "callUserDefinedMethod") {
        code {
            createExecutableAndGenerateMethod()
        }
    }
}

@CompileStatic(extensions='com.vector.cfg.groovy.extensions.AutomationTypeChecking')
def static createExecutableAndGenerateMethod() {
    // Needs access to an automation API in user defined method
    activeProject {
        def path = AsrPath.create( pathString: "/MyPackage", MIARPackage)
        def myPackage = mdfModel(path)
        if (myPackage) {
            transaction {
                def myExecutable = myPackage.element.byNameOrCreate(MIExecutable,  shortname: "MyExecutable")
            }
            generation {
                generate() // generate with default settings
            }
        } else {
            scriptLogger.error "$path does not exsit."
        }
    }
}
```

Figure 7.20: Refactor code into a user defind method with @CompileStatic

As this annotation is only applicable to classes or methods, it cannot be used in plain scripts. Thus, it is required to refactor the plain script into classes and class methods. For more information see chapter 4.4.8 on page 41.

**What is the difference between @TypeChecked and @CompileStatic?** The `@CompileStatic` annotation can be seen as a subclass of `@TypeChecked`. While `@TypeChecked` only enforces compile-time type checking without changing behavior at runtime, `@CompileStatic` additionally makes code compile to static JVM bytecode, trading the Groovy dynamic language features for increased runtime performance.

#### 7.9.3.4 Gradle Maven publishing of an AutomationProject

#### 7.9.3.5 Gradle dvCfgAutomation API Reference

The DvDevA build system provides a Gradle DSL API to set properties of the build. The entry point is the keyword `dvCfgAutomation`

```
dvCfgAutomation {
    classes project.ext.automationClasses
}
```

Listing 7.3: build.gradle

The following methods are defined inside of the `dvCfgAutomation` block in the build.gradle file:

- `classes` (Type `List<String>`) - Defines the automation classes to load
- `useJarSignerDaemon` (Type `boolean`) - Enables or disables the usage of the Jar Signer Daemon process.
- `setJarSignerWaitTimeoutMin` (Type `int`) - Sets the timeout for the jar signer daemon (0 means infinite wait time).
- `includeDependenciesIntoJar` (Type `boolean`) - Enables or disables the inclusion of dependencies during build
- `allowBetaApiUsage` (Type `boolean`) - Enables or disables the usage of beta APIs

**useJarSignerDaemon** The `useJarSignerDaemon` enables or disables the usage of the Jar Signer Daemon process. The process is spawned when a jar file shall be signed. This will speedup the build process especially when the project is built often. The daemon is closed automatically, when not used in a certain time span.

The default of `useJarSignerDaemon` is `true`.

The Gradle task `stopJarSignerDaemon` will stop any running Signer daemon.

```
dvCfgAutomation {
   useJarSignerDaemon true
}
```

Listing 7.4: build.gradle - useJarSignerDaemon

**setJarSignerWaitTimeoutMin** The `setJarSignerWaitTimeoutMin` sets the timeout for the jar signer daemon (0 means infinite wait time). The default is set to 30 minutes, if the property is not configured. The `setJarSignerWaitTimeoutMin` only gets used if `useJarSignerDaemon` is `true`.

```
dvCfgAutomation {
    setJarSignerWaitTimeoutMin 10
}
```

Listing 7.5: build.gradle - setJarSignerWaitTimeoutMin

**includeDependenciesIntoJar** The `includeDependenciesIntoJar` enables or disables bundling of gradle runtime dependencies (e.g. referenced jar files) into the resulting project jar. If `includeDependenciesIntoJar` is enabled the project jar file will contain all jar dependencies under the folder `jars` inside of the jar file.

The default of `includeDependenciesIntoJar` is `true`.

```
dvCfgAutomation {
    includeDependenciesIntoJar false
}
```

Listing 7.6: build.gradle - includeDependenciesIntoJar

**allowBetaApiUsage** The `allowBetaApiUsage` enables or disables the usage of beta APIs. See 3.1 on page 17 for more information.

CAUTION: By setting this flag, you lose the guarantee that this script will work in future releases.

```
dvCfgAutomation {
    allowBetaApiUsage true
}
```

Listing 7.7: build.gradle - allowBetaApiUsage

**generatorTests** The `generatorTests` allows to configure generator tests, see chapter 4.13.1.2 on page 130 for details.

# 8. AutomationInterface Changes between Versions

This chapter describes the supported functionality of different versions and all API changes between different MICROSAR releases.

## 8.1 Currently Supported Features

The table below contains a list of functionalities of the DvDevA Automation Interface.

**Legend**: A functionality is available if the `Since` column contains the DvDevA version (see Since). Otherwise the functionality is not yet available.

| Component | Functionality | Since |
|---|---|---|
| Scripts | Loading, Execution, Script-Projects | `ATS 2.3` |
| | User defined Script Task Arguments | `ATS 2.3` |
| | Stateful Script Tasks | `ATS 2.3` |
| | Remove Generation Step from Script Task Types | `DvDevA 2.7` |
| | Add Mex Migration into Script Task Types | `DvDevA 2.8` |
| Project | Open, modify, save and close project | `ATS 2.3` |
| | Accessing the active UI project | `ATS 2.3` |
| | Create a new project | `ATS 2.3` |
| | Accessing/Modifying the target project settings | `ATS 2.3` |
| | Accessing/Modifying the useCase project settings | `ATS 2.3` |
| | Open ARXML files as Project | `ATS 2.3` |
| | Access to project search | `ATS 2.3` |
| | Create a new adaptive project | `DvDevA 2.7` |
| Model | Access to the whole AUTOSAR model (SystemDesc) | `ATS 2.3` |
| | Transaction support (Undo, Redo) | `ATS 2.3` |
| | CE-States: UserDefined, Changeable, Deletable | `ATS 2.3` |
| | Access and modification of User Annotations at the configuration element | `ATS 2.3` |
| | Unresolved references API | `ATS 2.3` |
| | Relative search for model element based on a root element | `ATS 2.3` |
| | Access to model extension stored in ADMIN-DATA in a generic way | `DvDevA 2.7` |
| Generation | Generate code for specific modules | `ATS 2.3` |
| | Generate code for predefined code generation sequence | `ATS 2.3` |
| | Modify code generation sequence to enable/disable specific modules or generation steps | `ATS 2.3` |
| | SWC Templates and Contract Headers Generation | `ATS 2.3` |
| | Add a ScriptTask as external generation step | `ATS 2.3` |
| | Add generation report settings | `ATS 2.3` |
| | Remove external generation steps | `DvDevA 2.7` |
| Validation | Access to project validation result | `ATS 2.3` |
| | Access to validation results of specific model elements | `ATS 2.3` |

| | | |
|---|---|---|
| | Solve valdiation results (by group, by id, by solving action type (preferred solving action)) | `ATS 2.3` |
| | Request a full validation for the whole project | `DvDevA 2.7` |
| Persistency | Export of configuration artefacts | `ATS 2.3` |
| | Export of AUTOSAR model trees | `ATS 2.3` |
| Testing | Unit test execution with Junit | `ATS 2.3` |
| | Unit test execution with Spock | `ATS 2.3` |
| | Generator Black-box tests | `ATS 2.3` |
| | Generator White-box tests | `ATS 2.3` |

## 8.2 Changes in MICROSAR AR4-R24 - ATS2.3

### 8.2.1 Create Project

New Create Project API added to create a new ATS Project. See 4.5.3 on page 53 for details.

## 8.3 Changes in MICROSAR AR4-R25 - ATS2.4

### 8.3.1 General

The ATS2.4 (AR4-R25) automation interface is compatible to the ATS2.3. So a script written with ATS2.3 will also run in the ATS2.4 version.

#### 8.3.1.1 Gradle

Support for Gradle version 6.5 added.

## 8.4 Changes in MICROSAR AR4-R28 - DvDevA2.7

### 8.4.1 General

The DvDevA2.7 (AR4-R28) automation interface is compatible to the DvDevA2.6. So a script written with ATS2.6 will also run in the DvDevA2.7 version.

### 8.4.2 Deprecated external generation step

External generation step is no longer supported in DvDevA, DV_GENERATION_STEP and relative sample code are removed.

### 8.4.3 Full Validation

A new method `fullValidation()` is added for the validation, see section 4.9.3 on page 110 for details.

### 8.4.4 Model Extension API

Model Extension API supports to access model extension data, which is stored in ADMIN-DATA, in a generic way. See 4.7.6 on page 87 for details.

### 8.4.5 Create Project API

The Create Project API is now deprecated and will be removed. The new Create Adaptive Project API to create a new Adaptive Project has been added. See 4.5.3 on page 53 for details.

## 8.5 Changes in MICROSAR AR4-R29 - DvDevA2.8

### 8.5.1 Model Extension Migration Types

A task type `DV_MEX_MIGRATION` is introduced for the model extension migration, see section 4.3.1.5 on page 27 for details.

## 8.6 Changes in MICROSAR AR4-R30 - DvDevA2.9

### 8.6.1 Model Extension API

New methods `getRefTargets()`, `getMexRefTarget()` and `getMexRefTargets()` are added in `IMexReference`. See 4.7.6.5 on page 92 for details. New methods `isReferrable()`, `getAllReferencesPointingTo()`, `getAllReferencesPointingToSorted()`, `getAllMexReferencesPointingTo()` and `getAllMexReferencesPointingToSorted()` are added in `IMexObject`. See 4.7.6.6 on page 94 for details.

## 8.7 Changes in DvDevA2.10

### 8.7.1 Groovy Version

Included version of Groovy changed from version 2.5 to version 4.0

### 8.7.2 Migration

Script projects created before DvDevA2.10 need to be migrated. See 7.7.1 on page 151 for details.

### 8.7.3 Model Extension API

Introduced `Strict Mode` in 4.7.6 on page 87. New methods `childObjectCreateAndAdd(String)` and `childObjectWithNameCreateAndAdd(String, String)` are added in `IMexObject`. See 4.7.6.6 on page 94 for details.

### 8.7.4 IArxmlFile

Introduced `IArxmlFile` in 4.6 on page 61.

### 8.7.5 File mapping APIs

See Splitting Model Objects 4.7.3.10 on page 77, Moving Model Objects 4.7.3.11 on page 77 and Copying Model Objects 4.7.3.12 on page 78 for details.

### 8.7.6 Creating and adding Child List Members

Changes have been introduced to the methods in Creating and adding Child List Members (0:*) 4.7.3.6 on page 72 and Updating existing Elements 4.7.3.7 on page 75. The option to specify a location for new objects has been added and the creation of objects to multiple files are no longer allowed.

## 8.8 Changes in DvDevA2.12

### 8.8.1 IArxmlFile

Added delete() method for `IArxmlFile` in 4.6 on page 61.

### 8.8.2 File mapping APIs

Added section: Deleting a Model Object from a file 4.7.3.9 on page 77.

## 8.9 Changes in DvDevA2.13

### 8.9.1 Project References

Added section: Change project references in 4.5.5 on page 56.

### 8.9.2 Creating single Child Members (0:1)

Support choosing the arxml file when creating child object in 4.7.3.5 on page 71.

## 8.10 Changes in DvDevA2.14

### 8.10.1 Model exporter excluding elements under a certain path

Model exporter 'excludePath' allows exporting a merged model into a file while excluding elements located under a specified path. See examples in 4.10.1.1 on page 119.

### 8.10.2 Model Automation API

#### 8.10.2.1 Transaction History API

New method `clearUndoRedoHistory()` has been added. See 4.7.4.1 on page 82 for details.

## 8.11 Changes in DvDevA2.15

### 8.11.1 Project duplication

Support duplicating a project. See examples in 4.5.4 on page 55.

## 8.12 Changes in DvDevA2.16

### 8.12.1 Generation

#### 8.12.1.1 Set output folder

The output folder can be customized in the settings of generation. See chapter 4.8.1.1 on page 102 for details.

#### 8.12.1.2 Select scopes

Support scope selection for validation/generation. See chapter 4.8.1.1 on page 105 for details.

### 8.12.2 Wizard API

Wizard API provides support to create models in one go with consistent configuring. Methods `createPackage(Closure)` and `createServiceInterfaceDeployment(Closure)` are now available. See 4.7.7 on page 99 for details.

# 9. Appendix

# Nomenclature

| | |
|---|---|
| *AI* | Automation Interface |
| *ATS* | DaVinci Developer Adaptive |
| *AUTOSAR* | Automotive Open System Architecture |
| *CE* | Configuration Entity (typically a container or parameter) |
| *DV* | DaVinci |
| *IDE* | Integrated Development Environment |
| *JAR* | Java Archive |
| *JDK* | Java Development Kit |
| *JRE* | Java Runtime Environment |
| *MDF* | Meta-Data-Framework |
| *MSN* | ModuleShortName |

# Figures

# Tables

# Listings

# Todo list