



**Documentation**

**AutoRelease Generic SCons Integration**

**ITK Engineering AG**   
www.itk-engineering.de  
Im Speyerer Tal 6  
76761 Rülzheim  
Tel: +49-7272-7703-0  
Fax: +49-7272-7703-100

1. General

|  |  |
| --- | --- |
| Document state: | Draft |
| Date: |  |
| Version: |  |
| Author: | Roland Stark |
| Project: | AutoRelease Script |

Change history

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Date | Rev. | Description | Author |
| 1 | 15.04.2014 | 0.1.0 | Initial revision. | RoSt |
| 2 | 17.04.2014 | 0.2.0 | Changes against 1st concept. | RoSt |
| 3 | 04.07.2014 | 0.3.0 | Changed from concept to technical documentation. | RoSt |
| 4 | 04.07.2014 | 0.4.0 | Documented kind of delivery confirmation. | RoSt |
| 5 | 11.07.2014 | 0.5.0 | Updated order of release steps.  Updated description of setup selection. | RoSt |
| 6 | 11.07.2014 | 0.5.1 | Improved. Document review by Andre Fischer. | RoSt |
| 7 | 01.08.2014 | 0.6.0 | Added documentation of improved consistency check.  Placed parsing of project configuration prior to the consistency check. | RoSt |
| 8 | 07.08.2014 | 0.6.1 | Disabled custom setup selection for less than 3 setups.  Added version number check.  User confirmation before handover document is opened. | RoSt |
| 9 | 07.08.2014 | 0.6.2 | New file name for handover document. If not exiting the file is added to the sandbox. | RoSt |
| 10 | 11.08.2014 | 0.6.3 | Ignoring unused shared projects. | RoSt |
| 11 | 12.08.2014 | 0.6.4 | Added customized shared shared projects. | RoSt |
| 12 | 19.08.2014 | 0.6.5 | Added check for shared sandboxes not of type “build”.  Removed labeling of single members. | RoSt |
| 13 | 19.08.2014 | 0.6.6 | Added recursive member labeling. | RoSt |
| 14 | 20.08.2014 | 0.6.7 | Supports old and new SDL file delivery structure. | RoSt |
| 15 | 21.08.2014 | 0.7.0 | All changes from meeting 05 Aug are included.  Handled delivered include files. | RoSt |
| 16 | 15.10.2014 | 0.8.0 | Changed menu item text for “select all setups”.  Added 'Accepted' issues to issue list.  Updated re-scan feature at change package selection. | RoSt |
| 17 | 20.10.2014 | 0.9.0 | Added freezing/thawing members. | RoSt |
| 18 | 23.10.2014 | 0.10.0 | Changed code handover format to text.  Added validation of code handover. | RoSt |
| 19 | 24.10.2014 | 0.11.0 | Added code handover history list.  Changed revision label format.  Added interruption for manual component tests. | RoSt |
| 20 | 27.10.2014 | 0.12.0 | Added adaptive library check-in based on delivery type.  Removed files in 06\_Test\_Tools from consistency check and check-in. | RoSt |
| 21 | 30.10.2014 | 0.13.0 | Handled include files that are not shared from its member path.  Added mode for delivery from build path (needed by SR). | RoSt |

1. Terms, abbreviations and definitions
   1. Terms
   2. Abbreviations
   3. Definitions
2. Content

[I General 2](#_Toc402434981)

[II Terms, abbreviations and definitions 3](#_Toc402434982)

[II.1 Terms 3](#_Toc402434983)

[II.2 Abbreviations 3](#_Toc402434984)

[II.3 Definitions 3](#_Toc402434985)

[III Content 4](#_Toc402434986)

[IV List of figures 5](#_Toc402434987)

[V List of tables 5](#_Toc402434988)

[1 Scope of the document 6](#_Toc402434989)

[2 General description 6](#_Toc402434990)

[2.1 Configuration management 6](#_Toc402434991)

[2.2 Activation 7](#_Toc402434992)

[2.3 Build targets 7](#_Toc402434993)

[2.4 Integration as Python Builder 8](#_Toc402434994)

[3 Configuration 9](#_Toc402434995)

[3.1 Static configuration 9](#_Toc402434996)

[3.1.1 Component owner 9](#_Toc402434997)

[3.1.2 Project configuration 9](#_Toc402434998)

[3.1.3 Project root directory 9](#_Toc402434999)

[3.1.4 Kind of delivery 10](#_Toc402435000)

[3.1.5 Release setups 10](#_Toc402435001)

[3.2 Dynamic configuration 11](#_Toc402435002)

[3.2.1 Compliance report link 11](#_Toc402435003)

[3.2.2 Related releases 11](#_Toc402435004)

[3.2.3 Additional checkpoints 12](#_Toc402435005)

[4 AutoRelease sequence 13](#_Toc402435006)

[4.1 Initial environment 13](#_Toc402435007)

[4.2 Project Information 14](#_Toc402435008)

[4.2.1 Project URL and Development path 14](#_Toc402435009)

[4.2.2 Project name 14](#_Toc402435010)

[4.3 Type of delivery 15](#_Toc402435011)

[4.4 Freeze / Thaw members 15](#_Toc402435012)

[4.4.1 Low-level functions 16](#_Toc402435013)

[4.4.2 Hi-level function 16](#_Toc402435014)

[4.4.3 Output 16](#_Toc402435015)

[4.5 Parsing project configuration 17](#_Toc402435016)

[4.6 Consistency check 19](#_Toc402435017)

[4.6.1 Member revisions 19](#_Toc402435018)

[4.6.2 Check shared projects 20](#_Toc402435019)

[4.6.3 Shared project version 21](#_Toc402435020)

[4.6.4 Delivered include files 23](#_Toc402435021)

[4.7 Select change package 24](#_Toc402435022)

[4.7.1 List existing change packages 24](#_Toc402435023)

[4.7.2 Select change packages 25](#_Toc402435024)

[4.8 Select setups 25](#_Toc402435025)

[4.9 RAM/ROM usage information 26](#_Toc402435026)

[4.10 Lock members 27](#_Toc402435027)

[4.10.1 Lock single members 27](#_Toc402435028)

[4.10.2 Lock paths recursively 27](#_Toc402435029)

[4.10.3 Lock paths from build folder 28](#_Toc402435030)

[4.11 Generate version label 28](#_Toc402435031)

[4.11.1 Read version number 28](#_Toc402435032)

[4.11.2 Create version label 29](#_Toc402435033)

[4.11.3 Store new version number 31](#_Toc402435034)

[4.12 Compiling the changed libraries 31](#_Toc402435035)

[4.13 Manual component tests 31](#_Toc402435036)

[4.14 Test information from user 32](#_Toc402435037)

[4.15 Creating issue lists 32](#_Toc402435038)

[4.16 Creating the component handover 33](#_Toc402435039)

[4.16.1 Version history 34](#_Toc402435040)

[4.17 Submitting the release 36](#_Toc402435041)

[4.17.1 Check-In 36](#_Toc402435042)

[4.17.2 Update revision file version 36](#_Toc402435043)

[4.17.3 Initial handover document 37](#_Toc402435044)

[4.17.4 Create checkpoint 37](#_Toc402435045)

[4.18 Thaw members 38](#_Toc402435046)

[5 Appendix 39](#_Toc402435047)

[5.1 Return values 39](#_Toc402435048)

[5.2 Handover tags 40](#_Toc402435049)

[5.3 Issue workflow 41](#_Toc402435050)

1. List of figures

Fig. 1: Handover validation 34

Fig. 2: Issue workflow 41

1. List of tables

Table 1: AutoRelease Environment 13

Table 2: AutoRelease Environment 14

Table 3: Types of sub-sandboxes and actions 20

Table 4: List of AutoRelease return values 39

Table 5: Supported component handover tags 40

# Scope of the document

This document describes how the automatic release script (AutoRelease) is integrated into generic SCons build environment.

It also contains technical details about the implementation of the AutoRelase scripts.

# General description

## Configuration management

There are 4 scripts that implement the automatic release process. These scripts are located in:

\SCT\_Sconstools\scons\_adas\_extensions

For easier identification among the many scripts in that folder the script names are prefixed with ‘autorel\_’.

**autorel\_**main.py The main entry point and control flow.

**autorel\_**mks\_util.py All MKS related functions for the auto release sequence.

**autorel\_**functions.py All other functions for the auto release sequence.

**autorel\_**word\_util.py Not used since we have switched to a text document.

Additionally there is a shared “SConscript.py” which is responsible for the integration into the complete generic SCons build environment.

\SCT\_Sconstools\scons\_common\_scripts\utils\autorel

The code handover template “Algorithm\_Component\_Handover\_Template.txt” is placed in

\SCT\_Sconstools\scons\_templates\03\_Workspace\algo\xxx\autorelease

This file is shared from the folder

/nfs/projekte1/PROJECTS/MFC400/06\_Algorithm/01\_Supporting\_Processes/06\_Miscellaneous/Algorithm\_Component\_Handover\_Template.txt

Both files “SConscript.py” and “Algorithm\_Component\_Handover\_Template.txt” will be installed to the components workspace structure by the “sconscript\_setup”-feature (see 2.2). The directory inside the component path will be:

04\_Engineering\03\_Workspace\algo\<component>\autorelease

In this folder there is just one file checked in: the configuration.

autorel\_handover.scfg

The configuration is also available as template in the generic scons project:

\SCT\_Sconstools\scons\_templates\03\_Workspace\algo\xxx\autorelease

The handover document is part of the build process and therefore it is generated here:

\04\_Engineering\04\_Build\algo\<component>

Finally the handover is installed here:

\01\_Supporting\_Processes\01\_Project\_Management\05\_Release

## Activation

To activate the script add the following line to the file ‘sconstruct\_config.scfg’.

autorel\_support = True

Furthermore the following block has to be added to the file ‘sconscript\_setup\_config.scfg’ in order to copy the shared SConscript to the right place.

# auto release

{

"name" : "autorel",

"copy" : True,

"dest\_folder" : "04\_Engineering/03\_Workspace/algo/" +

component\_name + "/autorelease",

"source" : ["utils/autorel/SConscript.py", "../scons\_templates/03\_Workspace/algo/xxx/autorelease/Algorithm\_Component\_Handover\_Template.txt"]

},

## Build targets

The complete auto release process needs an extensive amount of runtime and it is only allowed to run if the build result was tested before. Therefore added as a second root target parallel to ‘all’.

all

cipp\_algo

cipp\_algo\_ecu

...

cipp\_sim

cipp\_sim\_hil\_ecu\_out

...

cipp\_sim\_pc

...

**autorel**

## Integration as Python Builder

The auto release has to be integrated as an own builder which is realized in the “SConscript.py” file.

execfile("autorel\_main.py");

def autorelease\_builder(target, source, env):

return autorelease(File(target[0]).abspath);

bld = Builder(action = autorelease\_builder)

env = Environment(BUILDERS = {'MakeAutoRelease' : bld})

doc\_file = build\_dir + "algo/" + component\_name + "/" +

component\_name.upper() + "\_AlgorithmCodeHandover\_AutoRelease.docx"

# pass output (docfile) and input (config file) to the builder

Alias("autorel", env.MakeAutoRelease([doc\_file],

[' autorel\_handover.scfg']));

# execute the script -

# even if the input file (configuration) hasn't changed!!

AlwaysBuild(autorel\_script)

Since input and output is mandatory for SCons builder we define it as this.

Input: autorel\_handover.scfg (more files could be added in the future)

Output: <COMP>\_AlgorithmComponentHandover\_AutoRelease.docx

# Configuration

The configuration of the AutoRelase script is expected to be located in the file

\04\_Engineering\03\_Workspace\algo\cipp\autorelease\autorel\_handover.scfg

This file is divided in two parts, a static and a dynamic one. For initial integration of auto release there is a configuration template available:

\SCT\_Sconstools\scons\_templates\03\_Workspace\algo\xxx\

autorelease\autorel\_handover\_template.scfg

## Static configuration

Static configuration values are those that change never or rarely.

### Component owner

This is the name of the component owner as free text. This name will be placed in the algo component handover document.

component\_owner = "Simon Sample"

### Project configuration

The project is configured in a set of XML files in the root folder. These files define the versions of all shared subprojects. The name of the file is typically as follows, but may be adapted.

config\_list\_file = "shared\_project\_config\_files.xml"

Anyhow this name should be the same for all components and will never change.

### Project root directory

This is the place where the project configuration is expected to be (see 3.1.2). Also the release checkpoint is going to be created here. This definition should be the same for all components and will never change.

sandbox\_root = engineering\_dir + "../"

The variable “engineering\_dir” is defined in “\scons\_common\_config\common\_config.scfg”.

### Kind of delivery

This is either “Library” or “Source”. Anyhow, it is realized as free text to not be restricted to these options.

kind\_of\_delivery = "Library"

This is the default delivery kind that should match most releases. Anyhow the script prompts the user to confirm that. See chapter 0 for more information.

### Release setups

The release setups define libraries that may be delivered within one package. The AutoRelease script will prompt the user to select one, all or a custom set of these options.

component\_release\_setups = (

{

'NAME':'Name of the setup',

'DIRS':['dir\_1', 'dir\_2'],

'BUILDS':['target\_1', 'target\_2'],

},

)

The NAME is the text that will be displayed in the script. It should be unique and meaningful.

DIRS is a set of directories in “\04\_Engineering\04\_Build\algo\{component}\_sim” (e.g. ‘ti\_c674x’ or ‘ti\_arp32’). These directories are scanned for ram/rom usages, which will be placed in the component handover document.

BUILDS is a set of SCons targets that will be compiled by AutoRelease. Since the version number is changed by the script, all libraries that contain the version number have to be rebuilt.

If there are multiple setups defined, duplicate DIRS and BUILDS will be merged.

*Background information:*

There are some scenarios where not all libraries are released at the same time, e.g.

* only VH28 libraries are tested
* only CortexA8 version of the library (not C674x) was tested

Defining setups allows supporting such scenarios.

Example:

component\_release\_setups = (

{

'NAME':'C674x-ARP32',

'DIRS':['ti\_c674x', 'ti\_arp32'],

'BUILDS':['cipp\_algo\_ti\_c674x\_release', 'cipp\_algo\_ti\_arp32\_release'],

},

{

'NAME':'C66xx-ARP32',

'DIRS':['ti\_c66xx', 'ti\_arp32\_vh28e1'],

'BUILDS':['cipp\_algo\_ti\_c66xx\_release', 'cipp\_algo\_ti\_arp32\_release'],

},

)

## Dynamic configuration

Dynamic configuration values are those that may change from release to release. When changed the configuration file shall be checked in - otherwise the consistency check stage will fail. Another benefit is that the configuration (which is input for the release) is part of the checkpoint.

### Compliance report link

This is an MKS-link to any compliance report. It is realized as free text that will be passed to the component handover document.

compl\_report\_link = "n/a"

### Related releases

This is a colon separated list of MKS release IDs that are related to component release. All releases that are containing "Planned Requests" are investigated for issues that are marked as “Realized” or “Closed”. Found issues will be passed to the handover document. Non-release IDs will be ignored and a warning message is printed by the script.

release\_id\_list = [id\_1, id\_2, …]

Example:

release\_id\_list = [174366, 186012]

### Additional checkpoints

This is a list of label-link-pairs that will be passed to the handover document. Any version label of a shared project inside the project may be added here.

additional\_checkpoints = [

{

"LABEL":"VERSION\_LABEL",

"LINK":"mks\_web\_link"

},

]

LABEL is a version label of a shared project that is configured in the sanbox.

LINK is an MKS link to the project that is shared. It’s realized as free text to be flexible.

Example:

additional\_checkpoints = [

{

"LABEL":"AL\_SR\_WRP\_03.00.14",

"LINK":"http://mks-psad:7001/si/viewproject?projectName=/nfs/projekte1/REPOSITORY/Base%5fDevelopment/05%5fAlgorithm/SR%5fSignRecognition/04%5fEngineering/01%5fSource%5fCode/algo/00%5fCustom/sr%5fwrp/project.pj"

},

{

"LABEL":"SW\_COMMON\_06.00.00\_INT-4",

"LINK":"http://mks-psad:7001/si/viewproject?projectName=/nfs/projekte1/REPOSITORY/Base\_Development/05\_Algorithm/CIPP\_CommonImagePreProcessing/04\_Engineering/01\_Source\_Code/common/rte/project.pj"

},

]

# AutoRelease sequence

Each stage of the AutoRelease sequence is explained below. The steps are executed in the same order as listed here.

## Initial environment

At first there is an initial environment (key-value-pairs) created which will be passed to the component handover document. In python nomenclature it’s a dictionary ({}). Some of the fields are filled in directly some are filled during the auto release sequence.

|  |  |
| --- | --- |
| Field | Value/Description |
| 'TYPE' | 'AUTOREL' – can be used to check the type of the dictionary. |
| 'PROJECT\_NAME' | String. MKS name of the sandbox root project. |
| 'COMPONENT\_OWNER' | Taken from configuration file. |
| 'CP\_LABEL\_LIST' | Taken from configuration file. |
| 'CP\_LABEL\_PATH' | Taken from configuration file. |
| 'KIND\_OF\_DELIVERY' | Taken from configuration file. |
| 'COMPLIANCE\_REPORT' | Taken from configuration file. |
| 'COMPONENT\_ROOT' | String. Complete MKS database path of the sandbox root project. |
| 'DEVELOP\_PATH' | Development path label (string). |
| 'CP\_LABEL\_LIST' | Taken from configuration file. |
| 'CP\_LABEL\_PATH' | Taken from configuration file. |
| 'CHECKPOINT\_LABEL' | AutoRelease checkpoint (string). |
| 'RESOURCE\_USAGE' | List of RAM/ROM usages according to selected release setups (see 3.1.5). |
| 'COMPONENT\_CONFIG' | List of shared projects and versions from XML files in the root folder. |
| 'TEST\_STATE\_SIL' | SiL test done? True if ‘yes’, False if ‘no’. |
| 'TEST\_STATE\_EVM' | EVM test done? True if ‘yes’, False if ‘no’. |
| 'TEST\_STATE\_ECU\_LAB' | ECU test done (in lab)? True if ‘yes’, False if ‘no’. |
| 'TEST\_STATE\_ECU\_CAR' | ECU test done (in car)? True if ‘yes’, False if ‘no’. |
| 'ISSUE\_LIST\_PR' | List of problems fixed in that release. |
| 'ISSUE\_LIST\_FR' | List of features realized in that release. |
| 'ISSUE\_LIST\_CR' | List of changes realized in that release. |

Table : AutoRelease Environment

In most cases the environment keys exist as tag in the handover document. Anyhow, the following tags exist only in the handover template and will be generated from the environment. See also chapter 5.2 for complete list of handover tags.

|  |  |
| --- | --- |
| Field | Value/Description |
| 'ADDITIONAL\_CP\_LIST' | Will be generated from 'CP\_LABEL\_LIST' and 'CP\_LABEL\_PATH'. |
| 'ECU\_HISTORY' | Will be replaced with history taken from last code handover document. |
| 'SIM\_HISTORY' | Will be replaced with history taken from last code handover document. |

Table : AutoRelease Environment

## Project Information

At first the project information is taken from MKS.

### Project URL and Development path

This function is realized in “autorel\_mks\_util.py”, function “MKS\_GetProjectInfo”. See API documentation in the python file for further interface information.

MKS command:

si projectinfo --nodevpaths --noassociatedIssues

--noshowCheckpointDescription --noattributes --noacl

From the output the following fields are passed.

"Project Name: …"

"Server: …"

"Development Path: …"

The values from both project name and server are used to create the project path.

Path = "http://" + Server + "/si/viewproject?projectName=" + Name

The development path is taken directly.

### Project name

This function is realized in “autorel\_mks\_util.py”, function “MKS\_GetProjectName”. See API documentation in the python file for further interface information.

MKS command:

si projectinfo --nodevpaths --noassociatedIssues

--noshowCheckpointDescription --noattributes –noacl

The command is the same as for project URL (see 4.2.1) but it’s parsed differently.

From the field "Configuration Path:" the string after the last occurrence of a slash (/) is taken.

## Type of delivery

The script prompts the user to confirm the type of delivery that was configured (see 3.1.4). Actually there are only 2 types existing: “Source” and “Library”.

------------------------------------------------

---------- TYPE OF DELIVERY --------------------

------------------------------------------------

Is the delivery type 'Source' correct? [y/n] n

Delivery type 'Library' is used.

If the configured (default) type was not confirmed the other one is choosen automatically (see above). If the type of delivery is not configured or not valid, a menu is presented in order to select the correct one.

------------------------------------------------

---------- TYPE OF DELIVERY --------------------

------------------------------------------------

[1] Library

[2] Source

Select one of the options above: 2

## Freeze / Thaw members

The first action in MKS is freezing the complete sandbox to make sure that no one changes member revisions during the auto release process. Therefore the last action of the auto release process is to thaw all members. This is also documented here because it is functionally related to each other.

These functions are realized in “autorel\_mks\_util.py”:

* MKS\_FreezeMembers()
* MKS\_ThawMembers()
* MKS\_FreezeOrThawSandboxRecurse()

See API documentation in the python file for further interface information.

The function thaw/freeze is additionally needed in the following situations:

* Checking locked members in (member revision is updated). See also chapter 4.17.1.
* Member revision of delivered version file is updated. See also chapter 4.17.2.

### Low-level functions

MKS command - MKS\_FreezeMembers():

si freeze --batch --norecurse –quiet [specific\_file]

MKS command - MKS\_ThawMembers():

si thaw --batch --norecurse –quiet [specific\_file]

### Hi-level function

The hi-level function uses the low-level functions to freeze/thaw all members in the current sandbox and iterates recursive into all non-shared sub-sandboxes. The MKS types of those are:

* subsandbox
* variant-subsandbox

MKS command - MKS\_FreezeOrThawSandboxRecurse():

si viewsandbox --batch --fields=name,type --forceConfirm=yes --quiet   
 --norecurse

If a sub-sandbox of the types above is found the hi-level functions calls itself with that sandbox as root directory.

### Output

------------------------------------------------------

---------- FREEZING SANDBOX MEMBERS ------------------

------------------------------------------------------

Succeeded./Failed.

------------------------------------------------------

---------- THAWING SANDBOX MEMBERS ------------------

------------------------------------------------------

Succeeded./Failed.

## Parsing project configuration

The project configuration is enclosed in a set of XML files located in the root folder of the sandbox. Parsing is realized by the function ‘XML\_ParseProjectConfig’ in file ‘autorel\_functions.py’. See API documentation in the python file for further interface information.

As XML parser “xml.dom.minidom” is used.

At first the list file, which is defined in the AutoRelease configuration (see 3.1.2), will be parsed.

Example:

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

<ConfigSharedProjectsFiles>

<FileCfg File="shared\_project\_config\_master.xml" Type="master" />

<FileCfg File="shared\_project\_config\_sim\_bundle.xml"/>

<FileCfg File="shared\_project\_config\_tools.xml"/>

<FileCfg File="shared\_project\_config\_test\_tools.xml"/>

<FileCfg File="shared\_project\_config\_cct.xml"/>

<FileCfg File="shared\_project\_config\_rte.xml"/>

</ConfigSharedProjectsFiles>

In the 2nd step it parses all files configured in a ‘FileCfg’ element. Those files are located in the same directory.

Example (shared\_project\_config\_master.xml):

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

<?xml-stylesheet type="text/xsl" href="Config.xslt"?>

<ConfigSharedProjects>

<BaseConfig>

<SharedProject CompName="CANT\_Cantata">

<Config Revision="1.6" Type="build"/>

</SharedProject>

<SharedProject CompName="CML\_CommonMathLibrary">

<Config Label="AL\_CML\_01.06.00\_INT-7" Type="build"/>

</SharedProject>

...

</BaseConfig>

</ConfigSharedProjects>

For each file the script parses for ‘Config’ elements. Only those with a ‘Type’ attribute equal to ‘build’ are taken, others are ignored.

The name and the label are stored. The name is taken from ‘CompName’ attribute of parent element ‘SharedProject’. If there is no ‘Label’ attribute, the ‘Revision’ attribute exists. In this case the label is generated: “Revision “ + <rev\_number>.

A list of names and labels is generated and passed to the component handover document.

The list is extended by a 3rd item of the type Boolean. This item indicates that the specific shared project exists in the sandbox. It is set ‘True’ by default and will be released later if the shared project was not found.

This is for filtering out the unused shared project to avoid listing them in the handover document. This Boolean item will also be parsed to the code handover, but will not be evaluated.

[["name\_1", "label\_1", exists\_1],

["name\_2", "label\_2", exists\_2],

[…],

["name\_n", "label\_n", exists\_n]]

The filtering will be done by the following python function:

filter(lambda a: a[2], [shared\_project\_list])

It checks whether the 3rd item of each list item is ‘True’.

Additionally there is support for customized configurations version 1 (V1) and version 2 (V2). There is **no support** for customized configurations version 3 (V3).

Customized configuration V1:

<CustomConfig>

<SharedProject SharedPath="...">

<Config Label="..." Revision="..." Type="build"/>

</SharedProject>

</CustomConfig>

Customized configuration V2:

<CustomConfigV2>

<SharedProject CompName="...">

<Config Label="..." Type="build"/>

</SharedProject>

</CustomConfigV2>

Customized configuration V1 overwrite normal configuration, V2 overwrites normal and V1 configuration. If a shared project configuration is overwritten this is logged in the console as follows.

---------------------------------------------------------------------------------

---------- READING SANDBOX CONFIGURATION ----------------------------------------

---------------------------------------------------------------------------------

Warning: custom configuration V3 is not supported   
 (shared\_project\_config\_master.xml).

Custom V2 - "AL\_SMFC4B0\_06.00.00\_RTE\_3" replaced by "MFC\_THIS\_IS\_JUST\_FOR\_TESTS".

Custom V2 - "AL\_CML\_01.06.00\_INT-7" replaced by "CML\_THIS\_IS\_JUST\_FOR\_TESTS".

Custom V1 - "AL\_STP\_MPY\_V01.00.01\_20100503" replaced by "MOVPY\_ JUST\_FOR\_TESTS".

Custom V1 - "AL\_STP V01.00.00\_20120404" replaced by "GRAPHVIZ\_ JUST\_FOR\_TESTS".

Succeeded.

## Consistency check

### Member revisions

In this step the consistency between member and working revision is checked. This function is realized in “autorel\_mks\_util.py”, function “MKS\_GetChangedFiles”. See API documentation in the python file for further interface information.

The members in the sandbox root directory are checked to ensure that project configuration (xml) is up to date (see **Fehler! Verweisquelle konnte nicht gefunden werden.**).

The following folders are checked recursively:

\04\_Engineering\01\_Source\_Code

\04\_Engineering\03\_Workspace\algo\{component}\*

\04\_Engineering\05\_Deliverables\include

\04\_Engineering\05\_Deliverables\cfg

A member is considered as “not consistent” if:

* the working revision is not the member revision
* the working file was modified
* the member is locked by the user that executes this script

MKS command:

si print --recurse --filter=changed:working,changed:sync,locked:<user>

--headerFormat="{membername}\t{memberrev}\t{workingrev}

\t{workingfileexists}\t{alllockers}\t{workingrevdelta}\n"

For detecting inconsistencies in the root folder the option ‘--norecurse’ is used instead of ‘--recurse’. This controlled by an optional function parameter.

The tag <user> will be replaced by the windows user which is normally the same as the MKS user id.

The type of inconsistency will be categorized in the following order:

1. NO WORKFILE (local file doesn’t exist) if field {workingfileexists} is 0
2. OUT OF SYNC if field {memberrev} is not equal {workingrev}
3. LOCKED if field {alllockers} is not empty
4. MODIFIED if field {workingrevdelta} is not empty

Both information member and type of inconsistency is logged.

If there are inconsistencies the AutoRelease is terminated with an error (see 5).

Example error log:

------------------------------------------------------------------------

---------- CONSISTENCY CHECK -------------------------------------------

------------------------------------------------------------------------

Checking members..............failed

> ...\03\_Workspace\...\autorelease\autorel\_handover.scfg (MODIFIED)

The auto release script was terminated due to an error.

Error description: 1 inconsistent files found.

### Check shared projects

In this step the validity of project configuration is checked. This function is realized in “autorel\_mks\_util.py”, function “MKS\_GetSharedProjListRecurse”. See API documentation in the python file for further interface information.

The script scans for shared sub-projects and returns them as 3 lists (valid shares, invalid shares and not shared). Invalid sandboxes are those who are shared and not configured as “build”.

MKS command:

si viewsandbox --batch --fields=name,type --forceConfirm=yes --quiet   
 --norecurse

This command returns a list of all sub-projects/members and their types, separated by a whitespace character. In case of build sandboxes there is also the checkpoint version returned.

The sub-sandbox types are handled as follows:

|  |  |
| --- | --- |
| Type | Action |
| "subsandbox"  "variant-subsandbox"  "build-subsandbox" | Recurse into this sandbox. Add those sandboxes the list of non-shared folder in order to label them for the checkpoint (see also 0). |
| "shared-variant-subsandbox"  "shared-subsandbox" | Invalid shared sandbox. Find out the path where it is shared from, add it to invalid-list and return it. |
| "shared-build-subsandbox" | Get version and also label (if available) and project name for comparison. |
| All others (i.e. members) | Ignored. |

Table : Types of sub-sandboxes and actions

If an invalid shared sandbox is detected the “Shared From:” field is evaluated in the function MKS\_GetProjectDatabasePath(). Finally the local sandbox directory and the MKS database path are returned as a list.

The list is evaluated in “autorel\_main.py”, function “ExecConsistencyCheck()”. It looks for internally shared projects by comparing the MKS database paths of the sandbox root and the “invalid list”. If the MKS database path of the invalid project starts with the sandbox root path, the project is considered as internal and removed from the list.

Those projects left in the invalid list are printed to the console and the script is terminated.

Example error log:

-------------------------------------------------------------------------

---------- CONSISTENCY CHECK --------------------------------------------

-------------------------------------------------------------------------

Checking members..............ok

Looking for shared projects...failed

Found shared projects that are not of type "build":

> D:\sandboxes\CIPP\_AutoReleaseTest\04\_Engineering\03\_Workspace\sd\_source

Shared from: /nfs/projekte1/REPOSIT...04\_Engineering/01\_Source\_Code/project.pj

The auto release script was terminated due to an error.

Error description: 1 non-build shared-projects found.

### Shared project version

In this step the consistency between the project configuration (see 0) and the shared project version is checked. This function is realized in “autorel\_functions.py”, function “CheckProjectConfig“. See API documentation in the python file for further interface information.

Only shared sub-sanboxes of type ‘build’ are considered (list of valid shared projects, see 4.6.2). A shared project is considered as “not consistent” if:

* the version label is not as configured and
* the version number is not as configured

To get the project version label the history is scanned for version number taken from the list above.

MKS command:

si viewprojecthistory --batch --fields=labels,revision --rfilter=labeled

This command returns a list of all revisions that are labeled. The label that belongs to the wanted revision is taken. If no revision is found this shared project has to be configured by the version number. If this is not the case it will be considered as “inconsistent”.

To compare the configuration with the sub-project versions the name of the subject is determined. For this purpose the same MKS command is used as for project name (see 4.2.2), but the path "Shared From" (output from si) is evaluated.

The method how a project name is built depends on the project itself. There are 3 different cases:

1. Use the subproject of a specific path.
2. Use the subproject of a specific path and add a prefix.
3. Use a pre-defined project name and ignore the path.

In fact there are more than 20 different paths that may occur in the shared project path. For to generate readable code there is a list ‘SEARCH\_ORDER’ defined in the top area of “autorel\_mks\_util.py”. Each list item is a dictionary of the keys ‘COMP’, ‘NAME’ and ‘PRFX’.

Abstract (demonstrating the 3 cases):

SEARCH\_ORDER = [

{'COMP':'05\_Algorithm/', 'NAME':None, 'PRFX':False},

...

{'COMP':'/projekte1/REPOSITORY/Base\_Development/Base\_CGEB/',   
 'NAME':'Base\_CGEB', 'PRFX':False},

...

{'COMP':'/SW\_MFC4xx/SW\_SMFC400\_PR/04\_Engineering/02\_Development\_Tools/',   
 'NAME':'SW\_', 'PRFX':True},

...

]

If '05\_Algorithm/' is contained in the path, the name of the sub project is taken (‘NAME’ is ‘None’ and ‘PRFX’ is ‘False’).

If '/projekte1/REPOSITORY/Base\_Development/Base\_CGEB/' is contained in the path, the shared project name is 'Base\_CGEB' (‘NAME’ is defined and ‘PRFX’ is ‘False’).

If '/SW\_MFC4xx/SW\_SMFC400\_PR/04\_Engineering/02\_Development\_Tools/' is contained in the path, the name of the sub project is taken and the prefix is 'SW\_' is added (‘NAME’ is defined and ‘PRFX’ is ‘True’, so ‘NAME’ is used as prefix).

Please note the order of the list items is important and may not be changed. The priority for the first item is low, for the last high.

*Example:*

SEARCH\_ORDER[0] = {'COMP':'05\_Algorithm/', 'NAME':None, 'PRFX':False}

SEARCH\_ORDER[5] = {'COMP':'/projekte1/REPOSITORY/Base\_Development/05\_Algorithm/  
 ETK\_EngineeringToolKit/04\_Engineering/',   
 'NAME':None, 'PRFX':False}

Both paths contain '05\_Algorithm/'. But if the path is '/projekte1/REPOSITORY/Base\_Development/05\_Algorithm/ETK\_EngineeringToolKit/04\_Engineering/' the shared project name will **not** be ‘ETK\_EngineeringToolKit’ but the sub project of ‘04\_Engineering’.

With this information both project configuration and currently configured shared project versions can be compared.

If there is shared project configured that is not used in the sandbox it is marked as ‘False’ in this step. This is for filtering out and not passing to the handover document (see also page 17 for more information).

If there are inconsistencies the AutoRelease is terminated with an error (see 5).

Example error log:

------------------------------------------------------------------------

---------- CONSISTENCY CHECK -------------------------------------------

------------------------------------------------------------------------

Checking members..............ok

Looking for shared projects...ok

Checking shared projects......failed

Compontent "GS\_GlobalScheduler":

Found: Label="AL\_GS\_03.03.00\_INT-32", Rev.=1.42

Exp : "AL\_GS\_03.03.00\_INT-11"

Compontent "CANT\_Cantata":

Found: Label="AL\_EMO\_01.23.01\_INT-1", Rev.=1.2

Exp : "Revision 1.6"

The auto release script was terminated due to an error.

Error description: 2 inconsistent shared projects found.

### Delivered include files

In this step the member revisions of the shared include files in “05\_Deliverables\include” and their pendants in “01\_Source\_Code” are checked. This function is realized in “autorel\_main.py”, using utility functions from “autorel\_mks\_util.py”. See API documentation in the python file for further interface information.

In a first step all shared archive files are parsed from “05\_Deliverables\include”.

MKS command - MKS\_GetSharedMembersRecurse():

si viewsandbox --batch --recurse --quiet   
 --fields=name,memberarchive,type,memberrev   
 --filter=archiveshared

The filter hides the non-shared files from the output and the fields just show the required information. This generates a list of files, each list item is dictionary:

{'name':<sandbox\_location>,

'archive':<mks\_archive\_path>,

'rev':<member\_revision\_string>}

The MKS archives are checked to start identically as the path of “01\_Source\_Code”. All others are removed from the list.

The revisions of those files left over are compared to revisions of the corresponding files in “01\_Source\_Code”. The files where the revision is equal are also removed from the list. A non-empty list indicates inconsistencies.

Sometimes the file location in “01\_Source\_Code” doesn’t match the member archive path. In this case the directory “01\_Source\_Code” is scanned to find the real location. If the file is not found it is removed from the list.

Example error log:

------------------------------------------------------------------------------

---------- CONSISTENCY CHECK -------------------------------------------------

------------------------------------------------------------------------------

Checking members..............ok

Looking for shared projects...ok

Checking shared projects......ok

Checking includes.............failed

> File: 05\_Deliverables\include\algo\cipp\ti\_arp32\CIPP\_ARP\_Interface.h

Shared from: 01\_Source\_Code\algo\cipp\ti\_eve\_arp32\inc\CIPP\_ARP\_Interface.h

Rev. in 05\_Deliverables: 1.1

Rev. in 01\_Source\_Code: 1.2

> File: 05\_Deliverables\include\algo\cipp\ti\_c674x\cipp\_ver.h

Shared from: 01\_Source\_Code\algo\00\_Custom\cipp\_wrp\cipp\_ver.h

Rev. in 05\_Deliverables: 1.31

Rev. in 01\_Source\_Code: 1.28

The auto release script was terminated due to an error.

Error description: 2 inconsistent include delivery files found.

## Select change package

A change package (CP) is required for all MKS actions like lock, check-in etc. Therefore the script makes sure that a change package is existing that can be used for the release process. This change package has to be created by the user before the AutoRelease is started.

The change package selection is divided in two steps, getting a list of CPs .

### List existing change packages

This function is realized in “autorel\_mks\_util.py”, function “MKS\_ListChangePackages”. See API documentation in the python file for further interface information.

MKS command:

si viewcps --fields=id,summary

This command returns just the change package ID and the summary. Both is parsed and returned as a 2-dimensional string array.

### Select change packages

This function is the first user interaction and is realized in “autorel\_functions.py”, function “SelectChangePackage”. See API documentation in the python file for further interface information.

Based on the CP list created in the previous step the following menu is printed.

-----------------------------------------------------------------

---------- CHANGE PACKAGE SELECTION -----------------------------

-----------------------------------------------------------------

The selected change package will be used for locking members.

Existing open change packages:

------------------------------

[ 1] 178143:5 - ARM EVE Commnuication

[ 2] 178143:6 - Demo change package

[ 3] 178143:7 - One more demo CP

----

[ s] Scan again

Select one of the options above:

The user may now enter the number of the change package that should be used. If a valid CP does not yet exist the user may create a new one in MKS and select ‘s’ to scan for new CPs and update the list.

The selected change package is returned as a string to be used whenever needed.

## Select setups

To select the setups a list that was defined in the configuration file (see also 3.1.2) is printed.

---------------------------------------------------------------------

---------- COMPONENT RELEASE SETUPS ---------------------------------

---------------------------------------------------------------------

Please select the setups you would like to release.

Select one setup, all setups or custom selection followed by <ENTER>.

[1] C674x-ARP32

[2] C66xx-ARP32

[3] CortexA8-ARP32

---

[a] ALL SETUPS

[c] Custom

Select one of the options above:

The user may select one, all or custom. If ‘custom’ was selected each single setup is printed to be selected by ‘y’ or skipped by ‘n’.

If there is only a single setup defined the user is not prompted to select one.

---------------------------------------------------------------------

---------- COMPONENT RELEASE SETUPS ---------------------------------

---------------------------------------------------------------------

Only one setup configured: "C66xx-ARP32" (no selection required).

If there are less than 3 setups defined the ‘custom’ option is useless and therefore not available.

---------------------------------------------------------------------

---------- COMPONENT RELEASE SETUPS ---------------------------------

---------------------------------------------------------------------

Please select the setups you would like to release.

Select one or all setups followed by <ENTER>.

[1] C674x-ARP32

[2] C66xx-ARP32

---

[a] ALL SETUPS

Select one of the options above:

The selection is required by the next steps (compilation and ram/rom usage).

## RAM/ROM usage information

The information is read from the libsize files at the following places.

\04\_Engineering\04\_Build\algo\{comp\_name}\_sim\<dir\_list>\

release\\*libsize.txt

Dir list is taken from the ‘DIRS’-key from configuration (see also 3.1.2).

The content of the file is returned as an array of dictionaries as returned by the function ‘readLibSizeFile’ from the script ‘ram\_rom\_algo\_libsize.py’. Refer to this script for further information on the file format.

The information contained is library, ram/rom/ignored/total usages, which is passed to the component handover script.

## Lock members

There are two types of members treated differently.

### Lock single members

The following members are locked:

* Version file
* Component handover document

This function is realized in “autorel\_mks\_util.py”, function “MKS\_LockMembers”. See API documentation in the python file for further interface information.

MKS command:

si lock --changePackageId=<cp> "<full\_path\_and\_filename>"

This command locks the specified file using the previously selected change package ID.

### Lock paths recursively

The following paths are locked, if delivery type is “Library” (see chapter 0), otherwise it is neither locked nor checked in.

\04\_Engineering\05\_Deliverables\sdl\\*

\04\_Engineering\05\_Deliverables\dll

\04\_Engineering\05\_Deliverables\lib\\*

The 1st directory (sdl) is scanned for sub-paths starting with either “algo” or the architecture selected from AutoRelease setups. Example for CIPP:

\04\_Engineering\05\_Deliverables\sdl\algo\cipp

\04\_Engineering\05\_Deliverables\sdl\ti\_arp32\algo\cipp

\04\_Engineering\05\_Deliverables\sdl\ti\_c66xx\algo\cipp

\04\_Engineering\05\_Deliverables\sdl\ti\_c674x\algo\cipp

If the component supports VH28 and ARP32 there might be an architecture “arp32\_vh28e1” existing. In this case the suffix “\_vh28e1” is removed.

The 3rd directory (lib) is scanned for paths containing the component name. Example for CIPP:

\04\_Engineering\05\_Deliverables\lib\pc\algo\cipp

\04\_Engineering\05\_Deliverables\lib\ti\_arp32\algo\cipp

\04\_Engineering\05\_Deliverables\lib\ti\_c66xx\algo\cipp

\04\_Engineering\05\_Deliverables\lib\ti\_c674x\algo\cipp

This function is realized in “autorel\_mks\_util.py”, function “MKS\_LockPathsRecurse”. See API documentation in the python file for further interface information.

MKS command:

si lock --recurse --changePackageId=<cp> --no

### Lock paths from build folder

The AutoRelease scripts checks the path listed in chapter 4.10.2 for files that are shared from the build folder. If such files were found, these paths are removed from the normal handling. Within these paths a mixed mode is not possible.

This means that libraries are checked in directly from build path using the generated version label. This label is used to update the shared files in the deliverables folder. Finally the deliverables folder is re-synchronized.

## Generate version label

The generation of the version label is separated into several steps.

### Read version number

The current version number, which is part of the version label, is taken from the version file.

The location of the version file is variable but shall meet the following rule:

\04\_Engineering\01\_Source\_Code\algo\...\{component\_name}\_ver.h

If the file was found the content is parsed expecting the following lines.

#define <COMP>ALL\_SW\_MAIN\_VER\_NO 0x??U

#define <COMP>ALL\_SW\_SUB\_VER\_NO 0x??U

#define <COMP>ALL\_SW\_BUG\_FIX\_LEV 0x??U

#define <COMP>ALL\_SW\_BUG\_FIX\_LEV2 0x??U

<COMP> has to be replaced by the upper case component name.

If not all lines were found in the file it is considered as invalid and the script terminates with an error code.

If the file was parsed correctly the 4-digit version number is returned.

Finally there is a range check implemented. All segments of the number have fit in an 8-bit unsigned integer value (0-255). The check also fails if all segments are 255, which means incrementing the number is impossible. In both cases the error AUTOREL\_ERR\_VERFILE is returned (see 5.1).

### Create version label

With this information the version label will be generated. This is controlled by the function “GenerateVersionLabel” in the file “autorel\_functions.py”. See API documentation in the python file for further interface information.

At first the user will be prompted to select which part of the version number has to be increased. The current version number is displayed and the possible options.

--------------------------------------------------------------

---------- INCREMENT VERSION ---------------------------------

--------------------------------------------------------------

Select the segment of the version number (4.1.0\_INT-0) to be raised:

[1] Major (=> 5.0.0\_INT-0)

[2] Minor (=> 4.2.0\_INT-0)

[3] Bugfix (=> 4.1.1\_INT-0)

[4] Integration (=> 4.1.0\_INT-1)

----

[0] None (keep current version number)

Select one of the options above:

If a version number violates the 8-bit unsigned integer range, the specific option is disabled. Example:

--------------------------------------------------------------

---------- INCREMENT VERSION ---------------------------------

--------------------------------------------------------------

Select the segment of the version number (3.255.5) to be raised:

[1] Major (=> 4.0.0\_INT-0)

[2] Minor DISABLED (RANGE VALIDATION)

[3] Bugfix (=> 3.255.6\_INT-0)

[4] Integration (=> 3.255.5\_INT-1)

----

[0] None (keep current version number)

Select one of the options above:

Once an option is selected the fix part of the label is created following these conventions (Python format string):

"AL\_%(comp)s\_%(main)02i.%(sub)02i.%(fix1)02i\_INT-%(int)i"

* comp = upper case component name (e.g. CIPP)
* main = 1st digit of version number
* sub = 2nd digit of version number
* fix1 = 3rd digit of version number
* int = 4th digit of version number

Then the user is prompted to enter a custom description.

--------------------------------------------------------------------------

---------- INCREMENT VERSION ---------------------------------------------

--------------------------------------------------------------------------

Select the segment of the version number (4.1.0\_INT-0) to be raised:

[1] Major (=> 5.0.0\_INT-0)

[2] Minor (=> 4.2.0\_INT-0)

[3] Bugfix (=> 4.1.1\_INT-0)

[4] Integration (=> 4.1.0\_INT-1)

----

[0] None (keep current version number)

Select one of the options above: 4

Enter description part of version label: AL\_CIPP\_04.01.00\_INT-1\_TestCase

The script takes the custom description and performs the following actions:

* Removing leading and trailing whitespace characters
* Converting whitespace characters left to underscore (\_)

Finally the user is prompted to confirm the label.

Enter description part of version label: AL\_CIPP\_04.01.00\_INT-1\_JustDoIt

Is the version label 'AL\_CIPP\_04.01.00\_INT-1\_JustDoIt' correct? [y/n]

If the user answers with ‘y’ the version label is returned, otherwise the process of generating the label restarts again.

### Store new version number

Finally the version number is updated in the version file. Additionally to the information read from the version file (see chapter 4.11.1), the custom description string is written to the file.

#define CIPPALL\_SW\_MAIN\_VER\_NO 0x04U

#define CIPPALL\_SW\_SUB\_VER\_NO 0x01U

#define CIPPALL\_SW\_BUG\_FIX\_LEV 0x00U

#define CIPPALL\_SW\_BUG\_FIX\_LEV2 0x01U

#define CIPPALL\_SW\_CUSTOM\_STR "\_JustDoIt"

Since the define “<COMPONENT>ALL\_SW\_CUSTOM\_STR” was introduced by the AutoRelease script it might not exist in the version header file. In this case it is inserted after “<COMPONENT>ALL\_SW\_BUG\_FIX\_LEV2”. Otherwise it is just repleaced.

## Compiling the changed libraries

Since the version number has been modified, some libraries have to be recompiled. Therefore the configuration file (see 3.1.2) defines some build targets to be re-built.

The script executes a new SCons build in the same where the AutoRelease was executed. The return value of the SCons script is evaluated. If ok, the script resumes, otherwise it terminates with an error.

An error is not expected at this stage because the compile process must have been run before to test the delivery.

## Manual component tests

After compilation the script is interrupted in order to execute some tests. You user may decide to skip that by just pressing ‘y’.

---------------------------------------------------------------------------------

---------- MANUAL TESTS ---------------------------------------------------------

---------------------------------------------------------------------------------

The script is interrupted for manual execution of tests. If the tests were

successful, return to this screen and confirm with 'y'. Otherwise the script will

abort!

Were the tests successfully? [y/n] y

## Test information from user

The script asks the user which tests have been performed. Each question may be answered with ‘y’ or ‘n’.

------------------------------------------------------------------

------- COMPONENT INTEGRATION TESTS ------------------------------

------------------------------------------------------------------

Please answer the following questions.

Press 'y' or 'n' followed by <ENTER>.

Tested on SIL? [y/n] y

Tested on Evalboard? [y/n] y

Tested on ECU/Lab? [y/n] n

Tested on ECU/Car? [y/n] n

The result is converted to a Boolean array (y=True, n=False) and passed to the component handover script.

## Creating issue lists

This function is realized in “autorel\_mks\_util.py”, function “MKS\_GetReleaseIssues”. See API documentation in the python file for further interface information.

Parsing issue lists is performed in two steps. In a first step it looks for planned requests related to releases.

MKS command:

im issues --fields="ID,Type,Planned Requests" <release\_ids>

The <release\_ids> is a space separated list of the configured release IDs (see 3.2.2).

Output example:

im issues --fields="ID,Type,Planned Requests" 174366 186012 53717

174366 Release 180980, 176044, 176052, 176053,..., 176076

186012 Release 186318, 186323, 186319, 186321, 186322

53717 Release

The script checks if the ‘type’ field is ‘Release’ and looks for a non-empty ‘Planned Requests’ field. All IDs in the ‘Planned Requests’ field will be collected and passed to the next step (duplicates are removed).

The 2nd step is generating 3 lists of issues (problem reports, feature requests and change request) that were accepted at least.

MKS command:

im issues --fields="ID,Type,Issue Type,State,Summary" <issues>

At first the output is checked for the ‘State’ field to be ‘Accepted’, ‘Realized’ or ‘Closed’ (see also chapter 5.3).

Issue Type = "Problem Report" or "Feature Request" or "Change Request"

Each item of the result issue lists is a Python dictionary with the following keys.

{

'issue\_id':int,

'issue\_state':string,

'issue\_sum':string

}

The 3 lists are passed to the component handover document.

## Creating the component handover

All the information collected above is now passed to the function ‘CreateCodeHandoverDoc’ in the script ‘autorel\_functions.py’. The information is passed using the environment described in chapter 4.1.

It replaces the tags (see 5.2) in the code handover template

Algorithm\_Component\_Handover\_Template.txt

by the information gathered by the script and stores it under

\04\_Engineering\04\_Build\algo\{comp\_name}

Before the text document is opened the user is informed about the consequences.

---------------------------------------------------------------------------------

---------- EDIT HANDOVER DOCUMENT -----------------------------------------------

---------------------------------------------------------------------------------

Please edit the code handover document. Since the document will not be changed

later, please make sure that all required parts - marked as '<...>' - are filled.

Finally save the document and return to this console window.

- PRESS ANY KEY TO OPEN THE DOCUMENT -

When the user has pressed any key, the text document opens in operating systems standard editor. In this step some manual information like stack usage, runtime, descriptions and others has to be added. The user has to confirm if the script was edited and saved.

Have you finished editing the document? [press any key]

In the next step the handover document is validated. This means all occurrences of ‘<…>’ have to be edited and no TAB characters may be included. The document is reopened as long as one of these checks fails (see Fig. 1: Handover validation).

This validation check is realized in the function ‘ValidateCodeHandover’ in the script ‘autorel\_functions.py’.

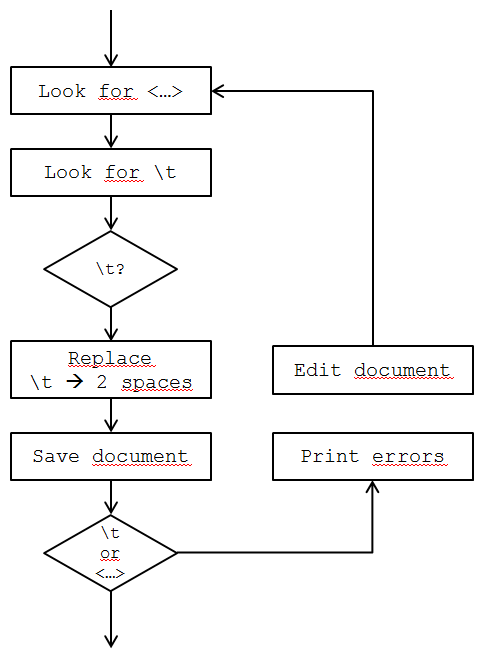


Fig. : Handover validation

After all modifications have been done and the text document is valid, it is installed at the final location.

\01\_Supporting\_Processes\01\_Project\_Management\05\_Release\

Algorithm\_Component\_Handover\_Template.txt

### Version history

This version history is realized in the function ‘GetHistoryFromHandover’ in the script ‘autorel\_functions.py’.

The function reads the last code handover file from the installation path (see above) and looks for the strings “SIM Code”, “ECU Code” and “==========”. The areas between these strings is considered as SIM history and ECU history.

Example:

========================================================================

ABSTRACT LIST OF CHANGES

========================================================================

SIM Code

--------

AL\_CIPP\_04.01.01\_INT-1:

- some comment

AL\_CIPP\_04.01.01\_INT-0:

- some comment

ECU Code

--------

AL\_CIPP\_04.01.01\_INT-1:

- some comment

AL\_CIPP\_04.01.01\_INT-0:

- some comment

========================================================================

LIST OF INTEGRATED MKS ISSUES

========================================================================

The first 2 lines in each block are skipped and all leading and trailing white spaces will be removed. So the example above will lead to the following history.

AL\_CIPP\_04.01.01\_INT-1:

- some comment

AL\_CIPP\_04.01.01\_INT-0:

- some comment

In the generated code handover the tags {SIM\_HISTORY} and {ECU\_HISTORY} will be replaced by the content. It looks like this:

AL\_CIPP\_04.01.01\_INT-2:

- <...>

AL\_CIPP\_04.01.01\_INT-1:

- some comment

AL\_CIPP\_04.01.01\_INT-0:

- some comment

If the code handover is generated the first time there is no version history available and the tags {SIM\_HISTORY} and {ECU\_HISTORY} will be removed from the code handover. This is also the case if parsing history blocks has failed.

## Submitting the release

Before everything is committed the user is prompted to confirm this. Then the final steps checking in, creating checkpoint and labeling project files are performed.

### Check-In

This function is realized in “autorel\_mks\_util.py”, function “MKS\_CheckIn”. See API documentation in the python file for further interface information.

Check in is performed for each locked path recursively and for each single member.

MKS command for members:

si ci --changePackageId=<cp> --nocloseCP --label=<label>

--description="New version released" --nolock --unlock

--update <file>

The <cp> is the previously selected change package ID and <file> the single member (full path) to be checked in.

The generated <label> is used for check-in.

MKS command for paths:

si ci --changePackageId=<cp> --nocloseCP --label=<label>

--description="New version released" --nocheckinUnchanged

--recurse --nolock --unlock --update <path>\\*.\*

The <cp> is the previously selected change package ID and <path> the absolute pathname to be checked in.

Prior to any check-in the frozen members have to be thawed. Therefore the functions described in 0 are used. After check-in the thawed members are frozen again.

### Update revision file version

This function is realized in “autorel\_main.py” using help functions from “autorel\_mks\_util.py”. See API documentation in the python file for further interface information.

If the version file is shared in the “05\_Deliverables\include” folder the member revision has to be equal to the one in “01\_Source\_Code”. Therefore the new version (after check-in) is read from the version file in “01\_Source\_Code”.

MKS command - MKS\_GetMemberInfo():

si viewsandbox --batch --quiet --fields=name,memberarchive,type,memberrev   
 <revision-file>

The filter hides the fields just enable the required information and the command is applied only on the revision file. The output is the same dictionary as defined in chapter 4.6.4.

The received version number will be used to update the shared member in “05\_Deliverables\include”.

MKS command - MKS\_UpdateMemberRev():

si updaterevision --batch --quiet --nocloseCP --changepackageid=<cp>   
 --revision=<new\_rev> <revision-file>

Finally the revision file is resynchronized (MKS\_ResyncMembers).

Prior to changing the frozen version file has to be thawed. Therefore the functions described in 0 are used. Finally the version file is frozen again.

### Initial handover document

If the handover document does not yet exist, it is added to the sandbox. This function is realized in “autorel\_mks\_util.py”, function “MKS\_IsMemberExisting” and “MKS\_AddMember”. See API documentation in the python file for further interface information.

MKS command (check if member exists):

si memberinfo --batch

If the member doesn’t exist the exit code from “si” differs from zero.

MKS command (add member):

si add --batch --description="Initial revision by AutoRelease."   
 --cpid=<cp>

### Create checkpoint

This function is realized in “autorel\_mks\_util.py”, function “MKS\_GenerateCheckpoint” and “MKS\_LabelMembers”. See API documentation in the python file for further interface information.

The checkpoint generation is separated into two steps. The 1st is the checkpoint itself and the 2nd is adding the revision label to all members recursively. The 2nd step is only performed on sub-sandboxes that are not shared.

MKS command for checkpoint:

si checkpoint --description="New version released: <label>"

--nonotify --label=<label>

MKS command for adding a label:

si addlabel --batch --forceConfirm=no --label=<label>

The <label> is the revision label generated by this script.

Both commands are executed in the sandbox root.

## Thaw members

The final action of the auto release process is to thaw all members. See chapter 0 for more information.

# Appendix

## Return values

|  |  |  |
| --- | --- | --- |
| Value | Label in script code | Description |
| 0 | AUTOREL\_OK | no errors |
| 1 | AUTOREL\_ERR\_CONSCHECK | consistency check failed |
| 2 | AUTOREL\_ERR\_LISTCHGP | getting change packages failed (not used anymore due to re-scan feature) |
| 3 | AUTOREL\_ERR\_VERFILE | the version file was not found |
| 4 | AUTOREL\_ERR\_INVVERFILE | invalid version file format (unexpected content) |
| 5 | AUTOREL\_ERR\_LOCKMEM | locking single member failed |
| 6 | AUTOREL\_ERR\_LOCKDIRS | locking release paths failed |
| 7 | AUTOREL\_ERR\_WRITEVERFILE | changing version file failed |
| 8 | AUTOREL\_ERR\_LIBSIZE | libsize files were not found |
| 9 | AUTOREL\_ERR\_REBUILD | rebuilding libs failed |
| 10 | AUTOREL\_ERR\_HANDOVER | creating code handover document failed |
| 11 | AUTOREL\_ERR\_INSTALL | the installation of the code handover failed |
| 12 | AUTOREL\_ERR\_CHECKIN | check-in failed |
| 13 | AUTOREL\_ERR\_CHECKPOINT | creating checkpoint failed |
| 14 | AUTOREL\_ERR\_PROJCONF | parsing xml project configuration failed |
| 15 | AUTOREL\_ERR\_INV\_VERNUM | invalid version number (8 bit unsigned it expected) or max number 255.255.255 (not increasable) |
| 16 | AUTOREL\_ERR\_ADD\_HANDOVER | adding handover to MKS sandbox failed |
| 17 | AUTOREL\_ERR\_NON\_BUILD | shared sub-projects found that are not of type build |
| 18 | AUTOREL\_ERR\_UPDT\_VER\_DELIV | updating delivered version file revision failed. |
| 19 | AUTOREL\_ERR\_FREEZE | Freezing complete sandbox or single member failed. See detailed error message. |
| 20 | AUTOREL\_ERR\_THAW | Thawing complete sandbox or single member failed. See detailed error message. |
| 21 | AUTOREL\_ERR\_MANUAL\_TEST | Manual tests are declared to be not successful. |
| 100 | AUTOREL\_ERR\_USERTERM\_CP | user terminated the script at change package selection (not used anymore due to re-scan feature) |
| 101 | AUTOREL\_ERR\_USERTERM\_CI | user terminated the script before committing the release |

Table : List of AutoRelease return values

## Handover tags

The following tags are included in the component handover document template and will be replaced by the “real” information.

|  |  |
| --- | --- |
| Tag | Description |
| {PROJECT\_NAME} | String. MKS name of the sandbox root project. |
| {COMPONENT\_OWNER} | Name of component owner -> taken from configuration file. |
| {DEVELOP\_PATH} | The development path if applicable. Taken automatically from sandbox. |
| {COMPONENT\_ROOT} | Complete MKS database path of the sandbox root project. Taken automatically from sandbox. |
| {CHECKPOINT\_LABEL} | Generated by script. |
| {ADDITIONAL\_CP\_LIST} | Will be generated from 'CP\_LABEL\_LIST' and 'CP\_LABEL\_PATH'. |
| {ECU\_HISTORY} | Will be replaced with history taken from last code handover docu-ment. |
| {SIM\_HISTORY} | Will be replaced with history taken from last code handover docu-ment. |
| {KIND\_OF\_DELIVERY} | "Library" or "Source" |
| {COMPONENT\_CONFIG} | List of shared projects and versions from XML files in the root folder. |
| {ISSUE\_LIST\_PR} | Problem reports, taken from release item. |
| {ISSUE\_LIST\_FR} | Feature requests, taken from release item. |
| {ISSUE\_LIST\_CR} | Change requests, taken from release item. |
| {TEST\_STATE\_SIL} | SiL test done? Yes or No. |
| {TEST\_STATE\_EVM} | SiL test done? Yes or No. |
| {TEST\_STATE\_ECU\_LAB} | SiL test done? Yes or No. |
| {TEST\_STATE\_ECU\_CAR} | SiL test done? Yes or No. |
| {USAGE\_RAM} | Number of used RAM in bytes. |
| {USAGE\_ROM} | Number of used ROM in bytes. |
| {COMPLIANCE\_REPORT} | Link to compliance report. |

Table : Supported component handover tags

## Issue workflow

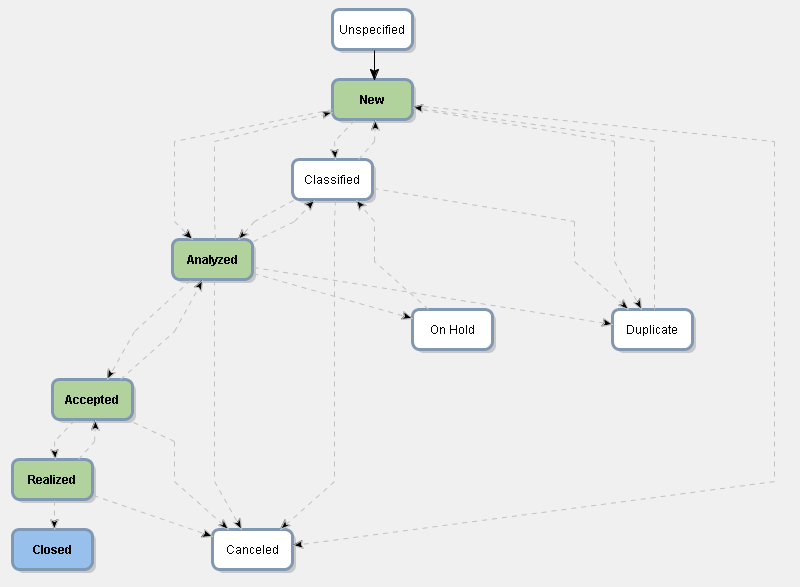


Fig. : Issue workflow