Copyright (c) 2020 [HAN University of Applied Sciences]

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Introduction

This document is intended to grasp the design decisions made during the process of developing HANtune. Each design choice is provided with a unique ID (DC\_xxx) which is used in the comments of the HANtune source code to indicate that additional documentation exists which can be found in this document.

It is encouraged to add screenshots, code snippets, links etc. to clarify the design choice.

Before starting to contribute to this document, please add your own developers’ information in chapter 2.

Developers

|  |  |  |
| --- | --- | --- |
| Initials | Developer name | Main topics |
| RvdB | Roel van den Boom | Mostly Debugging, occasionally adding small new features, quality management. |
| HvdM | Hans van de Meerendonk | Design and implementation. Mostly HANtune |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Design choices

DC\_001, HML files contain both HANtune version and HML version

HvdM, 20 jan 2017

Initial problem description: HANtune Version number in HML files not sufficient.

Detailed description of solution:

For HML and SHML files:

1) An extra version number tag will be added (in XML files):

* HML files should now contain: <version> and<HMLversion>. For compatibility reasons. HML version tag has now been placed in <Settings> section. (older HANtune version would invalidate on new tag <HMLversion>).
* SHML files should now contlain: <version> and<SHMLversion>

2) <version> denotes HANtune application version.

<HMLversion> / <SHMLversion> denotes HML/SHML layout version.

3) Version number: HANtune version="**X.Y**" will remain unchanged, due to compatibility reasons.

4) HMLversion and SHMLversion will be extended: -> "**X.Y[-<text>]**"**. X**: major, version. Y: minor version. **-<text>:** indicates an optional patch number. An xml file with only a higher patch number must always be compatible with its predecessor patch number. Whereas a higher minor number might not be compatible. A warning should be issued when minor version numbers don't match. No warning when only patch numbers don't match.

5) Assume HMLversion prior to this change to have legacy version value: **1.0-1**

Rationale: This way it is possible to warn user that HANtune <version> (or higher) of HANtune should be used when HMLversion (or SHMLversion) doesn't match.

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

DC\_002, A main branch shall be used to branch from and merge to until the first Alpha version has been released.

RvdB, 10 march 2017

Initial problem description: Branches grow too much apart from their origin making merge actions time consuming and error prone.

Detailed description of solution:

* Create a Main Branch to merge towards frequently
* This Main Branch is the future “release candidate”
* Changes which require less than a day’s work can be made directly in the Main Branch
* Changes which require more than a day’s work require branching off the Main Branch into a separate “Feature Branch”.
* Define Milestones for the Feature Branches, a merge action towards AND from the Main Branch is planned at each milestone. (to make sure both branches are in sync again)

Rationale: Frequent merge actions prevent the case in which the developers can’t exactly remember why they’ve made certain decisions, causing less errors to be made while merging. Defining milestones for feature branches forces the developer to think in sub-deliverables, making merging more efficient.

Also, having an up-to-date Main Branch enables us to showcase a preliminary version of HANtune with the newest features which are “under construction”.

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

DC\_003, Creating a new method or modifying an existing method requires new Unit tests to be made for that particular method.

RvdB, 10 march 2017

Initial problem description: Poor testing practices cause errors to go unnoticed into distribution

Detailed description of solution:

With each commit our Continuous Integration server will be triggered to build the committed code and run the GUI and Unit tests. Therefore the new functionality should be provided with new GUI and Unit tests.

* New methods will require new unit tests, following the red-green-red refactoring approach
* Existing methods which must be changed require Unit tests to be made before refactoring
* After successful implementation of the desired feature the corresponding ticket must be set to the resolution “Create GUI test scenario”.
* Only after the Unit- and GUI tests run well and pass the ticket resolution should be set to “closed”

Rationale: A good and maintainable testing procedure will significantly improve the quality of HANtune. Embedding it into the workflow prevents it from being neglected.

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

DC\_004, Students work with their own main branch, “Students\_Main\_Branch”

RvdB, 10 march 2017

Initial problem description: Students tend to isolate their main branch long enough to make merging very difficult. Merging their features with the main branch is not an option because it might involve features which may not be finished any time soon because it happens that students must stop their developments before the feature is finished due to time restrictions from the educational perspective.

Detailed description of solution:

Create a separate “Main Branch” for students to work in, according to the same principle of DC\_002.

* This branch is called “Students\_Main\_Branch”
* Each student gets his/hers own branch from the “Students\_Main\_Branch”
* The “Students\_Main\_Branch” is covered by the automated regression tests (CI)

Rationale:

This way of working prevents the unfinished features to clutter the Main Branch.

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

DC\_005, An “A3 Architecture Overview” will be maintained to facilitate new developers in getting to know HANtune.

RvdB, 11 apr 2017

Initial problem description: New developers are having trouble to get the overview of HANtune

Detailed description of solution:

An A3 architecture overview will be created and maintained. Adding new features to HANtune also means checking the A3 architecture overview for possible updates.

* A top-level overview of HANtune will be created for introductory and referencing purposes, to be used by new students, employees and later on the open-source contributors to get up to speed quickly and effectively.
* More in-depth overviews will be created for the main features of HANtune, focusing on the key features of HANtune
* These files will be stored in the doc folder

Rationale: These overviews can be used as a handy reference for new developers enabling them to get familiar with HANtune and the related aspects.

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

DC\_006, Pre-release versions of HANtune always contain the word “Test” and the branch name visible in the title bar of HANtune.

HvdM, 21 apr 2017

Initial problem description: During development it is difficult to distinguish which development state is currently running.

Detailed description of solution: HANtune versionName has been extended with a testVersionName appended. This text contains free text. It should at least contain the word 'TEST' and a global name reflecting the current branch name.

This testVersionName should ALWAYS be removed when making a RELEASE version.

Rationale: This way the title bar of HANtune gives a general idea of testversion and branch currently running.

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

DC\_007, The “Main\_Branch” will be used for Alpha, Beta and intermediate (test) releases.

RvdB, 26 apr 2017

Initial problem description: There is no clear distinction between feature branches and release candidates.

Detailed description of solution: Trunk must remain the latest stable release. In order to have an unambiguous release candidate the release candidate will always be named “<future version nr.>\_Main\_Branch”.

Rationale:

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

DC\_008, Pre-releases are only allowed to be distributed internally

RvdB, 26 apr 2017

Initial problem description: Pre-releases are not strictly coupled to a revision nr, therefore not traceable to the exact source with which HANtune was built. This makes it difficult and time consuming to trace bugs.

Detailed description of solution: Keep close track to whom you distribute intermediate test versions. The developer himself is responsible to keep track of which version is being used.

Rationale:

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

DC\_009, During Alpha or Beta stage, no new features may be added to HANtune

RvdB, 26 apr 2017

Initial problem description: Adding new features during testing stage may introduce new bugs.

Detailed description of solution: Only minor code changes are allowed after the initial Alpha release. Keep the impact of bugfixes minimal to keep risk at minimum.

Rationale:

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

DC\_010, Each Alpha, Beta and full release will be branched to a tag in the “tags” folder

RvdB, 26 apr 2017

Initial problem description: Each distribution should be traceable to the source code with which it has been built.

Detailed description of solution: Each Alpha, Beta and full release will be branched to a tag in the “tags” folder

Rationale: Keeping a traceable history is helpful in tracing old bugs

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

DC\_011, Each Alpha, Beta and full release will be merged into the Trunk

RvdB, 26 apr 2017

Initial problem description:

Detailed description of solution: Each Alpha, Beta and full release will be merged into the Trunk

Rationale: To keep the trunk “latest and greatest”

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

DC\_012, New Vx.x\_Main\_Branche will always be branched off the trunk

RvdB, 26 apr 2017

Initial problem description:

Detailed description of solution: New Vx.x\_Main\_Branche will always be branched off the trunk

Rationale: To keep the version history logical

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

DC\_012, New Vx.x\_Main\_Branche will always be branched off the trunk

RvdB, 26 apr 2017

Initial problem description:

Detailed description of solution: New Vx.x\_Main\_Branche will always be branched off the trunk

Rationale: To keep the version history logical

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

DC\_013, Features which are to be added to the Main\_Branch require their own “feature branch” when by estimate they take more than a day’s work.

RvdB, 26 apr 2017

Initial problem description: The Main\_Branch should contain finished functionality to be able to distribute at any point in time in behalf of demo’s or try-outs of new features.

Detailed description of solution: Make a separate branch for larger features to work in until the new functionality is finished, then merge back into the Main\_Branch. Make sure the feature branch has the same HANtune version nr. in the branch name and the branch name has a logical name resembling the feature on which is currently worked upon.

Rationale: To keep the version history logical

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

DC\_014, Features which are to be added to the Main\_Branch require their own “feature branch” when by estimate they take more than a day’s work.

RvdB, 26 apr 2017

Initial problem description: The Main\_Branch should contain finished functionality to be able to distribute at any point in time in behalf of demo’s or try-outs of new features.

Detailed description of solution: Make a separate branch for larger features to work in until the new functionality is finished, then merge back into the Main\_Branch. Make sure the feature branch has the same HANtune version nr. in the branch name and the branch name has a logical name resembling the feature on which is currently worked upon.

After merge, the feature branch will be deleted.

The main branch should be merged back into the feature branch frequently (>weekly) in order to minimize the eventual merge risks and time needed.

Less than a day’s work by estimate does not need a separate branch and therefore can be done directly in the main branch.

Rationale: Keep the main branch functional without generating too many separate branches.

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

DC\_015, SVN commit is only allowed when the code builds and the (sub)feature is expected to work

RvdB, 26 apr 2017

Initial problem description: A checkout should always result in clean, buildable code without runtime-errors.

Detailed description of solution: A commit may only be done if the code builds and Netbeans does not produce any errors or warnings. Committing should be done at least daily, preferably more frequent. When a (sub)functionality is working, commit it to the repository.

Rationale:

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

DC\_016, Minimize the amount of active branches

RvdB, 26 apr 2017

Initial problem description: There are too many branches

Detailed description of solution: Minimize the amount of branches by actively opening **AND** closing them when no longer needed. When a new HANtune is being released, all related branches should be closed. The upcoming HANtune version will be freshly branched off the trunk.

Additionally, from time to time, review and discuss the branches which are still active.

(note: to restore or view closed branches, use the Revision Graph of TortoiseSVN)

Rationale:

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

DC\_017, Added use of SVN revision/build number in HANtune

HvdM, 24 may 2018

Initial problem description: HANtune could not display actual build/rev number on screen.

Detailed description of solution: Now using SubWCRev (tortoise tool) to extract Highest committed revision number for a given branch. A template file (src/VERSION.tmpl) is now used to keep track of all version numbers within HANtune. When HANtune is built using NetBeans build script, SubWCRev creates a version.properties file from the template. This .properties file is placed in the base dir of a HANtune distribution. HANtune parses this file to show version and trial text at various places in the application

Rationale: make revision numbering automatic, so any version could always be traced.

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

DC\_018, In HANtune HML files, the values stored in layout area are always stored using a dot ('.') as a decimal separator. Regardless of the locale used. No grouping of thousands may be used.

HvdM, 26 oct 2018

Initial problem description: When filling in radio button options using the current locale, this would give problems.

Detailed description of solution: Now always converting a (possible) decimal comma (',' dutch locale) to a dot ('.') when saving it to HML file.

Rationale: Prevent compatibility issues with HML files.

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

DC\_019, In HML layout DBC data is now stored as <messagename>.<signalname> in the ASAP2 name field.

HvdM, 13 dec 2018

Initial problem description: When a project file with DBC data was saved, only the signal name was stored in the ASAP2 name field of a Layout. Signal names appear in multiple message names, therefore reloading a project file yielded unexpected behaviour.

Detailed description of solution: ASAP2 name field for DBC now has specification <messageName>.<SignalName>

Rationale: Prevents incorrect loading of DBC signals. Has the least amount of compatibility problems for older HML files.

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

DC\_020, Description file type now stored in source section of Layout in HML file.

Hvdm, 13 dec 2018

Initial problem description: No safe distinction could be made between ASAP2 data and DBC data in the layout area of a HML file. Filename as source is not useful when another DBC file is loaded.

Detailed description of solution: ASAP2 source section of a layout now contains 'DBC.<SendingNode>' when window holds DBC data. When holding ASAP2 data, source name remains unchanged: 'ASAP2'. The source section now starts with 'DBC' when the window contains DBC data. SendingNode (default 'Vector\_\_XXX') is now added for further enhancement, but not used now.

Rationale: Type of window can now be correctly loaded from HML files.

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

DC\_0xx, <title of DC>

<initials>, <date>

Initial problem description:

Detailed description of solution:

Rationale:

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

DC\_0xx, <title of DC>

<initials>, <date>

Initial problem description:

Detailed description of solution:

Rationale:

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

DC\_0xx, <title of DC>

<initials>, <date>

Initial problem description:

Detailed description of solution:

Rationale:

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

DC\_0xx, <title of DC>

<initials>, <date>

Initial problem description:

Detailed description of solution:

Rationale:

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

DC\_0xx, <title of DC>

<initials>, <date>

Initial problem description:

Detailed description of solution:

Rationale:

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

DC\_0xx, <title of DC>

<initials>, <date>

Initial problem description:

Detailed description of solution:

Rationale:

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

DC\_0xx, <title of DC>

<initials>, <date>

Initial problem description:

Detailed description of solution:

Rationale:

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

DC\_0xx, <title of DC>

<initials>, <date>

Initial problem description:

Detailed description of solution:

Rationale: