WFT

Workflow Tool - version 3.0

Functional Specification

For internal use only

The manufacturer has made every effort to ensure that the instructions contained in the documents are adequate and free of errors and omissions. The manufacturer will, if necessary, explain issues, which may not be covered by the documents. The manufacturer's liability for any errors in the documents is limited to the correction of errors and the aforementioned advisory services.

The documents have been prepared to be used by professional and properly trained personnel, and the customer assumes full responsibility when using them. The manufacturer welcomes customer comments as part of the process of continual development and improvement of the documentation in the best way possible from the user's viewpoint. Please submit your comments to the nearest Nokia sales representative.

Table of contents

1 Introduction

The Workflow Tool (WFT) is a web application which is used to support the management of Central Builds, SC builds and external deliveries for the Load & Build process. It acts as an umbrella for SCM and I&V tasks like the complete Load Planning and Load Building process.

The Workflow Tool models the Load & Build process as a finite state machine, where only predefined state changes are possible. The tool converts the incoming planning information into tasks, deadlines and production triggers.

It assists creating Load Plans, generating Freezing Reminders and Release Notes (Output format: HTML and XML). Information has to be entered only once and are reused to create needed output.
The WFT contains a list of features, change notes and faults which can be assigned to one or more builds.

Due to multisite development and distributed I&V across disparate geographical sites, complex product structures and the acute need for improved handover handling, the Workflow Tool was introduced.
Without the Workflow Tool all of the coordinating work has to be done via eMail and excel sheets. To avoid this error-prone manual procedure the Workflow Tool uses a predefined chain that unifies this process.

Additional the functionality of the complete WFT is also available via API (for details see chap. 4.2.2) which makes it possible to interact via CLI, e.g. doing a build handover via an XML Releasenote.

The technical realization of the Workflow Tool is based on the web framework "Ruby on Rails". For hosting the production environment of the Workflow Tool several Linux servers with Nginx and MySQL are used (for details see chap. 2.1).

1.1 Scope

This document details the overall functionality of the WFT. Due to the agile development of that tool the specification has to be adapted continuously.
The tool is independent from any product / project version. Currently it supports the LTE projects FDD (WMP, DCM and FlexiZone) and TDD, the DualMode projects microREC and HDBDE, PS, as also WCDMA.

1.2 Principles of the used Technologies

The technical realization of the Workflow Tool (WFT) is based on the web framework "Ruby on Rails" (RoR). For hosting the production environment of the Workflow Tool a Linux server with Nginx and MySQL has to be used.
Ruby on Rails is a web framework enabling you to create web applications very quickly.
Using RoR means to be familiar with:

- Ruby
- Ruby-Gems extensions
- HTML
- CSS
- JavaScript
- MySQL
  (remark: WFT uses MySQL, but Rails itself is database independent)

RoR uses the Model-View-Controller (MVC) concept which has the advantage that the user interface can easily be replaced.

For more details have a look into the various documentations dealing with RoR.

1.2.1 What is "Ruby on Rails"

Ruby on Rails is a web framework enabling you to create web applications very quickly. Ruby itself is not a new programming language, it first appeared in 1995. It was quite unknown, until Rails was offered as an add-on to Ruby. From this moment "Ruby on Rails" became widely accepted and subject to a certain amount of hype.
Rails is what is known as "a gem" (or add-on) for Ruby and consists of several parts: ActionMailer, ActionPack, ActiveSupport, ActiveRessource, ActiveRecord and Rails 3. All these parts together comprise the Rails framework.

1.3 Technology

- Framework: Ruby on Rails (Finite State Machine Plug in)
  - Ruby version: 2.1.0
  - RoR (Ruby on Rails) version: 4.1

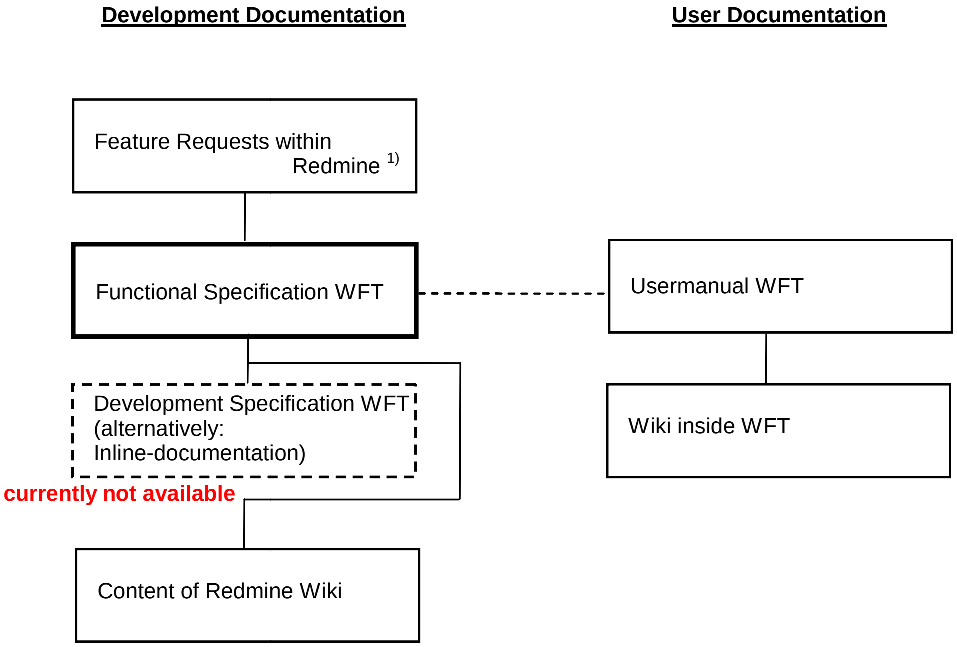
- Database: MySQL

- Web server
  - Nginx to deliver static content
  - Nginx Module "Phusion Passenger" with "Ruby Enterprise Edition"

1.4 Development History

	Ruby version	RoR version	Remark
WFT 1.0	1.8.7	2.3	Development started June 2008
WFT 2.0	1.9.3	3.2.8	Rollout: 30.07.2012
WFT 3.0	2.1.0	4.1	Rollout: 02.07.2014

1.5 Documentation Overview


Development Documentation
User Documentation
Feature Requests within Redmine [1)]
Functional Specification WFT
Usermanual WFT
Development Specification WFT (alternatively: Inline-documentation)
currently not available
Wiki inside WFT
Content of Redmine Wiki

[1)] the feature requests out of the Redmine WFT Backlog will be referenced throughout the document by their number (e.g. "#1234")

Figure 1: WFT Documentation Overview

Remark (general formatting conventions):
where applicable throughout of this document within each chapter either the model and the used method, or the controller and the used method, or the view where the main part of the described functionality can be found will be listed at the beginning of that chapter.
E.g. "BuildsController#index" => method index of class BuildsController (to be found within ../app/controllers/builds_controller.rb).

1.6 Scope of the document

This document represents a functional description of the Workflow Tool (WFT). Due to the agile development and the continuously improvement of the tool it is not possible to be up to date at any time. Therefore the document will be continuously under construction.
In case of some disagreements are figured out, let the WFT team know and the document will be corrected/updated soon.

1.7 Outstanding Issues

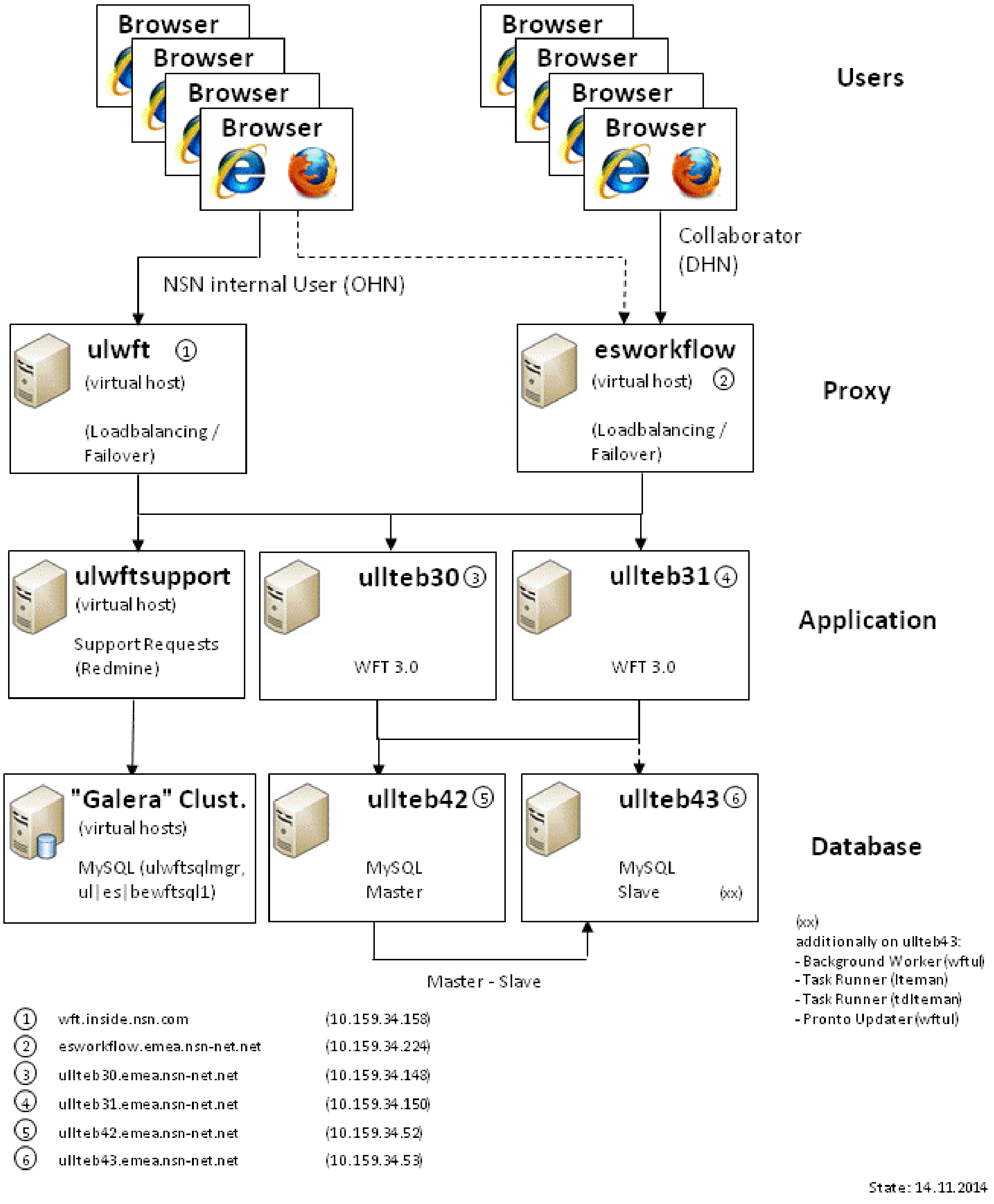
- API (see chap. 4.2.2 and chap. 3.12.1):
  - rework as soon as all API descriptions (CLI commands) are available.
  - sub-chapter about wrapper script has to be adapted.

- Chap. 6 "Set-up of a new Project" has to be reworked.

- All named controllers have to be verified.

2 WFT Hardware

2.1 WFT Server Setup



①	wft.inside.nsn.com	(10.159.34.158)
②	esworkflow.emea.nsn-net.net	(10.159.34.224)
③	ullteb30.emea.nsn-net.net	(10.159.34.148)
④	ullteb31.emea.nsn-net.net	(10.159.34.150)
⑤	ullteb42.emea.nsn-net.net	(10.159.34.52)
⑥	ullteb43.emea.nsn-net.net	(10.159.34.53)

(xx)
additionally on ullteb43:
- Background Worker (wftul)
- Task Runner (lteman)
- Task Runner (tdlteman)
- Pronto Updater (wftul)

State: 14.11.2014

Figure 2: WFT Server Setup

Server	Functionality	Specification
ulwft / wft.inside.nsn.com	Proxy-Server (ViHo) for internal Users Loadbalancing (WFT 3.0, Redmine) Failover Apache/SiteMinder/WebAgent	3 Cores, 8 GB
esworkflow.emea.nsn-net.net	Proxy-Server (ViHo) for Collaborators Loadbalancing (WFT 3.0) Failover Apache/SiteMinder/WebAgent	
ulwftsupport	Application-Server Redmine (wftul; Support Requests) Nginx/Phusion Passenger	3 Cores, 4GB
ullteb30	Application-Server WFT (wftul) Nginx/Phusion Passenger	24 Cores, 48GB
ullteb31	Application-Server WFT (wftul) Nginx/Phusion Passenger	24 Cores, 48GB
ullteb42	MySQL DB (Master)	8 Cores, 32 GB
ullteb43	MySQL DB (Slave) Background Worker (wftul) Task Runner (lteman) Task Runner (tdlteman) Pronto Updater (wftul)	8 Cores, 32 GB
ulwftsqlmgr	DB Galera Cluster Manager	3 Cores, 4GB
ulwftsql1	DB Galera Cluster Node Ulm (x)	4 Cores, 8GB
eswftsql1	DB Galera Cluster Node Espoo (x)	4 Cores, 8GB
bewftsql1	DB Galera Cluster Node Beijing (x)	4 Cores, 8GB

(x) hosts currently exclusively the Redmine DB

Remark (13.05.13): in future the proxy servers will offer a service which handles the loadbalancing for the application as also for the database (database will then be placed within the Galera Cluster).

Available Ports for https://wft.inside.nsn.com:

443	Standard Port - HTTPS connection
8080	WFT Backlog (Redmine)
8091	Port for API access (bypasses WAM authentication)

3 Functional Description

3.1 Overview

The Workflow Tool (WFT) is in the first place a web application. The GUI is divided into several areas (see below) to fulfill the given tasks.
Additionally the complete functionality offered by GUI is also available via API (see #267) which makes it possible to interact via CLI.
Some of these areas are hidden to a standard user [1] and need permission. These permissions have to be requested by means of "Permission Request" (see menu "SUPPORT") and are manually granted by the WFT administration.

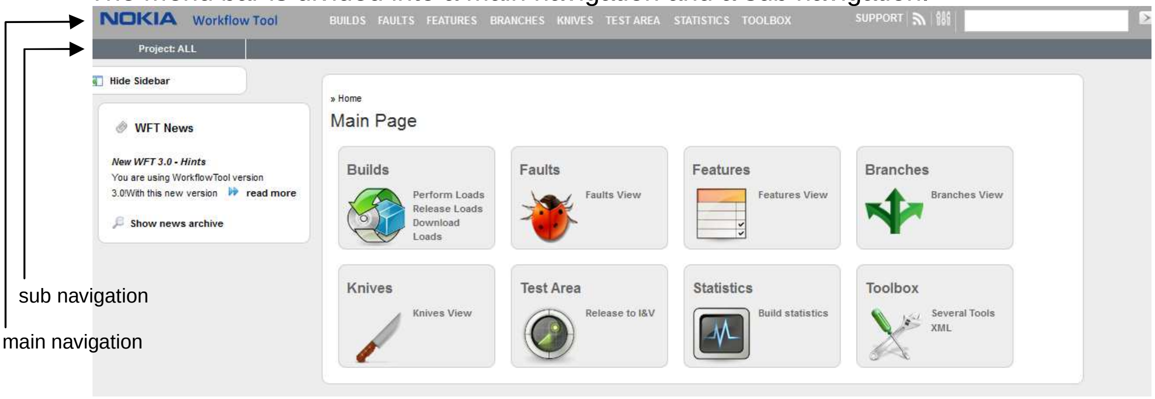
Area	Sub-area	Function
Builds		information and SW download about/of eNB builds; perform and release loads
	Load Planning	view and schedule loads
Faults		search for a specific pronto; pronto report; pronto diff
Features		search for a specific feature
Branches		overview of branches
Knives		viewing knives, requesting a knife
Test Area [1]		release to I&V and send release mail for Quicktest Phase 1 and Phase 2
Statistics		graphical view of several statistics
Toolbox		XML validation / import; FEP search
eNB Notification		subscription for RSS Feeds
Administration [1]		configuration, permissions, templates (restricted to WFT administration)
	Build Configuration	
	Configuration	
	Deliverers	
	Images	
	Logging	
	Permissions	
	RN-Templ. (HTML)	
	RN-Templ. (XML)	
	Storages	
	Translations	
Management [1]		user management
	Branches	
	Releases	
	Branch Types	
	System Releases	
	Permissions	permission handling for Key-User

Figure 3: Overview WFT Areas

3.2 General page elements of the GUI

3.2.1 Menu Bar

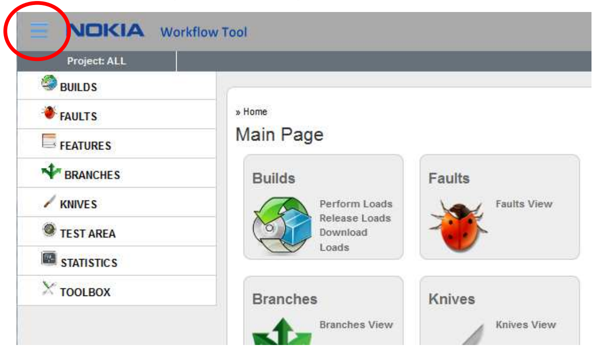
The menu bar is divided into a main navigation and a sub navigation.



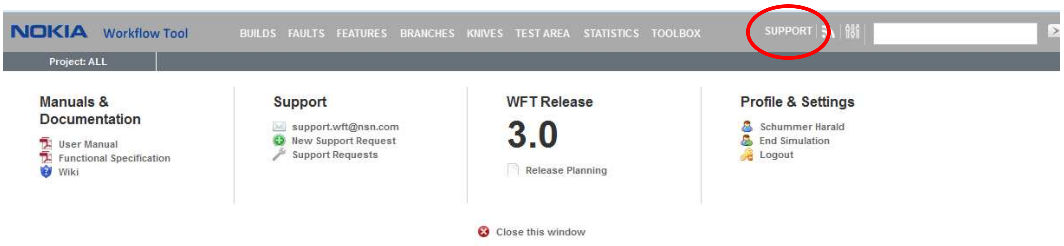
sub navigation

main navigation

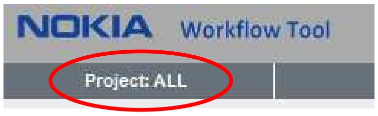
Selection of the areas can be done via the main navigation, or (in case the window is too small) via a menu selection on the left top corner, or at the main page using the relevant box.



Using menu "SUPPORT" the user
- has access to various documentations like the user manual, this functional specification and the WFT internal Wiki (various user information).
- can write a new support request or watch existing support requests
- gets information about the current WFT version
- has the opportunity to edit his profile



3.2.2 Project selection



Project selection works per browser tab using the menu "PROJECT" within the sub navigation.
(i.e. different browser tabs can be used for different projects).
The selected project is also visible within the URL, which can be used to set needed bookmarks (e.g. https://wft.inside.nsn.com:8081/WMP/builds/ chronological)

The menu bar offers the user an extensive search functionality (for details see chap. 3.2.3).

3.2.3 Search functionality

SearchController#index
SearchController#ajax_search

With the extensive search functionality an user can find an expression he is looking for separated within several categories (e.g. Build, Fault, Feature, Knife Requests, Wiki, etc.)
The search algorithm is not case sensitive.
As soon as he has entered the expression a pre-selection is showing the latest 3 hits of each category.
Clicking on the ">"-icon will show all hits and the user can walk through the displayed tabs to have a look to them.

3.2.4 Sidebar

Most of the pages within WFT have on the left-hand side a sidebar which offers additional information and/or links to other pages. This sidebar can be hidden if needed, e.g. because of space requirements.
As soon as hidden a touch with the mouse pointer on the link "Show Sidebar" makes the sidebar temporary visible.
To unhide the sidebar the link "Keep Sidebar" has to be used.

3.2.5 Publishing a page

Every page of the WFT can be published by printing it or sending the content to the user. Where applicable an XML file can be generated.

3.3 Area Builds

This area is the central and most used area of the WFT. Via that page all builds managed by the WFT can be accessed.
Within the Load & Build process there exists three different categories of builds. Each build category has different requirements and characteristics which have been modeled within the Workflow Tool.

Central builds are the highest level of builds, e.g. LTE / eNB builds. They are compiled out of the SC builds and External builds. This load will be delivered to Quicktest (QT).

SC builds are those deliveries (components), which are delivered to a central build (as also to other SC builds, or External deliveries) and are planed and controlled via WFT.

External deliveries are software packages needed for builds which are not planned in detail and are not controlled via WFT. They are produced outside WFT and provide information about the build via API and XML Release Note to WFT. These are SCs like BTSOM, MAC, TUP, PHY_RX, Linux OS, parts of Platform, etc..
External deliveries deliver to Central Builds, to SC builds, or to other External deliveries.

General remark: the expressions "builds" and "loads" are used synonymously throughout this document.

There are different views to display all the builds contained within WFT (chronological view of eNB builds, weekly overview of eNB builds, chronological overview about all builds). Using the baseline links the user gets detailed information of the chosen build (for more details see chap. 3.3.6).
At least every build can be displayed with a build specific detailed view. Within this view all data concerning that build is visible and depending on the build category the build can be managed (e.g. incrementing that build).

3.3.1 Chronological Overview of eNB builds

BuildsController#index

On this page, which is also the entry site of that area, all central builds of the past two weeks are shown in a chronological order.
Builds with the following states are displayed: released, released with restrictions, not released, error, released for Quicktest, canceled and blocked.

3.3.2 Weekly Overview of eNB builds

BuildsController#index

The weekly overview summarizes all central builds of one calendar week.
The baseline names are colored to show a quick overview of released/not released builds.

green	-> RELEASED
yellow	-> RELEASED_WITH_RESTRICTIONS
black	-> RELEASED_FOR_QUICKTEST
red	-> NOT_RELEASED

The sidebar offers the possibility to select a certain calendar week directly.

3.3.3 Overview of Trunk baselines

BuildsController#index

Same view as described within chap. 3.6.3.

3.3.4 Load Building - Builds Overview Table

BuildsController#index

3.3.4.1 Overview

This overview table lists all builds from the database table "builds". On the first call the users default filter is taken and used to generate the table.
A selection of the categories "CENTRAL", "SC" and "EXTERNAL" can be achieved.

3.3.4.2 Filter and sorting methods

Using a form each user can define own filters for several criteria's (e.g. baseline name, branch, state, date range, sorting and paginating criteria's, etc.), he can store them and can make them public.
These filters are stored in the table "custom_filters". The current applied filter information stored in that form can be edited and resent by the user.

3.3.5 Load Planning

This area is readable by all users, but only can be edited by Load Coordination.

Members of the Load Coordination write to the planning sheet and after fixing a new build configuration information is sent to the build team to start the build. All other users may read the planning sheet.
The user selects the branches for which he needs to manage builds out of the list of visible branches. For every selected branch a new tab within the GUI will be opened. Within each tab the user is able to create and configure a new eNB build.
The content of every tab can be exported to an excel sheet.
The user (Load Coordinator) needs permissions for that area as also the permissions for the eNB builds he wants to create to see and use the button "Create a new baseline" (visible within the first row of each column).
After pushing that button a formula is displayed where the user has to select the newly used baselines.
After pushing "Create a new baseline" a similar formula as that for creating a new baseline (compare chap. 3.3.10.3) is shown and here additional entries can be done
Within the two additional tabs ("Notes" and "Advance") some more information can be entered.
As soon as the build is started via this area an email notification is sent to build team.

3.3.6 Detailed view of a build

BuildsController#show

Every baseline / build has an own detailed view. Within this view the user gets detailed information about the build he has selected and the build responsible can manage and edit all information concerning that baseline.
The status of the build is shown and within 8 tabs all the information about the build can be found.

3.3.6.1 Details

BuildsController#edit

Within this tab the user gets some information about that build or he can do some basic settings for a build, e.g. branch selection, deadlines, or the selection of the used release note templates.
To see the predecessor and the successor of the build a hierarchy of the build is displayed.

3.3.6.1.1 Basic Settings

Basic settings of a baseline like the branch this baseline belongs to and the SVN repository of the load can be edited.

3.3.6.1.2 Deadlines

The build relevant dates as, the date of the freezing reminder, the freezing date, the planned production date and the production date itself can be edited.

3.3.6.1.3 Templates

The templates for the HTML release note (see also chap. 3.12.7) and the XML release note (see also chap. 3.12.8) can be selected.
The user selects if the build switches automatically from state buildable to frozen or if manual interaction is necessary (checkbox "check before freeze" is checked, see also chap. 3.3.7).
Additional the information concerning the alignment of build scripts ("SCM Tool") can be entered.

3.3.6.1.4 Build Config

The Build Configuration Spec has to be selected and if needed a patch can be entered.

3.3.6.1.5 Notes

Some notes concerning the build can be entered. This date will be overtaken to the release note.

3.3.6.1.6 Storages

The build responsible selects the needed storage(s). For details concerning these storage areas please refer to chap. 3.3.6.7.3, resp. chap. 3.12.9.

3.3.6.1.7 Build Notes

Build Notes are assigned to one build and are mainly used for information transfer regarding one specific build. This information is displayed as a yellow sticky note within the Details tab.
On the Builds Overview table a yellow sticky note icon is displayed if the build contains a build note.

3.3.6.1.8 Build state

The state of each baseline is displayed beside the baseline name.

3.3.6.2 Baselines - Deliverers

BuildDeliverersController#index

A Deliverer defines a particular set of baselines and is also called sometimes "component". Deliverers are used to assign necessary baselines to a build. A deliverer consists of a title and a regular expression (for details refer to chap. 3.12.1). The regular expression defines the format of the baseline. A deliverer can be assigned to a build and assigns in this way a matching baseline to that build.

This tab contains the list of the SC (resp. sub-) baselines. Baseline names of baselines which are available within WFT are linked. Names of baseline which are not known within WFT are listed only for information (see also chap. 4.2.3.3).
Fallbacks of a baseline are marked with the string "Fall Back!" and will also be mentioned within the relevant Rel.-Note and -Mail.

3.3.6.2.1 Baseline Configurations

For system components and external baselines it is possible to define several sets of baseline configurations (see #12143, e.g. LN7.0_MAC_PS_WMP_1407_191_00). The set of baselines used for the first import / the first generation of the baseline is used as "Original Configuration".
People responsible for this baseline can later on report further baseline configurations where this baseline is also compatible to.
There are 3 different ways of reporting this compatibility.

3.3.6.2.1.1 Use the "Edit Baseline Configurations" section via GUI

Using the GUI it is possible to copy an existing configuration or create a new one from scratch. New configurations are by default in a draft state. This means that baseline configurations are not active and can be adapted. Once a configuration is published there are no more changes possible. Only published configurations are considered for SC Compatibility Check.

3.3.6.2.1.2 Import an updated xml release note with changed baselines

When importing an updated xml release note with changed baseline section there's automatically a new baseline configuration added with this new set of baselines. This new baseline configuration is automatically published.

3.3.6.2.1.3 Report compatibility to a new ECL Sack Baseline

When reporting compatibility to a new ECL Sack Baseline there's automatically a new baseline configuration generated containing content of this ECL Sack Base.

3.3.6.2.2 Status

The Status of an assigned deliverer is defined as:

- no baseline selected	red / no
- baseline selected but not marked as delivered	yellow / pending
- baseline selected and marked as delivered	green / yes

The status of all assigned deliverers for one baseline has also effects on the build state. As long as there are some deliverers not available (marked as red or yellow) the baseline stays in ANNOUNCED and can’t be frozen.

3.3.6.2.3 Resolved Flag

To list e.g. all corrected faults (prontos) of a certain deliverer it is necessary to have an uninterrupted chain of parent and child baselines of that deliverer.
The resolved flag shows if the baseline history could be resolved, this means if there exists such a continuous chain of parent and child baselines.
If the WFT does not succeed to resolve the history that baseline is marked as "not resolved". In this case the user can try to resolve the history by entering the "based on" baseline(s).
Let’s explain this also with an example:
CentralBuild_1 uses Mac_1
Mac_1 is the parent baseline of Mac_2 and Mac_2 is the parent baseline of Mac_3
CentralBuild_2 uses Mac_3
In CentralBuild_2 the Mac_3 is only marked as resolved if the relationship between Mac_1 -> Mac_2 -> Mac_3 is correctly assigned. This ensures that e.g. faults from Mac_2 get listed in CentralBuild_2 release note.

This check is done automatically every time the page with the list of baselines of an eNB load is opened and the content of the "Warnings" tab of a build is refreshed.

3.3.6.2.4 Inheritance of Baselines

Beneath selection directly of a specific baseline per deliverer (e.g. Deliverer: LTE::LOM -> Build: LN6.0_LOM_:1305_24952) it is possible to chose "inheritance". With this a certain baseline inherits that particular baseline which is selected by the baseline from which is inherited.
To activate this inheritance it is necessary to (re-) create the relevant deliverer by checking the checkbox "Inheritance".
Within the column "Build" the deliverer has to be selected from which the deliverer in column "Deliverer" inherits.
E.g. for an ENB load deliverer LTE::ASN1_IF has to inherit from the selected baseline of ISAR_LIB, then the selected "Build" has to be "ISAR_LIB" for that deliverer.

3.3.6.3 Tasks

TasksController
(Within \lib directory there exists one file per task type containing execution details)

Tasks are script calls or other executions which are performed in the scope of a build. Using a form these tasks can be created, can be edited and can be deleted from every build. Details like actions or the result of the added task can be displayed.
Tasks are based on defined task types (for details refer to chap. 3.3.6.3.7) using different options.
Execution of the tasks is done by the Task Runner (a ruby script which is responsible for executing all tasks in the context of a specific UNIX user), for details refer to chap. 4.2.
Each task can be started, restarted, or recreated. In case of "recreate" a prepare script is called and the task switches to state "Tainted".

3.3.6.3.1 Task State

The task states per active build page get periodically every 5 seconds updated or can be updated on request. Tasks have a state attribute which defines the current status of the task.

New

Every task starts with this state. The task was just created and preparation steps have not been performed for this task. The Task Runner searches for tasks in state NEW and executes depending on the task type some kind of code/script.
When the execution succeeds the task will switch to PREPARING.

Preparing

The preparation steps for this task ran.

Prepared

The preparation steps for this task have been finished and the task could be started.
Tasks in state PREPARED can be started by the user or automatically as element of a chain.

Blocked

If there are problems with the task preparation it will switch to state BLOCKED. The preparation steps for this task didn’t succeed. Information regarding the errors may be found in the task log depending on the task type.

Started

Tasks which were started are in state STARTED. This is the indicator for the Task Runner to execute the script for this task.
The task is marked for running.
The task is currently not running, but will be started soon.

Starting

For Tasks where the starting process takes longer this state indicates that the starting process is running and prevents tasks to be started several times.

Running

As soon as the script is executed the Task Runner switches the task to state RUNNING.
The task has been started and is currently running.

Success

Depending on the result of the execution the task is switched to SUCCESS or ERROR. In case of SUCCESS the executed script succeeded and the task is completed.

Error

Depending on the result of the execution the task is switched to SUCCESS or ERROR. In case of ERROR the executed script

	failed and the errors may be found in the task log depending on the task type.
Tainted	If the task configuration has to be rebuilt this is done by switching to state TAINTED. The Task Runner will switch the task to state NEW as soon as the cleanup has finished.
Deleting	The task is being marked for deletion and will disappear soon.

3.3.6.3.2 Task Actions

For each task type there are predefined actions which depend on the current status of the task. Task Actions are used to start tasks or perform different triggers depending on the task type. Each task type can define its own actions.

3.3.6.3.3 Task Settings

Task Settings are attributes necessary for the task execution. Depending on the task type these attributes are passed as script parameter or as file. Task Settings can be free text or selectable lists which itself are provided by the task script. Task Settings have default values which can be restored when incrementing a build. All other Task Settings are taken from previous baseline when incrementing a build.

3.3.6.3.4 Task Result

The task result is usually taken from the exit code of the script and defines the task state after "running". The log is stored for a few days inside the database and can also be displayed.

3.3.6.3.5 Task Executions

For each task execution a log entry is added which describes the task settings. There's a history available displaying all executions and results for a specific task.

3.3.6.3.6 Task Interactions

Tasks can be automatically started depending on the state of another task and / or are able to interact with the state of the build they are assigned to. This way tasks can be chained together to run after each other.
Tasks can also change the status of a baseline depending on the task result.
Additionally it is possible to start a task on timer events.

3.3.6.3.7 Task Groups

Task groups are used to visually combine tasks which belong logically together. Inside a task group tasks can be sorted by using drag & drop.

3.3.6.3.8 Task Types

A task type defines the type of a task and the behavior in the back. Some task types interact with other tools like Jenkins (or Hudson) or Cruise Control and some task types execute a simple script waiting for the exit code.

Currently following task types are defined and will be described in details below:
Dummy; Sleep, Script Caller, Hudson/Jenkins Build, Mailer, ECL Check, Resolved Check, Jenkins, BranchFor Check, SC Compatibility Check and Correction Allowed Check.

3.3.6.3.8.1 General Settings for Tasks

For every task the user has to set:

- Server or Group: the server or the server group on which the task has to run. (hint: not necessary for definition of several tasks, but has to be filled as this is a mandatory field)
- User: the user (functional account) which has to run the task.
- Identifier: has to be used for an unique identification of that task. (hint: will be replaced in future by the task_id)
- Title: the title (resp. the name) of the task (will be displayed within the task list)
- Settings: here the user can set several task type specific settings (e.g. Result, Command, View).
  For task type specific details see the chapters of the relevant task types.
- Build Interactions: within that tab the user can define several build interactions:
  as soon as the selected action(s) of that task occur(s) the build switches to the selected build state.
- Task Interactions: within that tab the user can define one or more other task(s) and a state for that/these task(s) on which the concerned task will be auto started.
- Task Timers: within that tab the user can define a time after which the task will be auto started.
- Task Notifications: within that tab the user can define notifications (mails) to be sent depending on the state of the task.

These settings can be changed using the "Edit" link of the task or can be displayed using the "Details" link.

3.3.6.3.8.2 Dummy Task

This task is used by SCs for generating state changes.
Task specific settings:
- Result ("success" or "error"): the predefined result of that task.

3.3.6.3.8.3 Sleep Task

This task type provides a configurable delay.
Task specific settings:
- Timeout: the timeout in seconds.

3.3.6.3.8.4 Script Caller Task

This task type calls the script wftRunner.pl (a kind of wrapper script) which is in the responsibility of the SW Build Team. All the commands contained within this script are display and can be selected.
Task specific settings:
- Command: command to be selected.

3.3.6.3.8.5 Hudson/Jenkins Build Task

This task is used to start a Hudson or Jenkins build.
Task specific settings:
- Hudson Project: the relevant Project out of Hudson has to be selected.
- Hudson JOB-ID and Unique Identifier are set by WFT.
  The Hudson Job-ID is an identifier created by Hudson and set within WFT as soon as the Hudson Job is running on the Hudson Server.
  The Unique Identifier is a random number created by WFT during start of the task and is linked to the Hudson-Job-ID.
- Prod Type: parameter defined and used by SW Build Team.
- Hudson Server: Server where the build is running.

- Force Rebuild: selection if the build can be rebuild or if it has to be build completely new.
- Send Quicktest Information: information to the Quicktest Team, indicating the new baselines and the fixed prontos, can be send as soon as the build has started.
- Build Parallel: parameter defined and used by SW Build Team.
- System Module: parameter defined and used by SW Build Team.
- Check existence of previous tag: the by default activated check can be disabled.

3.3.6.3.8.6 ECL Check Task

This task type serves for a task doing the ECL Check (for details refer to chap. 3.3.12.3)
Task specific settings:
- Inform on Failure: the task result in case of a failure can be sent to those email addresses (separated by a ";") which are entered here.

3.3.6.3.8.7 Resolved Check Task

This task type serves for a task doing the Resolved Check (for details refer to chap. 3.3.6.2.3).
Usually the resolved check is done every time the page with the list of baselines of an eNB load is opened.
This task can be used e.g. in case of the relevant build has to go to error in case that check fails, or it supports the build responsible as he gets informed in case there are some unresolved components (baselines).
Task specific settings:
- Inform on Failure: the task result in case of a failure can be sent to those email addresses (separated by a ";") which are entered here.

3.3.6.3.8.8 Jenkins Task

Similar to Hudson/Jenkins Build Task (see chap. 3.3.6.3.8.5) but without build specific parameters. Has to be used to start a Jenkins-Job only.

3.3.6.3.8.9 BranchFor Check Task

This task type serves for a task doing the BranchFor Check (see also chap. 3.3.12.5).
Task specific settings:
- Inform on Failure: the task result in case of a failure can be sent to those email addresses (separated by a ";") which are entered here.

3.3.6.3.8.10 SC Compatibility Check Task

This task type serves for a task doing the SC Compatibility Check (see also chap. 3.3.12.2).
Task specific settings:
- Inform on Failure: the task result in case of a failure can be sent to those email addresses (separated by a ";") which are entered here.

3.3.6.3.8.11 Correction Allowed Check Task

This task type serves for a task doing the Correction Allowed Check (see also chap. 3.3.12.6).
Task specific settings:
- Inform on Failure: the task result in case of a failure can be sent to those email addresses (separated by a ";") which are entered here.

3.3.6.3.8.12 Build Interaction Task

(see #12411)
Task specific settings:
- Call method: method of Build model which will be executed. Currently only "run_autorestart" can be selected.
- Parameters for method call: parameters passed to method. For "run_autorestart" should be empty.
- Call condition: boolean value of Build model that will be checked.
- Condition value: state of "Call condition" that must be set to prepare task.
  Call condition works simply:
  If "call condition" == "Condition value" task will be started.
  For "run_autorestart" only "autorestart" boolean is possible. Call condition value should be set to "false", because all new builds have "autorestart" set to "false" and after "run_autorestart" triggered it switches to "true".

3.3.6.4 Build Configuration

BuildsController#build_config
Build#generate_config_spec

Each CB build has a Build Configuration template assigned. This template defines the format of the configuration file necessary for the build scripts used by the Build Teams. The template consists of several block elements (for details refer to 3.13.2).

3.3.6.5 Build Content

BuildsController#show
Build#processed_corrected_faults
Build#processed_important_notes
Build#processed…

The content of each Build consists of Features, Unsupported Features, Corrected Faults, Reverted Faults, Changes, Important Notes, Workarounds, Restrictions and Needed Configurations.

3.3.6.5.1 Features / Unsupported Features

BuildFeaturesController#index, BuildUnsupportedFeaturesController#index

Features have an unique ID and a title and can be marked as visible (default value) or not visible for Panasonic (PMC). They can be assigned to baselines.
Additionally there exist so called "unsupported features". Unsupported features are features which should be supported within a build/baseline, but cannot be supported for any reason.
Features and unsupported features are entered per SC via the GUI or per XML Release Note.
For a central build features and unsupported features are taken from LDR. For details concerning the data import/handling of the LDR refer to chap. 3.17.2, resp. chap. 4.4.5.
Features can be added to, can be edited and can be deleted from an SC build.
Supported and unsupported features are displayed per build separately.

3.3.6.5.1.1 Corrected and Reverted Faults

BuildFaultsController#index

Faults (also called "prontos") have a unique Pronto ID and are basically handled within the Pronto Tool. Within the Workflow Tool faults are used in the meaning of "corrected

faults (prontos) with this baseline". That means, within tab "Corrected Faults" all corrected faults compared to the predecessor of that build are displayed.
A reverted fault is a fault which was corrected within a previous build but is currently due to any reason no more corrected. Within tab "Reverted Faults" all these faults are listed (#7982).
For interaction with the Pronto Tool please refer to chap. 4.4.2.

3.3.6.5.1.2 Mechanism of fault collection

For a central build faults are collected from all assigned baselines in a special way. Only prontos corrected for the first time appear in a central builds release note. This means that prontos are only collected from changed baselines. For all of these changed baselines there is a special mechanism in use which collects also prontos from baselines that were released between the current and the last used baseline. More details can be found in chap. 3.3.6.2.3.

3.3.6.5.1.3 Attached prontos

"Attached prontos" are faults which are connected to or depend from another pronto and are automatically solved when the first one is solved. WFT searches in the attached column of a pronto for Pronto IDs and assigns the "attached prontos" automatically to the according baselines.

3.3.6.5.2 Changes / Changenotes

ChangenotesController

A Changenote describes a small or late feature. Changenotes have a unique CN ID and a title and can be assigned to baselines.
Changenotes are entered per SC via the GUI or per XML Release Note.
In a Central Build Changenotes are inherited from all assigned baselines.

3.3.6.5.3 Important Notes

BuildsController#edit

Important Notes is information to be displayed in the release note and the release mail (depending on the template). The information can be assigned to a build and for central builds this information is inherited from the assigned baselines. Important Notes can be entered in free text fields.

3.3.6.5.4 Workarounds

WorkaroundsController

Workarounds are textual information provided by an SC to be placed into the relevant release note and can be assigned to a baseline.
Workarounds are entered per SC via the GUI or per XML Release Note.
In a central build workarounds are inherited from all assigned baselines.

3.3.6.5.5 Restrictions

RestrictionsController

Restrictions are textual information provided by an SC to be placed into the relevant release note and can be assigned to a baseline.
Restrictions are entered per SC via the GUI or per XML Release Note.
In a central build restrictions are inherited from all assigned baselines.

3.3.6.5.6 Needed Configurations

NeededConfigsController

The "Needed Configuration" is some kind of textual information, which is needed by some SCs (e.g. CPlane) and is a pair of component/configuration which defines configuration values for a certain component (deliverer) of this baseline.
A needed configuration is entered per SC via the GUI or per XML Release Note.
In a central build needed configurations are inherited from all assigned baselines.

3.3.6.6 Assignments

3.3.6.6.1 Used In

Build#processed_used_in
Build#processed_ecl_used_in

(see #212, #643, #817)
For builds of category "SC" or "External" those baselines (out of all categories) are listed within the table "Used In", which are using this build. The baselines are grouped by the branches they belong to.

For ECL_SACK_BASE baselines an additional table "ECL Used In" is displayed.
This table contains those "SC" and "External" baselines which are compatible (see also chap. 3.3.6.6.2) to that ECL_SACK_BASE. This means here are also baselines listed which have that ECL_SACK_BASE in their "ECL Compatibility" list, but not in their "Baseline" list (tab "Baseline, resp. the XML section "baselines").

3.3.6.6.2 ECL Compatibility

Build#processed_ecl_baselines

If applicable the compatibility of the baseline to one or more ECLs ("ECL_SACK_BASE" baselines) is shown within that list.
Each baseline can be compatible to more than one ECL. This list is filled each time
a) for EXTERALs: a new XML file for that baseline, containing another ECL, is imported, or
b) for SCs: can be edit using menu "Edit" / "ECL Compatibility".

For general information about ECLs refer to:
- https://bts.inside.nsn.com/twiki/bin/view/PreIntegration/PO_ECLMaintenance-Branches, or
- https://swikis.inside.nsn.com/bin/view/LTECI/ECL, or
- https://confluence.inside.nsn.com/display/ECL/Home.

3.3.6.7 Release / Download Information

This tab is furthermore divided into several sub-tabs and contains release information and all downloadable data concerning that build.

3.3.6.7.1 Release history

BuildsController#show

The release relevant dates, such as the date / time when the build released / succeeded, when it was released by CB to Quicktest and when it was released / released with restrictions / or not released by Quicktest are listed.

Where applicable the quicktest team results (CRT Info) are also shown.

3.3.6.7.2 Attachments

BuildAttachment

Attachments of a build are files like release notes or other documents. The files are stored on a file share and ordered automatically in a predefined path structure. Only metadata about the attachments is stored within the database. There are several attachment types which define the type of attachment.
Attachment Types: HTML Release Note, XML Release Note, Needed Configuration or Others.
These attachments consist of (mostly) all automatically generated release notes (XML as also HTML files) as also every document which the user wants to add manually.

Via that tab the user gets those build attachments belonging to the concerned build as also those of its sub-builds.

Hint: HTML and XML release notes are generated in case of the build (SC and CB) is released. For eNB builds in state not released also release notes are generated and stored.

3.3.6.7.3 Build results

Build#processed_storages

Build Results:
Build results are the outcome of a build, such e.g. the complete load, MAP & OUT files, etc. This data is stored on the file share /lteRel within so called "storages" and ordered automatically in a defined path structure (the definition of these storages is described within chap. 3.12.9). Only metadata about the build results is stored within the database.
The build results within theses storages are those build results which have been created during build creation using the GUI of WFT.

Build Artifacts:
(see #7786 and #10453)
The build artifacts are used for storing build result information out of the download section of the XML releasenote, in other words: the build artifacts are available after an XML import (table build_artifacts; model BuildArtifact).
If the storage is named jenkins or hudson it will be tried to load a list of all available files from jenkins server and provide them as download links.

3.3.6.7.4 Tools

Within that tab the user can fetch all used build tools for testing and/or installation purpose (e.g. BTS Site Manager) for that build.
These build tools are stored on "LTESDK_Root" (see share /ltesdkroot/ltesdkroot/SC. The storage places are configured (i.e. definition of the mapping "build tool" <-> "storage place") within the tables "tools" or "deliverer_attachments" (how to administer, refer to chap. 3.12.3.1.12).

For some build tools (e.g. "BMU Images") only a link to SVN is provided (i.e. no direct download is possible, as providing such a direct download out of subversion causes too much overhead on WFT side as it would be necessary to make an "export" for each download).
In special cases where the SVN links may change (see e.g. #10456 for BTSSM, or #9992 for FRM), the storage place can be read out of the concerned XML file. This has to be controlled using the flag "taken_from_xml" (within table "deliverer_attachments").

3.3.6.7.5 Links

This tab is offered only where applicable.
The user gets links to builds stored within SVN repositories (e.g. to the IDA2 sack).
The relevant links are stored within table "build_links".

3.3.6.7.6 XML Releasenote

Using a convenient form the content of the builds XML file is displayed.

3.3.6.8 Warnings

Build#warnings_available?

The building of a load is supervised by several checks (see chap. 3.3.12). The results of these checks are displayed within sub-tabs (one own tab for each check) within the Warnings tab of the build. Only for failed checks such a sub-tab is visible.
The whole Warnings tab is visible only for users who have the needed permissions for the concerned build.

3.3.6.9 Build history

BuildsController#history

For each baseline many actions which have been performed by the user or which are done automatically (e.g. by scripts) are logged within a history table. This information is chronologically sorted and displayed.

3.3.6.10 Auto Release mechanism

Build#release

Central and SC builds can use the auto release mechanism. Therefore the auto_release flag of the build has to be set to true. This can be done using the "Auto Release" button displayed in the builds details section.
The Auto Release mechanism is handled inside the Background Worker (see chap. 4.3.2). The Background Worker checks periodically if there are builds available which are in state SUCCEEDED and have the auto_release flag set. If there are builds found the build is added to a ReleaseQueue and all prontos corrected with this release are marked to be updated. Once all prontos are updated the build is being deleted from the release queue and set to RELASED FOR QUICKTEST (Central Build) or RELEASED (SC build). The release note is being generated from the assigned release note template (see also chap. 3.12.7).
The release note is being sent to the recipients defined in the release note template and stored as build attachment.

3.3.7 Build Attributes

BuildsController#show

As the Builds area is the central and most used area of the WFT also the concerning table builds is the central and most used one.
In the following the currently used (out of the various) attributes describing a build are explained:

ancesrty longtext
Lists separated by a "/" all id's of the predecessor baselines.
This column is used by the “ancestry” gem for hierarchical relations. The ancestry gem will replace the acts_as_tree gem using the based_on_id column.

auto_release boolean
The auto_release flag (see also chap. 3.3.6.10) is used to indicate that this baseline can automatically be released once it switches to state "succeeded".

based_on_id integer
The based_on_id contains the id of the parent build. This field is used to create a parent/children relationship between the baselines. The plugin acts_as_tree is used to handle this relationship, more information can be found here https://github.com/rails/acts_as_tree.

baseline string
The baseline is a unique field which is the builds name used inside eNB project.

branch_id integer
References this baseline to a branch.

build_id string
In the past there were different names used for the (internal) baseline title of a Central Build and the official title of the same eNB build. For e.g. baseline LN1.0_LTE_22_10_05 the build_id was set to LN1.0 22.10-5.
With the current baseline format the build_id is no longer used and it is identical to the baseline format, e.g. LN2.0_ENB_1009_001_00.

category integer
This field defines the type of the baseline and is one of 1, 2 or 3 (1 = Central build, 2 = System Component build, 3 = External build).
The category has impacts on the possible build state transitions.

check_before_freeze boolean
This flag defines if the build switches automatically from state buildable to frozen (false) or manual interaction is necessary (true).

config_spec_id integer
References to a build configuration template (for details refer to chap. 3.13.2).

created_at date
Automatic filled field which stores information about when this build entry was created.

created_by string
Stores the ID of the user which created this build

custom_config_spec text
When switching to state frozen the user can edit the generated build configuration (see also chap. 3.13.2). This edited and currently used configuration is stored in this field.

deliverer_id integer
References to the deliverer of that baseline.

delivery_date date
Field for the real delivery date for external baselines.

faults_cached_at date
Date where the faults of that build had been cached the last time (mainly an indicator that faults have been cached).

flags_mask integer
Contains a binary mask for several flags which can be set/unset per build. Some of them are used for Fast-Track mechanism (see chap. 3.17.8), or to mark Quicktest Phase 1 and Phase 2 as started (see #12561).

freezing_date date
seems to be no longer used (was used by ECL Sack Base) probably to be deleted by a DB correction.

freezing_reminder_date date
freezing reminder date is the date 7 days before the freezing date.
Seems to be no longer used (was used by ECL Sack Base) probably to be deleted by a DB correction.

important_note longtext
Field for the important note information displayed in release mail and release note (see also chap. 3.3.6.5.3).

important_notes_qt text
An additional important note which can be entered during releasing from Quicktest Phase 1 module.

important_notes_qt_author string
The author from field import_notes_qt.

important_notes_qt_created_at date
The creation date from field import_notes_qt.

important_notes_qt2 text
An additional important note which can be entered during releasing from Quicktest Phase 2 module.

important_notes_qt2_author string
The author from field import_notes_qt2.

important_notes_qt2_created_at date
The creation date from field import_notes_qt2.

note longtext
Used by Build Team for writing notes.

notified boolean
This field is still referenced in the code. Before deleting this column it needs to be checked if column is really necessary. Was used for indicating that a warning/information mail was sent in some specific state.

patch longtext
Field for patches inserted into the build configuration. This field is inherited when incrementing a build (see also chap. 3.3.6.4).

planned_delivery_date date
Field for the expected delivery date for external baselines.

planned_prod_date date
Field for the planned production date for Central and SC builds.

prod_date date
Field for the real production date for Central and SC builds.

purpose longtext
Used by Load Coordination within Load Planning area for writing notes.

quick_test_template_id integer
References to a Quicktest Template (for details see chap. 3.8.1.1).

quicktest2_template_id integer
References to a Quicktest2 Template (for details see chap. 3.8.2.1).

release_note_template_id integer
References to a Release Note Template (for details see chap. 3.12.7).

repository_branch string
Tag out of the XML release note for specification of the repository path (see #8254).

repository_revision string
Tag out of the XML release note for specification of the repository path (see #8254).

repository_url string
Field for the subversion repository information. This field is filled by Central Build via API (see chap. 4.2.2) after the build has succeeded.

responsible_id integer
References to the build responsible.
In combination with column responsible_type. This is a polymorphic association to either a user or a group.

responsible_type string
Indicates whether the build responsible is a single user (User) or if it is an user group (Group). Details concerning permission handling and user groups can be found within chap. 3.13.6.

scm_tool string
#8941
The content of that flag is used for aligning build scripts. For older builds this flag is set to "legacy".

state string
Contains the build state information. Please refer to chap. 3.3.8 or more information.

updated_at date
This column is automatically updated when the build entry is updated in the database. It shows the last modification time of the build entry.

xml_releasenote_id integer
References to an XML Releasenote Template (for details see chap. 3.12.8).

3.3.8 Build States

Model: Build

There are several build states available. Each build has always one specific state assigned. The transitions between these states are well defined. For handling the states and state transitions the plug-in http://agilewebdevelopment.com/plugins/acts_as_state_machine is used.
Some of the state transitions are done automatically based on some conditions, some state transitions are done by task interactions and some transitions have to be done manually.
For manual state transitions the user has to use the "Switch State" dialog displayed on each builds detail page. Within the "Switch State" dialog all possible state transitions depending on the current state are displayed.

Following build states are implemented:

(1) Creating
This build state is used for planning new builds, that shouldn't be visible to other users (not used at the moment).

(2) Planned
Builds that are officially planned and visible to other users. A build in this state can't no more be changed.

(3) Announced
As soon as the build has entered this state, the freezing reminder could be sent and handovers are possible.
(the possibility to send a freezing reminder is up to now not used)

(4) Buildable
When all handovers are performed the build will switch to this state and the build responsible will be informed.
(No notification will be sent.)

(5) Frozen
In case of the flag "Confirm Build Config" is not set the build will automatically switch from buildable to frozen once the freezing date is reached. Otherwise this has to be done by the build responsible.
As soon as the build is frozen the configuration files for the build configuration are provided and the build configuration is invoked.
In case of the build responsible notifies that something was forgotten it is possible to unfreeze the build and to return to state "Buildable".

(6) Building
The transition into that state can be started automatically or manually and will be set by some task which is configured to switch the state of the build.

(7) Succeeded
Changing to this state can only be carried out by some task which is configured to switch the build to this state.
The build responsible gets notified about this event.
In case of the build responsible notifies that something was forgotten it is possible to return to state "Buildable".
It is also possible to restart the build (switch to state "Building").

(8) Released for Quicktest
Quicktest Team gets informed that the CENTRAL build is released for Quicktest Phase 1.
A release note will be sent to Quicktest Team.

(9) Released
An EXTERNAL build will be automatically set to Released after

entering into WFT.
An SC build reaches this state after it has been successfully finished. A release note will be sent.
A CENTRAL build has successfully been tested and released by Quicktest Team. A release note will be sent.

(10) Not Released
The CENTRAL build failed during Quicktest Phase 1 or Phase 2.
A release note will be sent.
An SC build can be switched to that state.

(11) Released with Restrictions
The CENTRAL build is partly successful tested and is released by Quicktest Team.
A release note will be sent.

(12) Canceled
Builds can only be deleted if they are in the state "Creating", later they can only be canceled.
A build which has not yet reached the state "Building" (all builds coming from states (2), (3), (4), (5) and (13)) can be canceled.
A notification will be sent.

(13) Blocked
A build gets automatically into this state if the freezing date is reached and handovers are missing.
The build responsible is notified and he can decide to postpone or to re-announce this build.

(14) Error
If the build configuration or Hudson reports an error, the build will change to this state and the build responsible is notified.
The build responsible can
a) restart the build (go to state "Building"; e.g. the VCS was not reachable)
b) change the configuration and restart the build (go to state "Buildable").
c) set the build to "Succeeded" (in case of e.g. a build script has stopped).

(15) Build Failed
If a build ran on an error, the build responsible can mark this build as failed.
A notification will be sent.

(16) Testing
Testing of a system component sub-component. Depending on the test result state can be changed to "Released" or "Test Failed".

(17) Test Failed
If testing of a system component sub-component failed this component can be marked as failed.

(18) Pre-released
Interim state for handling the "Fast-Track" mechanism of PS_REL (for details see chap. 3.17.8)

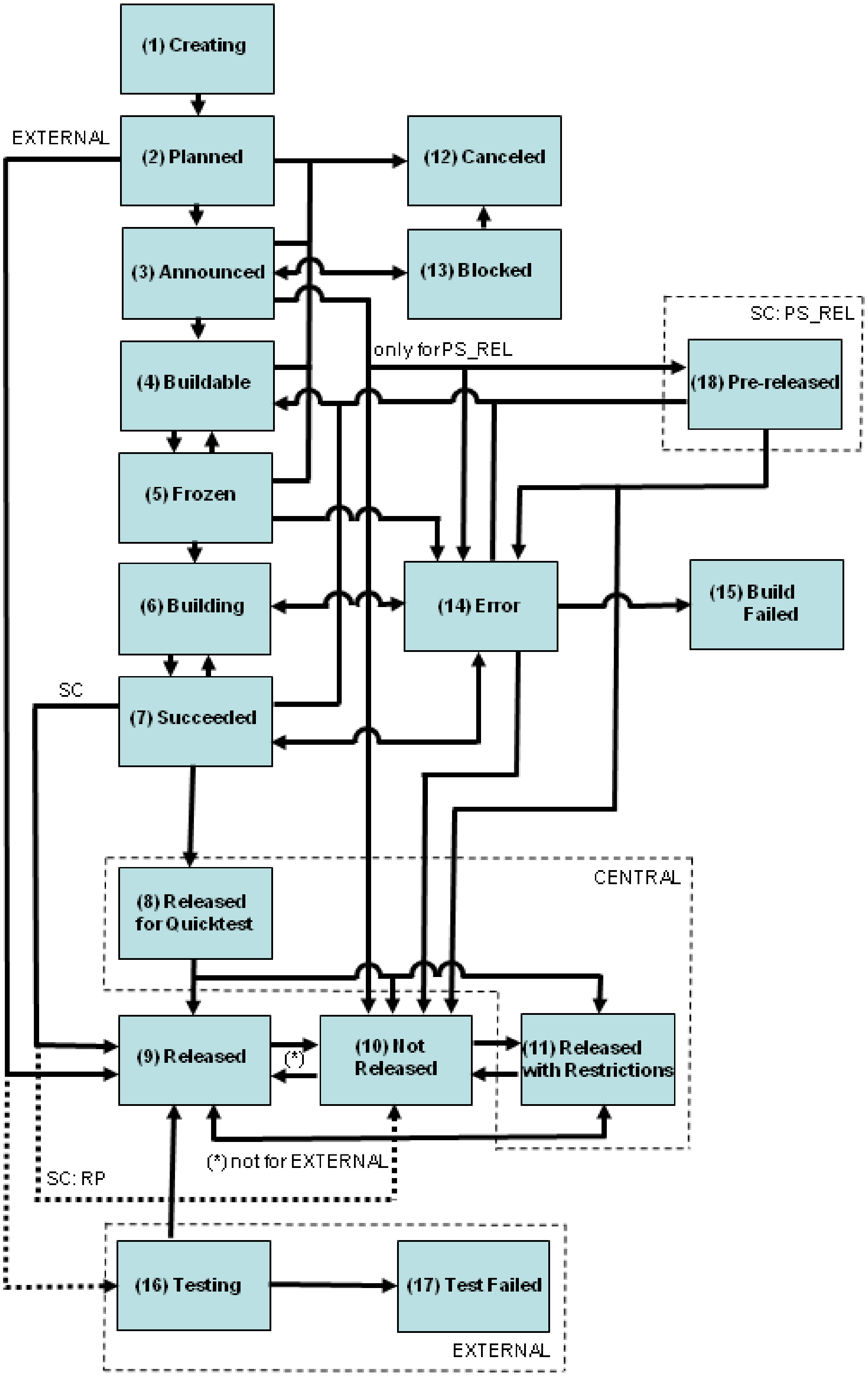


Figure 4: Build Process states

3.3.9 Switch State of a Baseline

BuildsController#switch_state

Depending on the current state of a build and in some cases on the category of the build the user can switch the state. Possible transactions can be seen within Figure 4.

In emergency cases a "hard reset" of a build can be done (see #22) This is possible only for builds in state Creating, Planned, Announced, Buildable, Frozen, Building, Succeeded, Canceled, Blocked and Error.
With this "hard reset" the status is set to Planned, the production date (will be set after switch to Succeeded) is removed, the build configuration is deleted and all tasks are set to state New (settings within the tasks are not changed).
A Reason for that reset has to be entered by the user and a mail with this reason and the username is sent to SW Build Team.

3.3.10 Create a new baseline

BaselineController#new

Creating new baselines can be done from scratch or by incrementing an existing baseline.

3.3.10.1 Create a new baseline from existing one

Creating new baselines is mostly be done by incrementing (using the "Increment" dialog) an existing baseline. This has the advantage that most of the information belonging to that build is taken from the previous baseline. Another advantage is that the parent/children relationship between the baselines is automatically assigned.
Using this increment function the user gets a form displayed which is already populated with all necessary information. The baseline number is being incremented, almost all other information (e.g. branch, production date, used templates, contained baselines, etc.) is taken from the previous baseline.
This data has to be adapted by the user to the needed entries.
After submitting the form some checks (see chap. 3.3.12) are performed to ensure valid baseline information. If the information entered was correct the baseline with all subsequent information is stored within the database.

Remark (see also #10234): for central builds the user gets a warning as soon as he changes the already suggested baseline name, that this will also effects the naming of the next "autobuilded" baseline (for details concerning the automated building refer to chap. 3.17.4).

3.3.10.2 Create a new baseline from scratch

Creating a baseline from scratch is done using the same form as creating a baseline from existing one. The difference is that no form fields are populated. The user needs to know how to fill all information like templates, contained baselines, etc.

3.3.10.3 Formula to create a new baseline

All basic settings of the new baseline have to be set here. These setting can be edit after the build is created and as long as it is not frozen (for details see chap. 3.3.6.1).

3.3.10.3.1 Basic

The category, the name, the based on baseline, the concerned branch and the repository URL of the new baseline are entered here.

3.3.10.3.2 Date

The relevant production dates have to be entered within that tab.

3.3.10.3.3 Templates

The used Config Spec template and the used HTML Release Note template have to be selected here.
Some additional setting (Confirmation of Build Config File before switching to state FROZEN; Information for SCM Tool; automatic releasing the build as soon as in state SUCCEEDED) also have to be done within that tab.

3.3.10.3.4 Baselines

The needed baselines can be select within here.

3.3.10.4 Create an eNB using newer baselines

BaselineController#newer

(see #9173)
The link "Create eNB using newer baselines" within the sidebar leads to a formula where the user can select a branch or/and an eNB build. In case of no eNB was selected the newest build of that branch is used.
Based on these entries a list with newer baselines belonging to that build is displayed. Out of this list the user can select baselines to create a new build.
After pushing button "Create a new baseline" links to the formula for creating an incremental build (see chap. 3.3.10.1).

3.3.11 Baseline Diff

BuildsController#baseline_diff

Three possibilities for comparing builds related to the used baselines or the content of the build exist: "Compare Builds", "Diff to Parent" and "Content Diff".

3.3.11.1 Compare Builds

The "Compare Builds" function can be used to show the different contained baselines between two or more builds. The baselines which have to be compared need to be chosen before the "Compare Builds" page is opened. This can be achieved by filtering the needed range baselines out of the Builds Overview Table (see chap. 3.3.4) and using the "Compare/add" Icons within this table.

3.3.11.2 Diff to Parent

The "Diff to Parent" function works exactly as the "Compare Builds" function. But it only shows the difference between the current and the parent baseline. This function can be accessed via the menu bar.

3.3.11.3 Content Diff

This function can be accessed via the menu bar. It shows the difference concerning the complete content (features, faults, important notes, etc.) between the currently chosen baseline and one of the previous baselines, which has to be selected out of an offered selection list.

3.3.12 Build Checks

The building of a load is supervised by several checks. For SCs the XML Releasenotes and the existence of prontos named within the XML Releasenote are checked, for eNB

loads additional checks concerning the "SC Compatibility", "ECL", Build Configuration, "BranchFor" and "Correction Allowed" are available.
The results of the "SC Compatibility", "ECL", "BranchFor" and "Correction Allowed" checks are generated dynamically every time the page is opened and are displayed within the "Warnings" tab of the build. These checks are done only and the "Warnings" tab is visible only for users who have the needed permissions for the concerned build.

3.3.12.1 Release Check

Build#release_check

The content of the XML Releasenote (see chap. 4.2.3) of the concerned build is checked for various items against the content of the build.

- the "Releasenote Count" indicates how many XML files are attached to the build (e.g. in case the XML file was deleted the check fails and the count is set to 0).
- the "Releasenote File Check" indicates if the XML file is stored on the file share (/lteRel/build/…).
- the "Releasenote Read Check" checks if there is a read protection for that file.
- the "Releasenote Parsing Check" checks if the XML Releasenote could be parsed.
- the" Releasenote Validation Check" checks if the XML file is valid.
- the "Attribute Check" checks whether the attributes for "baseline" and "basedOn" are consistent with the content of the build.
- the "Features Check" checks the correlation between the XML Releasenote and the build content concerning the contained features (e.g. in case a feature could not been taken over from XML Releasenote).
- the "Unsupported Features Check" checks the correlation between the XML Releasenote and the build content concerning the contained unsupported features.
- the "Faults Check" compares the content of the fault section within the XML Release Note against the stored faults of the build.
  1. Remark: in case of a fault (indicated within the XML Releasenote) does not exists within the DB this one will be ignored, i.e. will not be imported to the build.
  2. Remark: in case of the fault is described within the XML Releasenote with the prefix "PR" but the fault is stored without that prefix within the Pronto DB the import is done anyhow, but without the prefix and the check indicates a failure.
- the "Faults Title Check" verifies if all faults have a title assigned.

3.3.12.2 SC Compatibility Check

There are several SC Compatibilities predefined. The definition is done by Load Coordination (see /6/) and is additionally described within /3/ (section "Note for "baselines" section:").

If an SC A (e.g. PHY_TX, MAC) is based on another SC B (sub-component of SC A; e.g. PS_REL, LOM) or an ECL_ SACK_BASE, the SC A must list the relevant baseline version of SC B within its baseline section.
Or in other words:
A certain baseline (SC B, the SC which should be compatible) has to be used by (or "must be compatible to") a predefined group of specific baselines. Therefore several deliverers are grouped (e.g. deliverer group PHY_RX contains all PHY_RX deliverers).

E.g. "LOM compatibility" means that PHY_TX, PHY_RX_COMMON, TUP, MAC, MAC_DATA and MAC_PS_ have to use ("must be compatible to") the same LOM as selected with the concerned eNB build.
(Or in other words: for an eNB that has contained e.g. a certain PHY_RX the definition of the concerned deliverer group ensures that also a certain LOM is contained.)

To enable this check for a specific SC A a relevant entry has to be defined by WFT administration within the table "deliverer_group_chains".
Remark: the attribute "deliverer_group_id" contains the deliverer group ID of the deliverer group of SC A, whereas "required_deliverer_group_id" contains the deliverer group ID of (resp. SC B; sub-component of SC A) to which the SC B has to be "compatible").

3.3.12.3 ECL Check

Build#ecl_check

This check can be performed for Central Builds only. It describes a test to verify that all planned baselines of an eNB build suite together. WFT applies the ECL-Check in the following areas: Autobuild, Load Planning and the "ECL Check" task during production (for details also see /7/).
Per SC of a Central Build (eNB load) the ECL Check checks if the ECL of the checked SC fits for the Central Build. In other words, it checks if the ECL of the eNB load is equal to one of the ECLs out of the "ECL Compatibility" list (see also chap. 3.3.6.6.2) of the checked SC.
In case the check fails a warning for that particular SC will be displayed within the "Warnings" tab of the eNB load. Depending on the task settings a mail notification ("Build Preliminary Information" mail) is sent to the build team.
E.g. CentralBuild_1 uses ECL_1 and MAC_1, but MAC_1 is compiled against ECL_2. Therefore in CentralBuild_1 an ECL warning is displayed, that MAC uses a different ECL.
The ECL-Check can be regarded as a special SC Compatibility Check. Whether this check has to be done or not is defined for each SC by Load Coordination (see /7/). It is enabled for the concerned deliverer by checking the flag "Check for matching ECL" (area Administration, sub-area Deliverers).

3.3.12.4 Build Config Check

Build#config_warnings

The Build Config Check looks if for the baselines entered within the "Baselines" tab an entry within the associated Config Spec exists.
The mismatches are indicated with "…was not found in build config template".

3.3.12.5 BranchFor Check

Build#branch_for_warnings

The BranchFor Check checks if the newly added sub-baselines to an eNB load have the same branch named within the BranchFor tag (of the concerned sub-baselines XML Releasenotes) as the branch assigned to the eNB load.
In case of a newly added sub-baseline has not listed a branch equal to the eNB loads branch the particular SC will be displayed within the "Warnings" tab of the eNB load and a mail notification is sent to the build team.
Depending on task settings (see chap. 3.3.6.3.8.9) the build may proceed in error case (this can vary from branch to branch).

3.3.12.6 Correction Allowed Check

Build#top_flag_warnings

As soon as an SCs wants to deliver to a restricted branch, indicated in "branchFor", the Correction Allowed Check checks whether all corrections (or at least one of the attached PRs) have the required TOP flag.
For details concerning the functionality "Correction Policy", refer to chap. 3.17.5.

If the check fails, that means if at least one correction does not have the required TOP flag, this will be indicated within the "Warnings" tab of the eNB load and a mail to build team will be sent.
The Check is realized as a task (see chap. 3.3.6.3.8.11) which can be skipped if needed.

3.3.12.7 Pronto Existence Check

After an SC has released a new build via XML import, it is checked if the Pronto ids listed in section correctedFaults really exist within the Pronto database. The not existing Prontos are listed in a mail to the SC releasing responsible (according to the used access-key), to the mail address out of the XML and to the load coordination.
As sometimes also very old Prontos are listed within the XML Release Notes, the search is extended to the Pronto archive too.

3.4 Area Faults

Within that area the user:

- can search for specific faults and gets detailed information about it.
- gets on a daily basis detailed information about the corrected faults within each build and about possible differences to the Pronto tool (link "Pronto Report").
- gets all those faults listed which have been corrected between two builds (link "Pronto Diff"). The faults corrected within the "from" build are excluded.
- can view the list of the "Pronto Group Mapping"

3.4.1 Fault View

FaultsController#fault

Clicking on a fault name where every possible (e.g. within the chronological or the weekly build view) or after the search for a specific fault the user gets a detailed view of that fault.
The information concerning the fault is cached within the database and can be updated by the user with a "Refresh" link.
Information within which build(s) that pronto is corrected is added.
Additionally a link to the pronto tool is available.

3.4.2 Pronto Report

FaultsController#pronto_start

On a daily basis the Pronto Report displays all builds produced on that day, with all corrected faults and with the correction target.
This page is used to find faults which correction target needs to be updated within the pronto database.

3.4.3 Pronto Diff

FaultsController#pronto_diff

The Pronto Diff can be used to show pronto corrections from sequent builds (that means the builds have to be out of the same branch and must have an uninterrupted hierarchy or they must have a common predecessor).
The user has to enter the start and end baseline so that all corrected faults from the start up to the end baseline are collected and displayed.
The faults corrected within the start baseline are excluded from the list.

It is possible to define a filter for specified author groups (see chap. 3.12.2.2, parameter "PRONTO_AUTHOR_GROUPS") to exclude some prontos for specific projects.
The link "Show/Hide list" shows all defined filters. To activate the filter the checkbox has to be checked.

3.4.4 Pronto Group Mapping

FaultsController#pronto

(see #7171)
This view lists a table, which shows the mapping between the Pronto Group and the System Component.
The table is imported once a day out of https://sharenet-ims.inside.nsn.com/Open/D469389660.

3.5 Area Features

Within this area the user can search for features.

3.6 Area Branches

3.6.1 Overview of active Branches

BranchesController#overview

Depending on the selected project the Overview of active Branches shows a table with all active branches of that project and various information about them.
Within this view the content of "Correction Policy" (see chap. 3.17.5.) is shown.
The user gets an overview of the used branch delivery types within the selected project and per branch information about:

- Branch name, used eNB template, and branch delivery type.
- The Project, the Last Development Period, Release and System Release.
- The last build baseline is shown.
- The next planned ECL_SACK_BASE and the deadline for SC deliveries.
- The Correction Policy for that branch.
- A link to the ECL SVN repository.
- The information whether the branch is "writable", means can be used by Build Teams.

3.6.2 Build selection per Branch

BuildsController#branch

Here the user can select a certain branch out of the active branches and gets a list of all contained builds from where he can select one to get detailed information about it (for more details see chap. 3.3.6).

3.6.3 Overview of Trunk baselines

Here the user can select a certain branch out of the branches for trunk baselines and gets a list of all contained trunk builds from where he can select one to get detailed information about it (for more details see chap. 3.3.6).

3.7 Area Knives

KnifeRequestsController#index

Within this area every user can request a knife, finds a list of the past knife builds and can search for knife builds. The search criteria for a specific knife build is the underlying eNB base line or the Pronto ID which leads to the knife.
Additionally the user can select the knives done at a specific date, or to those done by himself and he can have a look to the statistic of the past knife builds.
With #9310 the Knife Building interface has been simplified (less states) and made more user friendly (step by step handling with consistency checks; additional descriptions and information).

3.7.1 General about Knife Building

A general concept about Knife Building and a step-by-step guide for knife production can be found here: https://bts.inside.nsn.com/twiki/bin/view/PreIntegration/PO_Knife Handling.

The usage of the WFT interface is explained within the WFT Wiki:
https://wft.inside.nsn.com/wiki/knife_request
Information how to create a dummy knife can be found here:
https://wft.inside.nsn.com/wiki/api_description#header_7.

Specifics how to create/collect the artifacts needed for a knife; for the knife.zip; or the .config file can be found for

- WMP / DCM macro / TDD: https://confluence.inside.nsn.com/display/BTSSCMLTE/Knife+Manual
- uREC: https://confluence.inside.nsn.com/display/LDC/microREC+Knife
- FDD and TDD trunk knives: https://confluence.inside.nsn.com/display/BTSSCMLTE/Trunk+knives

3.7.1.1 Trunk Knife

(see #7174)
Beneath knives for Maintenance branches also knives for Trunk can be requested and built.
The baseline of the requested knife has to exist in subversion (and must not be available within WFT DB). The existence is checked by WFT in 4 different repositories (see /lib/subversion.rb).
WFT fetches and parses the .config from subversion and displays a list of available change options within Tab "Knife configuration" (/app/models/knife_request.rb). Then the .config changes are stored in column "config" in table "knife_requests. The changes of .config are provided in ext/knife/20993/info.
The version number field contains 90..99 for Maintenance Knifes and A..Z for Trunk Knifes.
A list of all trunk baselines (and not only to those which are stored within WFT) is cached (table "data_caches") because otherwise the interface for creating knives would be to slow.
The default expire value for that cache is 5 minutes (after this time the get method will return nil although the value is available in database).
Using the force flag for the get method will return content from database although cache is expired (e.g. when SVN isn't reachable).

For Trunk knives the user has to add explicitly those SCs which he wants to be changed. For information about which SCs are contained within the based build a link to the concerning .config, displaying the builds content, is offered.

3.7.1.2 Dummy Knife

(see #5919)
A Dummy Knife is no real knife, it is only for documentation. The knife type is "report-only". For this kind of knives only meta data will be handled and no real knife will be created.
Intention for this type of knife is to use the statistic module for knives which cannot be produced via WFT.
The PR number can be modified later on, the PR field is mandatory and takes PR number or feature number or "n.a.".

3.7.2 Knife States

Knives have a state attribute which defines the current status of the knife. Following knife build states are implemented:

(1) New
This is the initial knife state after creating a new knife by a user.
The knife is automatically prepared and started by a script.
A notification will be sent to the build team.

(2) Running
The knife is currently being built.
A notification will be sent to the knife requestor.

(3) Ready
The knife has been successfully built.
A notification indicating state "ready" will be sent to the requestor, the build team and the tester of the knife.

(4) Error
The knife build has failed.
A notification indicating state "failed" will be sent to the requestor, the build team and the tester of the knife.

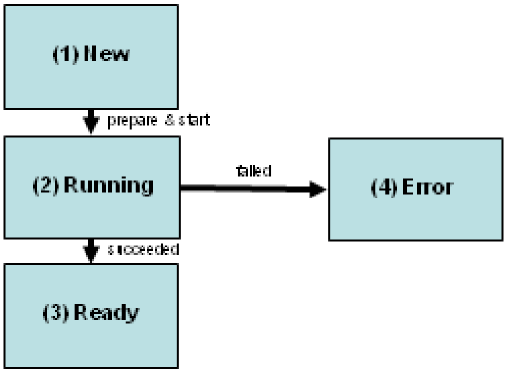


Figure 5: Knife states

3.7.3 Request a Knife

To request a knife the user needs to enter some information in the "Create a new request" form.
There exist 5 tabs where data can be entered:

1. Type of Request
2. Options for Maintenance branch knives
3. Knife configuration
4. Expected results
5. Additional information

3.7.3.1 Type of Request

The knife requestor can request a new knife from a chosen baseline (if that knife has to be a Maintenance or Trunk knife will be fixed with the selected baseline the knife is based on), or can copy a knife configuration from an already existing knife (entering the knife_ID), or can create a dummy knife.

3.7.3.2 Options for Maintenance branch knives

Within the second tab the user has to select the needed HW module(s) (FSMr2 / FSMr3).

3.7.3.3 Knife configuration

The third tab offers some knife configuration setting.

For maintenance knives as also for trunk knives the user can either specify the knife-dir or the SCs which have to be changed (one of them must be filled out; see also #10120). Therefore the .config file has to be parsed for the changed baselines.
For an incremental building the user can decide if he provides the changed baselines via the .config file or he provides the files within the Knife Dir.

Knife Dir
The requestor has to upload the knife sources to a FTP server and gives here the directory where the files are stored.
Hint: it is checked if the named file(s) also exist within the SVN repository. This prevents that a knife build could not be built correctly in case of a wrong file name is given (e.g. in case of a typo) and so this file was not been taken.

Verify knife.zip content
In case of this checkbox is enabled, all files in knife.zip must exist in original repository, otherwise knife will fail.
In case of this checkbox is disabled new files (which do not exist in original repository) have to be added to knife workspace.

Patch version
This patch version number is needed for knife builds based on an eNB productions until 13th, May 2014 and shows the last two digits in a knife build label. Per default "99" is used. To force a following knife download a different number (98 to 90) can be used.
For knife builds based on eNB baselines produced after 14th, May 2014 this patch version number is eliminated and is replaced by the knife_id.

E.g. for a knife build based on build LNT4.0_ENB_1311_536_00 (produced **before** 14th, May 2014) the knife label will be: LNT4.0_ENB_1311_536_**99**, but
for a knife build based on build LNT5.0_ENB_1406_910_81 (produced **after** 14th, May 2014) the knife label will be: LNT5.0_ENB_1406_910_81_**54321**.

Server
The server where the knife build has to be done.

In general a knife build is a full rebuild of the eNB load on which the knife build is based on (depending on the used knife building scripts of the build teams). With the configuration setting (see also chap. 3.12.1) "KNIFE_REBUILD" the WFT administration is able to enable the selective rebuild either per project, per eNB load or per branch (the relevant parameter has to be entered into field "Reference").
Selective rebuild means, the user is able to select one or more SC's which have to be rebuilt.
A full rebuild lasts longer, a selective rebuild is faster.

Rebuild Settings
The user can select if he wants to have a full rebuild (will take longer) or he can select the changed SC's.

Changed Components
Out of the given list the user selects the changed SC's, which have to be rebuilt.

3.7.3.4 Expected results

The user can chose one or all of following expected results:

Standard BTS package (BTSSM)
FEP files for DCM, puttibatti and BTSSM downloadable for WMP or uREC eNB package.

DSP BIN files
All *.BIN files packed into dspbin.tgz

DSP MAP and OUT files
All *.map and *.out files packed into <eNB_base>_release_mapAndOutFiles.zip

MCU Application Not stripped
Files BTSOMexe, CELLCexe, ENBCexe, LOMexe, RROMexe, TUPcexe, UECexe, UMFSPexe and BSTATexe packed into <eNB_base>_release_mcuApplicationsNot Stripped.zip.

MCU Application
Files BTSOMexe, CELLCexe, ENBCexe, LOMexe, RROMexe, TUPcexe, UECexe, UMFSPexe and BSTATexe packed into apps.tar.gz.

Test packages
For WMP knives only: test packages T1/T2.

3.7.3.5 Additional information

Reference (Fault or Feature)
The user has to enter a Fault No. or a Feature No. (as soon as a sub-string is entered within this field the user gets a list out of which he can choose the desired fault or feature).

No Reference
Only in the exceptional case, if there is no reference available, this checkbox has to be checked.

Purpose
For statistical purpose the user has to set out of a select-box the category of the knife to: "debug", "correction" or "other" (see #8703).

Knife Info
Here the user can enter additional information for the Build Team as free text.

Recipients List
All mail recipients which should be informed via mail about the knife result have to be entered by the user.
Hint: for adding a new recipient the user has simply to enter a sub-string of the desired user-name to get a list of possible recipients where he can select the needed one.

Knife Tester
The person, if already known, who should test the knife, can be named. The Knife Tester gets a note that the knife is ready.
Later on a tester can additional be added within the "Knife Details" (see #11615).
Hint: Within the sidebar a link is offered showing all knives assigned to the calling user (see #12705).

At least, before the request will be created a "Preview for Knife Request" is displayed. This enables the user to check all his entries before sending the request.
In case of there are errors within the entered data no knife build creation will be offered.
After knife build has been created, the request mail will be sent to the Build Team and the knife build will be started automatically (in case of simple knives).

3.7.4 Knife - Task - Build Interactions - How is the knife executed - Knife API

Once a knife is created it is automatically started by a special mechanism. Each knife has a central build assigned which was selected by the requestor.
After saving the knife to the database a trigger is called which starts a new job in the Jenkins running on the server selected for building this knife. All further notifications, e.g. inform build team on knife failure, is done by the executed script.

3.7.5 Knife results

KnifeRequestsController#index

The main page of the area Knife shows a list of Knife Requests on a daily basis.
Usually the latest knives of the current day are displayed. Using a calendar every other specific day can be chosen.
Additional knife views are: "knives requested by me", "knives assigned to me as tester" or "unfinished knives".

Details of every knife can be shown using the link "Details" (same as on clicking on the Knife ID No.).
The knife Details show all relevant information per knife (the data which is entered with the request) as also the duration of the knife building (see #8639).
Depending on the access rights the user additionally is able to edit some of these details, can assign himself to the "informed list" or can assign himself as knife tester.

3.7.6 Knife Configuration

KnifeRequestsController#matrix

Using the link "Configuration Matrix" within the Sidebar (see #10839) the user gets a complete overview about the current knife building configuration.

In detail: this configuration matrix shows for which project it is possible to build which kind of knives (Maintenance or Trunk) on which servers and which Build Team is responsible for the building.

3.7.7 Knife Statistic

KnifeRequestsController#stat
StatisticsController#knives

There exist various possibilities to show statistics for knife builds.
First of all a graphical view shows the amount of knife requests per day (link "Knives/Day") over the last 30 days.
More details are shown using the link "Statistics" (see chap. 3.9.6 and chap. 3.9.7).
Parameters which can be chosen are: date range, release, fault ID, feature, test team, development team, etc.

3.8 Area Test Area

QuicktestsController#index

This area is used by the Quicktest Teams to release a specific eNB build to I&V after testing it within "Phase 1" and within "Phase 2".
For using this area permission has to be granted by the WFT administration.

3.8.1 Quicktest Phase 1

QuicktestsController#index

This page is used by the Quicktest Teams to release a specific eNB build to I&V after testing it within "Phase 1". It shows all builds performed within the last two weeks ordered by day. The overview table shows all relevant information necessary for the Quicktest Teams.

The builds can have one of the following states:

• Error:	Build is in state error
• Succeeded:	Build has been succeeded
• Released for quicktest	Build has been released for quick-test
• Released:	Build has been released to I&V
• Not Released:	Build is not released
• Released with restrictions:	Build is released to I&V with restrictions

Using the search bar on top of the page helps to find older builds, which are no longer listed in the table.

Within the overview table the Quicktest Teams have the possibility to set some information flags (currently only the flag "Released to PMC" is realized).

The column "Default Template" shows the used quicktest template (see chap. 3.8.1.1).
In case of the Quicktest Team has used the API for releasing no template is set.

To release a specific eNB build to I&V or to send an update of the release note the user (Quicktest Team) has to use the relevant link displayed next to the baseline name (for details refer to chap. 3.8.1.2 and chap. 3.8.1.3).

3.8.1.1 Quicktest Templates

QuicktestTemplatesController#index

The Quicktest Teams can define release note templates based on their own needs. Using the link "Manage Templates" the members of the Quicktest Teams are able to create and/or to edit the quicktest release note templates. These templates contain information regarding the test execution done by Quicktest Team, links, as also information for sending an email (like mail subject, mail content, mail sender, mail recipient and mail reply to).
The content to be entered can be mandatory or optional.

Each eNB build has a template assigned. It is being copied during incrementing the build. As soon as the release note was sent all information is stored together with the data of the eNB build and is available in case of resending the release note.

3.8.1.1.1 API specifics

In case the Quicktest Team uses the API for releasing no default template will be set, which is also displayed within the overview table within column "Default Template".

3.8.1.2 Send Releasenote

QuicktestsController#release

The first time a baseline is released by a Quicktest Team to I&V all release information is taken from the associated quicktest template. The user gets a form displayed and is able to edit step by step all fields on its own needs.

Within the first tab "1. Template" the user must select first the template which has to be used and the test result before all other tabs are offered.
For testing purpose a "Send Preview" link is offered which will set the recipients address to the users mail address and send the release note towards the user.

3.8.1.3 Send Releasenote Update

QuicktestsController#release

Sending an update of the release note works the same way as sending a release note for the first time. All information entered the first time (resp. for the latest sending) is already displayed in the form fields. The user can then perform the changes and resend the release note.

3.8.2 Quicktest Phase 2

Quicktest2sController#index

This page is used by the Quicktest Teams to release a specific eNB build to I&V after testing it within "Phase 2". It shows all builds performed within the last two weeks ordered by day. The overview table shows all relevant information necessary for the Quicktest Teams.

The Handling concerning management of the quicktest release note templates as also concerning the release process is identical to that of quicktest "Phase 1" (see chap. 3.8).

3.8.2.1 Quicktest Templates

Quicktest2TemplatesController#index

Identical handling like for quicktest "Phase 1" (see chap. 3.8.1.1).

3.8.2.2 Send Releasenote

Quicktest2sController#release

Identical handling like for quicktest "Phase 1" (see chap. 3.8.1.2).

3.8.2.3 Send Releasenote Update

Quicktest2sController#release

Identical handling like for quicktest "Phase 1" (see chap. 3.8.1.3).

3.9 Area Statistics

StatisticsController#index

Within that area several statistic views concerning Central Build results as also concerning Knife Build results can be found.

Following diagrams exist:

- an overview of the results of the selected central builds (pie).
- a weekly overview of the results of the selected central builds (the user can select various chart views).
- build times of the selected central builds.

Depending on the selected statistics the user can select various chart settings, e.g. the project (WMP, DCM, TDD, etc.), the branch and the time frame.

Following statistic tables exist:

- no. of days / builds since last released build.
- faults by group in charge
- Knife Statistic
- Knife Statistic by delivered corrections

3.9.1 Central Build Results (Pie)

StatisticsController#central_build_results_pie

Depending on the selected project, the selected branch and the time frame the user gets a chart showing the amount and the percentage of "release", "not released", "released with restrictions" and "canceled" eNB builds.

3.9.2 Central Build Results (weekly)

StatisticsController#central_build_results_weekly

Depending on the selected project, the selected branch and the time frame the user gets a chart on a weekly basis showing the amount of "release", "canceled" and "QT failed" eNB builds.
The appearance of the chart can be chosen.

3.9.3 Central Build Times

StatisticsController#central_build_times

Depending on the selected project and the selected branch the user gets a diagram showing the build times of each contained eNB build.

3.9.4 Since last working Build

StatisticsController#since_last_working_build

Depending on the selected project, this table gives an overview about some figures concerning the builds released by Quicktest to I&V.
Listed are the:

a) days gone since last working build.
b) no. of builds released in the meantime with restrictions.
c) no. of failed builds since the last working build.

The figures are listed by default for currently visible branches; additionally the table can be enhanced with the data of no more visible branches.

3.9.5 Faults by Group in Charge

StatisticsController#faults_by_gic

This table lists depending on the selected project for all currently visible branches the amount of not released (by Quicktest) builds in relation to the group(s) in charge.
As reason for a "not released" or a "released with restriction" build Quicktest adds the ProntoID. Based on this ID the group in charge is assigned.
Additionally the table can be enhanced with the data of no more visible branches.

3.9.6 Knife Statistic

StatisticsController#knives

Depending on the selected values for date range, release, fault ID, feature, test team, development team, etc. this statistic shows several data grouped and summarized concerning the affected knives.
On demand additionally a table with all details of the selected data can be shown. This table can be exported as *.xlsx file.

3.9.7 Knife Statistic by delivered corrections

StatisticsController#knives_per_fault

Depending on the input data (the selected values for date range, release, fault ID, feature, test team, development team, etc.) this statistic shows data concerning the affected prontos which have been resolved (within eNB loads) and the related knives (if applicable).
The table "Details" lists all prontos (delivered corrections) which have been resolved.
The number of knives shown within that table is the total amount (independent from the input data) of knives requested for these prontos.

3.10 Area Toolbox

ToolboxController#index

This area contains a collection of some tools supporting the user's daily work.

3.10.1 XML Validation

ToolboxController#xml_validate

The XML Validation checks performed via GUI are identical to the "xml_validate" method of the WFT API (see chap. 4.2.2) and those checks which are done during XML import (see also chap. 4.2.3.2; see #8503).
Beneath the validation of the XML schema (it is checked whether the XML file is conform to the XML schema version named within this file), following additional checks are done:
- if the baseline name and the deliverer fit
- if the user has the relevant permissions for importing
- if the basedOn is correct
- if branchFor lists a correct branch
- if only one item downloadItem exists within section download
- if the tags repositoryUrl, repositoryBranch and repositoryRevision are not empty (the correctness of the contend is not checked; see #10967)

The user will be informed about the failed reason (e.g. "invalid schema", "invalid permissions", etc.).

3.10.2 XML Import

ToolboxController#import

This formula can be used for doing an XML Release Note import (for more details see chap. 4.2.3.2).

3.10.3 FEP Search

ToolboxController#fep

With the (DCM specific) "FEP Search" the user is able to find the eNB baseline name belonging to the entered FEP Version number (format "0123").
The FEP Version is generated automatically right before DCM eNB build is started. The version number is a consecutive numbering and it is unique across all releases and builds.

3.10.4 Commit Counter

ToolboxController#xxx

(see #12201)
The Commit Counter counts all SVN commits done between two chosen baselines.

3.11 RSS Feeds - eNB Notification

NotificationController#index

Users have to use this area for subscription for RSS Feeds to get informed about a released build. The user can choose if he wants all releases, only "released for

Quicktest" releases or released releases for one of the implemented deliverers and for one out of the visible branches.

3.12 Area Administration

AdminController#index

This area is restricted to the WFT administration for administering the tool. The possible tasks are described in the following.

Some administrative tasks are externalized to area Management (see chap. 3.13) for special user groups. E.g. Load Coordinators who are responsible for defining and managing branches, releases and system releases. Or for Key-Users who are enabled to manage the permission group they are responsible for (for details see chap. 3.13.6).

3.12.1 API Description

ApiController#index

The description of relevant API methods (for details see chap. 4.2.2) and with this the existence of the link (button) "API DESCRIPTION" has to be added by the WFT Administration by creating a new "API Description".
Beneath the title of that "API Description" the route of the affected page as also an appropriated description has to be added.
The API description can afterwards be edited and deleted.
(Hint: first "API Description" has been done for "Builds / Assignments / ECL Compatibility / Edit Compatibility". More descriptions have to be added step by step.)

3.12.2 Configuration

ConfigurationController#index

For operating the WFT the administration can set needed working parameters (e.g. definition of Host address, definition of email-address sent out from WFT, deliverer pre-/suffix, etc.). These settings are used for global settings of the Workflow Tool installation.

3.12.2.1 Usage of configuration

A configuration parameter consists of a name, the settings of it and a description.
Settings for a configuration can be done globally, per project, per branch and per specific build.
The hierarchy of these settings is as follows: a build specific setting surpass a branch specific one, which itself surpass a project specific one and at least the global configuration setting will be used.
Hint: separation of different values of a setting depends from the usage within the code (e.g. can be a comma or a blank).

3.12.2.2 Configuration parameters

In the following the list of all currently used configuration parameters:

Name of Configuration	Global Conf. Setting	Description
ALLOWED_KNIFE_SERVER	Ulm, Hangzhou	Allowed servers for Maintenance Knife building

ALLOWED_TRUNK_KNIFE_SERVER	Ulm, Hangzhou, Beijing	Allowed servers for Trunk Knife building
AUTOBUILD_ACTIVE	1	Global switch (on/off) for autobuild functionality
AUTOBUILD_CC	andreas.hutstein@nsn....	Informed addresses (CC) of notes for autobuild
AUTOBUILD_TO	lte-sw-build-team@mli...	Recipients addresses (To) of notes for autobuild
BUILD_HARD_RESET_CC	hendrik.keller@nsn.co...	Informed addresses (CC) informing about a build's hard reset
BUILD_HARD_RESET_TO	lte-sw-build-team@mli...	Recipients addresses (To) informing about a build's hard reset
DELIVERER_PREFIX		Definition of deliverer prefix (regexp)
DELIVERER_SUFFIX	[0-9]{4}(_[A-Za-z0-9,...	Definition of deliverer suffix (regexp)
DISABLE_FAULT_CALCULATION	0	Switch for disabling calculation of corrected faults (in case of fault aggregation)
docs	{"DRIS Header"=>"IMS...	Current versions of imported documents (e.g. RDB, FBP, SPP, IVB, RPB, RPSAB) into LDR
FAST_TRACK	0	In order to activate the Fast-Track mechanism for eNBs, it must be set to 1
FBPlan	Version:1.151 ; uploa...	Latest uploaded LDR version
FBPLAN_IMPORT_STATE	Just Importing! new v...	Latest uploaded LDR version if import is finished, during import: the currently uploading version and the last uploaded version
GUEST_USER	Anonymous	A new user will get all action approval, deliverers and areas assigned from this group
KNIFE_BUILD_TEAM	lte-sw-build-team@mli...	Email address of concerned knife build team
KNIFE_ENABLED	0	Knife building can be disabled per project/branch by setting this value to 0
KNIFE_HIDE_MODULE_TAB	0	Disable module tab for specific projects/branches
KNIFE_MODULES		Available HW modules (e.g. FSMr2, etc.)
KNIFE_MODULE_DEFAULT		The default selected module in knife area
KNIFE_REBUILD	1	Rebuilding of selected components possible
KNIFE_REQUESTS_MESSAGE	if Knife is used for ...	Global message displayed on knife area
KNIFE_REQUEST_SHOW_DCM_OPTION	0	If set the check-box "Dcm Knife" will be visible (see #10987)
KNIFE_REQUEST_SHOW_FSM_COMMENT	0	Show some special text for module build selection
KNIFE_SERVER	Ulm	By default selected knife server for maintenance knives
KNIFE_SERVER_BEIJI	knife_wft	Jenkins project to be used on

NG		Beijing server
KNIFE_SERVER_ULM	Knife	Jenkins project to be used on Ulm server
KNIFE_TRUNK_SERVER	Ulm	By default selected knife server for trunk knives
KNIFE_TRUNK_SERVER_BEIJING	Knife	Jenkins project to be used on Beijing server for trunk
KNIFE_TRUNK_SERVER_ULM	Knife	Jenkins project to be used on Ulm server for trunk
LDAP_IGNORE_USER	lteman tdlteman root ...	LDAP Update will ignore the listed users during automated cleanup
MAIL_FROM	Workflow Tool <norepl...	Default sending address
PLANNING_BUILD_CC	hendrik.keller@nsn.co...	Informed addresses (CC) of notification about builds started from area Load Planning
PLANNING_BUILD_TO	lte-sw-build-team@mli...	Recipients addresses (To) of notification about builds started from area Load Planning
PRONTO_AUTHOR_GROUPS		Filter for specific projects to exclude some prontos (see also chap. 3.4.3)
PRONTO_FILTER	0	Switch for activating PRONTO_AUTHOR_GROUPS filter for the relevant eNB builds (not to be used for menu "Pronto Diff", see chap. 3.4.3)
RELEASENOTE_PNS_SHOW	2	Show pns_show flag in release note
RELEASE_NOTES_DIR	/lteRel/ReleaseNotes	Directory of HTML and XML release note storage
STORAGE_PRODUCT	wmp	Folder name for build result storage on lteRel share
TMP_UPLOAD_DIR	/var/fpwork/wft/curre...	Path for generating temporary files needed for releasing
TRANSITION_TASK_CHECK	1	If set to 1 builds will wait for all tasks to be started before switching build to next state
XML_CANDIDATE_VERSION	9	Next upcoming XML release note version
XML_VERSION	9	Current used XML schema version

3.12.3 Deliverers

Admin/DeliverersController#index

In WFT all entities, which contribute to the SW delivery process, are called deliverer (sometimes also call "component"). A deliverer defines a particular set of baselines; it belongs to a project and a deliverer group. It has a title, a display name and a regular expression.
Hint: the title and the regular expression are used for granting and checking permissions for all the components stored within WFT (for details see chap. 3.12.3.2.7).

WFT administration defines within this sub-area the deliverers. The allocation of a deliverer to a certain permission group or in exceptional case to a particular user is done within sub-area "Permissions" (see chap. 3.12.6). With this mechanism the users get a list of regular expressions assigned and they are allowed to create and edit builds matching these regular expressions.

3.12.3.1 Parameters

3.12.3.1.1 Project

The project to which the deliverer belongs to.

3.12.3.1.2 Title

The title has to be unique for identifying this deliverer.
Therefore it also has to indicate to which project the deliverer belongs, e.g. LTE::MAC, LTETDD::TUP, etc.

3.12.3.1.3 Display Name

The display name is used for displaying purpose (e.g. within the release note). This name can be completely identical in different products, in order to mark that the deliverer implements the same functionality in different products, e.g. MAC.

3.12.3.1.4 Deliverer Group

The Deliverer Group has to be selected (or created) for checking that deliverer by the "SC Compatibility Check" (for details, see chap. 3.3.12.2).

3.12.3.1.5 Regular expression

The regular expression (regexp) has to be unique for every deliverer.
It has to be defined as strict/exact as possible. All baselines that belong to this deliverer need to match this regular expression (for details how to define a regular expression, see chap. 3.12.3.2).
Hint: the regexp is checked by creation, resp. import of a baseline.

For checking if a regular expression matches the relevant baseline-name-string the Ruby regular expression editor "Rubular" (see http://rubular.com/) can be used.

3.12.3.1.6 Path for Releasenote storage

The optional path parameter defines the deliverer specific part of the storage path for the release notes of that deliverer (hint: it has to be used in case of the deliverer’s display name contains blanks).
If this parameter will be left empty the deliverer display name will be used for that part of the storage path.
E.g. deliverer name: Central, path: LTE_eNB => storage for release notes for Central Builds: /lteRel/ReleaseNotes/LTE_eNB/.

Hint: after creation of a new deliverer the needed path will be created automatically within directory /lteRel/ReleaseNotes according the information given for that deliverer.

3.12.3.1.7 Check for newer Versions

The flag "Check for newer Versions" can be set and indicates for an eNB build which is in state "Buildable" or "Frozen", that there exists a newer version of this deliverer as currently chosen.

3.12.3.1.8 Check for matching ECL

The flag "Check for matching ECL" performs a check for matching ECL baselines if this baseline is used.

3.12.3.1.9 Disable XML Check

The flag "Disable XML Check" disables, when importing the XML release note, the checks for:
- if the basedOn is correct
- if branchFor lists a correct branch
(for details about existing XML validation checks see chap. 3.10.1).

3.12.3.1.10 Repository Check

The flag "Repository Check" disables the repository tag check (for details see chap. 3.10.1) to mark a baseline/deliverer as 3rd party SW.
In other words, for the deliverer which is marked in that way no check concerning the XML tags "repositoryUrl", "repositoryBranch", "repositoryRevision" will take place (see #11407).

3.12.3.1.11 Send Update on XML Release Update

(see #10153)
In case of the flag "Send Update on XML Release Update" is activated all changes of the content of a sub-component are over-taken to the baseline.
E.g. in case of an additional fault is added to a sub-component via an XML release note update, this fault is also visible within the current component and if the component is of category "SC" or "Central" an updated release note is sent.

3.12.3.1.12 Edit Deliverer Attachments and Edit Tools

The storage places of build tools (for details see chap. 3.3.6.7.4) have to be configured using these formulas.

3.12.3.2 Rules for creation

To get unique and reliable deliverers during creation the following rules have to be followed.

3.12.3.2.1 Rule 01 "possible characters for title"

The following signs can be used: [A-Z|a-z|0-9|_|\-|\:|\s]

3.12.3.2.2 Rule 02 "unique identifier"

The title as also the regexp of every deliverer has to be unique for identifying this deliverer.
The notation of the title has to lean on the SC name / baseline name.

3.12.3.2.3 Rule 03 "project specific"

Each project gets an own deliverer except the baseline is used commonly or it is not project specific.
The project relationship is indicated by a project specific prefix for the deliverer’s title.

The following table lists the currently available projects and the concerned prefixes for the deliverer’s title as also the prefixes for the baseline names (see also 3.12.3.2.10).
(This table has to be expanded as soon as needed, e.g. in case of a new project will be introduced.)
Remark: with LTE2157 "Introduce the new MBB release names in LTE" new prefixes were introduced for some projects.

Project	Prefix for deliverer title baseline name (/ baseline name according LTE2157)	Remark
Common	none none	
not project related	none none	e.g. PS
LTE common	LTE: LN, LND, LNL, LNT, DND	
FDD common	LTEFDD:	currently not used
FDD:WMP	LTEWMP_INC: LN	currently not used; Increment-numbering
FDD:WMP	LTEWMP: LN / FL	
FDD:DCM	LTEDCM: LND	
FDD:CRAN	LTECRAN: LNC	
FDD:FlexiZone	LTE_FZM_FDD: LNF / FLF	
FDD:FZController	LTE_FZC_FDD: -- / FLC	
~~FDD:FlexiLite~~	~~LTEFLEXILITE:~~ ~~LNL, LL~~	not an own project within WFT (last LNL-eNB Build: 14.05.12)
(KDDI)	LTEKDDI: LNK	not an own project within WFT
TDD	LTETDD: LNT / TL	
TDD:FlexiZone	LTE_FZM_TDD: LNZ / TLF	
TDD: FZController	LTE_FZC_TDD: -- / TLC	
Dualmode	DM: DND	currently not used
Dualmode:uREC	DualMode: (DM_UREC:) DND	
Dualmode:HDBDE	HDBDE: (DM_HDBDE:) DNH	"LRC"
WCDMA	WCDMA: WN	

3.12.3.2.4 Rule 04 "possible characters for display name"

The following signs can be used: [A-Z|a-z|0-9|_|\-|\:|\s]

Hint: Best choice is to use only [A-Z|a-z|0-9|_|\-]. This guarantees that, naming can also be used for the path notation (see chap. 3.12.3.2.12).

3.12.3.2.5 Rule 05 "no project specific part within the display name"

Within the string for the display name there has to be no project specific part, as the display name could indicate that this deliverer implements the same functionality in different products (see also chap. 3.12.3.1.3).

3.12.3.2.6 Rule 06 "avoid to change existing regexp"

Only in exceptional cases and after a deeper analysis an existing regexp can be changed. It has to be checked whether after the change the same builds match as before (otherwise the change may lead to unexpected behavior).

3.12.3.2.7 Rule 07 "accuracy of regexp"

The regular expression (regexp) should be defined **as strict and as exact as possible**.
(Details are handled by Rules 08 and 09).
Reason: for granting and checking permissions for all the components stored within WFT the title and the regular expression are used. Also the opportunity must be maintained to have the chance to differentiate future coming variants of these components.

Remark: in case of a deliverer is defined too widely, e.g. .*Component.* and later on there comes the need to differentiate this component like ABC_Component.* and DEF_Component.* there will be the need to split the existing (widely defined) deliverer. That means: it will be necessary to define a) new deliverers and b) the existing baselines (based on the "old" deliverer) have to split for these new deliverers (=high risk because error prone).
To avoid such a situation it's necessary to define the regular expression as precise as possible.

3.12.3.2.8 Rule 08 "margins for regexp"

Each regular expression should be limited by the start sign "^" and the end sign "$"
e.g. ^LN[0-9]\.[0-9]_xxxxx_[0-9]{4}_[0-9]{3}_[0-9]{2}$

3.12.3.2.9 Rule 09 "notation"

The notation for the LTE builds has to follow the naming rules defined within the "eNB Build and SC baseline Naming Rules" (see /4/). Short extract for a quick overview:

"Old Style":	<REL>_<SC>_<FB>_<nmo>_<pp>
"New Style short 1":	<REL>_<SC>_<FB>_<number>
"New Style short 2":	<REL>_<SC>_<FB>_<number>_<number>
"New Style long 1":	<REL>_<SC>_<FB>_<SC specific>_<number>
"New Style long 2":	<REL>_<SC>_<FB>_<SC specific>_<number>_<number>

Hint: to fit that rule for other projects besides LTE similar naming rules also would be needed.

3.12.3.2.10 Rule 10 "deliverer prefix and suffix"

For deliverers which follow the naming rules defined within the "eNB Build and SC baseline Naming Rules" (see /4/) project specific prefixes and unique suffixes are defined within the WFT "Configuration Settings" (see chap. 3.12.2.2) and have to be used for definition of the deliverers regular expressions.

<u>Project specific prefixes (DELIVERER_PREFIX):</u>

WMP:	^(FL[0-9]{2}(\.[0-9])?A?|LN[0-9]{1,2}\.[0-9])
DCM:	^LND[0-9]{1,2}\.[0-9]
TDD:	^(TL[0-9]{2}(\.[0-9])?A?|LNT[0-9]{1,2}\.[0-9])
HDBDE:	^DNH[0-9]{1,2}\.[0-9]
microREC:	^DND[0-9]{1,2}\.[0-9]
C-RAN:	^LNC[0-9]\.[0-9]
FZM_FDD:	^(FLF[0-9]{2}(\.[0-9])?A?|LNF[0-9]{1,2}\.[0-9])
FZM_TDD:	^(TLF[0-9]{2}(\.[0-9])?A?|LNZ[0-9]{1,2}\.[0-9])

KDDI	^LNK[0-9]\.[0-9]
FZC_FDD	^FLC[0-9]{2}(\.[0-9])?A?
FZC_TDD	^TLC[0-9]{2}(\.[0-9])?A?

<u>Unique suffix (DELIVERER_SUFFIX):</u>

[0-9]{4}(_[A-Za-z0-9,\(\)\.\-@]{1,100})?(_[0-9]{1,6}){1,2}$

Therefore the definition of such a deliverers regexp may look like, e.g.: [[DELIVERER_PREFIX]]_LOM_[[DELIVERER_SUFFIX]].

Hint: within the Admin sub-area "Deliverers" the regular expression of a deliverer is displayed in both ways: a) definition using the deliverer prefix / suffix key and b) the decoded form.
Whereas within Admin sub-areas "Build Configuration" and "Permissions" only the decoded form of the regular expression is shown.

3.12.3.2.11Rule 11 "MySQL specifics"

Due to MySQL restrictions a regexp may not use "\d" to express a digit, use instead "[0-9]".

Definitions of regular expressions using MySQL see http://dev.mysql.com/doc/refman/5.1/de/regexp.html.

3.12.3.2.12Rule 12 "possible characters for path"

The following signs can be used: [A-Z|a-z|0-9|_|\-]

3.12.3.3 Content collection over baseline

There exist 4 kinds (Inheritance, Collection, Summarization, Differentiation) of collecting content items (such as changes, corrected faults, reverted faults, features, unsupported_features, workarounds, restrictions, important_notes and needed configurations over several baselines / sub-baselines).
For every of these types a mask exist. The WFT administration can select for every deliverer with this mask those content items which have to be collected.
This mechanism has to be activated via console by setting per deliverer the relevant mask.

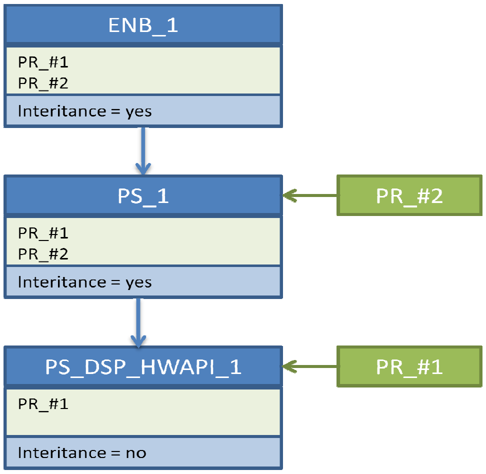
Mask value settings are:

content to inherit	**value**
changes	1
corrected faults	2
features	4
unsupported_features	8
workarounds	16
restrictions	32
important_notes	64
reverted faults	128
needed configurations	256

(e.g. value "511" means that e.g. collection of content items is set for everything).

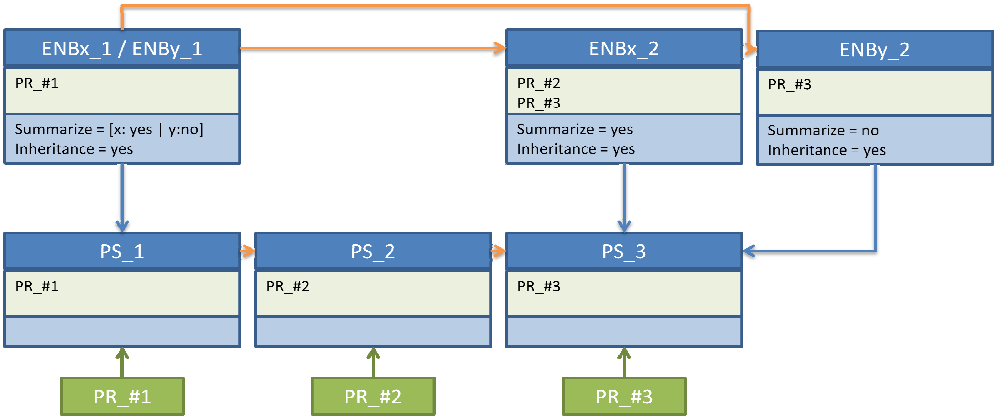
3.12.3.3.1 Inheritance

The inheritance flag works "vertical". This means from baseline to sub-baseline and so on. It is independent from the build category.
The Baseline where inheritance flag is set will include content information (e.g. concerning corrected faults, features, important notes, etc.) from all directly contained sub-baselines of next lower level. The content generation of sub-baselines is done the same way which means that the inheritance flag controls only one level below but not over all levels of sub-baselines.



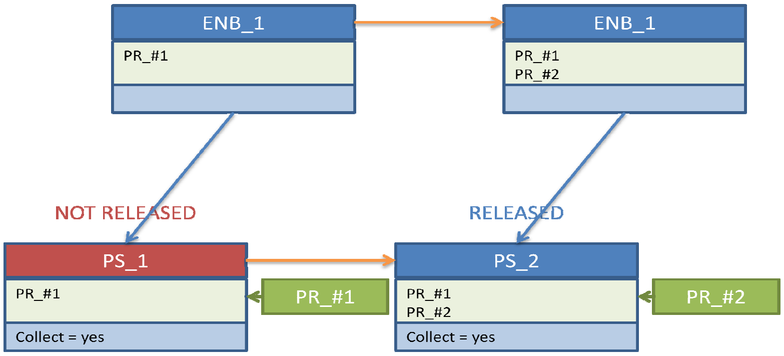
3.12.3.3.2 Summarization

(see #8581)
The summarize flag works "horizontal", which means that only between deliverers, or in other words between baselines on the same level the summarization can be activated. It is independent from the build category.
The summarize flag controls if the content of the baseline should include only new content compared to previous baseline ("no", see "ENBy"), or sums up content from all sub-baselines (“yes", see "ENBx"), from the same level, between these two baseline versions.
As precondition for Summarization Inheritance (see chap. 3.12.3.3.1) has to be active.



3.12.3.3.3 Collection

The collect flag works "horizontal", which means that only between deliverers, or in other words between baselines on the same level the collection can be activated. It is independent from the build category.
The baseline where collect flag is set will include content information also from predecessor baselines which are in state "not_released" This might be important for baselines which had been switched from state "released" to state "not_released", e.g. PS_REL baselines and therefore is used to publish content which would be forgotten with those not_released baselines.
Collection is an enhancement of Summarization. Therefore as precondition for Collection Summarization (see chap. 3.12.3.3.2) and Inheritance (see chap. 3.12.3.3.1) have to be active.



3.12.3.3.4 Differentiation

(see #9923)
For trunk builds the correction list (the list of corrected faults) of the imported release note has to be treated differently as these for SCs releases.
Since each trunk release note lists old and new corrections, only the new corrections have to be listed within WFT.
Therefore WFT has to compare the correction list from the new and the basedOn release.
The differentiation flag has to be set for those deliverers which have to be treated in that manner (the mask value for the correction item has to be set to "2").

3.12.4 Images

Within this sub-area the WFT administration is able to add, replace, and delete images used within the WFT internal Wiki.

3.12.5 Logging

Within this sub-area the WFT administration is able to view changes within the WFT DB sorted by several filter criteria’s (user, effected model, date- and time-range, etc.).

Hint: as the log file is growing very fast it is cleaned up every 30 minutes before the next MySQL backup take place. Currently the data of the past 30 days stay, as also the complete date concerning the branch handling.

3.12.6 Permissions

PermissionController#index

All users, who contribute to the SW Delivery and Build Process, have to get permissions, which enable and limit them to their assigned roles.
Permissions are granted to users and also to groups. Users can have one or many groups assigned. The permission management consists of 4 types of permissions which can be assigned to users and groups: Areas, Action Approval, Deliverers and Storages.

Areas
An area defines a section inside the workflow tool, which can be accessed by the user. E.g. Builds, Test Area, etc.

Action Approval
An Action Approval allows permission for one specific action a user or a group can perform, e.g. "Create central baseline".
With introduction of #11847 ("Permissions to modify build config templates") each Action Approval has to be assigned to a specific project.

Deliverers
All persons, which deliver their SW entity (deliverer) to the WFT, have to be enabled to do this. Therefore a deliverer defines with its assigned regular expression a set of baselines (for more details please refer to chap. 3.12.1). A user needs such a particular deliverer assigned to perform actions (for which he needs the relevant permissions) on baselines matching this deliverer’s regular expression.

Storages
(see #11539)
The access to a specific storage (for details see chap. 3.12.9) can be granted via that tab (see #11539).
Storage access is currently defined only for permission groups "Guest" and "Download". E.g. access to the IDA Sack is allowed for everybody (via group "Guest")

3.12.6.1 User Permissions

This view allows the administrator to administer the permissions per single user. This includes the group assignment, the areas which the user may see, all needed action approvals and the deliverers the user may handle.

3.12.6.2 Group Permissions

For most actions within the WFT a combination of all three kinds of permissions is necessary to enable users for their roles. E.g. a user who wants to be able to create a Central Build through the GUI needs the "Create Central Baseline" Action Approval and the according Deliverer assigned.
To ease the permission assignment it is preferred to define groups. So that assigning users to groups gives them all necessary permissions for a specific action. Granting permissions for new users will be done by Key-Users (see chap. 3.13.6).

3.12.6.3 Administrator Permissions

Currently the permissions for the WFT administrators are handled same as for other user groups (see chap. 3.12.6.2, via group Administrator).
This usage should change to a globally used flag "Admin", which can be set for the relevant user, so that it will be no longer necessary to grant specific rights (e.g. for deliverers, etc.) for the administrators.

This solution is currently partly implemented. Some sub-areas of Administration, e.g. Branches, Releases, System Releases are already using that flag.
Where implemented, these sub-areas then can be accessed by an user who has the relevant rights explicitly granted or for whom the flag "Admin" is set.

3.12.6.4 How to grant Permissions

It is preferred to grant permissions for a deliverer via a group. This means one deliverer should be strictly assigned to one permission group (exceptions can be the central build teams or load coordination).

Every user is by default assigned to the permission group "guest". This is the lowest permission level and grants only read permissions for some basic areas (see chap. 3.1).
The permission group "Administrator" grants complete control for configuration of the WFT.
All permissions that are necessary for assigned roles have to be given to the actors explicitly by the WFT administration.

3.12.6.5 Restricted eNB download access

(see #9719)
Only organizations and persons explicitly named in a list of allowed organizations/persons (positive list) shall be granted the permission to download eNB builds (for background see #9350).
Access rights can be given to a whole organization (department), mailing lists as well as to individual persons. Therefore an explicit permission group "Download" has been created. A few nominated access right owners (e.g. from CPD and I&V) are allowed to update these access rights when needed.
The organization (department) list consists of the sum of the departments (as shown within the Outlook address book) of all registered users within WFT.
The content of the Outlook list are those mailing lists which are contained within tab "Member Of" of the Outlook address books contact window.

As soon as somebody tries to download an eNB load who does not have access rights, this user gets informed with an "Access to download eNB builds is restricted - In case you need access please contact NN" message.

3.12.7 Release Note Templates (HTML)

ReleaseNoteTemplatesController#index

For each, out of WFT, released build an HTML release note can be generated and can be attached to the release mail.
These release notes are stored by WFT on a special share (/lteRel/ReleaseNotes/[SC], e.g. /lteRel/ReleaseNotes/MAC) and are additionally sent via email to the appropriate receivers.
There are release notes for an SC, for a central build to be released to Quicktest and also for a central build after the quick test is done to release it to I&V.

According to the needs each SC or variant of it (e.g. for a specific project like e.g. WMP or DCM) and/or each increment/feature build of an SC can have its own release note.
For each of these different kinds of release notes a release note template has to be defined containing all relevant information, email addresses (sender / receiver) and so on.
The HTML release note is then created from such a Release Note Template.

Within 3 tabs the administrator can enter data to form the template. Within tab "Mail" several values for the release note sending process, e.g. subject, sender, recipients, etc. and the mail text can be defined. Within tab "Settings" several general setting of the release note (e.g. title of the template, time zone to be used, attachments of the release mail, etc.) are defined.
The Release Note section itself is defined within tab "Template" and consists of several block elements. Each of them has a specific behavior. These predefined elements belong to an element pool. The administrator has to form the Release Note Template copying and pasting these elements out from the element pool into the Release Note section. Where needed the relevant elements have to be edited.

In the following some examples of these elements are listed. (These elements are defined inside /app/views/release_note/__*_.html.erb.)

Headers
Used to define HTML headers.

Table of Contents
Uses headers of type2 to create a table of contents.

Table of Baselines
Used to return a table with a list of all contained sub-baselines.
Hint: in case of a deliverer name is added to such a Table of Baselines, all contained baselines of that sub-baseline ("sub-baseline" in relation to the baseline for which the template is defined) are listed.
E.g. see template "RP" or "HDBDE:CB_2" (required with #9821).

Table of Changes, Faults, Features, Restrictions, etc.
Used to return tables with the according table of information.

Text
Used to define free editable content for the release note.

3.12.8 Release Note Templates (XML)

XmlReleasenotesController#index

XML Release Notes are generated (for builds of category "SC" and "Central") in parallel to the HTML Release Notes. They are stored and sent to the appropriate receivers, like HTML Release Notes.
There exist several types of XML Release Notes, e.g. one for all SC's (to have a consistent content for all SC’s), one for Central Build and one for Platform releases. Also for different XML schema versions there exist different versions of these templates.

Every SC as also Central Build has to provide with each Release an XML Release Note. As the SCs and Central Build have different requirements according these XML-release notes (e.g. the release note for an eNB contains all faults of all contained baselines) there also have to exist different XML Release Note Templates.
Out from an XML Release Note Template the XML Release Note is being created.
The XML Release Note Template has a title and the content. The template has to be entered into the content field with erb (embedded ruby) syntax. In this way the XML Release Note Template is much more flexible than the HTML Release Note Template but editing is more complicated.
The relevant XML Release Note Template has to be chosen during creation of each build.

For more details concerning the content and the definition of an XML Release Note see chap. 4.2.3.
The overview table shows all existing XML Release Note Templates.

3.12.9 Storages

admin/StoragesController#index

The storages are used to store the results of a central build production (e.g. the complete load, MAP & OUT files, etc.) in order to be able to download them from the WFT download area (see area Builds, chap. 3.3.6.7.3 "Build results"). This download area ("Build results") contains per load different sections (the storages).
These storages define the storage locations of the build results. One build may have many storages assigned. Within the sub-area Storages the administrator defines for these storage details, settings and items.
During the build process the build responsible can select these storages.

Details
The Storage Details define the title of the storage and the source of it.
Storage sources can be e.g. a share, the used Hudson or Jenkins server or SVN (due to historical reasons still called "clearcase").
Additionally a flag can be set, that in case a build is not released, no data for download is offered.

Settings
A Storage Setting is a global setting for this storage, e.g. baseline name, tag name.
E.g. setting Title to "Label" and Value to "#{baseline}" will display e.g.
"Label: LN3.0_ENB_1103_089_00" within the tab "Storages" on the concerned build page.

Items
Per storage several storage items can be defined. Each storage item has a title, a path and a description.
The path defines the location of the file. Within the path several variables are possible which are evaluated.

Available Variables:
- baseline
- increment
- product
- product_lower (lower case for project title, e.g. WMP -> wmp)
- short_version (six-digit form of a baseline title)
- fep_version
- feature_build
- fb_or_inc
- build_id_us (us: replaces a blank with an underscore)
- faraday
- faraday_nus (nus: deletion of the underscore)

3.12.10 Translations

admin/TranslationController#index

Within this sub-area the WFT administration is able to define a pair of key and value for easy managing textual parts within the tool. That means, changing such a defined text needs no longer adaptation of the code, only changing the relevant configuration entry.

3.13 Area Management

ManagementController#index

This area is a kind of administration area for users with special rights. E.g. for Load Coordinators who are responsible for defining and managing branches, releases and system releases and for Build-Team members to manage build configuration templates (see #11847), or for Key-Users who are enabled to manage the permission group they are responsible for.

3.13.1 Branches

BranchesController#index

This sub-area acts as a central point to control the load planning process. Therefore the managing of that section has to be done by the concerned (project dependent) Load Coordination.
Tasks like creation, editing or deletion of branches, starting automated building of eNB loads or importing data out of the LDR have to be done herein.

Every project has more than one branch beside the main branch. LTE projects are e.g. built on main branch (trunk, CI) for implementing the required features. After the so called feature build exit only such corrections are allowed which are based on the last feature build.
This is done on so called maintenance branches, which are an attribute of every baseline, which is generated within WFT or is published as external delivery to WFT.
Each build has one branch assigned.
Only branches which are not assigned to any build can be deleted.

3.13.1.1 Parameters used for branch administration

The following listed parameters are visible within the overview of the branches and are used mainly for branch administration.
The overview of the branches can be filtered and sorted on several of these parameters.
A warning, relevant for the "autobuild functionality", on top of the page points to a problem concerning overlapping next eNB builds (see column "Next Baseline").

3.13.1.1.1 Title

The title has to be unique for identifying this branch. No spaces should be used.
This parameter can be edited only once, during creation of a new branch.

3.13.1.1.2 Template

This is a pattern of those baselines which belong to that branch. It is just for Information how the baseline names have to look like.
These templates are also displayed on the page "Overview of active Branches" (see chap. 3.6.1).

3.13.1.1.3 Delivery Type

For classification of a branch (e.g. "Maintenance Package", "Priority Package", "Factory Load", "Pilot Delivery Package", "Others", etc.) this delivery type has to be used.
The amount of these packages per project is displayed on the page "Overview of active Branches" (see chap. 3.6.1).

3.13.1.1.4 Project

The project to which the branch belongs to.

3.13.1.1.5 Release

The release to which the branch belongs to.

3.13.1.1.6 System Release

The system release to which the branch belongs to. The system release is imported from LDR.

3.13.1.1.7 Featurebuild

The feature build to which the branch belongs to. For no assignment "any" is selected.

3.13.1.1.8 Top Flag

At this place the indication of the TOP flag is only for information.
For details concerning the functionality "Correction Policy" refer to chap. 3.17.5.

3.13.1.1.9 Visible

Indicates if the branch is visible and can be selected within area "Branches".

3.13.1.1.10 Writable

Only branches which are writable may be used for creating new builds. This will be checked.

3.13.1.1.11 Autobuild

Within the column "Autobuild" of the table "overview of branches" the "autobuild functionality" can be managed. For details refer to chap. 3.13.1.2 resp. to chap. 3.17.4.
- It is possible to toggle between Autobuild active/inactive,
- in case the autobuild functionality is activated the amount of concerned/changed SCs is displayed,
- the user can start a forced autobuild (independent from the timeout settings) and
- a preview of the autobuild can be displayed.

3.13.1.1.12 Actions

Within that column of the table "overview of branches" actions like copying or editing (settings for the "autobuild functionality" also have to be done herein, see chap. 3.13.1.2) of the branch and viewing a log about edit activities can be triggered.

3.13.1.1.13 Build Deletion

Within that column of the table "overview of branches" the deletion of no longer used builds of a branch can be managed, resp. the result is displayed. For details refer to chap. 3.17.6.

3.13.1.2 Parameters used for automated building

Details about the "autobuild functionality" can be found in chap. 3.17.4. In the following those parameters are described which have to be handled for managing this functionality. The parameters have to be edited using the relevant link within column "Actions" of the table "overview of branches" (see chap. 3.13.1.1.12) and are divided up within the tabs "Details" and "Autobuild" of the "Edit Branch" formula.

3.13.1.2.1 Parent branch

Within tab "Details" for the branch point definition the concerned parent branch has to be selected.

3.13.1.2.2 Branchpoint

Within tab "Details" for the branch point definition the relevant eNB load for the branch point has to be selected.

3.13.1.2.3 Next baseline

Within tab "Details" the possible name of the first baseline of the new branch is offered. The name can be edited.

3.13.1.2.4 Tab Autobuild

Within that tab various settings for controlling the autobuild functionality have to be set, e.g. activation of the functionality, timing settings, check settings and baseline settings.

Settings for a time period during working time and during weekend/night are possible:
- for the time period during working time the relevant weekdays, the duration of the working time (start and end time according the chosen time zone), the time zone and the grace period (the interval for building the next build) can be set,
- for the time period during weekend/night a different the grace period can be set.

Within section "Check Settings" the Correction Allowed Check, ECL Check and the SC Compatibility Check can be activated for that build.

Within the baseline settings those SCs have to be selected for which it has to be checked if newer deliveries are available. The check for new baselines to be built is running periodically in the background. Once all conditions for a new eNB are fulfilled it is started.
In case of only those SCs have to be considered for the up-coming eNB build which are younger then the based on eNB build the Checkbox "Take only baselines delivered after last eNB" has to be checked.

3.13.1.3 Triggering LDR import

Within the tab "FB Plan" of the "Edit Branch" formula:
- the download/import of data out of the LDR can be triggered on demand using the link "Import features from LDR" (usually the LDR import is performed every hour).
- and an email can be ordered, containing a list of all Supported/Unsupported Features taken from the LDR and belonging to the selected System Release and Featurebuild, which are assigned to the branch.

For details concerning the data import/handling of the LDR refer to chap. 3.17.2, resp. chap. 4.4.5.

3.13.2 Build Configuration

Admin/BuildConfigTemplatesController#index

Build Configurations are used for Central builds to provide all needed information (e.g. contained baselines) to the build scripts. Therefore a special format - aligned with the build script - is defined.

Usually different products resp. different branches use different Build Configurations. These Build Configurations are represented by various Build Configuration templates. With introduction of #11847 all templates are using a versioning. This means that one Build Configuration template can have several versions and each eNB build has one Build Configuration template with a specific version of it assigned. This assignment is being copied when incrementing the build.

With introduction of #11847 all templates can be managed by permitted users (these special and project specific rights have to be granted by WFT administration).

Each template consists of several elements, each element represents one or more lines within the Build Configuration File (see also chap. 3.3.6.4).
These elements are predefined within an element pool.
The permitted user has to form the Build Configuration template by copying and pasting these elements out from the element pool into the configuration template. Where needed the relevant elements have to be edited.

Two types of elements exist: Text and Deliverer.
The sorting of the elements during creation of the template also defines the sorting of the elements after creation of the Build Configuration.

Text
A Text element consists of static text. The only interpolated variable is [baseline] and [based_on]. These variables are replaced by the baseline or parent baseline when the Build Configuration file is generated.

Deliverer
Each element of this type of element is a placeholder for a specific baseline (so called deliverer; for details please refer to chap. 3.12.1). If that baseline/deliverer is used inside the build, this element will be evaluated and the contained variable [baseline] will be replaced with the real baseline (the build scripts use this information to evaluate the place where to find the baseline: SVN or on a share).
In case the variable [repository_path] is used (see #8173) the repository path where the relevant baseline is stored can be directly evaluated for the Build Configuration out of the information given within the XML Release Note (e.g. [cleaned_repository_url][repository_branch]@[repository_revision]).
In case the variable [download_path] is used (see #10715) the download path where the relevant baseline is stored can be directly evaluated for the Build Configuration out of the information given within the XML Release Note (e.g. [downloadItem]).
(hint: an XML check checks if there is only one and a correct "downloadItem" named within the XML Release Note, see also chap. 3.10.1).

3.13.3 Releases

ReleasesController#index

WFT administrators and load coordinators can create a new release. Within sub-area "Branches" (see chap. 3.13.1) the same group of people is able to assign one of these releases to one branch. Within the Release table all assigned branches for one release are shown.

3.13.4 Branch Types

ReleasesController#index

Within this sub-area the user is able to create a new branch delivery type (see also chap. 3.13.1.1.3), which is used for classification of a branch (e.g. "Maintenance Package", "Priority Package", "Factory Load", "Pilot Delivery Package", "Others", etc.; see also chap. 3.6.1).

3.13.5 System Releases

SystemReleasesController#index

System Releases are created during import of the LDR. In exceptional cases WFT administrators and load coordinators are able to create also a new System Release. Within sub-area "Branches" of area "Management" (see chap. 3.13.1) the same group of people can assign one of these System Releases to one branch. Within the System Release table all assigned branches for one system release are shown.

3.13.6 Permissions - Key-Users

(see #7195)
Key-Users are able to manage the members of the permission groups they are responsible for by their own. They had to be member of the group "Management User".
WFT Administration selects users which should act as Key-Users for a specific group.
Each Key-User is able to see the area "Management". Using this area he is able to see those permission groups he is responsible for.
A Key-User is allowed to add or remove other users to the group (but he is not allowed to create or delete "Key-Users" by him own). Additionally he can create an access key for that group. If an access key is created at least one email address (or a mailing list) for this group is required. More email addresses have to be separated by a ";" (these email addresses work equivalent to a functional account).
Using the access key has the effect that the concerned group is displayed as the user who performs the relevant action(s).
The "Grant" Flag can be used by WFT administration to add a new user to a permission group as Key-User.

3.14 Permission Request

CommonController#request_permission

In case a user needs special permissions (more than granted by default, see also chap. 3.12.6), he has to search for the required group and has to apply for membership. The application will be reviewed and granted by the Key-User(s) of the concerned group.

3.15 WFT News

NewsArticlesController#index

The WFT News section is displayed on the main page and lists the latest news published by WFT team. These news are intended for each user and contain important information, e.g. maintenance breaks, new features, etc.

3.16 Wiki

WikiController#index

The Wiki is used for providing instructions for several parts of the Workflow Tool. The Wiki is maintained by WFT team.
The link to the Wiki can be found using the Help menu. Within certain areas the area specific Wikis can additionally be selected directly via the Help menu.

3.17 General functionality

3.17.1 Project handling

All content of WFT is project related. Every deliverer has to be related to one specific project, e.g. such as WMP, DCM, TDD (see also chap. 3.12.3.2.3).
In case of it is not possible to do such an assignment (e.g. for Platform baselines) the project "ALL" is defined.
The visibility of the content of WFT is depending on the selected project.

3.17.2 Release Note generation using data of the LDR

(see #9383)
Every eNB release note has to be correctly filed up with supported and unsupported feature information.
Originally all features contained within an eNB build had been collected out of the XML Release Notes from the concerned SCs. This handling has been replaced by selecting the features out of the Feature Build Plan (FB Plan).
As the FB Plan does not contain all features (older features are deleted to downsize the FB Plan), but data of all features is still needed for release note generation within WFT, the LDR (LTE Data Repository) is used from now on for feature data import.

The way how to assign features into a Supported and an Unsupported Feature list of one eNB build, a link to the LDR and the process description about the LDR can be found here /5/.

The Release Note generation functionality using the data of the LDR can be divided into following steps:

1. Importing the data out of the LDR (for details see chap. 4.4.5).
2. Assigning the features to the build.
   a) Each eNB is assigned to a branch. Each branch can be assign to a system release and a feature build. If the assigned SystemRelease is marked with "fbplan" (flag has to be set by WFT Administration), then the features for the eNB build are taken from LDR.
   b) Out of LDR all those features are taken that
      - have assigned given SystemRelease,
      - have assigned given featurebuild (if featurebuild "any" assigned-> featurebuild is no condition) and
      - have flags different than M (module level feature) or D (development level feature) in level field.
   c) Features are assigned when build is created or switches to frozen. After that it can be done only manually by WFT team.
3. Features divided up into a Supported and an Unsupported feature list.

3.17.3 Release Queue

(see #364)
The Release Queue improves the releasing mechanism.
Behavior before introducing that queue: the condition for releasing a build was that the build (e.g. eNB) was in state "Succeeded" and the Auto Release Button was activated.
With introducing that queue the build is added to that Release Queue (/app/models/release_queue.rb).
As soon as the build is added all prontos assigned to this build are marked to be updated and are updated (by Pronto Updater). Once all Prontos are updated the build is released.
A timeout of 5 minutes avoids hanging releases (and users claiming about missing release note).

3.17.4 Automated Building

(see #8341)
The automated building ensures an unattended production and has to be enabled only for dedicated branches. The activation and the setting for that kind of building have to be done by Load Coordinators.
The baselines which have to be incorporated for the automated building can be selected. Also there is the choice to take only baselines which have been delivered after the last eNB production.
It is permanently checked after a previous build was successful built if a new baseline is available for a next build.
The production sequence depends on the weekday (normal working time / weekend) and is configurable per branch due to different time zones.

Sequence / conditions:

- the last eNB of that branch is succeeded, released_for_quicktest, not_released, released or released_with_restirctions.
- for the selected baselines the successor are figured out using the "based_on".
- only such baselines which have the same branch named in "BranchFor" as the concerned eNB are regarded.
- only baselines matching correction policy and SC Compatibility check are taken
- for all selected baselines it is checked if there exists a successor.

3.17.5 Correction Policy for active branches

(see #6799)
For some branches only corrections with a defined TOP Pronto label (TOP flag) are allowed. This is defined within the Correction Policies. For details concerning TOP Pronto process see: https://bts.inside.nsn.com/twiki/bin/view/PreIntegration/PO_ShowStopperProcess).
The table of the Correction Policy (see https://sharenet-ims.inside.nsn.com/Open/D481401855.latest) is imported every 5 minutes.
Out of that Excel file the information per branch of the column "SCM" of the project related tabs is taken.
This information is displayed for information within the branch overview of the area Administration (see chap. 3.13.1.1.8), is shown within the Overview of active Branches (see chap. 3.6.1) and is needed for the Correction Allowed Check (see chap. 3.3.12.6).

3.17.6 Deletion of unused build results

(see #6497 and #7423)
Sequence:

- I&V is marking the branch which builds could be deleted (area Administration; sub-area Branches) as "requested".
- After that an email containing a list of all loads to be deleted will be sent (currently to I&V and WFT: Andreas H.).
- This email has to be forwarded to LTE SW Build Team to delete the requested builds.
- LTE SW Build Team confirms the deletion and I&V sets the branch to "deleted" (within the formula for editing branches the relevant selection of the field "Build Deletion" has to be selected).
- As soon as the indication "deleted" is set this fact is shown to the user by an according hint (#7423).

3.17.7 CSS Sprites

(see #6113 and #7309)
By using CSS sprites (several single graphics are pooled to one file, a kind of table and via CSS only the relevant part will be selected) it is possible to reduce the number of HTTP requests necessary for one page to load.
The gem "sprite-factory" is used for generating sprite images and style sheets (Instructions can be found at: https://github.com/jakesgordon/sprite-factory and http://codeincomplete.com/posts/2011/8/6/sprite_factory_1_4_1/).
A rake task which can be executed in development environment regenerates sprites for images in app/assets/images/sprites and app/asssets/images/icon.
The files which are generated will be app/assets/images/sprites.png, app/assets/images/icons.png, app/assets/stylesheets/sprites.css and app/assets/stylesheets/icons.css.

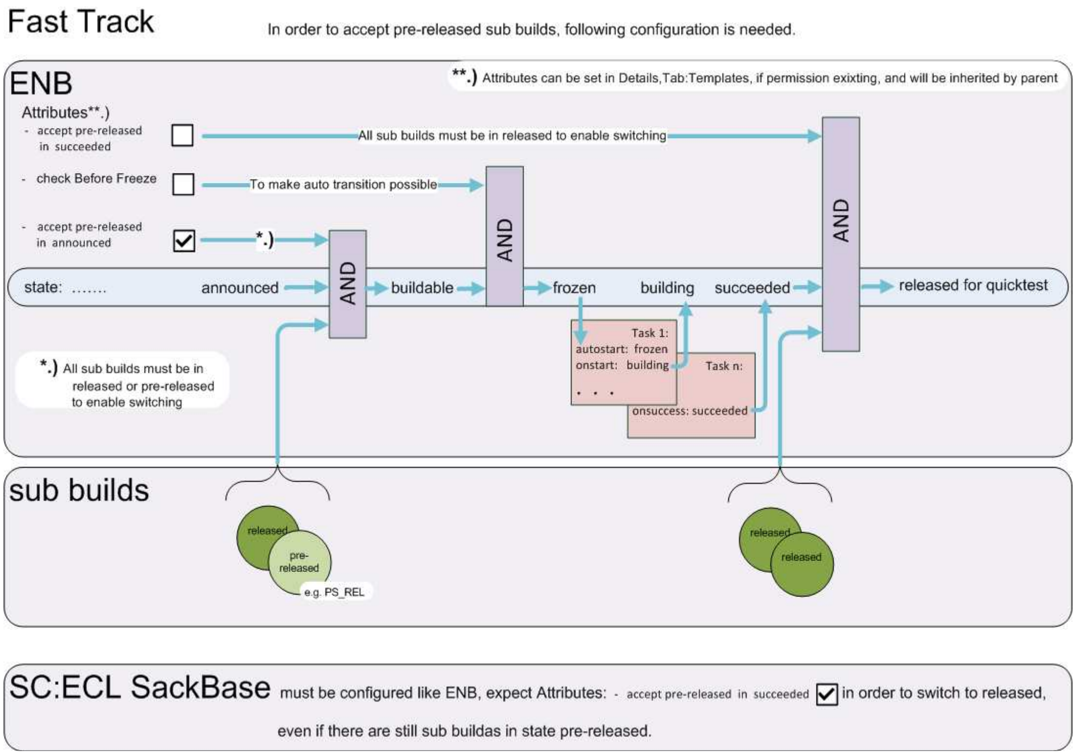
3.17.8 Fast-Track Releasing of PS_Rel builds

(see #10823, released with v2.18)
To speed up the PS_Rel releasing process in situation where an urgent correction build for PS_Rel is requested a "pre-released" PS_Rel build has to be provided to bypass some of the time consuming PS_Rel CI tests.
As soon as these long running PS_Rel CI tests are passed the PS_Rel build switches from "pre-released" to "released", resp. "not released".

A new state "pre-released" and several state transitions (e.g. to and from "pre-released" and have been introduced (see chap. 3.3.8).
The additional state "pre-released" indicates, that a sub build can already be used, the eNB is built, but not released for quicktest.

Handling within the concerned SCs:
PS_Rel is tagged and switched afterwards to "pre-released". There it waits until all sub builds switch from "testing" to "released".
eNB accepts released and pre-released sub builds in state "announced". All tasks are executed and if successful, ENB is switched to sate "succeeded". When all sub builds (here: PS_REL) are switched to "released", eNB switches to "released-for-quicktest".
ECL_Sack_Base accepts released and pre-released sub builds in state "announced". All tasks are executed and if successful, it is switched to sate "succeeded" and afterwards immediately to "released", even if sub build is in "pre-released".

Needed configuration for Fast-Track:



Fast Track

In order to accept pre-released sub builds, following configuration is needed.

**.) Attributes can be set in Details,Tab:Templates, if permission exixting, and will be inherited by parent

*.) All sub builds must be in released or pre-released to enable switching

SC:ECL SackBase must be configured like ENB, expect Attributes: - accept pre-released in succeeded ☑ in order to switch to released, even if there are still sub buildas in state pre-released.

Figure 6: Needed configuration for Fast-Track mode

Enhancements of state diagram:

- from state "Announced":

  new: transition to "Pre-released" or "Error" possible (
  (transition can be done only by task)
  new: transition to "Not Released" manually possible.

- from state "Pre-released":

  switches to "Buildable", if all sub-components are switched to "Released".
  switches to "Not released, if at least one sub-component is switched to "Not Released".

Behavior:

1. Fast-Track behavior on PS_Rel side:
   - Behavior of state transition from "announced" to "buildable" is blocked if all sub_components are in state "released".
   - Instead: tagging task tags PS_REL within SVN and switches build state from "announced" to "pre-released" in case of success.
   - In state "pre-released" PS_REL waits for following configuration:

     At least one of (next level)-sub_components is set to "not_released" -> PS_REL state switches to "not_released".

     All (next level) sub_components are set to "released" -> PS_REL state switches to "buildable" and immediately (in PS_REL configuration) to state "frozen".
   - In state "frozen" the next task is started, which switches via building to "succeeded".
2. Modifications to be done by PS_Rel build responsible:
   - Increment API command will be extended by attribute "-F flags[]=fast_track"
   - Modification of tagging task:

     Function: tag PS_REL
     Autostart: announced
     on success: pre-released
     (on error: error)
   - Creating a second task:

     Function: write SVN property file for WCDMA / insert revision number
     Autostart: frozen
     On start: building
     On success: succeeded
     On error: error
3. Fast-Track behavior on eNB side:
   - eNB will switch from state "announced" to "buildable", if (next level) sub_components are either in "released" or "pre-released".
   - eNB will not switch automatically to "released_for_quicktest" until all sub_components are in "released".

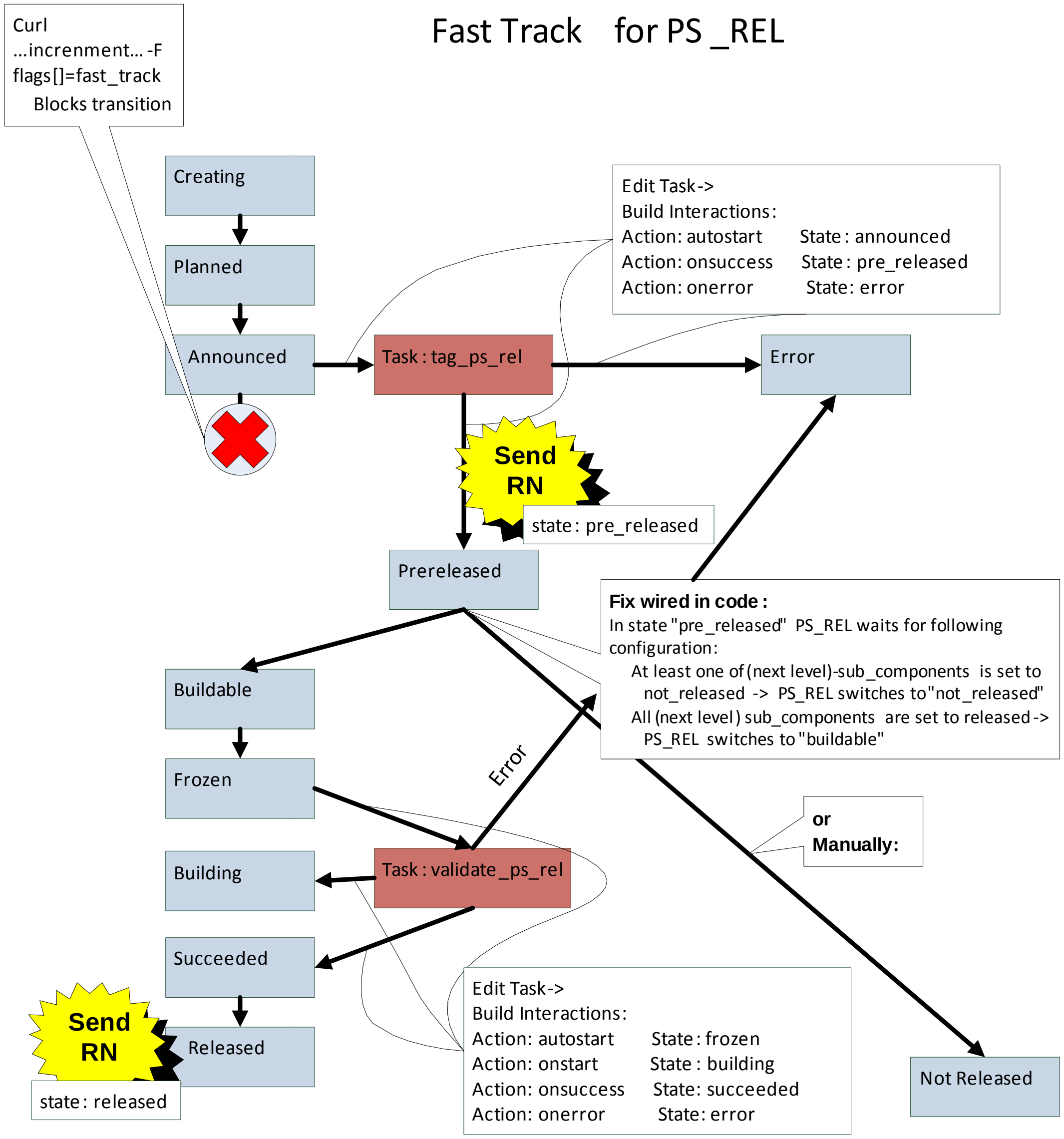

Fast Track for PS _REL
Curl
…incrnement… -F
flags[]=fast_track
Blocks transition
Creating
Planned
Announced
Task : tag_ps_rel
Error
Edit Task->
Build Interactions:
Action: autostart State : announced
Action: onsuccess State : pre_released
Action: onerror State: error
Send
RN
state : pre_released
Prereleased
Fix wired in code :
In state "pre_released" PS_REL waits for following configuration:
At least one of (next level)-sub_components is set to not_released -> PS_REL switches to "not_released"
All (next level) sub_components are set to released -> PS_REL switches to "buildable"
Buildable
Frozen
Error
or
Manually:
Building
Task : validate_ps_rel
Succeeded
Send
RN
Released
state : released
Edit Task->
Build Interactions:
Action: autostart State : frozen
Action: onstart State : building
Action: onsuccess State: succeeded
Action: onerror State: error
Not Released

Figure 7: PS_Rel releasing using Fast-Track mode

4. Modifications to be done by WFT administration by request of eNB build responsible:
   - Enable Fast-Track behavior for eNB's in Admin Configuration (see chap. 3.12.2.2) by setting configuration FAST_TRACK:
     (this configuration works / is needed only for builds of category 1 ("Central"), as per default the state "pre-released" is excluded for this kind of builds)
     Key: FAST_TRACK value:1 reference: ALL
     (all eNB's are now in Fast-Track mode)
   - In order to activate Fast-Track only for e.g. project WMP the entry has to be modified / extended as follows:
     Key: FAST_TRACK value:0 reference: ALL
     (all eNB's are now disabled for Fast-Track mode)
     Key: FAST_TRACK value:1 reference: WMP
     (except all WMP eNB's are enabled)

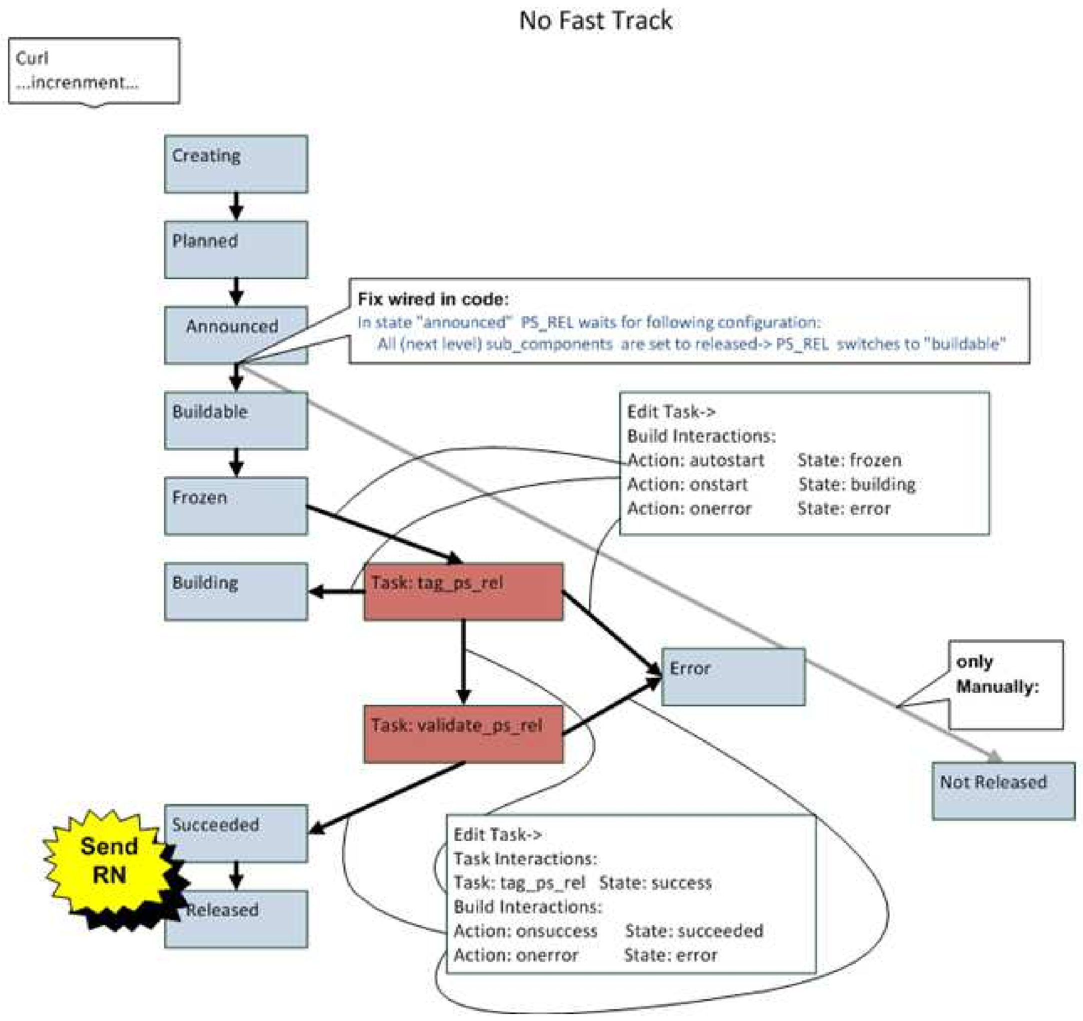

No Fast Track
Curl
...incrnement...
Creating
Planned
Announced
Fix wired in code:
In state "announced" PS_REL waits for following configuration:
All (next level) sub_components are set to released-> PS_REL switches to "buildable"
Buildable
Frozen
Building
Edit Task->
Build Interactions:
Action: autostart State: frozen
Action: onstart State: building
Action: onerror State: error
Task: tag_ps_rel
Error
only
Manually:
Task: validate_ps_rel
Not Released
Send
RN
Succeeded
Released
Edit Task->
Task Interactions:
Task: tag_ps_rel State: success
Build Interactions:
Action: onsuccess State: succeeded
Action: onerror State: error

Figure 8: normal PS_Rel releasing (without using Fast-Track mode)

4 Interfaces

4.1 Overview

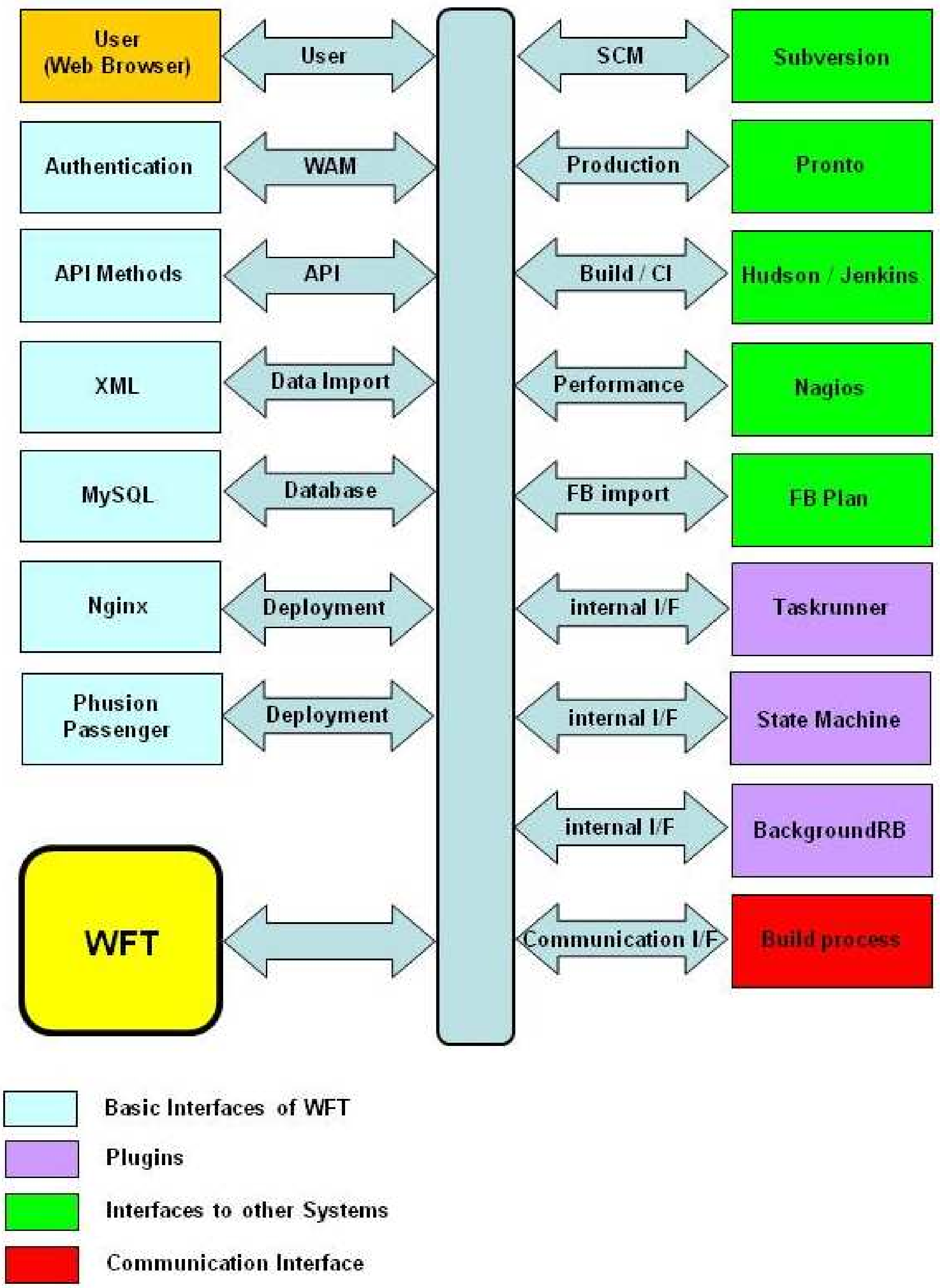


Figure 9: Overview of WFT Interfaces

4.2 Basic Interfaces

4.2.1 WAM Authentication

WFT uses the login mechanism of "Silent SSO" (Silent Single Sign On), NSN internally called WAM.
If the user enters the first time the WFT or if SSO is not active one will be automatically be redirected to the WAM Login Page.
To enable "Silent SSO" one has to modify some settings described here:
https://inside.nsn.com/global/MyServices/IT/ITforIT/WAM/Pages/WAMServiceFAQ.aspx.
(The user has to look for the caption "How to enable Silent SSO in Firefox?". On the same page one also gets more information about SSO and Silent SSO).

4.2.2 API Methods

4.2.2.1 General

The Workflow Tool API is an interface which allows performing all actions (see #11793 and #267) offered by the WFT GUI without GUI interaction. This makes it possible to interact with WFT also via CLI, resp. via scripts.
A description of the possible API methods (CLI commands) is offered on the applicable WFT pages to the users via a link (button) "API DESCRIPTION" (see also chap 3.12.1).

WAM is bypassed for these API methods, therefore an individual access_key is necessary, which has to be generated by each user itself. All actions done using this access_key are done in the scope of this user’s permissions.

Up to now not for all possible pages such an "API DESCRIPTION" is existing, but will be added step by step by WFT administration.
In the following an overview of the currently existing implementation is described.

4.2.2.2 Basic Commands

ExtController#api

The main purpose of this API is creating baselines from well formed XML (see chap. 4.2.3), validation of an XML file, uploading attachments and delivering build contend for further proceeding.

To perform actions in the scope of a baseline, the user needs the associated deliverer(s) assigned.

This API can be used using the commands "curl" or "wget" ("wget" can only be used for file or xml uploading).

Overview of available API methods

Method	Description	Parameters	Arguments
xml	Creates a Baseline from Uploaded XML file in predefined format	none (baseline is detected from XML)	File sent with post request as variable file See example below for further details!
xml_ validate	Validates the uploaded XML file	none (baseline is	File sent with post request as variable file See example below for further details!

Method	Description	Parameters	Arguments
	against xsd schema	detected from XML)	
upload	Uploads an attachment to a baseline	baseline	File sent with post request as variable file. Optional you can provide type of uploaded file with parameter "type=", where type is one of rn, rn_xml, needed_config, test_result or other. See example below for further details!

link	Adds a link to a baseline	baseline	name=IDA2
			link=http://...

repository	Changes repository URL for this build	baseline	repository=
build_content	Shows baseline information in XML format	baseline	See description below for further details!
knife_create	Creates a Dummy Knife	none	See description below for further details!
increment	Creates a new PS, incremented from Parent PS	xml_releasenote_id parent , sub_builds, branch_for	See description below for further details!
quicktest_api	send quicktest release mail	templates get preview release	See description below for further details!
quicktest2_api	send quicktest2 release mail	templates get preview release	See description below for further details!

Command syntax

```
curl -k https://wft.inside.nsn.com/ext/api/<method> <parameter>
-F "access_key=<access-key>" [arguments]
```

e.g.

```
curl -k https://wft.inside.nsn.com/ext/api/xml -F
"access_key=1234" -F "file=@relnote.xml" [-F "update=yes"] [-F
"testing=yes"]
```

If an SC needs to update the SC release an additional flag -F "update=yes" is required to indicate that this is an update.
E.g.:
Without this update flag, WFT ignores the API call, since the release already exists in WFT.
Hint: For the first import (the baseline still doesn't exist) it is not allowed to use the flag "update=yes".

Baselines of category "External" can be set to state "Testing" using the flag "testing=yes".

Returncode for API commands

Wget automatically returns an exit code which tells that the command was executed correctly.
For curl it is necessary to add the -f option ("fail silently") to get an exit code (e.g. for usage within a script).
Hint: In case the command line is used and it is needed to get more detailed failure output the -f option has to be left.

<u>Examples</u>:

1. Create a Baseline from well-formed XML file:

```
curl -k https://wft.inside.nsn.com/ext/api/xml -F "access_key=
1234" -F "file=@relnote.xml"
```

2. Upload an attachment to a Baseline:

```
curl -k https://wft.inside.nsn.com/ext/api/upload/LN1.0_PHY_
RX_1003_001_00 -F "access_key=1234" -F "file=@relnote.html" -F
"type=rn"
```

4.2.2.3 Knife API

ExtController#knife

The Knife API is used by the knife execution script to switch the state of executed knife requests. To use the Knife API the user needs to have the "build_knife" permission.
Allowed actions are info, started, preparing and succeeded.

4.2.2.4 Task API

ExtController#task

The Task API is currently used by the knife execution script to delete the knife task after successful execution.
Allowed actions are start and delete.

4.2.2.5 Increment

Release PS load with cmd increment.
Parametes:

- xml_releasenote_id #this param selects the correct template for the release note
- parent #this param indicates the parent PS on which this build is based on.
- sub_build[] #can be repeated as often as possible
- branch_for[] # can be repeated as often as possible
- access_key

Example: see chap. 4.2.2.9.

4.2.2.6 Quicktest API

Release ENB Load for Quicktest Module 1
Following methods are available:

a) templates:
get a list of all quicktest templates consisting of template name (not unique) and template id (unique) , which belong to a project (e.g. WMP).
Project will be indicated by baseline_name.

Example:

```
curl -k https://wft.inside.nsn.com/ext/quicktest_api -F "method=
templates" -F "build=LN5.0_ENB_1210_011_90" -F "access_key=1234"
```

b) get:
Get content of the quicktest results, or quicktest_template in xml syntax.
Following cases are covered:

1. Build was just created and no quicktest preview or release was sent:
Then the content of the quicktest_template is delivered, if no template id given, or template id is default. Placeholders are substituted.
2. A quicktest preview or release was already sent, but template_id != default id.
Then the content of the quicktest_template is delivered. Placeholders are substituted.
3. A quicktest preview or release was already sent, and template_id not given, or identical to default id.
Then the content of the quicktest_result is delivered for all components, which are set in the quicktest result.
Components, which are not set in quicktest_result, are taken from quicktest_template.

Example:

```
curl -k https://wft.inside.nsn.com/ext/quicktest_api -F "method=
get" -F "build=LN5.0_ENB_1210_011_90" -F "access_key=1234"
```

c) preview:
Preliminary remark:
To every Central build a quicktest_template is assigned (inherited from predecessor), where as quicktest result will be created by either a quicktest preview, or a release. This is the case either by doing it in quicktest module in browser, or by quicktest_api.
But a next sent preview will not change this quicktest_result any more. (This can be verified by get method).
A preview of course never changes build state, doesn't assign and store any result file to the build, and sends quicktest release mail only to current user.
Using the mail parameter, mail can be sent to anybody else.
Method preview expects a xml file, which has to contain at least the above mentioned result-, and if necessary the reason-part.
Content, which are not set in the xml file are taken from the quicktest_result, or if not existing there from the quicktest_template.
Hint recipients are always taken either from the xml file, and if not set there from the quicktest_template.
User must have rights to do a quicktest release

Example:

```
curl -k https://wft.inside.nsn.com/ext/quicktest_api -F "method=
preview" -F "build=LN5.0_ENB_1210_011_90" - F "access_key=1234"
-F "file@qt.xml" -F "mail=x.y@nsn.com"
```

d) release:
Like preview, but build state is set to result value, which is given in xml file. Results as html and xml ReleaseNotes are stored and release mail is sent to either "to"- and "cc"- list in xml file, or if not given here, to the recipients list, set in quicktest_tem-

plate.
User must have rights to do a quicktest release.

Example:

```
curl -k https://wft.inside.nsn.com/ext/quicktest_api -F "method=
release" -F "build=LN5.0_ENB_1210_011_90" -F "access_key=1234" -
F "file@qt.xml"
```

XML Syntax:

```
<quicktest action=.release.>
<template>template id</template>
<result>released|not_released|released_with_restrictions</result
>
<tests><![CDATA[
<p>Html Content can be placed here</p>
]]></tests>
<links><![CDATA[
<p>Html Content can be placed here</p>
]]></links>
<reason><![CDATA[
<p>Html Content can be placed here</p>
]]></reason>
<recipients>
<to>a@nsn.com;b@nsn.com;c@nsn.com</to>
<cc>a@nsn.com;b@nsn.com;c@nsn.com</cc>
<reply_to>d@nsn.com</reply_to>
</recipients>
<subject>Text without HTML</subject>
<message><![CDATA[
<p>Html Content can be placed here</p>
]]></message>
<attachments>
<attachment type=.zip.>true</attachment>
</attachments>
</quicktest>
```

4.2.2.7 Quicktest2 API

Same as for quicktest module 1 (see chap. 4.2.2.6), but valid for release of quicktest module 2.

4.2.2.8 Wrapper Script for various tasks

(see# 9866)
The WFT wrapper script allows using the WFT API in an easier manner. It provides a common script which can be used by all WFT users to access several functions via script.
For most options a file .wft_access_key has to be created in the user's home directory, containing the user's WFT access key. (The user has to take care, that he is the only one who can access this file).

Command syntax
wft -r [rn_path] -k [knife_id] -b [baseline] -p [pronto_id] -a [access_key] [command] -h

4 options can be used:

-k [knife_id]	passes a knife ID to the script

-b [baseline_name]	passes a baseline name to the script
-r [releasenote_name]	passes a release note file name to the script
-p [pronto_id]	passes a pronto ID to the script

After passing an argument you are able to select one of the following commands:

verify_xml	allows you to verify the release note against the currently used release note schema
upload_xml	allows to create a baseline by uploading its release note
update_xml	allows to update the baseline data in WFT by uploading the release note
get_info_ENB	returns a build content as an xml
get_info_knife	returns the knife info as an xml
get_info_fault	returns a fault info as an xml
register_knife	allows to register a knife within WFT

Example:

```
./wft -b LN5.0_ENB_1304_645_00 get_info_ENB
```

Returns the data of eNM build LN5.0_ENB_1304_645_00 in xml format.

4.2.2.9 Collection of some more APIs

Some of these APIs use port 8091 to bypass WAM authentication mechanism.

Some API commands can be used as URL as well as CLI. The relationship looks like:
URL: https://wft.inside.nsn.com/ext/...
CLI: `curl --insecure -g "https://wft.inside.nsn.com/ext/..."`
(hint: take care of the quotation marks!)

Some hints:
- for a get request (data is called from the server) the needed parameters are collected using a "&".
- for a post request (data is sent to the server) the curl parameter "-F" is used to collect the relevant data.
- (the possible parameters can be found within model CustomFilter.)
- the curl parameter "--insecure" is needed using the URL "nsn.com" (not for "nokiasiemensnetworks.com") and additionally depends on the used curl version (newer curl, from version 7.19.7, also does not need it).
- the curl parameter "-g" is needed for correct interpretation of the square bracket ("[" and "]").
- parameters using an additional empty square bracket (e.g. "…&custom_filter[branch_names][]=branch_yxz…") can be used multiple: e.g. "…&custom_filter[branch_names][]=branch_yxz&custom_filter[branch_names][]=branch_abc…" (see also chap. 4.2.2.9.12).

4.2.2.9.1 Add "Change" info

```
curl https://wft.inside.nsn.com:8091/builds/RP_1304_004_00/
placeholders/set.xml -F "key=changes" -F "content=the changes
you want to describe"  -F "access_key=1234"
```

4.2.2.9.2 Add "Faults found during RP testing" info

```
curl https://wft.inside.nsn.com:8091/builds/RP_1304_004_00/
placeholders/set.xml -F "key=faults_found" -F "content=nothing
found"  -F "access_key=1234"
```

4.2.2.9.3 Set the build state to "Not Released"

```
curl https://wft.inside.nsn.com:8091/builds/RP_1304_004_00/
unrelease.js -F "sender=template" -F "to=rp-sw-build-external-
release@mlist.emea.nsn-intra.net" -F "cc=" -F "subject=Radio
Platform RP_1304_004_00 is not released" -F
"content=RP_1304_004_00 is NOT RELEASED" -F "access_key=1234"
```

Release Mails will be sent automatically by WFT as soon as the “AutoRelease” mechanism is activated (precondition: the baseline is set to category "SC").

4.2.2.9.4 Dynamically generation of XML and HTML release notes

Using the URL https://wft.inside.nsn.com/builds/[build name]/html| xml a HTML as also a XML release note can be dynamically generated (independent from the build state). (e.g. https://wft.inside.nsn.com/builds/RP_1308_001_00/html).

4.2.2.9.5 Get a list of all (e.g. LN6.0) eNB in XML format

(see #9634)
https://wft.inside.nsn.com/ext/builds?custom_filter[baseline]=LN6.0&custom_filter[category]=1&custom_filter[sorting_field]=prod_date&custom_filter[sorting_direction]=DESC&custom_filter[items]=200
or
https://wft.inside.nsn.com/ext/builds?custom_filter[baseline]=LN6.0_ENB&custom_filter[sorting_field]=prod_date&custom_filter[sorting_direction]=DESC&custom_filter[items]=2000
or with additional filter for RELEASED builds:
https://wft.inside.nsn.com/ext/builds?custom_filter[baseline]=LN6.0_ENB&custom_filter[sorting_field]=prod_date&custom_filter[sorting_direction]=DESC&custom_filter[items]=2000&custom_filter[state][]=released
or with additional filter for RELEASED or NOT_RELEASED builds:
https://wft.inside.nsn.com/ext/builds?custom_filter[baseline]=LN6.0_ENB&custom_filter[sorting_field]=prod_date&custom_filter[sorting_direction]=DESC&custom_filter[items]=2000&custom_filter[state][]=released&custom_filter[state][]=not_released

4.2.2.9.6 To update the "ECL compatibility" for a given baseline

(see #10514)
Has to be done in two steps.
First you need to retrieve the ID for the new ECL_SACK_BASE
https://wft.inside.nsn.com:8091/ext/build_content/LN7.0_ECL_SACK_BASE_1401_110_82
=> ID = 159361

In the second step you need to mark your baseline to be compatible to this ECL SACK BASE by executing the following line:

```
curl https://wft.inside.nsn.com:8091/builds/LNT4.0_LTE_SCM_
1311_005_00/ecl_baselines -F "ecl_baseline[ecl_build_id]=159233"
-F "access_key=1234"
```

4.2.2.9.7 Delete knives

(see #10555)

```
curl https://wft.inside.nsn.com:8091/knife_requests/KNIFEID -X
"DELETE" -F "access_key=1234"
```

e.g.

```
curl https://wft.inside.nsn.com:8091/knife_requests/123456 -X
"DELETE" -F "access_key=1234"
```

4.2.2.9.8 Release PS load with cmd increment

Parameter sub_build[] and branch_for can be repeated as often as needed.

```
curl https://wft.inside.nsn.com/ext/api/increment/MB_PS_REL_
2014_02_011 -F xml_releasenote_id=33 -F parent=MB_PS_REL_2014_02
_010 -F sub_build[]=LNT2.0_PS_DSPHWAPI_BUILD_2011_07_46 -F
sub_build[]=TI_CGT_ 7_3_8_BL-GENERIC -F "branch_for[]=WMP Main"
-F access_key=1234
```

4.2.2.9.9 Release for Quicktest Module1

```
curl https://wft.inside.nsn.com/ext/quicktest_api -F "method=
release" -F "file=@filename" -F access_key=1234
```

4.2.2.9.10 Release for Quicktest Module2

```
curl https://wft.inside.nsn.com/ext/quicktest2_api -F "method=
release" -F "file=@filename" -F access_key=1234
```

4.2.2.9.11 Setting of repositoryUrl/repositoryBranch/repositoryRevision

```
curl -k https://wft.inside.nsn.com:8091/builds/MB_PS_REL_2014
_04_007 -X PUT -F "build[repository_branch]=MB_PS_REL_
2014_04_007" -F "build[repository_revision]=123456" -F
"build[repository_url]=https://svne1.access.nsn.com/isource/svnr
oot/BTS_D_PS_REL_2014/tags/MB_PS_REL_2014_04_007" -F
"access_key=1234"
```

4.2.2.9.12 How to get the latest baselines (of a branch)

1. latest baselines:
https://wft.inside.nsn.com/ext/builds?custom_filter[baseline]=DND0.0_ENB&custom_filter[sorting_field]=delivery_date&custom_filter[sorting_direction]=DESC&custom_filter[items]=200

2. latest baselines belonging to a certain branch:
https://wft.inside.nsn.com/ext/builds?custom_filter[baseline]=LN5.0_ENB&custom_filter[branch_names][]=WMP_FB13.04&custom_filter[sorting_field]=delivery_date&custom_filter[sorting_direction]=DESC&custom_filter[items]=200

4.2.3 XML Release Note Import

XmlController
lib/xml_note.rb
lib/releasenote_schema/*

The content of every Build can be described with an XML file (also named XML Release Note). The tags to be used within this XML file are defined within the XML-Releasenote Specification (see /2/), LTE specifics can be found in /3/.
XML schemas define a parse able XML format. For the various versions of the XML schemas and for different types (such as e.g. "SC", "PS_REL", or Central Builds) exist XML Release Note templates (see also chap. 3.12.8).
An XML file matching this schema contains the information for one specific baseline in a predefined format.

Baselines of category "External" import (upload via the API, see chap. 4.2.2) their content using such an XML file in a special manner to get the content processed and evaluated. All information from the XML file will then be imported into the database. Baselines of this category can be set to state "Testing" using the "testing-flag".

For baselines of category "SC" or "Central" such an XML Release Note is generated by WFT during state change to "Released" resp. "Released for Quicktest".

4.2.3.1 XML-Import: "basedOn"

Rule: for importing a new baseline the parameter "basedOn" is mandatory (see /3/).
During the XML import it is checked if this predecessor (the "basedOn" baseline) is existing.
In case there is no or no correct "basedOn" the error massage "basedOn is mandatory and must be a released version in WFT" is sent and the import fails.

For creation of a new build, which has up to now no predecessor within WFT (e.g. a first build of a brand new baseline, or a first build of a new feature build) this check accepts a "basedOn" information containing the string "000_00" (see also /3/).

4.2.3.2 Validation, Import/Download and Update of an XML Release Note

Before importing an XML Release Note it is validated. These checks are the same as they can be done manually by the user (for details see chap. 3.10.1).

If the correct version (defined by load coordinators) is chosen for doing the relevant import is checked by importing rules.

Import of an XML Release Note can be done using the "xml" method of the WFT API (see chap. 4.2.2) or can be performed via GUI using the "XML Import" (see chap. 3.10.2).

An already downloaded XML Release Note can be updated later on. Baselines of category "SC" do simply an import once more.
Whereas "External" baselines have to do an additional upload via API (see chap. 4.2.2) using the update flag.
Hint: For the first import (the baseline still doesn't exist) it is not allowed to use the flag "update=yes" (e.g. for prophylactic reasons in a user scripting, see #8614).

4.2.3.3 Baselines not handled within WFT

Baselines which are listed within the XML file of an SC build (tag <baselines>), but which are not supported by WFT are displayed within the "Baselines" tab of that build only for information (and therefore no link to that baseline is available; part of #7789; e.g. see "FTM_DND10_274.00").

4.2.4 MySQL

MySQL is a relational database management system. The MySQL database contains all data (production as also process) from the Workflow Tool.
MySQL was chosen because of its propagation for web.

4.2.5 Nginx

Nginx is the frontend web server and passes all dynamic requests through to the passenger module.

4.2.6 Phusion Passenger

Phusion Passenger is a module (plug-in) for the Nginx web server for running of Rails web applications. It is available as a Gem package and allows easy deployment of

Ruby on Rails applications. There is no Ruby (on Rails)-specific server configuration required.
Using the Phusion Passenger the Nginx should never crash, even in case of crashing Rails applications.
The Phusion Passenger distributes the different requests of the users to the various processes of the WFT application.

4.3 Internal Interfaces

4.3.1 Task Runner

The Task Runner is a ruby script which is responsible for executing all tasks in the context of a specific UNIX user. It is used to execute tasks associated to baselines and it can run for several times as different UNIX user. This allows the tasks to be executed as the particular user.
The Task Runner communicates with the Workflow Tool across the database. It has a database connection and is performing periodically searches for tasks in state STARTED. If the Task Runner founds such a task it picks up all necessary information and starts an instance of the associated script. This way it’s possible to execute unlimited number of tasks simultaneously.
Once the executed script finishes the exit code is interpreted and the task result is written back to the database.

4.3.2 Background Worker

The Background Worker is a self written script (which replaces the formerly used BackgroundRB) and performs all actions which need to be done periodically. It is responsible for the following tasks:

State Switch PLANNED->ANNOUNCED
Builds in state PLANNED are switched to ANNOUNCED once the freezing reminder date is reached.

State Switch ANNOUNCED->BUILDABLE
Builds in state ANNOUNCED are switched to state BUILABLE once all baselines are marked as "delivered".

State Switch BUILDABLE->FROZEN
Builds in state BUILDABLE are automatically FROZEN if the check_before_freeze flag is not set.

Task State Calls
Some tasks do not execute scripts directly. e.g. Hudson Build.
For these tasks no exit code is available and some status fetching mechanism has to be triggered periodically.
For the Hudson Build this is done by using the Jenkins API (API provided by Jenkins).

Autorelease
Central and SC Builds can automatically be released once they switched to state SUCCEEDED and the auto_release_flag is set.
The Background Worker switches the build to RELEASED FOR QUICKTEST (central builds) or RELEASED (sc builds) and sends the release note for those builds.

4.3.3 State Machine

During a build's lifecycle the software build moves through several predefined states. There are predefined state changes that will be triggered by a user, a cron-job or an external tool via a curl call. To track a builds state history, each state change is stored in the database.
For more detail about the states / state changes of the build process refer to chap. 3.3.8.

4.4 Interfaces to other Systems

4.4.1 Subversion

The data of the SCs as also the build results are stored within the VCS Subversion (SVN). Fetching this data out of SVN or importing data into it is supported by WFT (see chap. 3.3.6.7).

4.4.2 Pronto Tool

4.4.2.1 Access to Pronto Tool

Due to missing API in Pronto Tool fetching fault information (also called "pronto") is done using simulating a browser call. For faster interaction the pronto information is cached in WFT database and only the missing information is directly fetched from the Pronto Tool.
Before starting new builds, in case of an XML for an SC baseline is uploaded or the user presses the "Refresh" button on page "Fault Details" all associated faults are marked for update. The Pronto Updater, a ruby script, looks every 60 sec if there are faults marked for update and if yes, the current information is taken from the Pronto Tool.
The faults can be assigned to one or more builds.

4.4.2.2 Feature List export to Pronto Tool

Periodically - currently twice a month - WFT delivers (send by a mail) a Feature List (generated by a cron job out of the imported LDR) of all active features to I&V Fault Management which then will be imported (by I&V Fault Management) into Pronto Tool.

4.4.3 Hudson / Jenkins

Hudson / Jenkins is an extendable, web based system for CI (Continuous Integration). It supports different build tools and different VCSs.
The WFT starts all Central Builds via Hudson whereas Jenkins is used within the development.

4.4.4 Nagios

Nagios is an open source software application for network monitoring. It watches services and alerts users when things go wrong and again when they get better.
Nagios (http://ulmon01.emea.nsn-net.net/ulm/check_mk/) monitors the system performance and load of the Workflow Tool.

Following services are checked by Nagios:

1. General Checks (CPU load; CPU utilization; Disk Io; Disk usage: / /boot /opt /tmp /usr /var/*; Mount options; NTP; processes (must running) Cron, nscd; Zombie processes; TCP connections; uptime; amount of users) on:

ullteb30, ullteb31, ullteb42, ullteb43, ulwfttest1-4, ulwft, bewftsql1, eswftsql1, ulwftsql1, ulwftsqlmgr, ulwftsupport (not on esworkflow, as no access possible)

2. Special checks:

- ulwftsupport	http (test if redmine answers on port 8080)
- ulwft, esworkflow, ullteb30, ullteb31	https (test if WFT answers on port 443)
- ulwft	SSL certificate
- ullteb30,ullteb31	(http)-check if page /info contains the string "Build Information" (1)
- ullteb30,ullteb31	(http)-check if page /knife_requests contains the string "Listing Knife Requests" (1)
- ullteb30,ullteb31	(http)-check if page /builds contains the string "Central builds" (1)

(1) these checks are checking if WFT is still alive.

3. Passive checks (host has to send messages within a certain timeframe)

ullteb43	background_worker wftul	5 min
ullteb43	taskrunner lteman	10min
ullteb43	taskrunner tdlteman	10min
ullteb43	taskrunner fzmscm	10min
ullteb43	taskrunner pronto_preinfo	10min
ullteb43	scheduler	(not yet active)

The state of all WFT hosts can be seen here (user: guest; pw: guest):
http://ulmon01.emea.nsn-net.net/ulm/check_mk/view.py?view_name=hostgroup&selection=f009fced-319b-40f4-bcd4-bf993c9cacdf&hostgroup=Workflow

4.4.5 LDR Import

The LDR (LTE Data Repository) is an xlsm file, which is stored in IMS (https://sharenet-ims.inside.nsn.com/livelink/livelink?func=ll&objaction=overview&objid=488381395) and is used to fetch feature information of a build for release note generation (for details refer to chap. 3.17.2).
This document is generated automatically from numerous documents (i.e.: R&D Back-log, I&V Backlog, Feature Build Plan and more). A list of all documents and their versions which were used for creating the current LDR can be found behind the tab named "DRIS" within the LDR (users are not allowed to change data directly in LDR, they can change them only in the documents which will later be used for generating the LDR).
The content and format of the LDR is expected to be working: Therefore no validation from WFT side will be done.

For performance purposes (see #12139; the amount of data to be imported has decreased from 12GB to 0.5GB) only some sheets are used from this workbook and they are converted by LDR maintainer to CSV format. Users can also access them in the IMS:

1. LDR_data.csv (containing features from Feature Build Plan):
   https://sharenet-ims.inside.nsn.com/livelink/livelink?func=ll&objaction=overview&objid=521313697

2. LDR_History.csv (used to fetch version of LDR):
   https://sharenet-ims.inside.nsn.com/livelink/livelink?func=ll&objaction=overview&objid=521321441

3. DRIS.csv (used to fetch version of documents used to generate LDR):
   https://sharenet-ims.inside.nsn.com/livelink/livelink?func=ll&objaction=overviewversion&objid=521332662&vernum=793

The import is handled with the following steps:

1. Download all three CSV files from IMS by webDAV.

2. After download theses files are converted from Windows format to Linux format.

3. DRIS.csv (used to fetch version of documents for generating LDR):
   https://sharenet-ims.inside.nsn.com/livelink/livelink?func=ll&objaction=overviewversion&objid=521332662&vernum=793

a. Features are imported to database from LDR_Data.csv.
   For every feature (equivalent to one row within the LDR) the following values are stored within the DB, and are later used to create the eNB release note:

a)	Feature or Subfeature	ID of a feature
b)	(FP) Feature or Subfeature Title	title and description
	(feature description is limited to 255 characters. All content beyond this limit is truncated.)	
c)	COMMON DEV STATUS	feature development status
d)	WMP Release	1)
e)	WMP Release info	1)
f)	TDD Release	1)
g)	TDD Release info	1)
h)	DCM Release	1)
i)	DCM Release info	1)
	1) this release field informs in which system release the feature occurs.	
j)	PNS show / no show	
k)	Common FB plan	version of Common FB plan
l)	Level	level of the feature (System and Entity level features are from interest; Module and Development level features are skipped)
m)	Feature Lifecycle	current life phase of the feature

The import is performed every hour. It is triggered by a crone job which runs on ullteb30. If for some reason there is the need for an earlier import, there exists the possibility to start it manually.
In case of the user edits a feature he gets a specific formula (see also chap. 3.13.1.3) where the import can be triggered. With this formula it is also possible to request an email containing all LDR features.

4.5 Communication Interfaces

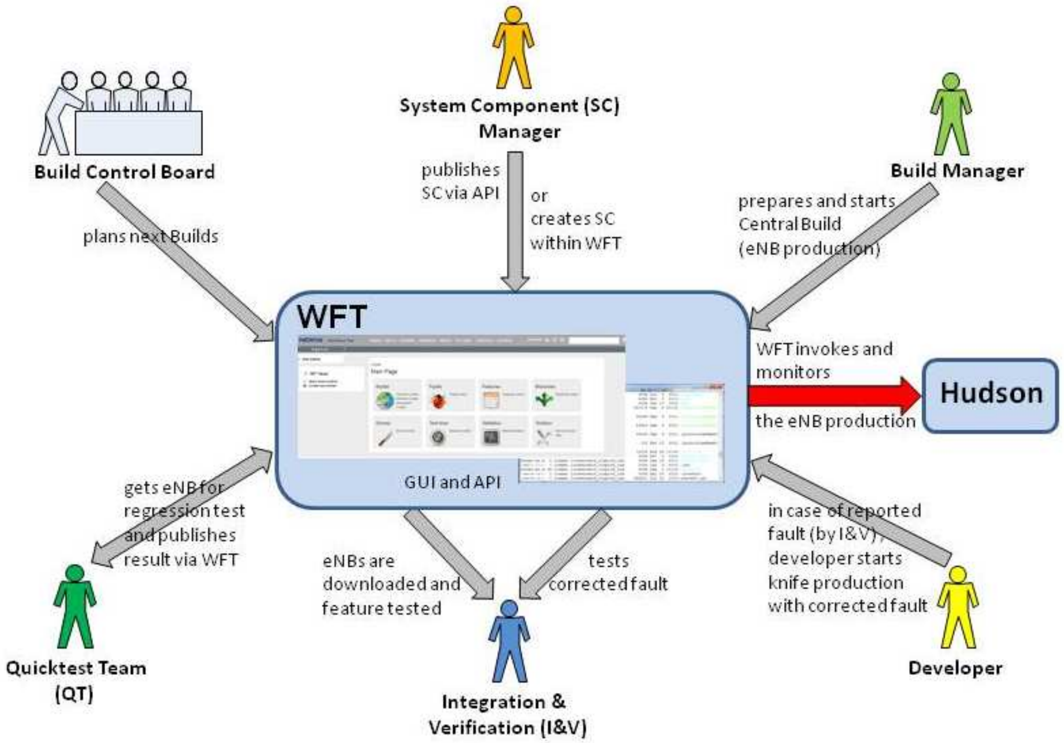

System Component (SC) Manager
publishes SC via API
or creates SC within WFT
Build Control Board
plans next Builds
Build Manager
prepares and starts Central Build (eNB production)
WFT
WFT invokes and monitors
the eNB production
Hudson
GUI and API
gets eNB for regression test and publishes result via WFT
Quicktest Team (QT)
eNBs are downloaded and feature tested
tests corrected fault
Integration & Verification (I&V)
in case of reported fault (by I&V), developer starts knife production with corrected fault
Developer

Figure 10: Communication Interfaces of the Workflow Tool to Build Process

5 Data Model

Due to continuous improvement of that tool the current data model will be displayed online.
The database consists currently of more than 150 tables.

With ticket #11795 the actual Entity Relationship Diagrams of the WFT model will be provided. This ERD will be generated after every deployment of the WFT and will be found here:
https://wft.inside.nsn.com/WFT_ERD.jpg
https://wft.inside.nsn.com/WFT_ERD.pdf

6 Set-up of a new Project

Within this section it is described what has to be done in which sequence in order to setup a new project:

- Definition of the naming conventions for the new Project in a consistent manner.
- Definition of the SW entities (deliverer) in the SW Delivery and Build Process.
- Definition of the roles of the actors in the SW Delivery and Build Process.
- Definition of the templates used for the SW Delivery and Build Process.
- Definition of branches.
- Definition of storages, where results are stored in order to download them.
- Definition of the build scripts.
- Definition of the servers, where the scripts shall run.
- Definition of accounts, which shall be used for the build scripts.
- Definition of the entity behaviour and entity dependencies

Already introduced products to WFT.

- LTE:FDD and LTE:DCM were the starting products, which were setup within the WFT.
- FDD:TDD is the third product, which is already setup in WFT.

6.1 Naming Conventions

WFT by its own does not distinguish explicitly between different products. Therefore use as far as possible consistent naming conventions to mark all entry names belonging to one specific product.

Following groups have to be defined.

- Names of the SW entities (deliverer).
- Names of the user groups (for the different roles).
- Names of the templates which are used in the Delivery and Build Process.
- Names of the branches (which are e.g. used for maintenance builds).
- Names of the baselines, which represent the concrete delivery (with version numbers).

Even though it is not a must, that these names have to be define consistently, it is very strong recommended.
A must is only the uniqueness of names within a list of entries (e.g. the branch list).

For new products use always the project name as prefix.
E.g. for project TDD use "TDD", or "TDD:" or "TDD_"

6.2 Deliverers

A list of all deliverers, which are needed to build a load, have to be provide to the WFT administration.
For the definition of a deliverer see chap. 3.12.1., the definition of the deliverer categories can be read within chap. 3.3.

Hint: The dependencies (which delivery belongs to which deliverer category) are defined as last step within the project setup.

6.3 Roles and permissions

All users, who contribute to the SW Delivery and Build Process have to get permissions, which enables and limits them to their assigned roles.
Only a member belonging to the permission group "Administrator" is able to configure WFT for a new project line.

Although it is possible to assign each of the 3 kinds of permission (for general information about WFT permissions see chap. 3.13.2) directly to each user of the WFT, it is strongly recommended to do this only via groups.

Therefore for a new project the roles and requested groups with appropriate permissions have to be defined and forwarded to the WFT administration. Keeping the naming conventions has to be done by the WFT administration.

6.4 Templates

Templates are used in the SW Delivery and Build Process for following purposes.

- Templates for HTML Release Notes (see chap. 3.12.7).
- Templates for XML Release Notes, which are generated within WFT (see chap. 3.12.8).
- Templates for Config Files (see chap. 3.13.2).
  (the Config Files are used in all three LTE Products to provide the build scripts in central build with the appropriate baseline names and the location, like share or SVN, where they can be found).

6.4.1 Templates for HTML Release Notes

For a new project HTML Release Note Templates have to be requested for each SC (e.g. LTETDD:LOM) which is done within WFT, for central build and for quick test.
Keeping the naming conventions has to be done by the WFT administration.
The components of an HTML Release Note Template are described in chap. 3.12.7.
Using major domo email lists for all receivers and all informed ("CC") of the Release Notes will easy the handling.
The request have to be sent to the WFT administration. The WFT administration will then generate such a Release Note template.

6.4.2 Templates for XML Release Notes

XML Release Notes are generated in parallel to HTML Release Notes (see also chap. 3.13.2). The XML Release Note templates are supplied by the WFT administration.
In case of there is the need for a new XML Release Note Template the WFT administration will generate such a template.
The components of an XML Release Note Template are described in chap. 3.12.8.

6.4.3 Templates for Config Files

As mentioned within chap. 3.13.2 Build Configurations, the Config Files are necessary, in case of entities like SC's or Central Build are build within the WFT using build scripts (called via tasks).
These build scripts need several information, e.g. about the contained baselines.

Needed Config Files have to be requested from WFT administration, which will then generate them.

The components of a Config File Template are described in chap. 3.13.2 Build Configurations.

6.5 Branches

The general usage of branches is described within chap. 3.13.1 Branches. Branches needed for new products have to be requested by the WFT administration.
Naming conventions should be: "[product] [version/feature build] [additional info]", e.g. WMP FB10.9 PCD2.2, WMP FB11.03 Korea Lab Trial, TDD FB94.05.

Hint: Branches within WFT need not to have the same naming as branches in SVN, but can correspond to them.

6.6 Storages

Storages are used to store the results of the central build. For detail refer to chap. 3.12.9.
The WFT administration has to define these storage places.

6.7 Build Scripts

Build Scripts are used to do some processing on a special platform, e.g. compiling and linking sources, or make a tar file and so on.
Scripts are started via tasks by WFT.
These tasks have to be adapted by the WFT administration in order to start and communicate with the script. Communication is done via API (see also chap. 0 Interfaces).

6.8 Build Account

The account(s) under which the scripts shall be executed have to be communicated to the WFT administration.
A corresponding entry will be created by WFT administration and supplied with an access_key, which enables the scripts running under this account to communicate with WFT via API.
This access_key will be communicated by the WFT administration to the requestor.

6.9 Build Servers

If the build uses build scripts, it must be specified, where these scripts shall be executed.
Therefore the WFT administration has to be informed which servers are available to do that job.
If there are more than one server available, and if it isn’t important where the scripts are running, than these servers can be grouped in order to select one of them randomly.

6.10 Build Behaviour and Build Dependencies

As already mentioned within chap. 3.3 Builds, there exist three different types of builds: Central builds, SC builds and External deliveries.

When setting up a new project it has to be defined, as one of the first tasks, to which of these types the concerned build has to belong to.

Creating the first concrete baselines for a build (see chap. 3.3.10.2) it has to be chosen to which of these types the build has to belong to. The user can only select these types for which he has got the relevant permissions.

Also the dependencies between different builds are defined, when creating the first baseline of a build.
E.g., if creating the first baseline of a Central Build, all deliverers which are necessary have to be inserted into the baselines list.
And only if there are already baselines created for these deliverers, also these concrete baselines are selectable.

This procedure has to be done with all those builds, which are handled within WFT (SC and Central Build). External Deliveries can be created within WFT too, but are usually created by using an XML Release Note via the API.

7 Testability

Rails implements a full featured testing environment to perform testing of models with unit tests, methods with functional tests and testing complete activities with integration tests.

For testing WFT 3 different frameworks (to be installed via a gem) are used:
1. **RSpec** tests the code itself. TCs are file specific.
   (TCs reside within /spec)
2. **Cucumber** has to be used for functional test cases (TCs), "testing from outside WFT".
   (TCs reside within /features)
3. **Factory Girl** creates needed dummy or test data.
   (TCs reside within /spec/factories)

7.1 RSpec

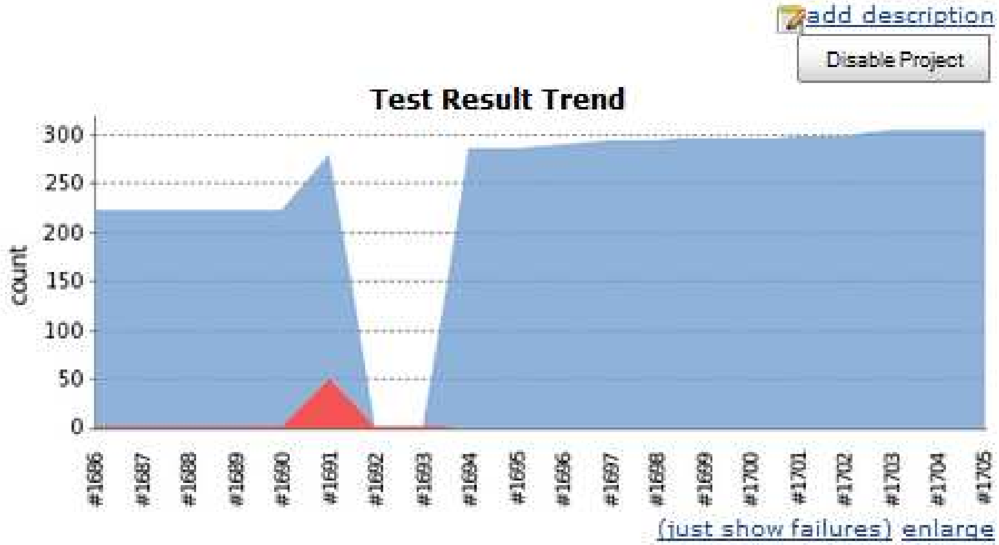
RSpec is a test-framework for Ruby. More detailed information can be found within http://rspec.info/.

Example for a TC (branch_type_spec.rb):

```
require 'spec_helper'

describe BranchType do

  it 'is not deletable when assigned to any branch' do
    branch_type = BranchType.create(delivery_type: "Other")
    branch = FactoryGirl.create(:branch, branch_type:
branch_type)
    branch_type.reload
    branch_type.deletable?.should == false
  end

  it 'returns a list for select boxes' do
    branch_type1 = BranchType.create(delivery_type: "Other")
    branch_type2 = BranchType.create(delivery_type: "Common")
    BranchType.all_for_select.should
eq([["Select...",nil],[branch_type2.delivery_type,branch_type2.i
d],[branch_type1.delivery_type,branch_type1.id]])
  end

end
```

The implementation status of the RSpec test cases can be found within Redmine using the link https://wft.inside.nokiasiemensnetworks.com:8080/projects/wft/test_suite.

A TC statistic (diagram; Firefox is recommended) can be found here:
http://ulwfttest1.emea.nsn-net.net:8080/job/01%20WFT%20Develop/



(x-axis: build no. / y-axis: count of TCs)

7.2 Cucumber

Cucumber is a tool that can execute plain-text functional descriptions as automated tests.
So called "features" (in a certain format) have to be defined.
Every scenario consists of a list of steps, which must start with one of the keywords **Given**, **When**, **Then**, **But** or **And**.

Test routines for the whole Application will be written to be able to ensure all features are still working when updating to a new version.
More detailed information can be found within http://cukes.info/

How to install and work with Cucumber can be found here:
https://wft.inside.nokiasiemensnetworks.com:8080/projects/wft/wiki/Test_Driven_Development_(using_cucumber)

7.3 FactoryGirl

FactoryGirl is a library containing various methods for setting up Ruby objects as test data. FactoryGirl has to be used in order to fill DB with test data.

For more details, please see http://www.rubydoc.info/gems/factory_girl/1.3.3/frames.

7.4 Test case coverage

Every own written method (within /models and /controllers) and every view has to be tested with at least one RSpec test case (scenario).

8 Development Support

8.1 Ticket tracking with Redmine

Redmine is a project management web application for issue tracking and includes a wiki, forums, email notification, Gantt chart, etc. (for details see www.redmine.org). It supports the WFT development by delivering a planning tool (for new features, improvements and bugs) as also with issue tracking of support requests.

8.1.1 States within the ticket system

For all issues handled for WFT within Redmine the following states have been defined:

(1) New — Users create tickets with Tracker Support, Bug, Feature or Improvement.

(2) In Progress — The work on the ticket began. [1]
(For Tracker Bug, Feature or Improvement the ticket is assigned to a defined version (see chap. 8.1.2).)

(3) Ready — The work on the ticket is finished. [1]
(For Tracker Bug, Feature or Improvement the code is deployed.)

(4) Rejected — Tracker Support:
Ticket rejected as no (more) relevant, or no WFT issue.

Tracker Bug, Feature or Improvement:
The ticket won‘t be implemented, or the bug couldn‘t be reproduced.

The following states are exclusively for Tracker Bug, Feature or Improvement:

(5) Pending — Ticket clarification is ongoing and requests additional information from the user. Maybe approval of other users is needed as they could be affected by the change.

(6) Accepted — The ticket is ready to be implemented.
(Ticket will be stored within the "Backlog" and will next be assigned to a version).

(7) Coding finished — The implementation is finished and committed.

[1] Hint: In case of an answer to an already closed ticket the ticket gets again into state "In Progress". To have the possibility to close this ticket again the direct status change "In Progress" -> "Ready" is enabled.

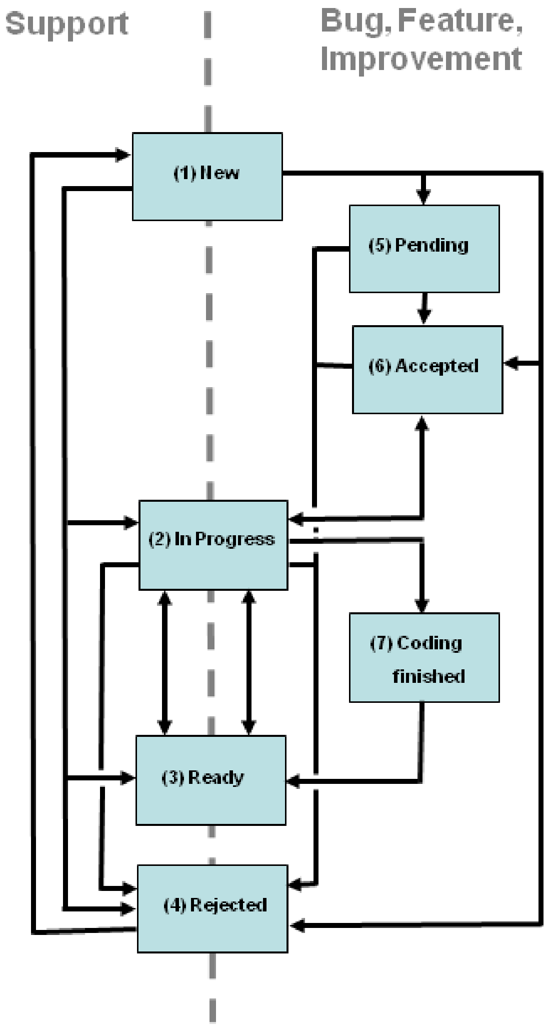


Figure 11: Redmine states for Tracker Support and Bug/Feature/Improvement

The tracker "Action Item" exists for tracking WFT internal issues which are neither related to a version nor need code adaptation (status handling is equal to tracker "Support").

8.1.2 Version handling

WFT follows a rudimentary SW development process (see chap. 8.1.3).
The defined version has to be tracked using the Gantt Diagram offered by Redmine (Prerequisite: do not install the "BetterGantt Chart" Plugin).
Versions are defined for a one or two week timeframe.
All incoming features / improvements / bugs are collected within version "backlog".

Each issue (feature / improvement / bug) must have a version, has a begin date, which is identical or older then the begin date of the version and must have a due date, which is identical or younger then the end date of the defined version.

8.1.3 SW development process

To focus on quality the following simplified SW development process is defined:

1. **Collecting** the incoming features / improvements / bugs within Redmine.
2. **Feature-planning**: packetizing the features in defined (small, i.e. to be implement-ted within one or two weeks) packages.
   (bug fixing has to be prioritized and perhaps features have to be shifted because of that)
   - proceeding: all new feature requests are collected within a version "backlog" and every 2 (1) weeks it is decided (within the WFT internal meeting) which out of all these features have to be implemented for the next release.
   - no need for an strict release time schedule, can be handled flexible (in a weekly range) and has to be handled depending on the amount of features or on the priority of the feature(s).
   - preferred timeframe: every **two weeks** (or in urgent case one week) <u>one</u> new and defined release.
     (exception: urgent bug fixes or very important und urgent features can be done beside the weekly releasing within a special interim or bug fixing release!)
3. **Commit** <u>Thursday</u> EOB
4. **Presentation of next feature packages** within WFT Development Meeting.
5. **Regression testing** on <u>Friday</u>.
6. **Releasing** (deployment) has to be done on <u>Monday</u> morning, **latest at 10:00am** (so that time left for bug fixing! Also the releasing of an interim release has to be done <u>only</u> at the morning).
7. Every release gets an own Releasenote,
   which has to be stored (e.g. as a simple text file) and has to be visible (menu Help) within WFT.

8.2 Error tracking with Errbit

All errors caused by WFT users ("SWW", "Something went wrong") are collected and same errors are summarized within Errbit (http://ulwftsupport.emea.nsn-net.net:9090/, access possible only without WAM; read access using WAM: https://wft.inside.nsn.com:9090/).

The user gets with each SWW an error number on which he can refer. This error number ("ErrorID") is displayed during the creation of the new ticket within Redmine (see chap. 8.1) as soon as the user uses the link "You can create a ticket here".
The WFT administration has the possibility to create a Redmine ticket out of Errbit.

8.3 RubyMine

RubyMine is a Ruby on Rails IDE (for details see www.jetbrains.com/ruby/).

9 Terms and Abbreviations

9.1 Abbreviations

API	Application Programming Interface
BTS	Base Transceiver Station
CB	Central Build (Team)
CC	ClearCase
CI	Continuous Integration
CPD	Continuous Product Development
DCM	DoCoMo
ECL	Environment Control List
eNB	enhanced Node B
FB	Feature Build
FBP	Feature Build Plan
FDD	Frequency Division Duplex
ffs	for further study
I&V	Integration and Verification
LBTS	LTE BTS
LDR	LTE Data Repository
LinSEE	Linux-SEE (SW Engineering Environment)
LTE	Long Term Evolution
MVC	Model-View-Controller
NPF	Network Planning File
PDDB	Parameter Dictionary Database
PMC	Panasonic Mobile Communication
PR	"Pronto" (fault number)
PS	Platform Services ("PS_REL")
RISE	Reference Information Service
RoR	Ruby on Rails
SC	System Component
SCBM	System Component Build Manager
SCM	SW Configuration Management
SCT	System Component Test
SSO	Single Sign on
SVN	Subversion
tbd	to be defined
TDD	Time Division Duplex

WAM	NSN Web Access Management
WFT	Work Flow Tool
WMP	World Market Product

9.2 Definitions

branch

A branch describes the various development lines of a SW, e.g. for different products (e.g. WMP or DCM) and/or for different versions, i.e. feature builds (FB).

component

See deliverer.

deliverer

Set of baselines, sometimes also called "component".

ECL

Definition of ECL, see https://swikis.inside.nsn.com/bin/view/LTECI/ECL and https://confluence.inside.nsn.com/display/ECL/Home

ECL_SACK_BASE

Baseline containing the ECL definition

fallback

A minor SW version of a sub-baseline has to be used within the super ordinate build (due to various reasons).
In case of a fallback this has to be regarded for e.g. no longer corrected faults, etc.

FEP-File

Tar-file with an own header used for DCM.

knife

A knife is a kind of a patch, it is an eNB load containing the corrected SW parts. It is built on base of an existing eNB load including the corrections, which have to be tested.

PDDB

Parameter Dictionary Database is a back end application for managing parameters. It is used by System Verification.
One specific PDDB version is related to an specific object model version.
Details can be found here:
https://inside.nsn.com/global/company/organization/cdo/pe/cpe/documentation%20solutions%20excellence/cudo/documentation_databases/pages/riseandpddb.aspx.

pronto

Synonym for a fault stored within the Pronto Tool.

RISE

Reference Information Service is a back end application for managing alarms, counters, measurements and KPIs.
Details can be found here:
https://inside.nsn.com/global/company/organization/cdo/pe/cpe/documentation%20solutions%20excellence/cudo/documentation_databases/pages/riseandpddb.aspx.

9.3 List of Figures

10 References

/1/ *WFT User Manual*
https://sharenet-ims.inside.nsn.com/Open/D481174345

/2/ *XML-Releasenote Specification*
https://sharenet-ims.inside.nsn.com/Open/D426376045

/3/ *XML Release Notes - LTE specifics*
https://bts.inside.nsn.com/twiki/bin/view/PreIntegration/PO_XmlRn

/4/ *eNB Build and SC baseline Naming Rules*
https://bts.inside.nsn.com/twiki/bin/view/PreIntegration/PO_BuildNamingRules

/5/ *Supported/Unsupported Features in Release Notes (RN)*
https://bts.inside.nsn.com/twiki/bin/view/PreIntegration/PO_FeaturesRn

/6/ *Superior BL has to be listed by an SC in its RN*
https://bts.inside.nsn.com/twiki/bin/view/PreIntegration/PO_RN_Inheritance

/7/ *ECL-Check (defined rules)*
https://bts.inside.nsn.com/twiki/bin/view/PreIntegration/PO_ECL_Check

11 Revision history

Version	Date	Modified by	Main changes
1.0	31.05.11	A. Hutstein C. Rolny W. Elster	first issue
2.0	18.02.14	A. Hutstein C. Rolny W. Elster	new Issue after upgrade to Rails V3 ("WFT 2.0"); (WFT release state: v2.14)
3.0	17.07.14	W. Elster	- major changes: - build state changes adapted - introduction of PS_Rel Fast-Track mechanism - chap. 8.1.1: state change adapted - several minor textual adaptations - adaptations for WFT 3.0
3.1	24.11.14	W. Elster	adaptations according (WFT release state: v3.9): - new naming conventions according LTE2157 - #11793 "Delivering of API description per WFT page" - #11847 "Permissions to modify build config templates" - #12139 "Improved LDR import" - #12143 "Definition of several baseline configuration sets" - chap. 7 "Testability" reworked

End of Document