Klaros-Testmanagement User Manual

Version 5.7.3

Publication date May 08 2025

verit informationssysteme

Klaros-Testmanagement User Manual

by Talal Arif, Sabrina Gidley, Fabian Klaffke, Claudia Könnecke, Klaus Mandola, Patrick Reilly, and Torsten Stolpmann

Version 5.7.3

Publication date May 08 2025

Abstract

This document serves as the reference documentation for the Klaros-Testmanagement application. It gives a detailed description of the user interface, and the provided functionality.

Trademarks

Table of Contents

List of Figures

List of Tables

List of Examples

Chapter 1. Feature Overview

1.1. Test Data Management

Management of Test-related Objects
All data of a test project like *test cases*, *test suites*, *test requirements*, *test environments*, *test objects/(SuT)*, *test runs* and *test case results* are stored together in a single database. The administration, processing and evaluation of the data is performed via a modern and comfortable web interface.

Agile and Classic Processes
In addition to classical development processes such as the waterfall or V-Model, iterative and agile methods such as Scrum or Kanban are also supported.

Requirements Coverage
Test requirements can either be managed directly in Klaros-Testmanagement or synchronized with external sources such as JIRA. The current requirements coverage can be displayed at any time.

Modularization
Frequently recurring test steps can be combined into segments and can be reused. Test steps can be inserted as in a modular system and maintained with minimal effort.

Sharing
Test cases and test suites can be centrally maintained and used simultaneously in several projects. Changes are only necessary in a single location and will be propagated to selected projects.

Tracking Changes
For a seamless tracking of all changes applied to the managed data, such as test cases and test steps, all changes are logged automatically and displayed in a detailed change history.

User-Defined Custom Fields
Each test project can be configured individually using custom fields. These fields are available for jobs, requirements, iterations, test segments, test cases, test suites, test environments, test systems and test runs.

Categorization, Individual Views
All involved objects can be categorized and arranged in multiple tree structures as required. This is similar to a file system based on individual criteria and allows different views of a test project.

Saving Binary Attachments
Additional information such as test data, text documents, graphics, screenshots etc. can be uploaded into the application, saved and assigned to the objects iteration, job, requirement, system under test, test environment, test segment, test case, test step, test suite and test case results.

Import Interfaces for Test Cases
Existing requirements and test cases can be imported from XML and Excel files. A REST interface, if needed, provides continuous synchronization with an external data source or application.

Revisioning Klaros-Testmanagement supports the versioning of test requirements, test segments, test cases and test suites. Each version can be maintained, executed, and evaluated separately.

1.2. Test Planning

Task Planning At the push of a button, jobs for executing test cases and test suites are created and assigned to individual persons. The status of the job execution is visible at any time.

Test Coordination Jobs can be broken down into sub jobs, arranged hierarchically and assigned to testers. By defining dependencies, you can set preconditions for successive test sequences.

Utilization, Progress and Success Clear tables and graphs show usage, duration, and progress of each user's test activity. The progress and success rates of the tests are directly available and always up to date.

Test Case Reviews Even test cases can be defective. Checking and fixing these tests can be assigned to users as a special job. The further course is logged automatically.

1.3. Test Execution

Guided Manual Test Execution Klaros-Testmanagement guides the tester step by step through the manual tests. The test process is automatically logged and makes the data obtained available for evaluation. For each step, annotations and attachments can be added to document the test result.

Continuation of Interrupted Work Manual execution can be paused whenever and be resumed any later time.

Separate Test Data and Test Instructions The input and test data used in test cases, such as credentials, IDs, etc., can be defined and stored externally in Excel sheets. Tests can thus be parameterized in a targeted manner and can be reused in different scenarios.

Interoperability with continuous integration servers (Jenkins/Hudson) A plug-in for the Jenkins Continuous Integration Server automatically imports the test results generated by a Jenkins build into Klaros-Testmanagement.

Integration with Issue/Defect Management Found defects can be created and modified directly in issue management systems such as Bugzilla, GitHub, GitLab, JIRA, Mantis, Redmine and Trac during test execution, without having to leave Klaros-Testmanagement.

Import Interface for Test Automation Test results generated by test automation tools can be imported manually or via REST interface and then be merged with manual test results. Over 30 different formats and tools are supported: *AUnit*, *Boost Test*, *Check*, *CppTest*, *CppUnit*, *ctest*,

CUnit, *embUnit*, *Fitnesse*, *Free Pascal Unit*, *Gauge* (via xml-report plugin), *GoogleTest*, *Glib / gtester*, *JBehave*, *JMeter*, *Jubula / GUIDancer*, *JSUnit*, *JUnit*, *MbUnit*, *MSTest*, *NUnit*, *PHPUnit*, *QF-Test*, *QTestLib*, *Ranorex*, *Selenium* (via JUnit XML result format), *Squish* (via JUnit XML result format), *TESSY*, *TestComplete*, *TestNG*, *TUSAR*, *UFT / QTP*, *UnitTest++*, *Valgrind* and *xUnit.net*.

Integration with Performance Test Tools

For comprehensive reporting, the results of non-functional tests such as load and performance tests often need to be evaluated and prepared. Klaros-Testmanagement offers the possibility to directly import results from leading tools. Supported load test software includes: *BlazeMeter*, *JMeter*, *Gatling*, *OctoPerf*, *SmartMeter*.

1.4. Evaluation, Statistics, Reports

Configurable Dashboard

The Dashboard provides numerous reports that show the state and progress of each activity and the entire project at a glance. It can be configured by the users according to their individual needs. Individual reports can be exported from there in several formats.

Reports and Statistics

Evaluations and statistics can easily be prepared and retrieved from the information stored in the database. Klaros-Testmanagement ships with numerous predefined reports that can be applied directly.

Customized Reports

Using a programming interface that provides access to the entire database, custom reports in PDF or Excel format can be freely defined and applied to your data.

Excel Export of Data Tables

In case of a required post-processing or further evaluation, data tables can be exported to an Excel file.

Print View

Most views, diagrams and result tables provide a customizable print-friendly page view and can be sent directly from the browser to the printer.

Report File Export

The report files can be exported into various file formats, such as PDF, HTML and CSV.

1.5. Configuration

Rights and Roles

Klaros-Testmanagement supports a fine-grained role-based user management system, which is linked to a rights and roles system. In this way, individual assignments can be made or the access and editing rights of individual users for critical data can be restricted.

E-Mail Notifications

The status for selected events, such as "job assigned", "test execution failed", "job ready for execution" and many more, can

	be sent to the users or other persons involved as email notification on request. For this purpose there are numerous configuration options that can be individually tailored to individual projects.
LDAP and Active Directory for User Authentication	User authentication and password management can be handled directly in Klaros-Testmanagement or via an external LDAP or Active Directory.
Single Sign-On Authentication via CAS	Klaros-Testmanagement supports Single Sign-on via CAS (Central Authentication Service).
Docker Integration	In addition to the native installation on Windows and Linux systems, Klaros-Testmanagement can also be operated as a container within a docker-based environment. The application can be installed and managed in isolation, making updates easier to perform. Supported databases includes: *Apache Derby*, *MariaDB*, *MySQL*, *PostgreSQL*, *Microsoft SQL Server*.
Remote Data Interface (REST)	The data stored can be made available to other tools via the REST interface. Individual integrations with your own tools are easy to implement. This allows for a wide variety of additional integration options.
Backup and Restore	The backup data is stored in an XML format. Single projects or the entire database can be extracted and recovered.

Chapter 2. Introduction

2.1. Structure and Objects

Klaros-Testmanagement is a web-based test management solution for organizing, executing and evaluating all test activities. The functional scope covers all areas of the test process: test planning, test creation, test execution, assignment and evaluation of test tasks as well as test evaluation and report generation.

In this introduction the internal structure and function of the individual components, the objects, described. The following elements are defined as *Objects* in Klaros-Testmanagement: **Project**, **Test Case**, **Test Step**, **Test Segment**, **Test Step Result**, **Test Case Result**, **Test Suite**, **Test Suite Result**, **Test Environment**, **Test System**, **Test Run**, **Requirement**, **Iteration** and **Job**.

2.2. Project - Managing a Test Project

The top-level object is the *Project*. It contains all other objects required to define, plan, execute, and evaluate a particular test project.

Projects are typically self-contained units, but there are however also ways to reference objects in other projects. For more details on defining projects see Section 6.1.

2.3. Test Case - Defining a Test

Test Cases are the central objects of a project and may contain any number of test steps. They can be executed manually or automatically.

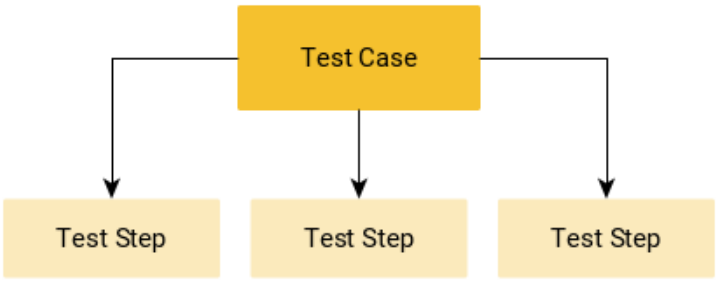


Figure 2.1. Test Case Structure

Test cases can be provided with attributes and include more detailed information:

Descriptive Attribute

Description

Precondition

Postcondition

Expected result

Test steps

Classifying Attributes

Execution type (Manual/Automated)

Priority (High/Medium/Low)

Status (Draft/Locked/Approved/Skip)

Test Type (Functional, Non-Functional, Structural, Regression, Re-Test)

Design Technique (Black-Box, White-Box)

Test Level (Component Test, Integration Test, System Test, Acceptance Test)

Variety (Positive, Negative)

Requirement reference (Document/traceability)

Only available in Klaros-Testmanagement Enterprise Edition

Additional fields can be defined for special requirements.

The test cases consist of individual test steps in which the test activities are defined and described.

2.4. Test Step - Determine the Test Activity Sequence

The *Test Steps* describe the test execution procedure. Each test step contains an instruction to the tester as to which action to perform. If desired, images and documents can be attached to each test step.

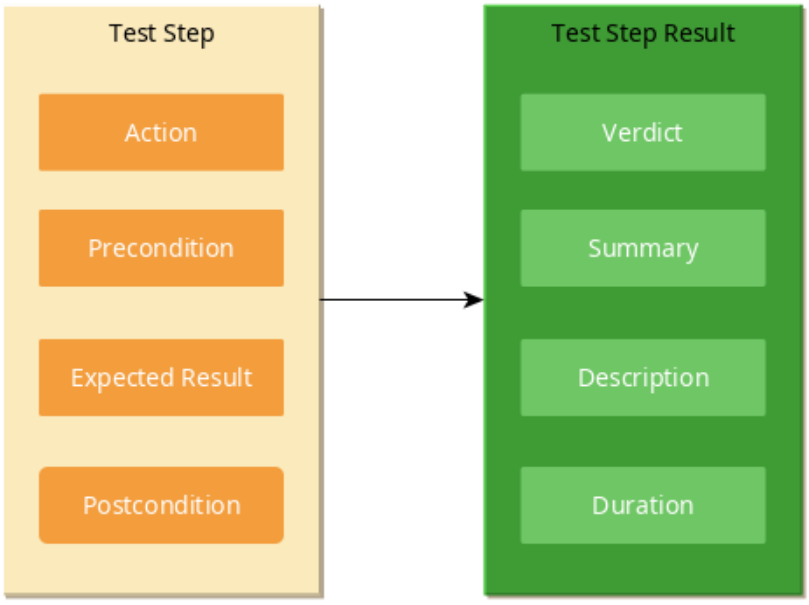


Figure 2.2. Test Step

The description of the test step includes the following components:

Action	The textual description of the action the tester has to perform.
Precondition	The state the test system is in before the action begins.
Expected Result	The predicted behavior of the test system or component, based on the specification.
Postcondition	The expected state of the test system after the action has been performed.

2.4.1. Test Step Result

Klaros-Testmanagement guides the tester through the test execution step by step. Each step is completed by the tester with a **Test Step Result** and automatically recorded.

The test step result contains the following information:

Result	**Passed:** The test was completed as planned. **Failure:** The test does not deliver the expected result. **Error:** The test could not be performed. **Inconclusive:** The test does not deliver a clear result **Skipped:** The test step has been skipped.
Summary (optional)	A short summary of the test step result.
Description (optional)	A textual description of the test step result.
Duration	The automatically logged execution time of each individual test step.

2.4.2. Automated Test Cases

Automated test cases usually do not have test steps, since they are executed by an automated test tool (e.g. JUnit). The test results are provided in the form of a result file which will be imported into the system. Once Manual and automated test results are available can be jointly evaluated.

2.5. Test Segment - Reusing Test Steps

Only available in Klaros-Testmanagement Enterprise Edition

A *Test Segment* is a predefined sequence of test steps that is created as a separate object that can be inserted into any test case.

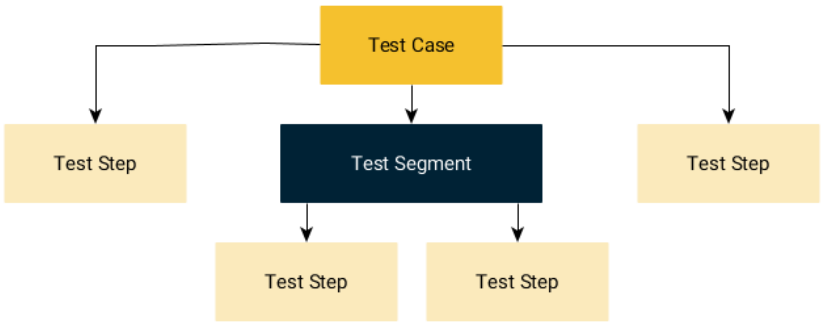


Figure 2.3. Test Segment

Often, certain test steps or sequences of test steps occur exactly the same in several test cases. To ensure that they are identical, these steps can be encapsulated in test segments and centrally managed, edited and stored.

The test steps of a test segment have exactly the same attributes as the test steps of a test case.

2.6. Test Suite - Arranging Test Cases

Test cases can be grouped into *Test Suites* to be executed together one after the other in one test run.

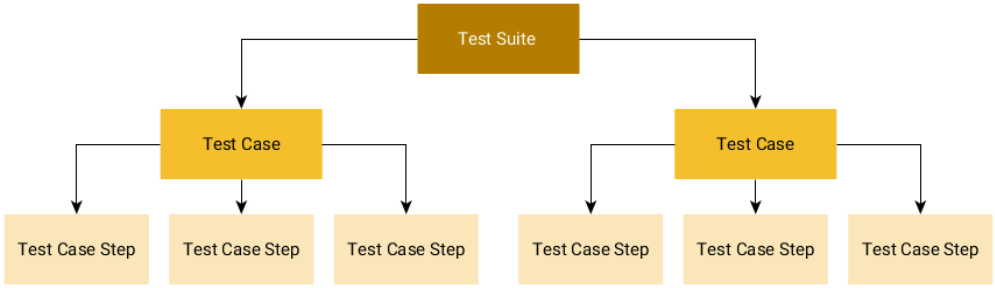


Figure 2.4. Test Suite

A test case can occur in multiple test suites. It is merely referenced, so that changes to the test case automatically affect all test suites that contain this test case.

2.7. System under Test - The Test Object

The *System under Test* is the component or system that is to be tested. Usually this is a software version, but can also be a specific device or other hardware. A test case can only be executed once a test system has been defined.

2.8. Test Environment - The External Influence on the Test Result

A *Test Environment* represents external conditions that may affect the test result. This can consist of hardware or software, such as a browser version used or the operating system in which the test system software was installed. A test case cannot be executed until a test environment has been defined.

2.9. Test Run - The Results of Executed Test Cases

A *Test Run* is automatically created when the tester starts to execute a test case or test suite. It is assigned to the test environment and system under test in use and collects all results that occur during the execution of the test case or test suite. The test runs thus document the entire test activities.

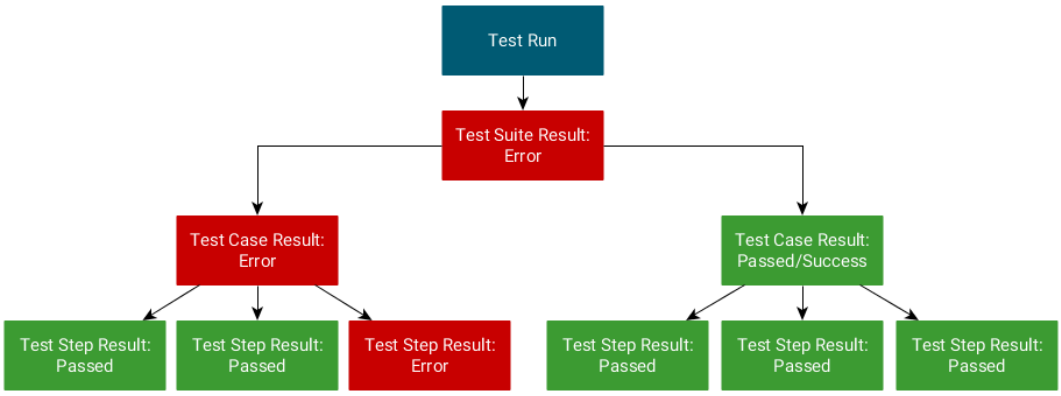


Figure 2.5. Test Run with Test Results

Test runs can be paused and continued later. The result (Passed, Failed, Error, Inconclusive, Skipped) appears in the evaluation only after all test cases of the test run have been completely executed. Until then, the test run with its collected results is not displayed.

Logged Parameters of a Test Run

During a test run, the following parameters are automatically logged and can be used for evaluation:

The test system used

The test environment used

The executed test case/test suite

The execution date

The executing user

2.10. Requirement – Properties of the System under Test

Only available in Klaros-Testmanagement Enterprise Edition

A *Requirement* is a predefined condition to be fulfilled that describes the desired properties of the system under test. To ensure that the system under test satisfies these properties, test cases are defined and these are assigned to the requirement that check the test cases. In this way, it can be verified which requirements are covered by tests (*coverage*) and which of them were executed with the result *passed* (*conformity*).

From this set of information Klaros-Testmanagement is able to automatically determine coverage and compliance information for the whole project or selected objects once the first test results are available.

Attributes

Name

Priority (High/Medium/Low)

Short Description

Long Description

2.11. Iteration - Subdividing a Project into Phases

Only available in Klaros-Testmanagement Enterprise Edition

An *Iteration* represents a selected phase of a project. It encapsulates a subset of objects including all tasks and test results executed during that phase. Grouping objects in this way makes it easier to identify different test cycles in a project and enables better integration of the testing process, especially with agile software development practices.

In an iteration, the test systems, test environments and requirements under consideration are defined within a project for a specific period of time.

2.12. Job - Planning the Test Process

Only available in Klaros-Testmanagement Enterprise Edition

Test activities are planned and coordinated by means of a *Job*. For this purpose, the execution of a test case or a test suite is specified in the job.

After a job is created, it can be assigned to a user and executed and evaluated by him. Jobs can be repeated and executed multiple times, so that they can be assigned to multiple test runs. Jobs can be arranged hierarchically and dependencies between jobs can be specified.

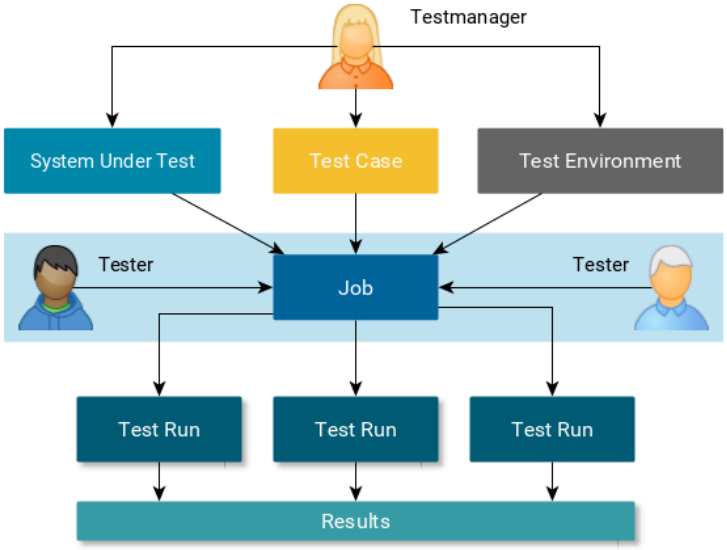


Figure 2.6. Job

In addition to test execution, other job types are supported. These are used for checking test cases, planning requirements and formulating any other jobs in text form.

Jobs can be nested and individually assigned to users. Therefore, it is possible to track progress of larger, distributed test activities.

2.13. User - Users and their Rights

Klaros-Testmanagement offers four different user roles, which are assigned when the user is created and can be changed later if necessary: *Administrator*, *Manager*, *Tester* and *Guest*.

Administrator — An Administrator always has access to all functions in Klaros-Testmanagement. In addition to the test-specific activities, this also includes user administration, backups and recovery as well as system-wide settings.

Test Manager — Test Managers have full read and write access to all data of a project. They can create and delete objects and manage project settings. This also includes creating, editing and deleting users with the “Tester” or “Guest”role.

Tester — Testers can run tests, edit test results and issues, and display reports. Beyond this, they have read-only access.

Guest — The “Guest” role is intended for project managers, customers or reviewers. This role has complete but read-only access to all data and can display, save and export reports in other formats.

The global user roles apply to all existing projects. However, if necessary, the role of a user can be assigned individually in each project.

Project Specific User Roles

Only available in Klaros-Testmanagement Enterprise Edition

In addition to the global assignment, it is possible to reassign the user roles per project or to exclude users from individual projects. The user retains his global role, but receives extended or restricted rights or no access in the selected project.

This is useful, for example, when integrating external employees who are only active in a specific project. On the other hand, a tester may need test manager rights in certain projects and can receive the corresponding assignment.

Detailed information about the role permissions can be found in Appendix A.

2.14. Overview of Objects

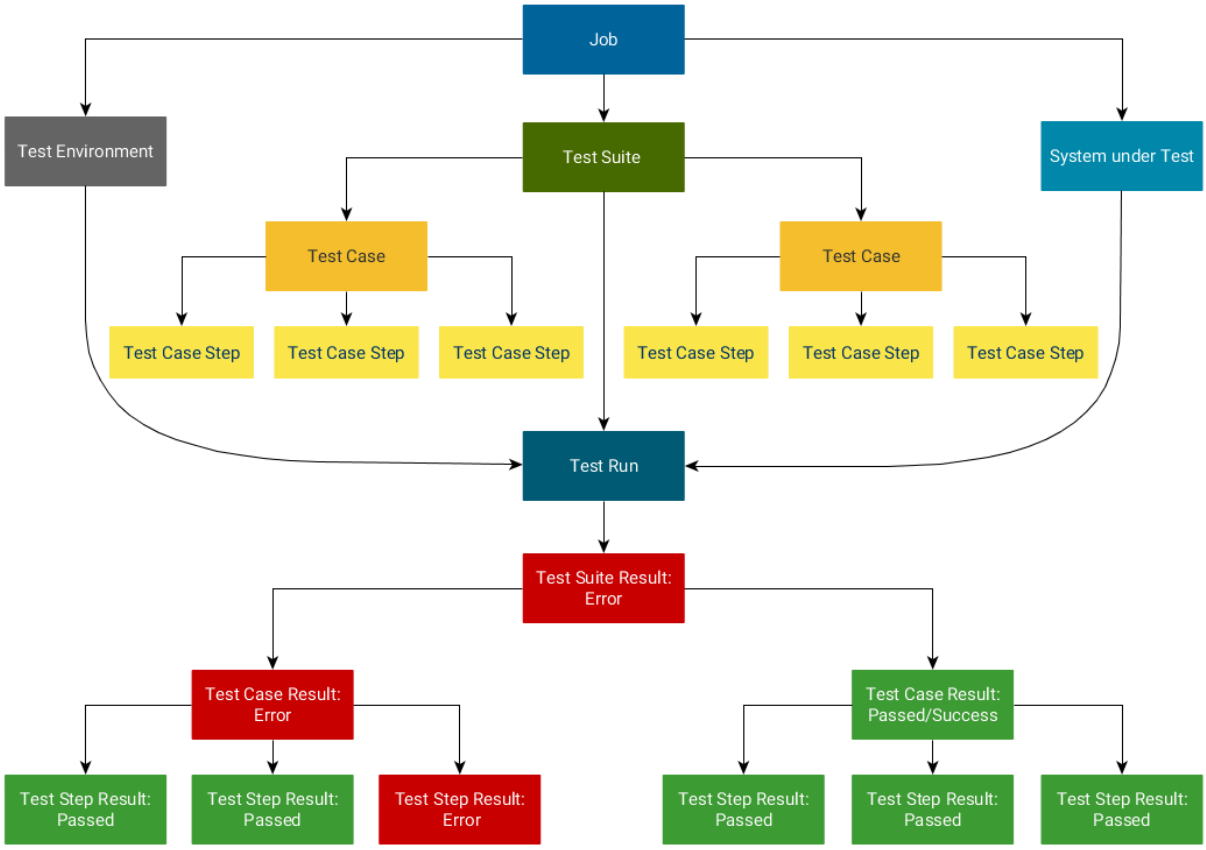


Figure 2.7. Objects Overview

2.15. Issue - Reporting and Resolving Defects

Klaros-Testmanagement integrates with issue tracking systems like Jira, Redmine, GitLab and many others (see Section 12.5). If a defect is found in a test step, it can be passed directly from Klaros to the connected issue management system without leaving the application.

The entry is automatically linked in both systems. Changes in the issue tracker are regularly synchronized with test management in the background. An automatically generated backlink in the issue management system leads back to the test case in Klaros-Testmanagement with one click.

Chapter 3. Installation

This chapter provides all information necessary to install Klaros-Testmanagement.

Klaros-Testmanagement is providing installers for Microsoft Windows- and Linux-based operating systems. The installer contains all additional programs which are needed for the first use of Klaros-Testmanagement. These include the file-based database (Apache Derby), the Apache Tomcat application server and a suitable Java runtime environment.

Ready to use Docker images for various databases are available as well.

These can be found along with detailed documentation at. https://github.com/klaros-testmanagement/klaros-docker.

Database not Suitable for Production Environments

The installer configures Klaros-Testmanagement to use the embedded Apache Derby database which is not intended for production use.

For production systems, it is strongly recommended to use one of the other recommended databases like MariaDB, MySQL, PostgreSQL or SQLServer. Each of them will provide significantly better performance. The database can be changed after running the initial installation process, (see Section 3.11, “Changing the Default Database”).

3.1. License Model for Community and Enterprise Edition

The license model varies depending on the Klaros-Testmanagement edition used. Compared to the Klaros-Testmanagement Community Edition the Klaros-Testmanagement Enterprise Edition offers a limited amount of functions, but does not limit the number of users who can work with the software.

With Klaros-Testmanagement Enterprise Edition the purchased license limits the number of active users who can work with the software.

3.2. System Prerequisites

Klaros-Testmanagement is a web-based application. Client and server side therefore have different system requirements, which are described in detail in the next sections.

3.2.1. Client Prerequisites

To access Klaros-Testmanagement, a modern web browser supporting JavaScript is required.

3.2.1.1. Supported Browsers

- Apple Safari
- Google Chrome

- Microsoft Edge
- Mozilla Firefox

Microsoft Internet Explorer - Limited Support Notice

Support for Microsoft Internet Explorer will be gradually discontinued in upcoming releases. As of now, basic functionality is still available, but display problems may occur.

Klaros-Testmanagement is designed for a minimum screen resolution of 1280x960 pixels. Full-HD (1920x1080) is the recommended resolution.

3.2.2. Server Prerequisites

To run Klaros-Testmanagement, a Microsoft Windows or Linux 64-Bit operating system running on Intel/AMD x64 is required.

Name	Version
Microsoft Windows	Windows 10, Windows 11, Windows Server 2012 Windows Server 2012 R2, Windows Server 2016, Windows Server 2019, Windows - Server 2022
CentOS, Red Hat - Enterprise Linux (RHEL)	8.x, 9.x
SuSE	SUSE Linux Enterprise Server (SLES) 15
Ubuntu	20.04, 22.04, 24.04
Debian	bullseye, bookworm
Other Linux Distributions	Other Linux distributions are expected to function without problems, as long as they are able to run the supplied 64-bit Java 11 - Runtime environment. Yet, we do not make any compliance statement regarding these distributions.

Table 3.1. Supported Operating Systems

For optimal performance, using a separate database installation is highly recommended. This database is not required to run on the same physical machine as the Klaros-Testmanagement server.

Name	Version
Apache Derby	10.15.1.3 (embedded and preconfigured as default)
Microsoft SQLServer	SQLServer 2008, 2008 R2, 2012, 2014, 2016, 2017, 2019, 2022
MySQL	Version 5.5, 5.6, 5.7, 8.0
MariaDB	Version 10.3 or later
PostgreSQL	Version 9.4 or later

Table 3.2. Supported Database Systems

3.2.2.1. Microsoft Windows

To run Klaros-Testmanagement on Microsoft Windows operating systems the following requirements should be met:

- **Minimum requirements**

  2 GB RAM, 3 GHz Dual Core CPU

- **Recommended requirements (Remote Database)**

  4 GB RAM, 3 GHz Quad Core CPU

- **Recommended requirements (Local Database)**

  8 GB RAM, 3 GHz Quad-Core CPU

- **Operating system**

  Microsoft Windows Server 2012, Microsoft Windows Server 2012 R2, Microsoft Windows Server 2016, Microsoft Windows Server 2019, Microsoft Windows Server 2022, Microsoft Windows 10, Microsoft Windows 11.

3.2.2.2. Linux

To run Klaros-Testmanagement on Linux operating systems the following requirements should be met:

- **Minimum requirements**

  2GB RAM, 2GHz Dual-Core CPU

- **Recommended requirements (Remote Database)**

  4GB RAM, 3GHz Quad-Core CPU

- **Recommended requirements (Local Database)**

  8GB RAM, 3GHz Quad-Core CPU

- **Operating System**

  Linux Intel64/x86-64 (Intel 64 bit architecture) distribution containing glibc version 2.17 or better and version 2.2.1 or better of the GTK+ widget toolkit and associated libraries.

GTK+ libraries only required in Windowed Mode

The GTK+ libraries are only needed to run the Klaros-Testmanagement installer in windowed mode. Running the installer in console mode is possible without this requirement.

- **Restricting the Number of Open Files**

Caution

Klaros-Testmanagement stores database search indices in the `.klaros/indices` folder in home directory of the user account that is running the Klaros-Testmanagement application server. During the indexing process the amount of open files may exceed the predefined limit in the Linux operating system.

The command **ulimit -n** shows the configured number of open files allowed, which should be set to a value of at least 4096. Please consult your system documentation on how to set this value (Usually by editing `/etc/security/limits.conf`).

3.2.2.3. Available Disc Space

Klaros-Testmanagement stores binary attachments in a file based repository inside the `.klaros` folder in home directory of the user account that is running the Klaros-Testmanagement application server.

Depending on the usage pattern a sufficient amount of disk space should be reserved for this account. If the file based Derby database is active, the database files will also be located in this directory.

Recommended Available Disc Space

Reserving 10 GB of disc space should be enough for typical usage patterns.

3.3. Installation

Under Microsoft Windows the installer is invoked by running the binary **Klaros-Testmanagement-<version>-Setup.exe**.

Under Linux the installer can be started by running the binary **Klaros-Testmanagement-<version>-Setup.bin** in a user shell.

The following screens show each step involved in installing Klaros-Testmanagement.

3.3.1. Language Selection

Upon startup the installer is requesting the language used throughout the following installation procedure.

Currently both English and German languages are available.

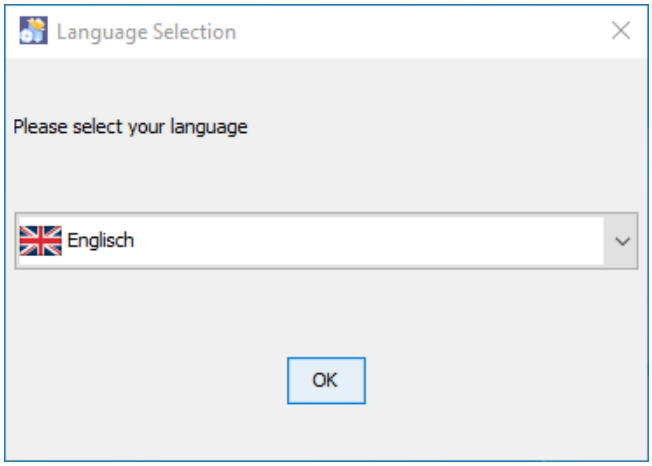


Figure 3.1. Language Selection

3.3.2. Step 1: Welcome

The initial step shows a welcome screen to the user (Figure 3.2).

Figure 3.2. The “Welcome” Screen

The installation may be aborted by clicking the Quit button. Clicking Next proceeds with the installation.

3.3.3. Step 2: Information

The second step shows information about the product and the revision history, listing the fixed issues and the newly added features (Figure 3.3).

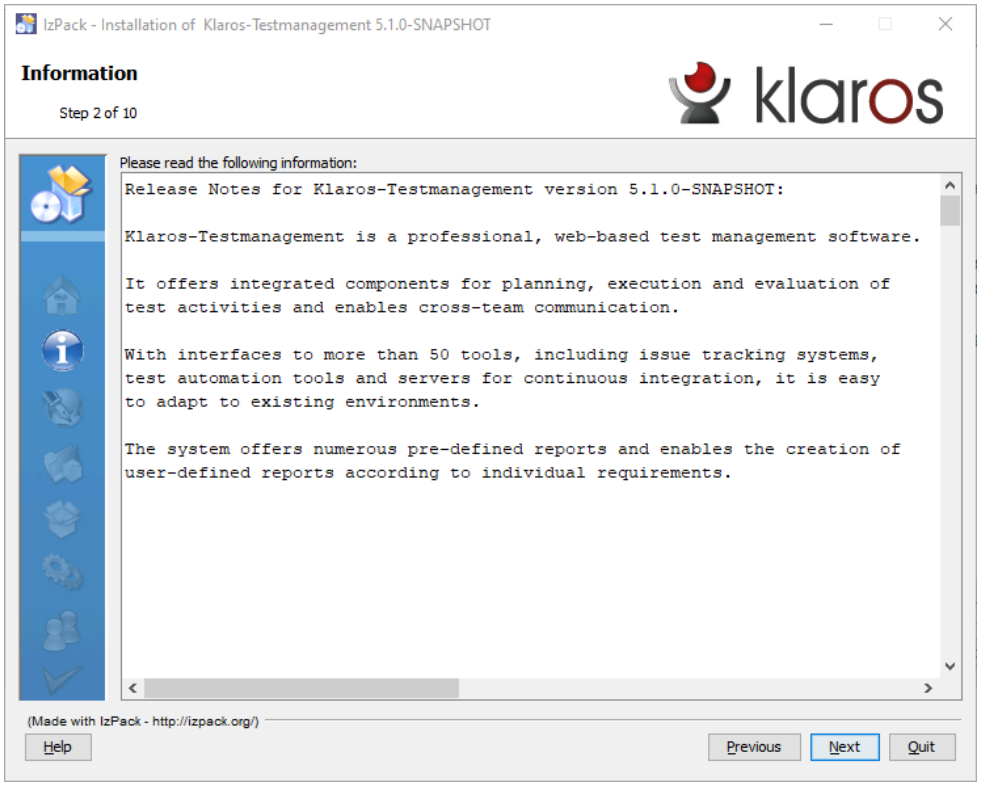


Figure 3.3. The “Information” Screen

The installation may be aborted by clicking the Quit button. Clicking Previous goes back to the *Welcome* step and clicking Next proceeds with the installation.

3.3.4. Step 3: Licensing Agreements

The third step shows information about the license agreement for Klaros-Testmanagement. The license must be accepted to continue the installation. (Figure 3.4).

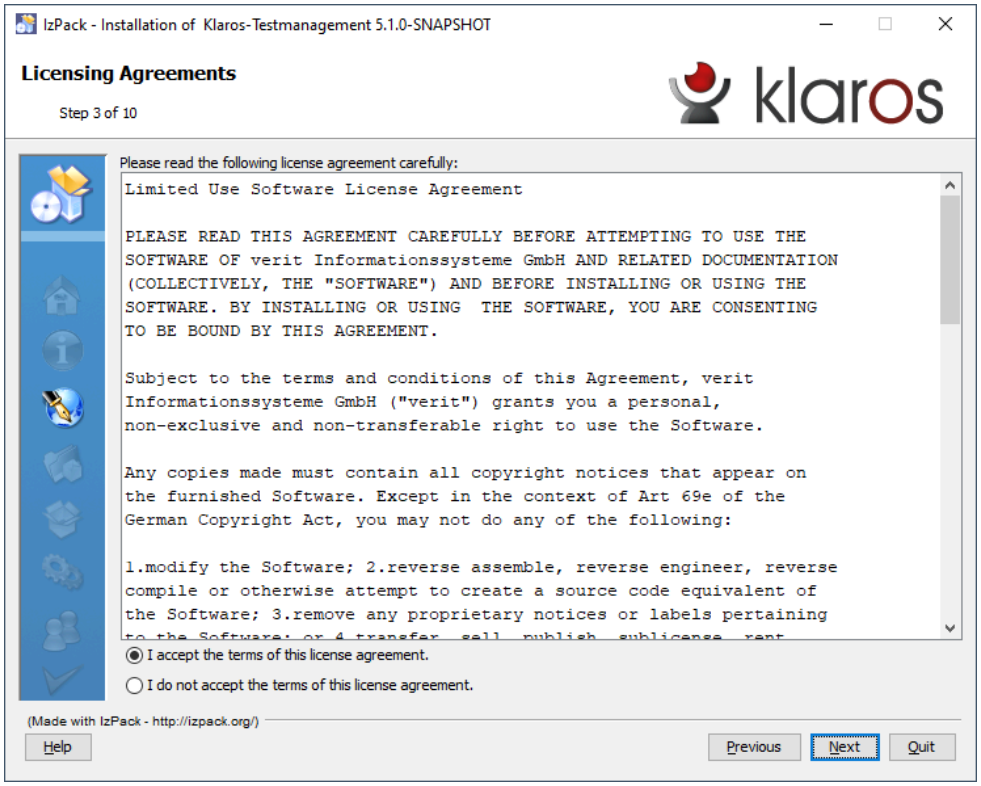


Figure 3.4. The “Licensing Agreement” Screen

The installation may be aborted by clicking the Quit button. Clicking Previous goes back to the *Information* step and clicking Next proceeds with the installation.

3.3.5. Step 4: Target Path

The fourth step requests the target path where Klaros-Testmanagement will be installed. The user can use the Browse button to search for the specific path in the local file system (Figure 3.5).

Install Folder Location

It is highly recommended using the suggested install location and **not** the Windows Programs folder. By default, Klaros-Testmanagement will not be able to start if installed there, as the Tomcat application server requires write access to the installation folder in order to deploy the web application. This is blocked by Microsoft Windows User Account Control (UAC).

If you wish to install Klaros-Testmanagement in this location, UAC must be disabled.

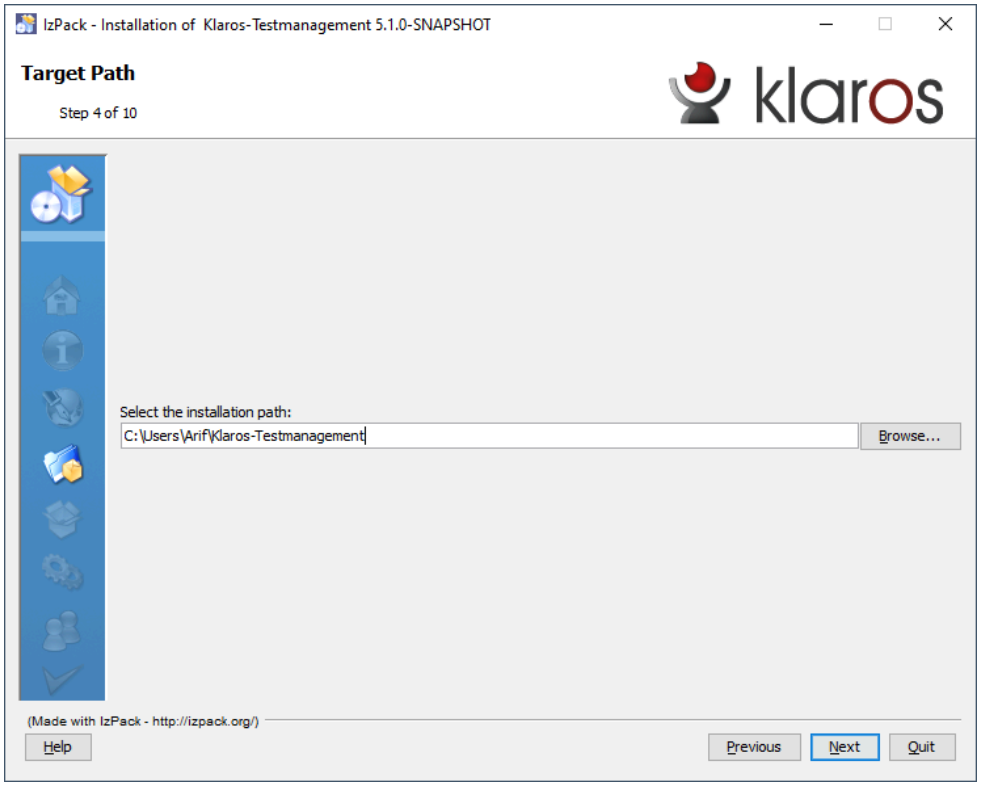


Figure 3.5. The “Target Path” Screen

The installation may be aborted by clicking the Quit button. Clicking Previous goes back to the *Licensing Agreement* dialog and clicking Next proceeds with the installation.

3.3.6. Step 5: Select Installation Packages

The fifth step allows selecting optional packages that are installed with Klaros-Testmanagement.

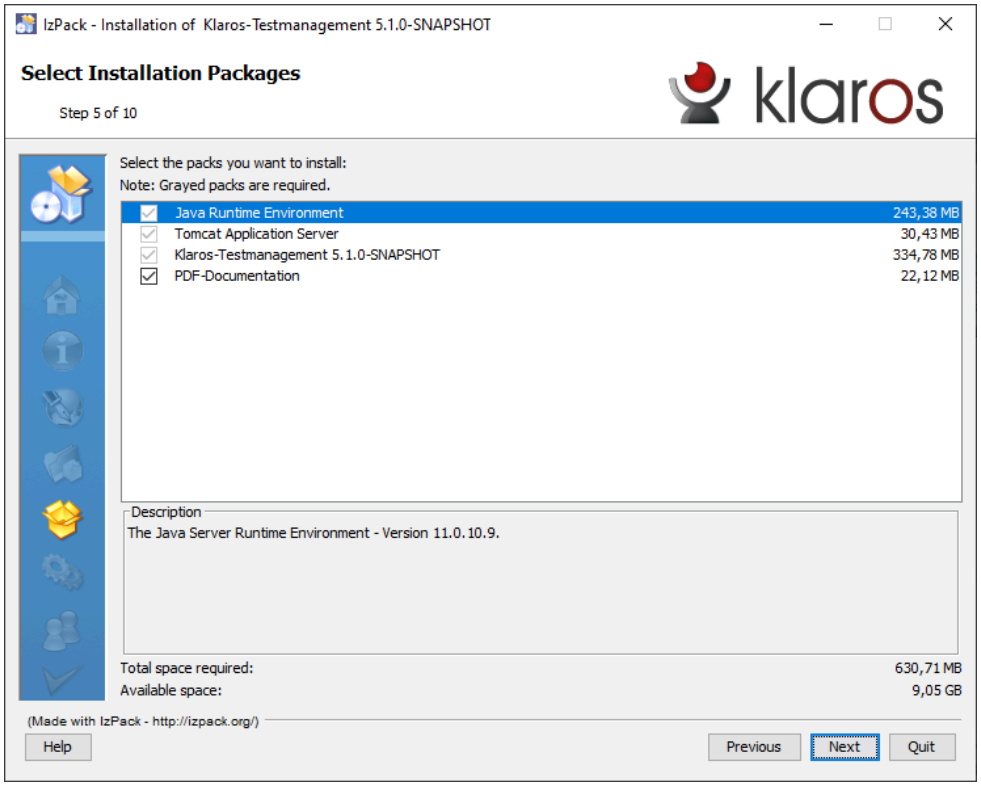


Figure 3.6. The "Select Installation Packages" Screen

The installation may be aborted by clicking the Quit button. Clicking Previous goes back to the *Target Path* dialog, clicking Next proceeds with the installation.

3.3.7. Step 6: User Data

In the sixth step the settings for the installed Tomcat application server can be changed.

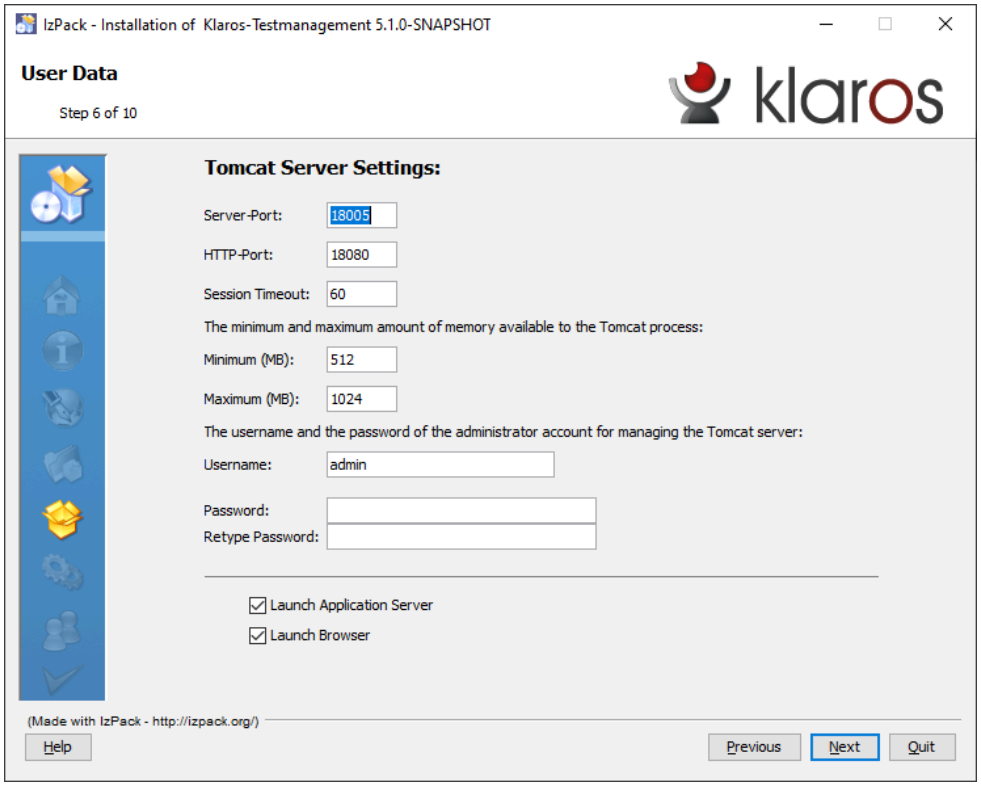


Figure 3.7. The "User Data" Screen

It is possible to set the ports used by Tomcat (Server-Port and HTTP-Port), the minimum and maximum amount of memory available to the Tomcat process as well as the user name and password of the Tomcat application server administrator.

In addition it is possible to control the startup of the application during the installation. If the *Launch Application Server* checkmark is set, the application server will be launched automatically during the installation process. If the *Launch Browser* checkmark is set, a native web browser instance will be redirected to the application login page. Both of these actions will be performed during the Section 3.3.9, "Step 8: Perform External Processes" phase.

The installation may be aborted by clicking the `Quit` button. Clicking the `Previous` button goes back to the *Select Installation Packages* step and clicking `Next` button proceeds with the installation.

3.3.8. Step 7: Installation

The seventh step starts the installation of Klaros-Testmanagement and shows the installation progress. The following screenshots show the *Installation in Progress* Screen and the *Installation finished* screen (Figure 3.8).

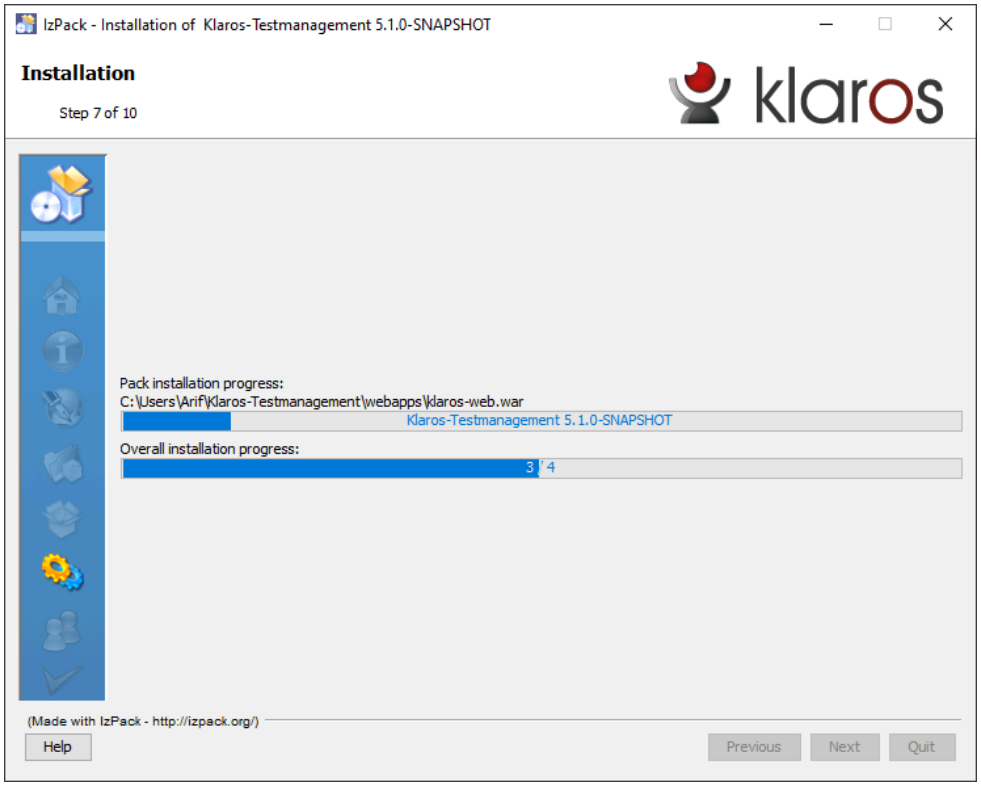


Figure 3.8. The “Installation in Progress” Screen

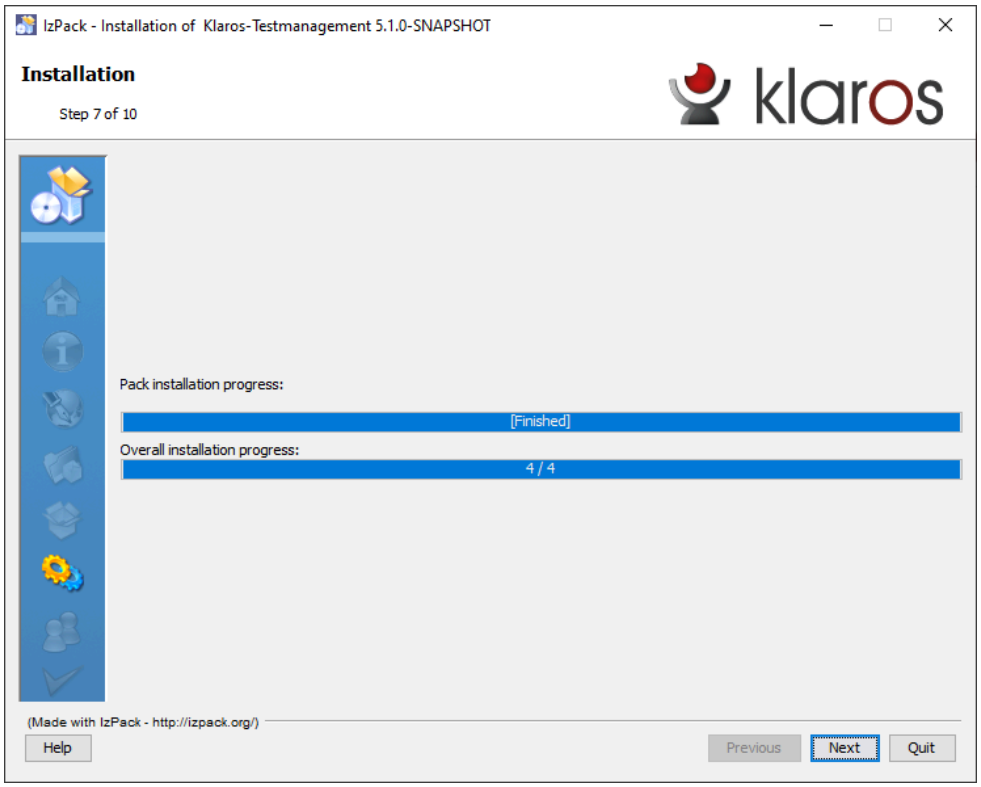


Figure 3.9. The “Installation Finished” Screen

The installation may be aborted by clicking the `Quit` button. Clicking `Next` proceeds with the installation.

3.3.9. Step 8: Perform External Processes

The optional eighth step performs external processes such as setting the environment variables and starting the Tomcat application server.

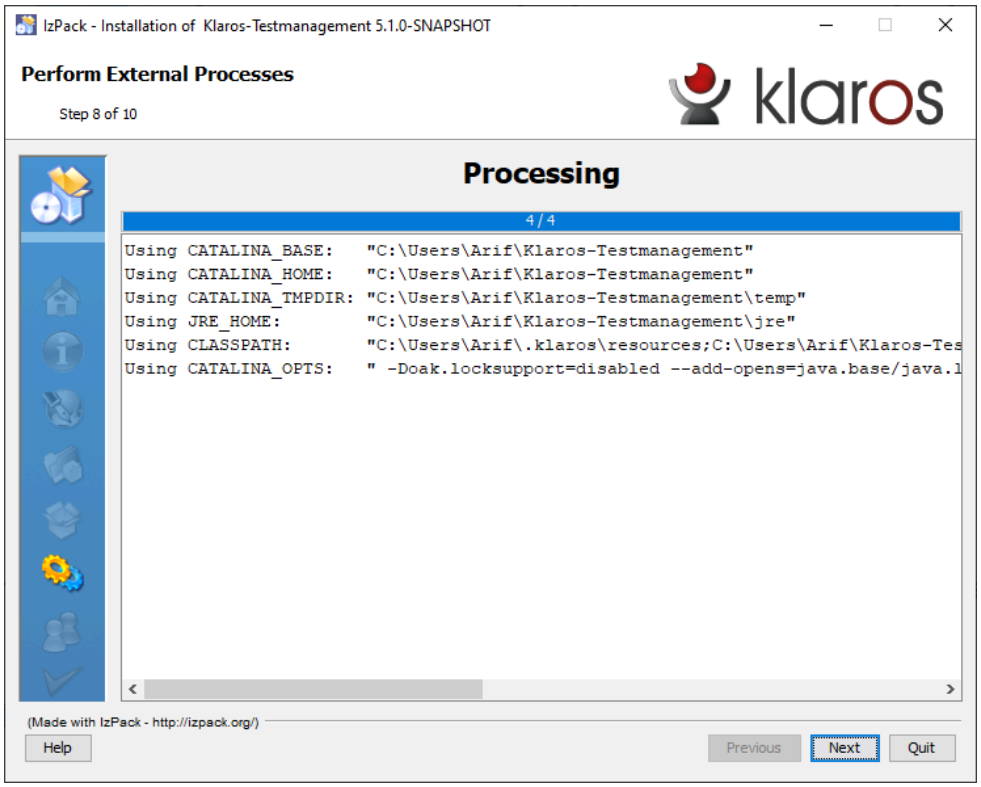


Figure 3.10. The “Processes” Screen

The installation may be aborted by clicking the Quit button. Clicking Next proceeds with the installation.

3.3.10. Step 9: Setup Shortcuts

The ninth step allows to set the options for shortcuts. The installer can create shortcuts in the Start-Menu and on the desktop (Figure 3.11).

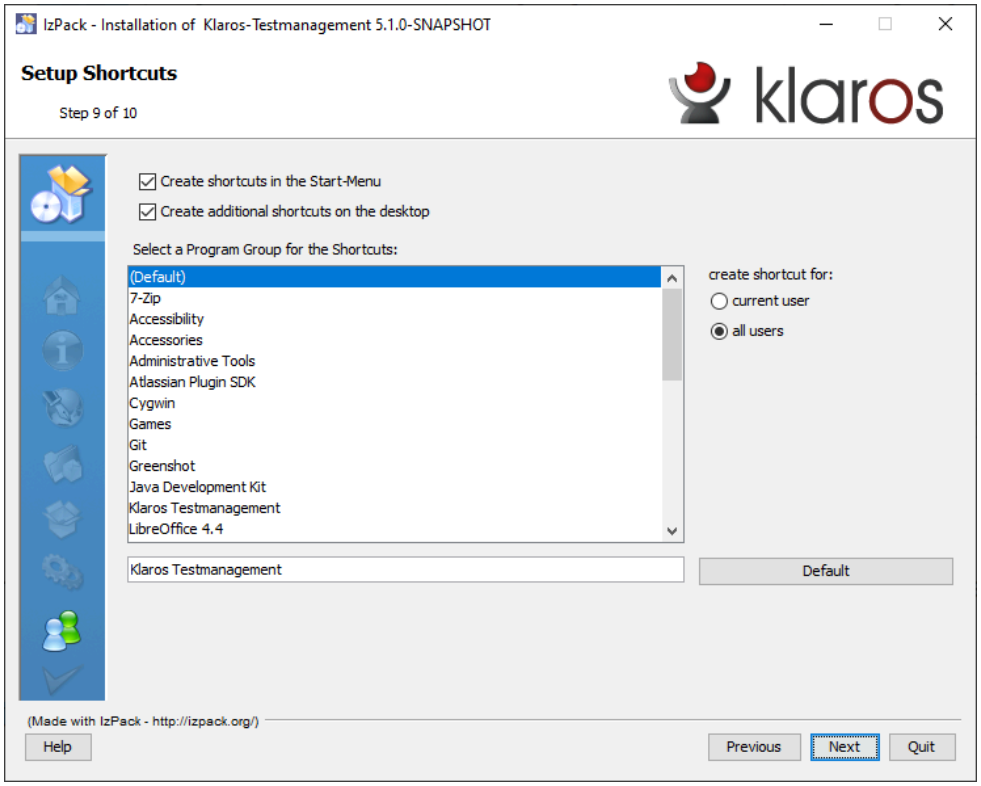


Figure 3.11. The “Setup Shortcuts” Screen

This step looks different on Linux installations but has the same functionality.

The installation may be aborted by clicking the Quit button. Clicking Previous goes back to the *Perform External Processes* step and clicking Next proceeds with the installation.

3.3.11. Step 10: Installation Finished

The last step notifies the user that Klaros-Testmanagement was installed successfully and shows the path to the to the uninstall program (Figure 3.12).

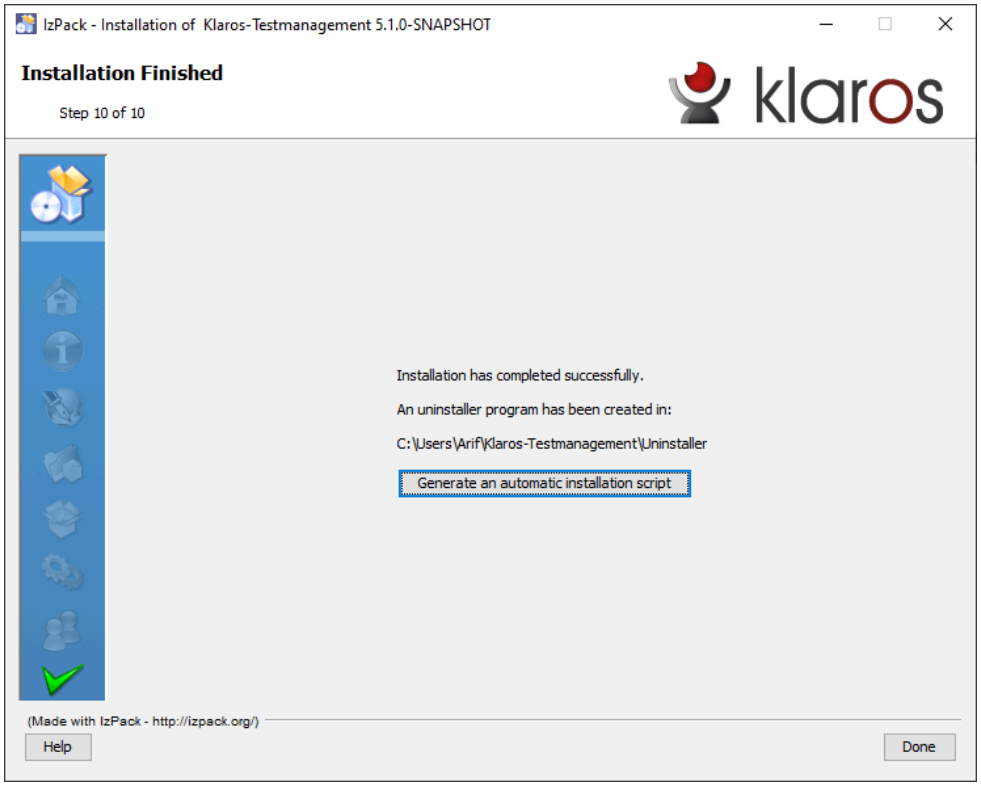


Figure 3.12. The "Installation Finished" Screen

The `Generate an automatic installation script` button saves a complete script of the installation with the selected user choices, so that the same installation could be repeated unattended or on other machines.

The installation can be completed by clicking the `Done` button.

3.4. Console-based Installation

The installer also supports a GUI-less variant which only needs a shell/command window and can be invoked by adding the option **-- -console** to the command line.

This installer displays the content of an installer page in a panel in the shell window and lets the user input her choices via keyboard.

The buttons are replaced by short key sequences:

Enter	shows the next page or accepts a default value
1+Enter	selects a checkbox
0+Enter	deselects a checkbox
n+Enter	chooses the nth item from a list

3.4.1. Step 1: Language

The *Language* panel (Section 3.3.1, "Language Selection") is displayed. Enter **0+Enter** for English or **1+Enter** for German Language.

```
Select your language
```

```
0  [x] eng
1  [ ] deu
Enter selection:
```

3.4.2. Step 2: Welcome

The *Welcome* panel (Section 3.3.2, “Step 1: Welcome”) is displayed and wants to be committed by **1+Enter**.

```
--------------------------------------------------------------------------------
Information
--------------------------------------------------------------------------------


Version 5.5.5
(c) Copyright 2009-2023 verit Informationssysteme GmbH. All Rights Reserved.



Press 1 to continue, 2 to quit, 3 to redisplay
1
```

3.4.3. Step 3: Information

The *Information* panel (Section 3.3.3, “Step 2: Information”) displays the release notes of the recent versions.

```
...

     Version 5.5.5

     New Features...


Press 1 to continue, 2 to quit, 3 to redisplay
1
```

3.4.4. Step 4: Licensing Agreements

The *License* panel (Section 3.3.4, “Step 3: Licensing Agreements”) displays the Licensing Agreements, which must be accepted to continue with the installation. Commit with **1+Enter**.

```
--------------------------------------------------------------------------------
Licensing Agreements
--------------------------------------------------------------------------------

Limited Use Software License Agreement

PLEASE READ THIS AGREEMENT CAREFULLY BEFORE ATTEMPTING TO USE THE
SOFTWARE OF verit Informationssysteme GmbH AND RELATED DOCUMENTATION
(COLLECTIVELY, THE "SOFTWARE") AND BEFORE INSTALLING OR USING THE
SOFTWARE. BY INSTALLING OR USING THE SOFTWARE, YOU ARE CONSENTING
TO BE BOUND BY THIS AGREEMENT....

limitations may not apply to you. You may also have other rights
that vary from state to state.

Press 1 to accept, 2 to reject, 3 to redisplay
1
```

3.4.5. Step 5: Target Path

The *Target Path* panel (Section 3.3.5, “Step 4: Target Path”) looks like this with the console installer:

```
-------------------------------------------------------------------------------
Target Path
-------------------------------------------------------------------------------


Select the installation path: [C:\Users\...\Klaros-Testmanagement]
```

You can press **Enter** to accept the proposed path or enter a different one. The path will be created, if it did not already exist. If the directory already exists, a warning is displayed, and you can cancel the installation or continue.

3.4.6. Step 6: Select Installation Packages

The *Select Installation Packages* panel (Section 3.3.6, “Step 5: Select Installation Packages”) is more complex in console mode. The required packages (Java, Tomcat and Klaros-Testmanagement) are displayed for information only. Entering **Y** selects the installation of the PDF documentation, while **N** deselects it. You are then prompted to confirm your entry with **1** and **Enter**.

```
-------------------------------------------------------------------------------
Select Installation Packages
-------------------------------------------------------------------------------

Select the packs you want to install:

  [x] Pack ‘Java Runtime Environment’ required
  [x] Pack 'Tomcat 9 Application Server' required
  [x] Pack 'Klaros-Testmanagement required
-----------------------------------------------
  [x] Include optional pack 'PDF-Documentation'
-----------------------------------------------
Enter Y for Yes, N for No: N
Y
Done!

Press 1 to continue, 2 to quit, 3 to redisplay
1
```

3.4.7. Step 7: User Data

After that, the installer requests user data (Section 3.3.7, “Step 6: User Data”). The installation program displays default values that can be accepted or overwritten by pressing **Enter**. Checkboxes show [x] if the checkbox is set, or [ ] otherwise. Pressing **Enter** accepts the current value, which can be set and unset explicitly with **1** and **0** respectively.

In this example we adopt most of the default values. Only the memory settings and the tomcat admin password (without default values) are entered and the two checkboxes are unset:

```
-------------------------------------------------------------------------------
User Data
-------------------------------------------------------------------------------
```

```
Tomcat Server Settings:

Server-Port: [18005]

HTTP-Port: [18080]

Session Timeout: [60]

The minimum and maximum amount of memory available to the Tomcat process:
Minimum (MB): [512]
400
Maximum (MB): [1024]
1024

The username and the password of the administrator account for managing the Tomcat server:
Username: [admin]

Password:
**********
Retype Password:
**********

-----------------------------------------
  [ ] Launch Application Server
Enter 1 to select, 0 to deselect:
0

  [ ] Launch Browser
Enter 1 to select, 0 to deselect:
0


Press 1 to continue, 2 to quit, 3 to redisplay
1
```

3.4.8. Step 8: Installation

Packages are displayed as soon as they are installed (Section 3.3.8, “Step 7: Installation”).

```
[ Starting to unpack ]
[ Processing package: Java Runtime Environment (1/4) ]
[ Processing package: Tomcat 9 Application Server (2/4) ]
[ Processing package: Klaros-Testmanagement (3/4) ]
[ Processing package: PDF-Documentation (4/4) ]
[ Unpacking finished ]
Installation finished
```

3.4.9. Step 9: Perform External Processes

This panel is skipped since we chose not to launch the server or a browser. Otherwise, startup messages of the server could be seen here. See Section 3.3.9, “Step 8: Perform External Processes” on how this is handled in GUI mode.

3.4.10. Step 10: Setup Shortcuts

In a GUI-less environment we typically do not want to create any menu entries or icons on the desktop, thus we choose **N**. See Section 3.3.10, “Step 9: Setup Shortcuts” for details on shortcut setup in windowed environments.

```
-------------------------------------------------------------------------------
Setup Shortcuts
-------------------------------------------------------------------------------

---------------------------------
Create shortcuts in the Start-Menu
---------------------------------
Enter Y for Yes, N for No:
N
-----------------------------------------
Create additional shortcuts on the desktop
-----------------------------------------
Enter Y for Yes, N for No:
N
```

3.4.11. Step 11: Installation Finished

An automatic installation script with the selected user choices can be generated here, so that the same installation could be repeated for unattended updates or on other machines. Automatic installation scripts will work for unattended installations in console mode as well.

```
Generate an automatic installation script
Enter Y for Yes, N for No:
Y
Select the installation script (path must be absolute)[C:\Users\...\Klaros-Testmanagement
  \auto-install.xml]

Installation was successful
Application installed on C:\Users\...\Klaros-Testmanagement
[ Writing the uninstaller data ... ]
[ Console installation done ]
```

After finishing the installation, Klaros-Testmanagement is located in the directory `/opt/klaros-testmanagement`. In Section 3.12, “Installing as a System Service” you can find more information how Klaros-Testmanagement can be set up as a system service to run it automatically at system startup.

3.5. Automated Installation Script

If an automated installation script was saved during Klaros installation, the program can be reinstalled with the same parameters. In this example it is assumed that the automated installation script was saved under `auto-install.xml`.

Linux: **Klaros-Testmanagement-<version>-Setup.bin -- -console auto-install.xml**

Windows: **Klaros-Testmanagement-<version>-Setup.exe auto-install.xml**

You may also use the following sample script. Please adapt the setting for `installpath` and the `entry` fields under `userInput`. The installation may issue some warnings of the form `Automation-Helper class not found for panel ...`, which can be safely ignored.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<AutomatedInstallation langpack="eng">
  <com.izforge.izpack.panels.htmlhello.HTMLHelloPanel id="hello" />
```

```
  <com.izforge.izpack.panels.htmlinfo.HTMLInfoPanel id="info" />
  <com.izforge.izpack.panels.licence.LicencePanel id="license" />
  <com.izforge.izpack.panels.target.TargetPanel id="target">
    <installpath>/opt/Klaros-Testmanagement</installpath>
  </com.izforge.izpack.panels.target.TargetPanel>
  <com.izforge.izpack.panels.packs.PacksPanel id="packs">
    <pack index="0" name="Java Runtime Environment" selected="true"/>
    <pack index="1" name="Tomcat 9 Application Server" selected="true" />
    <pack index="2" name="Klaros-Testmanagement" selected="true" />
    <pack index="3" name="PDF-Documentation" selected="false" />
  </com.izforge.izpack.panels.packs.PacksPanel>
  <com.izforge.izpack.panels.userinput.UserInputPanel id="portselection">
    <userInput>
      <entry key="tomcat_server_port" value="18005" />
      <entry key="tomcat_http_port" value="18080" />
      <entry key="tomcat_https_port" value="18443" />
      <entry key="tomcat_session_timeout" value="60" />
      <entry key="tomcat_memory_min" value="256" />
      <entry key="tomcat_memory_max" value="1024" />
      <entry key="tomcat_admin_name" value="tcadmin" />
      <entry key="tomcat_admin_password" value="IfYouDontChangeThisDontBlameMe" />
      <entry key="start_server_flag" value="false" />
      <entry key="launch_browser_flag" value="false" />
    </userInput>
  </com.izforge.izpack.panels.userinput.UserInputPanel>
  <com.izforge.izpack.panels.install.InstallPanel id="install" />
  <com.izforge.izpack.panels.process.ProcessPanel id="process" />
  <com.izforge.izpack.panels.shortcut.ShortcutPanel id="shortcuts">
    <createMenuShortcuts>false</createMenuShortcuts>
    <programGroup>Klaros Testmanagement</programGroup>
    <createDesktopShortcuts>false</createDesktopShortcuts>
    <createStartupShortcuts>false</createStartupShortcuts>
  </com.izforge.izpack.panels.shortcut.ShortcutPanel>
  <com.izforge.izpack.panels.finish.FinishPanel id="finish" />
</AutomatedInstallation>
```

Example 3.1. Sample auto-install.xml

3.6. Starting the application

The Klaros-Testmanagement application is executed within a Tomcat application server. It can be started from the command line under Windows with the following call:

```
<install-dir>/bin/startup.bat
```

Under Linux this is:

```
<install-dir>/bin/startup.sh
```

If the creation of start menu entries or desktop shortcuts was selected during installation, these too can be used to start Klaros-Testmanagement of course.

3.7. Stopping the application

A running Tomcat application server can be stopped under Windows with the following call from the command line:

```
<install-dir>/bin/shutdown.bat
```

Under Linux this is:

```
<install-dir>/bin/shutdown.sh
```

If the creation of start menu entries or desktop shortcuts was selected during installation, these too can be used to stop Klaros-Testmanagement of course.

3.8. Accessing the application

To access Klaros-Testmanagement, open the following URL in a web browser: `http://${host}:${port}/klaros-web/` where `${host}` is the IP address or domain name of the application server and `${port}` is the HTTP port defined in Section 3.3.7, “Step 6: User Data”.

Default URL (local)

Assuming the default installation parameters have been used, Klaros-Testmanagement can be accessed locally using the URL http://localhost:18080/klaros-web/.

3.9. Update Process

Klaros-Testmanagement can easily be updated by just installing the new version. Existing settings and data are automatically retained. Further information on the installation process can be found in Section 3.3, “Installation”. The Klaros-Testmanagement home folder will remain untouched, so all settings, database configuration and the content repository will be unaffected by the update process.

Important

The Tomcat application server must be shutdown before starting the update. Otherwise, the installation process cannot be finished successfully.

Backup your data before updating!

It is highly recommended creating a backup of both the Klaros home folder and your database before starting the update process.

Most database products offer utilities for backing up database schemes, so please refer to the relevant sections in the database vendor documentation for this procedure.

In case you are still using the pre-configured Apache Derby database no further action is required and backing up the Klaros home folder is sufficient.

3.10. Important Files and their Locations

There are two important directories in a Klaros-Testmanagement installation. One is the **installation folder** where the Tomcat application server and the web application will be installed. The other is the **home folder** containing runtime data like configuration files, search indices and at-

tachments. Per default, this folder is named `.klaros` and located in the home folder of the user running Klaros-Testmanagement.

To allow a simple upgrade the installation/upgrade process will only change files in the installation folder and leaves the content of the home folder untouched.

On Microsoft Windows systems, by default the home folder is located at `C:\Users\<username>\.klaros`. This can also be addressed by entering `%UserProfile%\.klaros` in an explorer window address bar or by entering `cd %UserProfile%\.klaros` on the command line.

On Linux systems this folder can be found using the path `~/.klaros` and is generally located at `/home/<username>/.klaros`.

Relocating the Klaros Home Folder

By setting the environment variable `KLAROS-HOME` before starting the application the location of the home folder can be moved to another location.

```
export KLAROS_HOME=/var/application-data/klaros
```

Heads up When Moving the Klaros Home Folder!

If the application does not find a previous installation in the home folder it will bootstrap an empty default installation including a new local database. If this happens accidentally, e.g. by specifying an invalid Klaros home directory, it seems that the previous data has disappeared.

Of course this is not the case. To resolve this situation point the KLAROS-HOME variable to the correct location and restart.

3.10.1. Log Files

The complete log files for Klaros-Testmanagement are located in the directory `logs` in the installation directory and also in compressed form in the directory `logs` in the Klaros home directory.

Attaching these log files to a bug report helps the support team to quickly identify and fix any problems you encounter.

3.10.2. Database Settings

The `.klaros/hibernate.properties` file is used to specify the database to be used for Klaros-Testmanagement. It defines the driver class and the URI under which the database is accessible as well as the required authentication information. For more information about changing the database settings, see Section 3.11, “Changing the Default Database”.

3.10.3. Language Files

The Klaros-Testmanagement user interface may be customized by adding support for additional languages. This process is described in Section 4.1.1, “Defining Language Files”. The language files can be found under `.klaros/resources/messages`.

3.10.4. The Quotes File

Klaros-Testmanagement displays a daily changing *quote of the day* on the login page. These quotes are stored in the `.klaros/resources/quotes.txt` file. For more information, see Section 10.4.4, “Interface” and Section 4.2, “Quote of the Day”.

3.10.5. The Derby Database

If you are using the default Apache Derby database, the database files are located in the `.klaros/klarosDerby` folder.

3.10.6. The Content Repository

All attachments and custom reports are saved in the `.klaros/contentRepository` folder. If the Klaros-Testmanagement installation and database are migrated to a different location, this folder **must** be moved with them.

3.11. Changing the Default Database

By default Klaros-Testmanagement uses the Apache Derby Database which requires no further installation or configuration but is severely lacking performance in comparison to other databases. For productive use, it is therefore strongly recommended to use a more powerful database system, such as the open source database servers PostgreSQL or MariaDB/MySQL or the commercial Microsoft SQLServer database product.

To switch to another database system, Klaros-Testmanagement must be stopped, and the file `hibernate.properties` located in `<user.home>/.klaros/hibernate.properties` must be edited as shown below.

The `hibernate.connection.url` property must match the location of the database in your network. Please consult your database administrator for the `hibernate.connection.username` and `hibernate.connection.password` credentials to use.

Warning

Blank spaces at the end of lines in the hibernate.properties file might get misinterpreted by hibernate! For example, 'hibernate.connection.password=root ' is not the same as 'hibernate.connection.password=root' (ignoring quotes).

An exhaustive list of all parameters can be found in the Hibernate Core Manual.

3.11.1. MariaDB

For MariaDB change the content of the file to:

MariaDB 10.3 or later:

```
hibernate.connection.driver_class=org.mariadb.jdbc.Driver
hibernate.connection.url=jdbc:mariadb://localhost:3306/klaros
```

```
hibernate.connection.username=root
hibernate.connection.password=root
```

3.11.2. Microsoft SQL-Server

To use a Microsoft SQL-Server change the content of the file to:

Microsoft SQL-Server 2008 or later Versions:

```
hibernate.connection.driver_class=com.microsoft.sqlserver.jdbc.SQLServerDriver
hibernate.connection.url=jdbc:sqlserver://localhost:1433;databaseName=KLAROS
hibernate.connection.username=root
hibernate.connection.password=root
```

3.11.3. MySQL

For MySQL change the content of the file to:

MySQL 5.5 or later versions:

```
hibernate.connection.driver_class=com.mysql.jdbc.Driver
hibernate.connection.url=jdbc:mysql://localhost:3306/klaros
hibernate.connection.username=root
hibernate.connection.password=root
```

Important

When using a MySQL database, it is important to set the following option in the my.ini file.

```
# The maximum size of a query packet the server can handle as well as
# maximum query size server can process (Important when working with
# large BLOBs).  enlarged dynamically, for each connection.
max_allowed_packet = 64M
```

3.11.4. PostgreSQL

To use PostgreSQL change the content of the file to:

PostgreSQL 9.4 or later versions:

```
hibernate.connection.driver_class = org.postgresql.Driver
hibernate.connection.url = jdbc:postgresql://localhost/klaros
hibernate.connection.username=root
hibernate.connection.password=root
```

3.11.5. Apache Derby

To switch back to the integrated Derby database:

```
hibernate.connection.driver_class=org.apache.derby.jdbc.EmbeddedDriver
hibernate.connection.url=jdbc\:derby\:${user.home}/.klaros/klarosDerby;create\=true
hibernate.connection.username=root
hibernate.connection.password=root
```

3.11.6. Setting up the Database Instance

Creating a Database Instance

Klaros-Testmanagement will not automatically create either the database instance (`klaros` in the above example) or the database user (user `root` with password `root` in the above example) in the database server.

Creating a database instance and adding a user is described in the corresponding database manual and will not be covered here. The database user needs permissions to create, drop and alter tables to properly bootstrap the Klaros-Testmanagement database instance.

Make sure to activate support for UTF-8 Character Sets!

Depending on your database product it may be needed to manually activate support for UTF-8 character sets, which may be needed to support languages with uncommon character sets. Especially older versions of MySQL and MariaDB are known for coming with a limited character support in their default configuration. The following example shows how to activate this by editing the `mysql.cnf` configuration file.

```
character-set-server=utf8
collation-server=utf8_general_ci
```

Example 3.2. Additional entries in mysql.cnf to support UTF-8 Character Sets

3.12. Installing as a System Service

For the long-term use of Klaros-Testmanagement it is essential that the application is available immediately after restarting the computer on which it is installed. Depending on the operating system, different measures have to be taken to achieve this.

3.12.1. Installing as a Linux Service

Depending on your Linux distribution system services are launched using different init daemons.

We are providing sample scripts for the most common ones here, SysVInit and systemd.

3.12.1.1. Installing as a SysVInit Service

To start Klaros-Testmanagement automatically, the startup script shown below should be saved to the `/etc/init.d` directory as `klaros` and selected for the appropriate run level. Please refer to the documentation of your linux distribution for details.

The values of the `KLAROS_USER` and `KLAROS_PATH` fields must first be replaced with the intended user id and installation path. It is important that the script is executable. This can be ensured with the command **chmod a+x klaros**.

The service can now be started via **service klaros start** and stopped via **service klaros stop**. **chkconfig --add klaros** enables the start of the service at system boot.

```
#!/bin/bash
#
#
# This will be ignored by systems that don't use chkconfig.
# chkconfig: 345 98 2
# description: klaros (inside Tomcat, a servlet container)
#
### BEGIN INIT INFO
# Provides:          klaros
# Required-Start:
# Required-Stop:
# Default-Start:     3 4 5
# Default-Stop:      0 1 2 6
# Short-Description: klaros
# Description:       klaros-testmanagement
### END INIT INFO
#
KLAROS_USER=klarosdemo
KLAROS_PATH="/opt/klaros-testmanagement"
/bin/su - "${KLAROS_USER}" -c "${KLAROS_PATH}/bin/catalina.sh $@ >/var/tmp/klaros.log 2>&1"
```

3.12.1.2. Installing as a Systemd Service

To start Klaros-Testmanagement automatically the startup script shown below should be saved to the `/etc/systemd/system` directory as `klaros.service`.

The values of the `User`, `ExecStart` and `ExecStop` entries must first be replaced with the intended user id and installation path.

The service can now be started via **systemctl start klaros** and stopped via **systemctl stop klaros**. **systemctl enable klaros** enables the start of the service at system boot.

```
[Unit]
Description=Klaros Testmanagement Service
After=syslog.target network.target

[Service]
Environment="JRE_HOME=/opt/klaros-testmanagement/jre"
Environment="PATH=/opt/klaros-testmanagement/jre/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin"
Type=forking
User=klaros

ExecStart=/opt/klaros-testmanagement/bin/startup.sh
ExecStop=/opt/klaros-testmanagement/bin/shutdown.sh

[Install]
WantedBy=multi-user.target
```

3.12.2. Installing as a Windows Service

Under the Windows operating system family the supplied **service.bat** script can be used to install the Tomcat application server as a Windows service.

Please refer to the official Apache Tomcat 9 Windows service HOW-TO for further details.

Changing the Service User Account

Per default, the created Windows service will use the `LocalSystem` user account to run the Klaros-Testmanagement application. This setting must be changed in the service account settings to the account of the user you intend to use for running Klaros-Testmanagement. This change is needed due to the fact that the default location for the `.klaros` folder will be located in the user account home.

Skipping this vital step will lead to a newly created `.klaros` home folder in the `C:\Windows\System32\systemuser` folder (this location may vary with your Windows version in use; newer version may use `C:\Windows\system32\config\systemprofile` instead). The service then uses the newly created, empty database instead of the database that is used when Klaros-Testmanagement is started manually.

If you encounter this situation, then shut down the service, delete the `.klaros` folder in the `Windows\System32` folder, change the user account settings of the service and finally restart the service.

Adjusting Memory Settings and Java Options

Depending on the database size and number of parallel users, Klaros-Testmanagement may require a larger amount of memory than the installer suggests with the provided default values. If you want to change the specified memory settings for an already installed installation you may supply these settings in the **tomcat9w** administration interface as shown below.

In addition, the following Java 9 options must be set to ensure smooth operation of the software:

```
--add-opens=java.base/java.lang=ALL-UNNAMED
--add-opens=java.base/java.io=ALL-UNNAMED
--add-opens=java.base/java.util=ALL-UNNAMED
--add-opens=java.base/java.util.concurrent=ALL-UNNAMED
--add-opens=java.base/java.net=ALL-UNNAMED
--add-opens=java.rmi/sun.rmi.transport=ALL-UNNAMED
```

To ensure that indeed the settings of the installed service are changed, add the service name as a command line argument as follows: **[path]\Klaros-Testmanagement\bin\Tomcat9w.exe //ES//Tomcat9**.

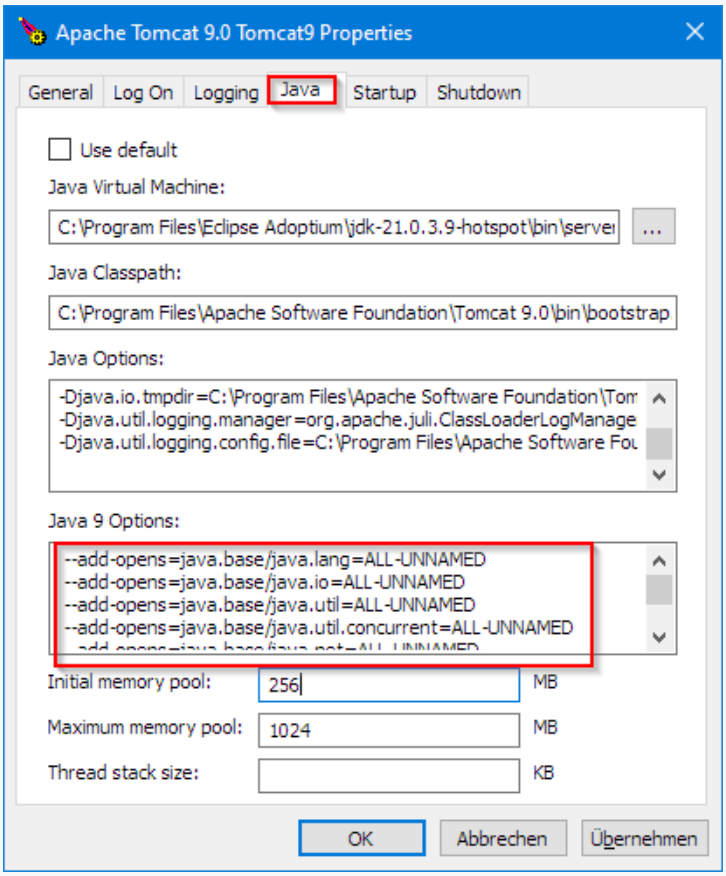


Figure 3.13. Additional Tomcat Memory and Java Option Settings

Additional Settings

The content repository (see Section 3.10.6, “The Content Repository”) requires another setting in the **tomcat9w** administration interface. Here the parameter **-Doak.locksupport=disabled** must be added.

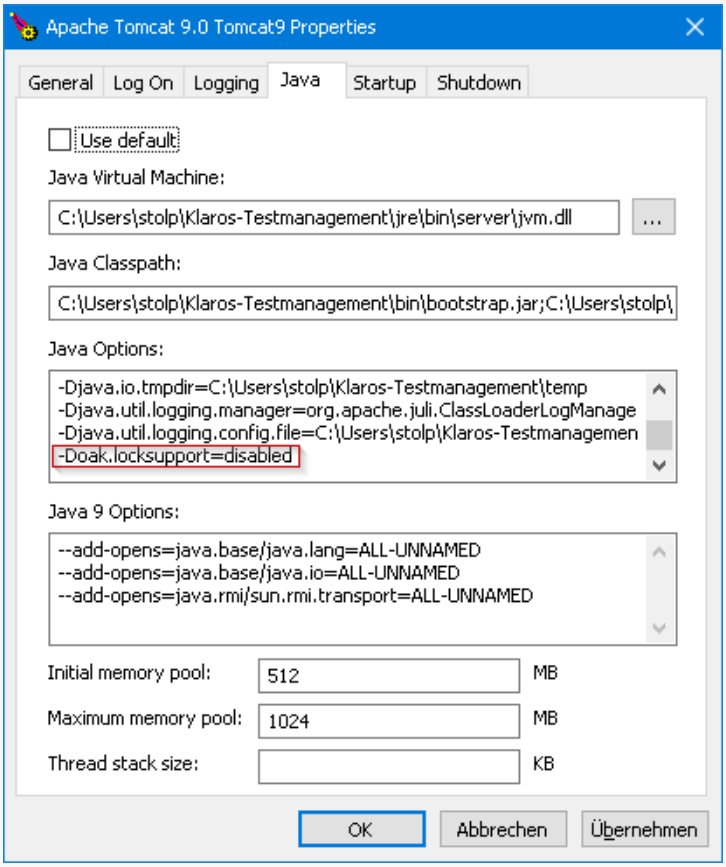


Figure 3.14. Additional Tomcat Settings

3.13. Monitoring

Using the built-in JMX Support in Tomcat it is possible to monitor the Klaros-Testmanagement installation using any JMX capable Monitoring tool (like e.g. VisualVM).

To do this, Add the following lines to `Klaros-Testmanagement/bin/catalina.sh`:

```
# Set JMX connector
RMI_SERVER='[FQDN]'

JMXREMOTE="-Dcom.sun.management.jmxremote"
JMXREMOTE="$JMXREMOTE -Dcom.sun.management.jmxremote.port=[PORT]"
JMXREMOTE="$JMXREMOTE -Dcom.sun.management.jmxremote.ssl=false"
JMXREMOTE="$JMXREMOTE -Dcom.sun.management.jmxremote.authenticate=false"
JMXREMOTE="$JMXREMOTE -Dcom.sun.management.jmxremote.ssl=false"

JAVA_OPTS="$JAVA_OPTS $JMXREMOTE -Djava.rmi.server.hostname=$RMI_SERVER"
```

Where `[FQDN]` and `[PORT]` denote the fully qualified domain name and port of the remote JMX server you want to deliver messages to.

3.14. Configuring External Issue-Management-Systems

Depending on the type of issue management system, some additional configuration may be needed to connect it to Klaros-Testmanagement. This section describes the needed configuration steps for each system.

Name	Version
Bugzilla	3.0.9 or later
GitHub	Any
GitLab	API V3/V4
Jira	3.7 or later
Mantis	1.1.8 or later
Redmine	3.0 or later
Trac	0.10 or later
YouTrack	2021.3 or later

Table 3.3. Supported External Issue-Management-Systems

3.14.1. Jira Configuration

For current Jira versions, no further configuration actions are necessary.

Jira Cloud Instances

Starting April 25th 2019 Atlassian limited access to their cloud instances to token based authentication, so authenticating via username and password is no longer supported.

Klaros-Testmanagement is supporting token based authentication scheme as well. Just use your email address as username and the token as password.

3.14.1.1. Jira Legacy Server (Version 4.2 or lower)

To allow the connection to a Jira legacy server instance the Jira RPC Plugin has to be activated in your Jira installation. You find this option as *Accept remote API calls* in **General Configuration** under **Global Settings**. Then you need to enable the JIRA RPC Plugin in **Plugins** under **System** in the left-hand menu.

3.14.2. Redmine Configuration

Enable REST Web Service

In order to connect to a Redmine installation, the *Authentication Required* and the *Enable REST Web Service* checkbox in the Authentication section of the Redmine administration settings must be activated (Figure 3.15, “The Redmine Authentication Section”).

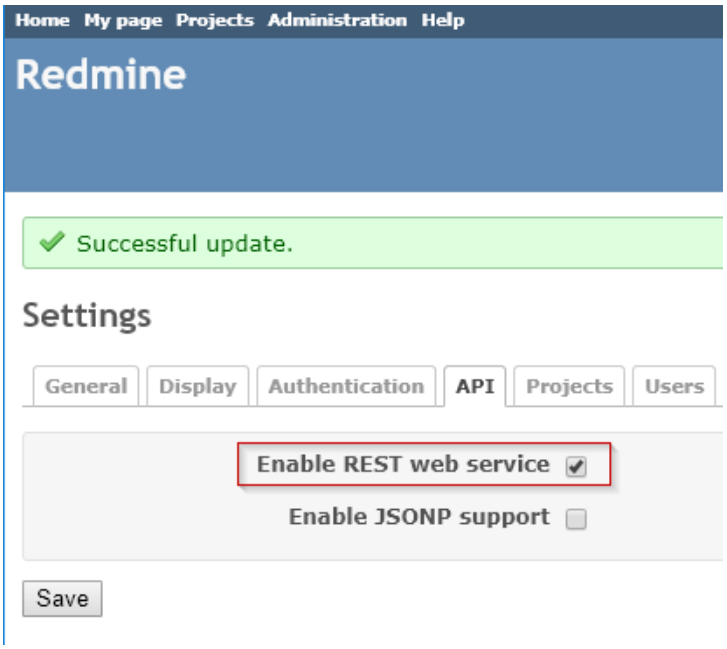


Figure 3.15. The Redmine Authentication Section

3.14.3. Trac Configuration

The Trac integration requires the installation of the TracXMLRPC Plugin. The download archives and installation notes are available at http://trac-hacks.org/wiki/XmlRpcPlugin/. After successful installation of the plugin, it has to be activated using the Trac web administration interface. Figure 3.16, “The Trac Plugin Section” shows the Trac Plugin section after the successful installation of the plugin.

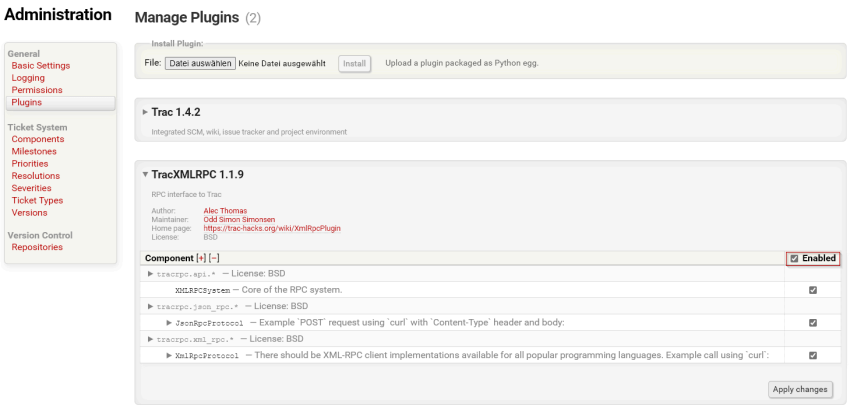


Figure 3.16. The Trac Plugin Section

To make the plugin accessible, authenticated users must be able to access the XML_RPC plugin.

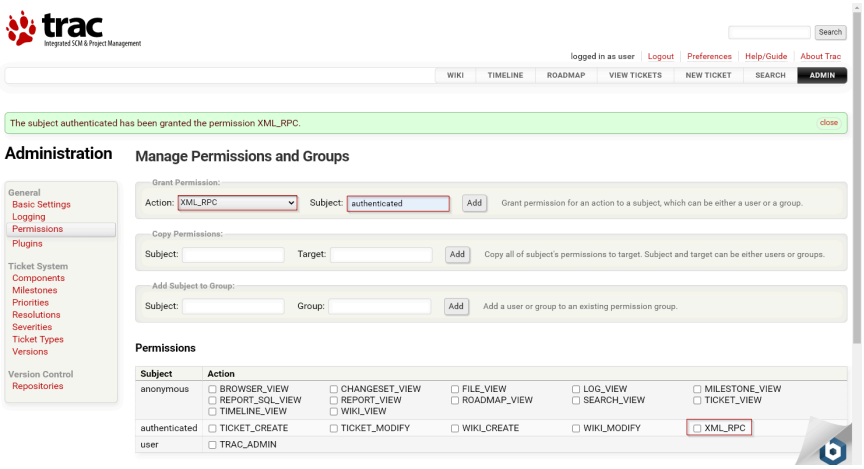


Figure 3.17. The Trac Permission Section

The following combinations have been successfully tested:

- Trac 0.12 / TracXMLRPC 1.1.0

Later versions are expected to work but currently not tested.

3.15. SSL Support

Note on self-signed certificates: If you are accessing HTTPS servers with self-signed certificates, you will need to import those certificates into your Java trusted keystore. One instruction on how to achieve this can be found at http://stackoverflow.com/questions/2893819/telling-java-to-accept-self-signed-ssl-certificate .

3.16. Upgrading from Version 4

This section describes important changes for users which are upgrading to version 5 from version 4. Please read this fully before attempting an upgrade. If you have questions about the upgrade process, please contact us at support@verit.de.

3.16.1. Required Version for Upgrade

For a successful upgrade to version 5, it is mandatory that the installed Klaros-Testmanagement version is 4.12.0 (released July 29th, 2019) or higher. If this is not the case, the application will refuse to start, and you will see a corresponding entry in the console and log. In this case, please download the latest available 4.xx version first, install it and update your installation by logging in with administrator credentials. After this, the upgrade to version 5 is possible.

3.16.2. Unattended Database Upgrade

In version 4 it was mandatory that an administrator triggered a database update after a new version has been installed. In version 5 this is no longer required, the system is now fully usable directly after an update has been made.

3.16.3. Startup Time for Initial Upgrade to Version 5

Version 5 rebuilds existing databases from previous installations on the first start. Depending on the size of the existing data and the processing power of your database installation this process can take from some minutes to up to an hour.

Please reserve a reasonable time frame for this and show some patience during startup. Each step of the rebuild process is logged on the console and/or logfile.

3.16.4. Embedded Java Runtime Environment

In contrast to previous versions it is no longer required to install a Java Runtime Environment prior to installing Klaros-Testmanagement. Klaros-Testmanagement is now packaged with a Java 11 runtime that is automatically used.

3.16.5. Automatic Database Version Detection

As of version 5, the database version used is automatically detected and should no longer be specified in the `<user.home>/.klaros/hibernate.properties` file. Please remove the `hibernate.dialect=...` line from this file before you perform the upgrade.

3.16.6. Oracle Database no Longer Supported

In version 5 we have discontinued support for Oracle databases. If you have an existing Oracle installation you like to migrate to another DBMS please contact us at support@verit.de for further instructions.

3.17. Uninstall

To uninstall Klaros-Testmanagement select *Klaros-Testmanagement Server Uninstaller* from the start menu (if running under Windows) or just delete the installation folder manually.

If you have added an installation as a service as described in Section 3.12, “Installing as a System Service” please remember to deactivate the Windows service or Linux init script.

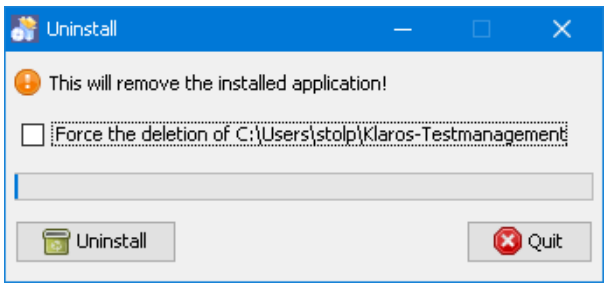


Figure 3.18. The Uninstaller

The Home Folder

Just like during the upgrade process uninstalling will not touch your home folder (`<user.home>/.klaros`).

So if you uninstalled by accident re-installing will let you restore your system safely, else remove feel free to remove that folder manually.

Chapter 4. Customization

This chapter shows how to customize the displayed language and the file for the “Quote of the Day”.

4.1. Languages

Klaros-Testmanagement comes with language files in German and English. Support for additional languages can be added by using additional language files if needed. The corresponding language file must be stored in the `.klaros/resources/messages` directory for this.

Language files are activated in the application under Configure / System / Languages. Detailed information can be found under Section 10.4.5.1, “Enabling and Disabling Languages”.

4.1.1. Defining Language Files

Adding a valid language file to the `.klaros/resources/messages` directory causes the language to become available in Klaros-Testmanagement. A valid language file has a name of the form `messages_${language}_${COUNTRY}_${variation}.properties` where `${language}` is the two-letter ISO 639-1 code for the language in lower-case, `${COUNTRY}` is the two-letter ISO 3166 code for the country in upper-case and `${variation}` is an optional, user-definable code for the variation of the language (e.g. dialect).

Language File Encoding

The language file must use ISO-8859-1 encoding with Unicode escapes (`\uXXXX`).

If you prefer editing these files as UTF-8 please use a converter like **native2ascii** or **iconv** to translate them to a proper format.

After Klaros-Testmanagement is run for the first time, the American English, German and fallback language files will be copied to the `.klaros/resources/messages` folder. In order to create a new language file, the text should be copied out of one of these files into an appropriately named file. The contents of the file consist of key value pairs in the form: `key=This sentence is the value`. None of the keys should be altered. The values (the text after the '=') should be translated into the target language.

When Klaros-Testmanagement is started, the default language files (`messages.properties`, `messages_en_US.properties` and `messages_de_DE.properties`) are copied to the `.klaros/resources/messages` folder, overwriting any changes made to these files. Therefore, it is advised that any user changes to the American English and German interfaces be made in a file with the variation in the filename set, e.g. `messages_en_US_companyName.properties` in order to persist changes.

As well as copying over the default files, Klaros-Testmanagement also merges any changes with user-defined files. If key-value pairs are added to the default files during an update, these will also be added to the end of any user-defined language files, along with a comment stating when the entries were added. These entries should be translated by the creator of the custom language file.

4.2. Quote of the Day

Klaros-Testmanagement displays a new “Quote of the day” on the login screen every day. The default quotes can be extended by own entries or deactivated. The quote file can be uploaded and activated in Klaros-Testmanagement on the page *General Settings* directly into the folder `.klaros/resources` in the user's home directory.

This file is a `.txt` document in ISO-8859-1 encoding and uses one line per quote. Section Section 10.4, “System” describes how to install this file. The respective quotes change daily.

Chapter 5. Functional Overview

This chapter introduces the user interface and its basic functions.

5.1. Login

Figure 5.1 shows the login screen of Klaros-Testmanagement. To login enter the username in the *Username* field and the associated password into the *Password* field.

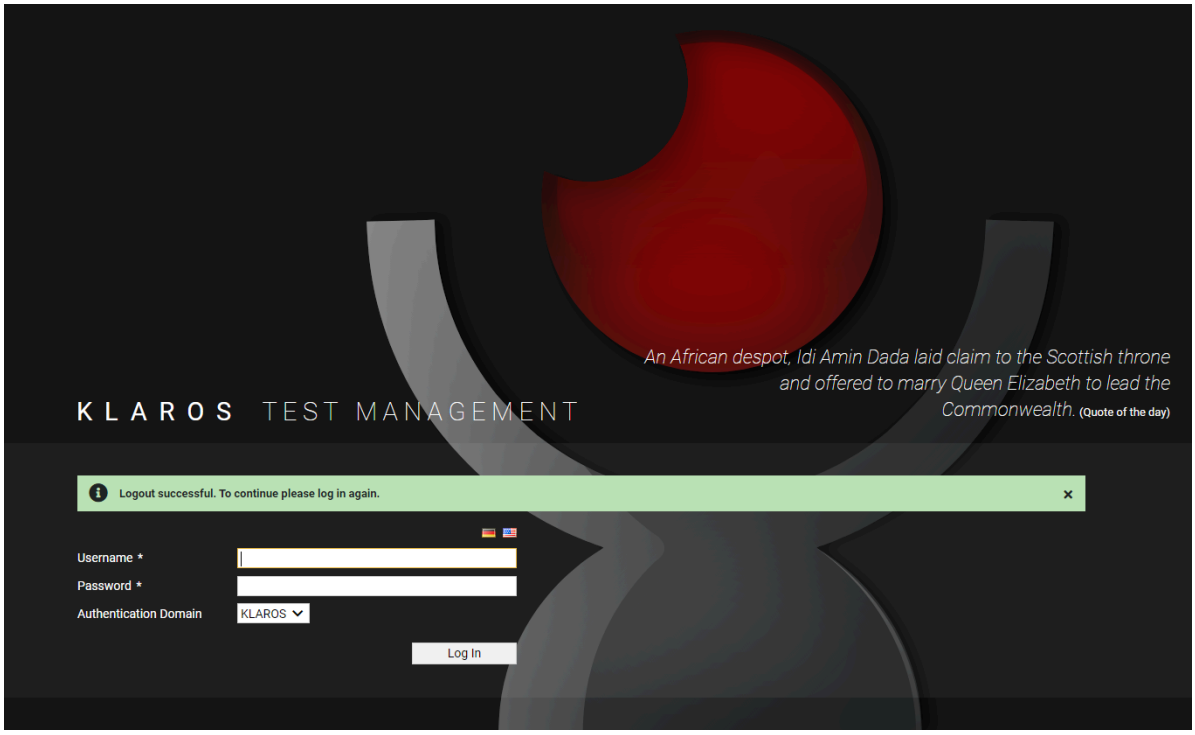


Figure 5.1. The "Login" Screen

Three user accounts are predefined in the database with the following roles:

Administrator Username: `admin` / Password: `admin`

Manager Username: `manager` / Password: `manager`

Tester Username: `tester` / Password: `tester`

Confirm the login to Klaros-Testmanagement by clicking the Log In button.

For a description of the users roles and the permissions associated with each role, see Appendix A, *Access Permissions*.

enterprise edition Only available in Klaros-Testmanagement Enterprise Edition

The Klaros-Testmanagement Enterprise Edition also supports authentication using an external directory service (LDAP / Active Directory). For more information about configuring an external directory service, see Section 10.5.5, "LDAP".

Automated User Creation Option

If a user is authenticated for the first time using an external directory service, a user account with the *Tester* role can be created automatically for this user. The LDAP attributes for the username and the e-mail address are automatically transferred to the newly created user account in this case.

Setting the User Interface Language at Login

By default, the user interface is displayed in the language specified by the localization setting of the server operating system. Clicking on one of the flags to the right above the *username* text box also allows you to select a different language for the user interface.

After a successful login the *Projects* page is shown (Figure 5.2).

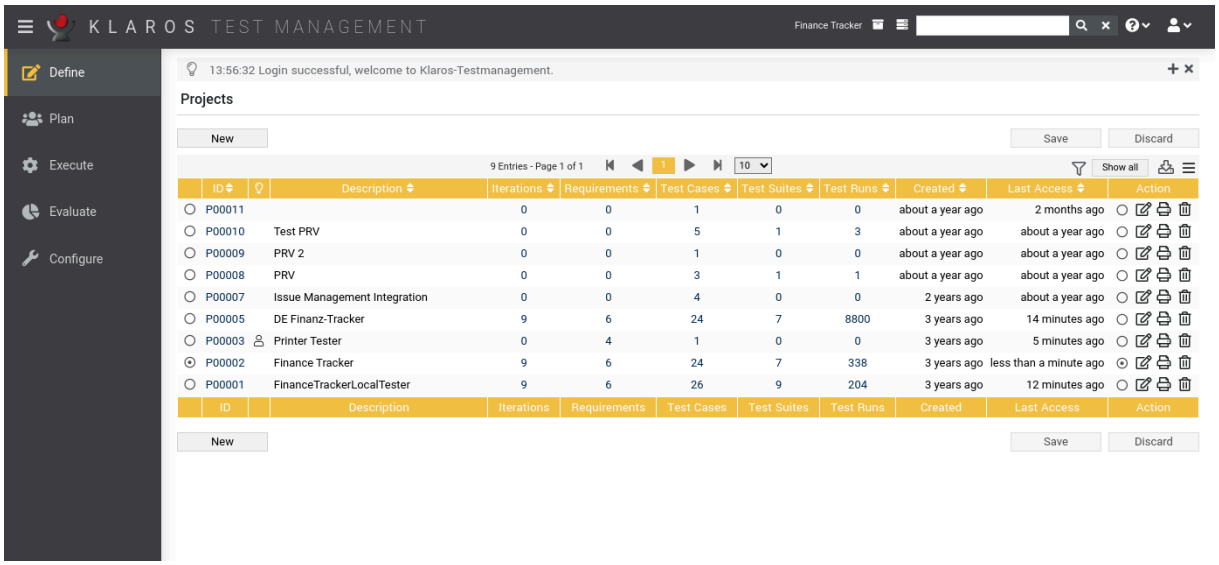


Figure 5.2. The “ Projects” Page

5.2. Page Overview

The individual elements of a screen page are presented below.

5.2.1. Side Menu

The entries in the menu in the left part of the screen are based on the activities in a test project. The individual sections are:

Define In the *Define* section, all objects required for a test project are created, edited and managed. The objects managed here include the project itself, test environments, systems under test, test cases with their individual test steps, and test suites. In

the Klaros-Testmanagement Enterprise Edition additional objects are iterations, requirements and test segments.

Plan — In the *Plan* section, jobs for executing or reviewing tests are created and managed. Tasks, can be assigned to individual users, and their progress can be tracked in detail. This section is only included in the Klaros-Testmanagement Enterprise Edition.

Execute — In the *Execute* section, jobs, manual test cases and test suites will be executed. Before a test can be executed, a system under test and a test environment must be created and selected. Klaros-Testmanagement guides the tester step by step through the manual test and automatically logs the results. Results and comments can be added to each test step. Tests can be interrupted and continued at a later time.

Evaluate — In the *Evaluate* section, a configurable dashboard and various reports provide an overview of the status of the tests. Both overview reports and fine-grained information are available so that each individual test execution can be tracked and visualized.

Configure — The *Configure* section is intended for administrative tasks, such as system settings, user management, creating report templates, integrating issue management systems, requirements management, LDAP and the like. Data backup and the import and export of projects are also performed here.

The following chapters will describe each category in detail.

5.2.2. Page Header

5.2.2.1. Search / Quick-Select

Depending on the Klaros-Testmanagement edition, you will find either a search box in the top right corner of the page (see Figure 5.3) for quick navigation to objects with a specific ID or for a full text search in the currently selected project.

Figure 5.3. The Search Field

5.2.2.1.1. Full Text Search

enterprise edition Only available in Klaros-Testmanagement Enterprise Edition

The *Search* field allows the user to search for object ids or words or phrases contained in fields of objects. This search function uses Apache Lucene, which provides a powerful syntax for searching individual fields or with wildcards. This syntax is described on the Apache Lucene website.

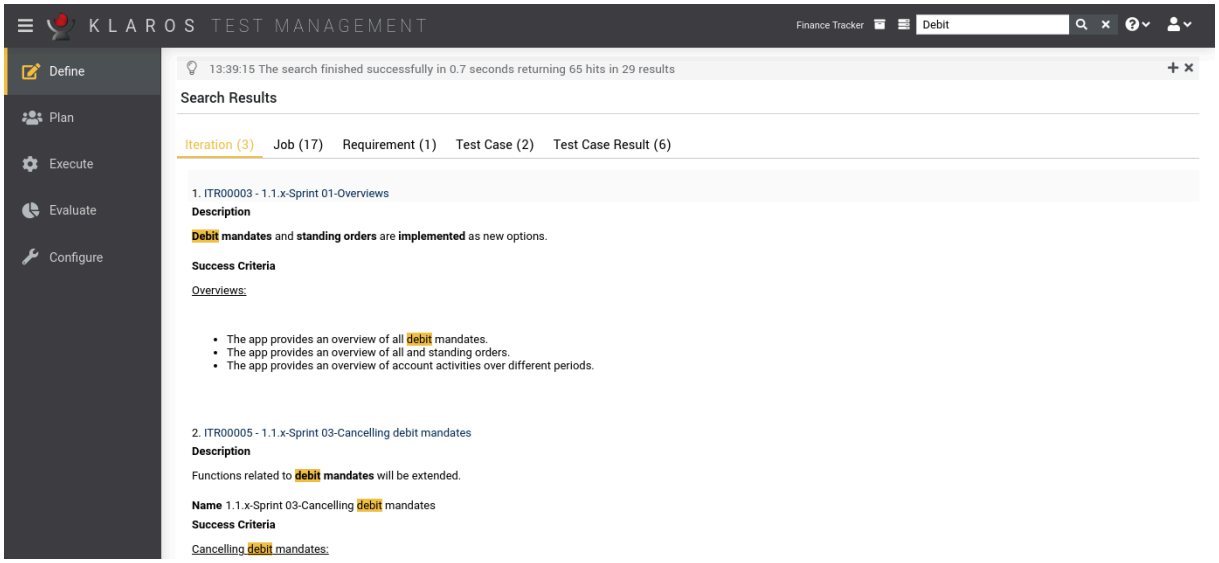


Figure 5.4. The “Search Results” Page

Figure 5.4 displays the *Search-Results* page. This page displays all occurrences per object type in a separate view. The search term is highlighted in color.

The status line displays the time needed for the search and the number of hits, see. Figure 5.5

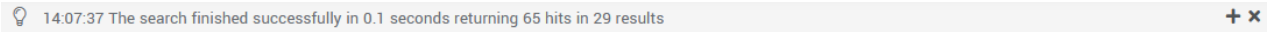


Figure 5.5. The Status Bar

5.2.2.1.2. Quick-Navigation

The full text search is only included in Klaros-Testmanagement Enterprise Edition. In the Klaros-Testmanagement Community Edition instead of the full text search a quick navigation field for known element IDs (like TC00001) is available. The following object types are supported there:

- Project (P)
- Test Environment (ENV)
- System under Test (SUT)
- Test Case (TC)
- Test Case Result (TCR)
- Test Suite Result (TSR)
- Test Run (TR)
- Issue-Management-System (IMS)

Please note, that ids will be searched for in the current project only.

Shortcuts in Quick Navigation

The quick navigation is case-insensitive and can also be abbreviated by omitting leading zeros. So `TC00001` , `TC1` or `tc1` all refer to the same element.

5.2.2.2. The Log Panel

Figure 5.6. The “Log Panel” Icon

The log panel displays status messages such as warnings or information. By default, only the last status message is displayed. Clicking the + icon on the right side of the log panel displays the status messages since the last login. Clicking the × icon closes the log panel.

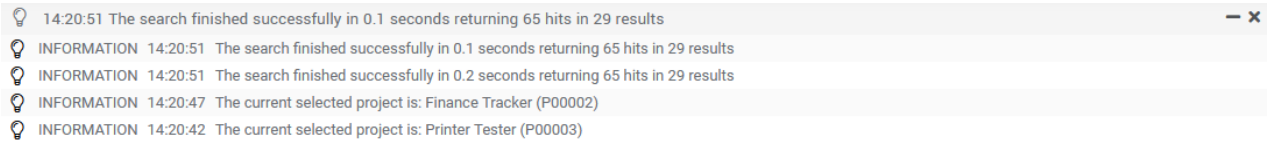


Figure 5.7. The Log Panel

5.2.2.3. The Help Menu

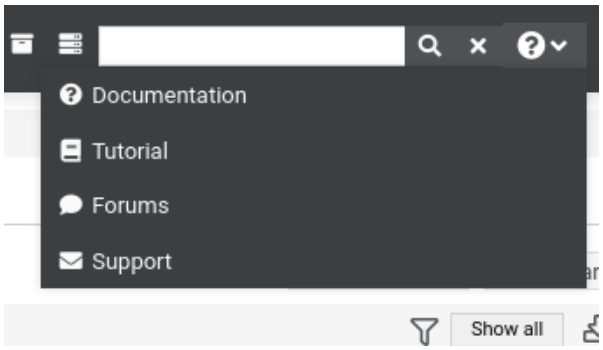


Figure 5.8. The Help Menu

The help menu provides quick links to various Klaros-Testmanagement resources. Pressing the help menu icon opens a menu with the following links:

Documentation — Opens the relevant page in the manual.

Tutorial — Opens up the Klaros-Testmanagement tutorial.

Message Boards — Opens up the Klaros-Testmanagement message boards.

Contact Support — Prepares an e-mail to be sent to the Klaros-Testmanagement support team. This option can be used when experiencing an issue using Klaros-Testmanagement which couldn't be solved using the documentation or the message boards.

5.2.2.4. The User Menu

Figure 5.9. The “User Menu” Button

The user menu icon on the upper right of Klaros-Testmanagement opens the user menu.

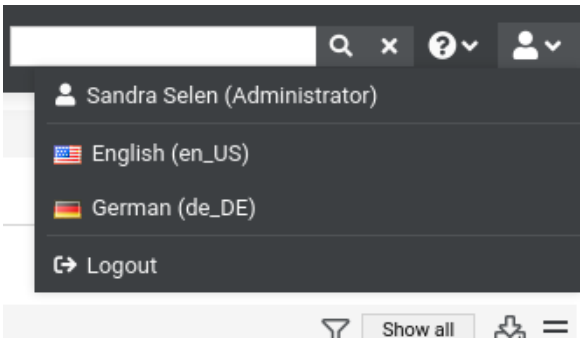


Figure 5.10. The opened user menu

The name and role of the currently logged-in user is displayed here. It is possible to change the user interface language and log out of the system.

5.2.3. Content

The content of a page changes due to navigation via the menu bar or other actions.

The pages generally have the same structure. The following two basic patterns exist: overview pages and detail pages

5.2.3.1. Overview Page

On an overview page several objects of the same type (e.g. test cases) are displayed and managed in a table.

5.2.3.1.1. Quick Table Filtering

Only available in Klaros-Testmanagement Enterprise Edition

On selected tables it is also possible to filter the listed entries by full text search as described in Section 5.2.2.1.1, “Full Text Search”.

By entering a search term in the input field above the table, the text fields of the displayed objects are searched for the specified search term and the list is filtered accordingly.

5.2.3.1.2. Filtering and Sorting

The objects displayed in a table can be filtered and sorted by their attributes. This includes attributes that are not directly visible in a list column. By clicking on the *Filter/Sort* icon above the table, a panel with two further tables opens. The left table contains the active filters, while the right table contains the active sort criteria. If a filter is used, the icon is marked with an orange color.

Figure 5.11. Number of Active Filter in Use

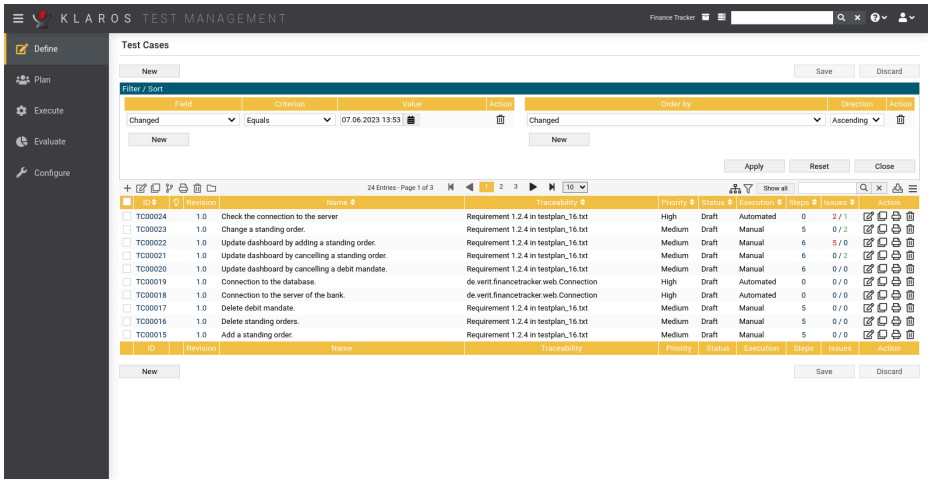


Figure 5.12. The Filter and Sort Panel

Below the two tables there are three buttons:

`Apply` applies the filters and orders to the table below.

`Reset` deletes all filters and sorting entries.

`Close` closes the panel.

Specifying Filtering Criteria

Each row in the filter table represents a filter which is applied to the table.

Clicking the `New` button adds a new, empty filter to the filter table. Each row has four columns:

- The *Field* column specifies the field to filter by. The fields that can be filtered by vary from object to object.
- The *Criterion* column defines the restriction type to be applied during filtering. The restriction types that can be selected in an entry will vary from field to field.
- The *Value* column specifies the value of a constraint, e.g. the user by which a field should be filtered.
- The *Action* column contains a button for deleting the filter criterion.

Multiple Selection

For some entries, such as users, multiple selection is also possible. To do this, hold down the CTRL key and select the desired entries individually.

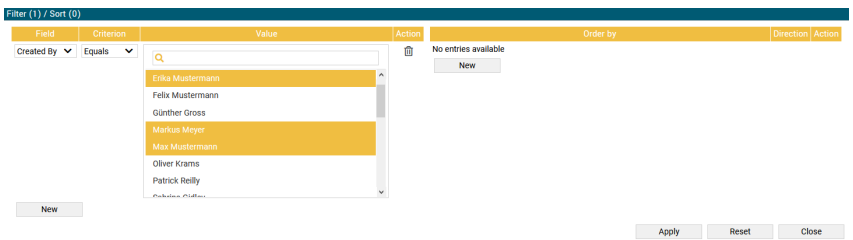


Figure 5.13. Filter by multiple selection

Specifying Sorting Criteria

Table entries can be sorted by multiple criteria. The sorting criteria are specified in the right table in the *Filter/Sort* panel.

It is possible to specify more than one sorting option. In this case, the top row of the table has the highest priority and the bottom row has the lowest priority.

Clicking the New button creates a new empty row for a sort criterion is created at the bottom of the table. To remove the row, use the Delete icon in the Action column.

The sorting options table contains three columns:

- The *Order by* column indicates the field of the listed objects to sort by.
- The *Direction* column defines the direction of the sort order, i.e. descending or ascending.
- The icon in the *Action* column removes the individual sort criteria.

5.2.3.1.3. Categorization

Only available in Klaros-Testmanagement Enterprise Edition

To manage a large number of objects, they can be categorized according to any criteria. This function is available for the following objects: *Iterations*, *Requirements*, *Systems under Test*, *Test Environments*, *Test Cases* and *Test Suites*. The categorization panel is opened by clicking the *Categories* icon above the table.

In the Categorization panel, *Category groups* and *Categories* can be created. Each category group contains any number of categories that can be arranged in a tree structure. Objects can be assigned to any category in each category group. The number of categories or category groups is not limited.

In order to apply categorization, the *Categories* panel must be open. It contains two different views: *Edit* and *Tree*. If no categories have been defined, only the edit view can be used. Each view allows selection of the active category group through the use of a dropdown menu, or selection of the blank category group to disable categorization.

Edit categories

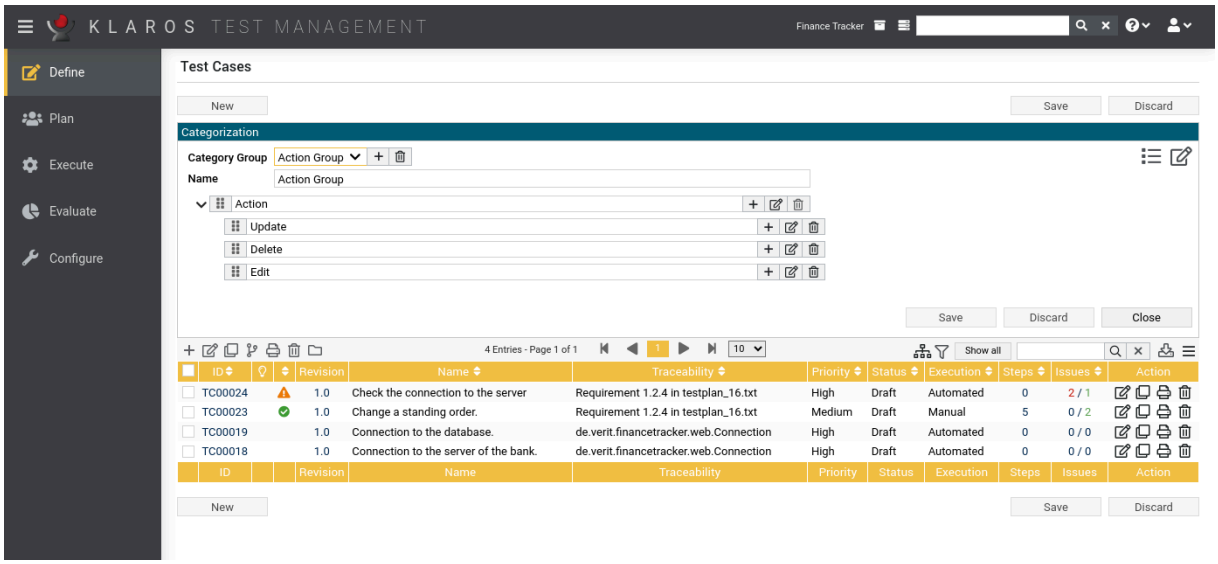


Figure 5.14. The “Categorization Edit” View

This view allows creating and editing category groups and categories as well as organizing category hierarchies. Clicking the + icon at the top of this view creates a new category group.

Categories can also be added, edited and deleted in this view. A category group always contains a root category that cannot be deleted. By clicking on the + a new subcategory is added to a category. Clicking on the 🗑 will delete a category.

Categories may be renamed by changing the value in the name field and also given a description through use of the ✎ edit button.

Changing the order and hierarchy of categories is also possible via drag and drop.

The Tree View

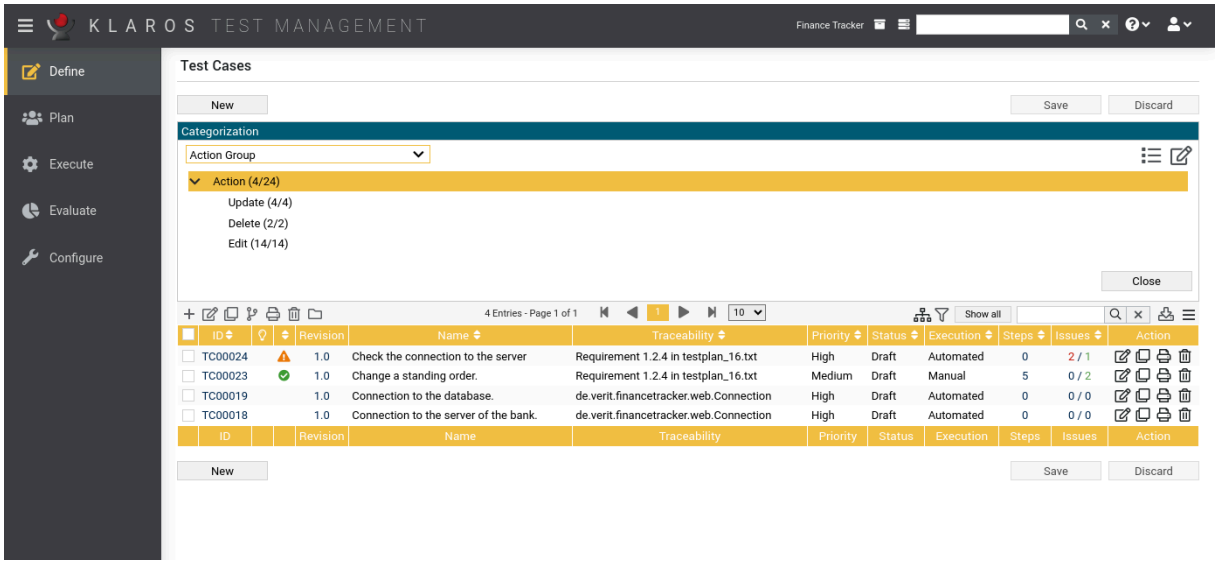


Figure 5.15. Select Categories

This view shows an overview of the category structure. The available categories are represented as a collapsible tree, with the name of each category is shown, along with the number of objects it contains. The description of each category is shown as a tooltip when its name is hovered over with the mouse pointer.

Assigning objects to categories

To assign objects to categories in a table, the *Categories* panel must be open and the *Tree* view must be enabled in it.

To assign objects first activate the category containing the objects that should be assigned and select the objects. Then click on the folder icon above the table. A dialog will appear. Choose the appropriate category to which you want to assign the objects and click the Assign button. The selected objects are now assigned to the chosen category.

Please note, that you can not assign objects from multiple categories to a single category at once. In case that the desired objects belong to different categories you must repeat the step for each category.

5.2.3.1.4. Display of Deleted Objects

If an object is deleted, it gets a deletion marker and is no longer displayed on the overview page. Administrators can view deleted objects by clicking the Show all button. Clicking on the button Only active will hide them again. This function is available for the following objects: *Iterations*, *Jobs*, *Projects*, *Requirements*, *Test Cases*, *Test Environments*, *Test Runs*, *Test Segments*, *Test Suites*, *Test Systems* and *Users*.

Automatic Saving of the Setting

The selection whether deleted objects should be displayed is automatically saved for each overview page and administrator.

5.2.3.1.5. Bulk Actions

Only available in Klaros-Testmanagement Enterprise Edition

In tables, bulk actions are often also available for the displayed objects. This means that several entries in the list can be selected and processed together. This applies, for example, to deleting, duplicating, editing or assigning to categories.

In order to use a bulk action, the user must first select one or more objects in the list using the check-boxes to the left of the table. The buttons above and to the left of the table then become active. Figure 5.16 shows the bulk action buttons on the test case overview page.

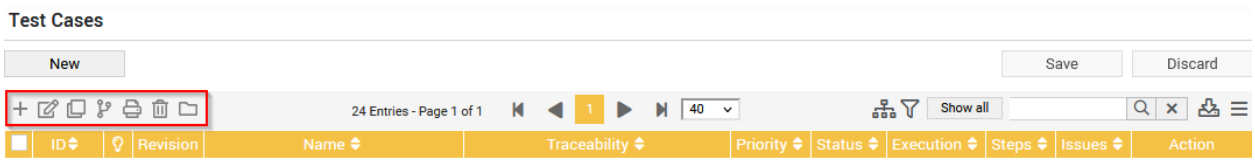


Figure 5.16. The Bulk Actions Section

Before a mass action can be executed, all changes in the table must be saved. If there are still unsaved changes, a dialog will appear.

A confirmation dialog is displayed once a bulk action icon has been clicked. These dialogs describe the action which will be carried out and sometimes contain input fields, e.g. to enter the revision comment for the bulk new revision action. All changes are written to the database as soon as this dialog is confirmed.

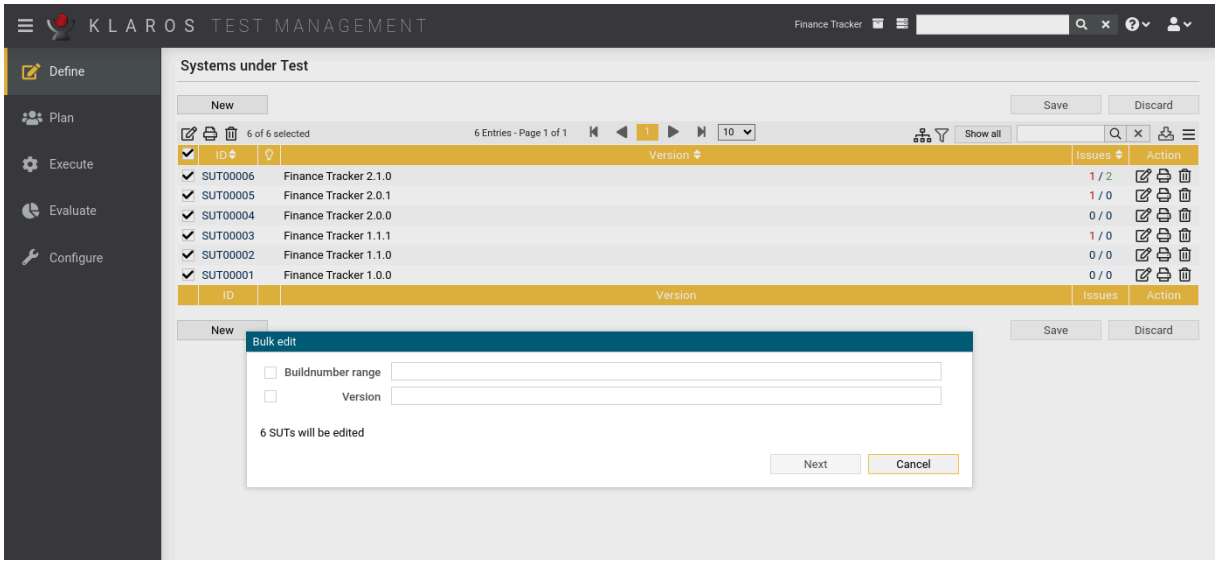


Figure 5.17. Bulk Editing Objects

5.2.3.2. Detail Page

A single object is displayed and managed on a detail page.

5.2.3.2.1. Print Pages

enterprise edition Only available in Klaros-Testmanagement Enterprise Edition

Objects can be displayed in a print-optimized representation on their own page by clicking on the icon. The icon can be found in the *Action* column in the table, in the list of mass operations at the top left of tables and as a large icon at the top right of *Detail pages*.

The display can be modified by the following parameters.

- The *Suppress empty fields* option hides fields content in the view.
- If the *Display diagrams* option is selected, diagrams are embedded in the print preview.
- With *Test environments* and *Systems under Test* the test environments and test systems can be selected whose results should be displayed on the page.
- The *Details* setting determines how detailed the data is displayed.

The selected print options will be preserved during a user session.

Figure 5.18 shows the print view for a test suite.

Browser print settings

The option *Print Backgrounds and Images* must be activated to achieve the best output quality.

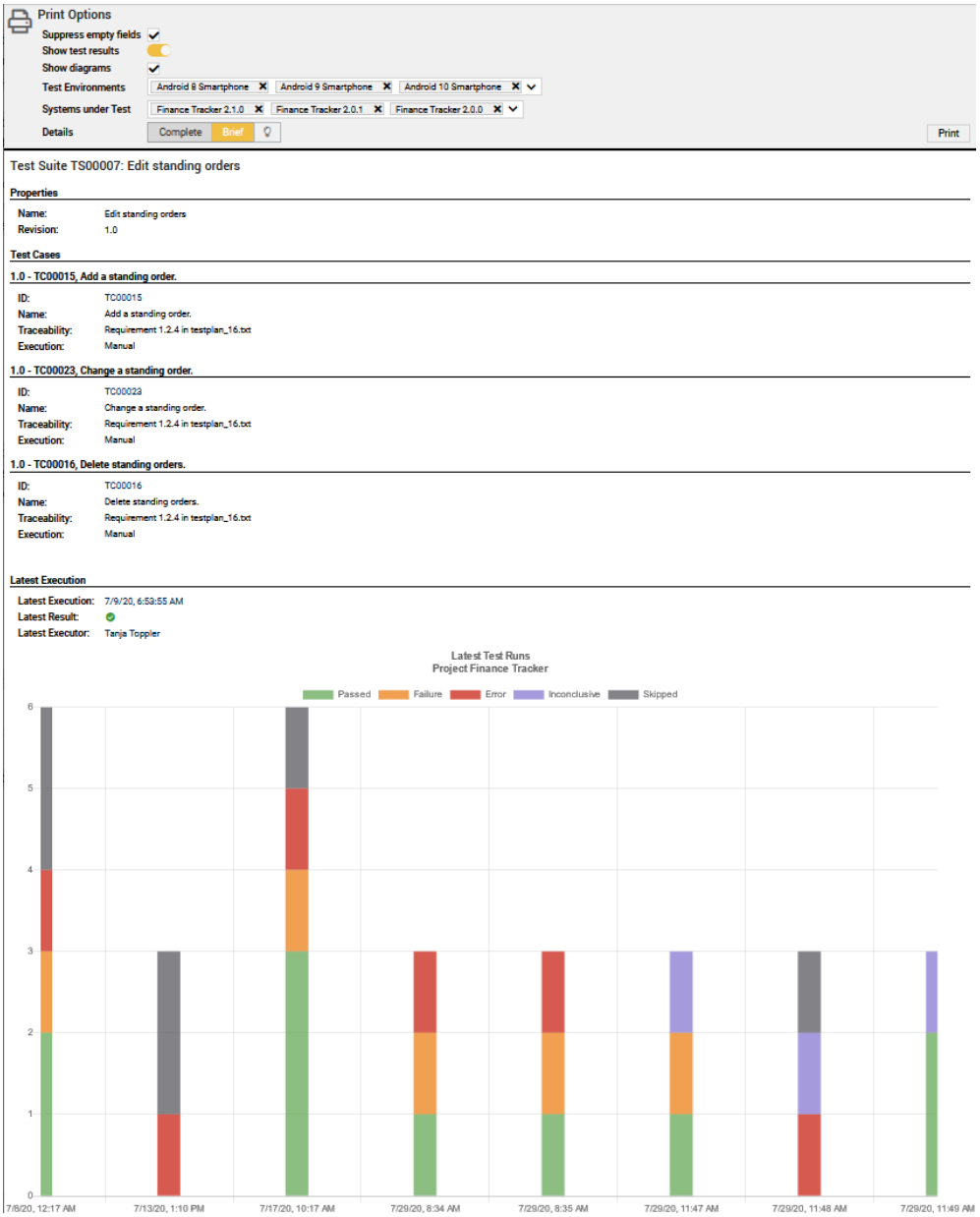


Figure 5.18. The “Test Suite” Print Page

5.2.3.2.2. Bookmarks

The detail page of any object can be accessed directly via a special URL generated by the system. By clicking on the icon this URL to the current page is copied to the clipboard and can be shared with any other application.

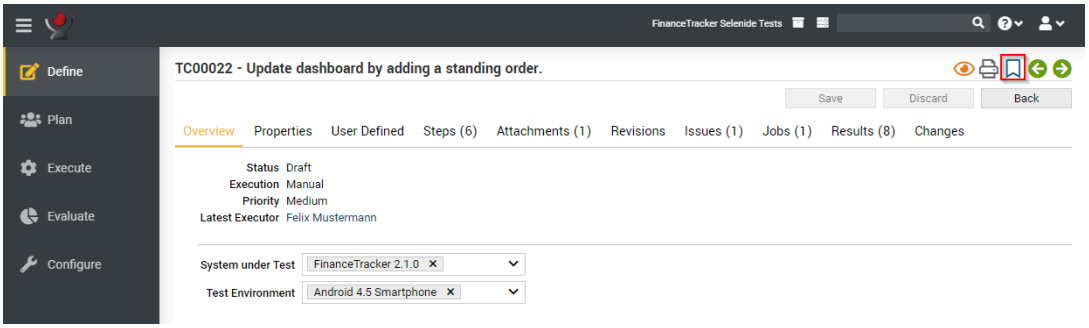


Figure 5.19. Creating Bookmarks

Section 6.1.2.5.1, “Assigning Project Specific Roles” explains the configuration of controlling access to bookmarked pages.

5.2.3.2.3. Overview

enterprise edition Only available in Klaros-Testmanagement Enterprise Edition

For every type of object in Klaros-Testmanagement there is a dedicated overview tab which shows the most important data for this object at a glance. In addition to plain data, many overview pages contain various tables and diagrams, like for example the success history for test suites, which helps to visualize the test progress.

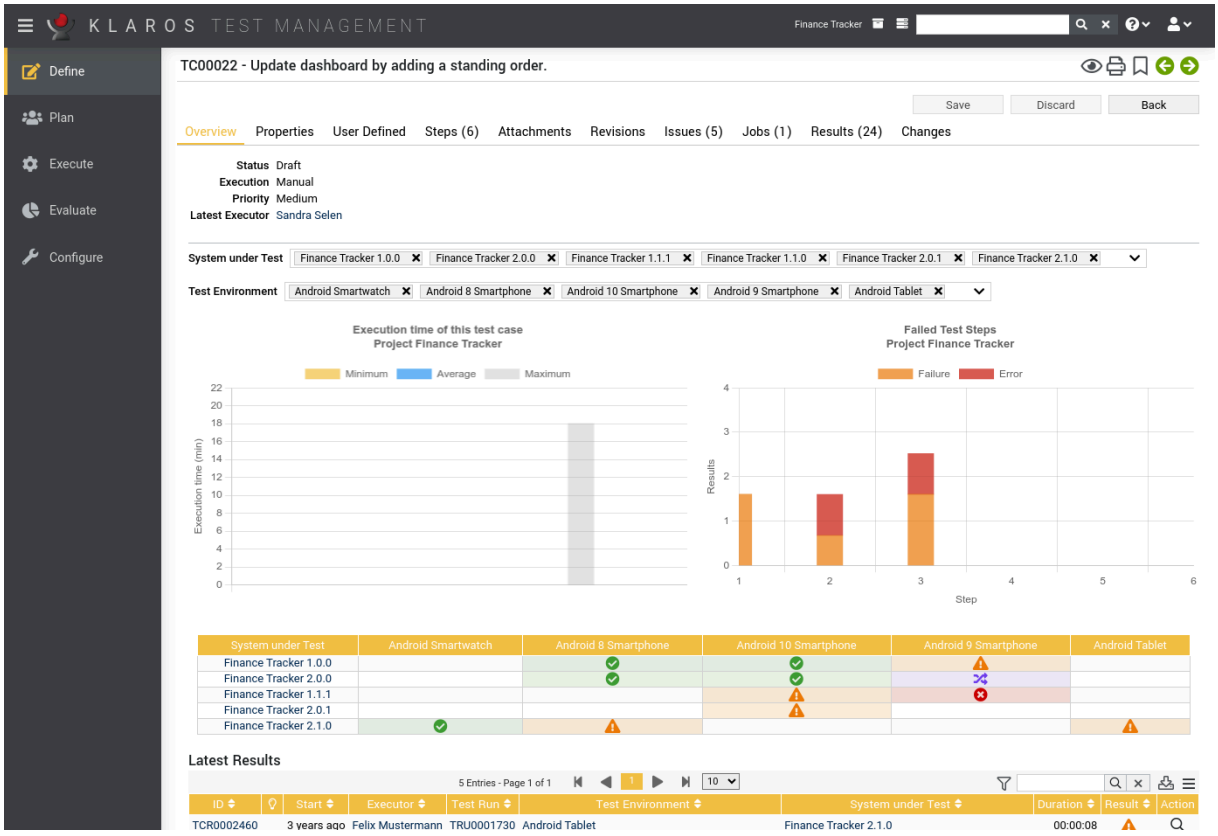


Figure 5.20. The “Overview” Tab

5.2.3.2.4. User Defined Properties

Only available in Klaros-Testmanagement Enterprise Edition

To customize projects to specific requirements, objects can be extended with user-defined fields.

These are created separately for each project in the *Project* section of the *User-Defined* view, see Figure 5.21.

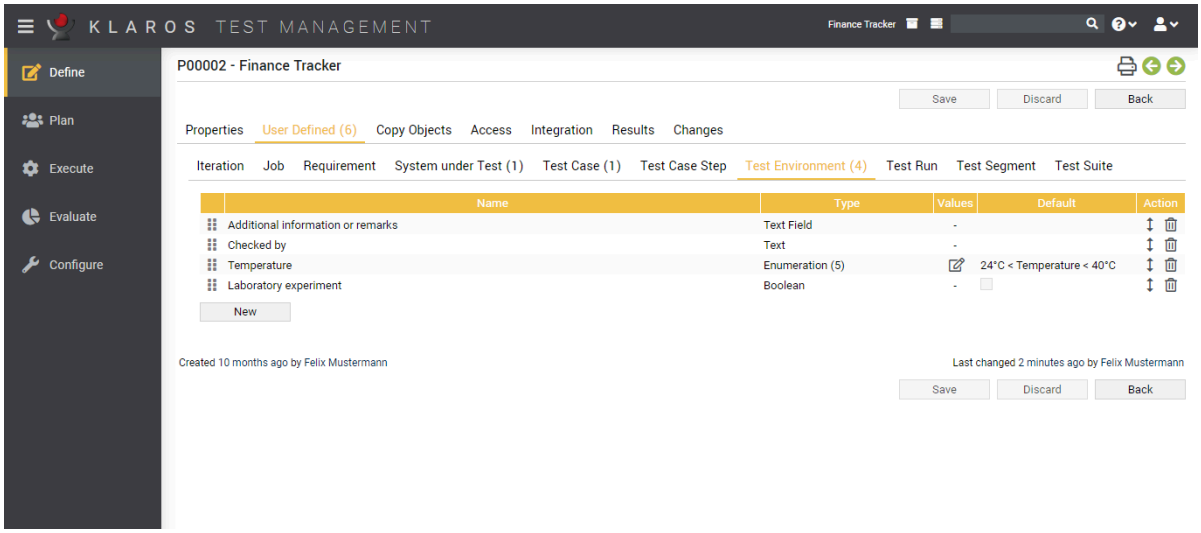


Figure 5.21. The “User Defined” Tab

Pressing the New button adds a new user defined property to the list. After adding a new property, the field type must be chosen.

User-defined fields can be created for the following objects, these are each managed in their own tab: **Test Case**, **Test Step**, **Test Segment**, **Test Suite**, **Test Environment**, **Test System**, **Test Run**, **Requirement**, **Iteration**.

The four possible property types are:

Text	The property will be a single-line text field.
Text Field	The property will be a multi-line text field.
Boolean	The property will be a check box.
Enumeration	The property will be a drop-down box to select a single entry from a predefined list of values.

The default value of the property can also be set here.

After the new property has been saved, the property type can no longer be changed. The name of the property and - if available - the list of enumeration values can be changed at any time.

Enumeration Values

The icon next to the enumeration name opens the dialog for editing the enumeration values, see Figure 5.22.

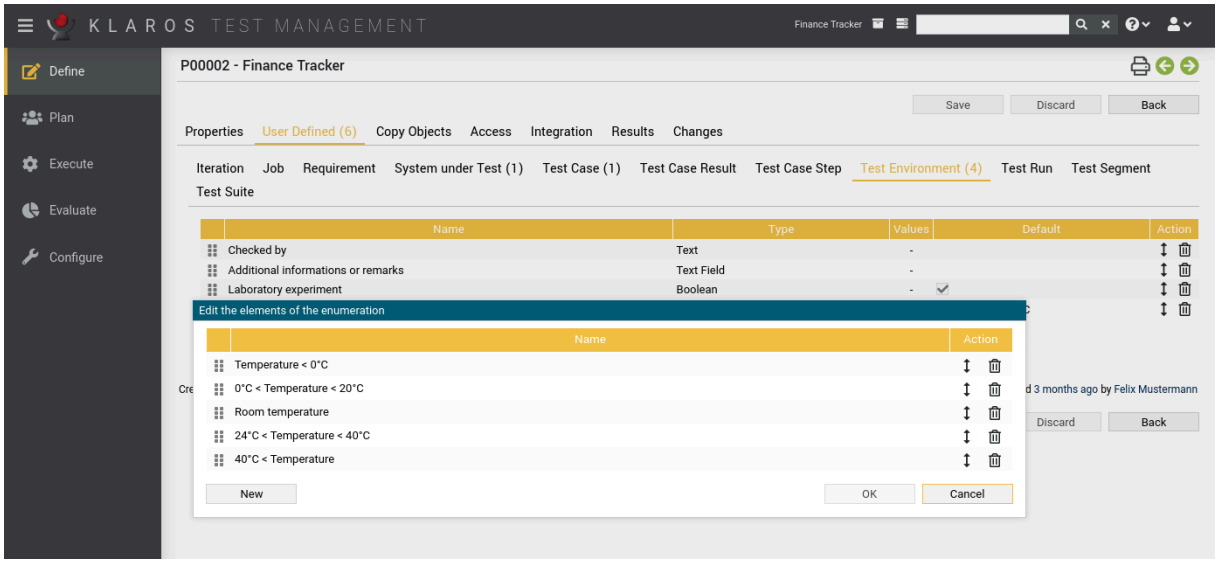


Figure 5.22. Editing an Enumeration Property

Once custom properties are created, they can be edited in the *Custom* view of the associated object (Figure 5.23).

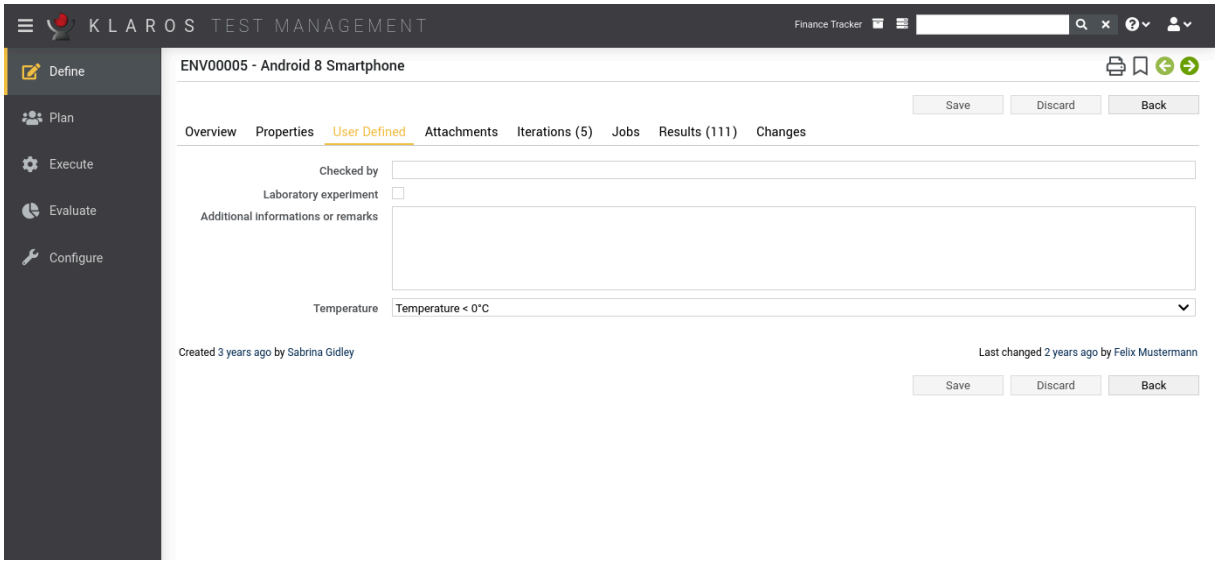


Figure 5.23. The “User Defined” Tab

For test runs, the values of the custom properties are captured in the respective execution dialog, see Figure 8.3.

5.2.3.2.5. Revisions

Requirements, *Test Cases*, *Test Segments* and *Test Suites* are revisionable objects.

A new revision of an object should be created when a major edit is taking place. For example, an older revision of a test case can only be executed when used with older versions of a system under test as newly added test steps are only applicable to never versions of the system under test.

Revisions of objects can be managed from the *Revisions* tab. This tab shows the revision history for the object, allows the user to change the revision comment associated with each revision and

also allows creating new revisions. The revision history table may be used to select the revision to display and edit.

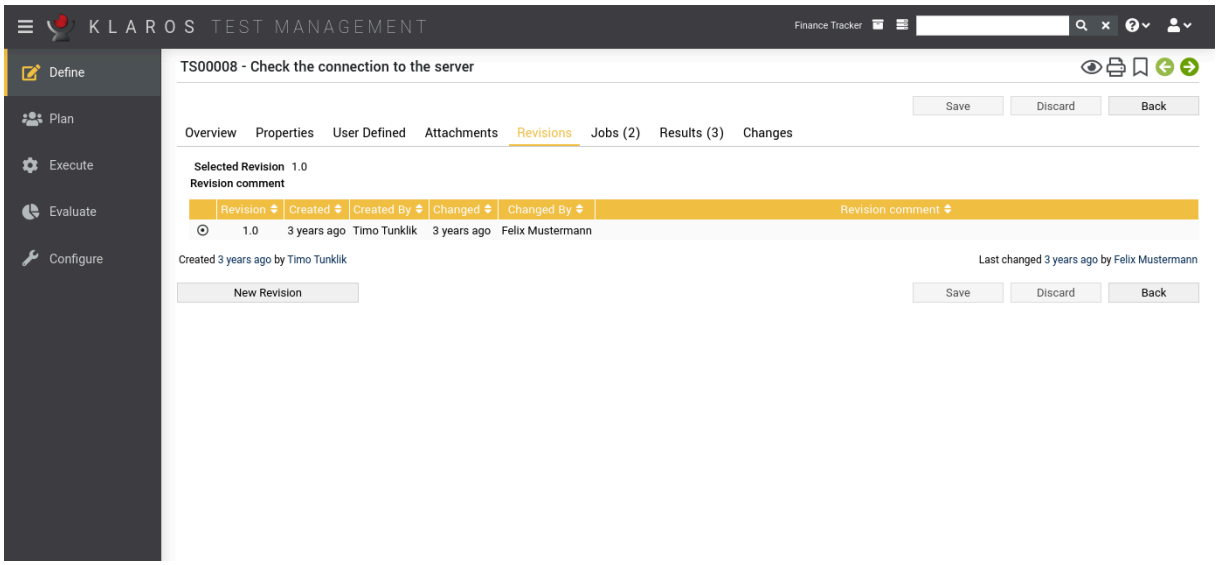
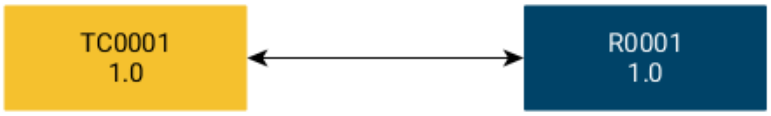


Figure 5.24. The “Revisions” Tab

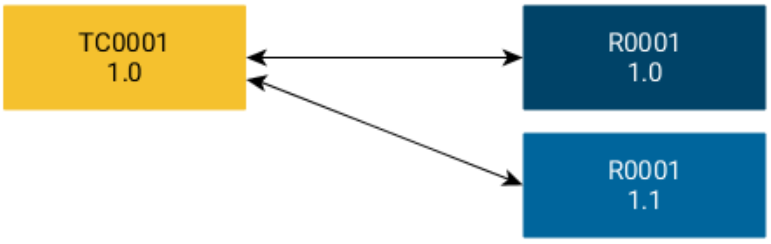
Relationships between object revisions

Many objects have a direct relationship to other objects. Objects like e.g. Test Cases or Requirements may exist in different revisions, so one revision of an object can be referenced by any revision of other objects. Every revision of an object may contain its own references to other objects, regardless of the relationships of the other revisions. In addition, every revision of an object can be related to one or more specific revisions of another object.

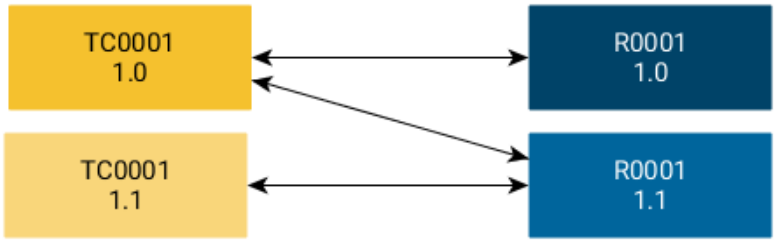
Example: Given is the test case TC0001 and the requirement R0001. Requirement R0001 is covered by the test case TC0001. Both are still in the initial revision 1.0.



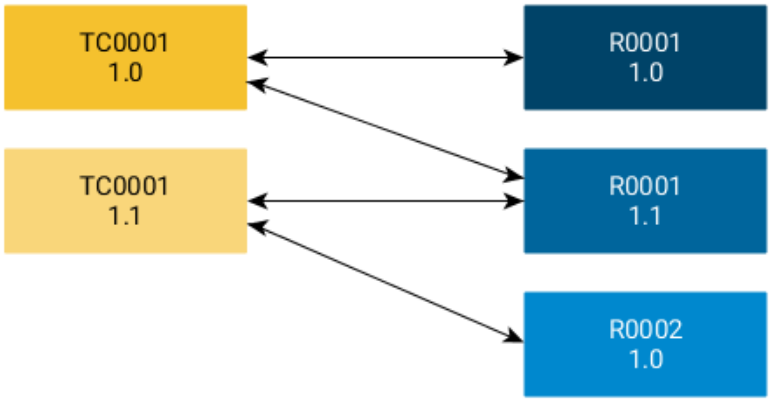
Now a new Revision 1.1 of Requirement R0001 is created which is also covered by Test Case TC0001. So TC0001 is now covering the Requirement R0001 in both revisions.



Next, a new revision of test case TC0001 is created, since revision 1.0 of requirement R0001 has become obsolete. Therefore, revision 1.1 of the test case will now reference the newest revision of the requirement.



After creating the new revision of test case TC0001, we discover that requirement R0002 is also covered by this test case revision. Therefore, we add further coverage between R0002 and the test case TC0001 in Revision 1.1.



This example shows, that although the Test Case and the Requirement are the same objects, their relationship can change over their revision history. References are defined per revision, and so can be different with each revision. This does not only apply to test cases and requirements, but also to other revisionable objects like test suites or test segments.

5.2.3.2.6. Attachments

For many objects file attachments can be uploaded and linked to this object. This can be done within the *Attachments* view on the corresponding detail page of the object or during the execution of a test case.

In the table on the attachment detail page, the *Name*, the *Size*, the *File Type*, the *Version* and the name of the creator. are displayed. The row contents can be edited directly in the table.

Click on the button `Upload attachment` to open a dialog. The button `Select` opens a dialog to select the attachment. With `OK` the selected file is uploaded to the application. If you want to upload more files, repeat the steps. Click `Save` to save the file permanently.

Clipboard Support

On browsers of the Chrome family (Chrome, Brave Microsoft Edge), an upload via **Ctrl+V** supported. Such, e.g. screenshots can be transferred without saving them in the file system.

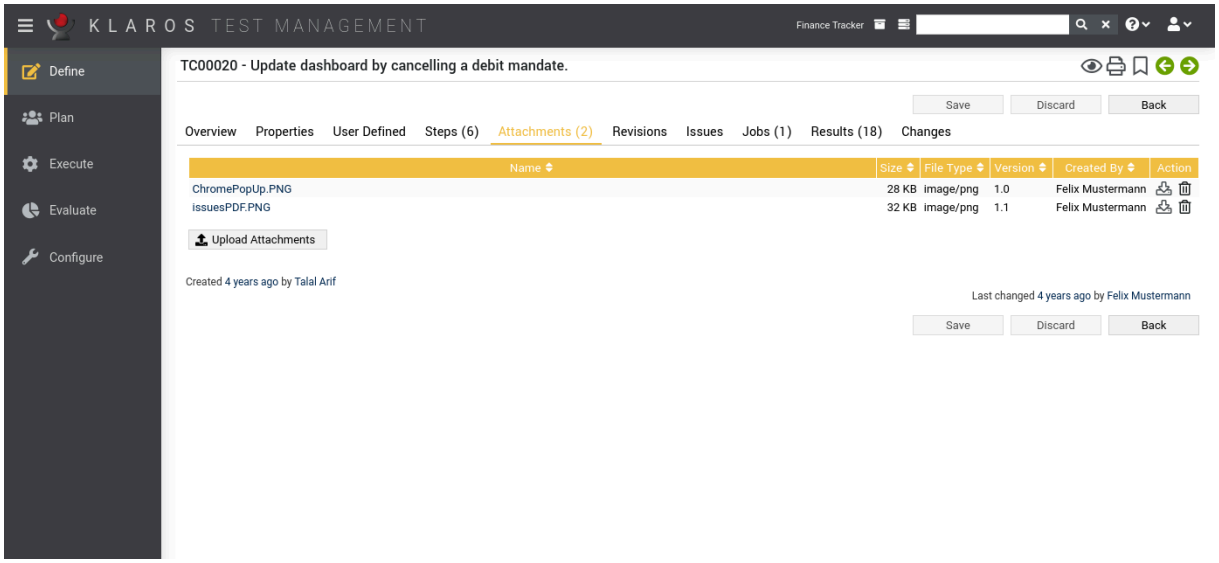


Figure 5.25. The “Upload Attachments” Page

In the action column saved attachments can be downloaded ( ) and deleted ( ).

File Size Limitation

The file size of an attachment can be limited to prevent the upload of excessively large files. This can be defined under *Configure* -> *System* -> *Other*.

If a file with the same name is uploaded multiple times, a new version of the attachment is created. This version contains the newly uploaded file.

5.2.3.2.7. Test Runs and Results

Only available in Klaros-Testmanagement Enterprise Edition

Some detail pages show a tab with test runs and test results which are related to the object being displayed, e.g. executed in the displayed test environment or executed by the displayed user. An example of a *Test Results* tab is shown in Figure 5.26.

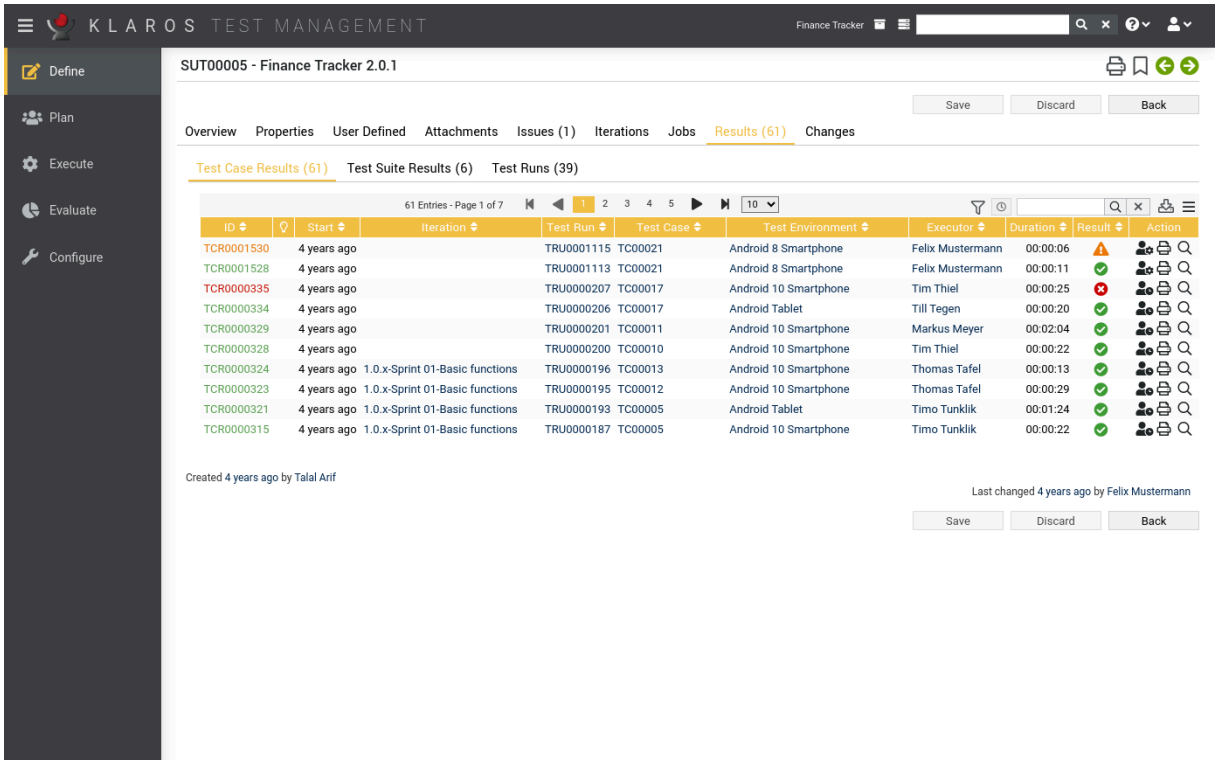


Figure 5.26. The “Results” Tab

In both of these tabs, the names of objects are links to the *Details Pages* of the objects. The 🔍 icon displays the *Details Page* of the corresponding test run or test result.

In the *Test Runs* tab (Figure 5.26), reports may also be generated, as described in Section 9.3.4, “The Test Run Report”.

5.2.3.2.8. Change History

enterprise edition Only available in Klaros-Testmanagement Enterprise Edition

This tab displays the entire change history of the object since its creation.

Figure 5.27 shows the *Changes* tab for a test case.

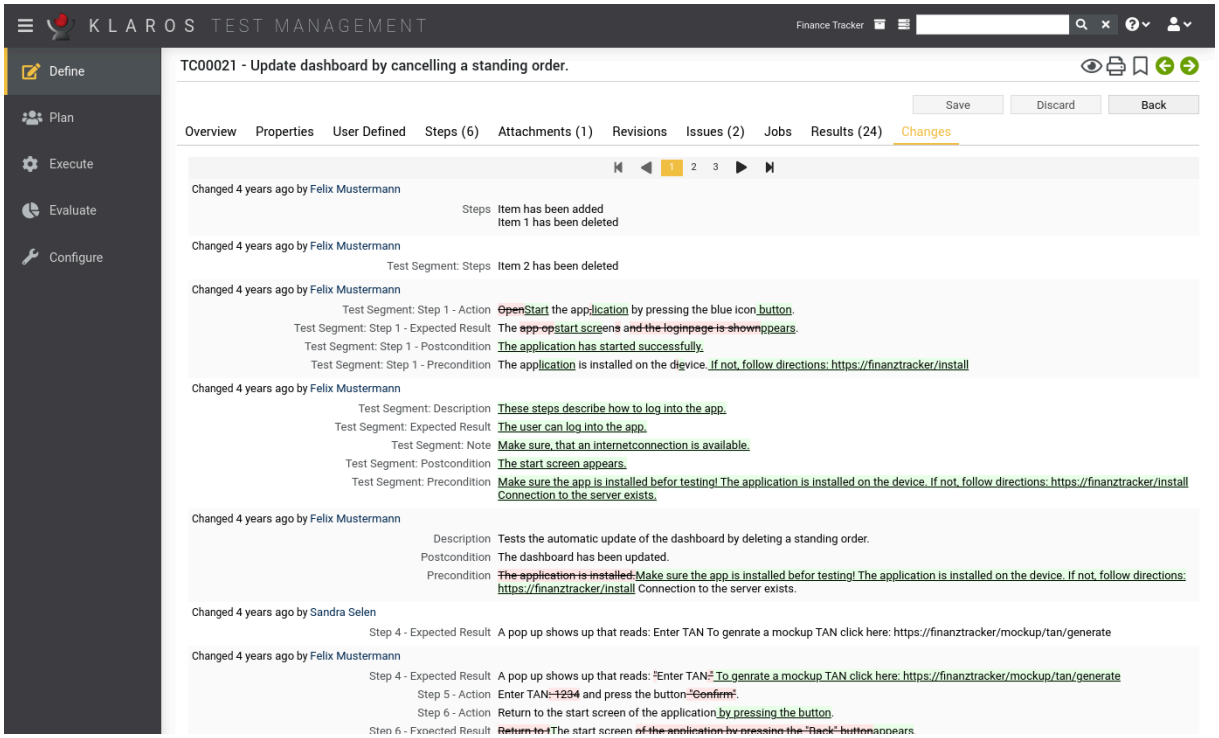


Figure 5.27. The “Changes” Tab

All changes are marked in color: green for newly added entries, red for deleted entries. Changes for non-text values are marked in the form *Old value -> New value*.

Furthermore, it is indicated which user has carried out a change and at what time it was made. If several changes were made at once, they are grouped together per save operation.

5.3. Main Functions

In this section, general cross-cutting functions of Klaros-Testmanagement are described in detail.

5.3.1. Resolving save conflicts

If an object is edited by two users at the same time, these changes may collide. The following measures are provided for this in the application.

When saving an object, an attempt is made to automatically resolve possible conflicts with concurrent changes as far as possible. Any changes made by other users that do not affect fields that the current user has changed are merged without additional user input.

If field contents have been changed that were also changed by another user in the meantime, a dialog is displayed. There the user can choose how the conflict should be resolved.

Figure 5.28 shows the *Conflict Resolution Dialog* for a conflicting test case change.

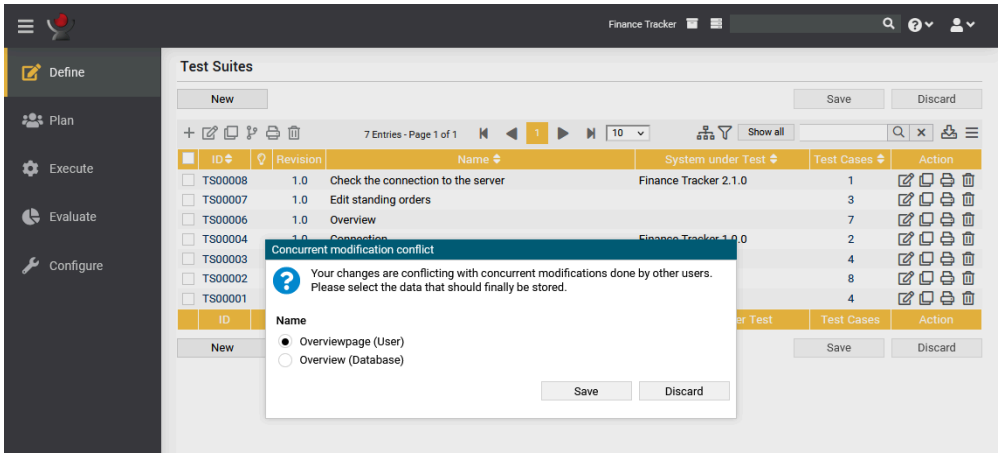


Figure 5.28. The “Conflict Resolution” Dialog

For each conflict, the user has the choice of saving the object with the value they entered or keep the value currently present in the database. For lists, e.g. the list of test steps in a test case, the user has the option to *Merge* the changes or to keep the version which is currently in the database. This is to prevent the inadvertent loss of data through the deletion of objects which were added by other users.

The *Merge* option builds a list containing all the elements of both lists (user input and database) in an approximation of the correct order.

Clicking the `Save` button persists the selected changes to the database, while clicking the `Discard` button discards them.

5.3.2. Deleting, Purging and Restoring Objects

5.3.2.1. Deleting Objects

Objects can be deactivated or deleted by clicking on the icon so that they are no longer included in reports and evaluations. If the object to be deleted is linked to other objects (e.g. a requirement linked to a test case) or is contained in other objects (e.g. a test case contained in an execution definition), a dialog is displayed (see Figure 5.29).

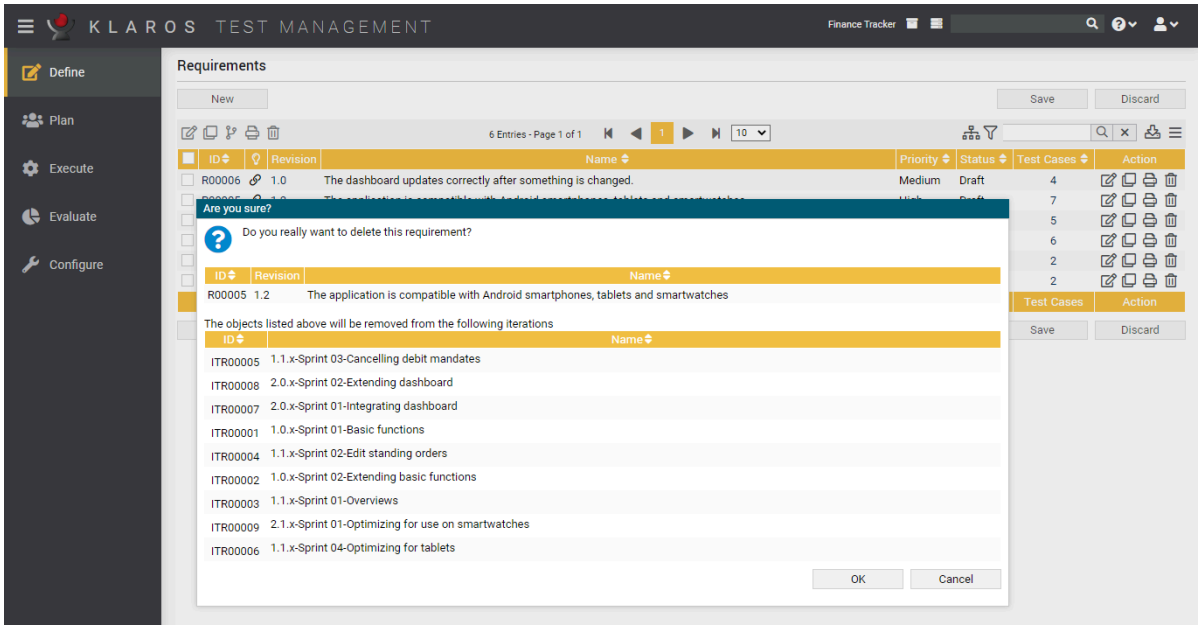


Figure 5.29. The “Delete Objects” Dialog

In the dialog window all dependencies, relations and occurrences of the object to be deleted are displayed. With OK the object is deleted, with Cancel the window is closed, and the object is not deleted.

5.3.2.2. Purging Objects

Objects can be irretrievably removed by clicking on the icon . For this, it is necessary that these objects have already been deleted or deactivated. By default deleted or deactivated objects are not displayed. By clicking on Show all you can show them. After clicking on the icon , a dialog appears (see Figure 5.30). Clicking on OK restores the object, clicking on Cancel closes the window and the object remains deactivated.

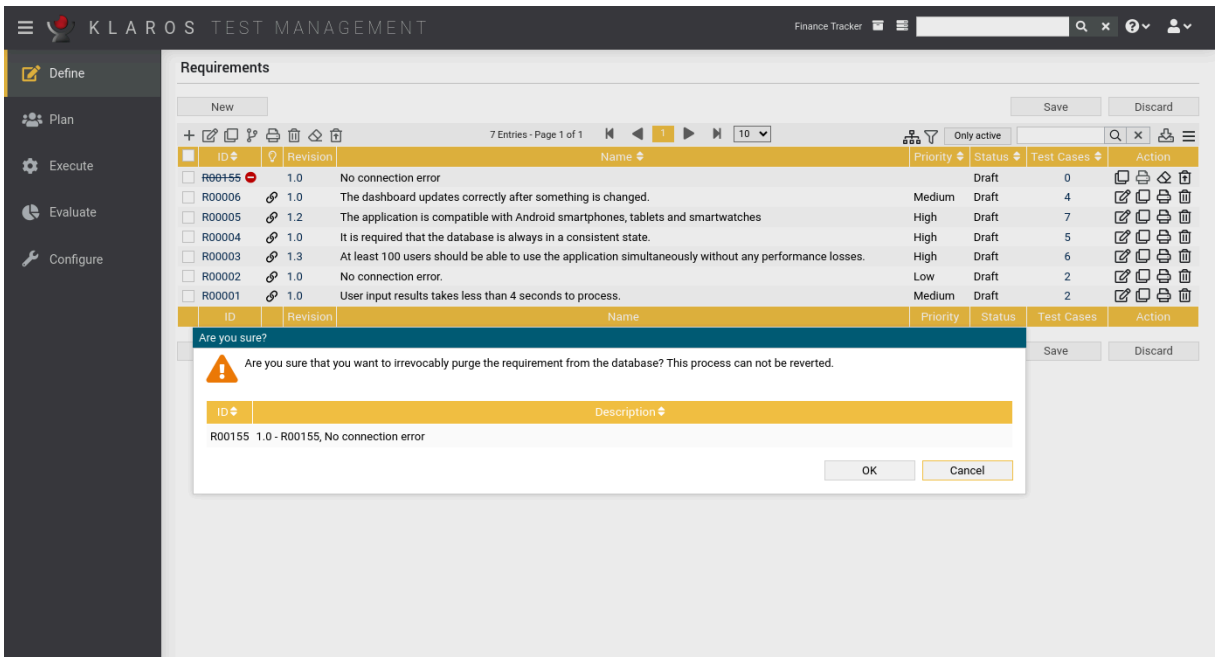


Figure 5.30. The “Purge Objects” Dialog

5.3.2.3. Restoring Objects

Deleted or deactivated objects can be restored by clicking on the icon . After clicking on the icon, a dialog appears (see Figure 5.31). Clicking on `OK` restores the object, clicking on `Cancel` closes the window and the object remains deactivated. Restored objects are included in reports and evaluations.

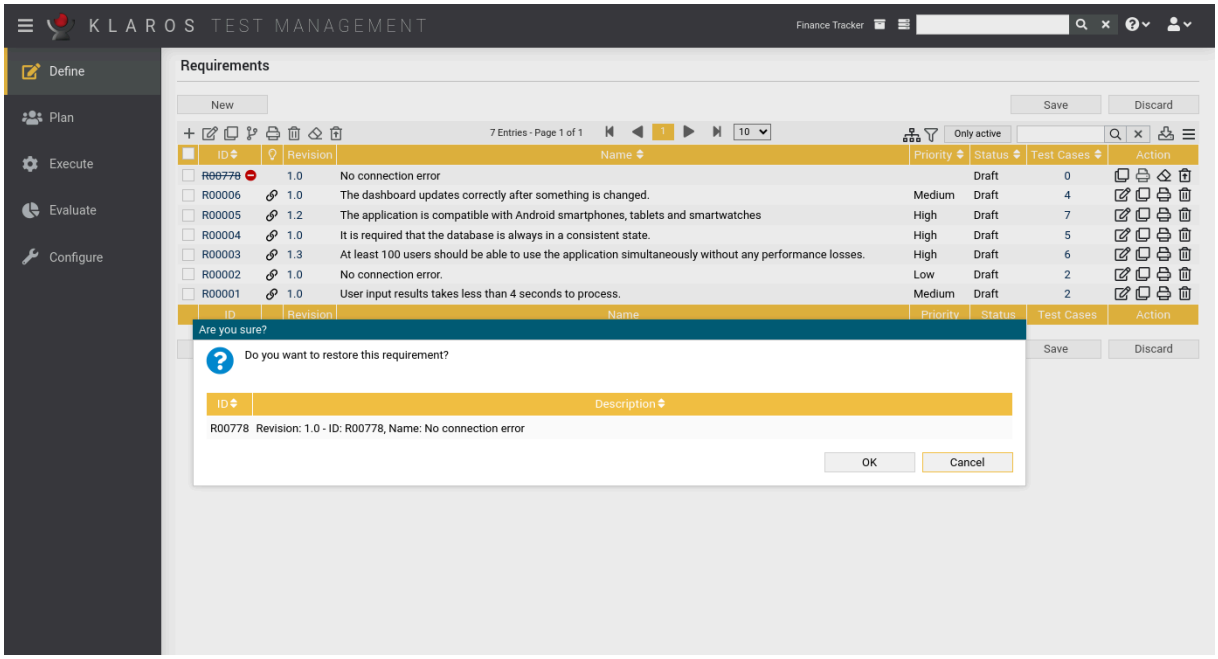


Figure 5.31. The “Restore Objects” Dialog

5.3.3. Referencing Object Properties

Properties and user defined properties (Section 5.2.3.2.4, “User Defined Properties”) can be referenced and resolved in the displayed text fields while executing jobs, test cases or test suites. For example, the following test case precondition references the system under test in which it is executed:

"Please ensure that the operating system version in /config/os matches %sut:buildnumber% before executing this test case."

When executing this test case, *%sut:buildnumber%* gets replaced by the value of the user defined property *buildnumber* of the system under test for which the test case is executed. So assuming the value for the property is `1.44_alpha3`, the text will be output as follows:

"Please ensure that the operating system version in /config/os matches 1.44_alpha3 before executing this test case."

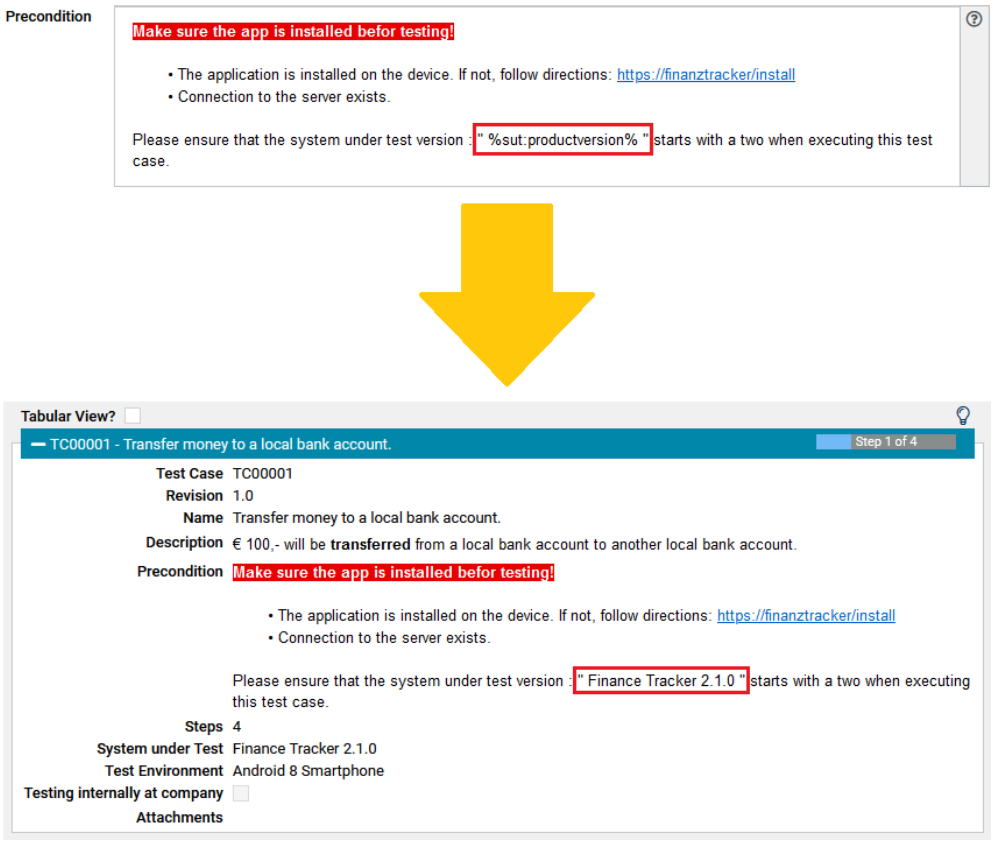


Figure 5.32. Referenced Object Property

Property substitution is supported both for the common pre-defined attributes of an object (e.g. the name or id) and the user defined properties. The name of the common attributes can be derived from the corresponding bean methods defined in the de.verit.klaros.core.model .

Properties of the following objects are currently supported:

Iteration	`itr`
Job	`job`
System under Test	`sut`
Testcase	`tc`
Test Environment	`env`
Test Run	`tr`
Test Segment	`seg`
Test Suite	`ts`

Assume that the following example text is used in the test case precondition:

"Please make sure that your system under test is using version %sut:productversion%."

The attribute name `productversion` is found in the class documentation for the `SUTImplementation` class in de.verit.klaros.core.model.KlarosSUTImplementation. Given that the ver-

sion defined there has a value of `1.0.1`, the following output will be rendered when displaying the precondition:

"Please make sure that your system under test is using version 1.0.1."

Property References are Case Insensitive!

The left-hand side of a reference is case-insensitive. `%sut:productversion%`, `%SUT:productversion%` and `%Sut:productversion%` will yield the same result.

5.3.4. Referencing Attachments

Similar to referencing user defined properties in test case and test case steps (Section 5.3.3, "Referencing Object Properties"), values from binary attachments can be referenced as well. Currently, CSV and XLS files are supported.

Binary attachments of the following objects are currently supported:

Iteration	`itr-att`
Job	`job-att`
System under Test	`sut-att`
Test Case	`tc-att`
Test Environment	`env-att`
Test Run	`tr-att`
Test Segment	`seg-att`
Test Suite	`ts-att`

In addition to the above, the global wildcard `att` can be used. The position of the attachment is thus not specified. In this case, all objects involved in the test execution are searched for the attachment.

A placeholder can be defined in the form *%[object type]:[file name]:[column]:[row]%* using the following parameters:

Placeholder Parameters

[object type]	The type of the object that is holding the binary attachment.
[file name]	The name of the binary attachment.
[column]	The column number of the value of the binary attachment.
[row]	The row number of the value of the binary attachment.

Referencing Excel Files

When referencing Excel files, cells can be addressed using Excel-Coordinates (e.g. C1 or AB24). This shortens the placeholder to *%[object type]:[file name]:[coordinate]%*.

The file name, column and row parameters can also be defined using user defined properties. So *%itr-att:customers.csv:%job:Customer%:%job:CustomerNr.%%* references the attachment `customers.csv` of the current iteration, using the value in the column defined in the attribute *Customer* of the current job and the row defined in the attribute *CustomerNr*.

Attachment References are Case Insensitive!

The left-hand side of an attachment reference is case-insensitive. `%itr-att:customers.csv:A1%`, `%ITR-att:customers.csv:A1%` and `%ITR-Att:customers.csv:A1%` will yield the same result.

Chapter 6. Define

In the *Define* section, Projects, Iterations, Requirements, Test Environments, Systems under Test, Test Segments, Test Cases and Test Suites can be created and edited. There is a menu entry on the left-hand side for each of these object types.

6.1. Projects

A *project* collects all objects needed and created in a test project, such as test cases, test runs and test results. This chapter shows how to create and edit *projects*.

6.1.1. Overview Page

The overview page displays all existing projects in the selected project in a table. New projects are created here.

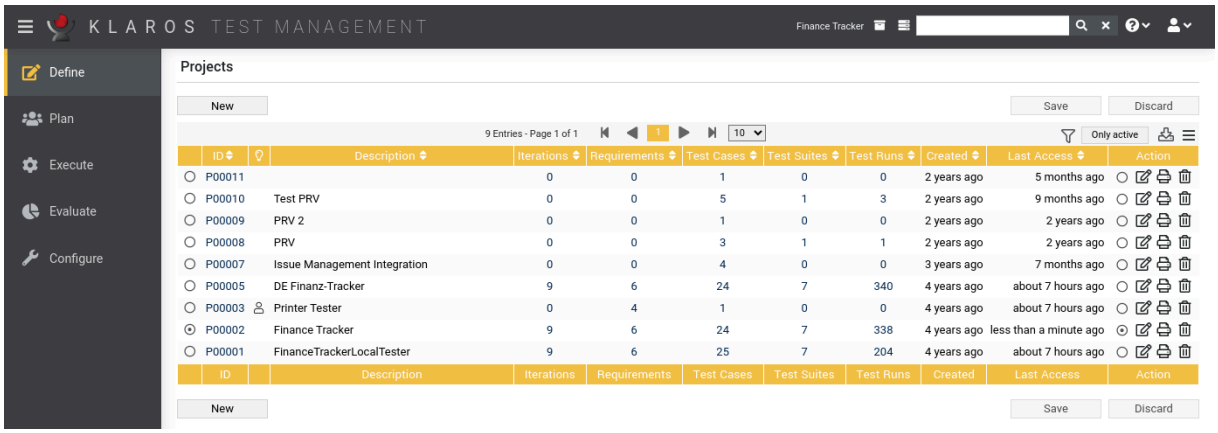


Figure 6.1. The “Projects” Page

The table shows the following values:

ID — Assigned automatically.

Additional Information — A tooltip appears when the cursor is placed over the icon shown here.

Description — The description of the project.

Iterations — The number of iterations in this project.

Requirements — The number of requirements in this project.

Test Cases — The number of test cases in this project.

Test Suites — The number of test suites in this project.

Test Runs — The number of test runs in this project.

Created — The date on which the project was created.

Actions	The executable actions.

The *Description* entry can be edited directly in the table rows with one click.

6.1.1.1. Create a new Project

By clicking on the button New a a new empty table row will appear. Now, the *Description* can be defined.

With Save the new project is created and saved. The ID of the project (P00001) is automatically assigned by Klaros-Testmanagement. Click on the ID *P00001* to get to the detail page of the project.

With New several table rows can be created and edited at the same time. Only when clicking on Save the data is stored in the database.

With Discard all changes are undone.

Red IDs

All rows with red IDs have been changed and are not yet saved!

6.1.1.2. Actions

The following actions can be performed in the action column:

- Activate
- Edit
- Open print view
- Delete

If a project has been deleted, it is initially marked with a deletion marker and is only visible to administrators. For deleted projects, the following actions are available instead of *Activate* and *Delete*:

- Restore (only Administrator)
- Irrecoverably remove the project from the database

6.1.1.3. Table Operations

The following operations can be performed in the line above the table on the right:

- Filter / Sort
- Show all / Only active (only Administrator)
- Export
- Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

6.1.1.4. Quick Selection

To quickly switch between projects, a quick selection is also available in the page header (see Figure 6.2, “Quick Selection of Projects”).

Figure 6.2. Quick Selection of Projects

6.1.2. Details Page

Each project has its own detail page with several additional tabs. Clicking on the ID of the respective project or on the icon on the right in the action column takes you to the tab that was selected last. When called for the first time, this is the *Overview* tab.

The following tabs are available: *Properties*, *User Defined*, *Copy Objects*, *Access*, *Integration*, *Results* and *Changes*.

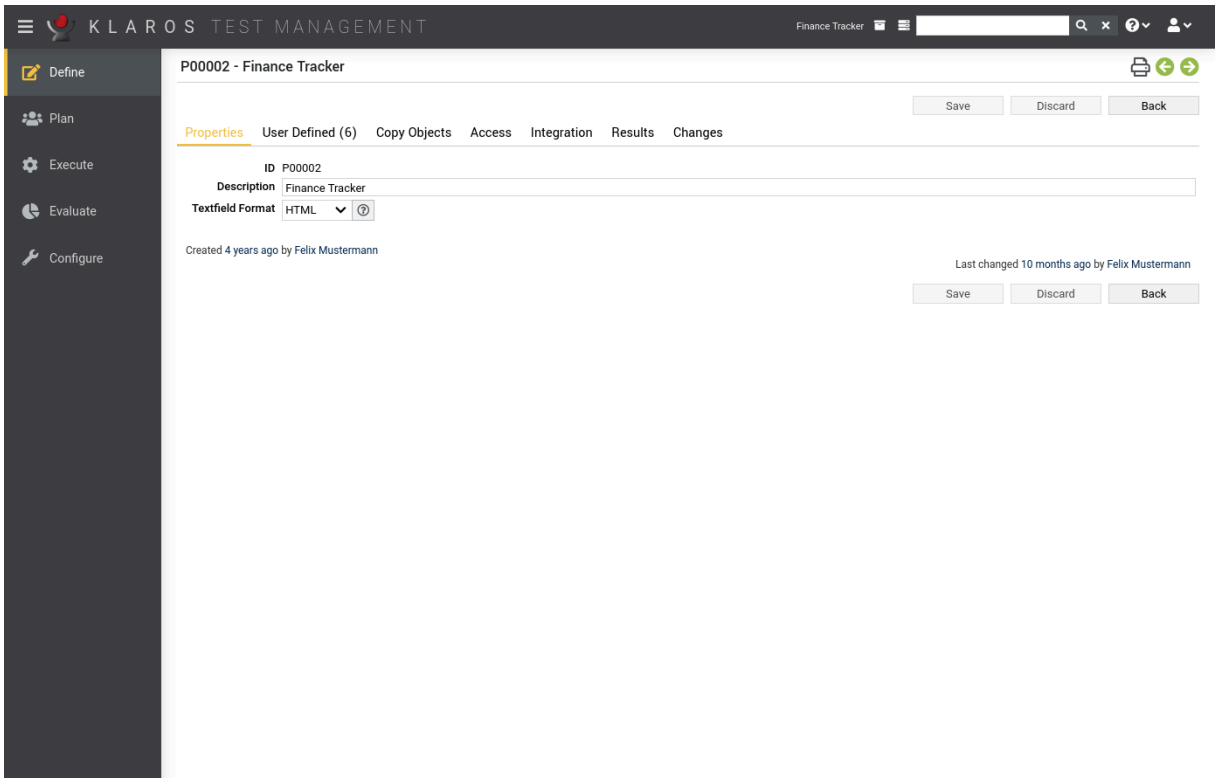


Figure 6.3. The “Properties” Tab - Projects

6.1.2.1. Actions

Auf den Detailseiten lassen sich jeweils folgende Aktionen vornehmen:

Open print view — A print-ready view of the project can be created here. With a click on the icon this opens in a new browser tab.

Print views are described in detail in Section 5.2.3.2.1, “Print Pages”.

Browse — Use the green arrows at the very top right to switch between the projects present on the previous page.

6.1.2.2. Properties

This tab (Figure 6.3) allows the user to view or change the following attributes of the project:

Description — The description of the project.

Textfield Format — Multiline text fields can contain HTML content or plain text. This is set here for the entire project.

6.1.2.3. User-Defined

Only available in Klaros-Testmanagement Enterprise Edition

A project can be adapted to specific requirements by defining additional fields for objects such as test cases. For further information please refer to Section 5.2.3.2.4, “User Defined Properties”.

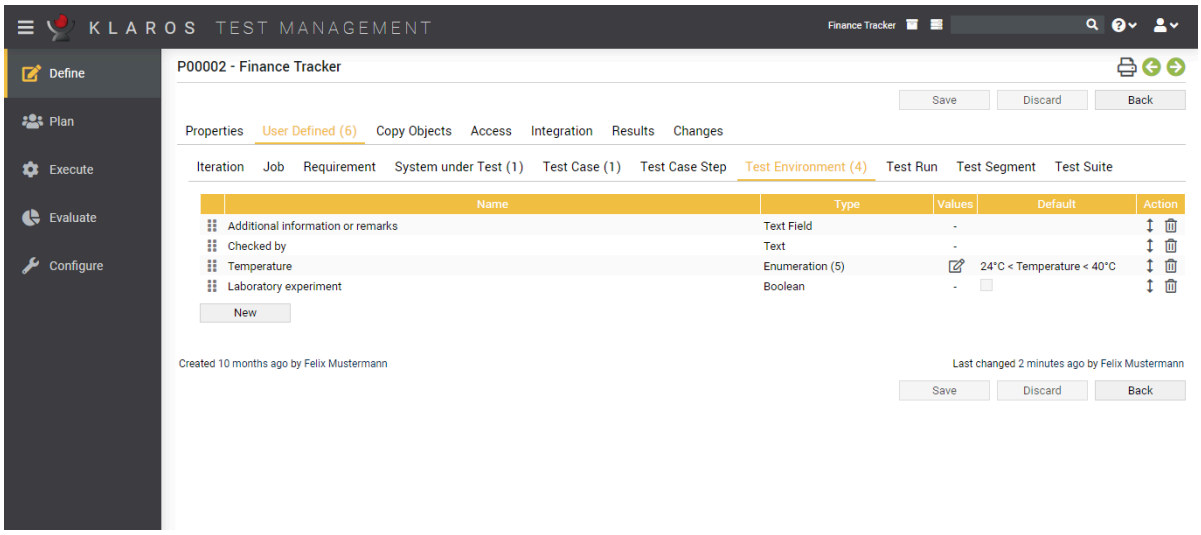


Figure 6.4. The “User Defined” Tab

6.1.2.4. Copy Objects

Only available in Klaros-Testmanagement Enterprise Edition

Objects can be copied or synchronized between projects. The *Copy objects* tab displays a drop-down list of all source projects that the user can access to and that contain at least one copyable object. The *Transfer user defined fields* checkbox determines whether user defined properties should be copied as well.

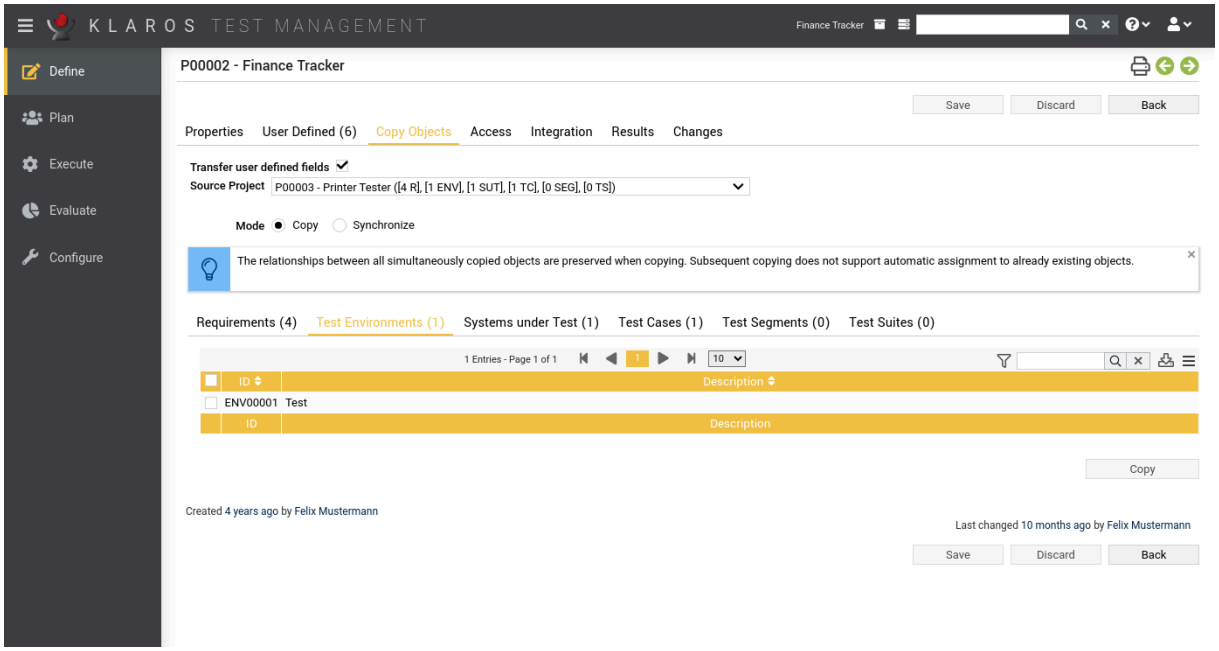


Figure 6.5. The “Copy Objects” tab

Once a source project has been selected, the lists of objects that can be copied (Requirements, Test Environments, test systems, test cases, test segments and test suites) appear, as well as a radio button to select the mode (Copy or Synchronize).

Two modes are available.

- Copy
- Synchronize

Copied objects have no reference to their source object, so subsequent changes to the source object are not propagated to the copies.

Synchronized objects have a reference to their source object and can *not* be edited. Changes to the source objects can be transferred to the copies at any time.

What are synchronized objects needed for?

Synchronized objects allow you to manage frequently used test cases in a reference project and reuse them in other projects.

Already synchronized objects are hidden and can be displayed by clicking the Show all button. If there are changes to the source object for a synchronized object, the icon is displayed in the *Additional notes* column. Clicking on it opens a dialog with which it can be synchronized.

Synchronization Mode

Currently only test cases and test suites are supported in synchronization mode.

After one or more objects have been selected, clicking the Copy (Synchronize) button opens a dialog in which all objects are shown in a list. Clicking the Ok starts the copy process (see Figure 6.6, “The “Copy Objects” Dialog ”).

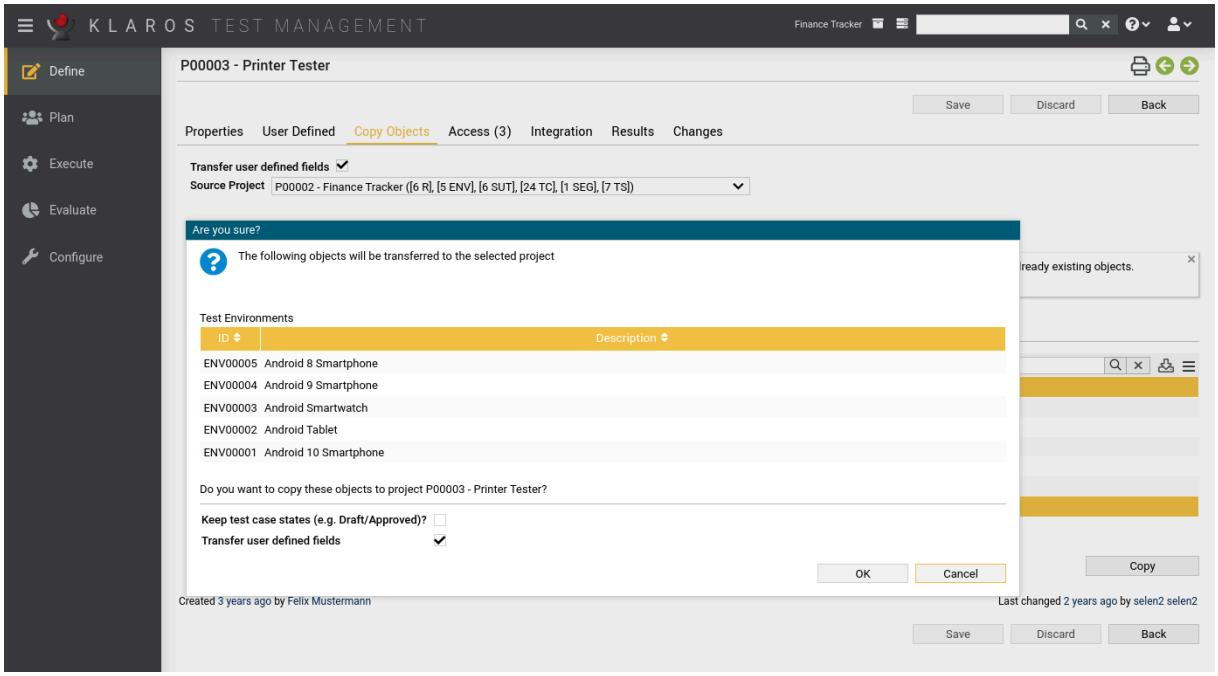


Figure 6.6. The “Copy Objects” Dialog

The *Keep test case status* checkbox determines whether the defined test case statuses should be copied as well or whether the *Draft* status should be set for all test cases instead.

Note

When a test suite is copied, all contained test cases are automatically copied as well.

6.1.2.5. Access

A project can be secured against outside access by enabling the *Disable anonymous project access* option.

When this option is enabled, a user must always be logged in to Klaros-Testmanagement to view pages referenced via bookmarks. For more information on bookmarking, see Section 5.2.3.2.2, “Bookmarks”.

If this option is disabled, a page referenced via bookmarks is accessible to everyone even without logging in. As soon as the user then tries to switch to another page, he is redirected to the login screen. If a user is not yet logged in, this is indicated by a corresponding icon in the Section 5.2.2.4, “The User Menu” at the top right.

6.1.2.5.1. Assigning Project Specific Roles

Only available in Klaros-Testmanagement Enterprise Edition

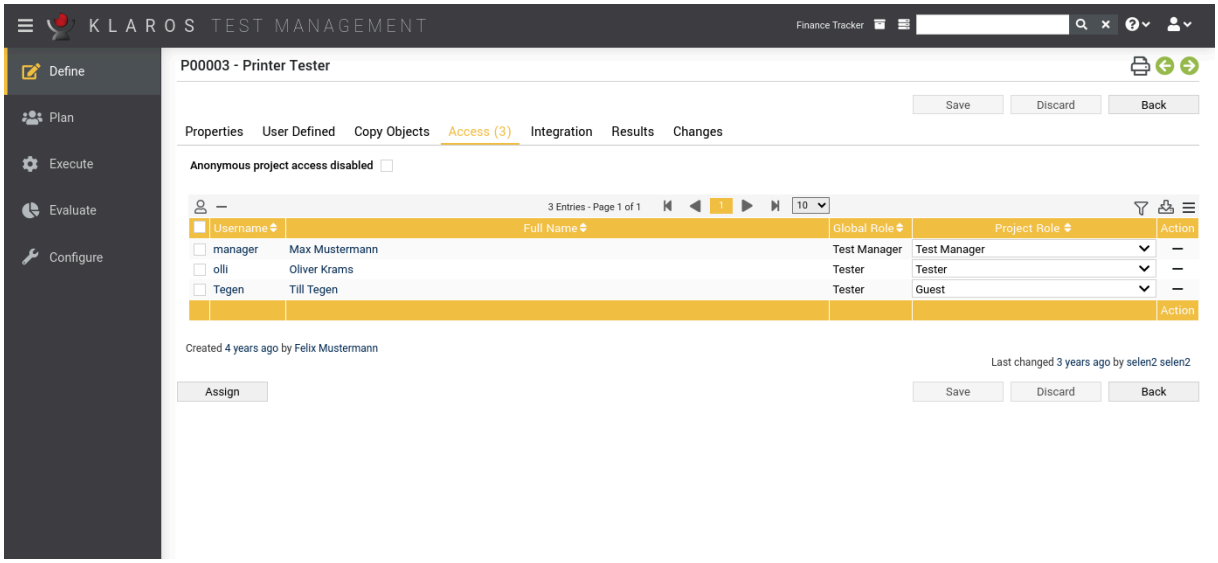


Figure 6.7. The “Access” Tab

Klaros-Testmanagement Enterprise Edition features a global role based access system (see Section 10.3, “Users”). In the *Access* tab, administrators and test managers can change the global role of a user to an individual project role. For example, a user with the global role tester can be assigned to one project as a tester and to another project as a test manager.

Initially, every project is accessible by every user. If at least one test manager has been assigned to a project, access to this project is restricted for all other users not explicitly assigned.

Pressing the Assign button opens up a dialog (see Figure 6.8, “The “Assign Project Role” Dialog ”) where new users can be added to the project.

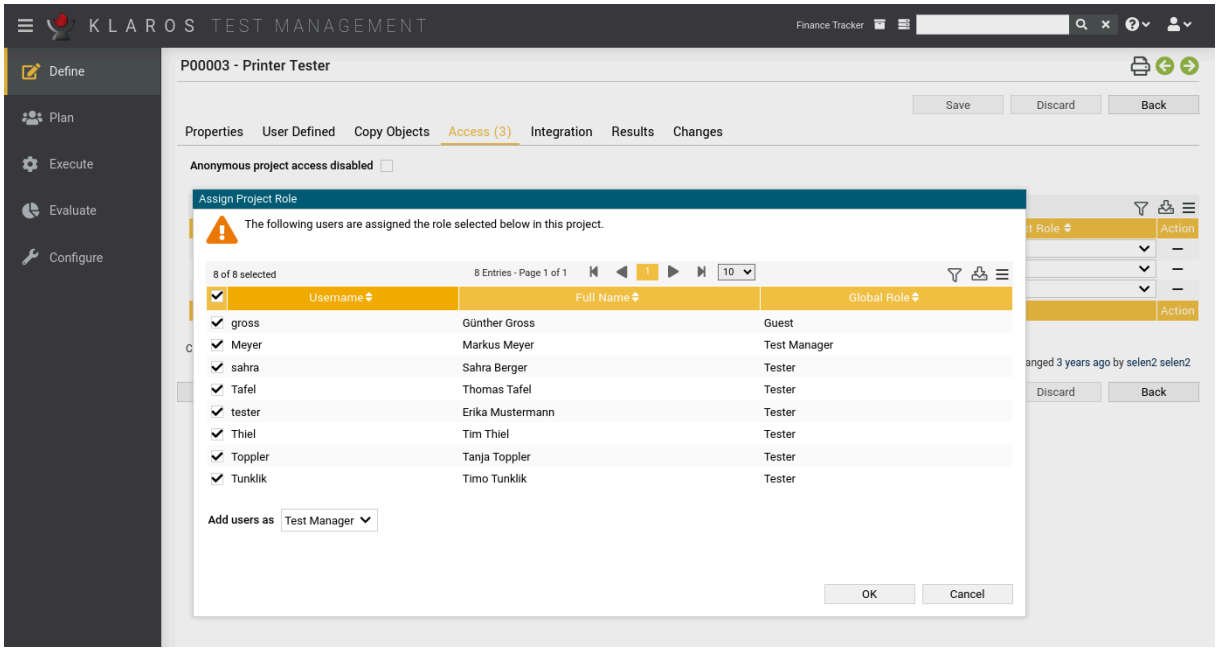


Figure 6.8. The “Assign Project Role” Dialog

Every project needs at least one test manager!

A project needs at least one assigned test manager. When trying to save a project which has no test manager assigned, a warning message will be displayed in the log panel (see Section 5.2.2.2, “The Log Panel”).

The following collection actions can be performed here:

- Change user role in project
- Delete

6.1.2.6. Integration

In this view, external systems such as issue management and requirements management can be linked with Klaros-Testmanagement.

6.1.2.6.1. Issue Management

The issue management systems that are in use in the project can be edited here.

Creating an Issue-Management-System

Before an issue management system is available on this page, it must first be created by an administrator.

For instructions on the creation of issue management systems, see Section 10.5.1, “Issue Management”.

The tab shows two tables: The left table contains the issue management systems that are available, while the right table shows the issue management systems that are already assigned to the project. Pressing the › icon adds the corresponding issue management system to the project. To add all issue management systems press the » icon. The ‹ and « are used in the same way to unassign issue management systems. It is also possible to drag and drop an issue management system to a list.

The icons in the center can be used to perform the following actions:

- › Assign the selected issue management system.
- » Assign all issue management systems.
- « Unassign all issue management systems.
- ‹ Unassign the selected issue management system.

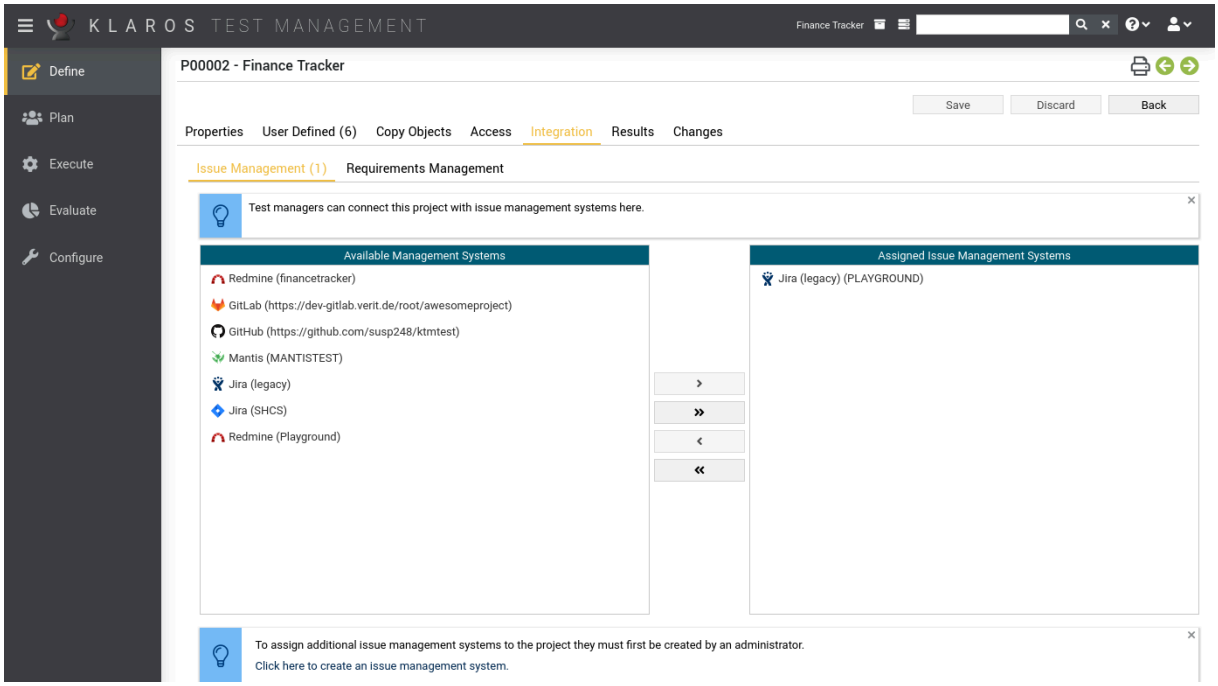


Figure 6.9. The “Integration/Issue Management” Tab

6.1.2.6.2. Requirements Management

enterprise edition Only available in Klaros-Testmanagement Enterprise Edition

See Section 10.5.2, “Requirements Management” for more information about how to connect remote requirement management systems to Klaros-Testmanagement.

Connecting a Requirements Management System

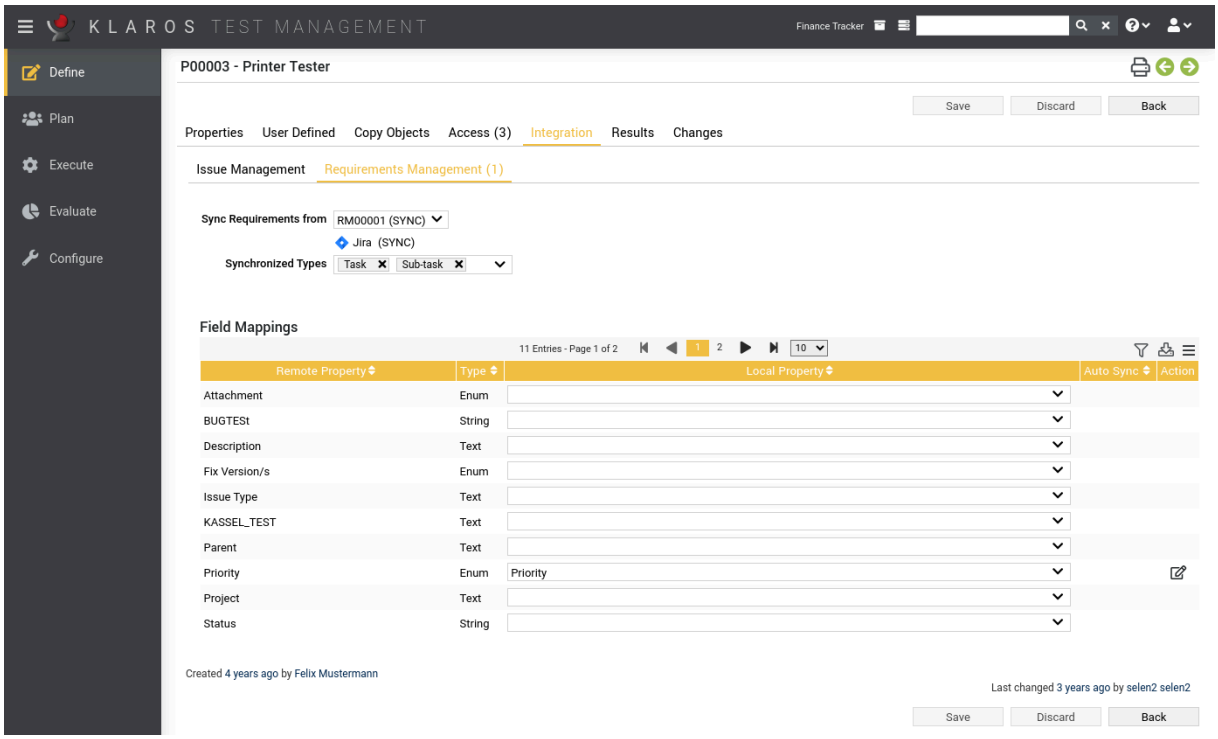


Figure 6.10. The “Integration/Requirements Management” Tab

A project can be connected to a requirement management system via the *Requirements Management* tab on the project detail page. This tab will show a dropdown list of all available requirement management systems.

Important

In order to connect an RMS to a project, the RMS has to be configured in the *Configure* section first (see Section 10.5.2, “Requirements Management”).

After pressing the Save button, a background synchronization will start that will load all requirements from the configured RMS and stores them locally.

Note

Synchronizing requirements can take up a huge amount of time depending on the number of requirements on the remote side. A built-in scheduler will synchronize all requirements of all connected requirement management systems for all enabled projects in specified intervals.

Warning

When enabling a remote RMS any local requirements that have already been created will be disabled. Switching back to local requirement management will enable those requirements again.

Linking Types

After selecting a requirement management system from the dropdown list, a list of issue types appears. Per default all types are selected, so a background synchronization will synchronize requirements of any type.

Figure 6.11. The Synchronized Types of the Requirements Management System

Linking Fields

Administrators and project managers can link individual fields between the remote requirement management system and Klaros-Testmanagement. When using Jira, as per default, the *Description*, *Priority* and *Summary* fields from the remote RMS (if they are available) are linked to the *Description*, *Priority* and *Name* fields in Klaros-Testmanagement. In addition to that, the Jira *Status* field is automatically mapped to a dedicated *Status* field that is only displayed if RMS synchronization is enabled.

Fields can be one out of three types: *String*, *Enum* and *Boolean*.

Note

An enum field in Klaros-Testmanagement can only be linked to an enum field of the remote RMS, but an enum field of the RMS can be linked to either a string or boolean field in Klaros-Testmanagement.

Linking Enum Values

When linking enum fields, it is necessary to indicate which enum values should be linked together. Otherwise, all values of the RMS will be mapped to the default enum value of the Klaros-Testmanagement field.

Clicking the icon of an enum linking opens up a dialog where individual values can be linked.

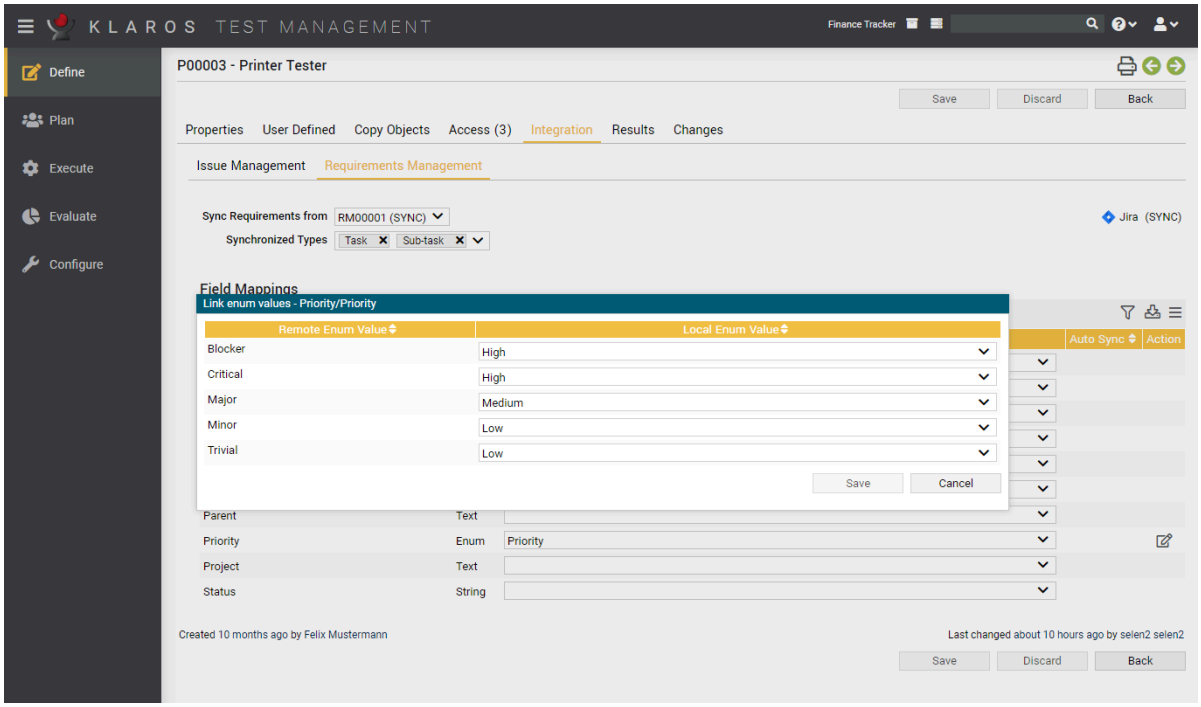


Figure 6.12. Linking Enumeration Values

Note

Multiple enum values of a remote enum field can be linked to the same enum value of a field in Klaros-Testmanagement.

Synchronizing Enum Values

When linking enum fields, it is often tedious to keep the list of selectable values in sync. For this, an automated synchronization mechanism for enumeration values is available using the *Auto Sync* column.

Using AutoSync requires that the remote field is an enum field. In addition, the local field must be set to a user defined requirement enum property. Once this is satisfied, the *Auto Sync* field will be displayed with a check mark in the list.

Note

It is not necessary to specify all the enum field values. Once the *Auto Sync* flag is set, the remote system is queried for available values and values for the local field are automatically changed accordingly.

Switching the Assigned Requirements Management System

The requirement management system (RMS) of a project can be changed any time. All currently existing requirements of the project will be disabled, as long as another RMS is used.

6.1.2.7. Results

The results tab is further divided into a *Test Runs*, *Test Case Results* and a *Test Suite Results* tab, showing the test results related to this project as described in Section 5.2.3.2.7, “Test Runs and Results”.

6.1.2.8. Changes

The tab *Changes* shows the change history of this project.

For a detailed description of the *Changes* view, see Section 5.2.3.2.8, “Change History”.

6.2. Iterations

Only available in Klaros-Testmanagement Enterprise Edition

A *Iteration* represents a test cycle in a project and allows to synchronize the test process with an agile development process (e.g. Scrum). This chapter shows how to create and edit *iterations*.

6.2.1. Overview Page

The overview page displays all existing iterations in the selected project in a table. New iterations are created here.

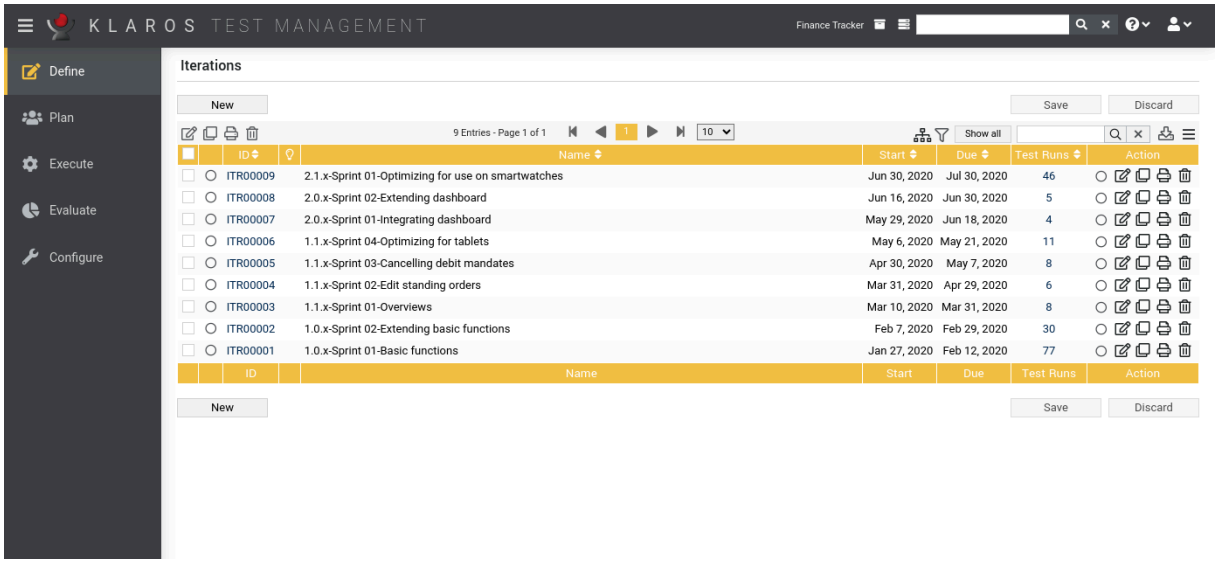


Figure 6.13. The “Iterations” Page

The table shows the following values:

ID — Assigned automatically.

Additional Information — A tooltip appears when the cursor is placed over the icon shown here.

Name — The editable name of the iteration.

Start Date	The date on which this iteration starts.
Due Date	The date on which this iteration is due to be finished.
Test Runs	The number of test runs.
Actions	The executable actions.

Name, *Start Date* and *Due Date* can be edited directly in the table rows.

6.2.1.1. Creating a new Iteration

By clicking on the button New a a new empty table row will appear. Now, *Name*, *Start Date* and *Due Date* can be defined.

With Save the new iteration is created and saved. The ID of the iteration (ITR00001) is automatically assigned by Klaros-Testmanagement. Click on the ID *ITR00001* to get to the detail page of the iteration.

With New several table rows can be created and edited at the same time. Only when clicking on Save the data is stored in the database.

Red IDs

All rows with red IDs have been changed and are not yet saved!

With Discard all changes are undone.

6.2.1.2. Actions

The following actions can be performed in the action column:

- Activate
- Edit
- Duplicate
- Open print view
- Delete

If an iteration has been deleted, it is initially marked with a deletion marker and is only visible to administrators. For deleted iterations, the following actions are available instead of *Delete*:

- Restore (only Administrator)
- Irrecoverably remove the iteration from the database

6.2.1.3. Bulk Actions

On the *Iterations* page, one or more iterations can be selected for bulk actions. Bulk actions are described in Section 5.2.3.1.5, “Bulk Actions”.

The following bulk actions are supported for iterations:

- Edit
- Duplicate
- Open print view
- Delete
- Restore (only Administrator)
- Irrecoverably remove the iteration from the database (only Administrator).
- Assign to a category (appears only after a category has been created).

Bulk actions are described in detail in Section 5.2.3.1.5, “Bulk Actions”

6.2.1.4. Table Operations

The following operations can be performed in the line above the table on the right:

- Categorize
- Filter / Sort

  Show all / Only active
- Search
- Export
- Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

6.2.1.5. Quick Selection

To quickly switch between iterations, a quick selection is also available in the page header (see Figure 6.14, “Quick Selection of Projects and Iterations”).

Figure 6.14. Quick Selection of Projects and Iterations

When an iteration is active, only test objects relevant to the test cycle represented by the iteration are visible. If a test run is executed while an iteration is active it will be linked to the iteration automatically. This helps to separate the test results of a given test cycle from other activity.

6.2.2. Details Page

Each iteration has its own detail page with several additional tabs. Clicking on the ID of the respective iteration or on the icon on the right in the action column takes you to the tab that was selected last. When called for the first time, this is the *Overview* tab.

The following tabs are available: *Overview*, *Properties*, *User Defined*, *Attachments*, *Test Environments*, *Systems under Test*, *Requirements*, *Jobs*, *Results* and *Changes*.

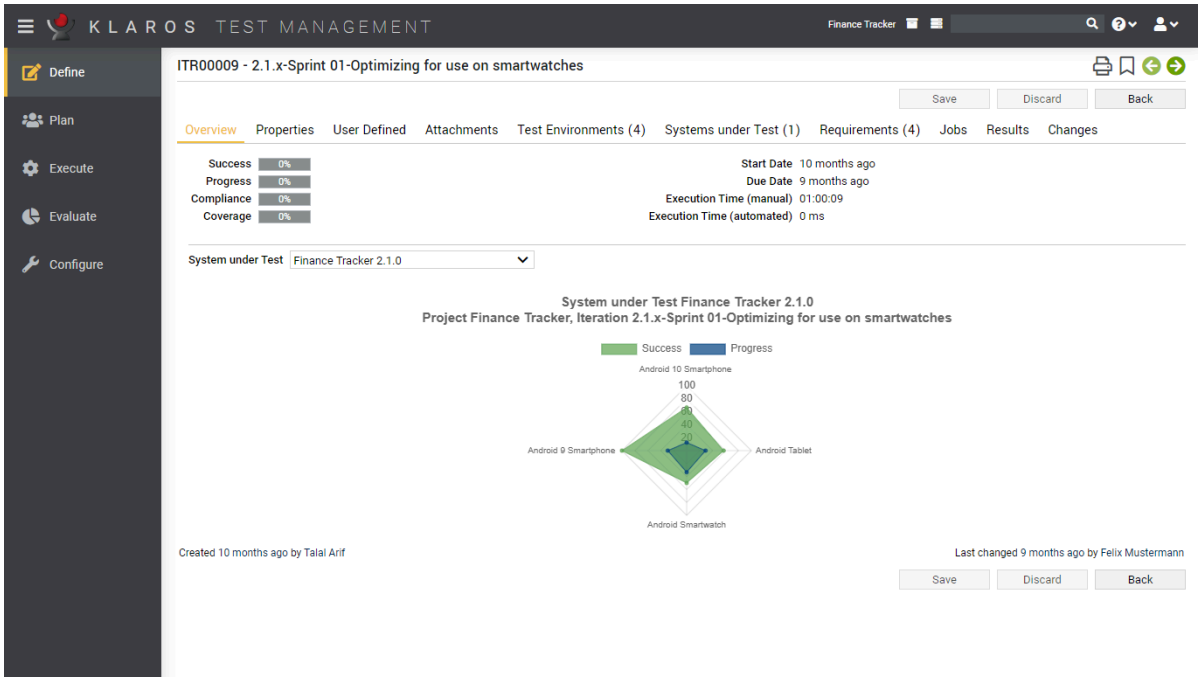


Figure 6.15. The “Overview” Tab

6.2.2.1. Actions

Auf den Detailseiten lassen sich jeweils folgende Aktionen vornehmen:

Open print view

A print-ready view of the iteration can be created here. With a click on the icon this opens in a new browser tab.

Print views are described in detail in Section 5.2.3.2.1, “Print Pages”.

Create Bookmarks

Each individual detail page can also be reached directly via a hyperlink. By clicking on the icon this link is copied to the clipboard.

The creation of bookmarks is described in detail in Section 5.2.3.2.2, “Bookmarks”.

Browse

Use the green arrows at the very top right to switch between the iterations present on the previous page.

6.2.2.2. Overview

All data displayed here always refer to the currently displayed iteration.

The following values are displayed on the overview tab:

Success

The overall success rate of all test cases. A success rate of 100% means that the latest test run for every test case has been successful. Even if a test case has been successfully executed in the past, only the latest result is considered for the success rate.

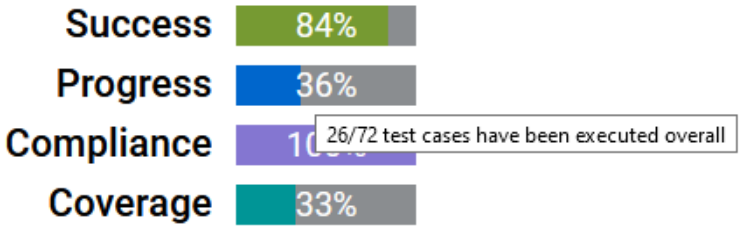


Figure 6.16. The Iteration Success Rate

Progress — The progress rate shows how many test cases have been executed. In contrast to the success rate, the progress rate only takes into account whether a test case was executed at least once in the iteration, regardless of its result.

Compliance — The compliance rate shows how many test cases assigned to requirements have been successfully executed.

Coverage — The coverage rate shows how many test cases assigned to requirements have been executed at least once. In contrast to the compliance rate, the coverage rate considers only if a test case has been executed at least once in the iteration, regardless of its result.

Start Date — The date on which this iteration starts.

Due Date — The date on which this iteration is due to be finished.

Execution Time (manual) and Execution Time (automated) — The overall execution times from manual and automated test executions of the iteration.

System under Test Radar Chart — This chart shows the success and progress rate of this iteration for every combination of test environment and system under test which have been configured for this iteration.

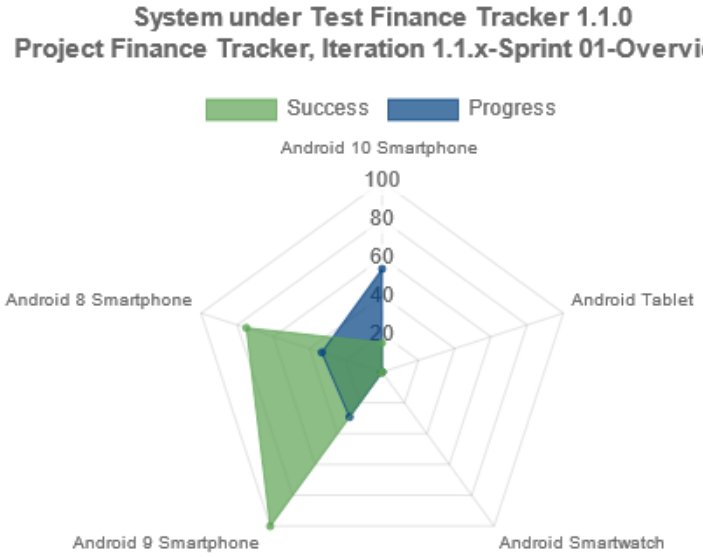


Figure 6.17. The Iteration Radar Chart

Note

All test cases that are part of the iteration (via requirements) contribute to the metrics of this tab. If none are explicitly registered for this iteration, then all test cases of the project are considered.

6.2.2.3. Properties

This tab (Figure 6.18) allows the user to view or change the following attributes of the iteration:

Description A description of the iteration.

Success Criteria A description of the conditions which need to be met for this iteration to be complete.

Start Date The date on which this iteration starts.

Due Date The date on which this iteration is due to be finished.

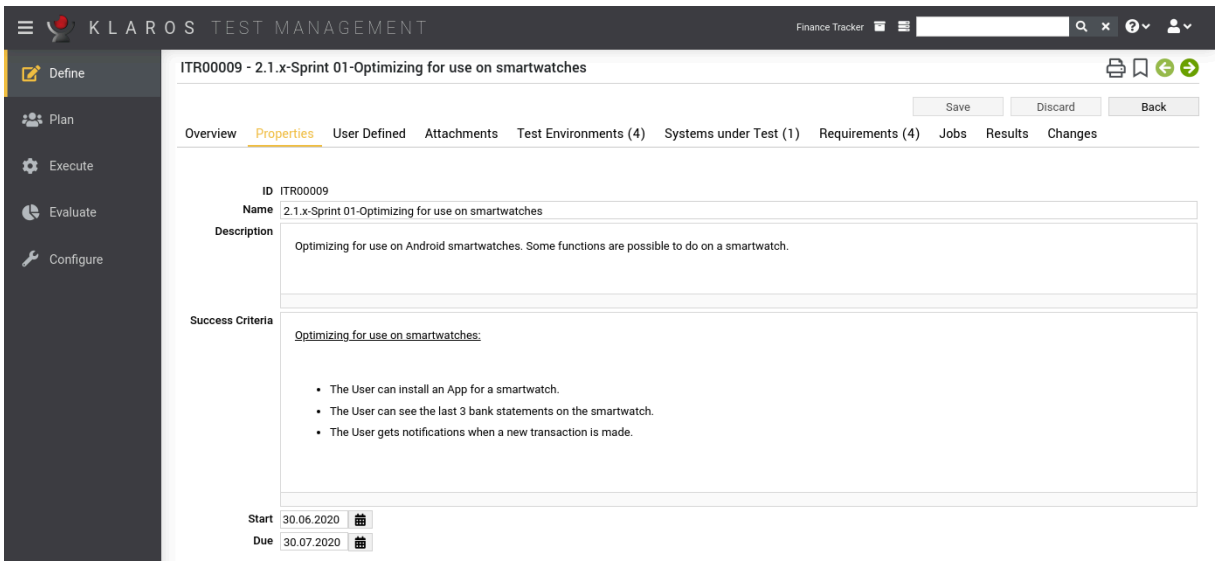


Figure 6.18. The “Properties” Tab - Iterations

6.2.2.4. User-Defined

You can create your own fields to meet individual requirements. For further information please refer to Section 5.2.3.2.4, “User Defined Properties”.

6.2.2.5. Attachments

You can add any files as attachments to an iteration. For more information, see Section 5.2.3.2.6, “Attachments”.

6.2.2.6. Test Environments

If you want to associate test environments with the iteration, click the Assign button and select one or more test environments.

6.2.2.7. Systems under Test

The page lists all *systems under test* assigned to the iteration, with their respective states:

ID	Assigned automatically.
Version	The Version of the system under test.
Success	The percentage of passed tests for this system under test that were executed in this iteration.
Progress	The percentage of tests executed in this iteration for this system under test.
Compliance *	The percentage of tests executed for this system under test that cover the requirements assigned to this iteration.
Coverage*	The percentage of tests executed for this system under test that cover the requirements assigned to this iteration and last had the result *Pass*.

* Only present if at least one requirement is associated with the iteration.

If you want to associate systems under test with the iteration, click the Assign button and select one or more test environments.

6.2.2.8. Requirements

The *Requirements* tab shows the requirements which have been assigned to this iteration, as well as their *Compliance* and *Coverage* rates. These metrics refer to the test cases linked with the requirements and the last results of these executions. The following values are displayed:

ID	Assigned automatically.
Revision	The revision of the requirement.
Name	The name of the requirement
Compliance	The percentage of executed test cases, which cover requirements of this iteration.
Coverage	The percentage of executed test cases, which cover requirements of this iteration and which were last executed with the result *Passed*.

If you want to associate requirements with the iteration, click the Assign button and select one or more requirements.

6.2.2.9. Jobs

The page displays a list of jobs assigned to the selected iteration.

The following values are displayed in the table:

ID	Assigned automatically.
Summary	The summary of the job
Priority	The priority of the job. Possible values are *Trivial*, *Minor*, *Major*, *Critical*, *Blocker*
State	The state of the job. Possible values are *New*, *Reopened*, *In Progress*, *Resolved*, *Closed*, *Rejected*.
Progress	The percentage of executed test cases of this job and its sub-jobs.
Success	The success rate of the executed test cases of this job and its subjobs.
Due Date	The date on which this job is due to be finished.
Assigned	The user to whom the job is assigned.
Action	Available actions are: *Edit*, *Open Print View* und *Execute*. For the possible representations for *Execute*, see Section F.2.2, “Execution Actions”.

6.2.2.10. Results

The results tab is further divided into a *Test Runs*, *Test Case Results* and a *Test Suite Results* tab, showing the test results related to this iteration as described in Section 5.2.3.2.7, “Test Runs and Results”.

6.2.2.11. Changes

The tab *Changes* shows the change history of this iteration.

For a detailed description of the *Changes* view, see Section 5.2.3.2.8, “Change History”.

6.3. Requirements

Only available in Klaros-Testmanagement Enterprise Edition

A *requirement* is a predefined condition to be fulfilled that specifies and describes the properties or performance to be achieved by a product, a process or a system. This chapter shows how to create and edit requirements.

6.3.1. Overview Page

The overview page displays all existing requirements in the selected project in a table. New requirements are created here.

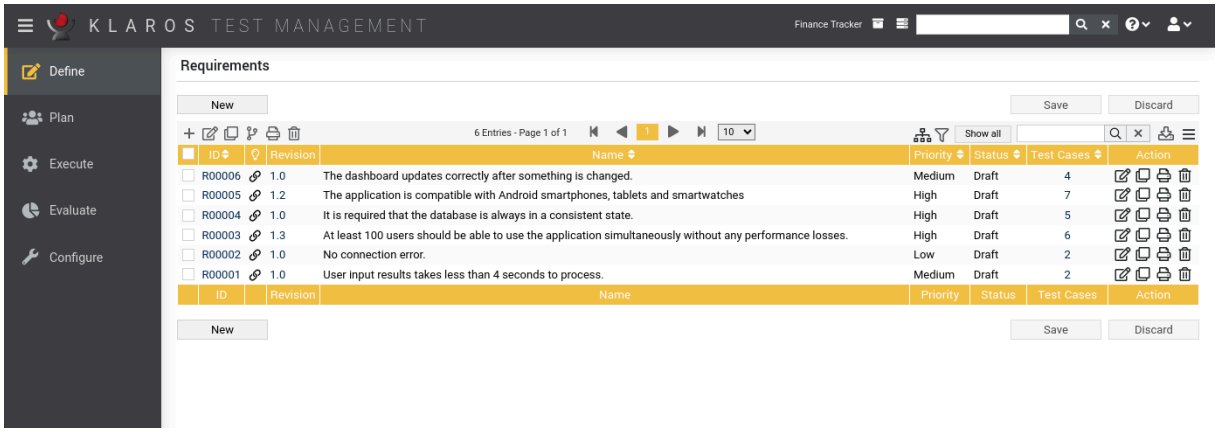


Figure 6.19. The “Requirements” Page

The table shows the following values:

ID	Assigned automatically.
Additional Information	A tooltip appears when the cursor is placed over the icon shown here.
Revision	The revision of the requirement.
Name	The name of the requirement.
Priority	The priority of the requirement. Possible values are *Low*, *Medium*, *High*.
State	The state of the requirement. Possible values are *Locked*, *Approved*, *Draft* and *Skip*.
Test Cases	The number of test cases assigned to this requirement.
Action	The executable actions.

Name, *Priority* and *Status* can be edited directly in the table rows.

6.3.1.1. Creating a new Requirement

By clicking on the button New a new empty table row will appear. Now, *Name*, *Priority* and *Status* can be defined.

With Save the new requirement is created and saved. The ID of the requirement (R00001) is automatically assigned by Klaros-Testmanagement. Click on the ID *R00001* to get to the detail page of the requirement.

With New several table rows can be created and edited at the same time. Only when clicking on Save the data is stored in the database.

With Discard all changes are undone.

Unsaved Changes

All rows marked with red IDs contain changes that have not yet been saved.

6.3.1.2. Action

The following actions can be performed in the action column:

- Edit
- Duplicate
- Open print view
- Delete

If a requirement has been deleted, it is initially marked with a deletion marker and is only visible to administrators. For deleted requirements, the following actions are available instead of *Delete*:

- Restore (only Administrator)
- Irrecoverably remove the requirement from the database

6.3.1.3. Bulk Actions

On the *Requirement* page, one or more requirements can be selected for bulk actions. Bulk actions are described in Section 5.2.3.1.5, “Bulk Actions”.

The following bulk actions are supported for requirements:

- Edit
- Duplicate
- Create new Revision
- Open print view
- Delete
- Restore (only Administrator)
- Irrecoverably remove the requirement from the database (only Administrator).

For additional information, see Section 5.2.3.1.5, “Bulk Actions”.

6.3.1.4. Table Operations

The following operations can be performed in the line above the table on the right:

- Categorize
- Filter / Sort
- Show all / Only active (only Administrator)
- Search
- Export
- Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

6.3.1.5. Remote Synchronization

If the active project is connected to a remote requirement management system, editing of requirements is disabled for all users. An icon will be shown on the upper right side of the requirements list screen. Clicking on it will start a background synchronization of all requirements from the requirement management system to the project.

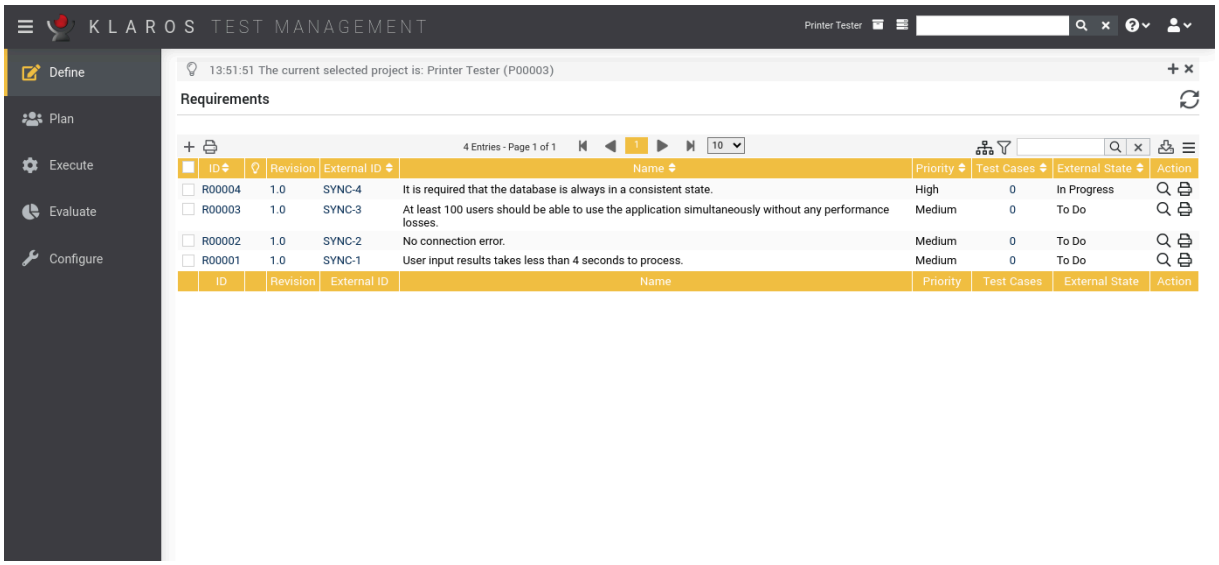


Figure 6.20. The “Requirements Page” Using Remote Synchronization

6.3.2. Details Page

Each requirement has its own detail page with several additional tabs. Clicking on the ID of the respective requirement or on the icon on the right in the action column takes you to the tab that was selected last. When called for the first time, this is the *Overview* tab.

The following tabs are available: *Overview*, *Properties*, *User Defined*, *Attachments*, *Iterations*, *Test Cases*, *Revisions*, *Results* and *Changes*.

6.3.2.1. Actions

On the detail pages there are more icons in the upper right corner. The following actions can be performed here:

Synchronize — Synchronizes the local requirement with the external source.

Open external link — Shows the requirement in the external source.

Open print view — A print-ready view of the requirement can be created here. With a click on the icon this opens in a new browser tab.

Print views are described in detail in Section 5.2.3.2.1, “Print Pages”.

Create Bookmarks — Requirements may be linked to from outside of Klaros-Testmanagement using the link on the icon. By clicking on the icon this link will be copied to the clipboard.

Weitere Informationen hierzu finden Sie unter Section 5.2.3.2.2, “Bookmarks”.

Browse Use the green arrows at the very top right to switch between the requirements present on the previous page.

6.3.2.2. Overview

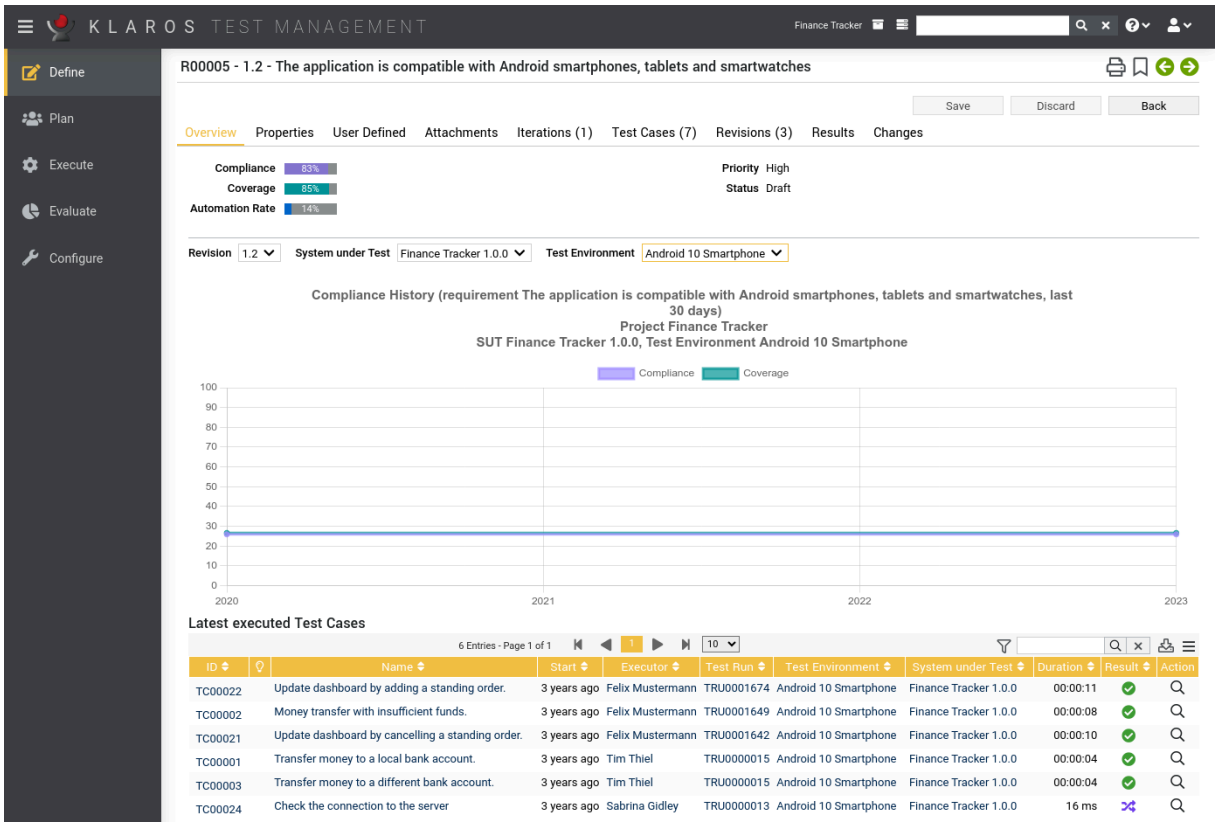


Figure 6.21. The “Overview” Tab

The following values are displayed on the overview tab:

Compliance The conformance rate shows how many test cases assigned to this requirement were performed with the result *Passed*.

Coverage The coverage rate shows how many test cases assigned to this requirement have already been executed.

Automation Rate The automation level indicates the percentage of test cases that can be executed automatically.

Priority The priority of the requirement. Possible values are *Low*, *Medium* and *High*.

Status / External Status The status of the requirement. If the requirement is synchronized from an external system, the externally managed status is displayed here.

Otherwise, the possible values are *Locked*, *Approved*, *Draft* and *Skip*.

Compliance History Chart

This diagram shows the compliance and coverage rate of this requirement for the combination of revision, system under test and test environment set here.

This graph shows how the conformance and coverage for this requirement has changed since it was defined. The timeline is automatically adjusted to reflect the available information.

In the lower area there is a table with the overview of the last executed test cases.

6.3.2.3. Properties

In this tab (Figure 6.22) the following attributes of the requirements are displayed and can be edited if the user has the appropriate permission:

ID

Assigned automatically.

Name

The name of the requirement.

Priority

Die Priorität der Anforderung. Possible values are *Low*, *Medium* and *High*

Status / External Status

The status of the requirement. If the requirement is synchronized from an external system, the externally managed status is displayed here.

Otherwise, the possible values are *Locked*, *Approved*, *Draft* and *Skip*.

Summary

The brief summary of the requirement.

Description

The detailed description of the requirement.

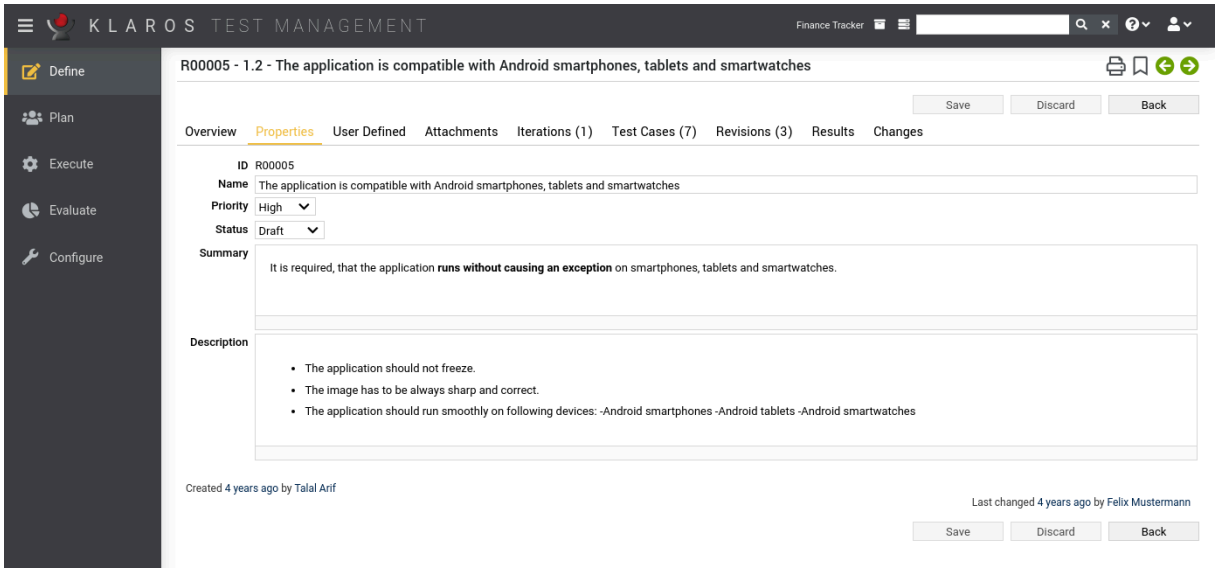


Figure 6.22. The “Properties” Tab

6.3.2.4. User-Defined

You can create your own fields to meet individual requirements. For further information please refer to Section 5.2.3.2.4, “User Defined Properties”.

6.3.2.5. Attachments

You can add any files as attachments to a requirement. For more information, see Section 5.2.3.2.6, “Attachments”.

6.3.2.6. Iterations

If you want to associate the requirement with an iteration, click the Assign button and select one or more requirements.

6.3.2.7. Test Cases

The coverage rate of requirements is tested by test cases. Test cases can be linked to requirements in the *Test Cases* tab. The test results of linked test cases count towards the coverage and compliance rates of the requirement.

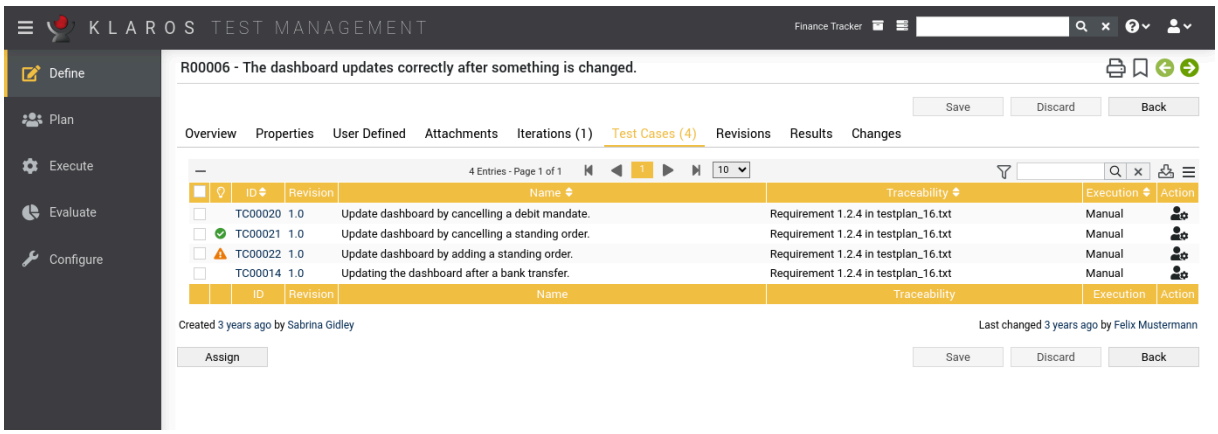


Figure 6.23. The “Test Cases” Tab

Test cases can be assigned to the requirement by clicking the Assign button. This opens up a dialog, where multiple test cases can be selected at once.

The following bulk actions are supported for requirements:

- Remove

The detail page of a test case shows a list of all requirements that are covered by it, as shown in section Section 6.7.2.7, “Revisions”.

6.3.2.8. Revisions

Requirements may exist in different revisions. For more information, see Section 5.2.3.2.5, “Revisions”.

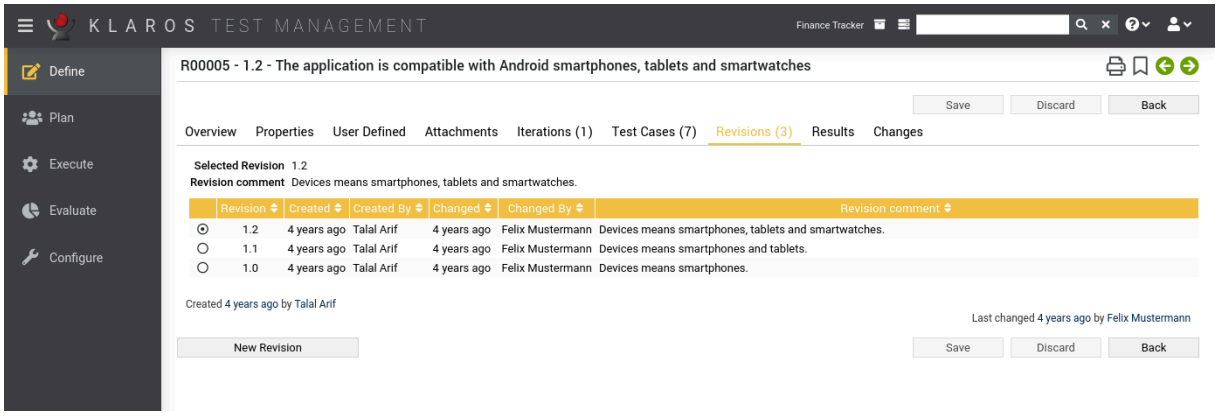


Figure 6.24. The “Revisions” Tab

6.3.2.9. Results

The results tab is further divided into a *Test Runs*, *Test Case Results* and a *Test Suite Results* tab, showing the test results related to this requirement as described in Section 5.2.3.2.7, “Test Runs and Results”.

6.3.2.10. Changes

The tab *Changes* shows the change history of this requirement.

For a detailed description of the *Changes* view, see Section 5.2.3.2.8, “Change History”.

6.4. Test Environments

A *test environment* summarizes the external circumstances that can influence a test result. This can be the operating system, the physical environment, or the version of a client system, web browser, or application server. This chapter shows how to create and edit test environments.

6.4.1. Overview Page

The overview page displays all existing test environments in the selected project in a table. New test environments are created here.

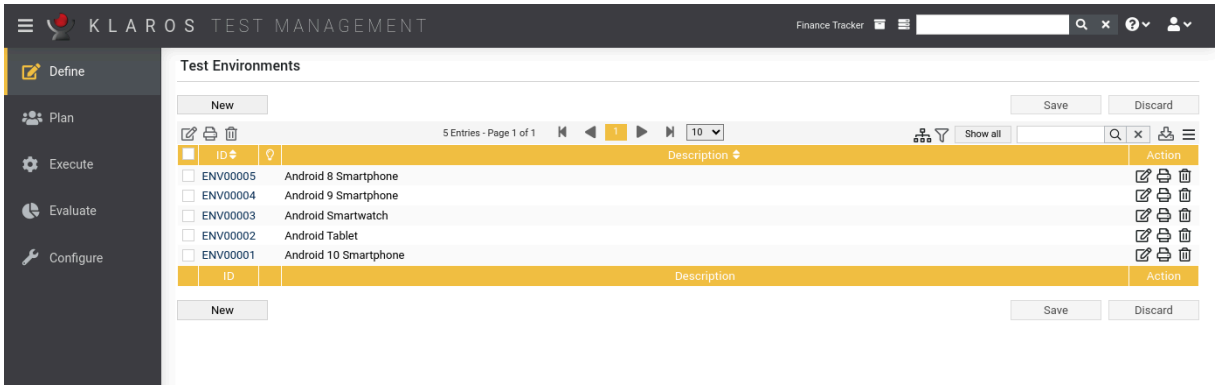


Figure 6.25. The “Test Environments” Page

The table shows the following values:

ID	Assigned automatically.
Additional Information	A tooltip appears when the cursor is placed over the icon shown here.
Description	The description of the test environment.
Actions	The executable actions.

The *Description* entry can be edited directly in the table rows by clicking in the corresponding field.

6.4.1.1. Creating a new Test Environment

By clicking on the button `New` a a new empty table row will appear. Now, a *Description* can be defined.

With `Save` the new test environment is created and saved. The ID of the test environment (ENV00001) is automatically assigned by Klaros-Testmanagement. Click on the ID *ENV00001* to get to the detail page of the test environment.

With `New` several table rows can be created and edited at the same time. Only when clicking on `Save` the data is stored in the database.

With `Discard` all changes are undone.

Red IDs

All rows with red IDs have been changed and are not yet saved!

6.4.1.2. Actions

The action column is located on the far right of the table. The following actions can be performed here:

- Edit
- Open print view
- Delete

If a test environment has been deleted, it is initially marked with a deletion marker and is only visible to administrators. For deleted test environments, the following actions are available instead of *Delete*:

- Restore (only Administrator)
- Irrecoverably remove the test environment from the database

6.4.1.3. Bulk Actions

Some actions can be applied to multiple test environments at the same time. To do this, select the test environments to which the action is to be applied in the leftmost column.

The following bulk actions are supported for test environments:

- Edit
- Open print view
- Delete
- Restore (only Administrator)
- Irrecoverably remove the test environment from the database (only Administrator).
- Assign to a category (appears only after a category has been created).

Bulk actions are described in detail in Section 5.2.3.1.5, “Bulk Actions”

6.4.1.4. Table operations

The following operations can be performed in the line above the table on the right:

- Categorize
- Filter / Sort
- Show all / Only active (only Administrator)
- Search
- Export
- Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

6.4.2. Details Page

Each test environment has its own detail page with several additional tabs. Clicking on the ID of the respective test environment or on the icon on the right in the action column takes you to the tab that was selected last. When called for the first time, this is the *Overview* tab.

The following tabs are available: *Overview*, *Properties*, *User Defined*, *Attachments*, *Jobs*, *Results* and *Changes*.

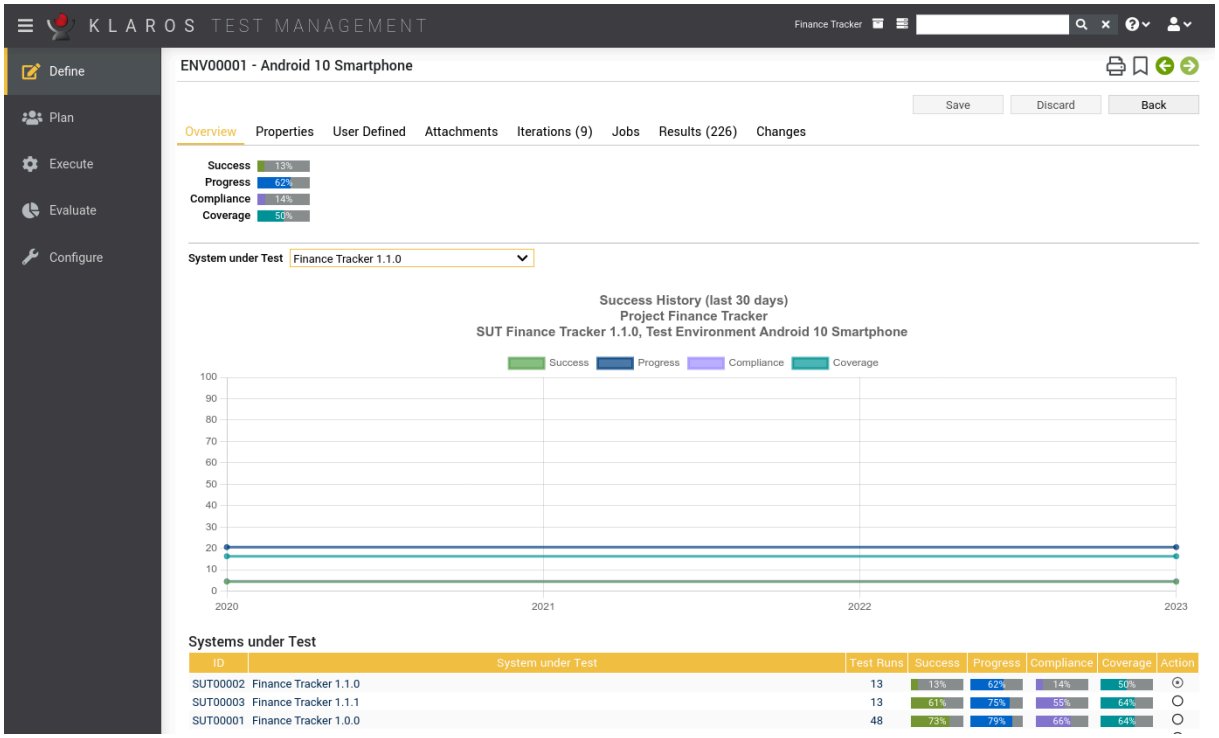


Figure 6.26. The “Overview” Tab

6.4.2.1. Actions

On the detail pages, there are additional icons in the top right corner of the header. The following actions can be performed here:

Open print view

A print-ready view of the test environment can be created here. With a click on the icon this opens in a new browser tab.

Print views are described in detail in Section 5.2.3.2.1, “Print Pages”.

Create Bookmarks

Each individual detail page can also be reached directly via a hyperlink. By clicking on the icon this link is copied to the clipboard.

The creation of bookmarks is described in detail in Section 5.2.3.2.2, “Bookmarks”.

Browse

Use the green arrows at the very top right to switch between the test environments present on the previous page.

6.4.2.2. Overview

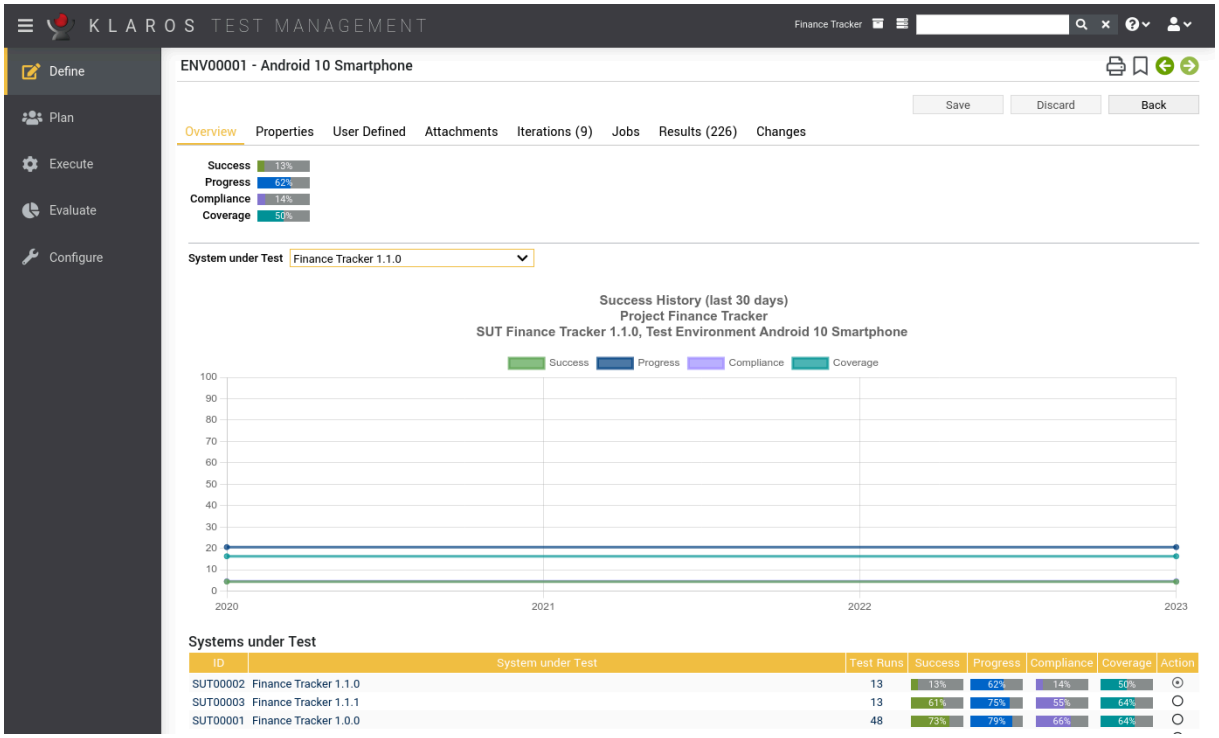


Figure 6.27. The “Overview” Tab

All data displayed here always refer to the currently displayed test environment.

The following values are displayed on the overview tab:

Success — The overall success rate of all test cases. A success rate of 100% means that the latest test run for every test case has been successful. Even if a test case has been successfully executed in the past, only the latest result is considered for the success rate.

Progress — The progress rate shows how many test cases have been executed. In contrast to the success rate, the progress rate only takes into account whether a test case was executed at least once, regardless of its result.

Compliance — The compliance rate shows how many test cases assigned to requirements have been successfully executed.

Coverage — The coverage rate shows how many test cases assigned to requirements have been executed at least once. In contrast to the compliance rate, the coverage rate considers only if a test case has been executed at least once in the iteration, regardless of its result.

Automation Rate — The automation level indicates the percentage of test cases that can be executed automatically.

Success History Diagram This chart shows the historical evolution of the success, progress, conformance, and coverage rate of this test environment for the selected system under test.

The timeline is automatically adjusted to the existing data.

In the lower area you find a table with the key figures of this test environment broken down by system under test.

6.4.2.3. Properties

This tab (Figure 6.28) allows the user to view or change the following attributes of the test environment:

Description A description of the Testumgebung.

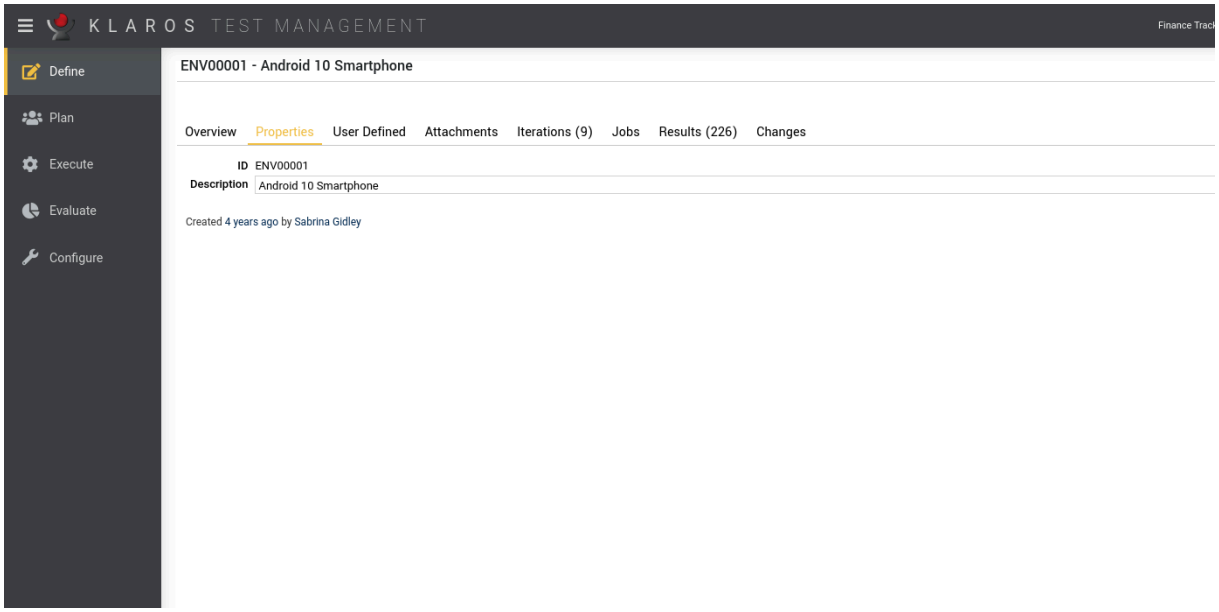


Figure 6.28. The “Properties” Tab

6.4.2.4. User-Defined

You can create your own fields to meet individual requirements. For further information please refer to Section 5.2.3.2.4, “User Defined Properties”.

6.4.2.5. Attachments

You can add any files as attachments to a test environment. For more information, see Section 5.2.3.2.6, “Attachments”.

6.4.2.6. Iterations

This page lists all iterations to which this test environment is assigned.

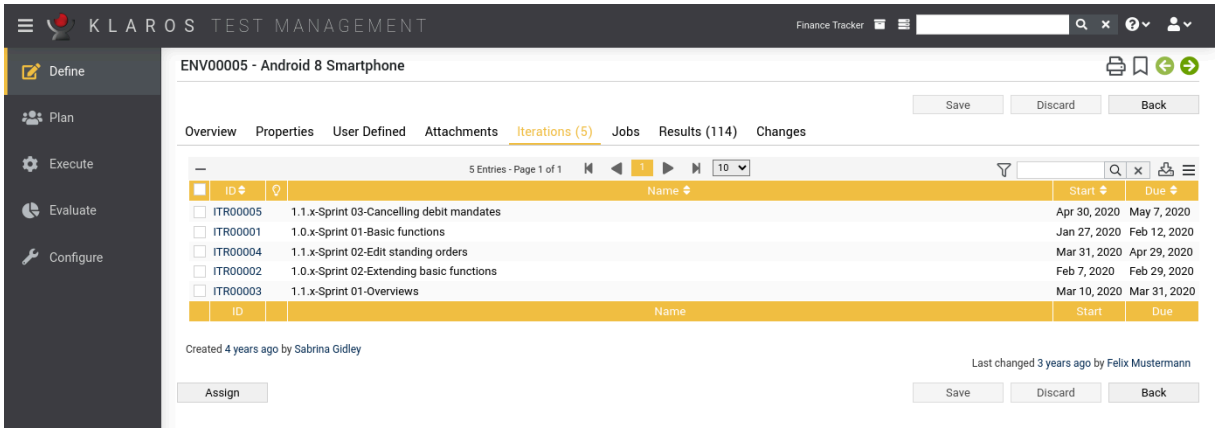


Figure 6.29. The “Iterations” Tab

This test environment can be assigned to one or more iterations by clicking the Assign button. This opens up a dialog, where multiple iterations can be selected at once.

The following bulk actions are supported for iterations:

- Remove

The detail page of an iteration shows a list of all test environments that are assigned to it, as shown in section Section 6.2.2.6, “Test Environments”.

6.4.2.7. Changes

The tab *Changes* shows the change history of this test environment.

For a detailed description of the *Changes* view, see Section 5.2.3.2.8, “Change History”.

6.5. Systems under Test

With *System under Test* the system/object is designated, which is to be tested. Other common terms for this are *test item* or *test object*. This can be identified, for example, by the version number of the tested software. This chapter shows how to create and edit *systems under test*.

6.5.1. Overview Page

The overview page displays all existing systems under test in the selected project in a table. New systems under test are created here.

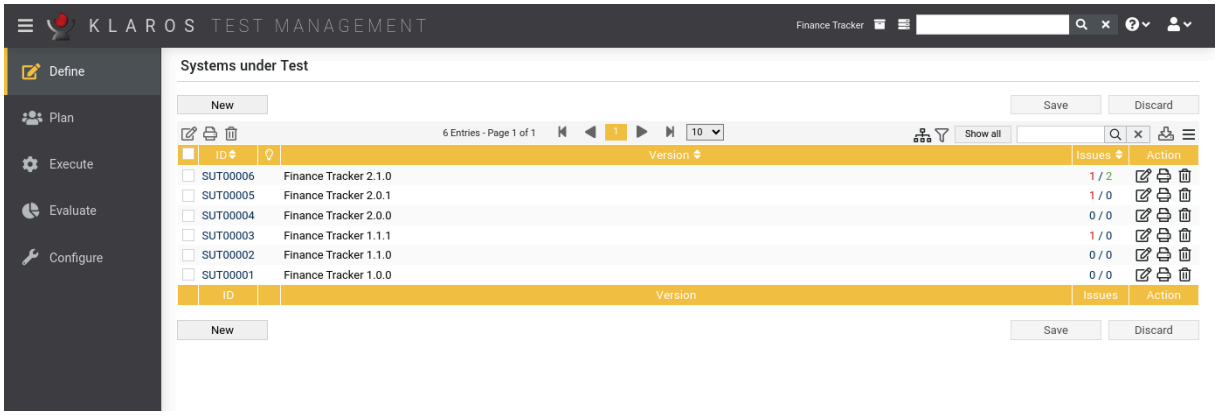


Figure 6.30. The “System under Test” Page

The table shows the following values:

ID Assigned automatically.

Additional Information A tooltip appears when the cursor is placed over the icon shown here.

Description The description of the system under test.

Actions The executable actions.

The *Description* entry can be edited directly in the table rows by clicking in the corresponding field.

6.5.1.1. Creating a new System under Test

By clicking on the button New a new empty table row will appear. Now, the *Version* of the system under test can be defined.

With Save the new system under test is created and saved. The ID of the system under test (SUT00001) is automatically assigned by Klaros-Testmanagement. Click on the ID *SUT00001* to get to the detail page of the system under test

With New several table rows can be created and edited at the same time. Only when clicking on Save the data is stored in the database.

With Discard all changes are undone.

Red IDs

All rows with red IDs have been changed and are not yet saved!

6.5.1.2. Actions

The following actions can be performed in the action column:

- Edit
- Open print view

- Delete

If a system under test has been deleted, it is initially marked with a deletion marker and is only visible to administrators. For deleted systems under test, the following actions are available instead of *Delete*:

- Restore (only Administrator)
- Irrecoverably remove the system under test from the database

6.5.1.3. Bulk Actions

Certain actions can also be applied to several systems under test at the same time. To do this, select the systems under test to which the action is to be applied in the leftmost column.

The following bulk actions are supported for systems under test:

- Edit
- Open print view
- Delete
- Restore (only Administrator)
- Irrecoverably remove the system under test from the database (only Administrator).
- Assign to a category (appears only after a category has been created).

Bulk actions are described in detail in Section 5.2.3.1.5, “Bulk Actions”

6.5.1.4. Table Operationen

The following operations can be performed in the line above the table on the right:

- Categorize
- Filter / Sort
- Show all / Only active (only Administrator)
- Search
- Export
- Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

6.5.2. Details Page

Each system under test has its own detail page with several additional tabs. Clicking on the ID of a system under test or on the icon on the right in the action column takes you to the tab that was selected last. When called for the first time, this is the *Overview* tab.

The following tabs are available: *Overview*, *Properties*, *User Defined*, *Attachments*, *Jobs*, *Results* and *Changes*.

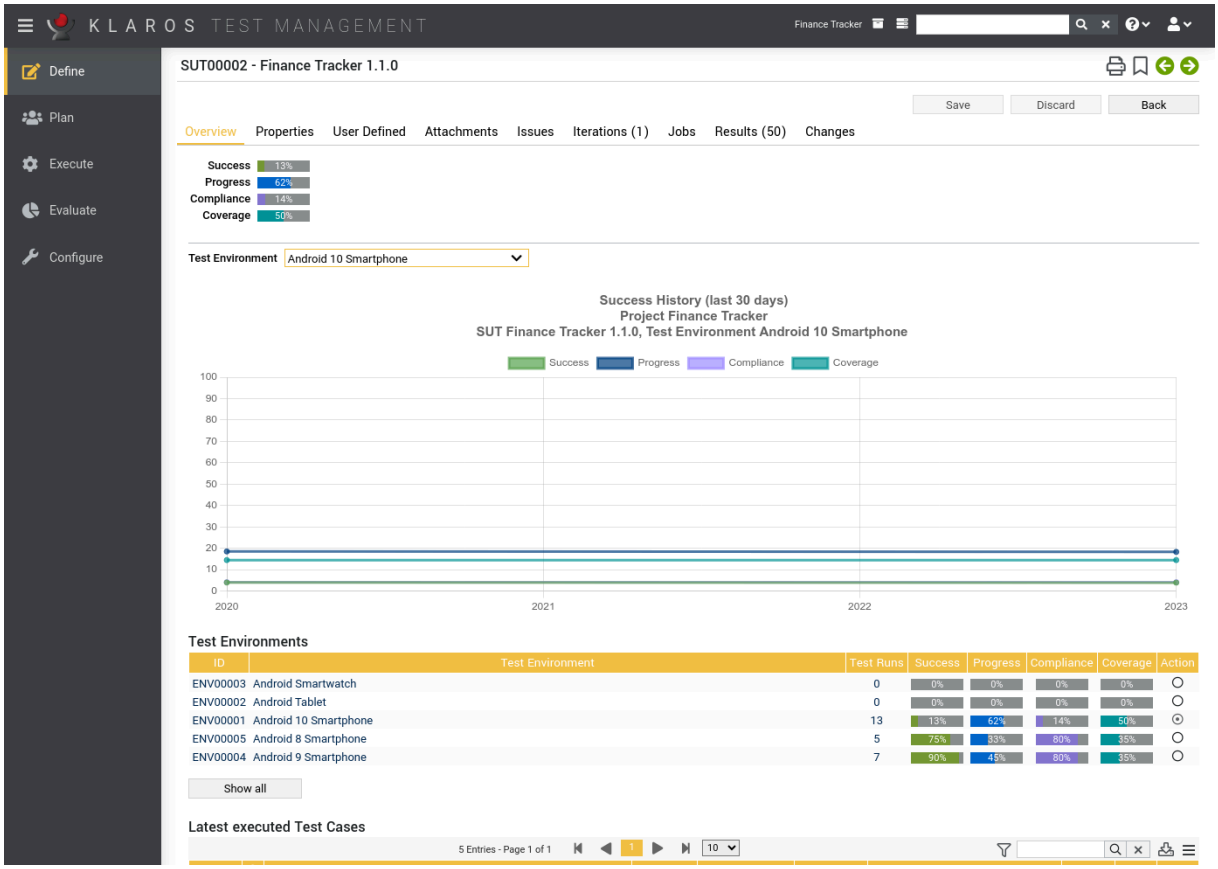


Figure 6.31. The “Overview” Tab

6.5.2.1. Actions

On the detail pages, there are additional icons in the top right corner of the header. The following actions can be performed here:

Open print view

A print-ready view of the system under test can be created here. With a click on the icon this opens in a new browser tab.

Print views are described in detail in Section 5.2.3.2.1, “Print Pages”.

Create Bookmarks

Each individual detail page can also be reached directly via a hyperlink. By clicking on the icon this link is copied to the clipboard.

The creation of bookmarks is described in detail in Section 5.2.3.2.2, “Bookmarks”.

Browse

Use the green arrows at the very top right to switch between the systems under test present on the previous page.

6.5.2.2. Overview

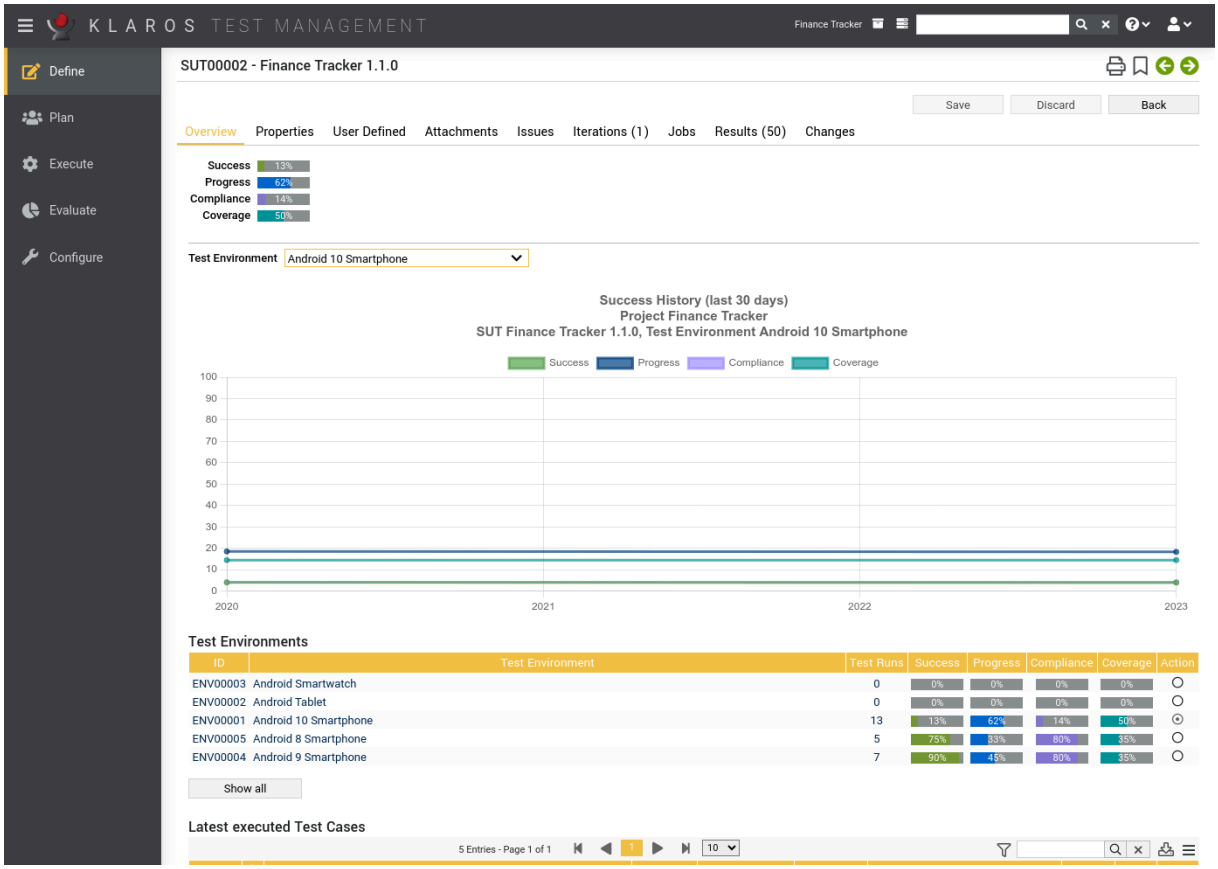


Figure 6.32. The “Overview” Tab

All data displayed here always refer to the currently displayed system under test.

The following values are displayed on the overview tab:

Success The overall success rate of all test cases. A success rate of 100% means that the latest test run for every test case has been successful. Even if a test case has been successfully executed in the past, only the latest result is considered for the success rate.

Progress The progress rate shows how many test cases have been executed. In contrast to the success rate, the progress rate only takes into account whether a test case was executed at least once, regardless of its result.

Compliance The compliance rate shows how many test cases assigned to requirements have been successfully executed.

Coverage The coverage rate shows how many test cases assigned to requirements have been executed at least once. In contrast to the compliance rate, the coverage rate considers only if a test case has been executed at least once in the iteration, regardless of its result.

Automation Rate The automation level indicates the percentage of test cases that can be executed automatically.

Success history diagram This chart shows the historical evolution of the success, progress, conformance, and coverage rate of this system under test for the selected test environment.

The timeline is automatically adjusted to the existing data.

In the lower area there is a table with the overview of the results for this test system broken down by test environment. In the *Action* column, the test environment for the above diagram can be selected.

6.5.2.3. Properties

This tab (Figure 6.33) allows the user to view or change the following attributes of the system under test:

Version The version of the system under test.

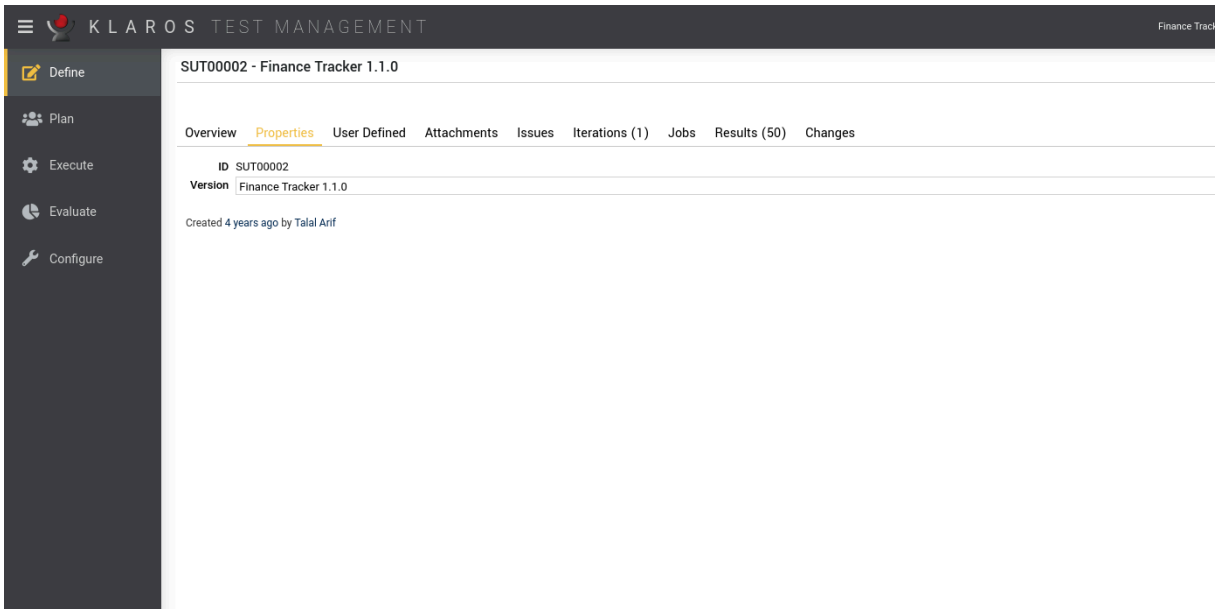


Figure 6.33. The “Properties” Tab

6.5.2.4. User-Defined

You can create your own fields to meet individual requirements. For further information please refer to Section 5.2.3.2.4, “User Defined Properties”.

6.5.2.5. Attachments

You can add any files as attachments to a system under test. For more information, see Section 5.2.3.2.6, “Attachments”.

6.5.2.6. Issues

This view lists all issues that are associated with this system under test.

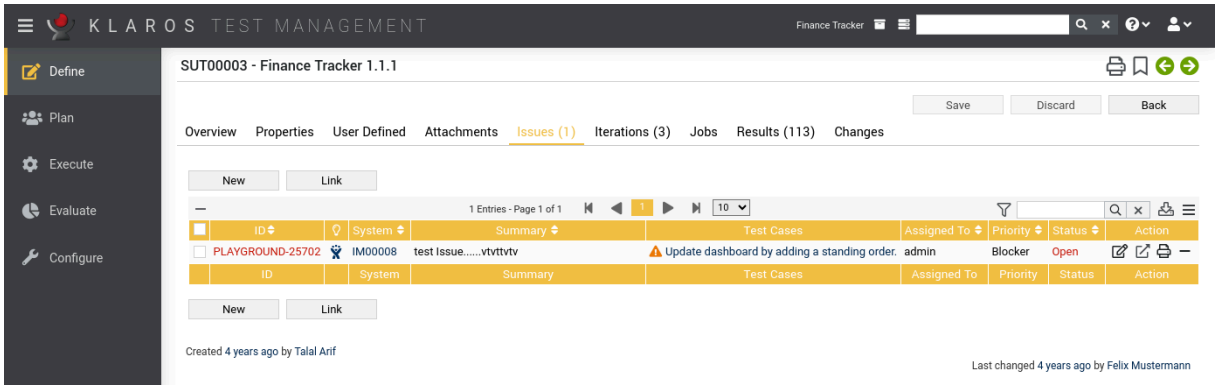


Figure 6.34. The “Issues” Tab

The table shows the following values:

ID	The ID of the issue in the external issue management system.
System	The external issue management system of this issue.
Additional Information	A tooltip appears when the cursor is placed over the icon shown here.
Summary	The summary of the issue.
Created	The date when the issue was created.
Created by	The user who created the issue.
Assigned	The user to whom the issue is assigned.
Priority	The priority of the issue. The possible values are specified by the external issue management system.
Status	The status of the issue. The possible values are specified by the external issue management system.
Actions	The executable actions.

6.5.2.6.1. Creating a new Issue

Clicking `New` opens the *Issue Details* page (Section 9.6.6, “Issue Details (Creating a new Issue)”), where new issues can be created and linked to the system under test. Clicking `Link` links existing issues to the system under test (Section 9.6.7, “Link Issue”).

6.5.2.6.2. Action

The following actions can be performed in the action column:

- Edit
- Browse
- Delete

6.5.2.7. Iterations

This page lists all iterations to which this system under test is assigned.

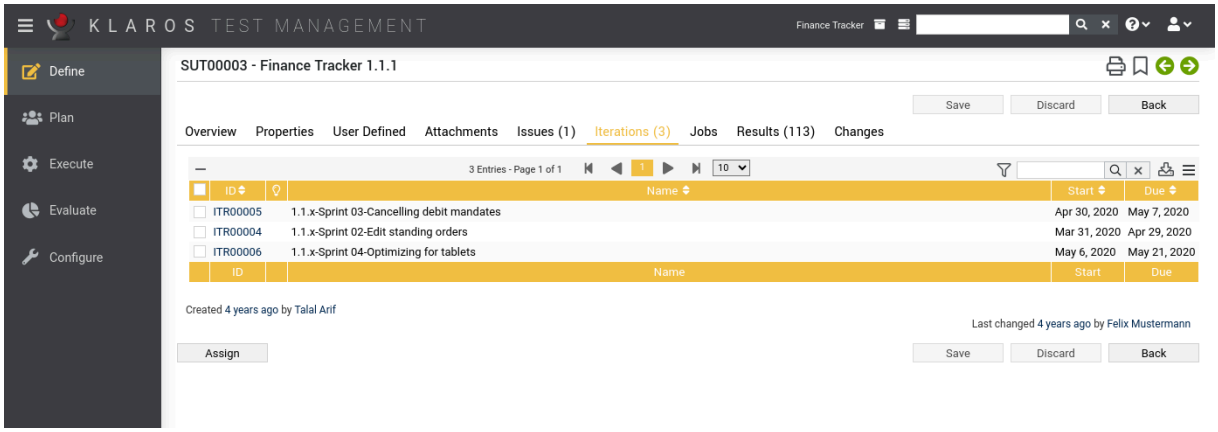


Figure 6.35. The "Iterations" Tab

This system under test can be assigned to one or more iterations by clicking the Assign button. This opens up a dialog, where multiple iterations can be selected at once.

The following bulk actions are supported for iterations:

- Remove

The detail page of an iteration shows a list of all systems under test that are assigned to it, as shown in section Section 6.2.2.7, "Systems under Test".

6.5.2.8. Jobs

The page displays the list of jobs assigned to the selected system under test.

The following values are displayed in the table:

ID	Assigned automatically.
Summary	The summary of the job
Priority	The priority of the job. Possible values are *Trivial*, *Minor*, *Major*, *Critical*, *Blocker*
State	The status of the job. Possible values are *New*, *Reopened*, *In Progress*, *Resolved*, *Closed*, *Rejected*.
Progress	The percentage of executed test cases of this job and its sub-jobs.
Success	The success rate of the executed test cases of this job and its subjobs.
Due Date	The date on which this job is due to be finished.
Assigned	The user to whom the job is assigned.
Action	Available actions are: *Edit*, *Open Print View* und *Execute* For the possible representations for *Execute*, see Section F.2.2, "Execution Actions".

6.5.2.9. Results

The results tab is further divided into a *Test Runs*, *Test Case Results* and a *Test Suite Results* tab, showing the test results related to this system under test as described in Section 5.2.3.2.7, “Test Runs and Results”.

6.5.2.10. Changes

The tab *Changes* shows the change history of this system under test.

For a detailed description of the *Changes* view, see Section 5.2.3.2.8, “Change History”.

6.6. Test Segments

Only available in Klaros-Testmanagement Enterprise Edition

A *test segment* defines and encapsulates a sequence of test steps. In test cases, these test segments can be shown as test steps. The content of a test segment is edited centrally here and the changes are automatically applied in the test cases. This chapter shows how test segments are created and edited.

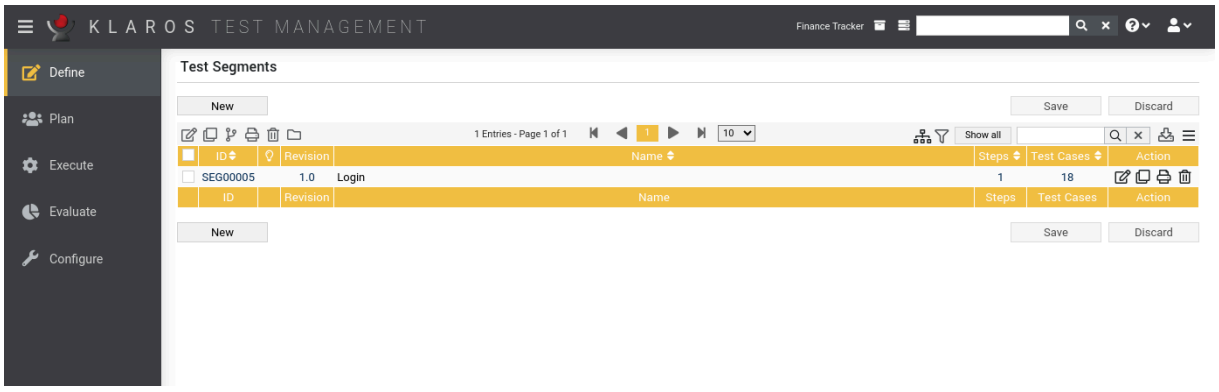


Figure 6.36. The “Test Segments” Page

6.6.1. Overview Page

The overview page displays all existing test segments in the selected project in a table. New test segments are created here.

The table shows the following values:

ID — Assigned automatically.

Additional Information — A tooltip appears when the cursor is placed over the icon shown here.

Revision — The revision of the test segment.

Name — The name of the test segment.

Steps — The number of defined test steps.

Test Cases The number of test cases in which this test segment occurs.

The *Name* entry can be edited directly in the table rows by clicking in the corresponding field.

6.6.1.1. Creating a new Test Segment

By clicking on the button `New` a a new empty table row will appear. Now, a *Name* can be defined.

With `Save` the new test segment is created and saved. The ID of the test segment (SEG00001) is automatically assigned. Clicking on the ID *SEG00001* takes you to the detail page of the test segment.

With `New` several table rows can be created and edited at the same time. Only when clicking on `Save` the data is stored in the database.

With `Discard` all changes are undone.

Red IDs

All rows with red IDs have been changed and are not yet saved!

6.6.1.2. Actions

The following actions can be performed in the action column:

- Edit
- Duplicate
- Open print view
- Delete

If a test segment has been deleted, it is initially marked with a deletion marker and is only visible to administrators. For deleted test segments, the following actions are available instead of *Delete*:

- Restore (only Administrator)
- Irrecoverably remove the test segment from the database

6.6.1.3. Bulk Actions

Certain actions can also be applied to several test segments simultaneously. To do this, select the test segments to which the action is to be applied in the leftmost column.

The following bulk actions are supported for System under Test:

- Edit
- Duplicate
- Create new Revision
- Open print view
- Delete

Restore (only Administrator)

Irrecoverably remove the test segment from the database (only Administrator).

Bulk actions are described in detail in Section 5.2.3.1.5, “Bulk Actions”

6.6.1.4. Table operations

The following operations can be performed in the line above the table on the right:

Filter / Sort

Show all / Only active (only Administrator)

Search

Export

Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

6.6.2. Details Page

Each test segment has its own detail page with several additional tabs. Clicking on the ID of the relevant test segment or on the icon on the right in the action column takes you to the tab that was selected last. When called for the first time, this is the *Properties* tab.

The following tabs are available: *Properties*, *User Defined*, *Steps*, *Attachments*, *Revisions*, *Test Cases*, and *Changes*.

6.6.2.1. Actions

On the detail pages, there are additional icons in the top right corner of the header. The following actions can be performed here:

Open print view

A print-ready view of the test segment can be created here. With a click on the icon this opens in a new browser tab.

Print views are described in detail in Section 5.2.3.2.1, “Print Pages”.

Create Bookmarks

Each individual detail page can also be reached directly via a hyperlink. By clicking on the icon this link is copied to the clipboard.

The creation of bookmarks is described in detail in Section 5.2.3.2.2, “Bookmarks”.

Browse

Use the green arrows at the very top right to switch between the test segments present on the previous page.

6.6.2.2. Properties

In this view (Figure 6.37). the following attributes of the test segment can be displayed and can be edited:

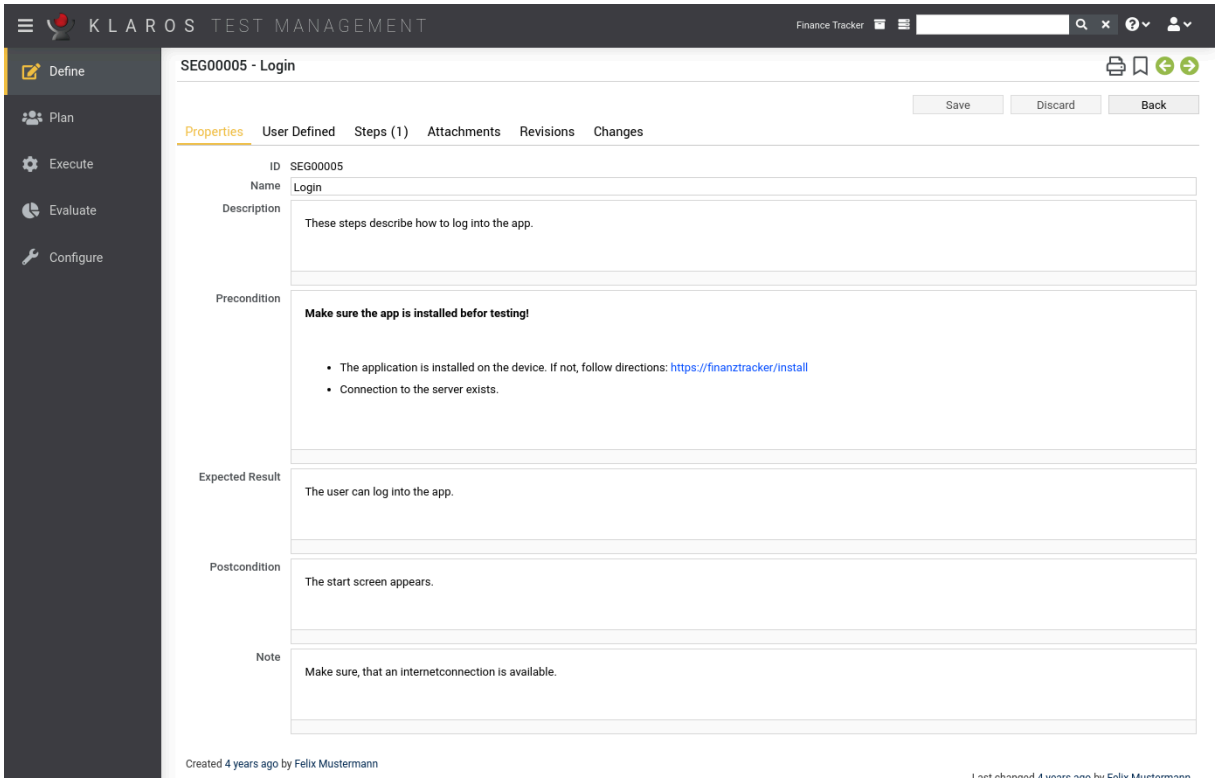


Figure 6.37. The “Properties” Tab

ID Assigned automatically.

Name The name of the test segment.

Description The textual description of the test segment.

Precondition The precondition of the test segment.

Expected Result The expected result of the test segment.

Postcondition The postcondition of the test segment.

Note An optional note.

Editor Settings

Under Define --> Project --> Properties you can set whether the editor displays only text or HTML. With the HTML setting, a bar with formatting options appears when you click in the text field.

Clicking the Save button saves the selected changes to the database, Discard discards the changes.

6.6.2.3. User-Defined

You can create your own fields to meet individual requirements. For further information please refer to Section 5.2.3.2.4, “User Defined Properties”.

6.6.2.4. Test Steps

The test steps contain the following information:

Actions — The description of the action to take, e.g. `Enter name and password, and click the Login button.`

Expected Result — The (visible) behavior of the system predicted while carrying out the test step e.g. `The user receives a message that the login succeeded.`

Precondition — The Condition that must be fulfilled before the execution of the test step, e.g. `The user is not yet logged in.`

Postcondition — The environmental and state conditions that must be fulfilled after the execution of the test case, e.g. `The user is authenticated and has access to the system..`

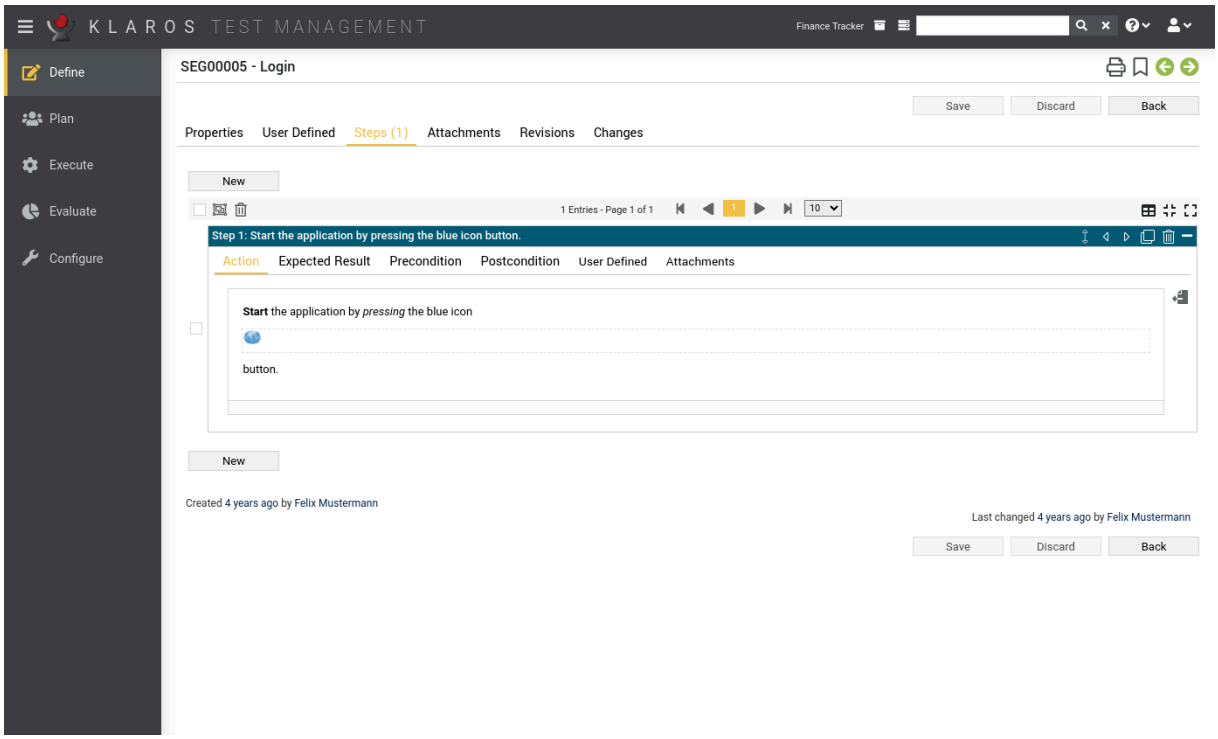


Figure 6.38. The “Steps” Tab

6.6.2.5. Attachments

You can add any files as attachments to a test segment. For more information, see Section 5.2.3.2.6, “Attachments”.

6.6.2.6. Revisions

Test segments may exist in different revisions. For more information, see Section 5.2.3.2.5, “Revisions”.

6.6.2.7. Test Cases

The page displays a list of test cases assigned to the selected system under test.

The following values are displayed in the table:

ID Assigned automatically.

Revision The revision of the test segment.

Name The name of the test segment.

Traceability A reference to the corresponding requirement, use case or work package.

State The state of the test case. Possible values are *Locked*, *Approved*, *Draft* and *Skip*.

Steps The number of test steps.

For more information see Section 6.7, “Test Cases”.

6.6.2.8. Changes

The tab *Changes* shows the change history of this test segment.

For a detailed description of the *Changes* view, see Section 5.2.3.2.8, “Change History”.

6.7. Test Cases

A test case is a set of input values, execution preconditions, expected results and execution post-conditions, developed for a particular objective or test condition, in order to determine whether an application or software system meets its specifications. This chapter shows how to create and edit test cases.

6.7.1. Overview Page

The overview page displays all existing test cases in the selected project in a table. New test cases are created here.

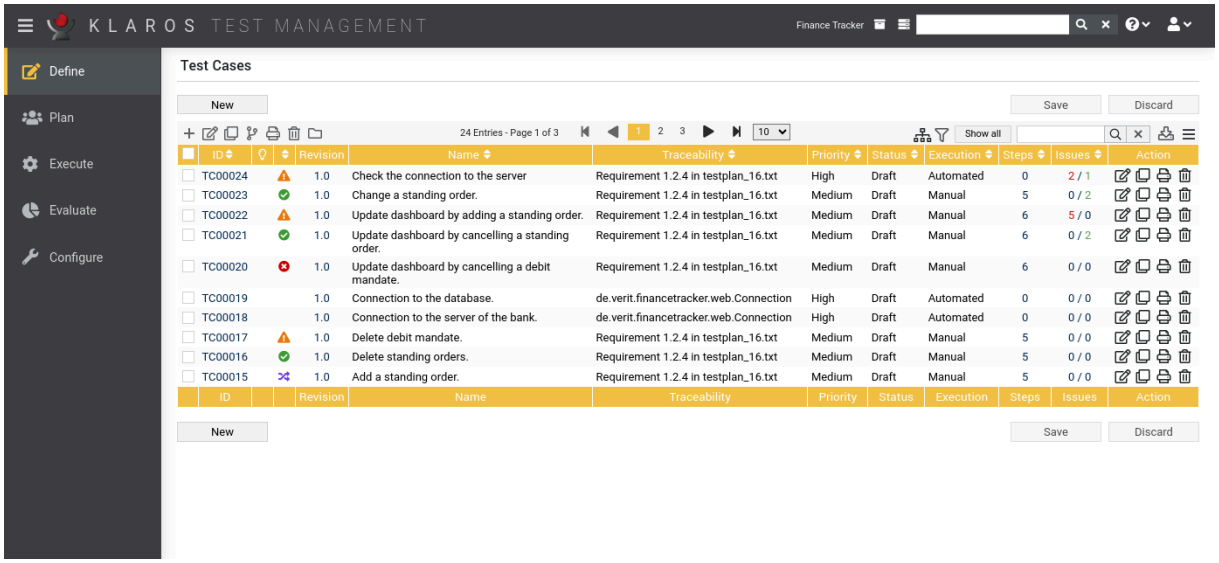


Figure 6.39. The “Test Cases” Page

The table shows the following values:

ID Assigned automatically.

Additional Information A tooltip appears when the cursor is placed over the icon shown here.

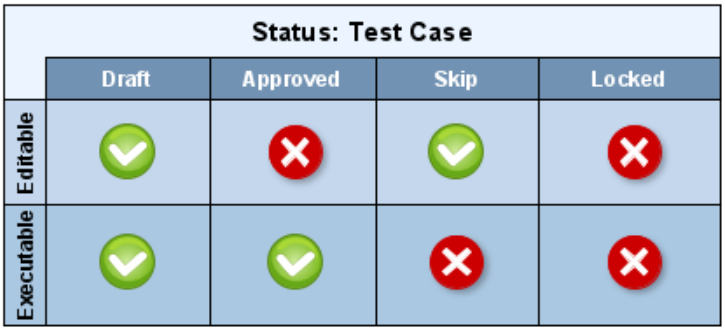
Revision Die Revision des Testfalles.

Name The name of the test case.

Traceability The requirement assigned to this test case.

Priority The priority of the test case. Possible values are *Low*, *Medium*, *High*.

State The test case state determines whether it is editable or executable. Only test cases with the state Draft or Skip are editable. Only test cases with the state Draft or Approved are executable.



Execution Manual oder Automated.

Steps The number of associated test steps.

Issues Die Anzahl der offenen und behobenen Issues.

Action The executable actions.

Name, *Traceability*, *Priority*, *Status* and *Execution* can be edited directly in the table rows.

6.7.1.1. Creating a new Test Case

By clicking on the button `New` a new empty table row will appear. Now, *Name*, *Traceability*, *Priority* and *Execution* can be defined.

With `Save` the new test case is created and saved. The ID of the test case (TC00001) is automatically assigned by Klaros-Testmanagement. Click on the ID *TC00001* to get to the detail page of the test case.

With `New` several table rows can be created and edited at the same time. Only when clicking on `Save` the data is stored in the database.

With `Discard` all changes are undone.

Unsaved Changes

All rows marked with red IDs contain changes that have not yet been saved.

6.7.1.2. Action

The following actions can be performed in the action column:

- Edit
- Duplicate
- Open print view
- Delete

If a test case has been deleted, it is initially marked with a deletion marker and is only visible to administrators. For deleted test cases, the following actions are available instead of *Delete*:

- Restore (only Administrator)
- Irrecoverably remove the test case from the database

6.7.1.3. Bulk Actions

On the *Test Cases* page, one or more test case can be selected for bulk actions. Bulk actions are described in Section 5.2.3.1.5, “Bulk Actions”.

The following bulk actions are supported for test cases:

- Assign to a new test suite
- Edit
- Duplicate
- Create new Revision
- Open print view
- Delete
- Restore (only Administrator)
- Irrecoverably remove the test case from the database (only Administrator).

For additional information, see Section 5.2.3.1.5, “Bulk Actions”.

6.7.1.4. Table Operations

The following operations can be performed in the line above the table on the right:

- Categorize
- Filter / Sort

Show all / Only active (only Administrator)

Search

Export

Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

6.7.2. Details Page

Each test case has its own detail page with several additional tabs. Clicking on the ID of the respective test case or on the icon on the right in the action column takes you to the tab that was selected last. When called for the first time, this is the *Overview* tab.

The following tabs are available: *Overview*, *Properties*, *User Defined*, *Steps*, *Attachments*, *Revisions*, *Issues*, *Jobs*, *Results* and *Changes*.

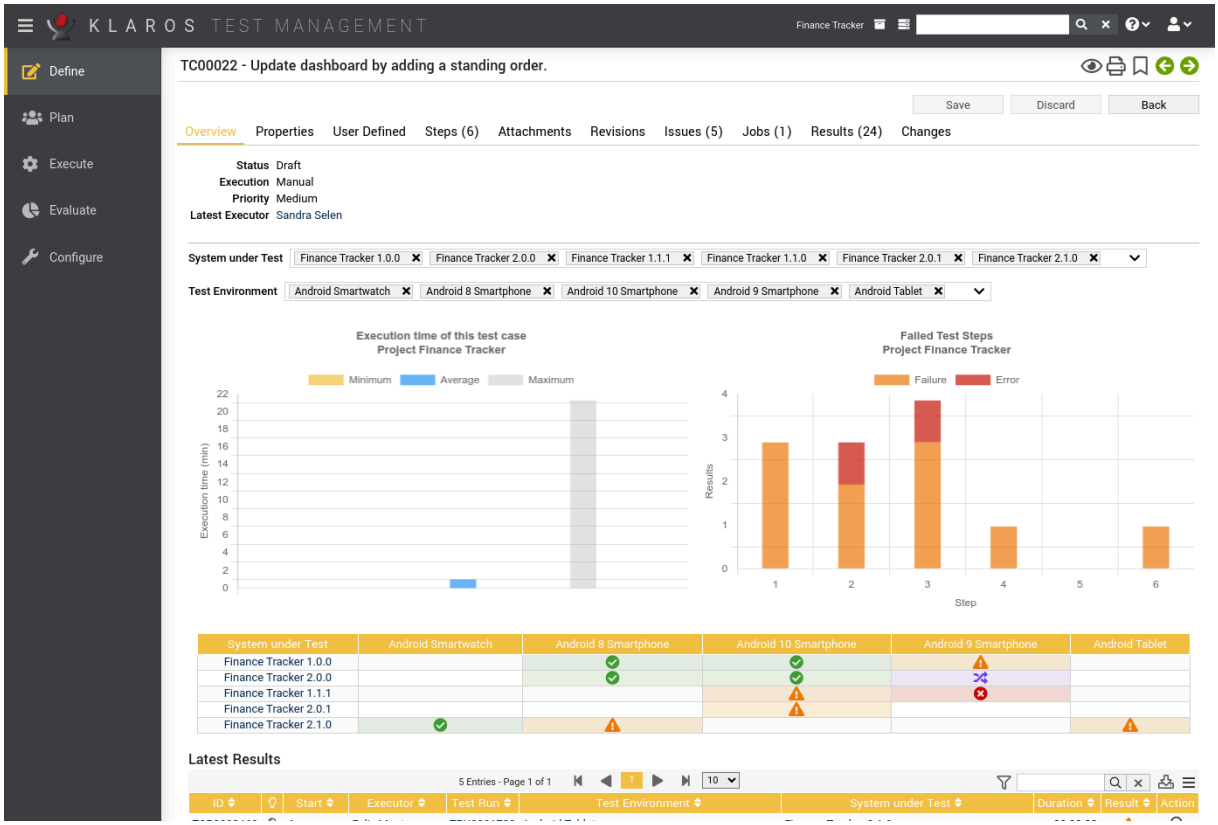


Figure 6.40. The “Overview” Tab

6.7.2.1. Actions

On the detail pages, there are additional icons in the top right corner of the header. The following actions can be performed here:

Review — Creates a review job.

Open print view — A print-ready view of the test case can be created here. With a click on the icon this opens in a new browser tab.

Print views are described in detail in Section 5.2.3.2.1, “Print Pages”.

Create Bookmarks — Each individual detail page can also be reached directly via a hyperlink. By clicking on the icon this link is copied to the clipboard.

The creation of bookmarks is described in detail in Section 5.2.3.2.2, “Bookmarks”.

Browse — Use the green arrows at the very top right to switch between the test environments present on the previous page.

6.7.2.2. Overview

The following values are displayed on the overview tab:

State — The state of the test case. Possible values are *Locked*, *Approved*, *Draft* and *Skip*.

Execution — The type of test case execution: Manual or Automated.

Priority — The priority of the test case: Low, Medium or High.

Latest Executor — The user who last executed this test case.

Execution Times of this Test Case — This graph shows the minimum, maximum and average execution times for this test case. This allows a better estimate of the time required for future executions of this test case.

Most unsuccessful Test Steps — This graph shows all test steps of this test case and which steps have been marked as failed or error during a test run. This graph can be used for determining possible misconfigurations, if errors happen more frequently at one specific step, for example.

System under Test Overview — This table shows the latest result of this test case for every system under test and test environment this test case has been executed in. Pressing the result icon will open up the appertaining test case result.

Latest Results — This table shows the latest results for this test case. This table is limited to five entries. The other results can be viewed by pressing the icon.

Latest Issues — This table shows the latest issues that are linked to this test case. Click on the icon to display the issue directly in the Issue-Management-System

6.7.2.3. Properties

This tab (Figure 6.41) allows the user to view or change the following attributes of the test case:

Editor Settings

Under Define --> Project --> Properties you can set whether the editor displays only text or HTML. With the HTML setting, a bar with formatting options appears when you click in the text field.

ID	Assigned automatically.
Name	The name of the test case.
Description	The textual description of the test case.
Precondition	The precondition for this test case, describing the condition for executing the test case, e.g `It is required that the database is initialized.`.
Expected Result	The visible behavior of the system predicted while carrying out the test step, e.g. `The user receives a message that the login succeeded`.
Postcondition	The environmental and state conditions that must be fulfilled after the execution of the test case, e.g. `The user is authenticated and has access to the system`.
Estimated Duration	The estimated duration required to complete this test case..
Note	An optional comment, such as notes to the tester.
Test Type	A group of test activities based on specific test objectives with the purpose of testing a component or system for specific characteristics (Functional, Non Functional, Structural, Regression, Retest).
Variety	The expected outcome of the test: positive or negative.
Priority	The priority of the test case: Low, Medium or High.
Team	The team that is responsible for the test case.
Docbase	A reference to the document this test case is based on, which may e.g. contain the requirements this test case is related to.
Evaluation	The type of test result evaluation: Manual or Automated.

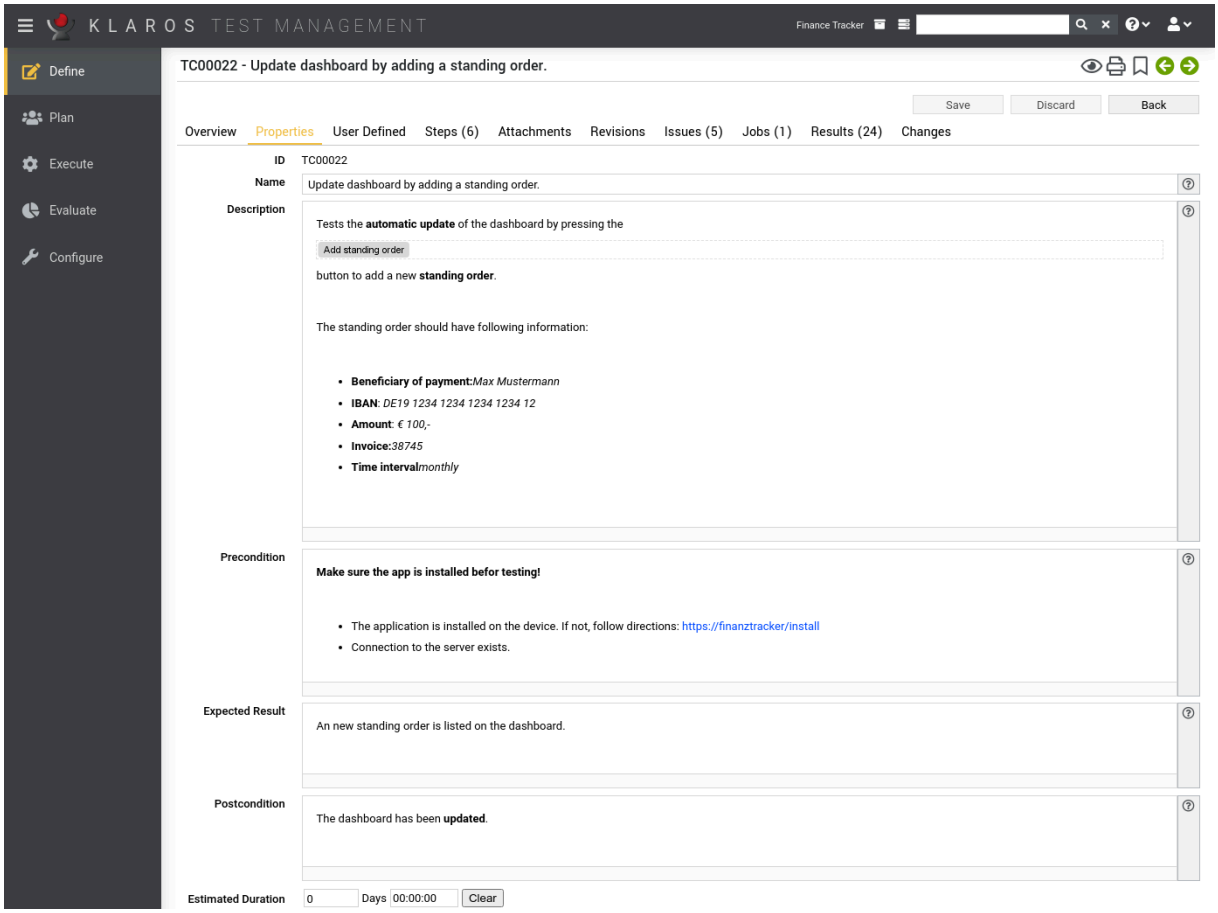


Figure 6.41. The “Properties” Tab

6.7.2.4. User-Defined

You can create your own fields to meet individual requirements. For further information please refer to Section 5.2.3.2.4, “User Defined Properties”.

6.7.2.5. Test Steps

In the *Steps* tab, test steps may be added to or removed from the test case. The following attributes of single test steps may also be changed:

New Creates a new test step.

Insert Test Segment Inserts an existing test segment.

Toggle View Mode Toggle the view mode between a tabbed and a non tabbed display.

Minimizes the display of all steps.

Maximizes the display of all steps.

6.7.2.5.1. Creating a new Test Step

Clicking on the New button opens the step editor for creating a new test step.

Moves the entry to another position.

- Inserts a new step before.
- Inserts a new test segment before.
- Inserts a new test segment after that.
- Inserts a new step after it.
- Duplicates the step.
- Deletes the step.
- and Opens and closes the field view.

As soon as the input field is activated, a toolbar appears in the text field. The texts can be formatted with this as well as be equipped with links and pictures.

Actions The description of the action to take, e.g. `Enter name and password, and click the Login button.`

Expected Result The (visible) behavior of the system predicted while carrying out the test step e.g. `The user receives a message that the login succeeded.`

Precondition Condition that must be fulfilled before the execution of the test step, e.g. `The user is not yet logged in.`

Postcondition The environmental and state conditions that must be fulfilled after the execution of the test case e.g. `The user is authenticated and has access to the system.`

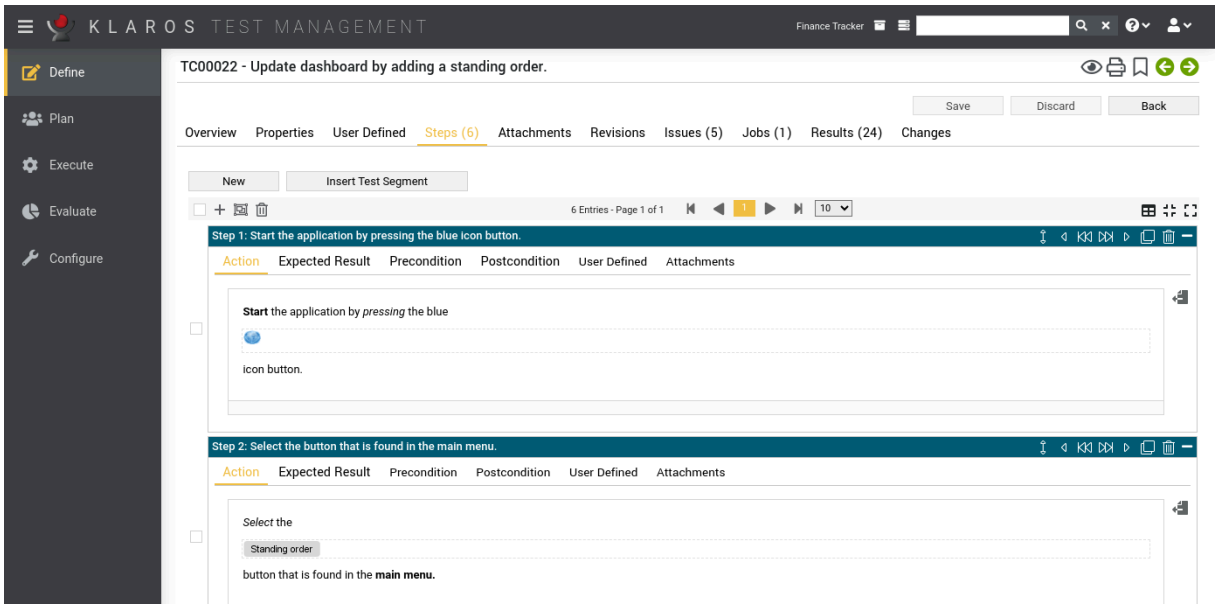


Figure 6.42. The “Steps” Tab

Pressing one of the Insert Attachment buttons next to the *Description*, *Expected Result*, *Precondition* and *Postcondition* text fields opens up a dialog which allows to embed an attachment into the text.

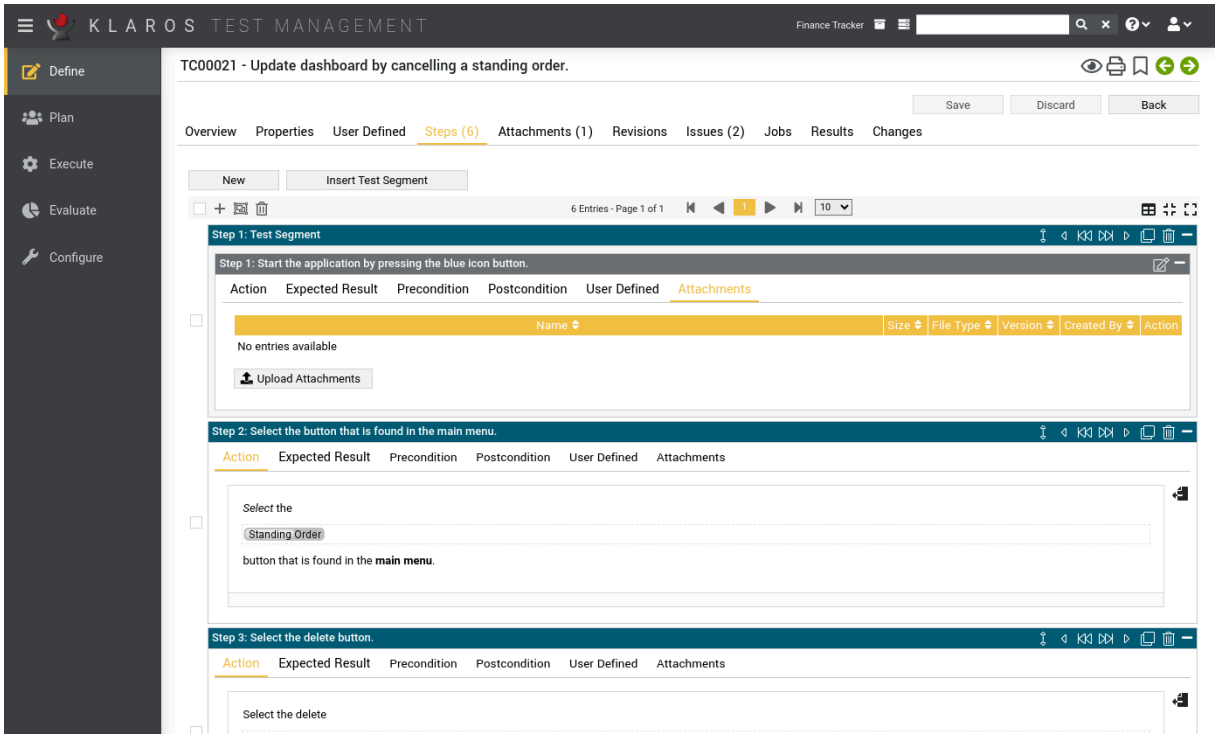


Figure 6.43. The “Insert Attachment” Dialog

Note

The attachments to be selected must have been previously uploaded in the *Attachments* view.

Pressing the Add button, while having an attachment selected, will insert a reference in the form of *%inline-att:[attachment-identifier]%*, e.g. *%inline-att:9e3df155-359f-40e7-8c78-7ae9aac6a8ab%*.

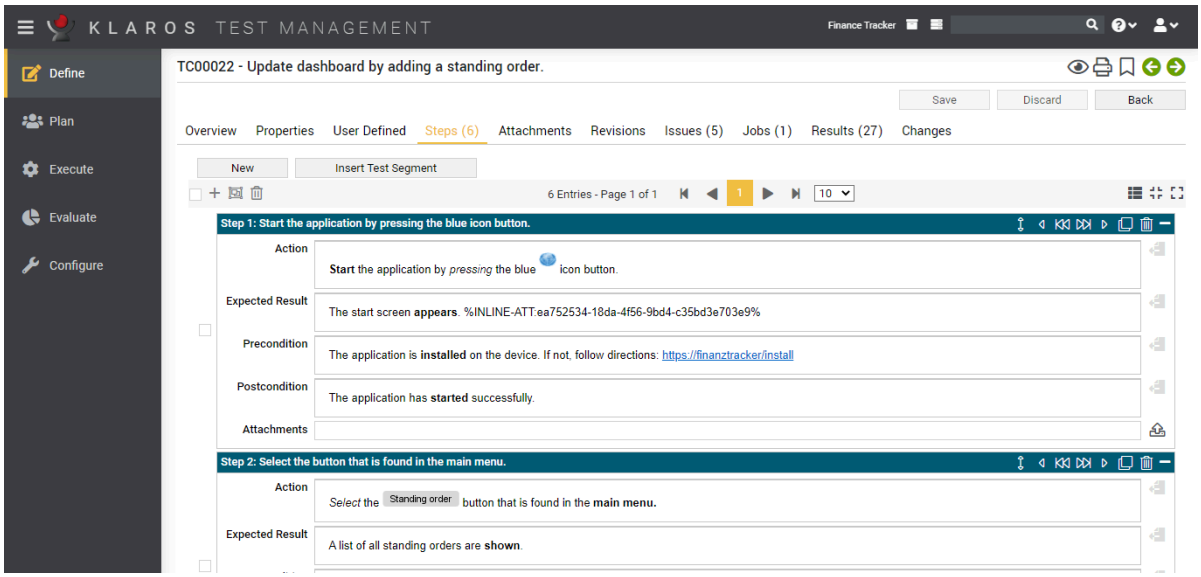


Figure 6.44. A Reference to an Attachment in the Expected Result Field

Depending on whether the attachment is an image, this reference will be replaced by a scalable preview image during the execution of the test case. Otherwise, the reference will be replaced by a hyperlink that will open the attachment in a new browser tab.

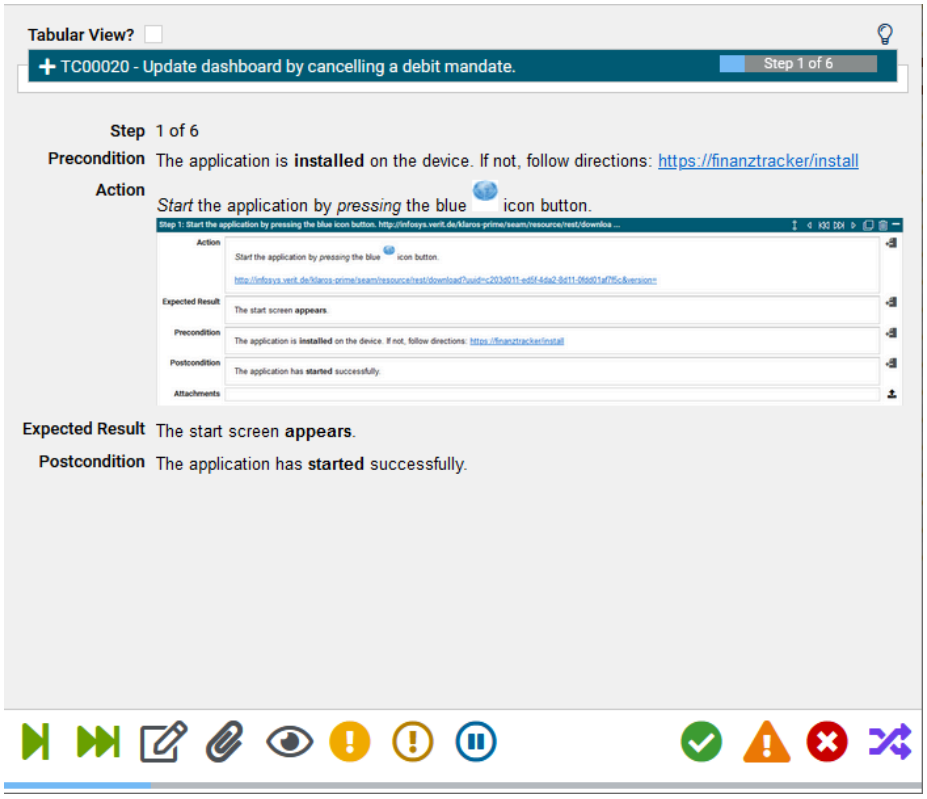


Figure 6.45. An Attachment Reference Replaced by a Preview of the Attachment

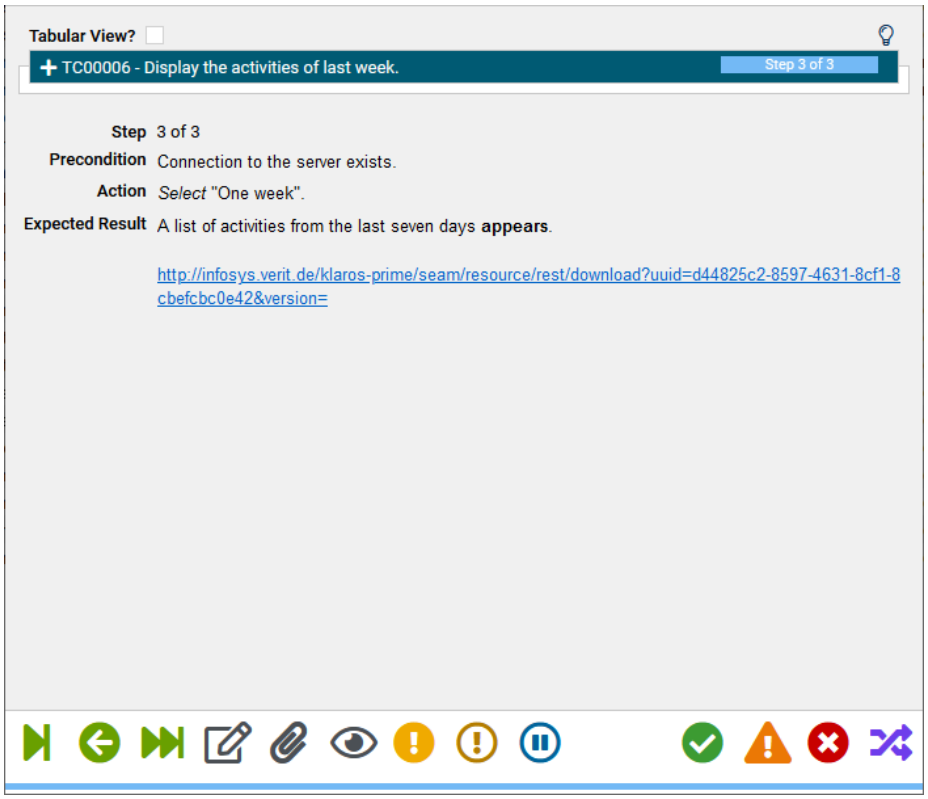


Figure 6.46. An Attachment Reference Replaced by a Hyperlink to the Attachment

6.7.2.6. Attachments

You can add any files as attachments to a test case. For more information, see Section 5.2.3.2.6, "Attachments".

6.7.2.7. Revisions

Test cases may exist in different revisions. For more information, see Section 5.2.3.2.5, "Revisions".

6.7.2.8. Issues

This view lists all issues that are associated with this test case.

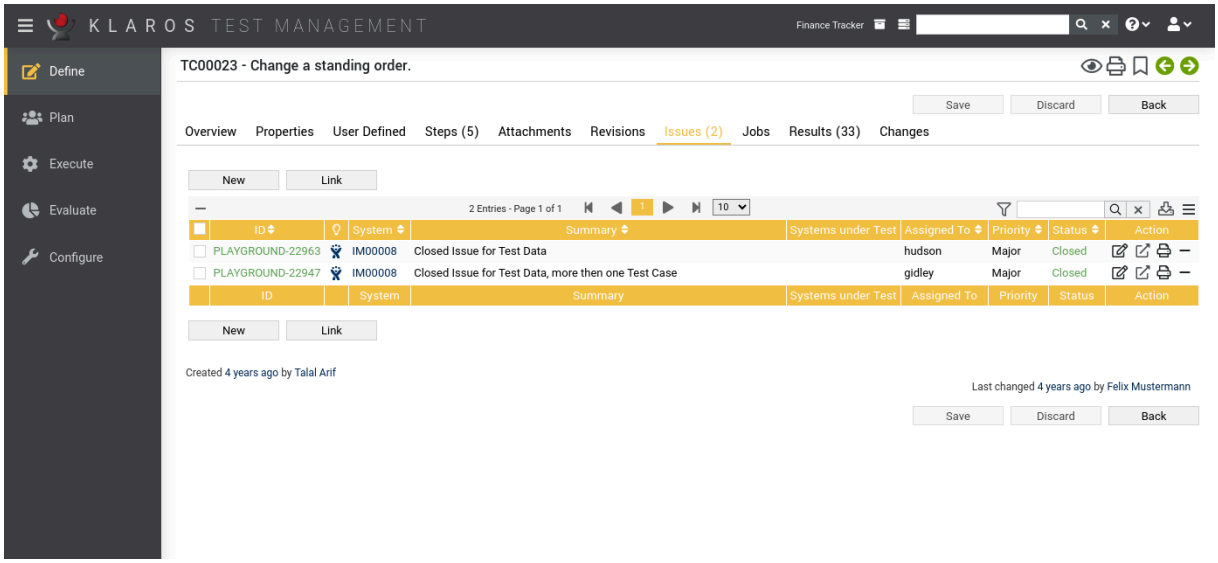


Figure 6.47. The "Issues" Tab

The table shows the following values:

ID	The ID of the issue in the external issue management system.
System	The external issue management system of this issue.
Additional Information	A tooltip appears when the cursor is placed over the icon shown here.
Summary	The summary of the issue.
Created	The date when the issue was created.
Created by	The user who created the issue.
Assigned	The user to whom the issue is assigned.
Priority	The priority of the issue. The possible values are specified by the external issue management system.
Status	The status of the issue. The possible values are specified by the external issue management system.

Actions The executable actions.

6.7.2.8.1. Creating a new Issue

Clicking New opens the *Issue Details* page (Section 9.6.6, “Issue Details (Creating a new Issue)”), where new issues can be created and linked to the test case. Clicking Link links existing issues to the test case (Section 9.6.7, “Link Issue”).

6.7.2.8.2. Action

The following actions can be performed in the action column:

- Edit
- Browse
- Delete

6.7.2.9. Jobs

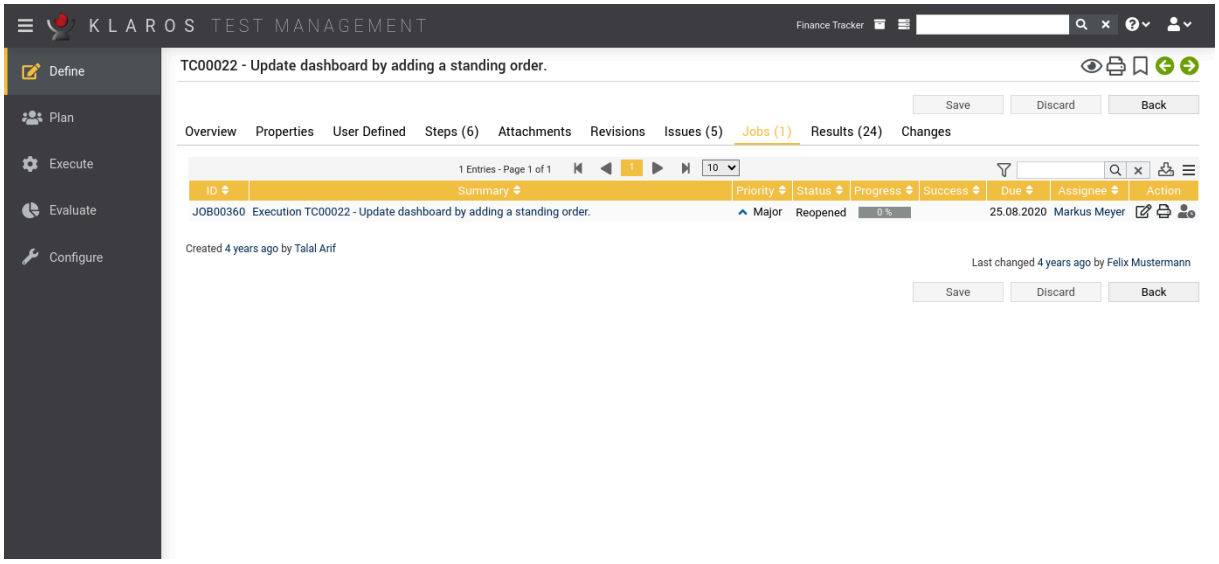


Figure 6.48. The “Jobs” Tab

This view lists all the jobs in which this test case is included. These jobs can be executed directly from this view. For more information on managing jobs, see Section 7.1, “Jobs”.

6.7.2.10. Results

The results tab is further divided into a *Test Runs*, *Test Case Results* and a *Test Suite Results* tab, showing the test results related to this test suite as described in Section 5.2.3.2.7, “Test Runs and Results”.

6.7.2.11. Changes

The tab *Changes* shows the change history of this test case.

For a detailed description of the *Changes* view, see Section 5.2.3.2.8, “Change History”.

6.8. Test Suites

A *test suite* is a collection of test cases that are to be executed together. The results of a test suite execution can be used to verify and ensure that a system meets certain specifications or requirements.

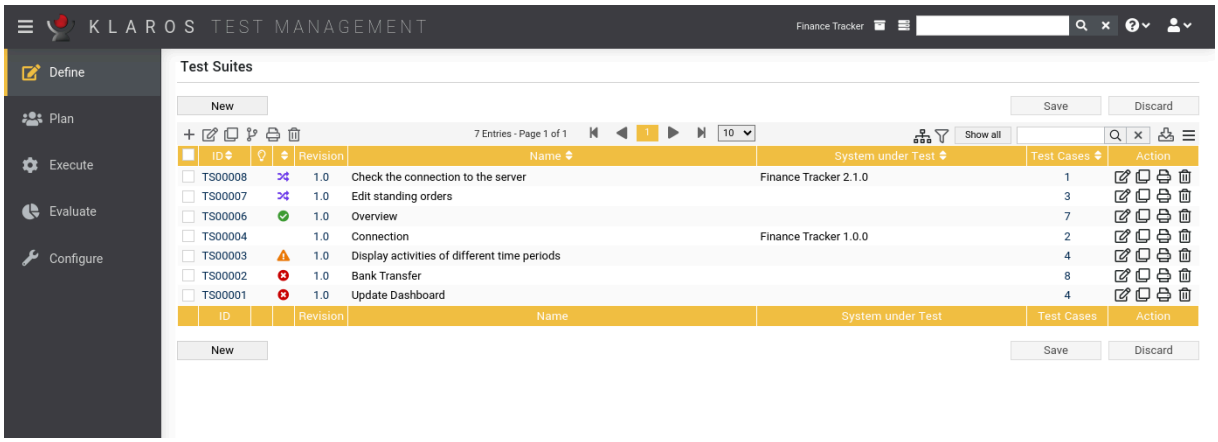


Figure 6.49. The “Test Suites” Page

6.8.1. Overview Page

The overview page displays all existing test suites in the selected project in a table. New test suites are created here.

The table shows the following values:

ID — Assigned automatically.

Additional Information — A tooltip appears when the cursor is placed over the icon shown here.

Revision — The revision of the test suite.

Name — The name of the test suite.

System under Test — The systems under test assigned to this test suite.

Test Cases — The number of test cases that this test suite contains.

Action — The actions that can be performed.

Name and *System under Test* can be edited directly in the table rows.

6.8.1.1. Creating a new Test Suite

By clicking on the button New a new empty table row will appear. Now, *Name* and *System under Test* can be defined.

With Save the new test suite is created and saved. The ID of the test suite (TS00001) is automatically assigned by Klaros-Testmanagement. Click on the ID *TS00001* to get to the detail page of the test suite.

With New several table rows can be created and edited at the same time. Only when clicking on Save the data is stored in the database.

With Discard all changes are undone.

Unsaved Changes

All rows marked with red IDs contain changes that have not yet been saved.

6.8.1.2. Actions

The following actions can be performed in the action column:

- Edit
- Duplicate
- Open print view
- Delete

If a test suite has been deleted, it is initially marked with a deletion marker and is only visible to administrators. For deleted test suites, the following actions are available instead of *Delete*:

- Restore (only Administrator)
- Irrecoverably remove the test suite from the database

6.8.1.3. Bulk Actions

On the *Test Suite* page, one or more test suites can be selected for bulk actions. Bulk actions are described in Section 5.2.3.1.5, “Bulk Actions”.

The following bulk actions are supported for test suites:

- Assign Test Cases
- Edit
- Duplicate
- Create new Revision
- Open print view
- Delete
- Restore (only Administrator)
- Irrecoverably remove the test suite from the database (only Administrator).

For additional information, see Section 5.2.3.1.5, “Bulk Actions”.

6.8.1.4. Table Operations

The following operations can be performed in the line above the table on the right:

- Categorize
- Filter / Sort
- Show all / Only active (only Administrator)
- Search
- Export
- Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

6.8.2. Detail Page

Each test suite has its own detail page with several additional tabs. Clicking on the ID of the relevant execution definition or on the icon on the right in the action column takes you to the tab that was selected last. When called for the first time, this is the *Overview* tab.

The following tabs are available: *Overview*, *Properties*, *User Defined*, *Attachments*, *Revisions*, *Jobs*, *Results* and *Changes*.

Warning Sign in the ID Column

A manual test case which that contains no steps is not executable. To indicate this, a warning sign (⚠) is displayed in the ID column.

6.8.2.1. Actions

On the detail pages there are additional icons in the upper right corner. The following actions can be performed here:

Review	Creates a review job.
Open print view	Here, a print-ready view of all entries of the respective test suite can be created. Click on the icon to open it in a new browser tab. For more information, see Section 5.2.3.2.1, “Print Pages”.
Create Bookmarks	Each individual detail page can also be reached directly via a hyperlink. By clicking on the icon this link is copied to the clipboard. The creation of bookmarks is described in detail in Section 5.2.3.2.2, “Bookmarks”.

Browse Use the green arrows at the very top right to switch between the test suites present on the previous page.

6.8.2.2. Overview

The following values are displayed on the overview tab:

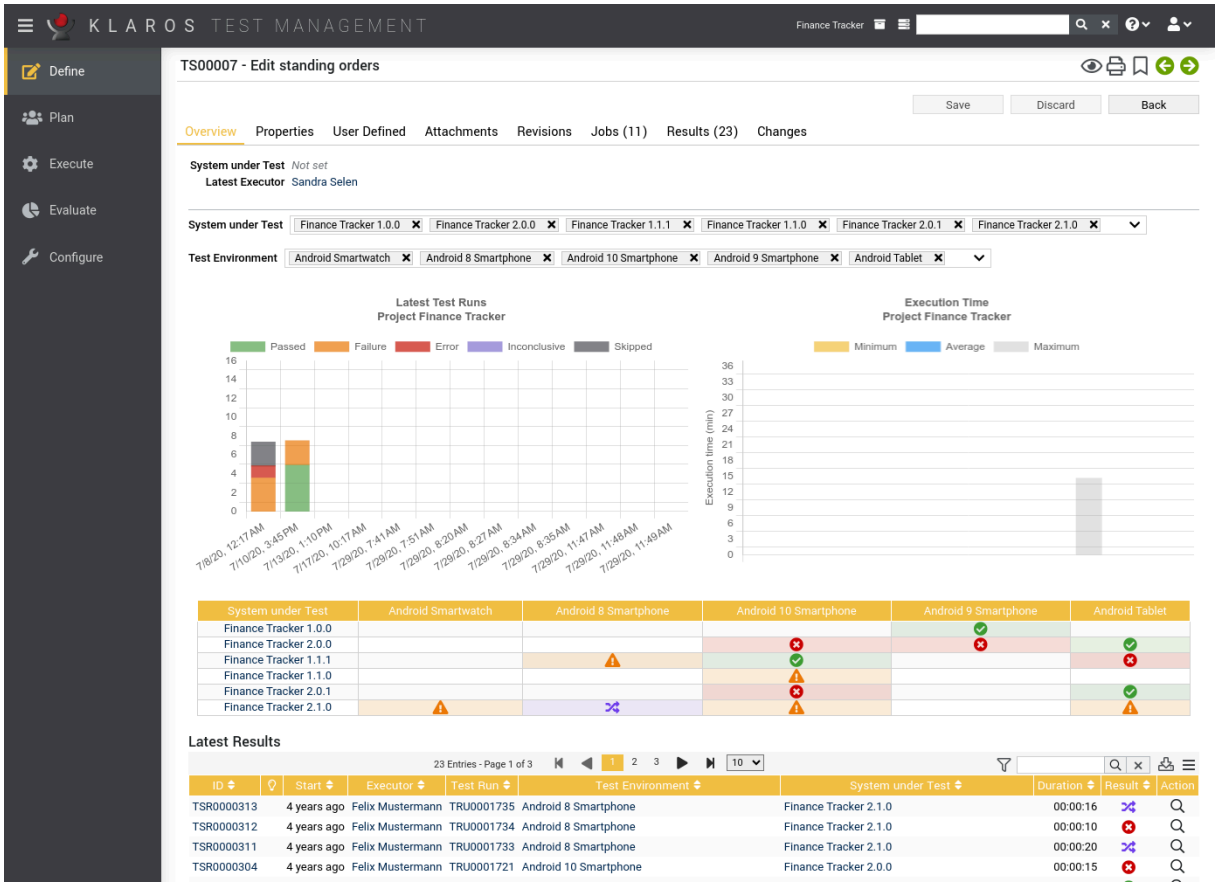


Figure 6.50. The “Overview” Tab - Test Suite

Assigning a System Under Test

The assignment of a test suite to a system under test only indicates this to the user. Assigned test-suites can still be executed with other systems under test.

Latest test runs This graph shows the number of passed, failed, erroneous, inconclusive and skipped test case results for the latest five test runs for this test suite.

Execution Times of this Test Suite This graph shows the minimum, maximum and average execution times for this test suite.

System under Test Overview This table shows the latest result of this test suite for every system under test and test environment this test suite has been executed in. Hovering the mouse cursor over a single result will show a more detailed overview of the individual test

case results of this test suite. Clicking on a result takes you to the test case result detail page.

6.8.2.3. Properties

In this tab (Figure 6.22) the following attributes of the test suite are displayed:

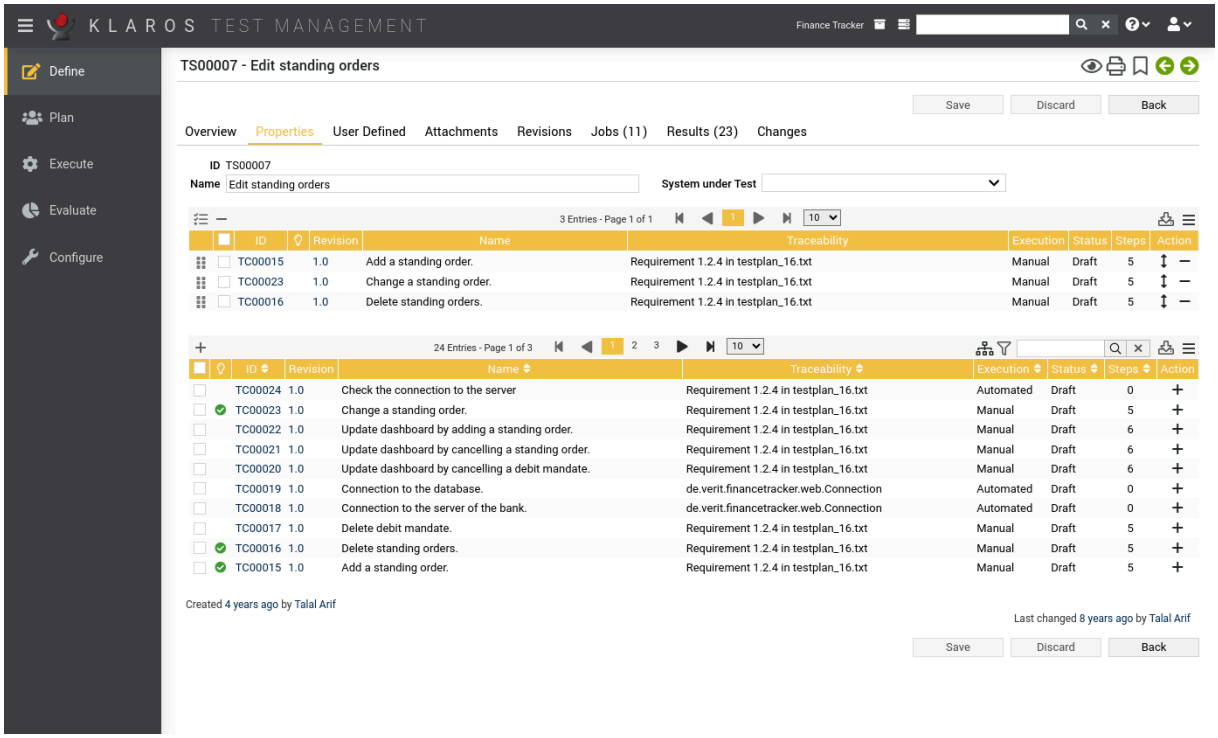


Figure 6.51. The “Properties” Tab

In this view, test cases can be added to or removed from the test suite. The upper table displays the test cases contained in the test suite, it supports reordering via drag and drop operations. The lower table shows the available test cases.

The following bulk actions are supported for the list of test cases in the test-suite:

- Remove multiple occurrences
- Remove test cases

This tab allows the user to view following attributes of the test cases:

ID — Assigned automatically.

Name — The name of the test case.

Traceability — Die Anforderung, die dieser Testsuite zugeordnet ist.

Execution — Manual oder Automated.

State — The state of the test case. Possible values are *Locked*, *Approved*, *Draft* and *Skip*.

Steps	The number of associated test steps.
Action	Entfernt oder fügt den Testfall zur Testsuite hinzu.

6.8.2.4. User-Defined

You can create your own fields to meet individual requirements. For further information please refer to Section 5.2.3.2.4, “User Defined Properties”.

6.8.2.5. Attachments

You can add any files as attachments to a test suite. For more information, see Section 5.2.3.2.6, “Attachments”.

6.8.2.6. Revisions

Test suites may exist in different revisions. For more information, see Section 5.2.3.2.5, “Revisions”.

6.8.2.7. Jobs

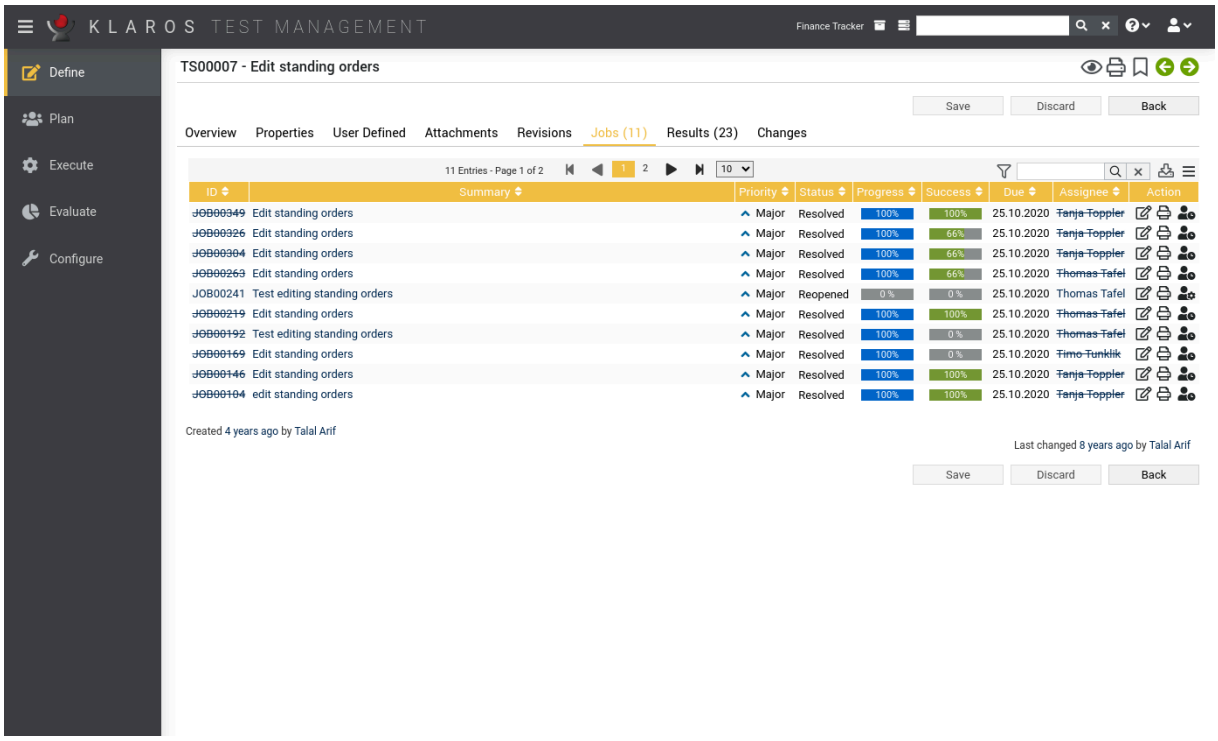


Figure 6.52. Jobs

This view lists all the jobs in which this test suite is included. These jobs can be executed directly from this view. For more information on managing jobs, see Section 7.1, “Jobs”.

6.8.2.8. Results

The results tab is further divided into a *Test Runs*, *Test Case Results* and a *Test Suite Results* tab, showing the test results related to this test suite as described in Section 5.2.3.2.7, “Test Runs and Results”.

6.8.2.9. Changes

The tab *Changes* shows the change history of this test suite.

For a detailed description of the *Changes* view, see Section 5.2.3.2.8, “Change History”.

Chapter 7. Plan

Only available in Klaros-Testmanagement Enterprise Edition

This chapter describes the *Plan* section in Klaros-Testmanagement. Here Jobs can be created and edited.

7.1. Maintain Jobs

A *job* is used to plan, organize and track a test activity. A job may consist of the execution or review of a test case or test suite or contain the textual description of any other possible activity. This chapter shows how to create and edit jobs.

7.1.1. Overview Page

The overview page displays all the jobs present in the selected project in a table. New jobs are created here.

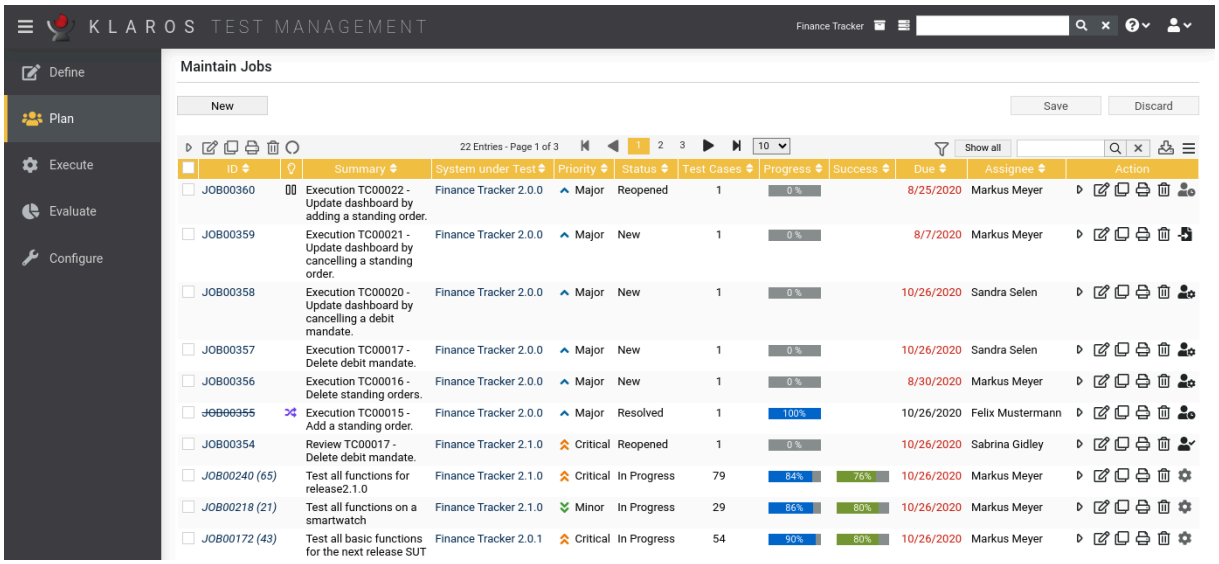


Figure 7.1. The “Maintain Jobs” Page

The table shows the following values:

ID — Assigned automatically.

Summary — The summary of the job.

System under Test — The test system assigned to this job.

Priority — The priority of the job. Possible values are *Trivial*, *Low*, *High-Critical* und *Blocker*.

Status — The status of the job. Possible values are *New*, *Reopened*, *In Progress*, *Resolved*, *Closed* and *Rejected*.

Test Cases — The number of test cases assigned to this job and its sub-jobs.

Progress	The percentage of executed test cases of this job and its sub-jobs.
Success	The success rate of the executed test cases of this job and its sub-jobs in percent.
Due	The date on which this job is due to be finished.
Assigned	The user to whom the kob is assigned.
Action	The actions that can be performed.

7.1.1.1. Creating a new Job

By clicking on the button `New` a new empty table row will appear. Now, a *Summary* can be defined.

With `Save` the new job is created and saved. The ID of the job (JOB00001) is automatically assigned by Klaros-Testmanagement. Click on the ID *JOB00001* to get to the detail page of the job.

With `New` several table rows can be created and edited at the same time. Only when clicking on `Save` the data is stored in the database.

With `Discard` all changes are undone.

Red IDs

All rows with red IDs have been changed and are not yet saved!

7.1.1.2. Actions

The action column is located on the far right of the table. The following actions can be performed here:

- Subordinate job
- Edit, (see Section 7.1.2, “Job - Details”)
- Clone
- Open print view
- Delete
- Execute, Execute manually, Execute Review, Import Test Results, Reopen or execute (see Section 7.1.1.6, “Executing Jobs”)

If a job has been deleted, it is initially marked with a deletion marker and is only visible to administrators. For deleted jobs, the following actions are available instead of *Delete*:

- Restore (only Administrator)
- Irrecoverably remove the job from the database

7.1.1.3. Bulk Actions

Some actions can be applied to multiple jobs at the same time. To do this, select the jobs to which the action is to be applied in the leftmost column.

The following bulk actions are supported for jobs:

- Subordinate jobs to another job
- Edit
- Clone
- Open print view
- Delete
- Restore (only Administrator).
- Irrecoverably remove the job from the database (only Administrator).
- Assign jobs to an iteration.

Bulk actions are described in detail in Section 5.2.3.1.5, “Bulk Actions”

7.1.1.4. Table operations

The following operations can be performed in the line above the table on the right:

- Filter / Sort
- Search
- Export
- Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

7.1.1.5. Arranging Sub-Jobs

Sub jobs can be rearranged using the arrow icons in the *Action* column. The ↕ icon opens a dialog that allows the user to move the job to a new position. The ◂ and ▸ icons change the job hierarchy. The ◂ icon moves the job to the hierarchy level of the parent job. The ▸ icon opens a dialog where a new parent job can be selected.

Parent jobs have the following characteristics:

- A parent job cannot be assigned any test case or test suite.
- A parent job can contain other parent jobs with additional sub jobs.
- If the ▸ icon of a job is grayed out, no target parent job is available for subordination.

To subordinate a job to another job, click on the ▸ icon. Now you can select which job this job should be subordinated to. The number of subordinate jobs is then displayed next to the ID of the

parent job. The displayed success and progress rate of the parent job is automatically derived from the test results of its sub jobs.

Warning

Jobs that already have a test case or test suite assigned to them cannot contain sub jobs and are therefore not shown as possible targets in the *New parent job* dialog.

Changes to parent jobs

Changes to parent jobs regarding test environment, system under test and iteration are automatically propagated to their sub jobs.

7.1.1.6. Executing Jobs

For a detailed description on how to execute jobs, see Section 8.1.3, “Executing a Job”.

7.1.2. Job - Details

Each job has its own detail page with several additional tabs. By clicking on the ID of the relevant job or on the icon on the right in the action column, you can access the detail view that was selected last. When called for the first time, this is the *Overview* view.

The following tabs are available: *Overview*, *Properties*, *User Defined*, *Dependencies Attachments*, *Results*, *Comments*, *Work Log* and *Changes*.

7.1.2.1. Actions

On the detail pages there are additional icons in the upper right corner. The following actions can be performed here:

Execute — Click on the displayed icon to execute the job.

The execution of jobs is described in detail in Section 7.1.1.6, “Executing Jobs”).

Open print view — A print-ready view of the job can be created here. With a click on the icon this opens in a new browser tab.

Print views are described in detail in Section 5.2.3.2.1, “Print Pages”.

Create Bookmarks — Each individual detail page can also be reached directly via a hyperlink. By clicking on the icon this link is copied to the clipboard.

The creation of bookmarks is described in detail in Section 5.2.3.2.2, “Bookmarks”.

Browse — Use the green arrows at the very top right to switch between the jobs present on the previous page.

7.1.2.2. Job Overview

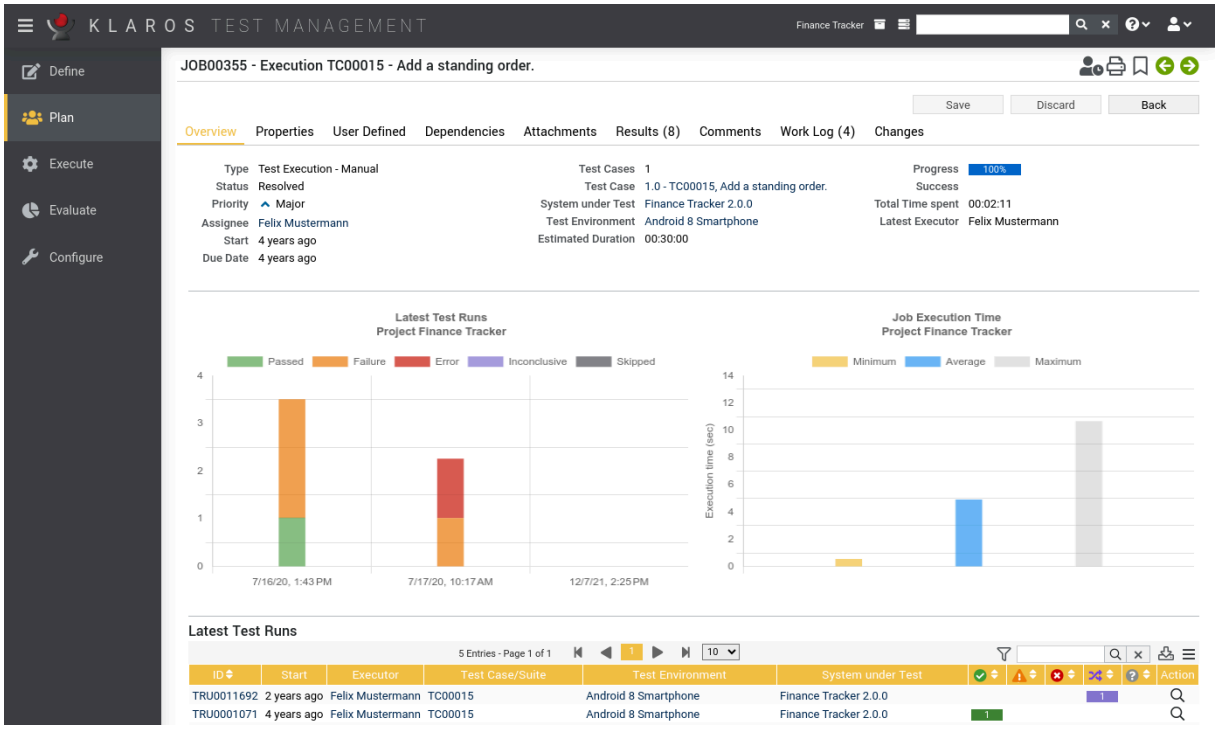


Figure 7.2. The “Overview” Tab

Type — The job type (*Test Execution - Automated*, *Test Execution - Manual*, *Review* or *Task*(if neither a test case nor a test suite is assigned to the job).

Status — The status of the job.

Priority — The priority of the job.

Assignee — The person responsible for this job.

Start Date — The earliest point in time this job can be executed by a tester.

Due Date — The point in time when the execution of the job has to be completed at the latest.

Test Cases — The total number of test cases of this job or its subjobs.

Test Case / Test Suite — The test case or test-suite assigned to this job.

System under Test — The system under test which should be used when executing this job.

Test Environment — The test environment which should be used when executing this job.

Estimated Duration — The estimated duration needed to complete this job.

Progress — The progress rate shows how many test cases which are covered by this job have been executed, regardless the result.

Success The success rate shows how many of the test cases which are covered by this job have been successfully executed in the latest test run.

Total Time spent The time spent on the test execution. This value results from the sum of the execution times from the work log.

Latest Executor The user who executed the job most recently.

Latest Test Runs This graph displays the accumulated test case results of the latest test runs for this job in descending order.

Execution Time This graph shows the minimum, average and maximum execution times for this job.

Latest Test Runs This table shows the latest completed test runs for this job.

Note

By default, this table shows only up to five entries. Pressing the *Show all* button opens up the results tab in which all test runs are listed.

7.1.2.3. Properties

In the *Properties* tab, the following attributes can be changed:

Type The job type (*Test Execution - Automated*, *Test Execution - Manual*, *Review* or *Task*(if neither a test case nor a test suite is assigned to the job).

Summary The summary of the job.

Description The detailed description of the job.

Priority The priority of the job. Possible values are, in ascending order of severity, *Trivial*, *Minor*, *Major*, *Critical* and *Blocker*.

Status The status of the job. Possible values are *New*, *In Progress*, *Resolved*, *Closed*, *Reopened* and *Rejected*.

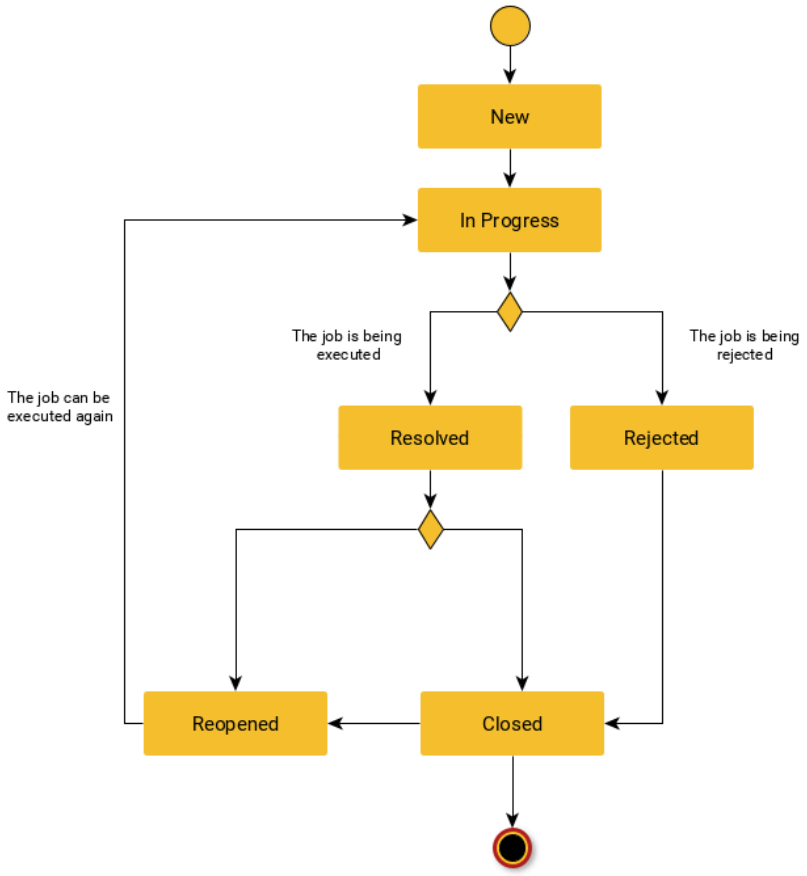


Figure 7.3. Status Graph of a Job

Progress — The current percentage of completion of the job.

Estimated Duration — The estimated time in days, hours, minutes and seconds (Format: Days HH:MM:SS) that the job will take.

Start Date — The earliest time at which this job can be performed by a tester.

Due Date — The time at which the execution of the job must be completed at the latest.

Assignee — The user responsible for the job.

Iteration — The iteration to which this job is assigned. A job can only be in one iteration at a time. Changing the iteration is possible at any time.

Test Case — The test case to be executed in this job. A job can contain either a test case or a test-suite. This field is not available if the job contains subjobs or is of the type *Task*.

Test Suite — The test suite to be executed with this job. A job can contain either a test case or a test suite. This field is not available if the job contains subjobs or is of type *Task*.

System under Test — The system under test for which this job should be executed.

Test Environment The test environment in which this job should be executed.

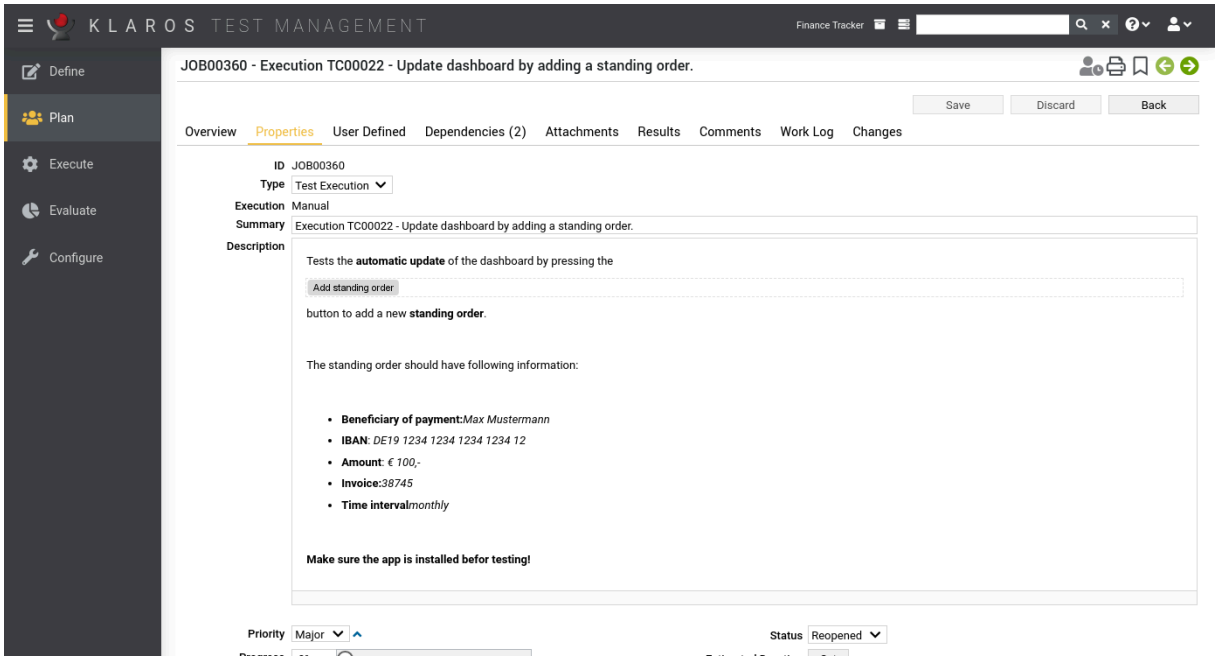


Figure 7.4. The “Properties” Tab

A test case or test suite can be assigned to a job, provided that the job does not contain any subjobs. A click on the icon opens a dialog in which a test case or test suite can be selected. Only one item can be selected at the same time. Clicking the icon removes the element from the job.

7.1.2.4. User Defined

You can create your own fields to meet individual requirements. For further information please refer to Section 5.2.3.2.4, “User Defined Properties”.

7.1.2.5. Dependencies

In the *Dependencies* view, dependencies can be defined between jobs so that certain jobs can only be executed when previously defined conditions are met. For each job, any number of dependencies can be defined to one or more other jobs that must be fulfilled before the job can be executed. In this view you will find an overview of all prerequisite and dependent jobs of the currently selected jobs.

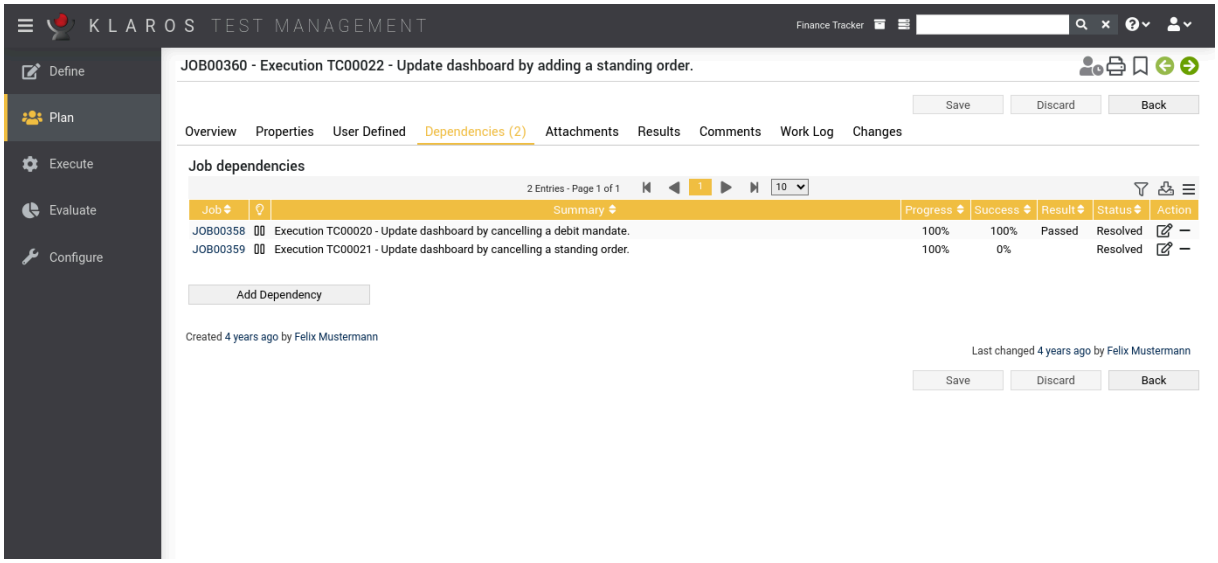


Figure 7.5. The “Dependencies” Tab

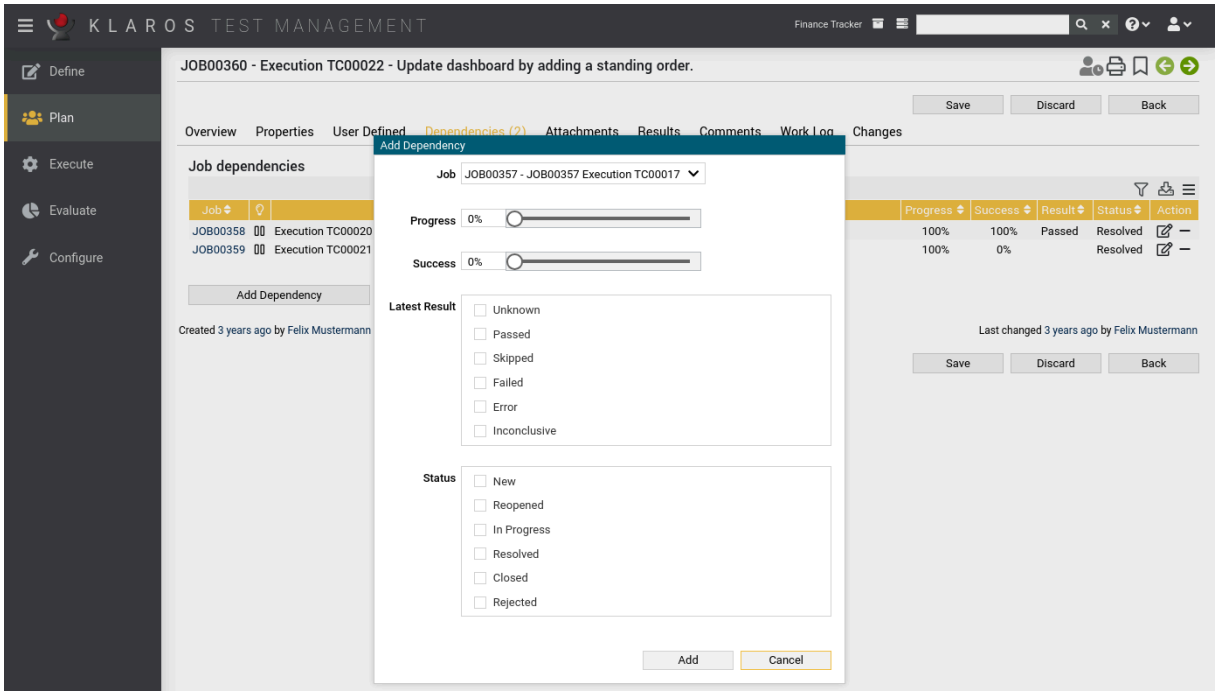


Figure 7.6. The “Add Dependency” Dialog

Dependency Criteria

Job	The prerequisite job.
Progress	The minimum progress rate of the prerequisite job (0-100%).
Success	The minimum success rate of the prerequisite job (0-100%).
Latest Result	The latest result of the prerequisite job (*Passed*, *Failed*, *Error*, *Inconclusive*, *Skipped* or *Unknown*). Multiple values can be selected.

Status The status of the prerequisite job. Possible values are *New*, *In Progress*, *Resolved*, *Closed*, *Reopened* and *Rejected*.

If a job cannot be executed because the dependency criteria of its prerequisite jobs are not fulfilled, the icon is displayed instead of the execute icon.

7.1.2.6. Attachments

You can add any files as attachments to a job. For more information, see Section 5.2.3.2.6, "Attachments".

7.1.2.7. Results

The results tab is further divided into a *Test Runs*, *Test Case Results* and a *Test Suite Results* tab, showing the test results related to this job as described in Section 5.2.3.2.7, "Test Runs and Results".

7.1.2.8. Comments

After clicking the Comment button, comments can be added here.

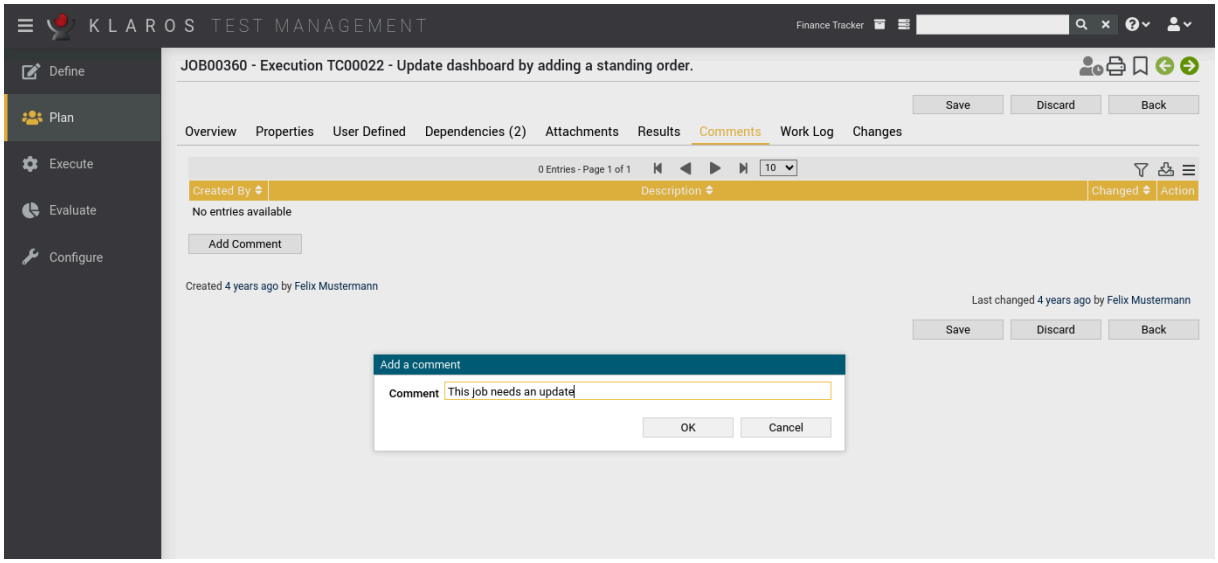


Figure 7.7. The "Add a Comment" Dialog

The text entered in this dialog will be saved as a comment on the job once the Add Comment button is clicked and saved. To modify the comment, use the icon, to remove it, click the icon.

7.1.2.9. Work Log

Each test run related to this job is displayed in the *Work log* tab. It is also possible to add your own additional work logs to the job. To do this, click on the Book time button. A dialog for entering the details will then be displayed.

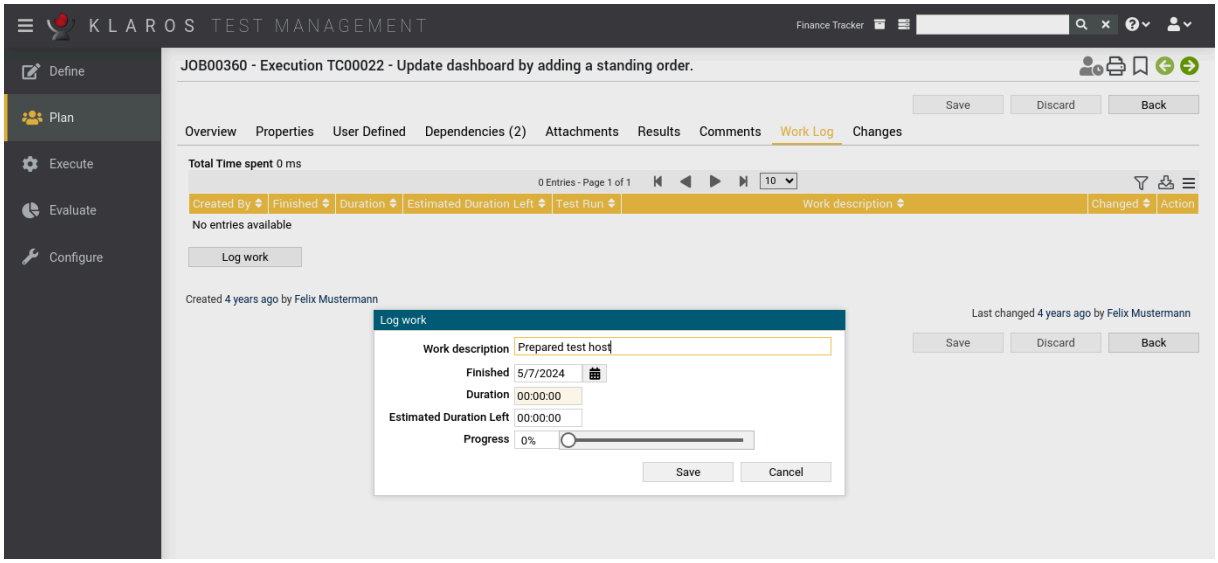


Figure 7.8. The “Work Log” Dialog

A duration must be entered in the *Duration* field in order for the activity to be logged. Then click Save to save your logged work. To modify a work log entry, use the icon to remove it, click the icon.

7.1.2.10. Changes

The page *Changes* shows the change history for this job.

For a detailed description of the *Changes* view, see Section 5.2.3.2.8, “Change History”.

7.2. Jobs from Test Cases

On this page, jobs can be created to run or review selected test cases.

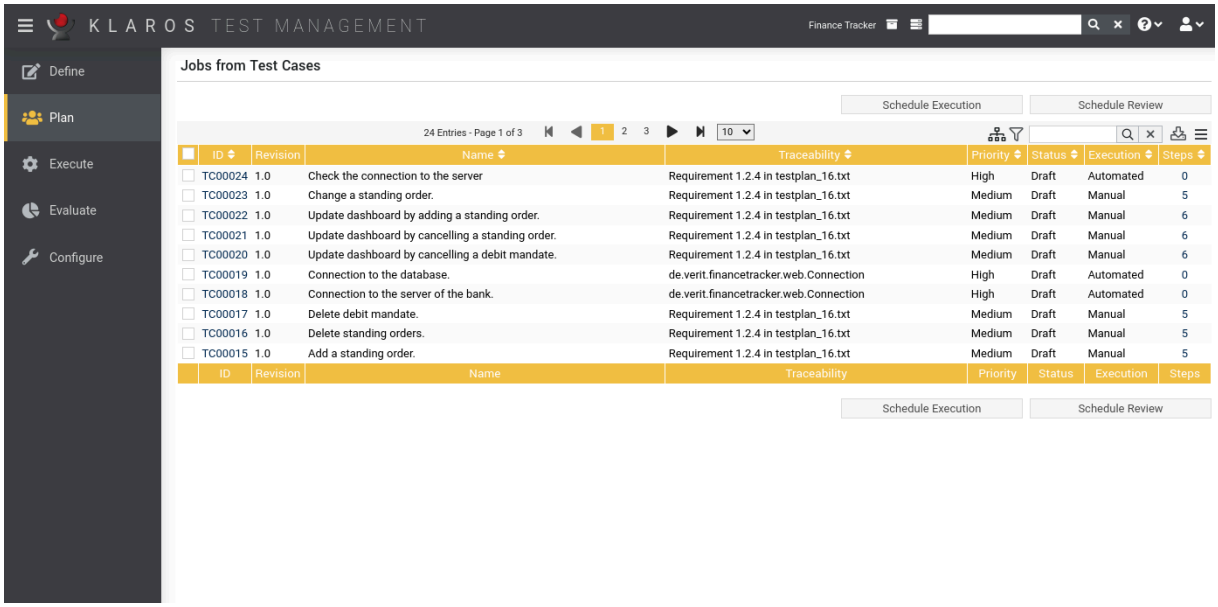


Figure 7.9. The “Jobs from Test Cases” Page

The jobs for the selected test cases can be created by clicking the Schedule execution or Schedule review button. A dialog will open where a description can be specified. In addition, it can be selected to whom this job is assigned and information about the priority, the test system to be used, the test environment, and the start and due dates.

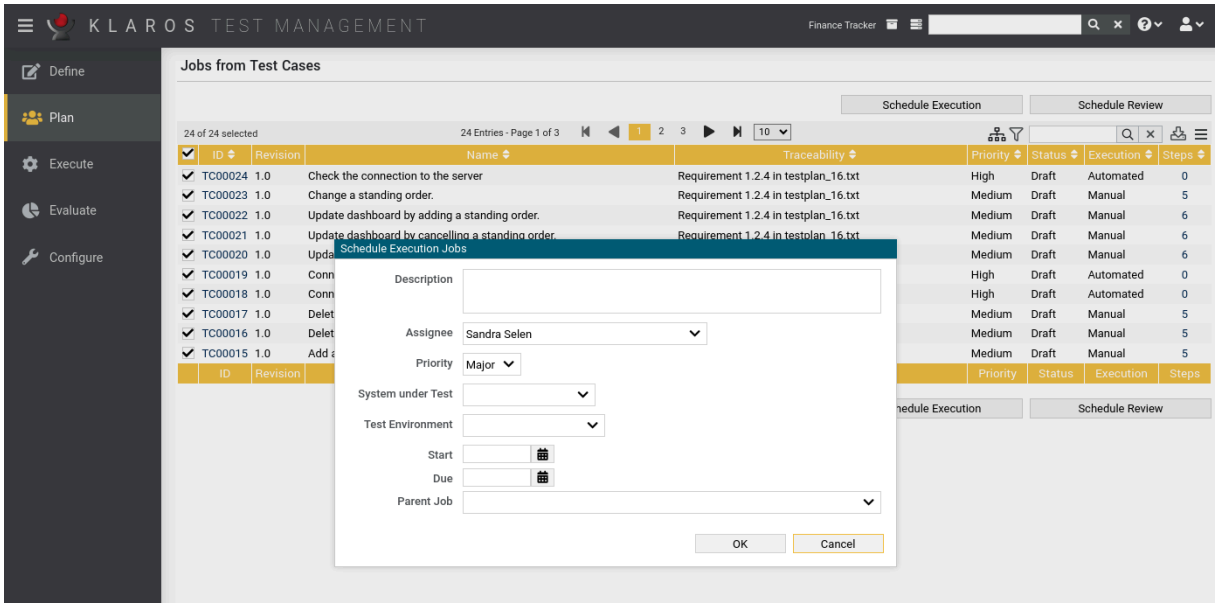


Figure 7.10. The “Schedule Execution Jobs” Dialog

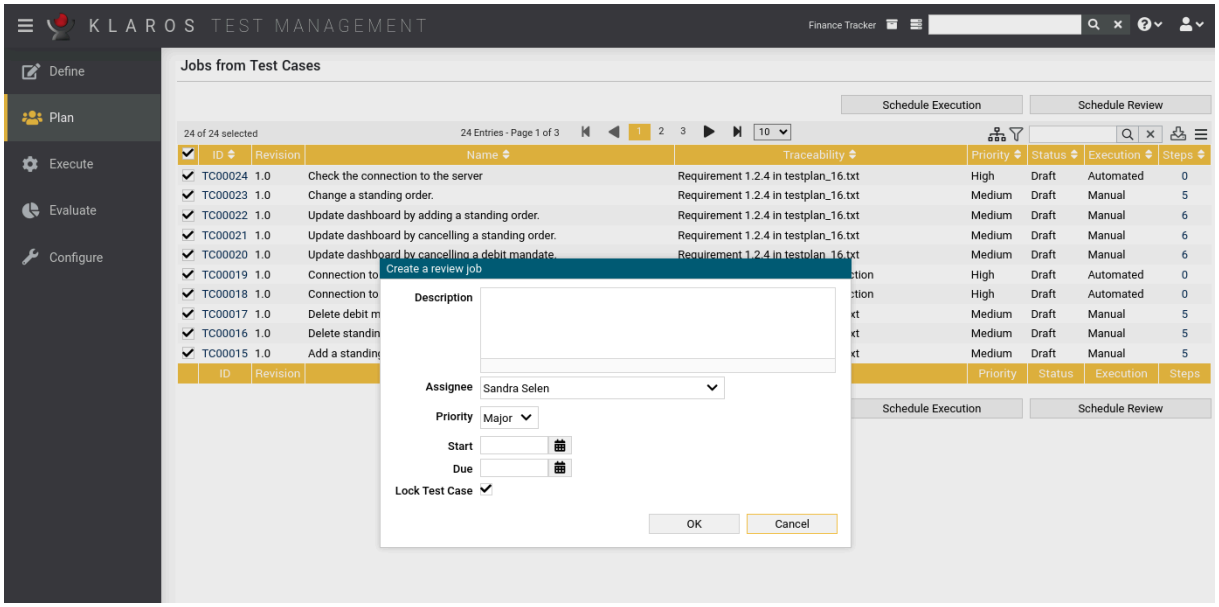


Figure 7.11. The “Create a review job” Dialog

If the jobs are to be subordinated to another job, it can be selected here using the drop-down menu.

The newly created jobs appear on the *Manage Jobs* and *My Jobs* pages and can be executed there.

7.3. Jobs from Test Suites

On this page, the user can generate jobs to run or review selected test suites.

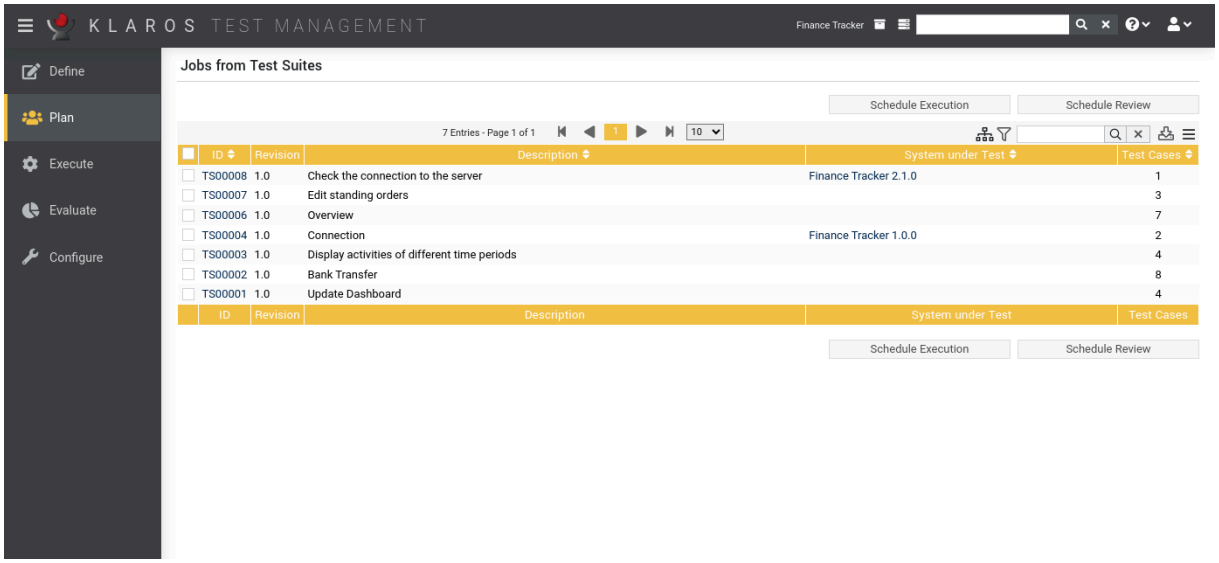


Figure 7.12. The “Jobs from Test Suites” Page

The jobs for the selected test suites can be created by clicking the Schedule execution or Schedule review button. A dialog will open where a description can be specified. In addition, it can be selected to whom this job is assigned and information about the priority, the system under test to be used, the test environment, and the start and due dates.

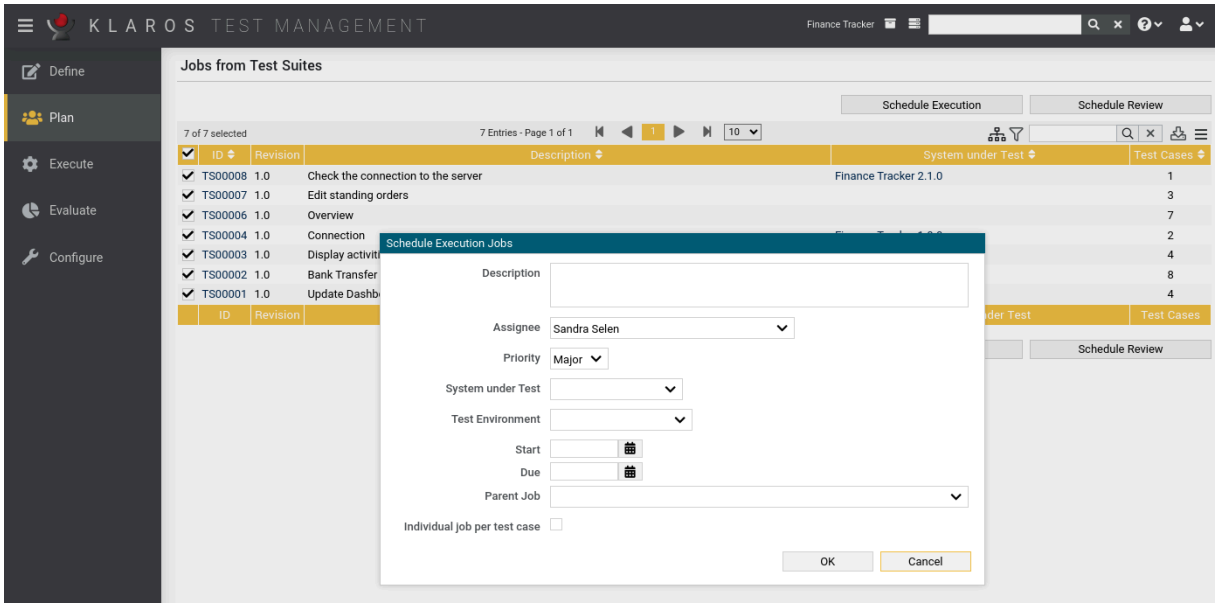


Figure 7.13. The “Schedule Execution Jobs” Dialog

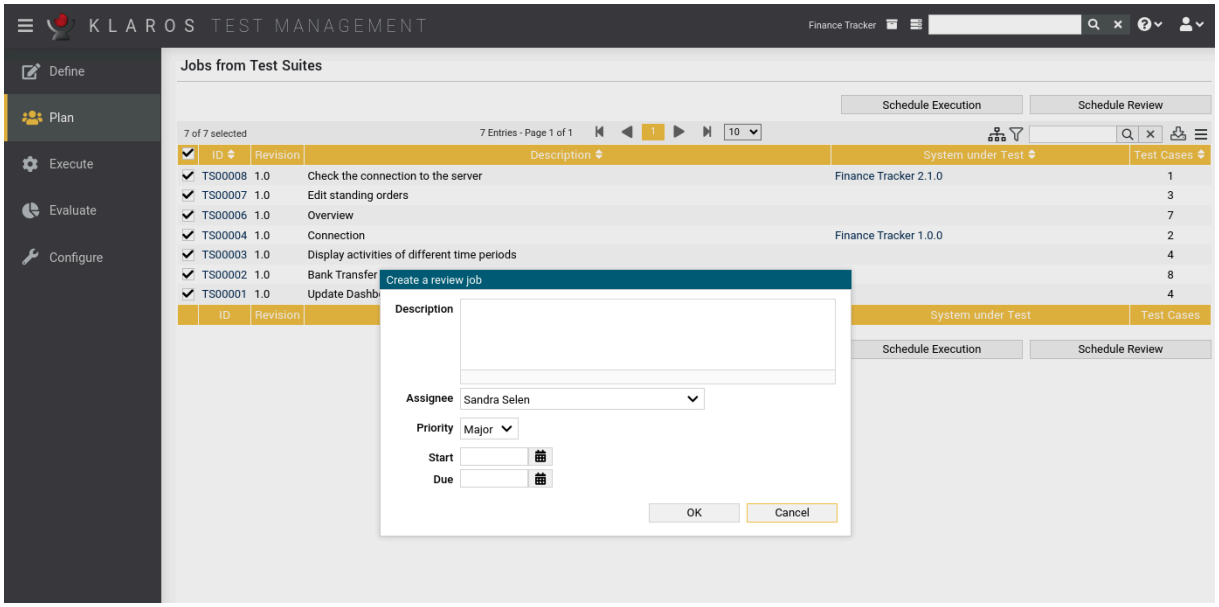


Figure 7.14. The “Create a review job” Dialog

The *Individual job per test case* selection can be used to specify, whether a separate job should be created for each test case in the test suite. These are then combined under a parent job. Splitting them into individual jobs makes it possible to distribute the execution of the test cases individually among several users.

If the jobs are to be subordinated to another job, this can be selected here using the drop-down menu.

The newly created jobs appear on the *Manage Jobs* and *My Jobs* pages and can be executed there.

7.4. Jobs by User

This section is used to track the workload and progress of individual users in the selected project.

On the *Jobs by User* page the workload, work duration and progress of every user for the current active project is shown. Clicking the 🔍 icon displays the details page (Figure 7.16) for the selected user.

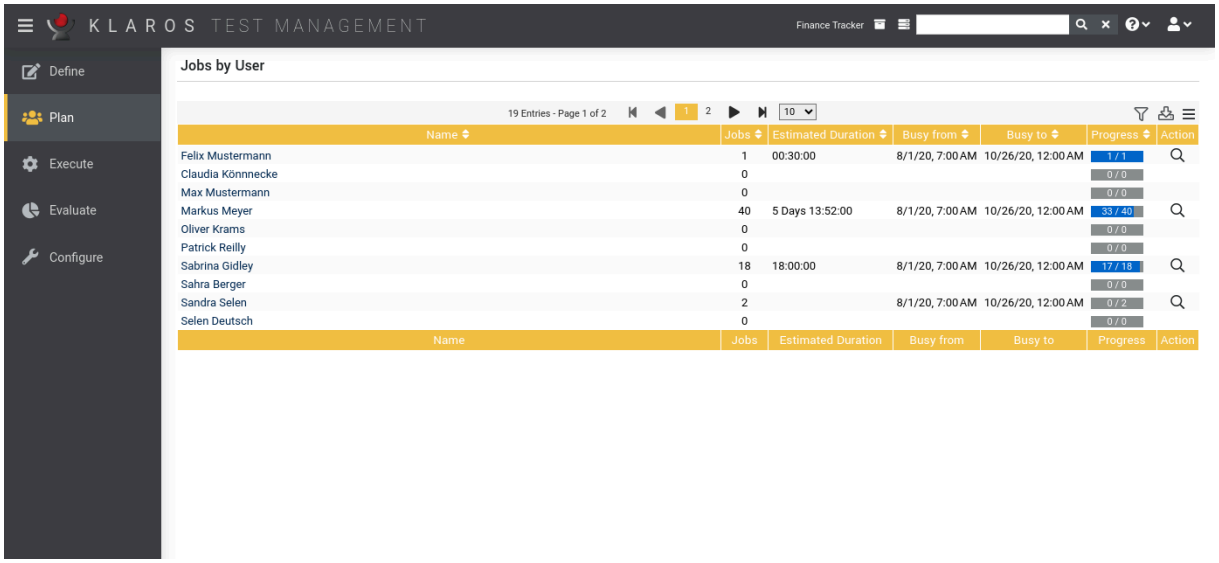


Figure 7.15. The “Jobs by User”Page

7.4.1. Jobs by User - Details

This page displays the workload, work duration and progress for a user in the currently selected project. The user's jobs can either be displayed listed in a table or visualized in a calendar view. Clicking the icon opens the detail page for the corresponding job. For more information, see Section 7.1.2, “Job - Details”.

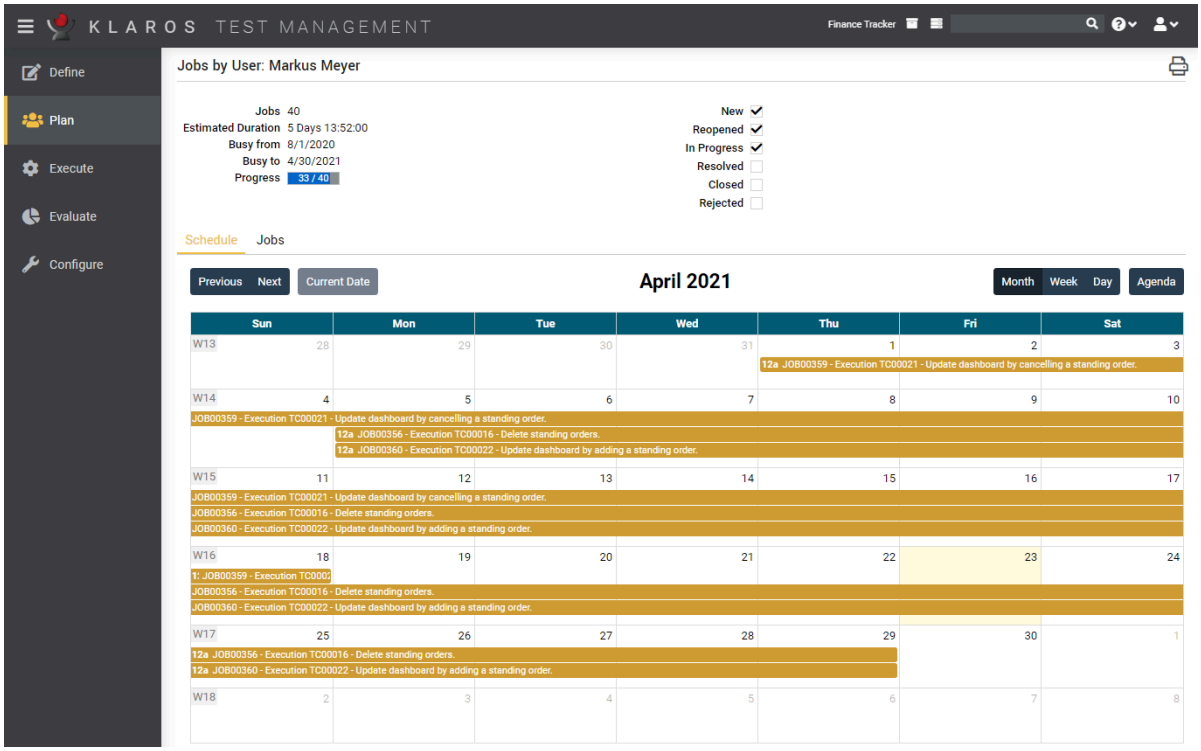


Figure 7.16. The “Jobs by User” Page

Chapter 8. Execute

This chapter describes the processes of executing test cases and test suites and jobs, continuing interrupted test executions and creating issues for failed tests.

8.1. My Jobs

Only available in Klaros-Testmanagement Enterprise Edition

On the *My Jobs* page, the jobs of the active user as well as all unassigned jobs can be executed. (Figure 8.1).

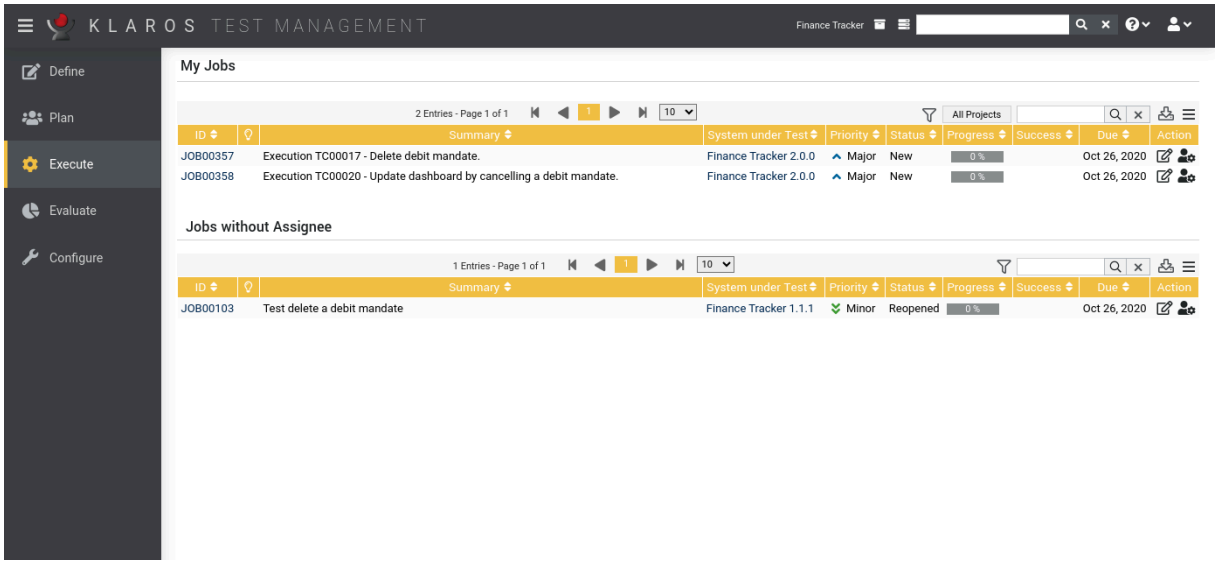


Figure 8.1. The “My Jobs” Page

The *My Jobs* and *Jobs without Assignee* tables show the following values:

Project — The project to which the job belongs to.

ID — The automatically assigned ID.

Summary — A summary of the job.

System under Test — The system under test assigned to this job.

Priority — The priority of the job. Possible values are *Trivial*, *Low*, *High-Critical* und *Blocker*.

State — The state of the job. Possible values are *New*, *Reopened*, *In Progress*, *Resolved*, *Closed* and *Rejected*.

Progress — The percentage of executed test cases of this job and its sub-jobs.

Success	The success rate of this job.
Due Date	The date on which this job is due to be finished.
Action	The actions that can be performed.

8.1.1. Action

The following actions can be performed in the action column:

- Edit
- Execute
- Manually execute
- Reopen or update
- Execute review
- Import results

8.1.2. Table Operations

The following operations can be performed in the line above the tables on the right:

- Filter / Sort

  All Projects / Current Project Only
- Search
- Export
- Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

8.1.3. Executing a Job

The available action for each job in the tables depends on the job type, whether it contains manual or automated test cases and whether it has dependencies on other jobs.

Non-executable Jobs

If a job cannot be executed, the corresponding icon in the action column is grayed out. Hovering over this icon will display the reason in a tooltip.

Execute	The job is a task and it contains no test case and no test suite.
Manually execute	The job contains at least one manually executable test case.

Reopen or update	The job has already been completed.
Execute review	The job is a review job.
Import results	The job only contains automated executable test cases.

8.1.3.1. Executing a Review

Review jobs are used to *Review* test cases or test suites. Clicking the Finish Review button opens a dialog in which the test case or test suite can be approved or rejected.

8.2. Run Test Case

Executing a test case means either executing the specified test steps one by one on a specific version of the system under test in a selected test environment (only for test cases with at least one step) or importing the associated test result files.

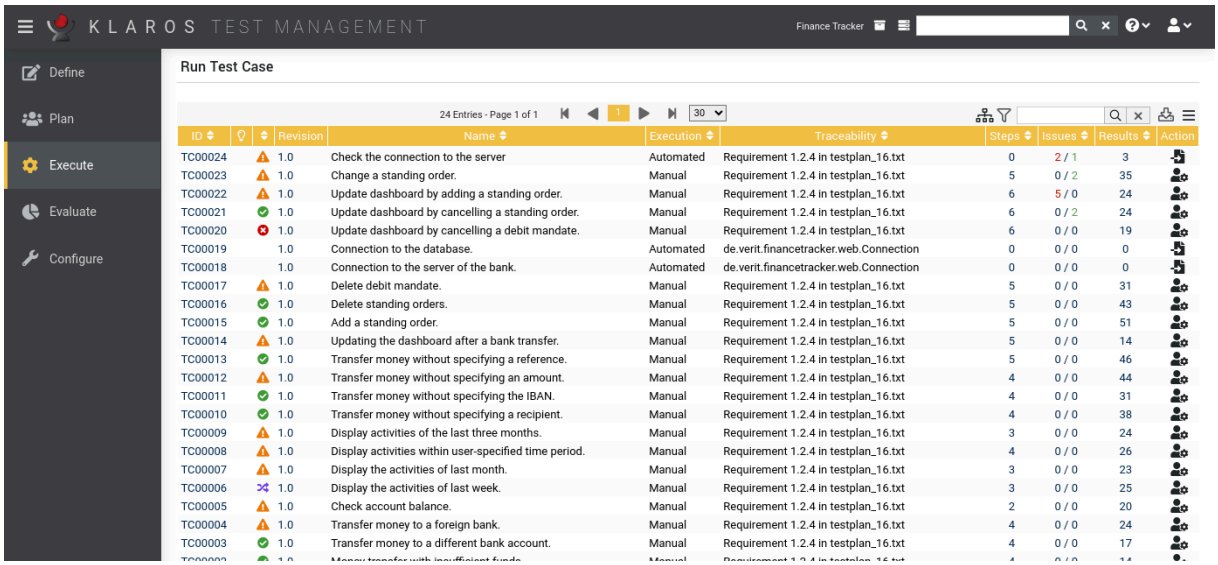


Figure 8.2. The “Run Test Case” Page

The *Run Test Case* page shows all test cases of the selected project in a table.

The table shows the following values:

ID	The automatically assigned ID.
Additional Information	A tooltip appears when the cursor is placed over the icon shown here.
Revision	The revision of the test case.
Name	The name of the test case.
Execution	Manual oder automated.
Traceability	The requirement assigned to this test case.

Steps	The number of associated test steps.
Results	The number of test case results.
Action	The executable actions.

8.2.1. Action

The following actions can be performed in the action column:

- Manually execute
- Import results

8.2.2. Table Operations

The following operations can be performed in the line above the table on the right:

- Categorize
- Filter / Sort
- Search
- Export
- Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

8.2.3. Executing a Test Case

Manual or Automated Test Execution

If a test case has at least one test step, it can be executed manually. Alternatively, test result files can be imported for all test cases (see Section 8.5, “Import Test Results”. for more information.

Note

Only test cases with the status *Draft* or *Approved* can be executed.

Clicking the icon displays a dialog with information about the test case (Figure 8.3).

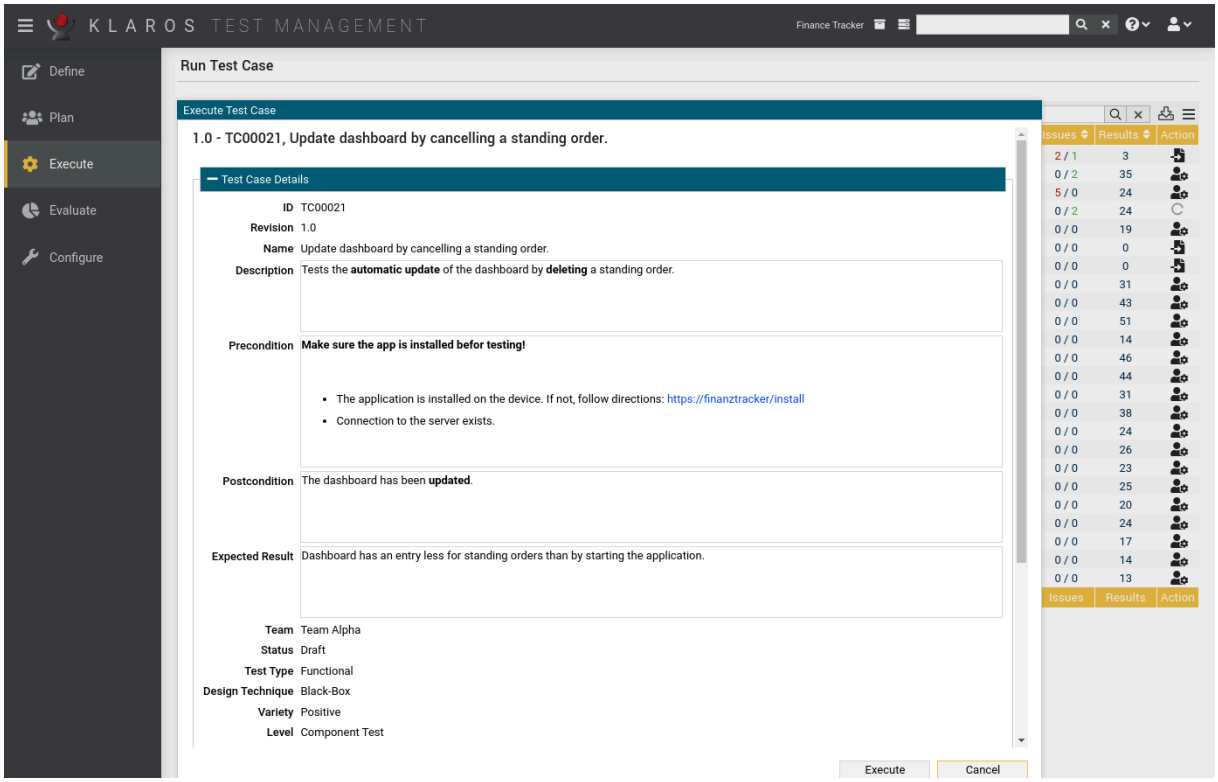


Figure 8.3. The “Execute Test Case” Dialog

Before a test case can be executed, the test environment the test case is run in and the system under test itself have to be selected. In addition, the user-defined properties of the test run can be set here (for more information, see Section 5.2.3.2.4, “User Defined Properties”).

Clicking the Execute button opens a second browser window where the test execution takes place. Make sure that pop-ups are allowed in your browser or add an exception for Klaros-Test-management. By default, a view with step-by-step instructions will be shown, but this can be changed to a tabular view during the execution.

8.2.3.1. The “Step-by-step Instructions” View

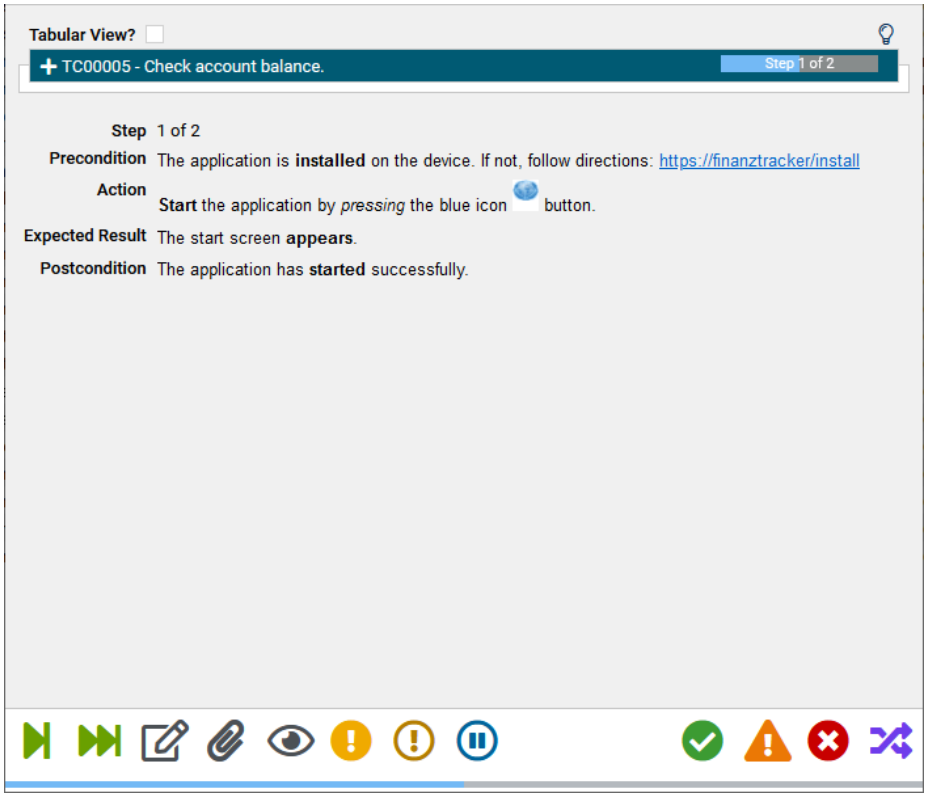


Figure 8.4. The "Step-by-step Instructions" View

The step-by-step instructions screen shows the *Action*, *Expected Result*, *Precondition* and *Postcondition* values of the current test step.

Depending on the results of the step, the user can click the following buttons:

Passed	Mark the test step as successfully completed.
Error	Mark the test step result as *Error*.
Failure	Mark the test step result as *Failure*.
Inconclusive	Mark the test step result as *Inconclusive*.
Skip	Skip the current test step without changing the result of the test case.
Skip All	Skip all remaining test steps without changing the result of the test case.
Edit	Edit this test step result.
Review	Request a review for this test case.
Create Issue	Create an issue related to this test case in an issue management system.
Link Issue	Link an existing issue from an issue management system to this test case.
Back	Go back to the last test step to repeat it or to edit it.

The same process is applied for each test step of the test case.

As soon as all steps have a result or have been skipped, the execution can be finished via a dialog.

Clicking the OK button, causes Klaros-Testmanagement to show the test run result of the test case (Figure 8.7).

8.2.3.2. The “Tabular Step Instructions” View

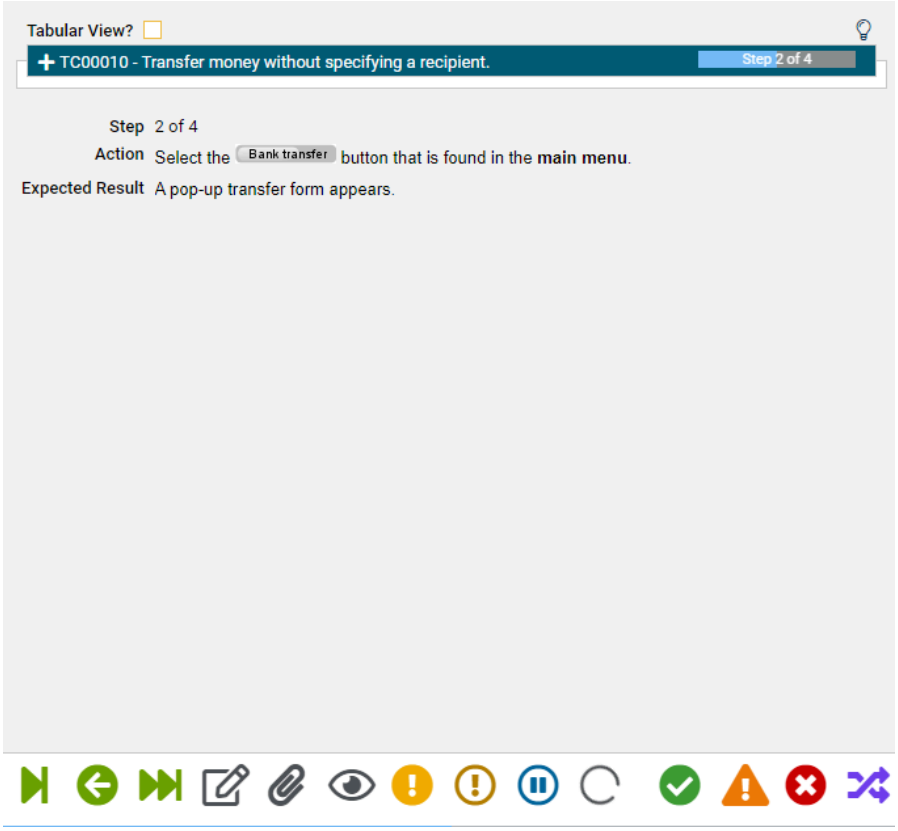


Figure 8.5. The “Tabular Step Instructions” View

The tabular step instructions screen shows the *Description*, *Expected Result*, *Precondition*, *Post-condition* and result of all steps, the default result being *Unknown*.

The following actions can be performed at each step:

Skip	Skip the current test step without changing the result of the test case.
Back	Go back to the last test step to repeat it again or to edit it.
Review	Request a review for this test case.
Create Issue	Create an issue related to this test case in an issue management system.
Link Issue	Link an existing issue from an issue management system to this test case.
Pause execution	Pauses the test execution and stops the time measurement.

⏭ Skip all remaining steps Finish the test and mark all remaining steps as skipped.

What is the difference between a failure and an error?

A failure is a discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.

An error is the inability of the system to perform the test correctly.

For each error, failure or inconclusive result the following dialog will be shown: (Figure 8.6).

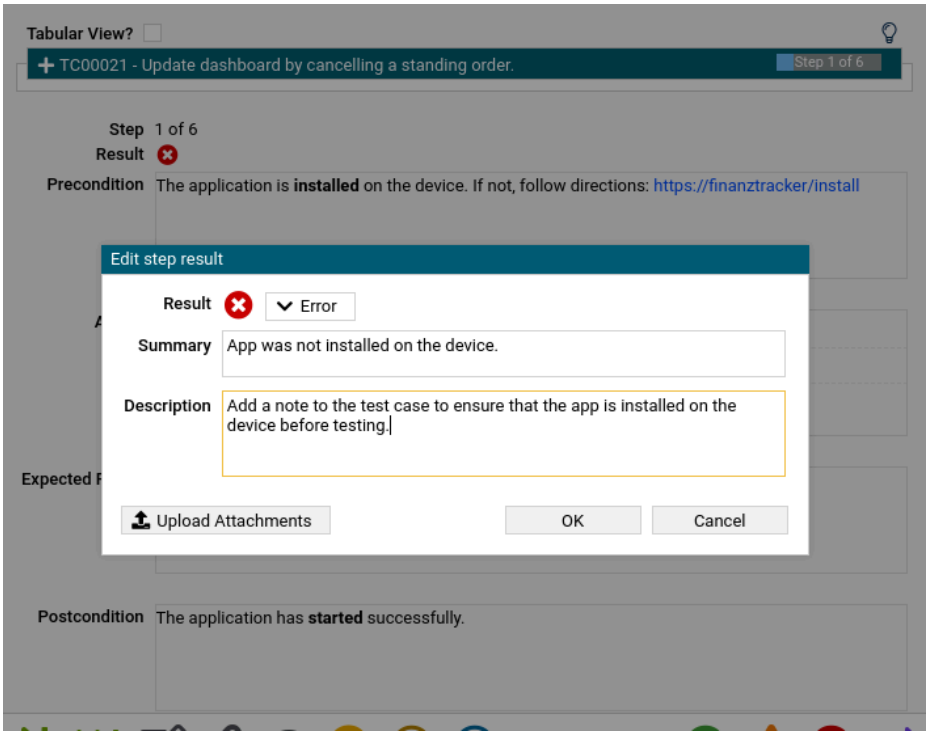


Figure 8.6. The “Edit step result” Dialog

- Upload Attachment opens a dialog for uploading attachments to the step.
- Finish test opens a dialog asking if the execution should be finished.
- Ok closes the dialog and displays the next step.
- Cancel closes the dialog.

Tip

The *Description* and *Summary* fields appear in all comment dialogs.

8.2.3.2.1. Finish Execution

After the last step result of a test case has been entered, a dialog is displayed asking whether the execution should be finished.

Clicking the OK button displays the result of the test run for this test case (Figure 8.7).

In the *Test Run - Overview* view, a summary and description can be entered for the test case result as well as for each individual test step result. (Figure 8.7).

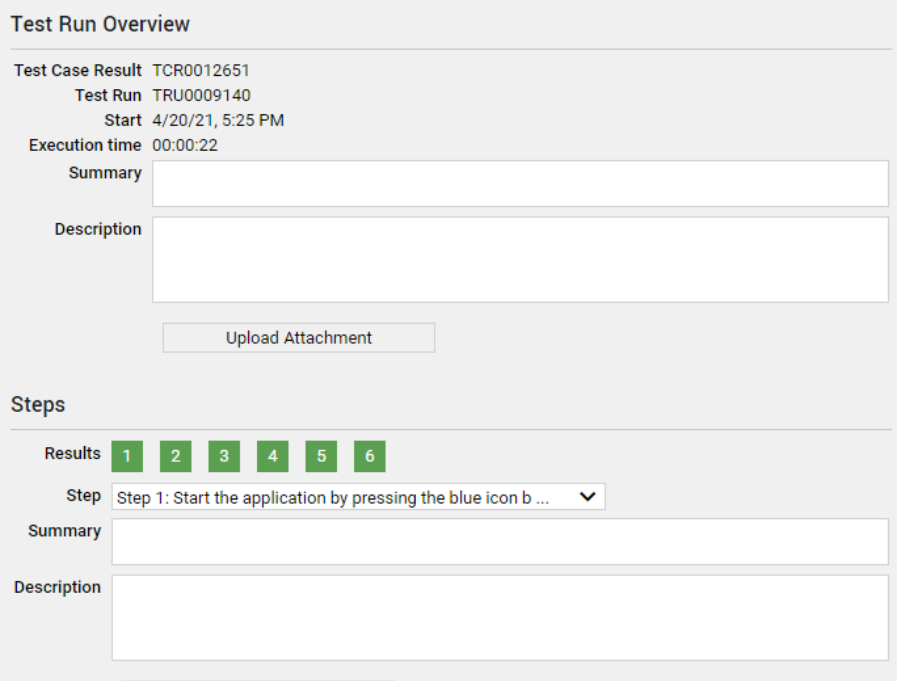


Figure 8.7. The “Test Run Overview” Page

Note

If an issue management system is configured for the active project, the Link Issue and Create Issue buttons are activated.

8.2.3.3. Creating Issues

During the execution of a test case, issues can be created for this test case in an external issue management system by clicking the button. Issues already created in an issue management system can be linked to the test case by clicking the button.

For detailed information on creating and linking issues, see Section 9.6.6, “Issue Details (Creating a new Issue)” and Section 9.6.7, “Link Issue”.

8.2.3.4. Creating Review Jobs

Only available in Klaros-Testmanagement Enterprise Edition

If an error is found in a test case description during test execution, a review job can be created by clicking on the button. A dialog window opens in which the details of the review job can be entered:

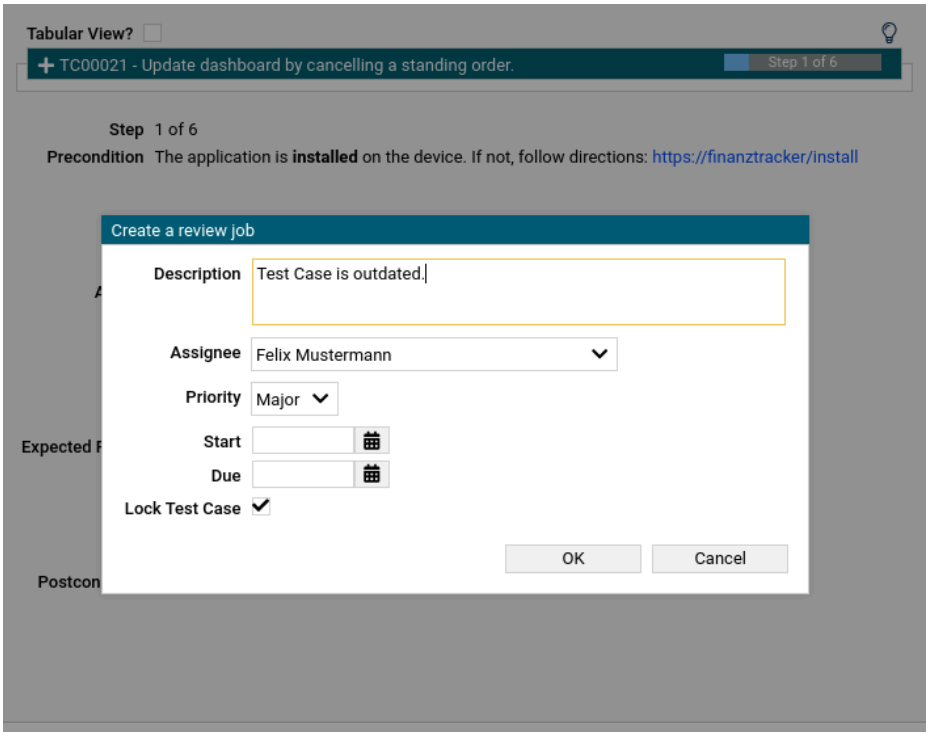


Figure 8.8. Creating a Review Job

8.3. Run Test Suite

The execution of a *Test Suite* consists of the execution of a defined set of test cases on a specific version of the system under test in a selected test environment.

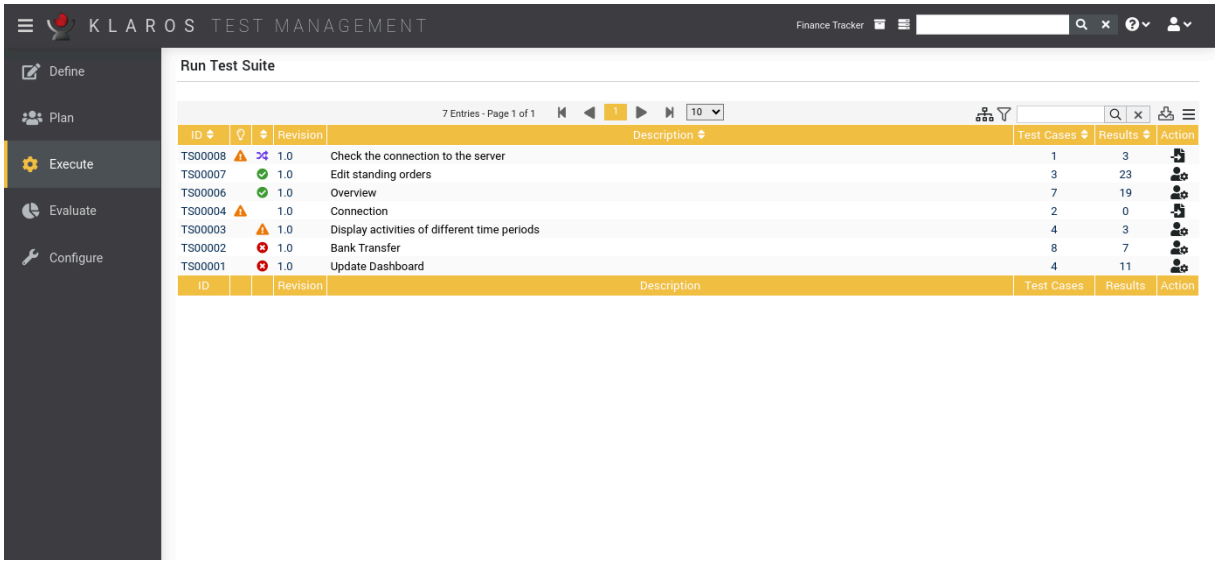


Figure 8.9. The “Run Test Suite” Page

The *Run Test Suite* page shows all test suites of the selected project in a table.

The table shows the following values:

ID The automatically assigned ID.

Additional Information	A tooltip appears when the cursor is placed over the icon shown here.
Revision	The revision of the test suite.
Description	The description of the test suite.
Test Cases	The number of test cases in the test suite.
Results	The number of test case results.
Action	The executable actions.

8.3.1. Action

The action column is located on the far right of the table. The following actions can be performed in the action column:

- Manually execute
- Import results

8.3.2. Table Operations

The following operations can be performed in the line above the table on the right:

- Categorize
- Filter / Sort
- Search
- Export
- Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

8.3.3. Executing a Test Suite

Important

A test suite with at least one test step in one of its test cases can be executed manually. For all other test suites, test result files can be imported (see Section 8.5, “Import Test Results”. for more information.

Warning

Test cases with the states *Locked*and *Skip* or without any steps will be skipped when executing the test suite.

Clicking the button displays a dialog with information about the test suite (Figure 8.10).

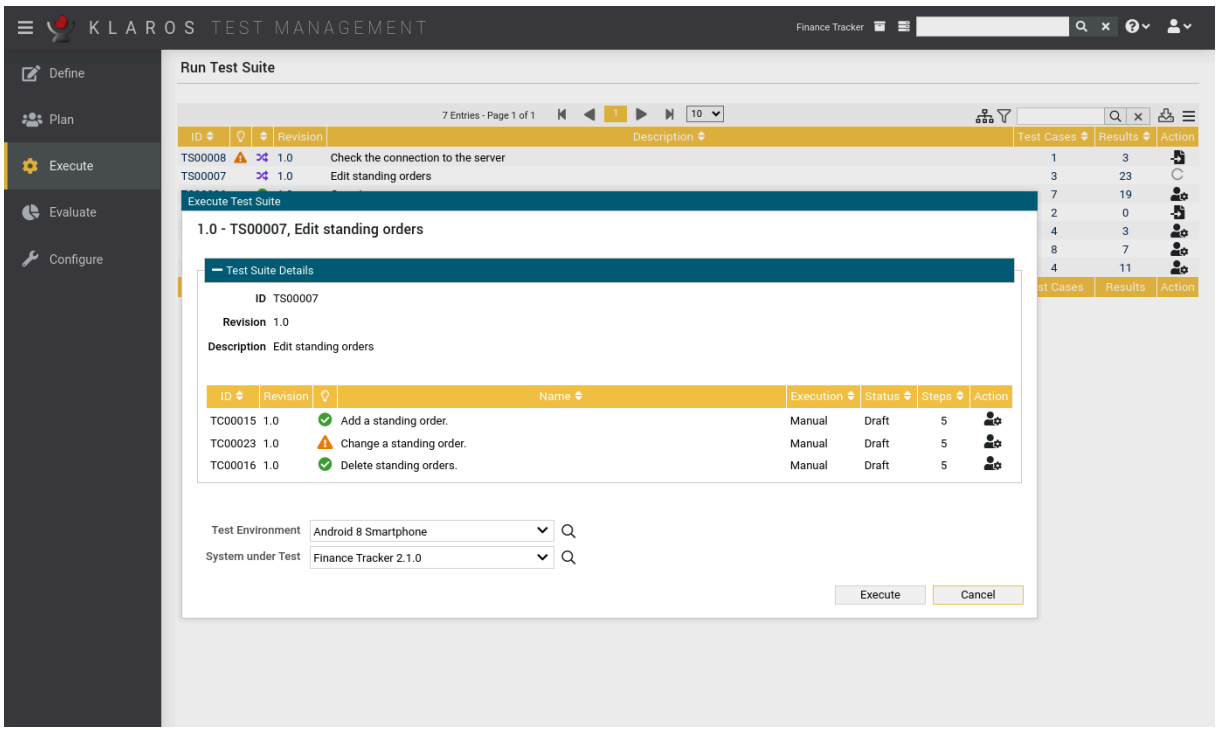


Figure 8.10. The “Execute Test Suite” Dialog

Before a test suite can be executed, the test environment the test suite is run in and the system under test itself have to be selected. In addition, the user-defined properties of the test run can be set here (for more information, see Section 5.2.3.2.4, “User Defined Properties”).

Clicking the Execute button opens a second browser window where the test execution takes place. Make sure that pop-ups are allowed in your browser or add an exception for Klaros-Test-management.

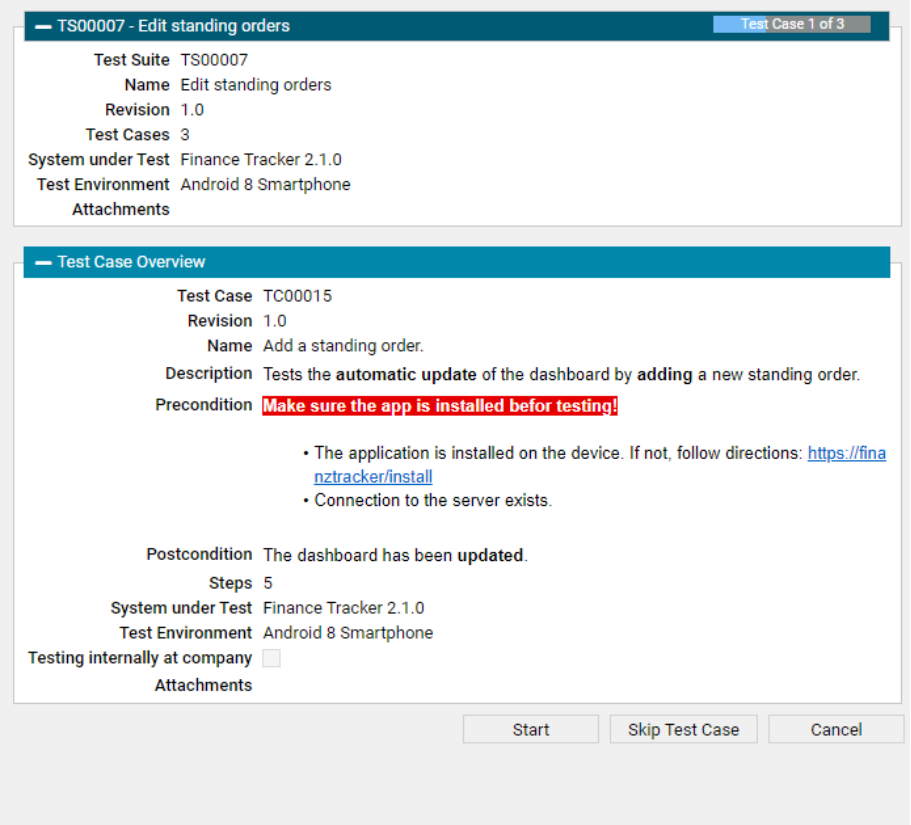


Figure 8.11. The "Test Case Overview" View

This view shows the overview of the next test case to be executed. Clicking the Start button starts the execution of the test case (the execution of a test case is described in detail in Section 8.2.3, "Executing a Test Case").

8.3.3.1. Skip Test Cases

During the execution of a test suite, individual test cases can be postponed for later execution or permanently skipped.

8.3.3.1.1. Skip Test Cases Permanently

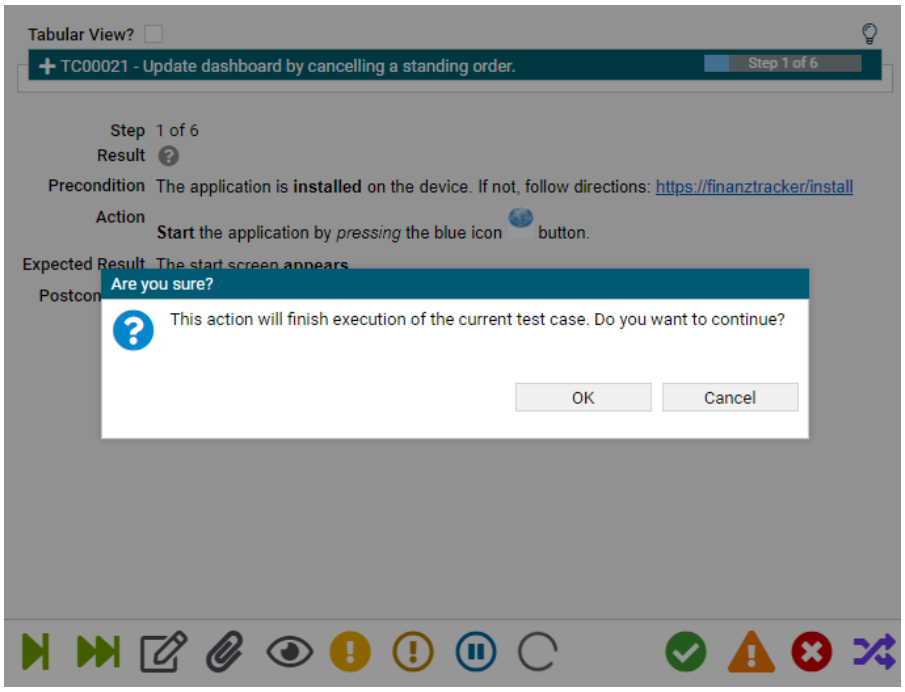


Figure 8.12. The “Permanently Skipping a Test Case” Dialog

After permanently skipping a test case, the tester is prompted to enter a reason why the test case was skipped. If explanatory templates are already defined (see Section 10.4.3, “Test Execution”), one of these templates can be selected.

By clicking on the + icon, the selected template is copied to the test case result summary.

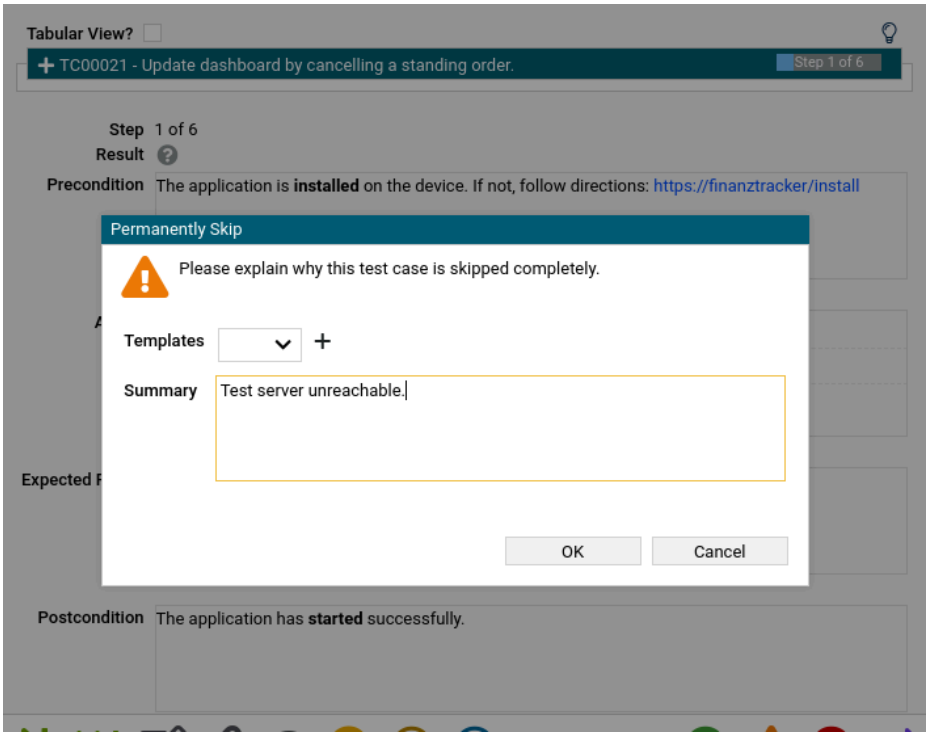


Figure 8.13. The “Inserting a Reason Template” Dialog

Note

Skipping all steps of a test case will mark the whole test case as permanently skipped.

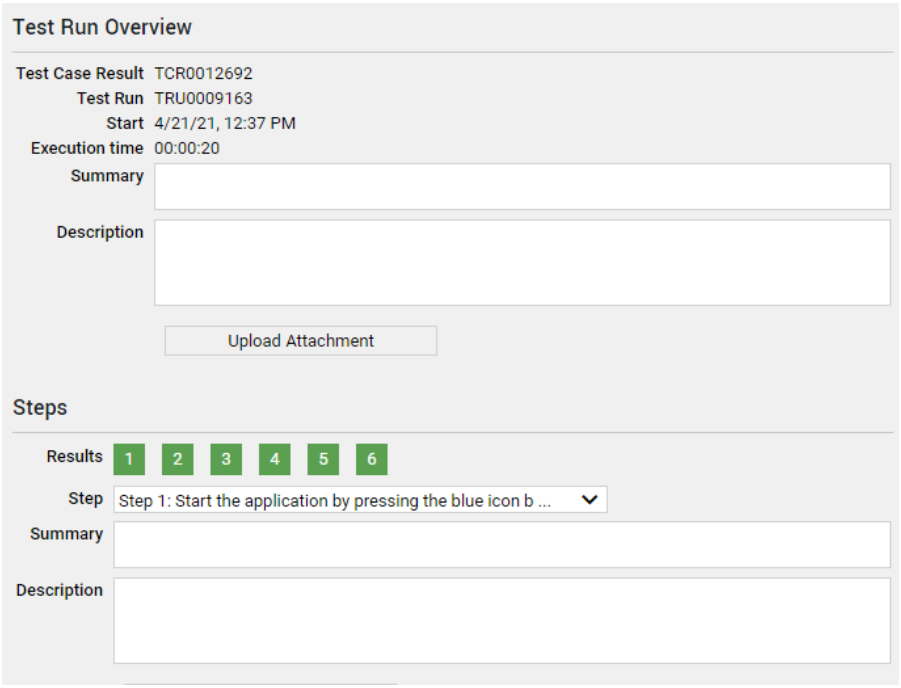


Figure 8.14. The “Test Run Overview” View

After finishing a test run, the tester can also edit the reasons for skipping test cases in the test run overview (see Figure 8.14).

8.3.3.2. Finish Execution

After all test cases have been executed, the *Test Suite Execution Overview* view is displayed. Here, a summary and description can be entered for each individual test case and test step of the test run (Figure 8.15).

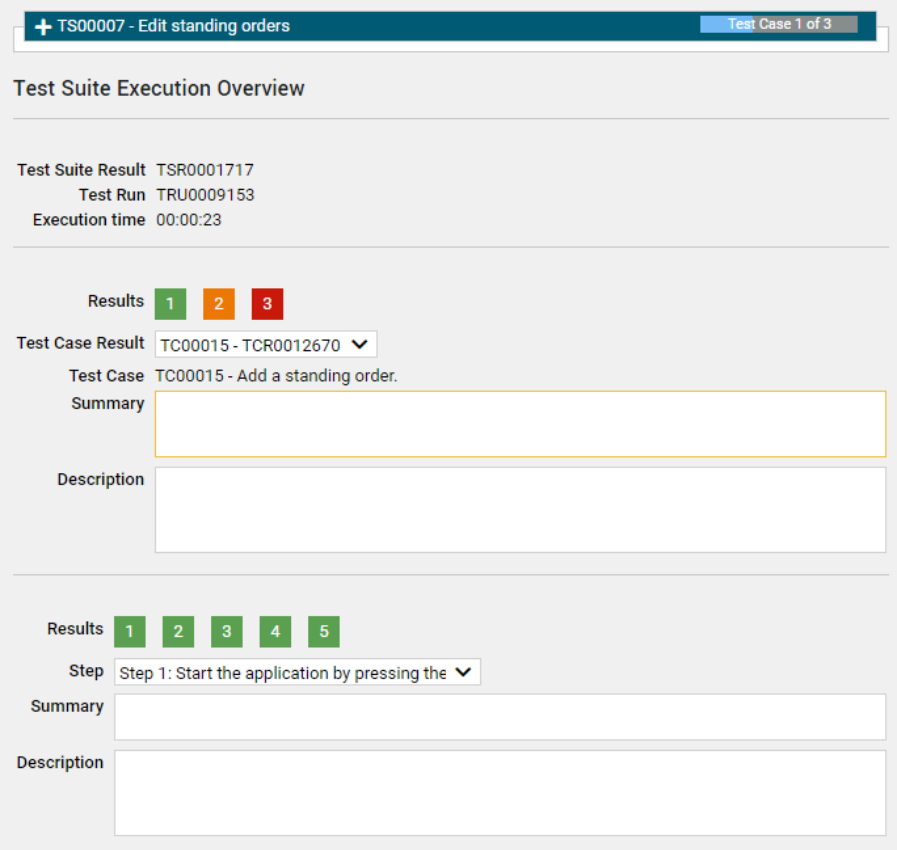


Figure 8.15. The “Test Suite Execution Overview” View

8.4. Continue Test Run

If a test run is not completed and must be suspended, execution can be continued later. Execution continues with the first test step that does not yet have a result.

The *Continue Test Run* page displays all unfinished test runs of the selected project in a table.

Note

The *Testers may only resume their own test runs* setting controls whether users with the *Tester* role can see and resume suspended test runs from other users (Section 10.4.1, “Miscellaneous”).

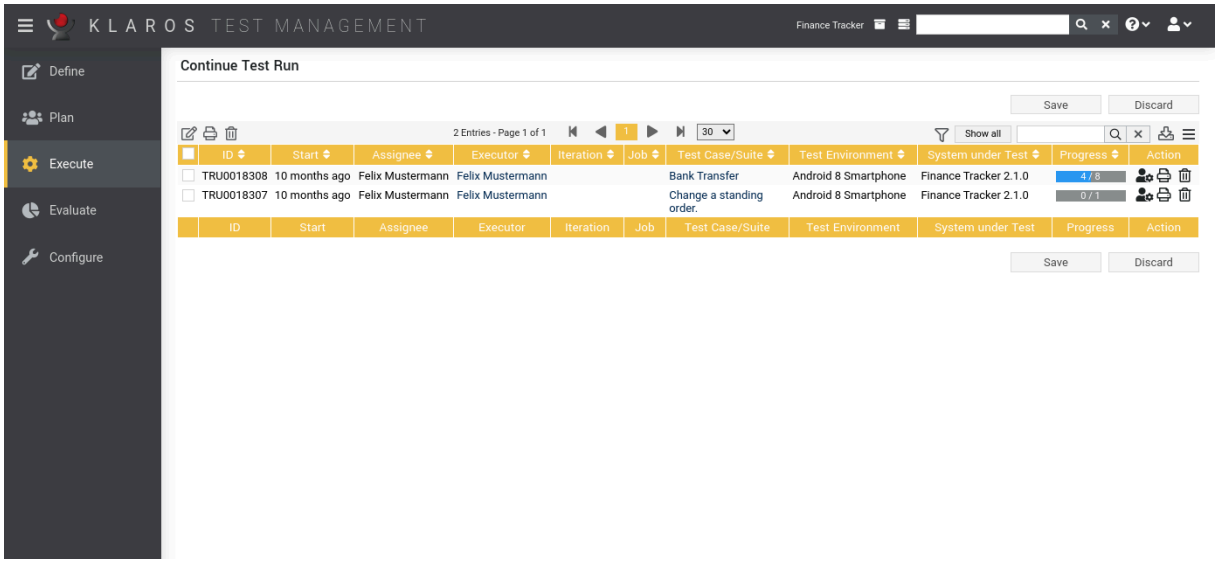


Figure 8.16. The “Continue Test Run” Page

The table shows the following values:

ID	The automatically assigned ID.
Start	The start time of the test run.
Executor	The name of the user who last executed the test run.
Iteration	The iteration in which the test run is being executed.
Test Case/Suite	The ID of the test case or test suite of the test run.
Test Environment	The test environment in which the test is being executed.
System Under Test	The system under test being tested.
Progress	The number of finished test cases.
Action	The executable actions.

8.4.1. Action

The following actions can be performed in the action column:

- Display the test case results captured so far of this test run.
- Display the test case results captured so far of this test run. Continue the execution of the test run
- Open print view.
- Delete this test run.

If a test run has been deleted, it is initially marked with a deletion marker and is only visible to administrators. For deleted test runs, the following actions are available instead of *Delete*:

Restore this test run.

Irrecoverably remove the test run from the database (only Administrator).

Deleting a Test Run

When a test run is deleted, the associated test case results and test suite results are also deleted

Once all test runs for a specific system under test or test environment have been deleted, the respective system under test or test environment may also be deleted. Otherwise, they remain locked for deletion.

8.4.2. Bulk Actions

Certain actions can also be applied to several test runs at once. To do this, select the test runs to which the action is to be applied in the leftmost column.

The following bulk actions are supported for test runs:

Edit the attributes of this test run.

Open print view.

Delete

Restore (only Administrator)

Irrecoverably remove the selected test runs from the database (only Administrator).

For additional information, see Section 5.2.3.1.5, “Bulk Actions”.

8.4.3. Table Operations

The following operations can be performed in the line above the table on the right:

Filter / Sort

Show all / Only active (only Administrator)

Search

Export

Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

8.4.4. Continue test run

By clicking on the in the action column, you get to an overview page to continue the test run. All test case results and, if available, the test suite result of the test run are displayed in tables here.

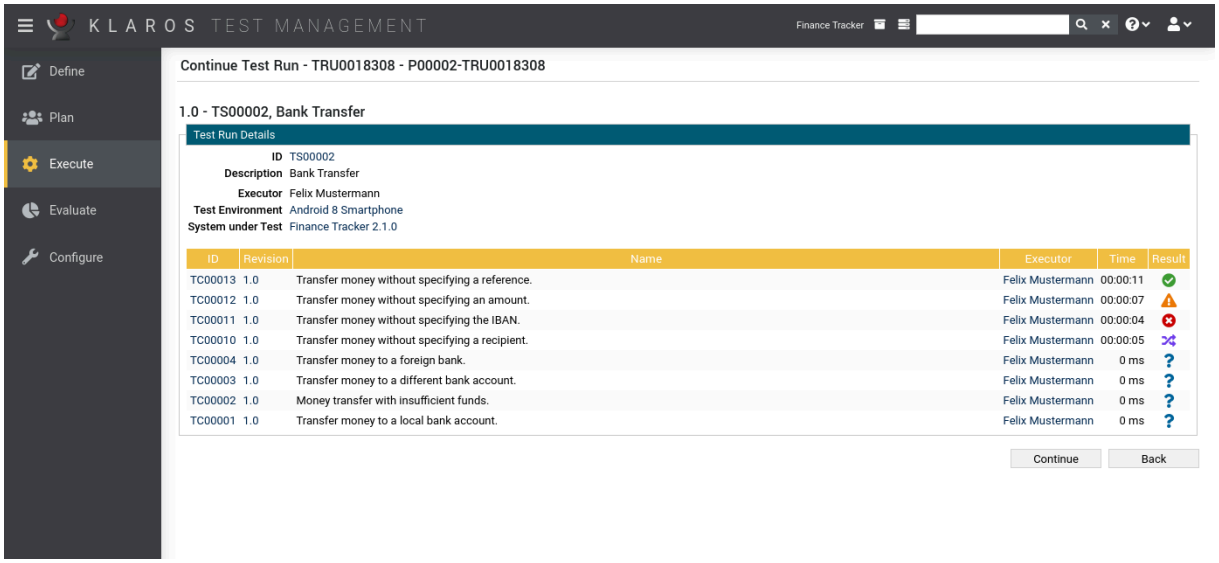


Figure 8.17. The “Continue Test Run” Page

Clicking the Continue button resumes execution of the test run at the last saved test step.

Clicking the Back button navigates back to the *Continue Test Run* page (Figure 8.16).

8.5. Import Test Results

In Klaros-Testmanagement test results from various test frameworks can be imported (see Section 8.5.1, “Supported Frameworks”) for a complete list of supported formats). If the associated test cases or test suites are not already defined in Klaros-Testmanagement, they can be created automatically during the import.

To start the import, an XML file with results from a supported test framework must be uploaded. When the file upload is completed it is mandatory to select the system under test and the test environment these results should be assigned to. The format of the result file is typically detected automatically.

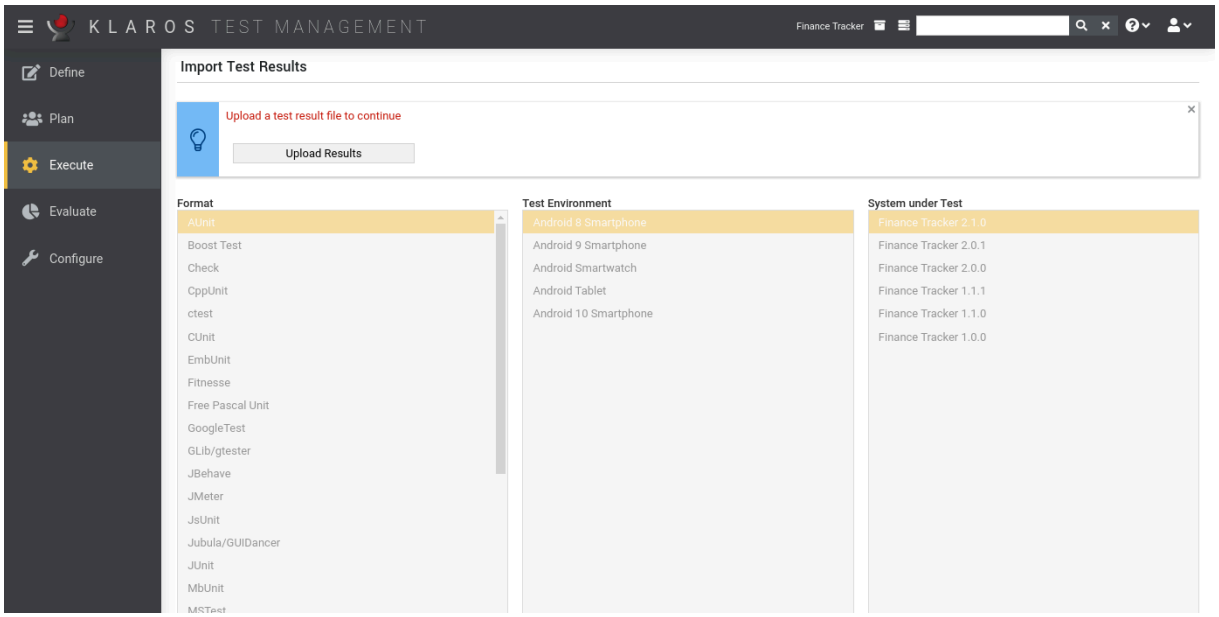


Figure 8.18. The “Import Test Results” Page

Clicking the Next button opens the *Assign Test Results* page. The test case of each test result can be selected here using the corresponding drop-down box. Each result must be assigned to a different test case. In addition, the test suite can be assigned as well.

Initial test suite / test case name matching is determined by the internal name of the test case or test suite in Klaros-Testmanagement vs. the external name of the test case or test suite in the result file. This heuristic can be overridden here. Once associated, the matching of a test case or test suite to an external name is automatically applied for further imports until defined otherwise. This also applies to result imports via REST API or other external sources like the Jenkins plugin.

You may decide to skip the import of individual results by deselecting the entry for this result using the checkbox on the left-hand side. If the import process is triggered by a test execution job, the initial selection is matched against the test cases or test suites contained in the job. Otherwise, all results are initially selected for import automatically.

If the option *Automatically create new test cases for unknown tests when importing test case results* is selected, a new test case is automatically created upon import when it is not already present.

When the option *Create additional test suite results and a corresponding test suite if necessary when importing test case results* is selected, test suite results are automatically created for the corresponding test suite information contained in the result file. This information may vary with the import format. Additionally, a corresponding test suite is created for the test suite result if it did not yet exist.

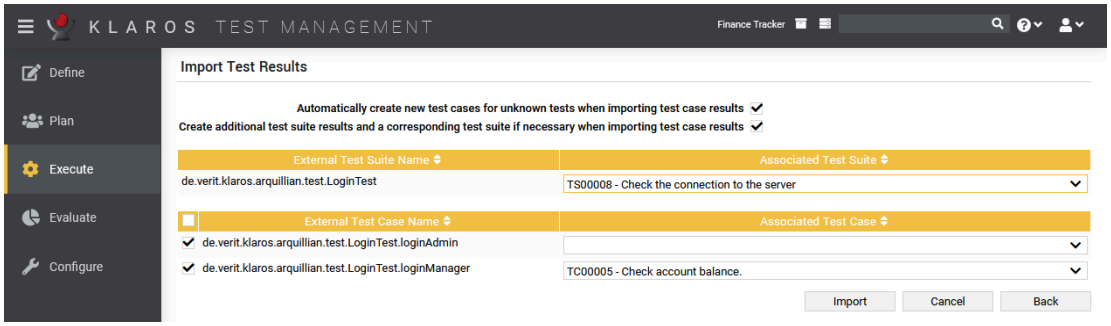


Figure 8.19. The “Import Test Results” Page

Pressing the Import button will import all results into Klaros-Testmanagement.

8.5.1. Supported Frameworks

The following frameworks are supported:

AUnit	**AUnit** is an adaptation of the Java JUnit and C++ CppUnit unit test frameworks for Ada code.
Boost Test	The **Boost Test**library provides a matched set of components for writing test programs, organizing tests into simple test cases and test suites, and controlling their runtime execution.
Check	**Check** is a unit testing framework for C.
CppTest	**CppTest** is a portable and powerful, yet simple, unit testing framework for handling automated tests in C++. The focus lies on usability and extendability.

CppUnit	**CppUnit** is a C++ unit testing framework.
ctest	**ctest** is the testing driver provided by CMake.
CUnit	**CUnit** is a lightweight system for writing, administering, and running unit tests in C.
embUnit	**Embedded Unit** is unit testing framework for Embedded C Systems.
Fitnesse	**Fitnesse** is a tool for specifying and verifying application acceptance criteria (requirements).
Free Pascal Unit	**Free Pascal Unit** is a port to Free Pascal of the JUnit core framework.
GLib/gtester	**GLib** provides the core application building blocks for libraries and applications written in C. It provides the core object system used in GNOME, the main loop implementation, and a large set of utility functions for strings and common data structures. **gtester** is a utility to run unit tests that have been written using the GLib test framework.
Gauge	**Gauge** is a light-weight cross-platform test automation tool with the ability to author test cases in a business language.
GoogleTest	**GoogleTest** is a unit testing library for the C++ programming language created by Google.
JBehave	**JBehave** is a framework for Behaviour-Driven Development (BDD). BDD is an evolution of test-driven development (TDD) and acceptance-test driven design, and is intended to make these practices more accessible and intuitive to newcomers and experts alike. It shifts the vocabulary from being test-based to behaviour-based, and positions itself as a design philosophy.
JMeter	**JMeter** is a 100% pure Java application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.
JsUnit	**JsUnit** is a Unit Testing framework for client-side (in-browser) JavaScript. It is essentially a port of JUnit to JavaScript. Also included is a platform for automating the execution of tests on multiple browsers and multiple machines running different OSs.
Jubula	**Jubula** provides automated functional GUI testing for various types of applications.
JUnit	**JUnit** is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks.

MbUnit	**MbUnit** is an extensible unit testing framework for the .NET Framework that takes in and goes beyond xUnit pattern testing. MbUnit is part of the Gallio bundle.
MSTest	**MSTest** is the command-line command that is used to run tests. This command has several options you can use to customize your test run.
NUnit	**NUnit** is a unit-testing framework for all .Net languages.It is written entirely in C# and has been completely redesigned to take advantage of many .NET language features, for example custom attributes and other reflection related capabilities.
PHPUnit	**PHPUnit** is a programmer-oriented testing framework for PHP. It is an instance of the xUnit architecture for unit testing frameworks.
QF-Test	**QF-Test** is a professional tool for automated testing of Java and Web applications with a graphical user interface.
QTestLib	The **QTestLib** framework is a tool for unit testing Qt based applications and libraries.
Ranorex	**Ranorex** is a GUI test automation framework for testing of desktop, web-based and mobile applications.
Selenium	**Selenium** is a web browser automation tool primarily used for automated testing of web apps. Selenium is able to produce **JUnit**-compatible test results, which can be imported into Klaros-Testmanagement.
Squish	**Squish** is a GUI Test Automation Tool for all kinds of cross-platform desktop, mobile, embedded and web applications. Squish is able to produce **JUnit**-compatible test results, which can be imported into Klaros-Testmanagement.
TestComplete	**TestComplete** is a cross-platform automated GUI testing tool from SmartBear Software.
TESSY	**TESSY** is a software tool to automate module-unit testing of embedded software written in various embedded dialects of the programming languages C and C++.
TestNG	**TestNG** is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use. TestNG is able to produce **JUnit**-compatible test results, which can be imported into Klaros-Testmanagement.
TUSAR	**TUSAR** stands for Thales Unified Software Analysis Report. It is a generic metric format composed of 4 categories: Coverage, Measure, Test and Violations.

Unified Functional Testing (UFT) / QuickTest Professional (QTP)	**Unified Functional Testing** is a functional automated testing software by Micro Focus formerly known as QuickTest Professional (QTP) by HPE Software.
UnitTest++	**UnitTest++** is a lightweight unit testing framework for C++. It was for test-driven development on a wide variety of platforms. Simplicity, portability, speed, and a small footprint are all very important aspects.
Valgrind	**Valgrind** is an instrumentation framework for building dynamic analysis tools.
xUnit.net	**xUnit.net** is a free, open source, community-focused unit testing tool for the .NET Framework. xUnit.net works with ReSharper, CodeRush, TestDriven.NET and Xamarin.

Support for Frameworks not Listed Here

Even if you did not find your favorite testing framework here this does not mean that it is not already supported.

Many tools offer to export their result files in the JUnit XML format (e.g. Selenium, Robot and others) which is a supported format.

If you like to see an additional format supported, please contact support@verit.de.

Chapter 9. Evaluate

This chapter shows how data and results can be evaluated, that were gathered during testing. It explains how the dashboard works and how it can be configured, how reports are configured and rendered and how issues can be created in a connected issue management system.

9.1. The Dashboard

A dashboard is an overview page for configurable reports displayed as a chart. Dashboards can be individually configured and shared with or blocked for other users. Each user can configure any number of dashboards. An example of a dashboard is shown in Figure 9.1.

New adds a new report to the dashboard.

Save saves all changes to the dashboard and its reports.

Cancel discards all changes to the dashboard and its reports.

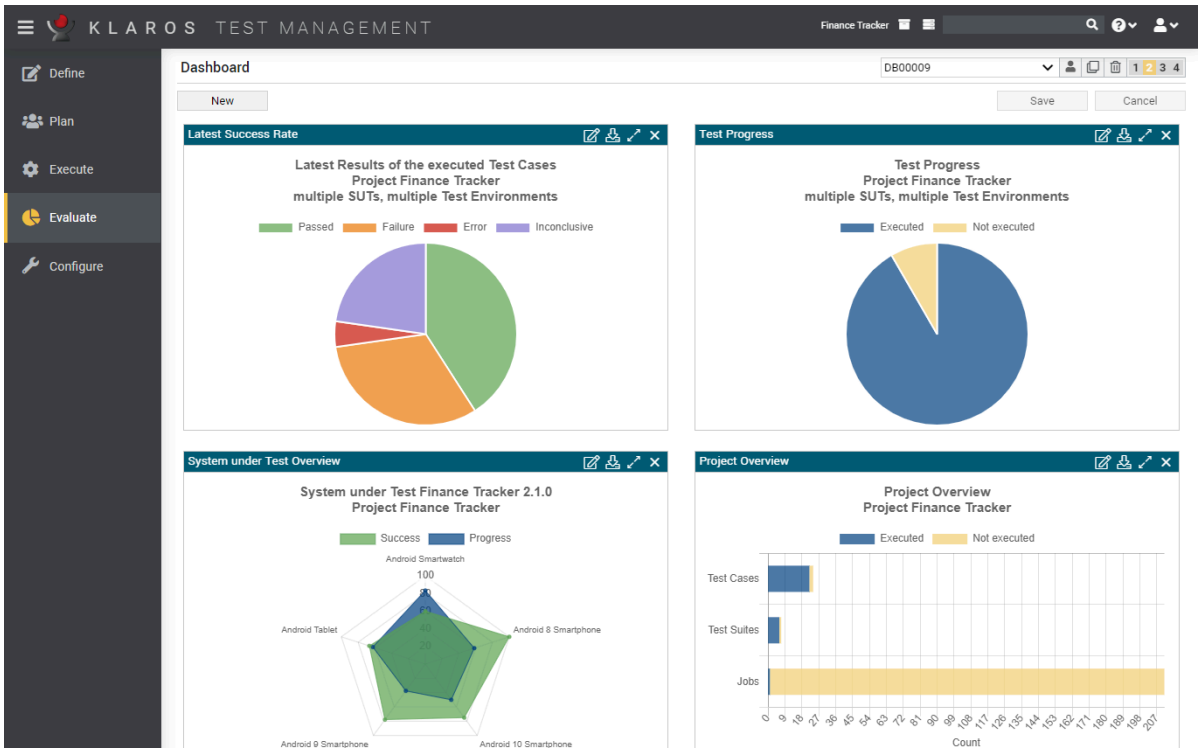


Figure 9.1. The “Dashboard” Page

Users can switch between dashboard at any time. The drop-down list on the upper right page shows all available dashboards (private and shared by other users). Next to this drop down list are several icons:

When selected, all users can use this dashboard.

When selected, this dashboard can only be edited by the owner.

If selected, marks this as the default dashboard.

Creates a private copy of this dashboard.

Deletes this dashboard and all of its reports.

The number of reports that should be shown in a row can be set with the numbered buttons (1 to 4) on the upper right. The currently selected number of reports per row is marked in orange. Reports can be moved to another position by dragging and dropping.

All dashboard reports have the same structure: they consist of a title bar and the area where the chart is displayed. The title bar contains the name of the report and several icons:

Opens up a dialog to edit the report configuration.

Downloads the report as a PNG file.

Expands/collapses the report.

Deletes the report.

9.1.1. Default Dashboard

Upon the first start of Klaros-Testmanagement, a default dashboard with three predefined reports is created. This dashboard is accessible by all users an is automatically displayed to each new user. Administrators can change the default dashboard by setting any other dashboard as the default dashboard.

Note

There is always only one default dashboard. If a new default dashboard is set, the previous default dashboard loses this status.

9.1.2. Editing a Dashboard Report

Clicking the icon opens up a dialog that shows all configurable parameters of the report as well as the report type (which cannot be changed). Changes can be applied to the report by clicking the `OK` button or discarded by clicking the `Cancel` button.

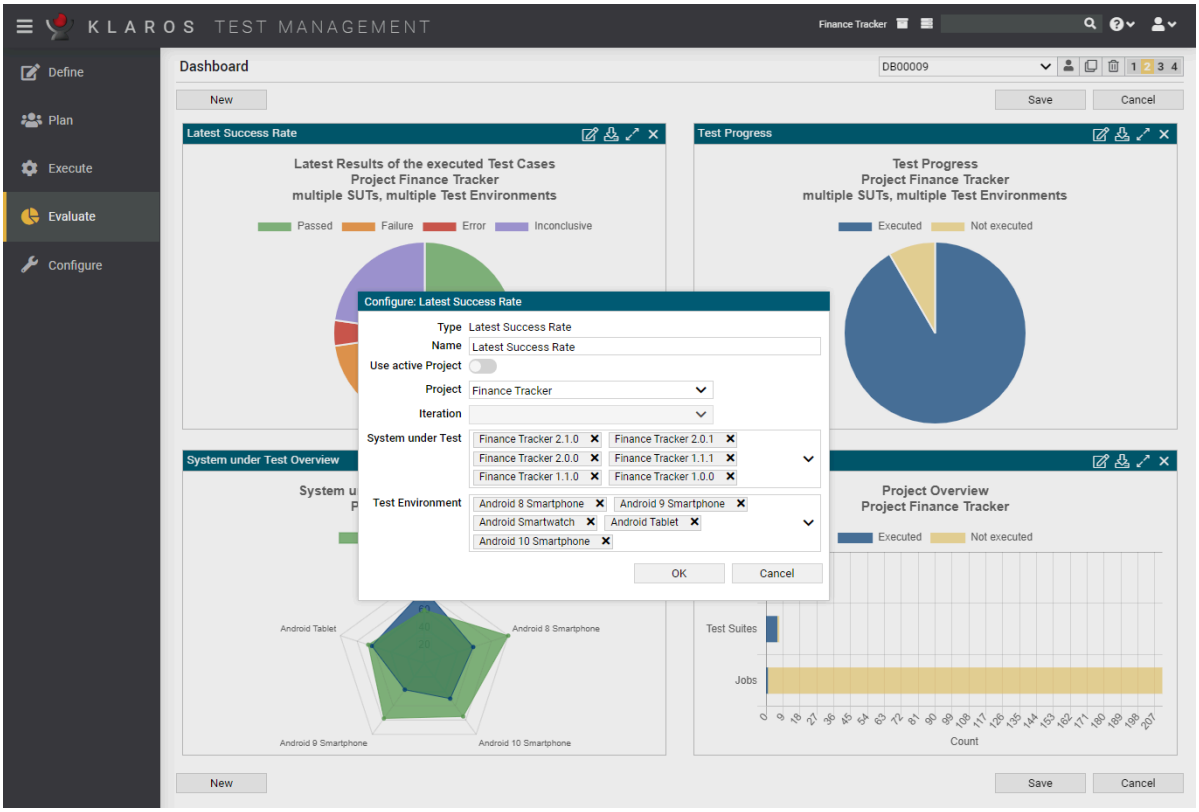


Figure 9.2. The “Configure Report” Dialog

9.1.3. Report Types

All reports initially show data from the currently selected project.

In addition to the data of the active project, data of specific projects can also be displayed in the Klaros-Testmanagement Enterprise Edition.

Klaros-Testmanagement Community Edition contains three report types:

- *Project Overview*
- *Latest Success Rate*
- *Test Activity*

enterprise edition Only available in Klaros-Testmanagement Enterprise Edition

The Klaros-Testmanagement Enterprise Edition also contains four additional report types:

- *System under Test Overview*
- *Test Environment Overview*
- *Test Progress History*
- *Test Progress*

These reports are described in detail in the following sections.

9.1.3.1. The “Project Overview” Report

The *Project Overview* report shows several metrics of a single project in a chart.

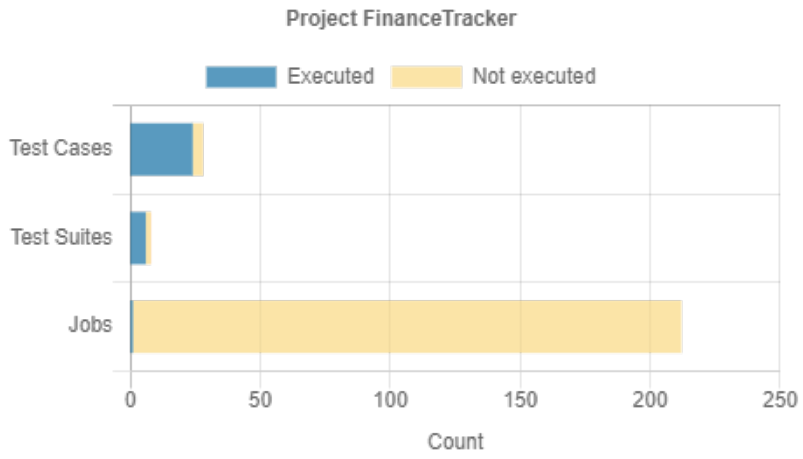


Figure 9.3. The “Project Overview” Report

This report displays the following metrics:

- The number of executed/not executed test cases.
- The number of executed/not executed test suites.
- The number of planned/executed jobs.

The following properties of the “Project Overview” report diagram are configurable:

- The name of the report diagram.
- The selection of whether to use the active project or another one in this report. (Klaros-Test-management Enterprise Edition only).
- The selection of whether to use the active iteration or another one in this report.
- If the active iteration is not used, a dropdown list with the available iterations is displayed here.

9.1.3.2. The “Latest Success Rate” Report

The *Last Success Rate* report shows the latest results of executed test cases for the selected combination of system under test (one or more) and test environment (one or more).

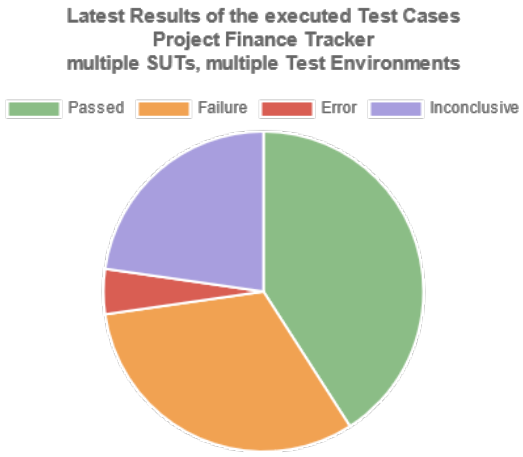


Figure 9.4. The “Latest Success Rate” Report

The following properties of the “Latest Success Rate” report diagram are configurable:

- One or more systems under test.
- One or more test environments.
- The name of the report diagram.
- The selection of whether to use the active project or another one in this report. (Klaros-Test-management Enterprise Edition only).
- If the active project is not used, the following objects to be viewed can be selected:
- The selection of whether to use the active iteration or another one in this report.
- If the active iteration is not used, a dropdown list with the available iterations is displayed here.

9.1.3.3. The “Test Activity” Report

The *Test Activity Report* shows the test case results for a selected combination of system under test and test environment in a selected period of time as a histogram.

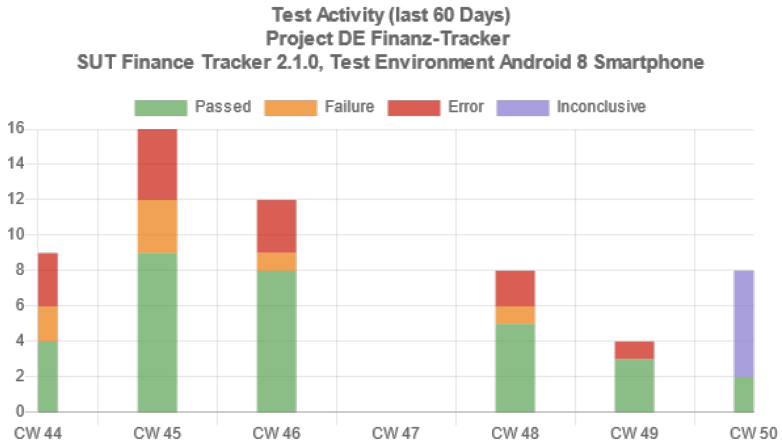


Figure 9.5. The “Test Activity” Report

The following properties of the “Test Activity” report diagram are configurable:

- Project
- Iteration (if available)
- One or more systems under test.
- One or more test environments.
- The name of the report diagram.
- The selection of whether to use the active project or another one in this report. (Klaros-Test-management Enterprise Edition only).
- If the active project is not used, the following objects to be viewed can be selected:
- The selection of whether to use the active iteration or another one in this report.
- If the active iteration is not used, a dropdown list with the available iterations is displayed here.
- The time period to be covered in days.

9.1.3.4. The “System under Test Overview” Report

Only available in Klaros-Testmanagement Enterprise Edition

The *System under Test Overview Report* shows the success and progress rate of test environments under a single system under test in a radar chart. If fewer than three test environments are configured, a bar chart will be displayed instead.

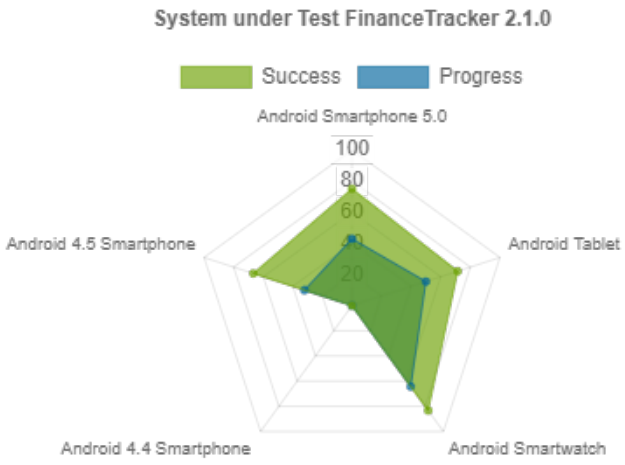


Figure 9.6. The “System under Test Overview” Report

The following properties of the “System under Test Overview” report diagram are configurable:

- The Project.

- Iteration (if available).
- A system under test.
- One or more test environments.
- The name of the report diagram.
- The selection of whether to use the active project or another one in this report. (Klaros-Test-management Enterprise Edition only).
- If the active project is not used, the following objects to be viewed can be selected:
- The selection of whether to use the active iteration or another one in this report.
- If the active iteration is not used, a dropdown list with the available iterations is displayed here.

9.1.3.5. The “Test Environment Overview” Report

Only available in Klaros-Testmanagement Enterprise Edition

The *Test Environment Overview Report* shows the success and progress rate of systems under test under a single test environment in a radar chart. If fewer than three systems under test are configured, a bar chart will be displayed instead.

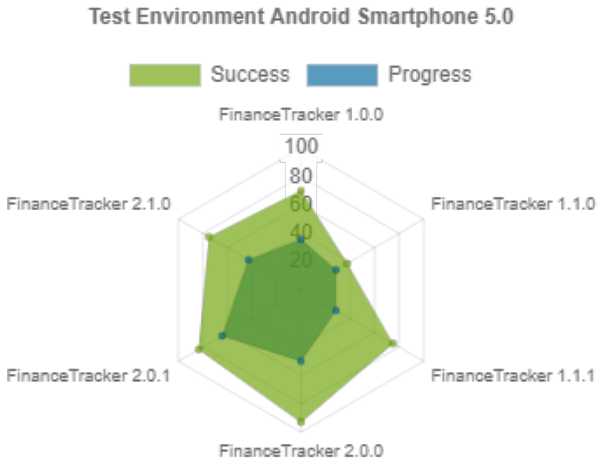


Figure 9.7. The “Test Environment Overview” Report

The following properties of the “Test Environment Overview” report diagram are configurable:

- Project
- Iteration (if available)
- One or more systems under test.
- A test environment.
- The name of the report diagram.

- The selection of whether to use the active project or another one in this report. (Klaros-Test-management Enterprise Edition only).
- If the active project is not used, the following objects to be viewed can be selected:
- The selection of whether to use the active iteration or another one in this report.
- If the active iteration is not used, a dropdown list with the available iterations is displayed here.

9.1.3.6. The “Test Progress” Report

Only available in Klaros-Testmanagement Enterprise Edition

The *Test Progress* report shows the rate of executed versus defined test cases of a project for a given set of test environments and test systems.

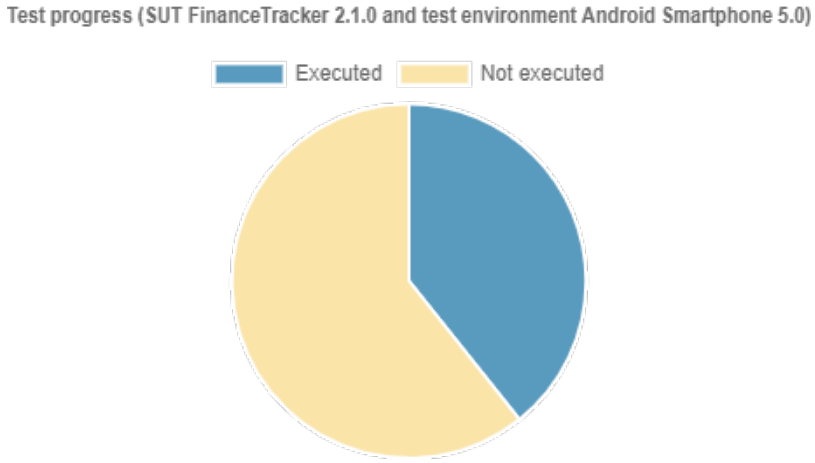


Figure 9.8. The “Test Progress” Report

The following properties of the “Test Progress” report diagram are configurable:

- Project
- Iteration (if available).
- One or more systems under test.
- One or more test environments.
- The name of the report diagram.
- The selection of whether to use the active project or another one in this report. (Klaros-Test-management Enterprise Edition only).
- If the active project is not used, the following objects to be viewed can be selected:
- The selection of whether to use the active iteration or another one in this report.

- If the active iteration is not used, a dropdown list with the available iterations is displayed here.
- A boolean value indicating that only test cases should be considered that are part of a job.

9.1.3.7. The “Test Progress History” Report

Only available in Klaros-Testmanagement Enterprise Edition

The *Test History* report shows the rate of defined test cases versus executed test cases for a project for one or more test environments and one or more systems under test in a given time period.

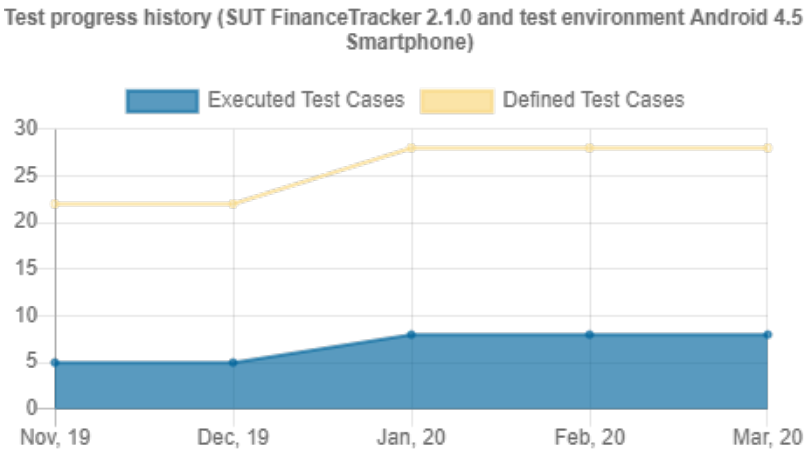


Figure 9.9. The “Test Progress History” Report

The following properties of the “Test Progress History” report diagram are configurable:

- Project
- Iteration (if available).
- One or more systems under test.
- One or more test environments.
- The name of the report diagram.
- The selection of whether to use the active project or another one in this report. (Klaros-Test-management Enterprise Edition only).
- If the active project is not used, the following objects to be viewed can be selected:
- The selection of whether to use the active iteration or another one in this report.
- If the active iteration is not used, a dropdown list with the available iterations is displayed here.
- The time period in days

9.2. Reports

On the *Reports* page, predefined reports can be generated as PDF or HTML files. The Klaros-Test-management Enterprise Edition also allows configurable reports to be created, customized and retrieved as PDF or Excel files.

9.2.1. Predefined Reports

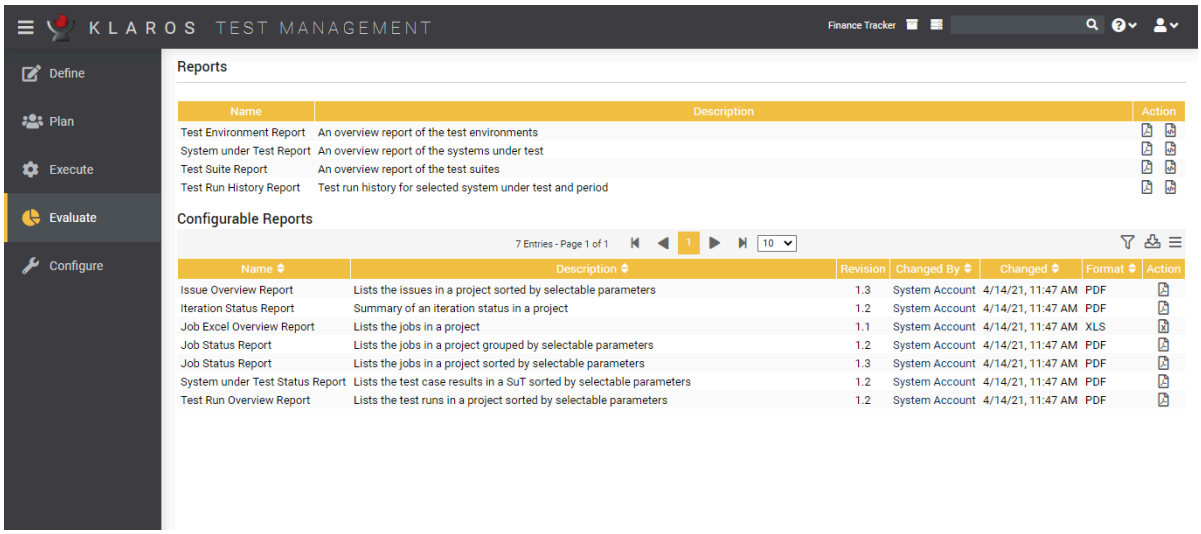


Figure 9.10. The “Reports” Page

The predefined reports are:

- The “Test Environment Report”
- The “System under Test Report”
- The “Test Suite Report”
- The “Test Run History Report”

The predefined reports can be generated in two different file formats:

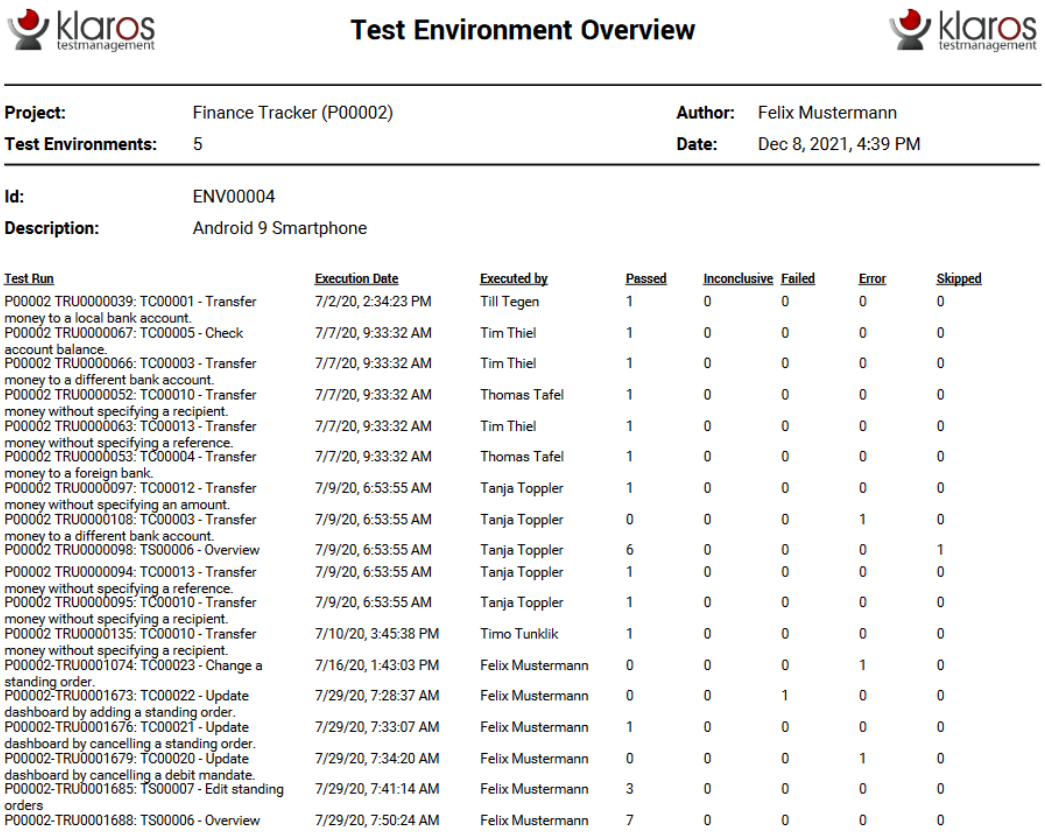
- PDF
- HTML

Clicking on one of the file type icon will generate the report in the selected format.

The upper part of the *Reports* page shows a table with the available predefined reports:

9.2.1.1. Test Environment Report

The *Test Environment Report* shows all test runs of a single test environment, including the execution date, executor and the quantity of *Passed*, *Failure*, *Error*, *Inconclusive* and *Skipped* results. An example of the report is shown in Figure 9.11.



Test Environment Overview

Project:	Finance Tracker (P00002)	**Author:**	Felix Mustermann
Test Environments:	5	**Date:**	Dec 8, 2021, 4:39 PM

Id: ENV00004

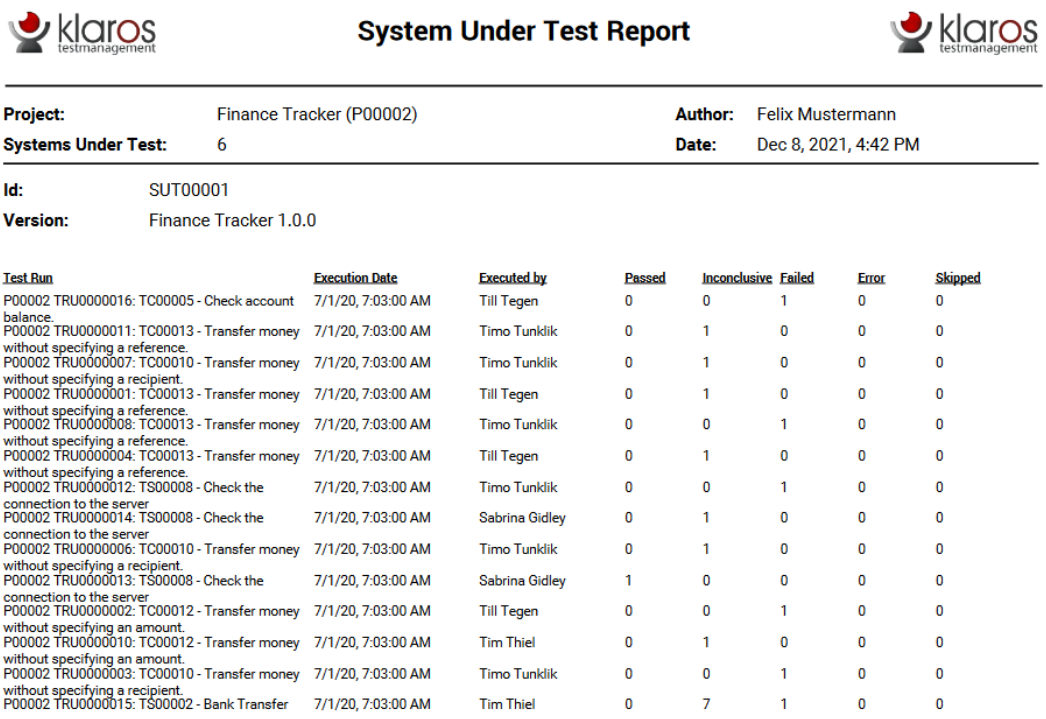
Description: Android 9 Smartphone

Test Run	Execution Date	Executed by	Passed	Inconclusive	Failed	Error	Skipped
P00002 TRU0000039: TC00001 - Transfer money to a local bank account.	7/2/20, 2:34:23 PM	Till Tegen	1	0	0	0	0
P00002 TRU0000067: TC00005 - Check account balance.	7/7/20, 9:33:32 AM	Tim Thiel	1	0	0	0	0
P00002 TRU0000066: TC00003 - Transfer money to a different bank account.	7/7/20, 9:33:32 AM	Tim Thiel	1	0	0	0	0
P00002 TRU0000052: TC00010 - Transfer money without specifying a recipient.	7/7/20, 9:33:32 AM	Thomas Tafel	1	0	0	0	0
P00002 TRU0000063: TC00013 - Transfer money without specifying a reference.	7/7/20, 9:33:32 AM	Tim Thiel	1	0	0	0	0
P00002 TRU0000053: TC00004 - Transfer money to a foreign bank.	7/7/20, 9:33:32 AM	Thomas Tafel	1	0	0	0	0
P00002 TRU0000097: TC00012 - Transfer money without specifying an amount.	7/9/20, 6:53:55 AM	Tanja Toppler	1	0	0	0	0
P00002 TRU0000108: TC00003 - Transfer money to a different bank account.	7/9/20, 6:53:55 AM	Tanja Toppler	0	0	0	1	0
P00002 TRU0000098: TS00006 - Overview	7/9/20, 6:53:55 AM	Tanja Toppler	6	0	0	0	1
P00002 TRU0000094: TC00013 - Transfer money without specifying a reference.	7/9/20, 6:53:55 AM	Tanja Toppler	1	0	0	0	0
P00002 TRU0000095: TC00010 - Transfer money without specifying a recipient.	7/9/20, 6:53:55 AM	Tanja Toppler	1	0	0	0	0
P00002 TRU0000135: TC00010 - Transfer money without specifying a recipient.	7/10/20, 3:45:38 PM	Timo Tunklik	1	0	0	0	0
P00002-TRU0001074: TC00023 - Change a standing order.	7/16/20, 1:43:03 PM	Felix Mustermann	0	0	0	1	0
P00002-TRU0001673: TC00022 - Update dashboard by adding a standing order.	7/29/20, 7:28:37 AM	Felix Mustermann	0	0	1	0	0
P00002-TRU0001676: TC00021 - Update dashboard by cancelling a standing order.	7/29/20, 7:33:07 AM	Felix Mustermann	1	0	0	0	0
P00002-TRU0001679: TC00020 - Update dashboard by cancelling a debit mandate.	7/29/20, 7:34:20 AM	Felix Mustermann	0	0	0	1	0
P00002-TRU0001685: TS00007 - Edit standing orders	7/29/20, 7:41:14 AM	Felix Mustermann	3	0	0	0	0
P00002-TRU0001688: TS00006 - Overview	7/29/20, 7:50:24 AM	Felix Mustermann	7	0	0	0	0

Figure 9.11. The “Test Environment” Report

9.2.1.2. System under Test Report

The *System under Test Report* shows all test runs of a single system under test, including the execution date, executor and the quantity of *Passed*, *Failure*, *Error*, *Inconclusive* and *Skipped* results. An example of the report is shown in Figure 9.12.



System Under Test Report

Project: Finance Tracker (P00002) **Author:** Felix Mustermann
Systems Under Test: 6 **Date:** Dec 8, 2021, 4:42 PM

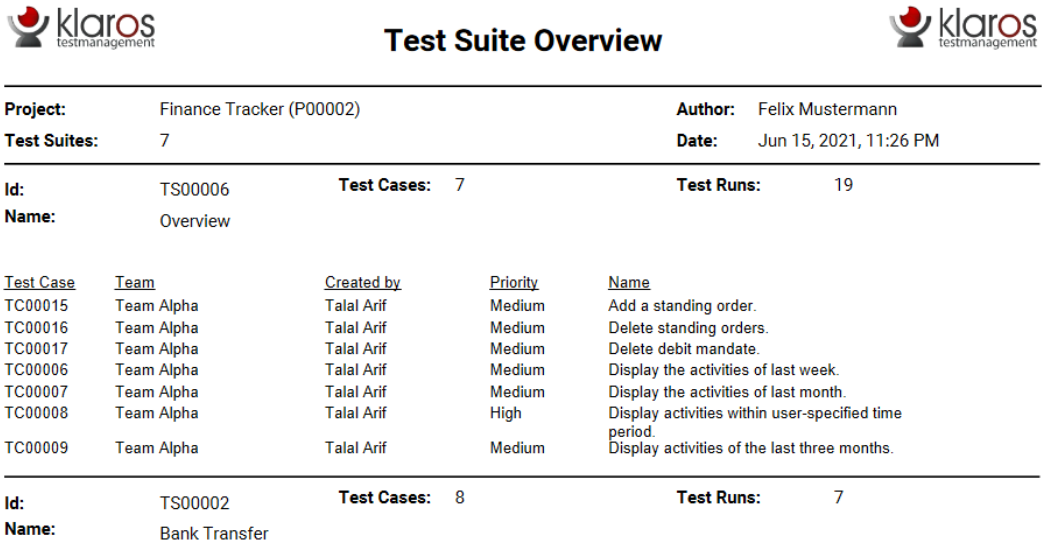
Id: SUT00001
Version: Finance Tracker 1.0.0

Test Run	Execution Date	Executed by	Passed	Inconclusive	Failed	Error	Skipped
P00002 TRU0000016: TC00005 - Check account balance.	7/1/20, 7:03:00 AM	Till Tegen	0	0	1	0	0
P00002 TRU0000011: TC00013 - Transfer money without specifying a reference.	7/1/20, 7:03:00 AM	Timo Tunklik	0	1	0	0	0
P00002 TRU0000007: TC00010 - Transfer money without specifying a recipient.	7/1/20, 7:03:00 AM	Timo Tunklik	0	1	0	0	0
P00002 TRU0000001: TC00013 - Transfer money without specifying a reference.	7/1/20, 7:03:00 AM	Till Tegen	0	1	0	0	0
P00002 TRU0000008: TC00013 - Transfer money without specifying a reference.	7/1/20, 7:03:00 AM	Timo Tunklik	0	0	1	0	0
P00002 TRU0000004: TC00013 - Transfer money without specifying a reference.	7/1/20, 7:03:00 AM	Till Tegen	0	1	0	0	0
P00002 TRU0000012: TS00008 - Check the connection to the server	7/1/20, 7:03:00 AM	Timo Tunklik	0	0	1	0	0
P00002 TRU0000014: TS00008 - Check the connection to the server	7/1/20, 7:03:00 AM	Sabrina Gidley	0	1	0	0	0
P00002 TRU0000006: TC00010 - Transfer money without specifying a recipient.	7/1/20, 7:03:00 AM	Timo Tunklik	0	1	0	0	0
P00002 TRU0000013: TS00008 - Check the connection to the server	7/1/20, 7:03:00 AM	Sabrina Gidley	1	0	0	0	0
P00002 TRU0000002: TC00012 - Transfer money without specifying an amount.	7/1/20, 7:03:00 AM	Till Tegen	0	0	1	0	0
P00002 TRU0000010: TC00012 - Transfer money without specifying an amount.	7/1/20, 7:03:00 AM	Tim Thiel	0	1	0	0	0
P00002 TRU0000003: TC00010 - Transfer money without specifying a recipient.	7/1/20, 7:03:00 AM	Timo Tunklik	0	0	1	0	0
P00002 TRU0000015: TS00002 - Bank Transfer	7/1/20, 7:03:00 AM	Tim Thiel	0	7	1	0	0

Figure 9.12. The “System under Test” Report

9.2.1.3. Test Suite Report

The *Test Suite Report* shows all test cases including team, creator, priority and name, for each test suite in the active project. An example report is shown in Figure 9.13.



Test Suite Overview

Project: Finance Tracker (P00002) **Author:** Felix Mustermann
Test Suites: 7 **Date:** Jun 15, 2021, 11:26 PM

Id: TS00006 **Test Cases:** 7 **Test Runs:** 19
Name: Overview

Test Case	Team	Created by	Priority	Name
TC00015	Team Alpha	Talal Arif	Medium	Add a standing order.
TC00016	Team Alpha	Talal Arif	Medium	Delete standing orders.
TC00017	Team Alpha	Talal Arif	Medium	Delete debit mandate.
TC00006	Team Alpha	Talal Arif	Medium	Display the activities of last week.
TC00007	Team Alpha	Talal Arif	Medium	Display the activities of last month.
TC00008	Team Alpha	Talal Arif	High	Display activities within user-specified time period.
TC00009	Team Alpha	Talal Arif	Medium	Display activities of the last three months.

Id: TS00002 **Test Cases:** 8 **Test Runs:** 7
Name: Bank Transfer

Figure 9.13. The “Test Suite Overview” Report

9.2.1.4. Test Run History Report

The *Test Run History Report* shows all test runs of a single system under test, including the execution date, executor and the quantity of *Passed*, *Failure*, *Error*, *Inconclusive* and *Skipped* results in a given time period.

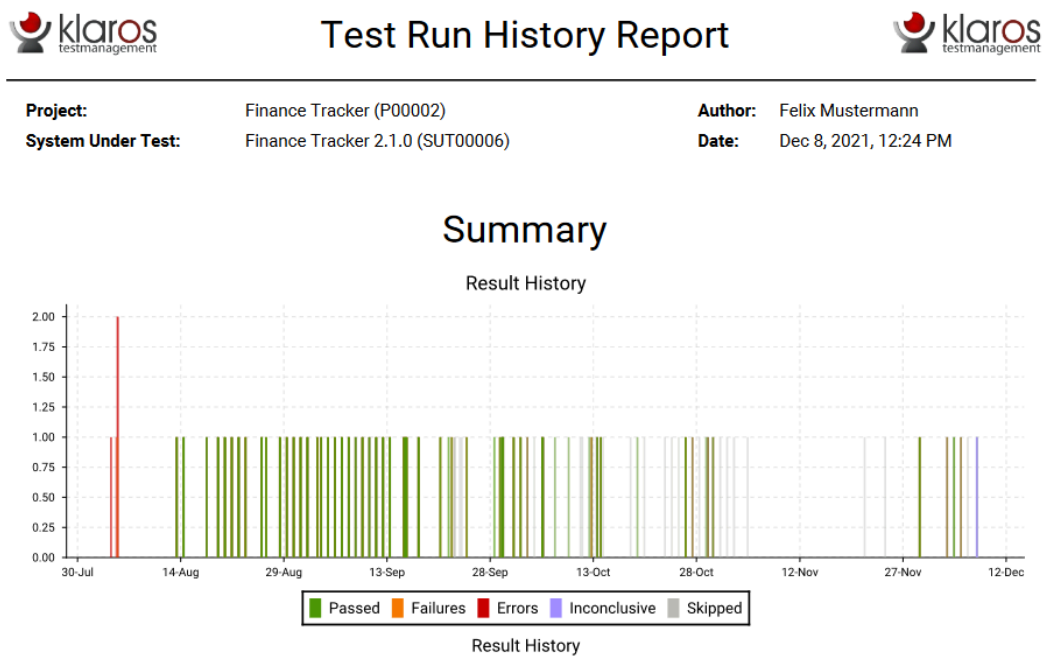


Figure 9.14. The "Test Run History" Report

9.2.1.5. Configurable Reports

Only available in Klaros-Testmanagement Enterprise Edition

In addition to predefined reports, the Klaros-Testmanagement Enterprise Edition can create PDFs and Excel files from user-defined reports. Creating user-defined reports is explained in Section 10.2, "Report Templates". Clicking on the and icons will generate the reports.

Note

Clicking on one of the icons or opens a dialog where report parameters can be set if any are needed for generating.

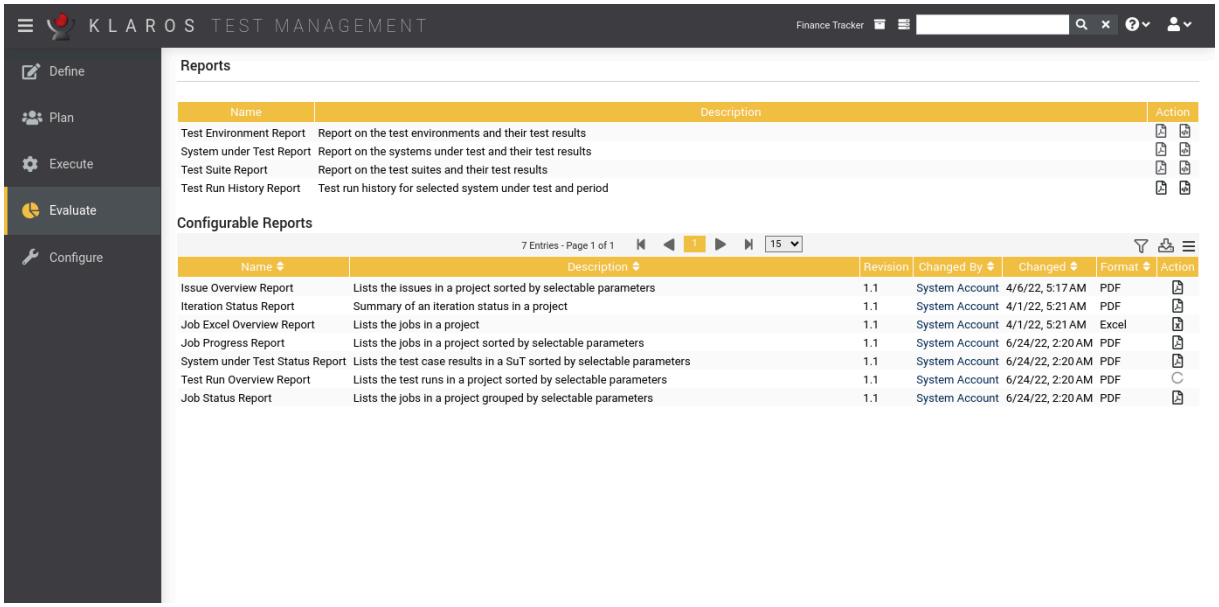


Figure 9.15. Generate a Parameterized Report

Note

The OK button becomes active after entering all required parameters.

9.3. Test Runs

This page lists all test runs that have been executed in the selected project.

Note

Only fully executed test cases are displayed on this page. Suspended test runs can be continued at Section 8.4, “Continue Test Run”.

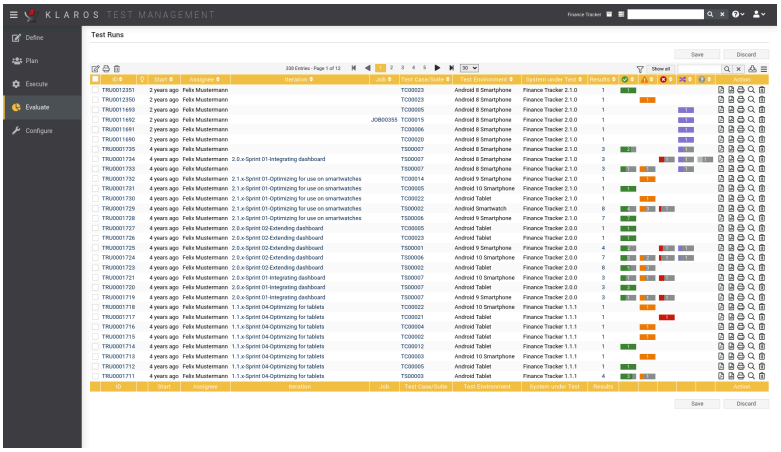


Figure 9.16. The “Test Runs” Page

The table shows the following values:

ID	The automatically assigned ID.
Additional Information	A tooltip appears when the cursor is placed over the icon shown here.
Start	The start time of the test run.
Assignee	The name of the user to whom the test run is assigned.
Iteration	The iteration in which the test run was executed.
Job	The task with which the test run was executed.
Test Case/Suite	The ID of the test case or test suite of the test run.
Test Environment	The test environment in which the test was executed.
System Under Test	The system under test that was tested.
Results	The number of test case results.
Passed	The number of test case results with the status/verdict "Passed".
Failed	The number of test case results with the status/verdict "Failed".
Error	The number of test case results with the status/verdict "Error".
Inconclusive	The number of test case results with the status/verdict "Inconclusive".
Skipped	The number of test case results with the status/verdict "Skipped".
Action	The executable actions.

9.3.1. Action

The following actions can be performed in the action column:

- Generate a test run report in PDF format.
- Generate a test run report in HTML format.
- Open print view.
- Show the test case results of this test run.
- Delete this test run.

If a test run has been deleted, it is initially marked with a deletion marker and is only visible to administrators. For deleted test runs, the following actions are available instead of *Delete*:

Restore this test run.

Irrecoverably remove the test run from the database (only Administrator).

Deleting a Test Run

When a test run is deleted, the associated test case results and test suite results are also deleted

Once all test runs for a specific system under test or test environment have been deleted, the respective system under test or test environment may also be deleted. Otherwise, they remain locked for deletion.

9.3.2. Bulk Actions

Certain actions can also be applied to several test runs at once. To do this, select the test runs to which the action is to be applied in the leftmost column.

The following bulk actions are supported for test runs:

Edit

Open print view.

Delete

For additional information, see Section 5.2.3.1.5, “Bulk Actions”.

9.3.3. Table Operations

The following operations can be performed in the line above the table on the right:

Filter / Sort

Show all / Only active (only Administrator)

Search

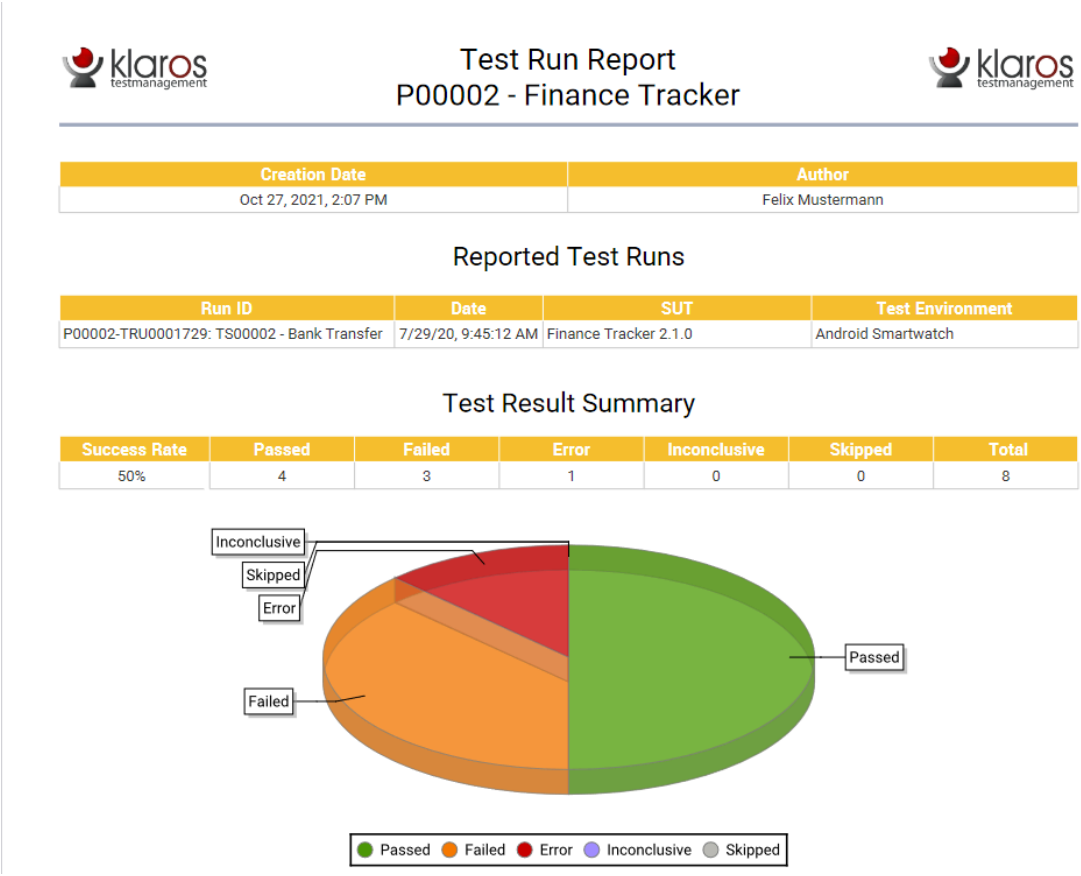
Export

Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

9.3.4. The Test Run Report

The “Test Run Report” shows the details of a test run including its execution date, executor, system under test, test environment and the number of *Passed*, *Failure*, *Error*, *Inconclusive* and *Skipped* results. It can be generated in PDF or HTML format. An example PDF is shown in Figure 9.17.



Test Run Report
P00002 - Finance Tracker

Creation Date	Author
Oct 27, 2021, 2:07 PM	Felix Mustermann

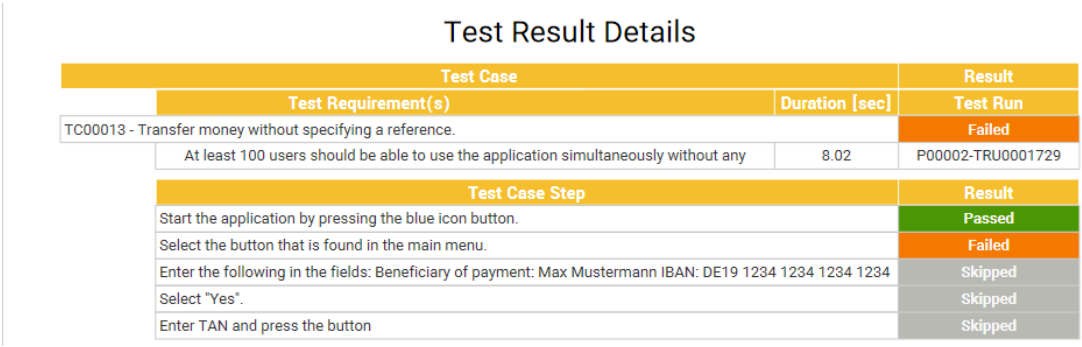
Reported Test Runs

Run ID	Date	SUT	Test Environment
P00002-TRU0001729: TS00002 - Bank Transfer	7/29/20, 9:45:12 AM	Finance Tracker 2.1.0	Android Smartwatch

Test Result Summary

Success Rate	Passed	Failed	Error	Inconclusive	Skipped	Total
50%	4	3	1	0	0	8

Figure 9.17. The “Test Run” Report



Test Result Details

Test Case		Result
Test Requirement(s)	Duration [sec]	Test Run
TC00013 - Transfer money without specifying a reference.		Failed
At least 100 users should be able to use the application simultaneously without any	8.02	P00002-TRU0001729

Test Case Step	Result
Start the application by pressing the blue icon button.	Passed
Select the button that is found in the main menu.	Failed
Enter the following in the fields: Beneficiary of payment: Max Mustermann IBAN: DE19 1234 1234 1234 1234	Skipped
Select "Yes".	Skipped
Enter TAN and press the button	Skipped

Figure 9.18. The “Test Run” Report (continued)

9.3.5. Details Page

Each test run has its own detail page. By clicking on the ID of the respective test run or on the in the action column, you can access the detail page. All test case results and, if available, the test suite result of the test run are displayed in tables here.

The following tabs are available: *Properties*, *User Defined* and *Results*. At first invocation this is the *Results* tab with the sub tabs *Test Case Results* and *Test Suite Results*.

9.3.5.1. Actions

The action toolbar is located in the top right corner of the title bar The following actions can be performed on the detail pages:

Open execute view — Re-execute this test run.

Open print view — A print-ready view of the iteration can be created here. With a click on the icon this opens in a new browser tab.

Print views are described in detail in Section 5.2.3.2.1, “Print Pages”.

Create Bookmarks — Each individual detail page can also be reached directly via a hyperlink. By clicking on the icon this link is copied to the clipboard.

The creation of bookmarks is described in detail in Section 5.2.3.2.2, “Bookmarks”.

Browse — Use the green arrows at the very top right to switch between the properties present on the previous page.

9.3.5.2. Properties

The *Properties* tab shows the following information:

ID — The automatically assigned ID.

Executor — The user who executed this test run.

System under Test — A drop-down menu for selecting the assigned system under test.

Test Environment — A drop-down menu for selecting the assigned test environment.

Iteration — The iteration in which the test run was executed.

Assignee — The user to whom this test run is assigned.

* A test manager or administrator can retroactively correct the entries here. This may be necessary, for example, in the case of an incorrect entry when starting the test run.

9.3.5.3. User Defined

You can create your own fields to meet individual requirements. For further information please refer to Section 5.2.3.2.4, “User Defined Properties”.

9.3.5.4. Results

The *Results* tab splits into the *Test Case Results* subviews. and *Test Suite Results*. The table displays the following values:

ID — The automatically assigned ID.

Additional Information — A tooltip appears when the cursor is placed over the icon shown here.

Start	The start time of the test run.
Iteration	The iteration in which the test run was executed.
Test Case	The test case.
Test Environment	The test environment in which the test was executed.
System Under Test	The system under test that was tested.
Executor	The name of the user who last executed the test run.
Duration	The execution time.
Result	The test case result (Passed, Failed, Error, Inconclusive, Skipped).
Action	The executable actions.

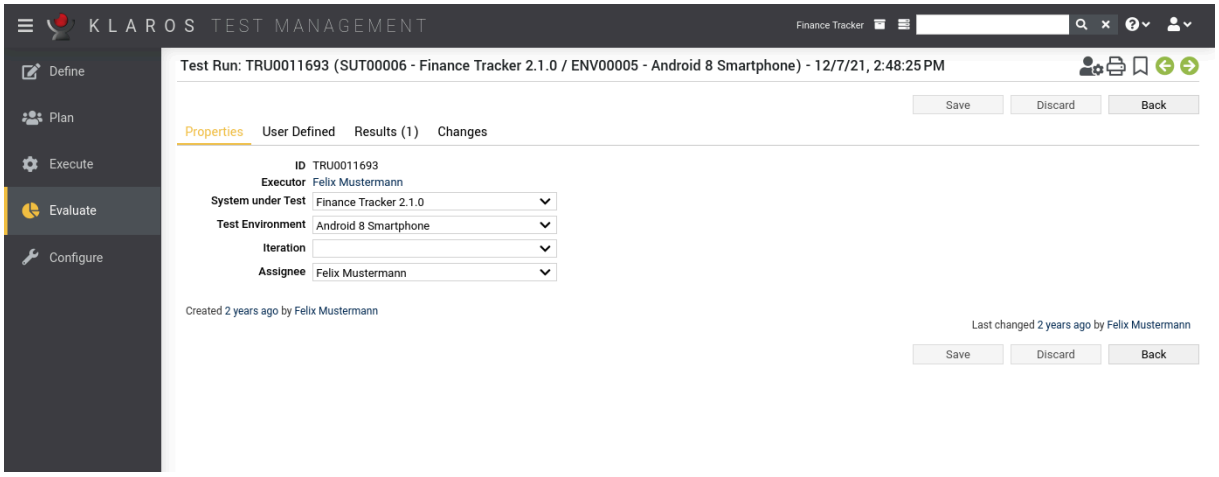


Figure 9.19. The “Test Run Details” Page

9.3.6. Changes

The tab *Changes* shows the change history of this test run.

For a detailed description of the *Changes* view, see Section 5.2.3.2.8, “Change History”.

9.4. Test Case Results

The *Test Case Results* page shows an overview of all test cases, including the quantity of *Passed*, *Failure*, *Error*, *Inconclusive* and *Skipped* results.

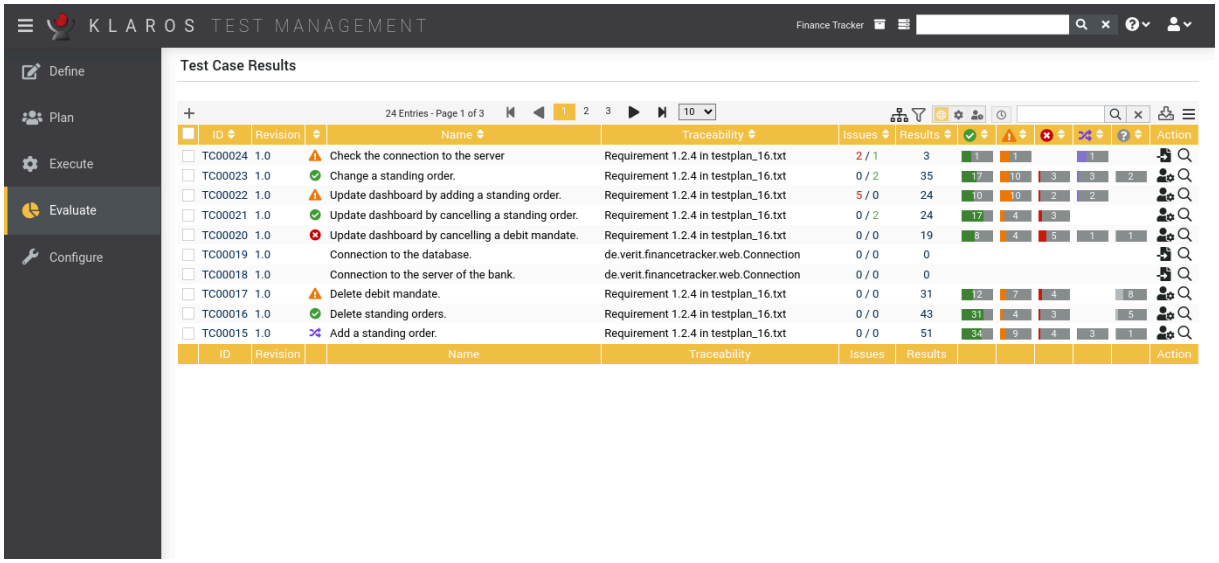


Figure 9.20. The “Test Case Results” Page

The table shows the following values:

ID — The automatically assigned ID.

Revision — The revision of the test case.

Additional Information — A tooltip appears when the cursor is placed over the icon shown here.

Name — The name of the test case.

Traceability — The requirement assigned to this test case.

Issues — The number of open and resolved issues.

Results — The number of test case results.

Passed — The number of test case results with the status "Passed".

Failed — The number of test case results with the status/verdict "Failed".

Error — The number of test case results with the status/verdict "Error".

Inconclusive — The number of test case results with the status/verdict "Inconclusive".

Skipped — The number of test case results with the status/verdict "Skipped".

Action — The executable actions.

Note

The *Result* column shows the latest result of the test cases.

9.4.1. Action

The following actions can be performed in the action column:

- Re-execute the test case.
- Import results.
- Show the test case results of this test case (see Section 9.4.4, “Single Test Case Results”).

9.4.2. Bulk Actions

Certain actions can also be applied to several test cases at once. To do this, select the test cases to which the action is to be applied in the leftmost column.

The following bulk actions are supported for test cases:

- Add test cases to a test suite (see Figure 9.21).

For additional information, see Section 5.2.3.1.5, “Bulk Actions”.

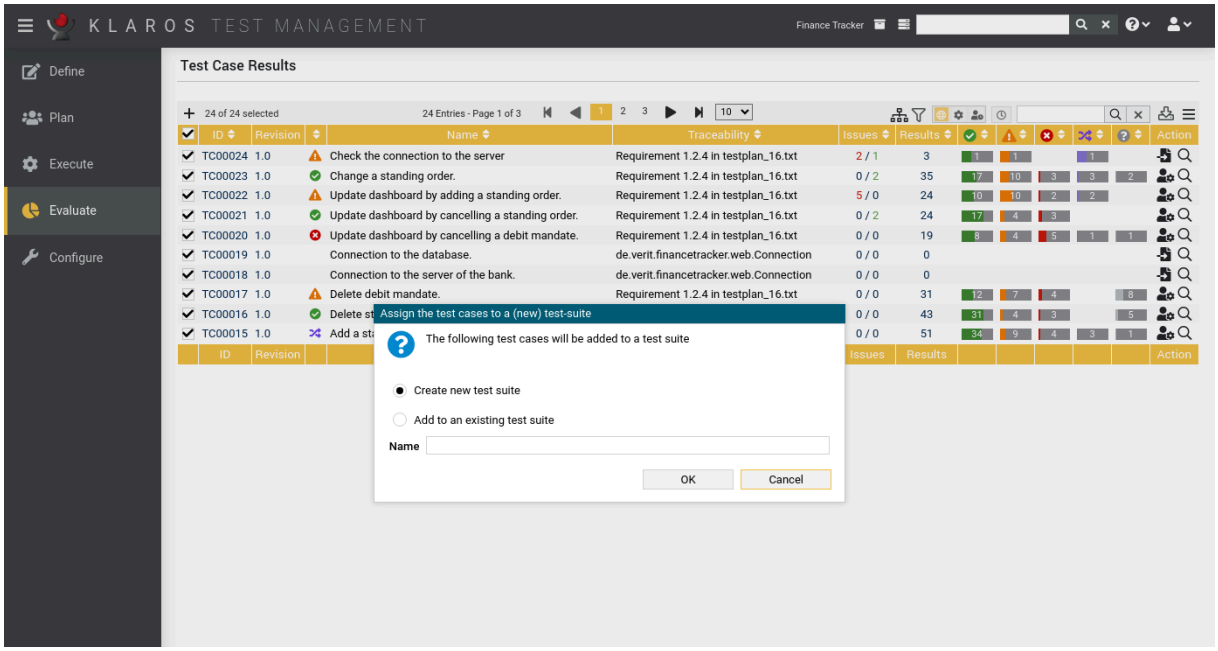


Figure 9.21. The “Add test cases to test suite” Dialog

9.4.3. Table Operations

The following operations can be performed in the line above the table on the right:

- Filter / Sort

Show Executed/All Test Cases

Show Newest Only/Show All

- Search
- Export
- Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

9.4.4. Single Test Case Results

The “Test Case Results” page shows all executions of the selected test case, including the result, the used system under test and the test environment (Figure 9.22).

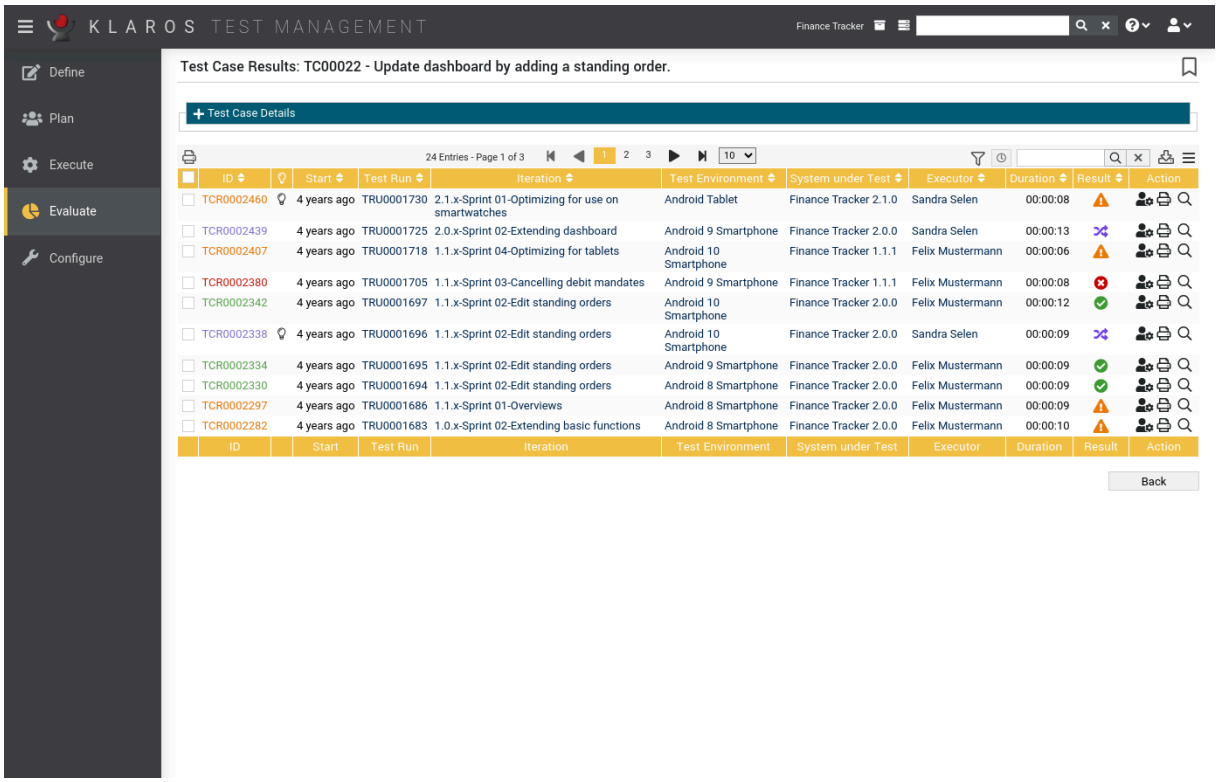


Figure 9.22. The “Test Case Results” Page

The collapsible panel on top of the page shows the details of the selected test case. Below it is a table with all results.

The table shows the following values:

ID — The automatically assigned ID.

Additional Information — A tooltip appears when the cursor is placed over the icon shown here.

Start — The start time of the test run.

Test Run — The test run, to which the result belongs.

Iteration — The iteration in which the test run was executed.

Test Environment	The test environment in which the test was executed.
System Under Test	The system under test that was tested.
Executor	The name of the user who last executed the test run.
Duration	The execution time.
Result	The test case result (Passed, Failed, Error, Inconclusive or Skipped).
Action	The executable actions.

The icon in the *Additional Information* column shows whether the corresponding test case result has been manually changed by a user.

Clicking the Back button navigates back to the test case results page (Figure 9.20).

9.4.4.1. Action

The following actions can be performed in the action column:

Re-execute this test case.

Import results.

Show the detail page of the test case result (see Section 9.4.4.4, “The “Test Case Result” Details Page ”).

9.4.4.2. Bulk Actions

Certain actions can also be applied to several test case results at once. To do this, select the test case results to which the action is to be applied in the leftmost column.

The following bulk actions are supported for test case results:

Open print view

For additional information, see Section 5.2.3.1.5, “Bulk Actions”.

9.4.4.3. Table Operations

The following operations can be performed in the line above the table on the right:

Filter / Sort

Show Newest Only/Show All

Search

Export

Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

9.4.4.4. The “Test Case Result” Details Page

This page shows a single test case result and its test case step results.

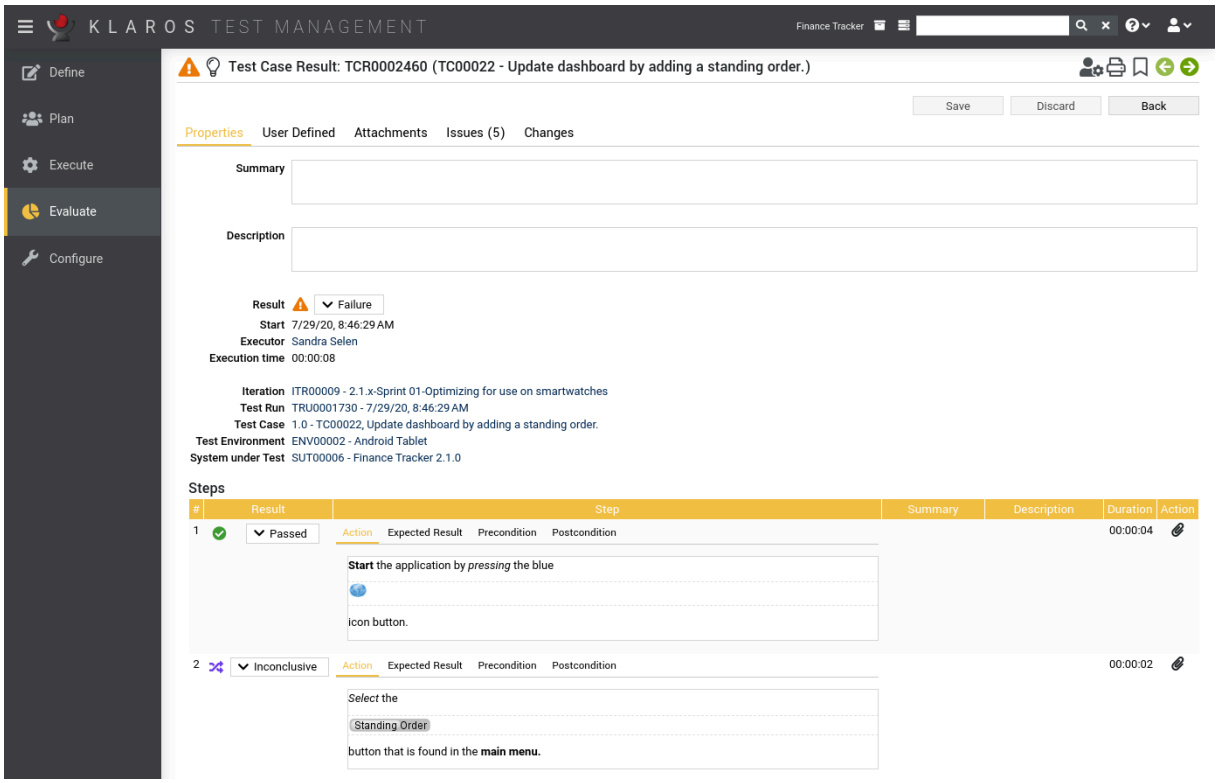


Figure 9.23. The “Test Case Result” Page

9.4.4.4.1. Actions

Auf den Detailseiten lassen sich jeweils folgende Aktionen vornehmen:

Execute this single test case/suite — Re-execute the test case or the test suite.

Execute this single test case/suite — Re-execute the test case or the test suite.

Open print view — A print-ready view of the test case result can be created here. With a click on the icon this opens in a new browser tab.

Print views are described in detail in Section 5.2.3.2.1, “Print Pages”.

Create Bookmarks — Requirements may be linked to from outside Klaros-Test-management using the link on the icon. By clicking on the icon this link will be copied to the clipboard.

Weitere Informationen hierzu finden Sie unter Section 5.2.3.2.2, “Bookmarks”.

Browse Use the green arrows at the very top right to switch between the test case results present on the previous page.

The *Summary*, *Description* and *Result* of the test case result and each individual test step result can be edited here. All changes are automatically logged and any change of a result will cause a recalculation of the corresponding test case result, test suite result and test run. If a test case result has been changed, the icon is displayed on this page and in all tables where the result is displayed. The *Result* field of a test case result or test step result may only be changed in Klaros-Testmanagement Enterprise Edition.

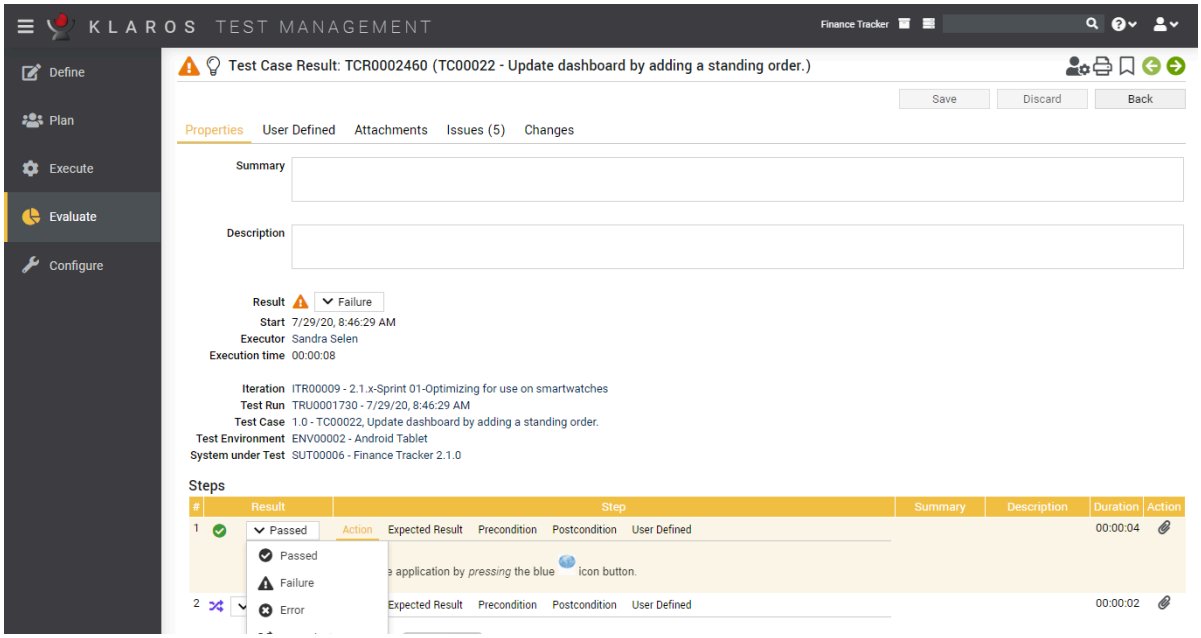


Figure 9.24. Editing a Test Step Result

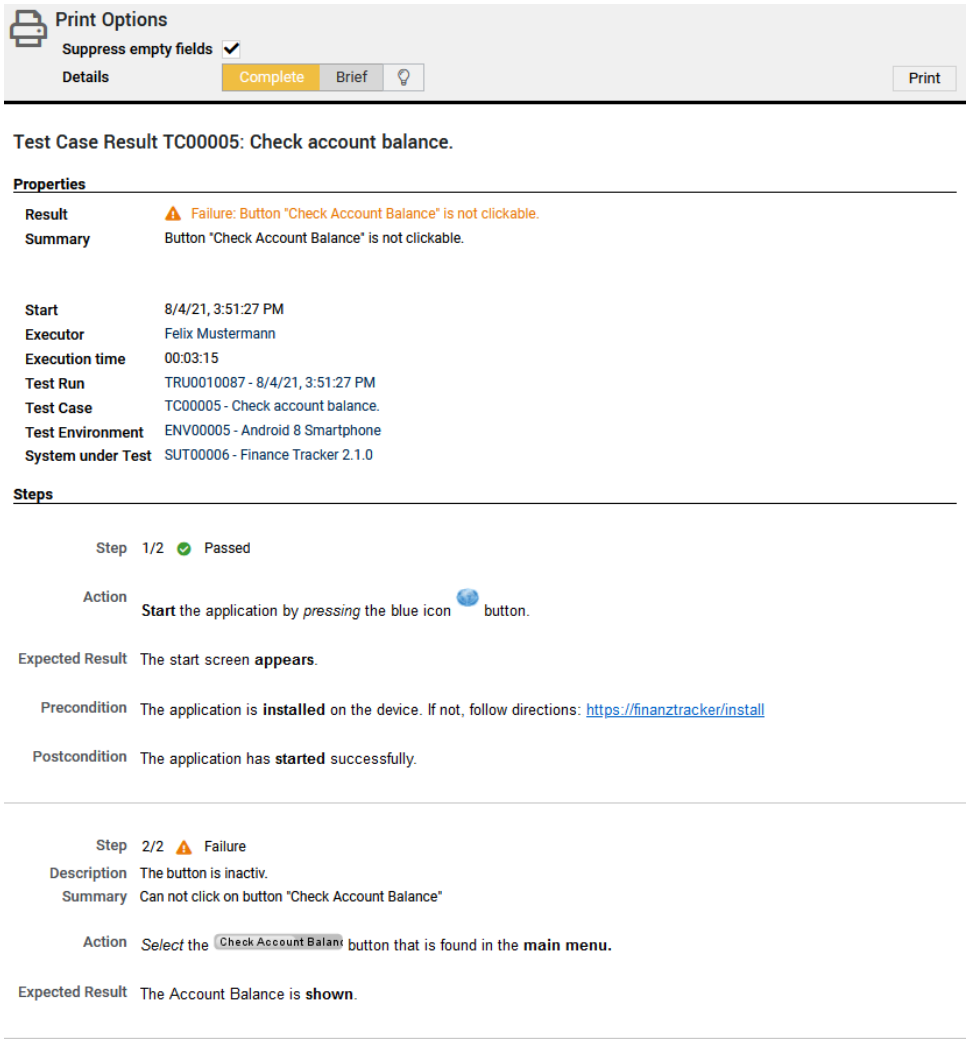


Figure 9.25. The “Test Case Result” Print Page

9.4.4.5. Reasons for Skipped Test Cases

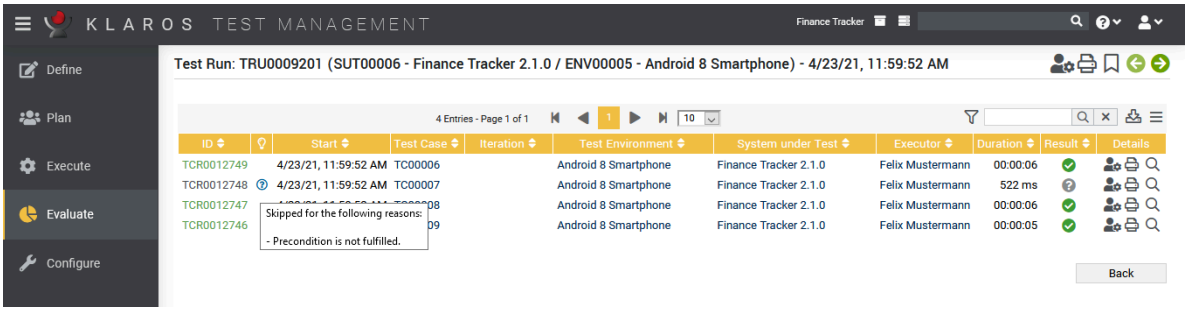


Figure 9.26. Viewing the Reason for Skipping a Test Case

If a test case has been skipped, the tester can enter one or more reasons from predefined templates (see Section 10.4.3, “Test Execution” and Section 8.3.3.1.1, “Skip Test Cases Permanently”). The icon ⓘ will be displayed on appropriate pages of the *Evaluate* section. Hovering over this icon will display the predefined reasons in a tooltip.

9.5. Test Suite Results

The *Test Suite Results* page shows an overview of all test suites, including the number of results.

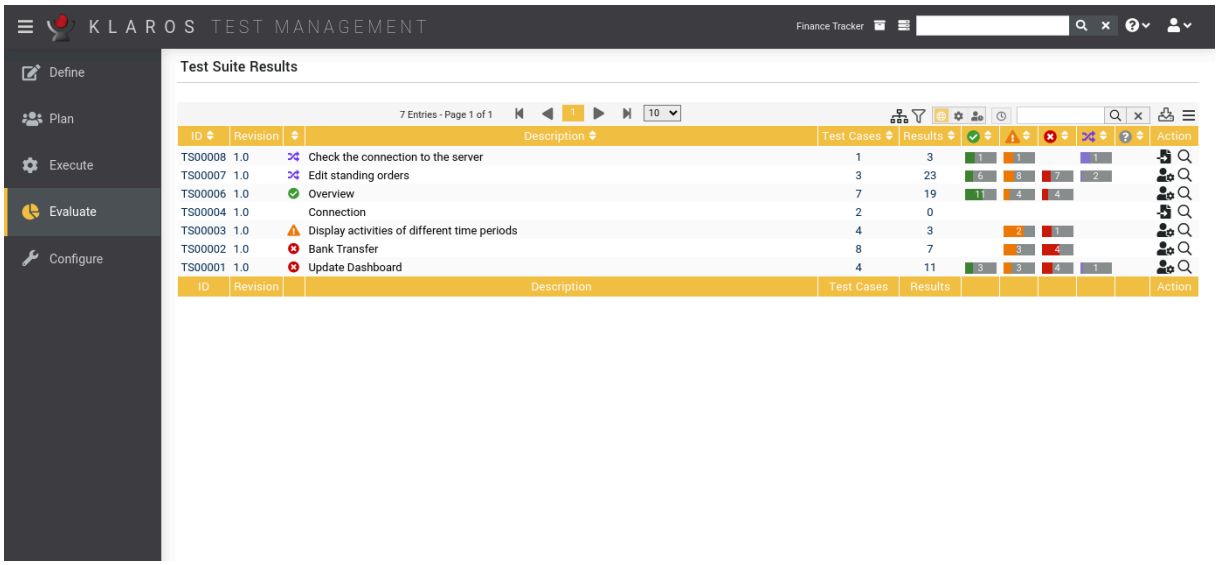


Figure 9.27. The “Test Suite Results” Page

The table shows the following values:

ID	The automatically assigned ID.
Revision	The revision of the test suite.
Additional Information	A tooltip appears when the cursor is placed over the icon shown here.
Description	The description of the test suite.
Test Cases	The number of test cases.
Results	The number of test suite results.
Action	The executable actions.

9.5.1. Action

The following actions can be performed in the action column:

- Re-execute the test suite.
- Import results.
- Show the test suite results of this test suite (see Section 9.5.3, “Single Test Suite Results”).
- Delete

9.5.2. Table Operations

The following operations can be performed in the line above the table on the right:

Filter / Sort

Show Executed Only/Show All Test Suites

Search

Export

Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

9.5.3. Single Test Suite Results

The “Test Suite Results” page shows all executions of the selected test suite, including the result, the used system under test and the test environment (Figure 9.28).

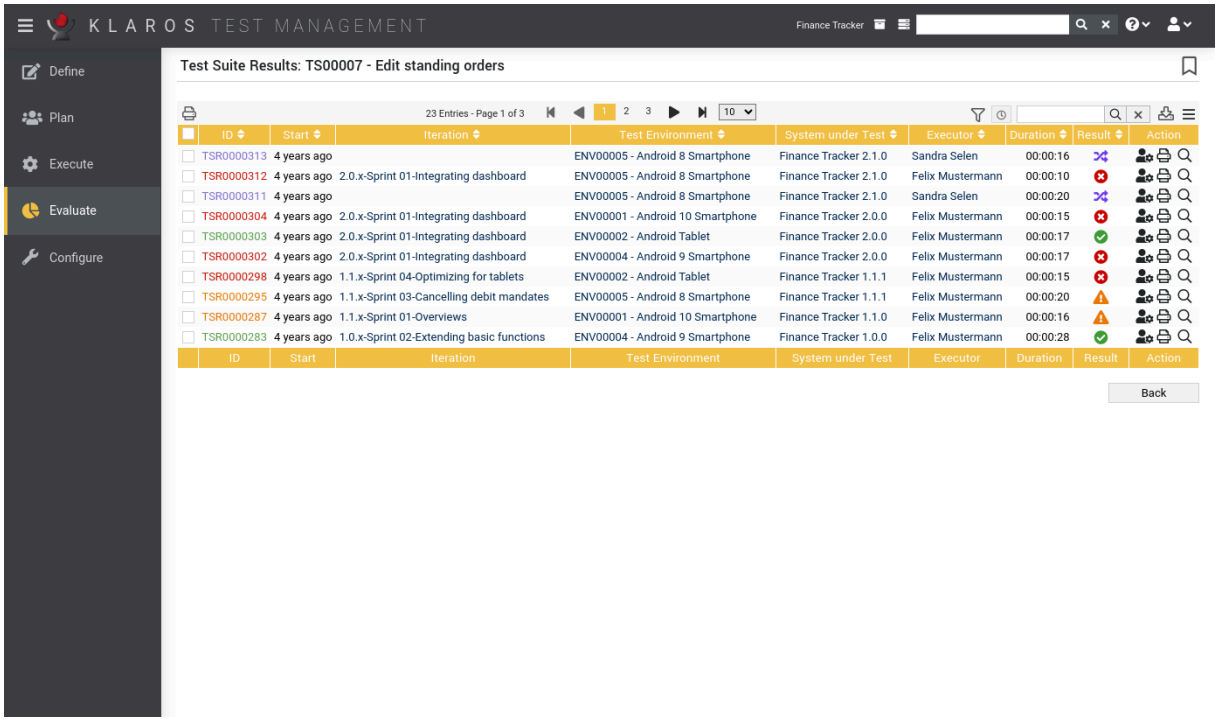


Figure 9.28. The “Test Suite Results” Page for a Single Test Suite

The collapsible panel on top of the page shows the details of the selected test suite. Below it is a table with all results.

The table shows the following values:

ID The automatically assigned ID.

Start The start time of the test run.

Test Environment The test environment in which the test was executed.

System Under Test The system under test that was tested.

Executor	The name of the user who last executed the test run.
Duration	The execution time.
Result	The test case result (Passed, Failed, Error, Inconclusive or Skipped).
Action	The executable actions.

9.5.3.1. Action

The following actions can be performed in the action column:

- Re-execute this test suite.
- Import results.
- Show the detail page of the test suite result (see Section 9.5.3.4, “The “Test Suite Result” Details Page ”).
- Delete

9.5.3.2. Bulk Actions

Certain actions can also be applied to several test suite results at once. To do this, select the test suite results to which the action is to be applied in the leftmost column.

The following bulk actions are supported for test suite results:

- Open print view
- Delete

For additional information, see Section 5.2.3.1.5, “Bulk Actions”.

9.5.3.3. Table Operations

The following operations can be performed in the line above the table on the right:

- Filter / Sort
- Search
- Export
- Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

9.5.3.4. The “Test Suite Result” Details Page

This page shows a single test suite result and its test case results.

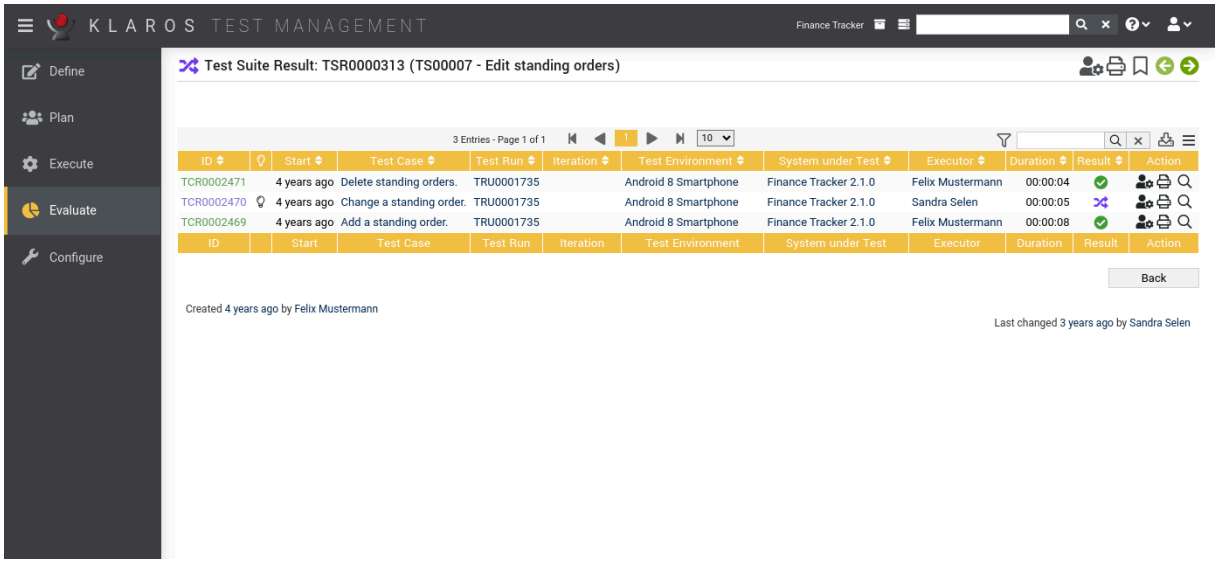


Figure 9.29. The “Test Suite Result” Details Page

9.5.3.4.1. Actions

Auf den Detailseiten lassen sich jeweils folgende Aktionen vornehmen:

Execute this single test suite Re-execute the test suite.

Execute this single test suite Re-execute the test suite.

Open print view A print-ready view of the test suite result can be created here. With a click on the icon this opens in a new browser tab.

Print views are described in detail in Section 5.2.3.2.1, “Print Pages”.

Create Bookmarks Requirements may be linked to from outside Klaros-Test-management using the link on the icon. By clicking on the icon this link will be copied to the clipboard.

Weitere Informationen hierzu finden Sie unter Section 5.2.3.2.2, “Bookmarks”.

Browse Use the green arrows at the very top right to switch between the test suite results present on the previous page.

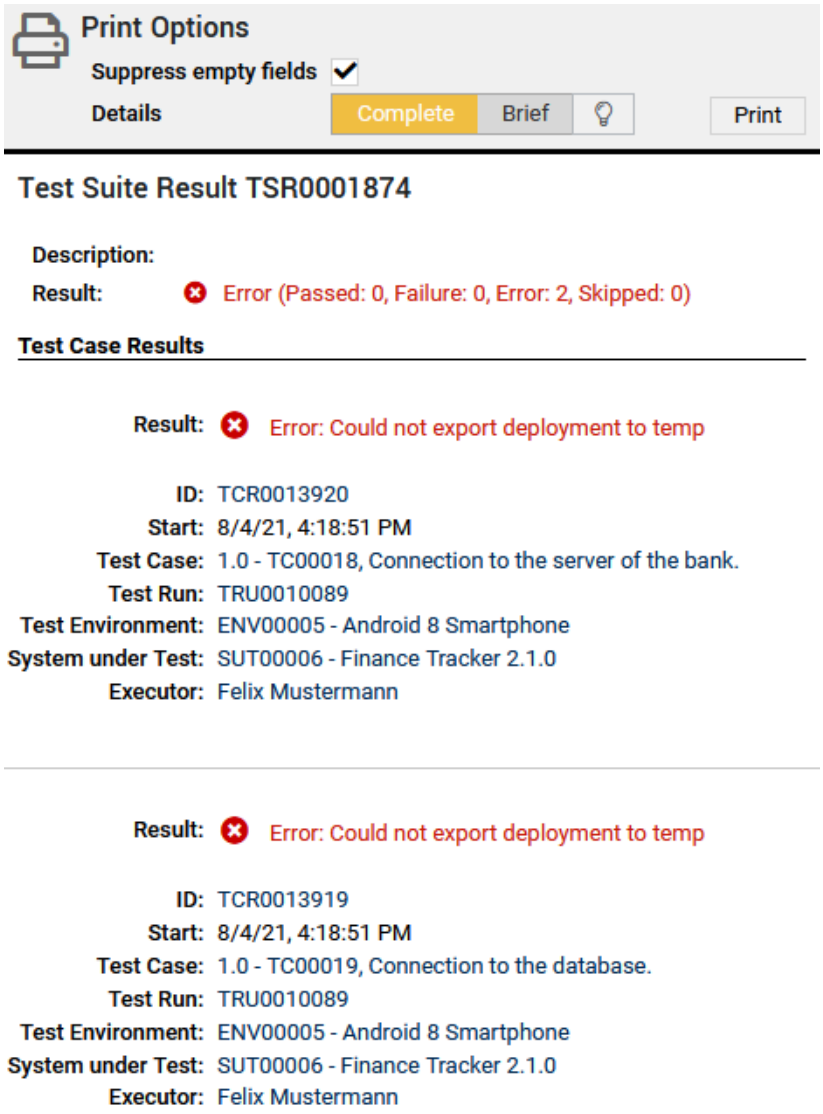


Figure 9.30. The “Test Suite Result” Print View

9.6. Issues

Issues from external issue management systems (like Jira or GitLab) can be linked with test cases and systems under test in Klaros-Testmanagement. In addition to this, issues can be created from Klaros-Testmanagement in connected issue management systems (even during test execution).

For this, at least one issue management system must be configured and assigned to the desired project. The configuration of an issue management system is described in Section 10.5.1, “Issue Management”, Section 6.1.2.6.1, “Issue Management” shows how to assign the issue management system to a project.

9.6.1. Overview Page

The overview page displays all issues present in the project in a table. This chapter shows how to create, edit and edit issues.

Pressing the New button opens up the Section 9.6.6, “Issue Details (Creating a new Issue)” page.

Pressing the Link button opens up the Link Issues panel, in which existing issues from issue management systems can be linked to test cases and systems under test (see Section 9.6.7, “Link Issue”).

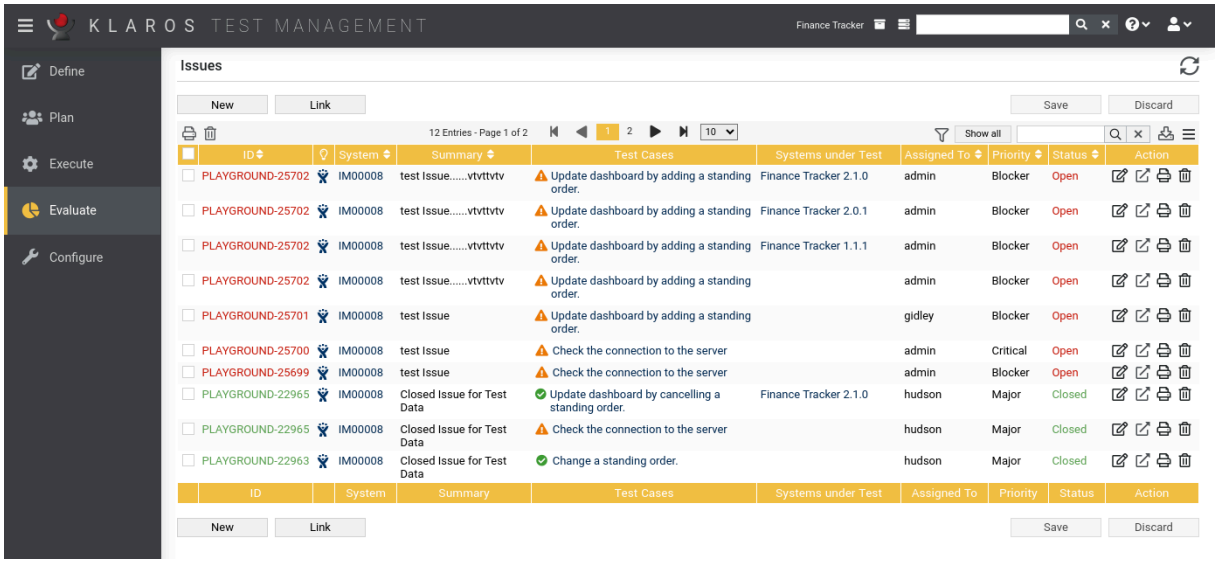


Figure 9.31. The “Issues” Page

The table shows the following values:

ID — The external issue Id.

Additional Information — On the one hand, the symbol of the issue management system is displayed here. Clicking this icon opens the issue in the issue management system. On the other hand, a ⚠ can appear. In this case, the administrator should be informed immediately. Reasons for displaying the ⚠ can be among others:

- Deactivation of the issue management system.
- Incorrect configuration of the issue management system in Klaros-Testmanagement or in the issue management system. The configuration of an issue management system is described in Section 10.5.1, “Issue Management”.

A tooltip appears when the cursor is placed over the icon shown here.

System — The ID of the issue management system.

Summary — The summary of the issue.

Test Cases — A list of test cases assigned to this issue with the result of their latest execution.

Systems under Test — A list of systems under test assigned to this issue.

Assigned To — The username of the responsible user.

Priority — The priority of the issue.

State — The state of the issue.

Action The executable actions.

9.6.2. Action

The following actions can be performed in the action column:

- Edit
- Open issue in external issue management system
- Open print view
- Delete

If a issue has been deleted, it is internally marked with a deletion marker and is only visible to administrators. For deleted issues, the following actions are available instead of *Delete*:

- Restore
- Remove the issue from the database

Important

Pressing the icon does not delete the issue in the issue management system, only the reference in the database!

9.6.3. Bulk Actions

On the *Issue* page, one or more issues can be selected for bulk actions. Bulk actions are described in Section 5.2.3.1.5, “Bulk Actions”.

The following bulk actions are supported for issues:

- Open print view
- Delete
- Restore the issue (only Administrator).
- Remove the issue from the database (only Administrator).

For additional information, see Section 5.2.3.1.5, “Bulk Actions”.

9.6.4. Table Operations

The following operations can be performed in the line above the table on the right:

- Filter / Sort

Show all / Only active (only Administrator)

- Search
- Export
- Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

9.6.5. Details Page

Each issue has its own detail page with several additional tabs. Clicking on the ID of the respective requirement or on the icon on the right in the action column takes you to the tab that was selected last. When called for the first time, this is the *Properties* tab.

The following tabs are available: *Properties*, *Test Cases*, *Systems under Test* and *Changes*.

9.6.5.1. Actions

On the detail pages there are more icons in the upper right corner. The following actions can be performed here:

Open external link	Shows the issue in the external issue management system.
Synchronize	Synchronizes the local issue with the external source.
Open print view	A print-ready view of the issue can be created here. With a click on the icon this opens in a new browser tab. Print views are described in detail in Section 5.2.3.2.1, “Print Pages”.
Create Bookmarks	Issues may be linked to from outside of Klaros-Test-management using the link on the icon. By clicking on the icon this link will be copied to the clipboard. Bookmarks are described in detail in Section 5.2.3.2.2, “Bookmarks”.
Browse	Use the green arrows at the very top right to switch between the issues present on the previous page.

9.6.5.2. Properties

In this tab (Figure 9.32) several attributes of the issues are displayed and can be edited if the user has the appropriate permission.

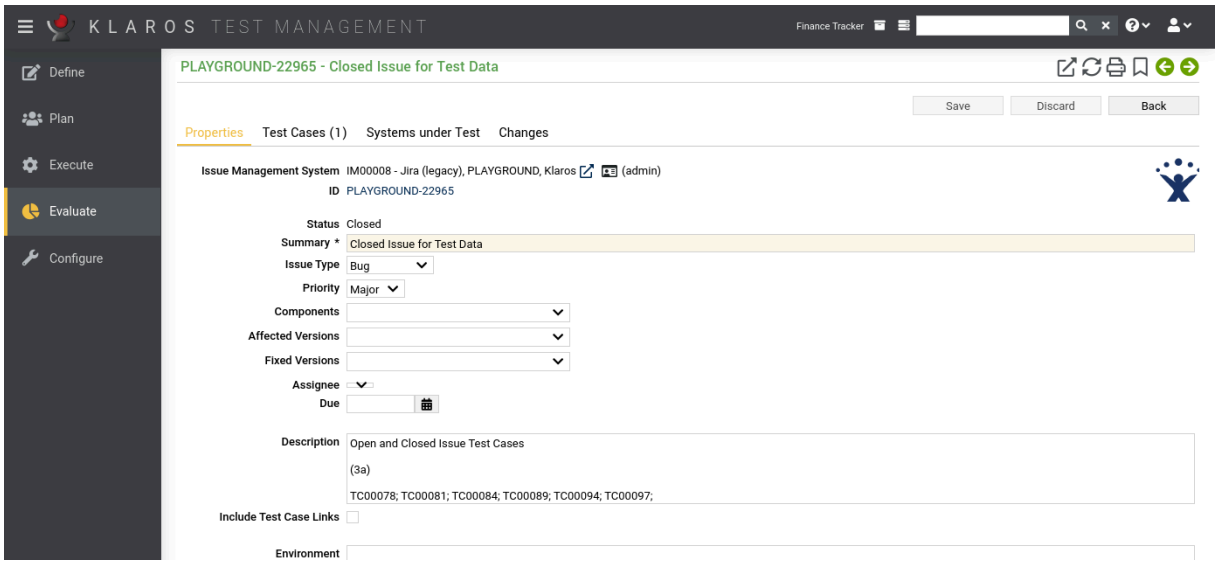


Figure 9.32. The “Properties” Tab

The issue *ID* is a hyperlink to the corresponding issue in the issue management system.

9.6.5.3. Test Cases

This view lists all test cases that are associated with this issue.

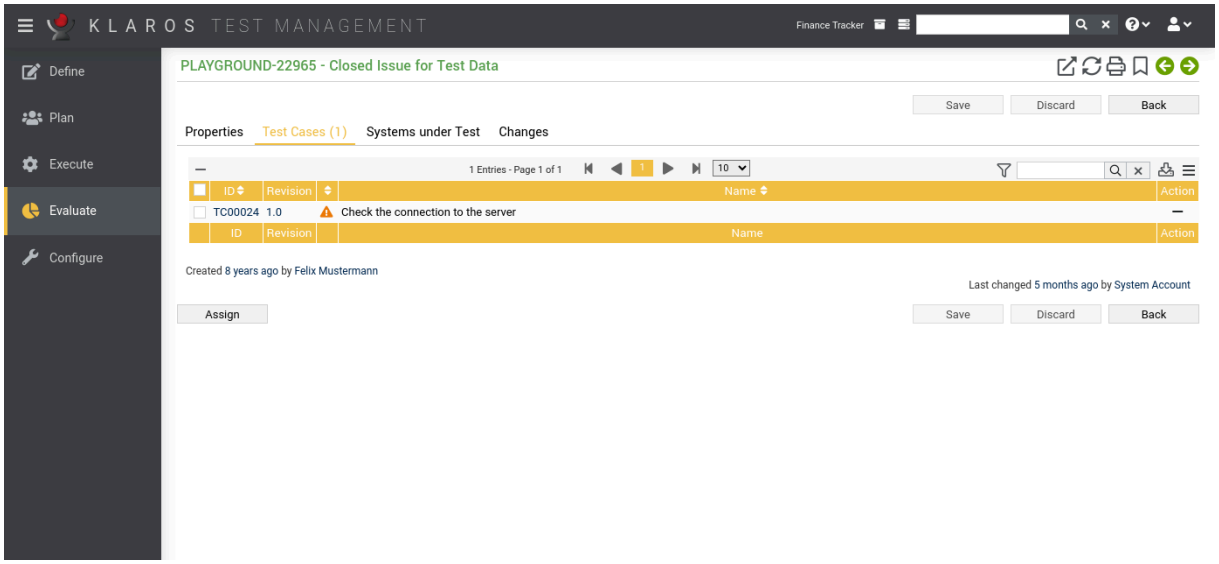


Figure 9.33. The “Test Cases” Tab

Test cases can be assigned to the issues by clicking the Assign button. This opens up a dialog, where multiple test cases can be selected at once.

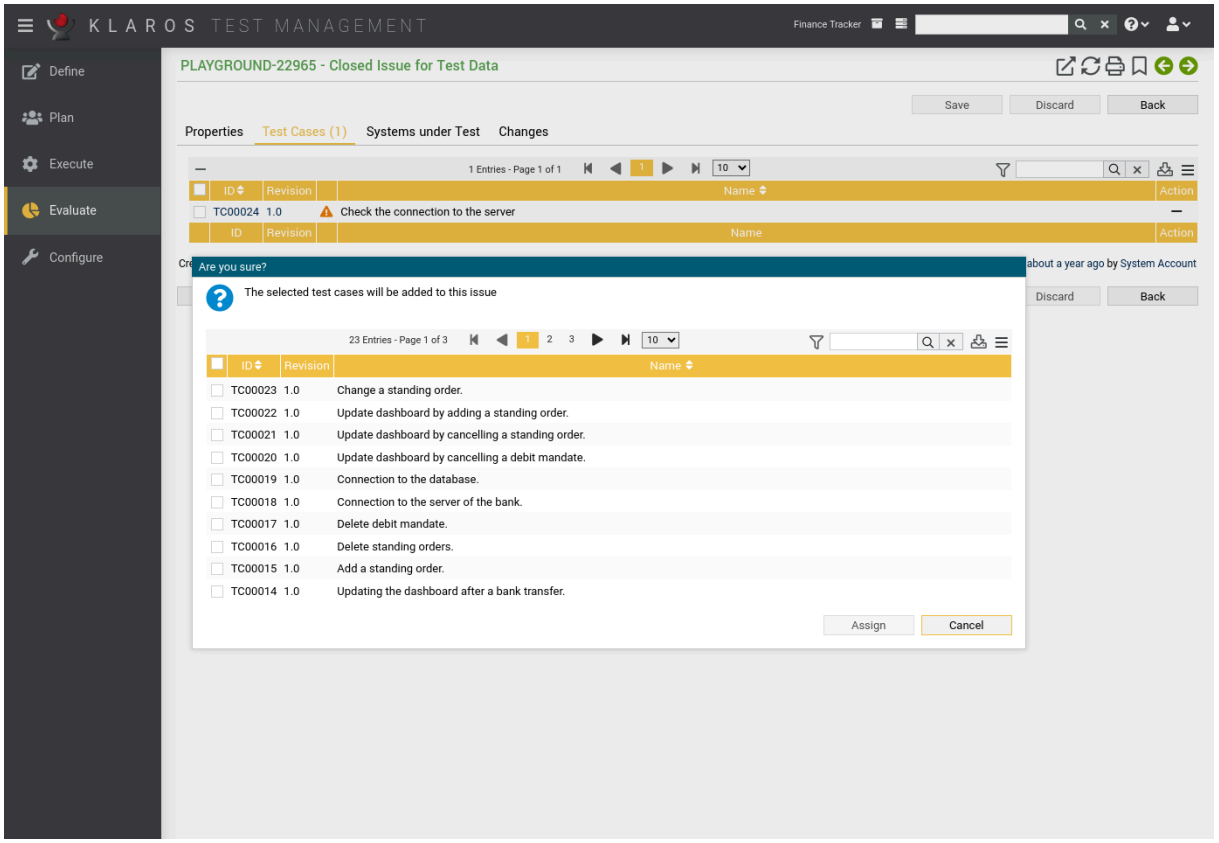


Figure 9.34. The “Assign Test Cases” Dialog

The following bulk actions are supported for test cases:

- Remove

The detail page of a test case shows a list of all issues that are associated with it, as shown in section Section 6.7.2.8, “Issues”.

9.6.5.4. Systems under Test

The page lists all *systems under test* assigned to the issue.

If you want to associate systems under test with the issue, click the Assign button and select one or more systems under test.

The following bulk actions are supported for systems under test:

- Remove

The detail page of a system under test shows a list of all issues that are associated with it, as shown in section Section 6.5.2.6, “Issues”.

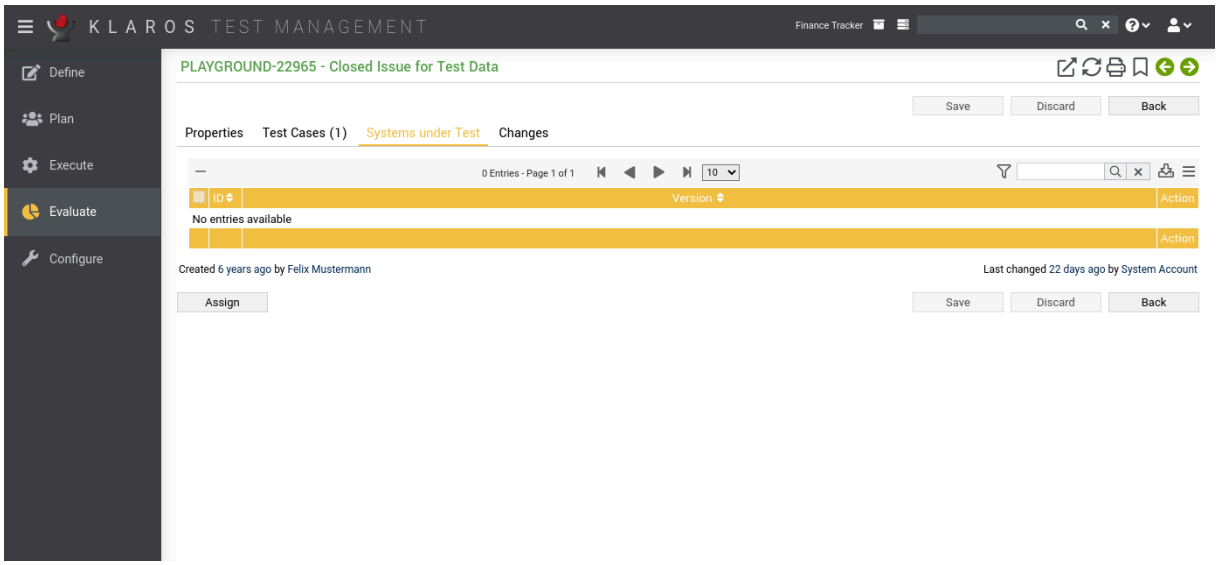


Figure 9.35. The “Systems under test” Tab

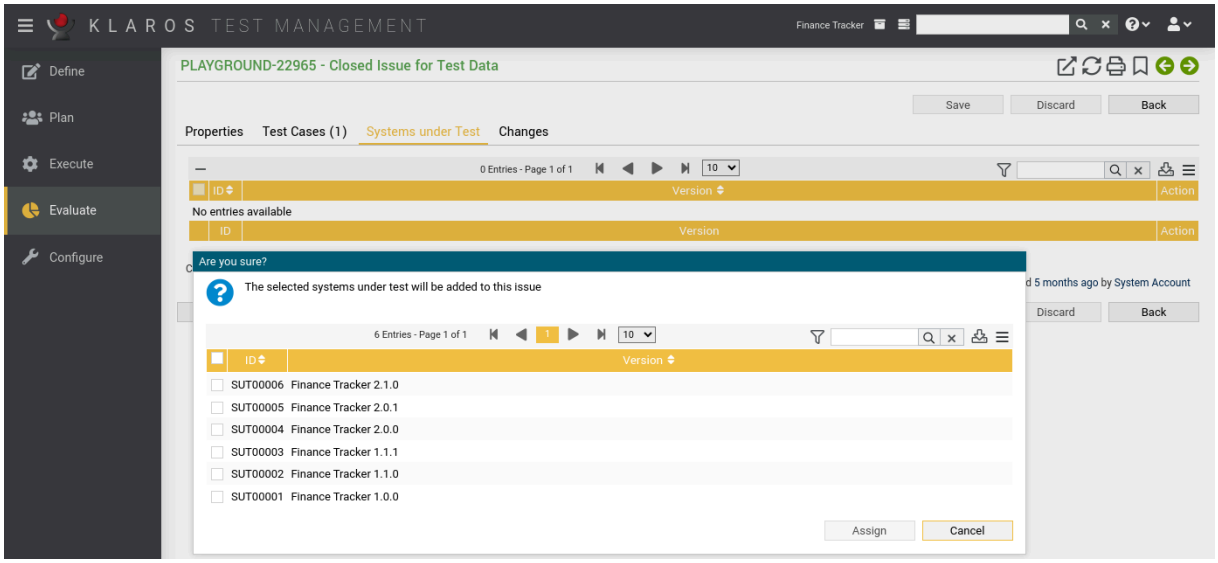


Figure 9.36. The “Assign Systems under test” Dialog

9.6.5.5. Changes

The tab *Changes* shows the change history of this issue.

For a detailed description of the *Changes* view, see Section 5.2.3.2.8, “Change History”.

9.6.6. Issue Details (Creating a new Issue)

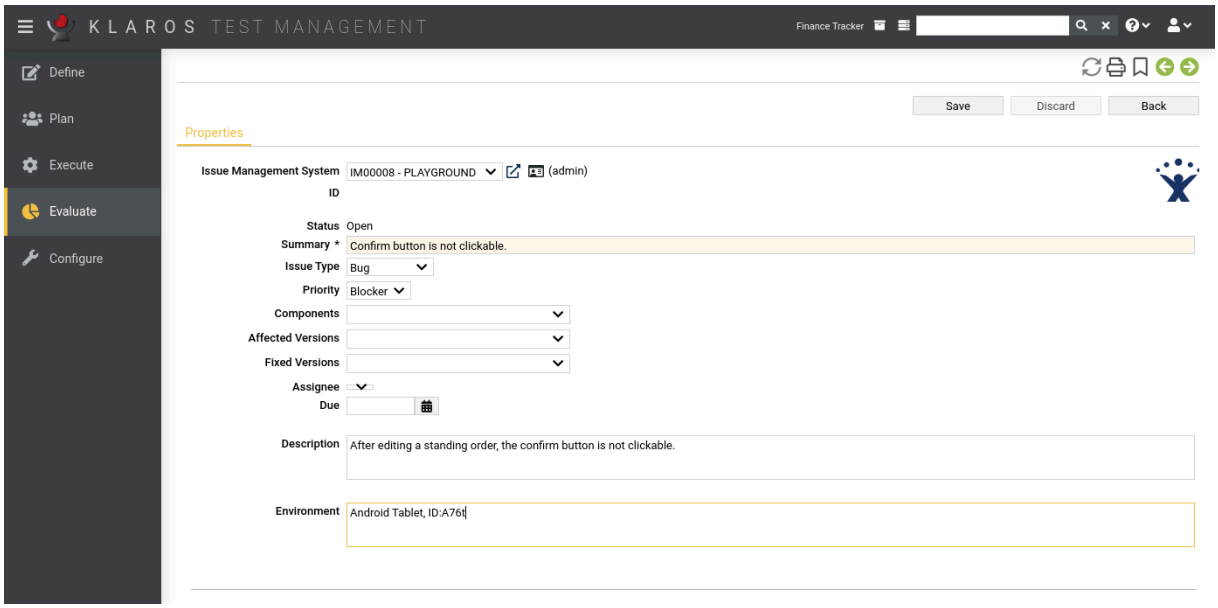


Figure 9.37. The “Create Issue” Page

The first defined issue management system of the active project is automatically selected from the *Issue-Management-System* dropdown list.

Note

If the list of issue management systems is empty, then at least one issue management system has to be configured and linked to the active project. The configuration of issue management systems is described in Section 10.5.1, “Issue Management”. Section 6.1.2.6.1, “Issue Management” contains instructions on how to add an issue management system to a project.

By default Klaros-Testmanagement uses the credentials entered at the login screen to authenticate the user to the issue management system. When using Bugzilla, the email/password combination of the user is tried as well, as this is the common case for a login name there. If the credentials entered at login do not work and the issue management system has not yet been used during the session, Klaros-Testmanagement will request new authentication information in a dialog window.

Token based authentication (Jira Cloud instances, GitLab)

Klaros-Testmanagement is supporting token based authentication scheme as well. Just use an empty username and the token as password.

Some Vendors like Jira and GitLab limit access to token based authentication Authenticating via username and password is no longer supported for these systems.

Clicking the icon opens up the external issue management system in a new tab.

Clicking the icon opens up a dialog to change the authentication information.

If an issue management system has been selected, a list of issue fields is shown. The exact number and names of those fields vary, depending on the type and configuration of the issue management system. Mandatory fields are shaded in a different color.

Note

All mandatory fields need to be filled, otherwise the issue cannot be saved.

Issue Fields for every Issue-Management-System

Test Cases	The test cases this issue is linked to.
Systems under Test	The systems under test this issue is linked to.
Update Test Case State	Selecting this checkbox opens up a new dropdown list in which the new state of the linked test cases can be selected.

9.6.6.1. Creating a Bugzilla Issue

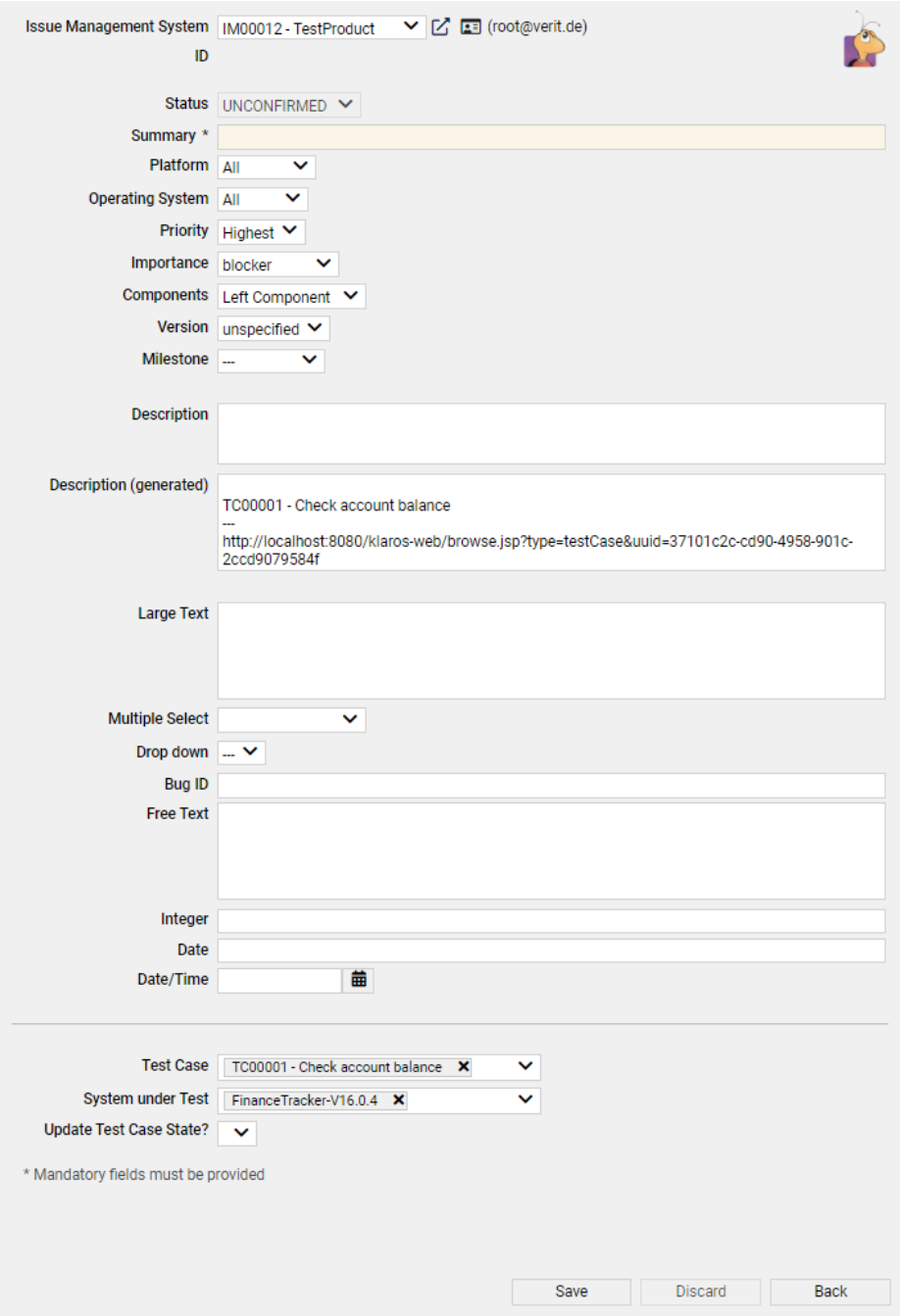


Figure 9.38. The “Bugzilla Issue ” Page

9.6.6.2. Creating a GitHub Issue

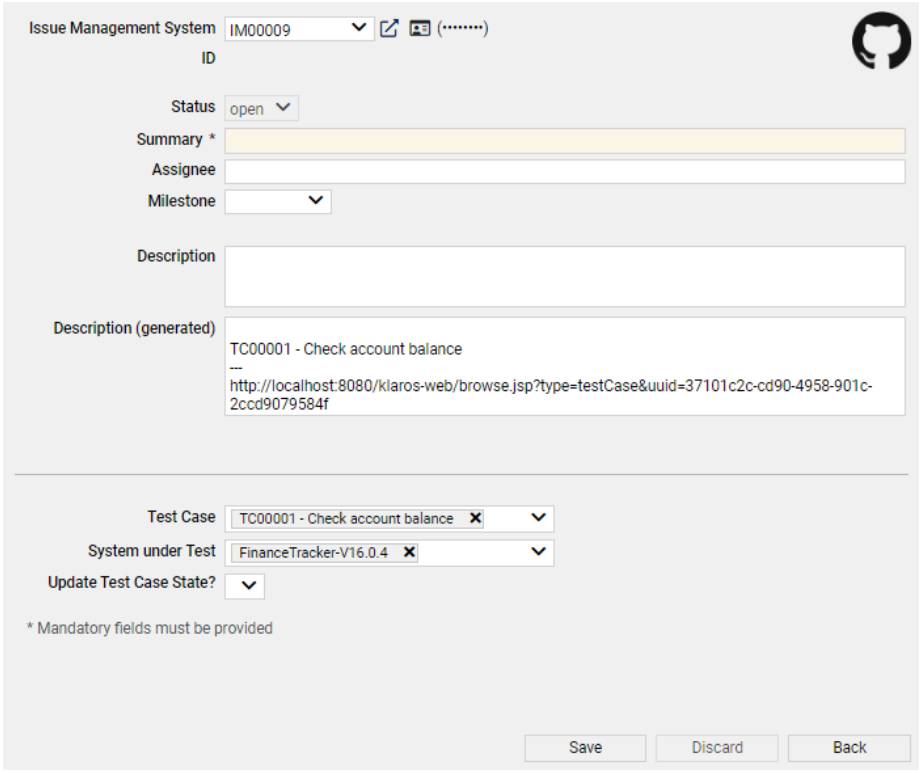


Figure 9.39. The “GiHub Issue” Page

9.6.6.3. Creating a GitLab Issue

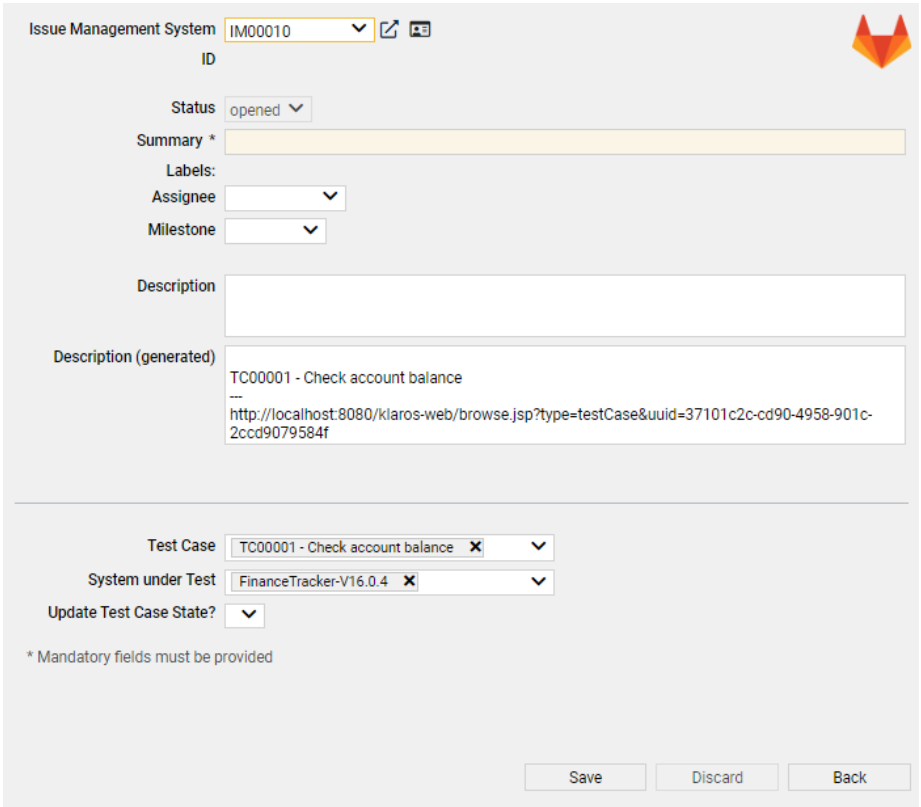


Figure 9.40. The “GitLab Issue” Page

9.6.6.4. Creating a Jira Issue

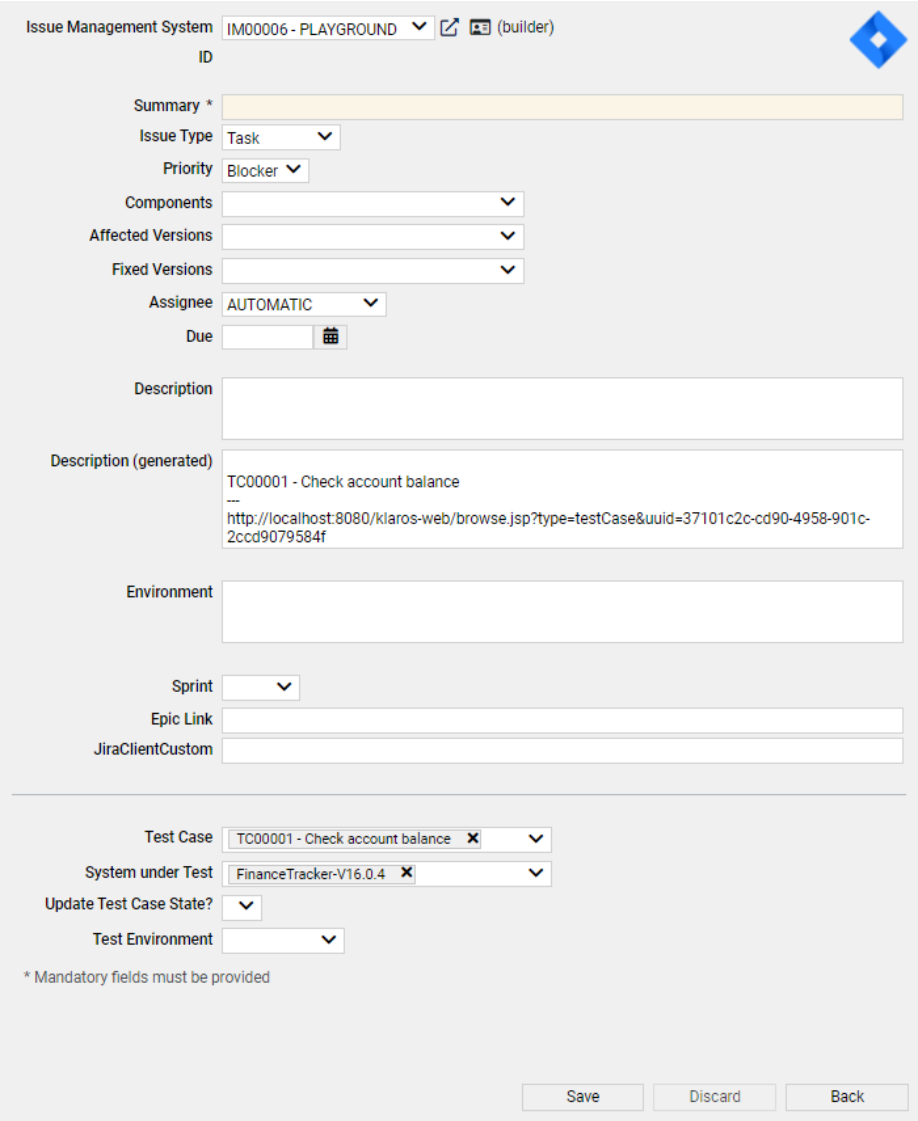


Figure 9.41. The “Jira Issue” Page

9.6.6.5. Creating a Mantis Issue

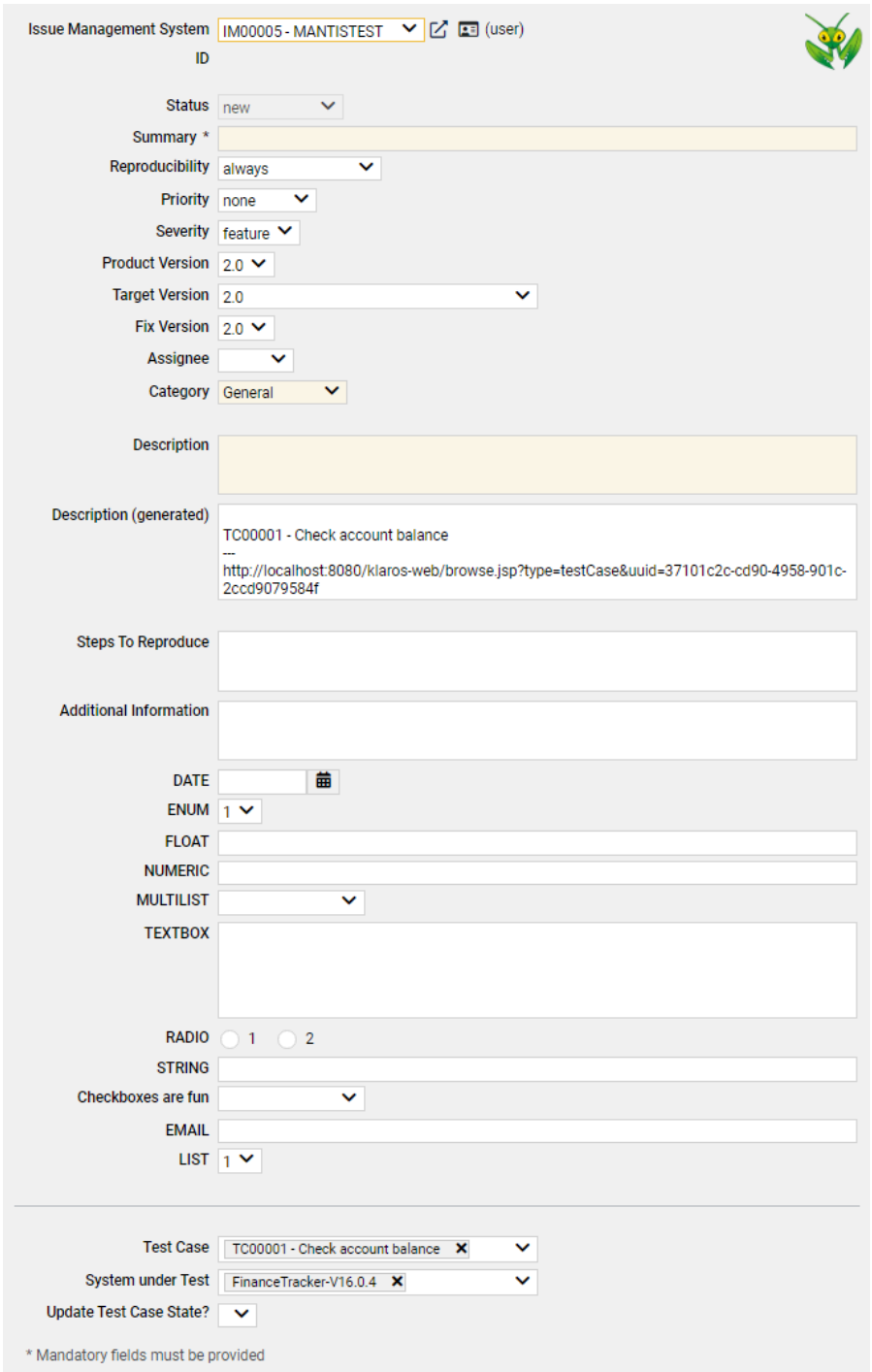


Figure 9.42. The “Mantis Issue” Page

9.6.6.6. Creating a Redmine Issue

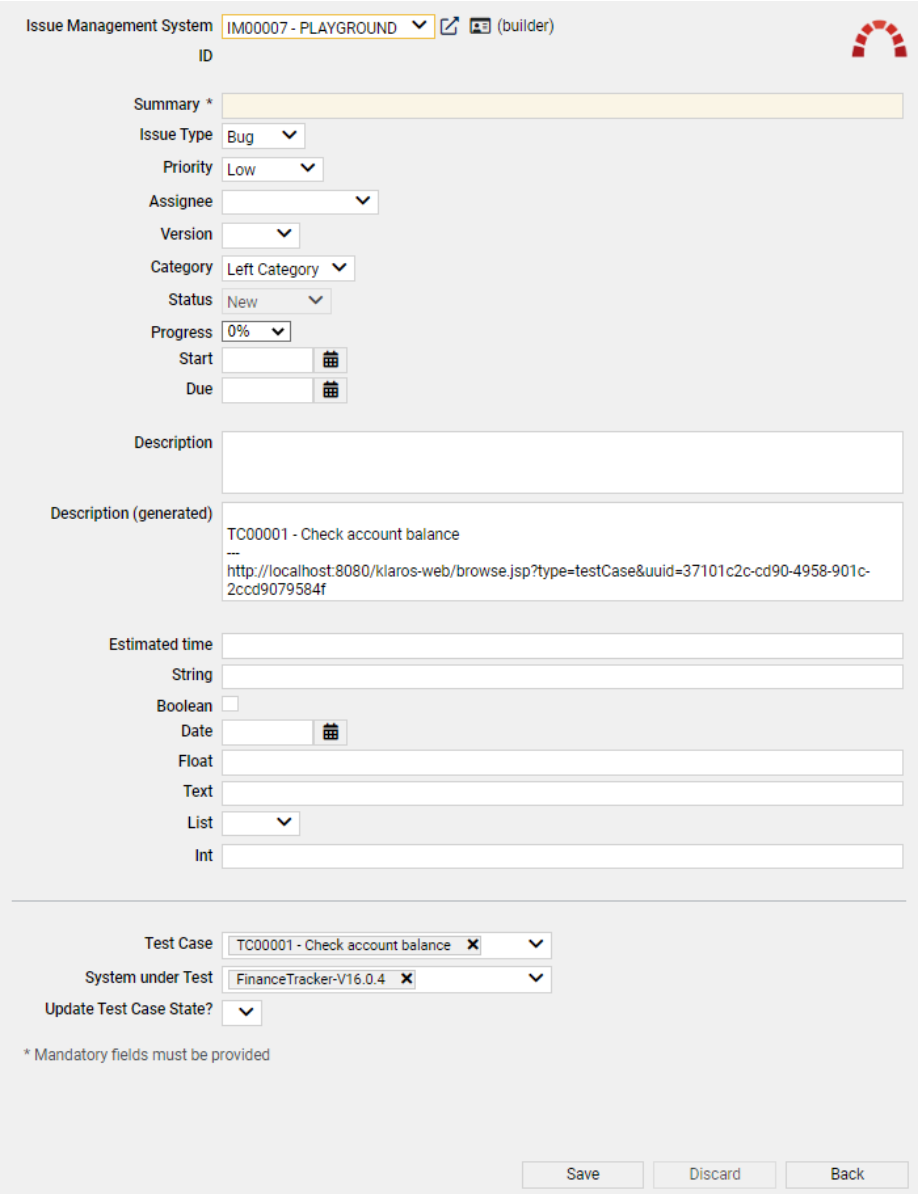


Figure 9.43. The “Redmine Issue” Page

9.6.6.7. Creating a Trac Issue

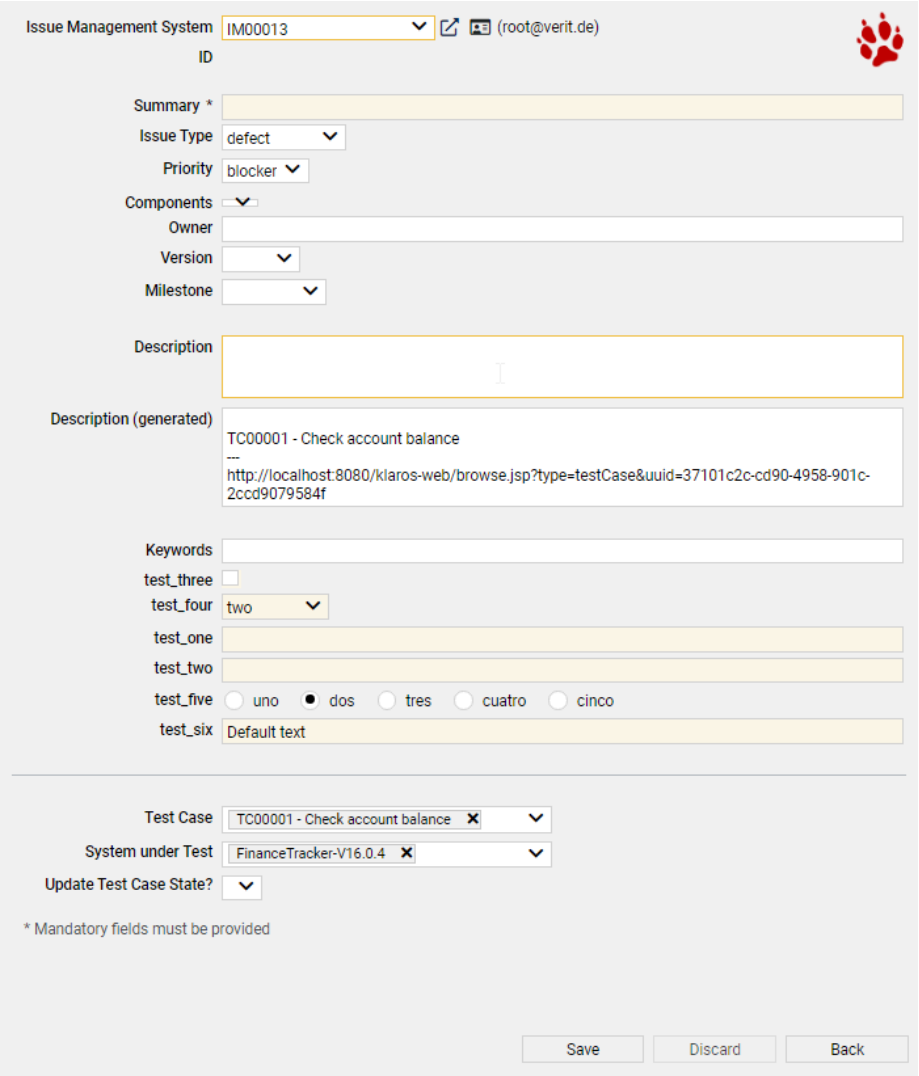


Figure 9.44. The “Trac Issue”Page

9.6.6.8. Creating a YouTrack Issue

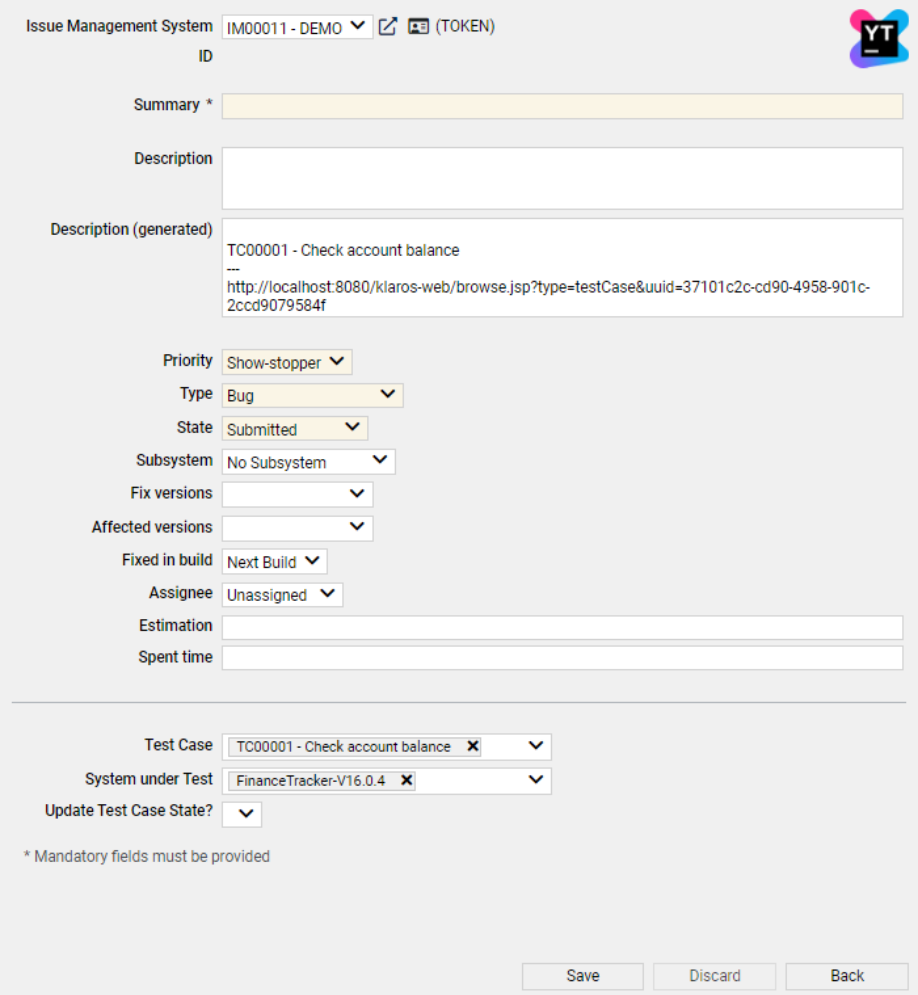


Figure 9.45. The "YouTrack Issue"Page

9.6.7. Link Issue

In addition to creating issues from within Klaros-Testmanagement, it is possible to link issues from an issue management system to test cases and systems under test in Klaros-Testmanagement.

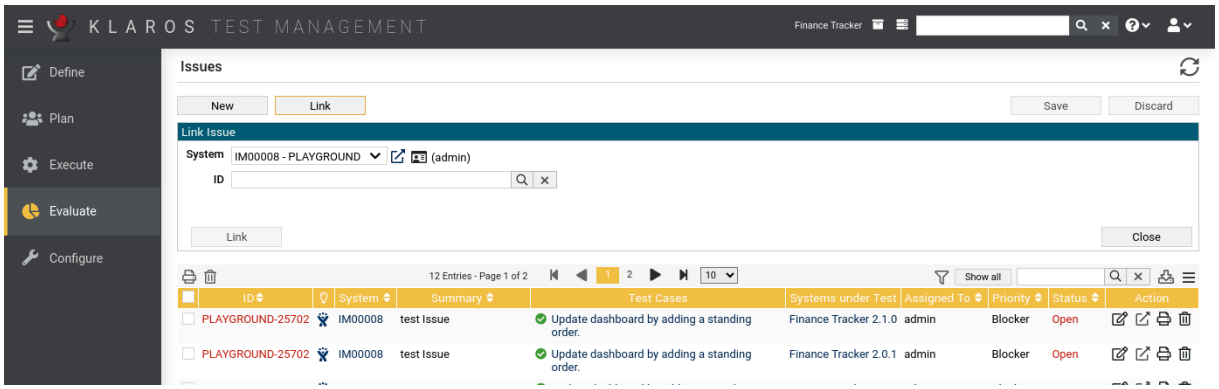


Figure 9.46. The "Link Issue" Panel

To link an issue to a test case or sytem under test, an issue management system must be selected first. Pressing the Link button links the issue with the entered *ID* to the selected test cases and systems under test.

System The issue management system from which issues are to be linked.

ID The id of the issue to be linked.

Pressing the icon searches the issue management system for an issue that matches the entered id. Upon a successful search, pressing the Link button links the issue to the selected test cases and systems under test.

Chapter 10. Configure

In the *Configure* section, all administrative tasks are performed. These include the **system configuration**, managing **custom reports**, creating and managing **user accounts**, changing **application settings**, configuring **remote system** parameters and **backing up and restoring** projects.

To access the majority of the functions described here, the global *Administrator* role is required.

10.1. Overview

On the *Overview* page you will find the following information:

- Server
- Database
- Operating System
- Locale
- Java
- Memory
- License
- Version

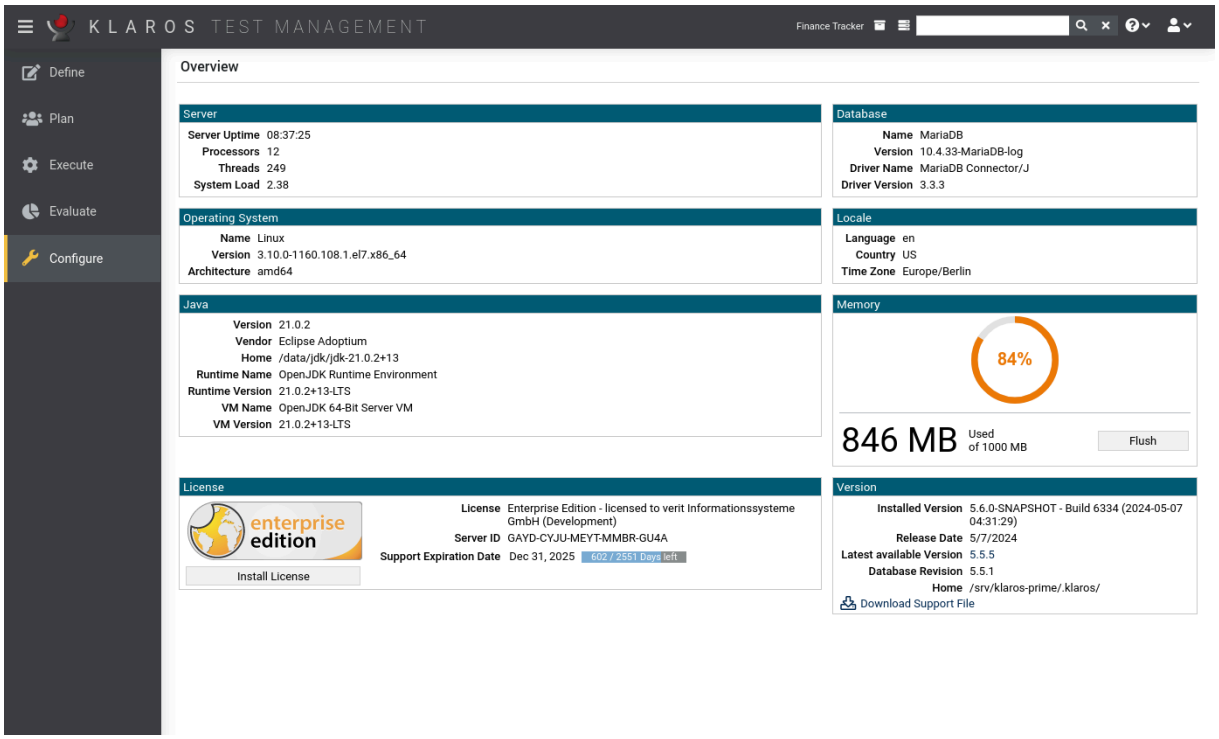


Figure 10.1. The “Overview” Page

10.2. Report Templates

Only available in Klaros-Testmanagement Enterprise Edition

On the *Report Templates* page, user-defined report templates can be created, edited, cloned and reports can be generated in PDF or Excel format.

Note

The predefined report templates are not editable and should be cloned for editing.

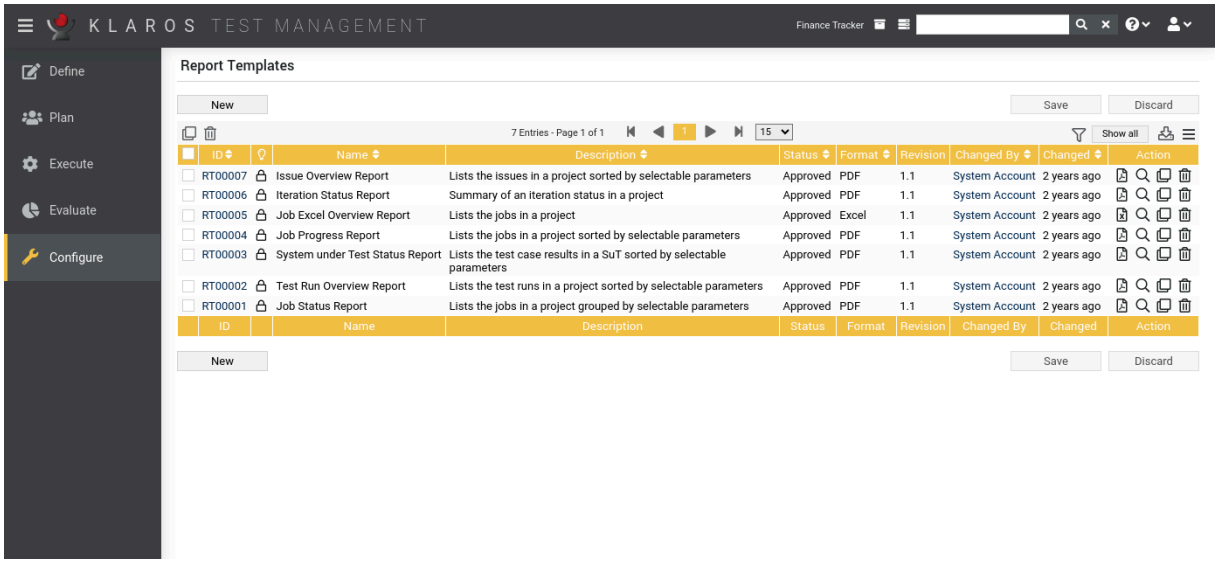


Figure 10.2. The “Report Templates” Page

The table shows the following values:

Additional Information — A tooltip appears when the cursor is placed over the icon shown here.

Name — The name of the report template.

Description — The description of the report template.

Revision — The revision of the report template.

Status — The status of the report template (draft, approved).

Format — The output format of the report (PDF or Excel).

Action — The executable actions.

Note

The *Additional Information* column indicates whether a report template is locked for editing.

10.2.1. Action

The following actions can be performed in the action column:

- Create the report as a PDF file.
- Create the report as an Excel file.
- Edit
- Duplicate
- Delete

If a report is to be generated that requires the input of parameters, a dialog opens. After confirmation with `OK` the report is generated. The button remains locked if mandatory parameters are missing user input. The `Cancel` button cancels the action.

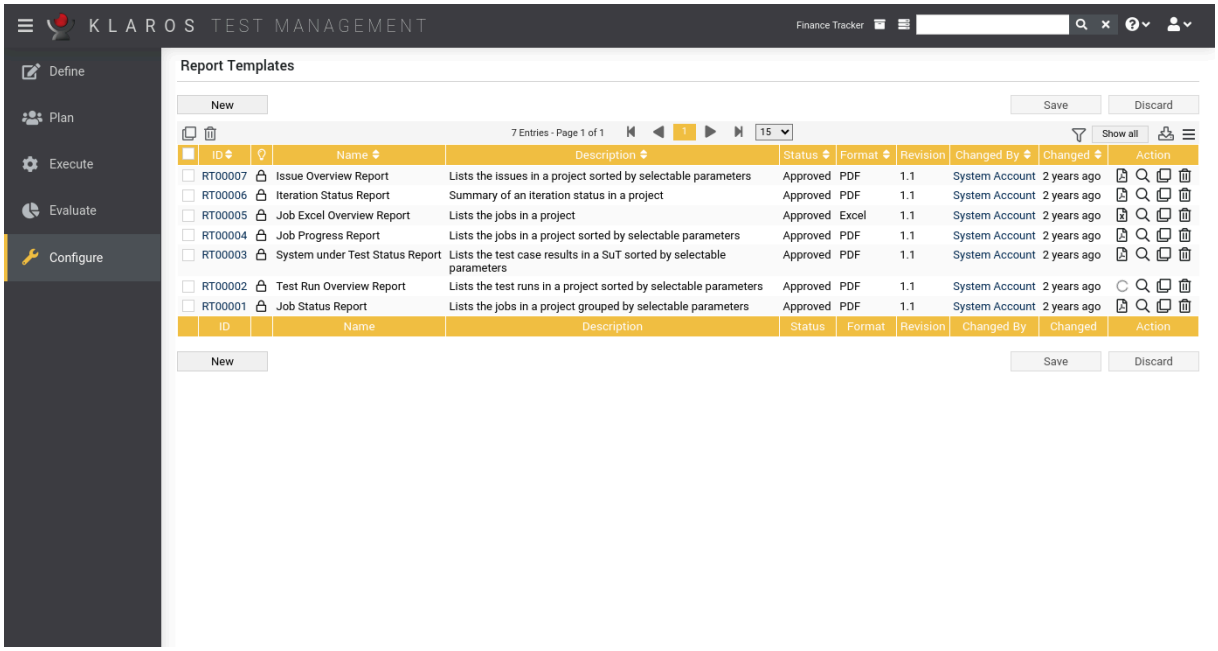


Figure 10.3. Generating a Parameterized Report

Note

Clicking the `New` button opens the page *Report Templates Details*. (Figure 10.4).

10.2.2. Report Templates Details

On the *Report Template Details* page, the name and description of the report template can be specified. A valid script and layout template must also be specified. The script and layout template can be edited directly in the text area or uploaded as an external file. In the Revisions tab, older versions of the report can be restored and differences to a previous version can be listed.

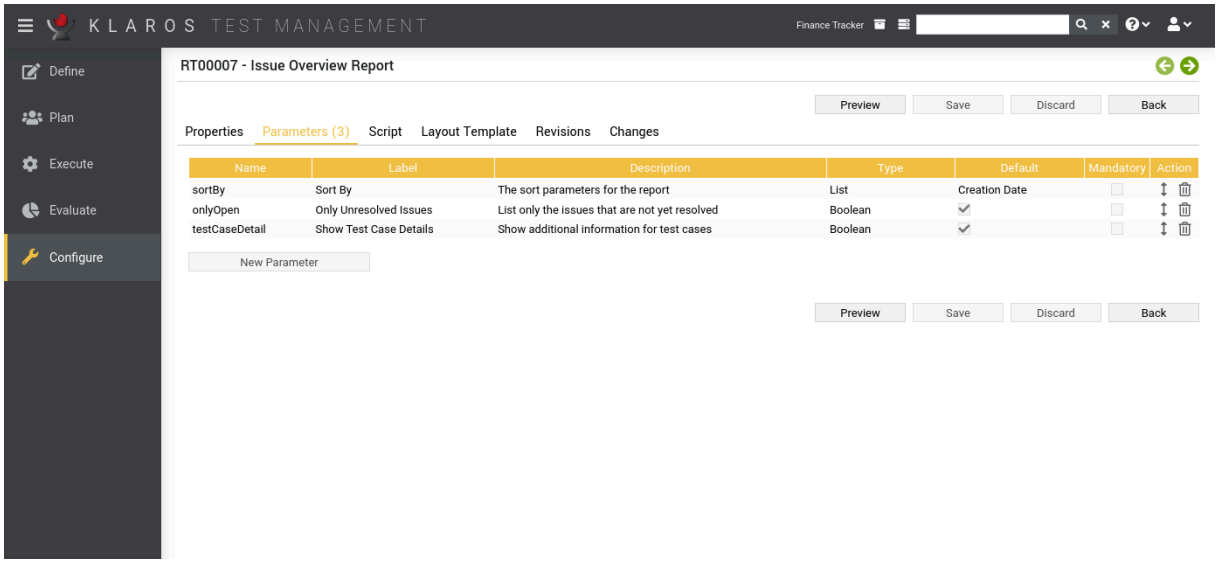


Figure 10.4. The “Report Details” Page

Integrated Syntax-Check

To be able to save the report template, the script and the layout template must not contain any syntactical errors. This is checked before saving.

Detailed information about creating user-defined reports is available in Section 11.2, “Creating a Report Template”.

10.3. Users

The user accounts are created and edited in the *Users* section.

10.3.1. Overview Page

The overview page gives an overview of the user accounts in Klaros-Testmanagement. The user accounts are created, edited, merged, activated and deactivated here.

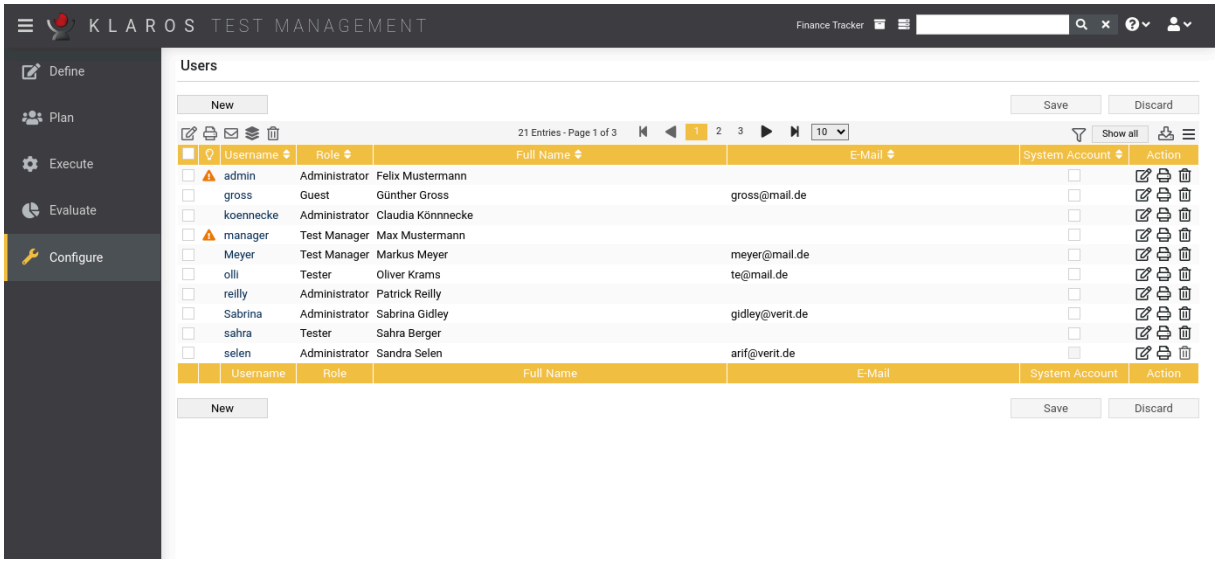


Figure 10.5. The “Users” Page

The table shows the following values:

Additional Information — A tooltip appears when the cursor is placed over the icon shown here.

Username — The unique username.

Role — The user roles (Administrator, Test Manager, Tester, Guest).

Full Name — The full name of the user.

E-Mail — The email address of the user.

Actions — The executable actions.

10.3.1.1. Actions

The action column is located on the far right of the table. The following actions can be performed here:

- Edit
- Open print view
- Delete

If a user has been deleted, it is initially marked with a deletion marker and is only visible to administrators. For deleted users, the following action is available instead of *Delete*:

- Restore (only Administrator)

10.3.1.2. Bulk Actions

Some actions can be applied to multiple user accounts at the same time. To do this, select the user accounts to which the action is to be applied in the leftmost column.

The following bulk actions are supported for user accounts:

- Edit

Open print view

Send Email

Merge User Accounts

It is not possible to permanently delete users directly, but users can be merged. The user to be kept will be selected in the dialog (Figure 10.6). The selected user takes over the job assignments, test results and test runs of the other users. The other users will then be permanently deleted and are therefore no longer listed in the user list.

Restore (only Administrator)

Bulk actions are described in detail in Section 5.2.3.1.5, “Bulk Actions”

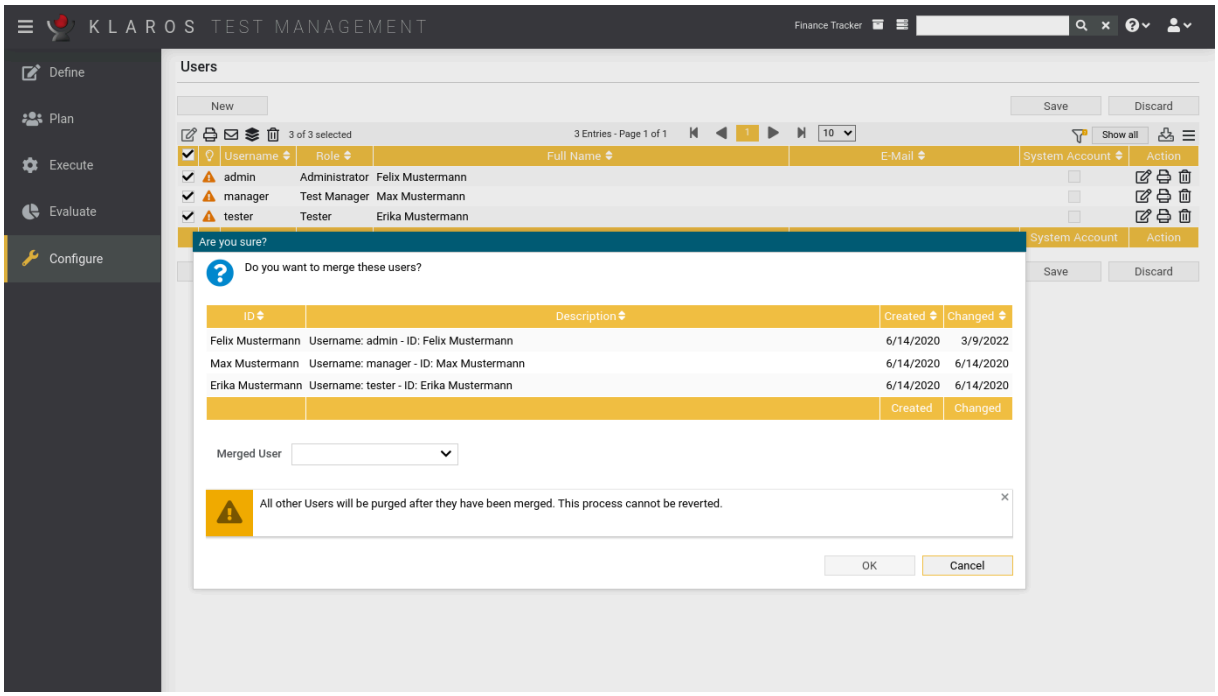


Figure 10.6. The “Merge User” Dialog

10.3.1.3. Table Operations

The following operations can be performed in the line above the table on the right:

Filter / Sort

Show all / Only active (only Administrator)

Export

Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

Changing User Roles

Users - including administrators - cannot change their own *System Account* flag or *role*. This can only be done from another user account that has the *Administrator* role.

10.3.1.4. Creating a new User

Clicking the New button creates a new user account (Figure 10.5). The *Username*, *Role*, *Full Name*, and *E-Mail* attributes can be set here.

When the Save button is clicked, a dialog will appear for each new user account to set the *Password* and *System Account* flag (Figure 10.7).

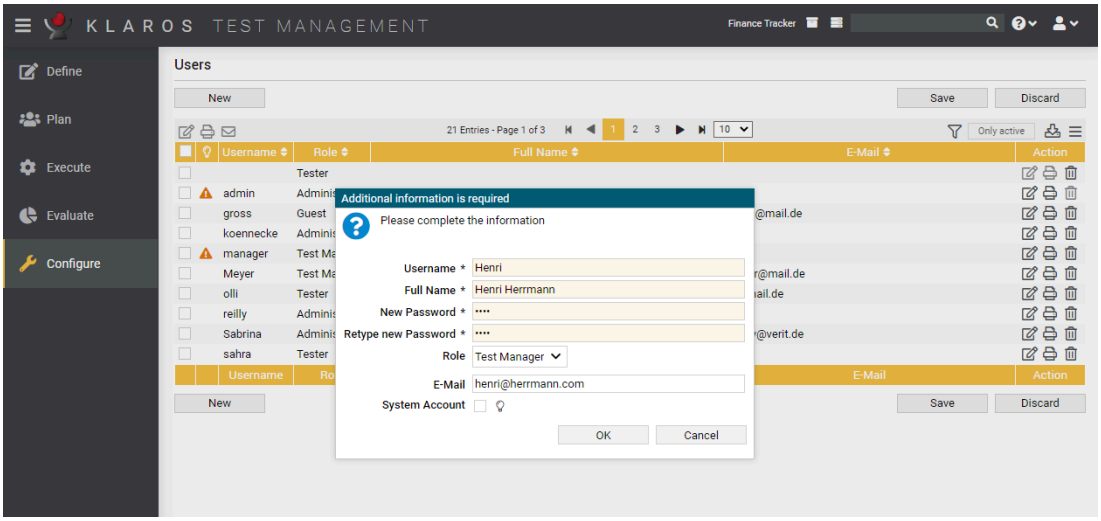


Figure 10.7. The “Save New User” Dialog

Per default users of the role *Administrator* or *Manager* are able to create new user accounts. Users of the role *Manager* can only create accounts of the role *Tester* or *Guest*.

Restricting Account Creation to Administrators

If a more restrictive policy is desired, the *Only Administrators can create Users* setting can be used. See Section 10.4.1, “Miscellaneous” for more information.

10.3.1.5. Deleting a User

Clicking the icon page deactivates the user account. Only users of the role *Administrator* can see and reactivate deactivated user accounts.

User accounts can only be deactivated, not deleted!

By toggling the Only Active and Show All button above the table, administrators can hide or show deactivated user accounts.

Instead of deleting user accounts, they can be merged with other existing user accounts. This operation will delete all user accounts to be merged except the remaining account.

10.3.2. Details Page

Each user account has its own detail page with several additional tabs. Clicking on the ID of the respective user account or on the icon on the right in the action column takes you to the tab that was selected last. When called for the first time, this is the *Overview* tab.

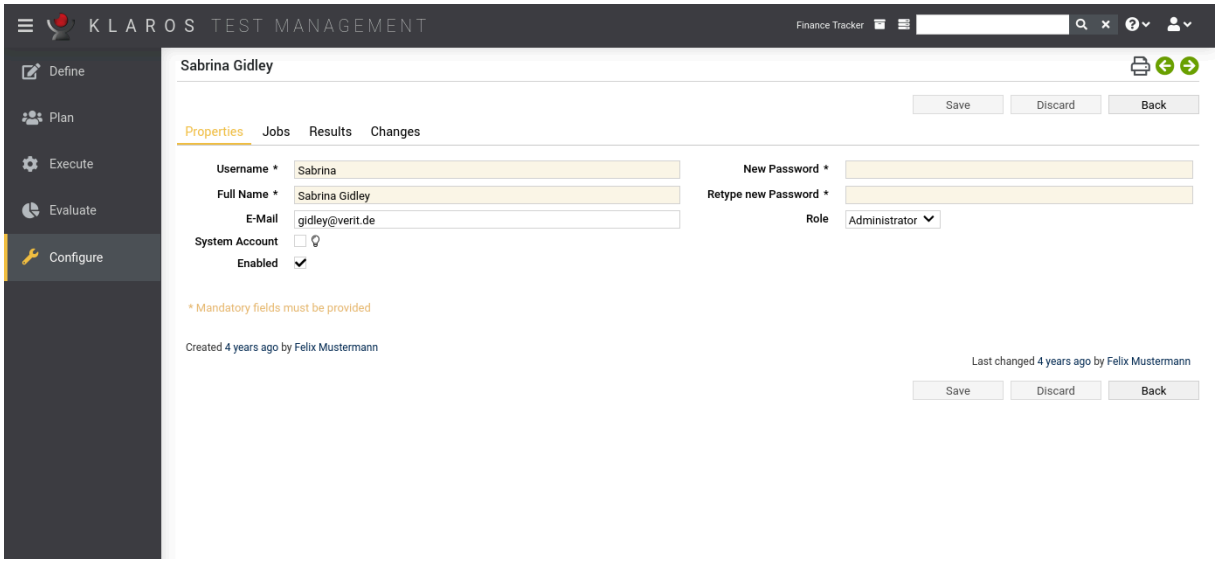


Figure 10.8. The "Properties" Tab

The following tabs are available: *Properties*, *Project Roles*, *Jobs*, *Results* and *Changes*.

10.3.2.1. Actions

On the detail pages, there are additional icons in the top right corner of the header. The following actions can be performed here:

Open print view — A print-ready view of the user account can be created here. With a click on the icon this opens in a new browser tab.

Print views are described in detail in Section 5.2.3.2.1, "Print Pages".

Browse — Use the green arrows at the very top right to switch between the user accounts present on the previous page.

10.3.2.2. Properties

This tab (Figure 10.8) allows the user to view or change the following attributes of the user account:

Username — The login name of the user

Full Name — The full name of the user

E-Mail — The user's email address which is used for email notifications. If a corresponding event has been configured (see Section 10.4.2, "Notifications") and an email address is specified

	when creating a new user (see Section 10.3.1.4, “Creating a new User”), this user will receive a registration email with username and password.
System Account	If this box is checked, this user will not be able to log in to the login page. System accounts are intended for automated tasks such as importing data.
Active	If this box is checked, this user will not be able to log in to the login page.
Password	The password to log in to Klaros-Testmanagement.
Retype Password	Confirmation of the password.
User Role	The user role. Available roles are *Administrator*, *Test Manager*, *Tester* or *Guest*.

Limitations when Changing User Roles

Users can not change their own *System Account* flag and *Role*. This must be done by another user with administrator role.

10.3.2.3. Project Roles

Only available in Klaros-Testmanagement Enterprise Edition

The *Project Roles* tab shows the project roles the user has in projects with project specific roles (see Section 6.1.2.5, “Access” for more information on how to assign project-specific roles to a project). Administrators can change the project roles for these projects here.

This tab is available once the first project has been assigned project-specific roles.

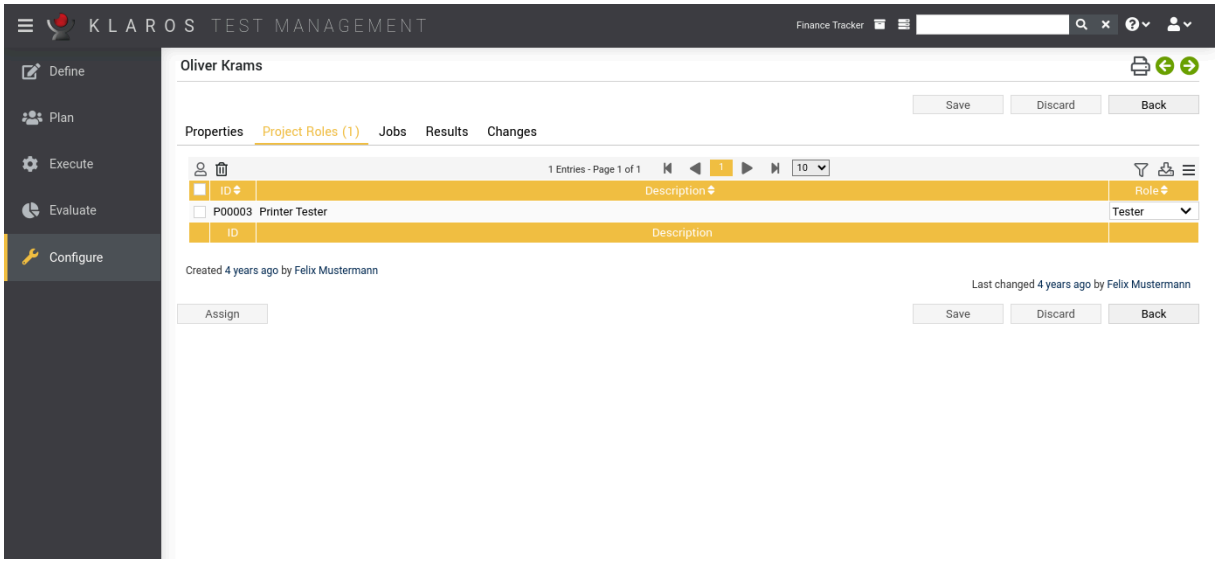


Figure 10.9. The “Project Roles” Tab

Pressing the Assign button opens up a dialog with a list of all projects with project specific roles that the user isn't assigned to yet. Pressing the Ok button assigns the user to the selected projects.

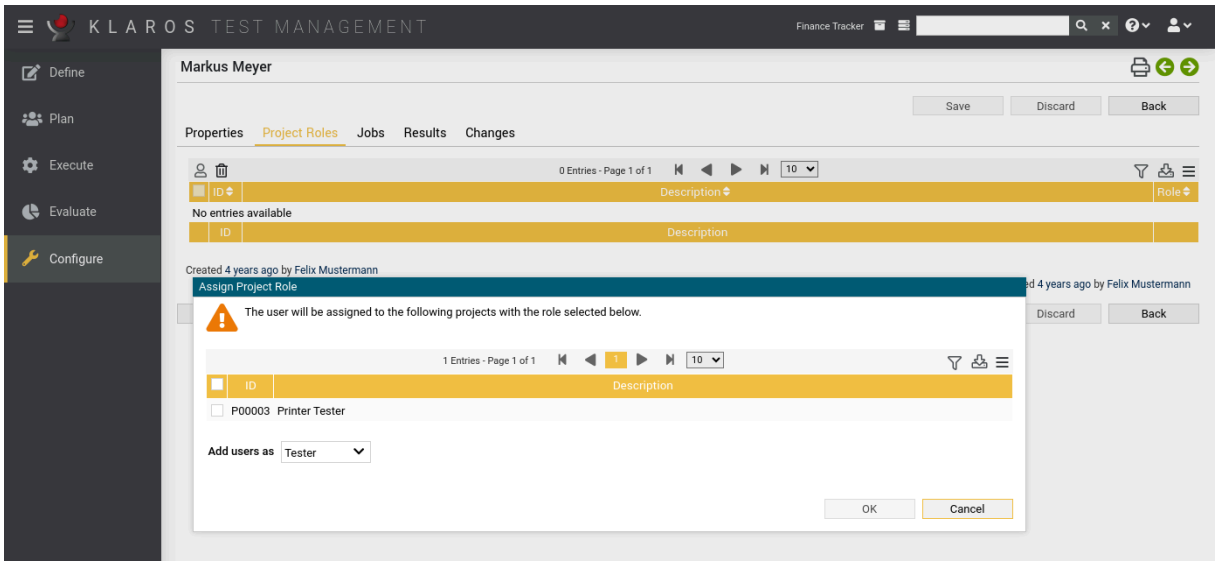


Figure 10.10. The “Assign Project Roles” Dialog

10.3.2.4. Jobs

The *Jobs* tab lists all the jobs assigned to the user. See Section 7.1, “Jobs” for more information on how to manage jobs.

10.3.2.5. Results

The *Results* tab shows all the test results of the tests that the user has executed, see Section 5.2.3.2.7, “Test Runs and Results”.

10.3.2.6. Changes

In the *Changes* tab the change history for the user can be found, see Section 5.2.3.2.8, “Change History”.

10.4. System

This page manages the settings for the behavior and appearance of the application. It contains five tabs, *Miscellaneous*, *Notifications*, *Test Execution*, *Interface* and *Languages*.

10.4.1. Miscellaneous

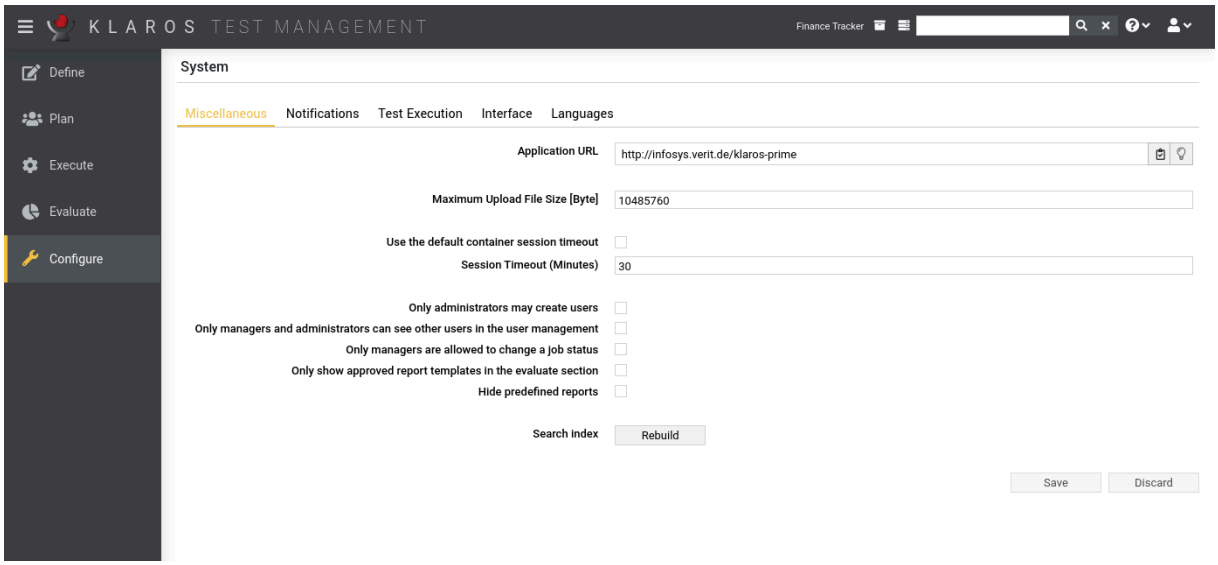


Figure 10.11. The “Miscellaneous” Tab

The following settings can be made here to define the behavior of the application:

Session Timeout

A changed session timeout does not affect the current session.

Application URL

If the application should be accessed via a proxy server, the configured URL must be specified here.

For detailed information on configuring a proxy, see https://httpd.apache.org/docs/2.4/mod/mod_proxy.html (Apache), https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/ (NGinx) https://docs.microsoft.com/en-us/iis/extensions/configuring-application-request-routing-arr/creating-a-forward-proxy-using-application-request-routing (IIS).

Clicking the will verify the specified connection.
.

Maximum Upload File Size [Byte]

The maximum size for attachments to be uploaded.

Use the default container session timeout

If checked, the default tomcat session timeout will be used (30 minutes).

Session Timeout (Minutes)

If the checkbox above is unchecked, this value is used as the session timeout.

Only Administrators can create Users

If checked, only administrators can create users. If unchecked, managers can also create users.

Only Managers and Administrators can see other users in the user management	If enabled, users in the *Tester* and *Guest* roles will only see their own user entry. Entries from other users will not be displayed.
Only managers are allowed to change a job status:	If checked, only users in the role of *Manager* or higher may change the status of jobs.
Only show approved report templates in the evaluate section	If enabled, only reports with the status *Approved* will be displayed in the *Configurable Reports* table on the *Evaluate / Reports* page.
Hide predefined reports	If enabled, the display of the *Reports* table is suppressed on the *Evaluate / Reports* page.
Search index	If problems should occur with the general search, a re-indexing of the database can be started here using the Rebuild button.

10.4.2. Notifications

Only available in Klaros-Testmanagement Enterprise Edition

For various events, such as the creation of a user account or the assignment of a task, notifications are sent by the system via e-mail. In the *Notifications* tab, the corresponding conditions are set.

Multiple notification settings can be defined in a notification scheme and reused for multiple projects. When a notification scheme is declared as the default notification scheme, it is automatically assigned to each existing project unless another scheme has already been defined for that project.

Email-Configuration Required!

For notifications to work properly, you must configure email server settings as described in Section 10.5.3, “E-Mail” and ensure that all user accounts have a valid email address.

10.4.2.1. Overview Page

The overview page displays all existing notification schemes in a table. New notification schemes are created here.

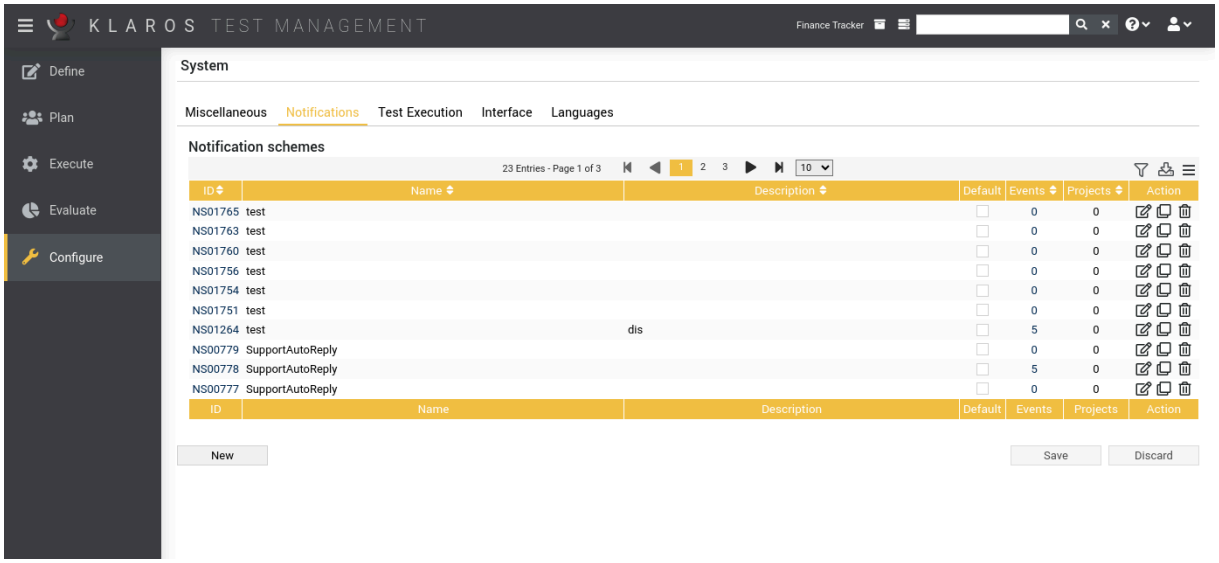


Figure 10.12. The “Notification Schemes” Page

The table shows the following values:

ID — Assigned automatically.

Name — The name of the notification scheme.

Description — The description of the notification scheme.

Default — When enabled, the notification scheme is the default scheme. If a notification scheme is declared as the default scheme, it is automatically assigned to any existing project, unless another scheme has already been defined for that project.

Events — The number of notification events.

Projects — The number of projects to which this notification scheme is assigned.

Actions — The executable actions.

10.4.2.1.1. Actions

The action column is located on the far right of the table. The following actions can be performed here:

- Edit
- Duplicate
- Delete

10.4.2.1.2. Table Operations

The following operations can be performed in the line above the table on the right:

- Filter / Sort

Export

Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

10.4.2.1.3. Details Page

Figure 10.13 shows the edit options for a notification scheme.

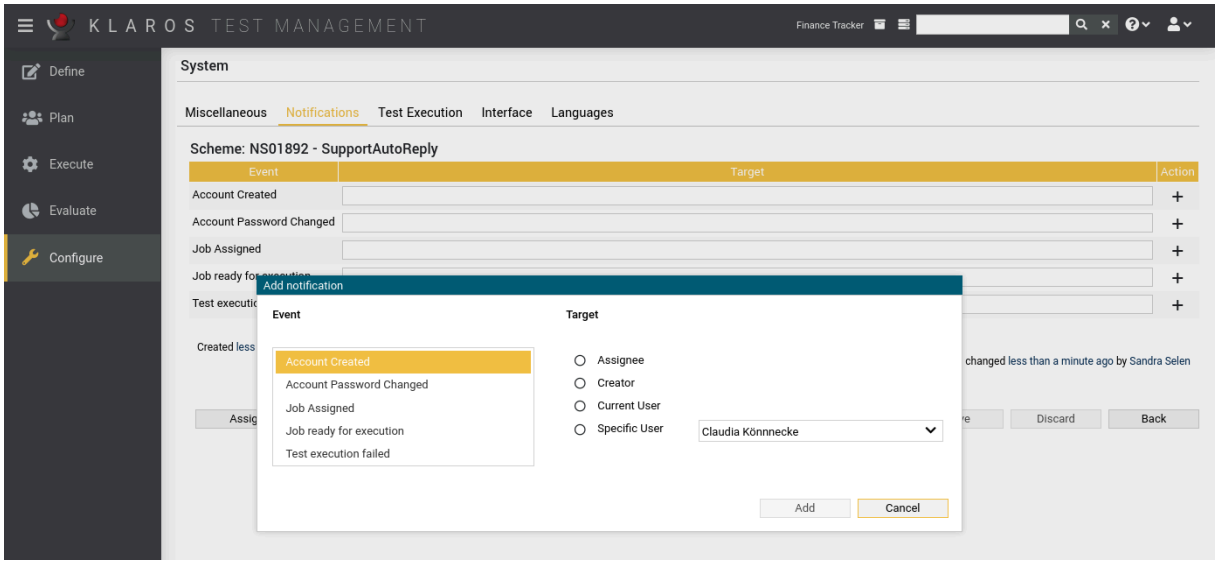


Figure 10.13. The “Add notification” Dialog

This page lists all notifications for the notification scheme. To create a notification, events from a predefined set of available notification events can be individually assigned to one or more recipients (user accounts).

The available notification events are:

Job Assigned — This event is triggered once a Job is saved with a new assignee.

Job ready for execution — This event is triggered whenever a non-executable job becomes executable, or all dependencies of the job are fulfilled (See Section 7.1.2.5, “Dependencies”).

Account Created — This event is triggered once a user account has been created.

Account Password Changed — This event is triggered once an account password has been changed in the local user database.

Test execution failed — This event is triggered after completing a test case execution if a test case step was marked as failed or error during execution.

The available notification targets are:

Assignee — The assignee of the job/test case or the created/edited user account.

Creator — The user responsible for triggering the notification event, e.g. by assigning a job.

Current User — The user currently active.

Specific User — A specific user.

By clicking the Assign projects button, the notification scheme can be assigned to one or more projects (see Figure 10.14).

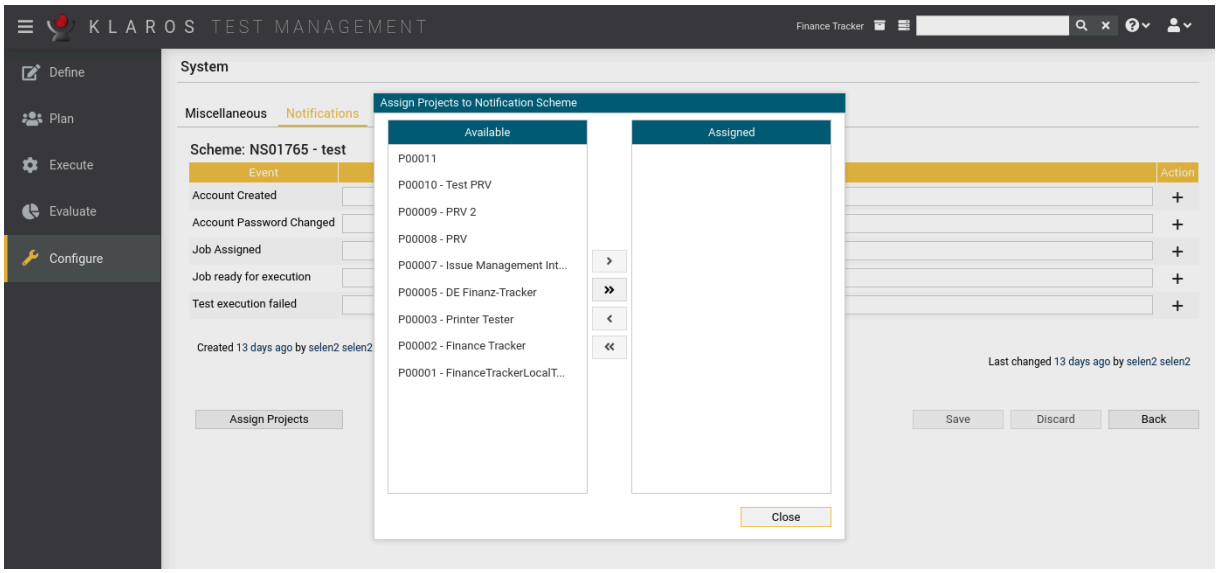


Figure 10.14. Assigning Projects to a Notification Scheme

10.4.3. Test Execution

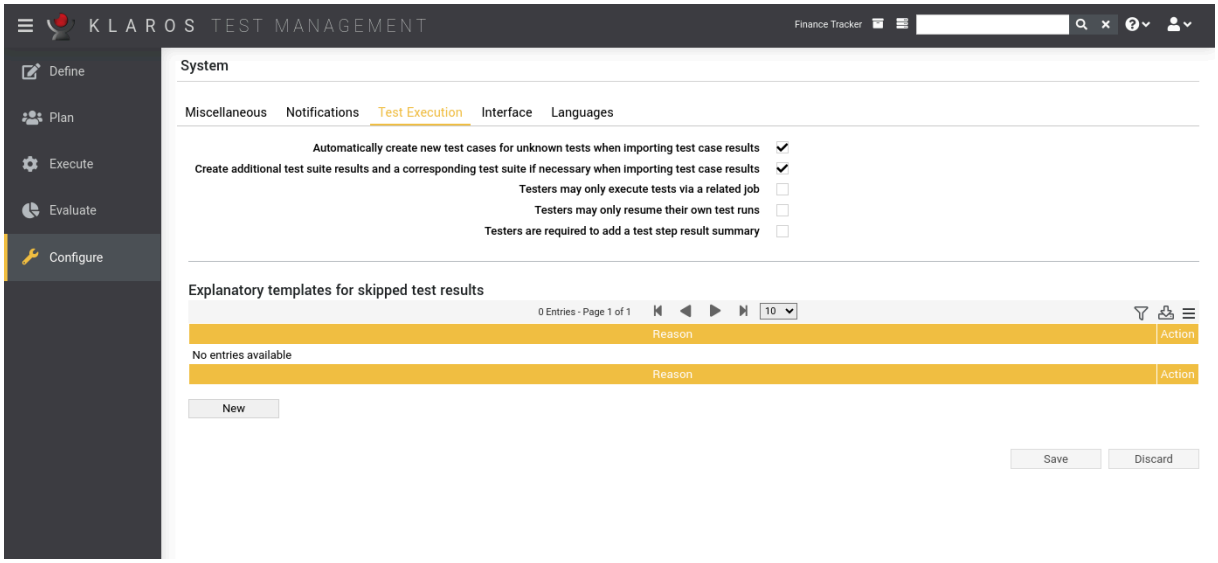


Figure 10.15. The “Test Execution” Tab

In this view, the following settings regarding test execution can be made:

Automatically create new test cases for unknown tests when importing test case results — If selected, a test case is automatically created for the respective test case information in the result file. This information may vary with the import format.

Create additional test suite results and a corresponding test suite if necessary when importing test case results — If selected, a test suite result is automatically created for the respective test suite information in the result file. This information may vary with the import format. In addition, a corresponding test-suite is created for the test-suite result, if it doesn't already exist.

Testers may only execute tests via related jobs — If selected, users in the role of *Tester* may only start tests by executing a job.

Testers may only resume their own test runs — If enabled, users in the *Tester* role can only resume test runs, that they have started themselves. If a test run was started via a job, they can only resume the test run only if the job is assigned to them.

10.4.3.1. Explanatory templates for skipped test results

If testers want to skip a test case during manual test execution, the system asks for a justification for this. This table contains the templates for these reasons (see Section 8.3.3.1.1, “Skip Test Cases Permanently”). These templates can be applied when skipping.

10.4.3.1.1. Actions

The following actions can be performed in the action column:

Edit

Delete

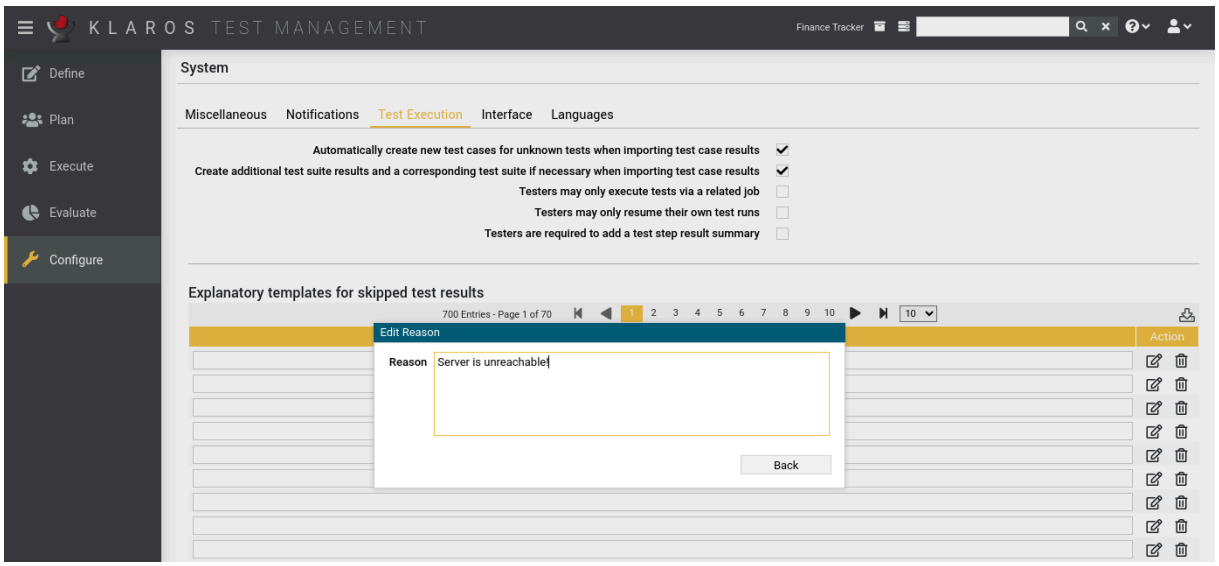


Figure 10.16. The “Edit Reason” Dialog

10.4.4. Interface

The *Interface* tab defines the behavior of the user interface.

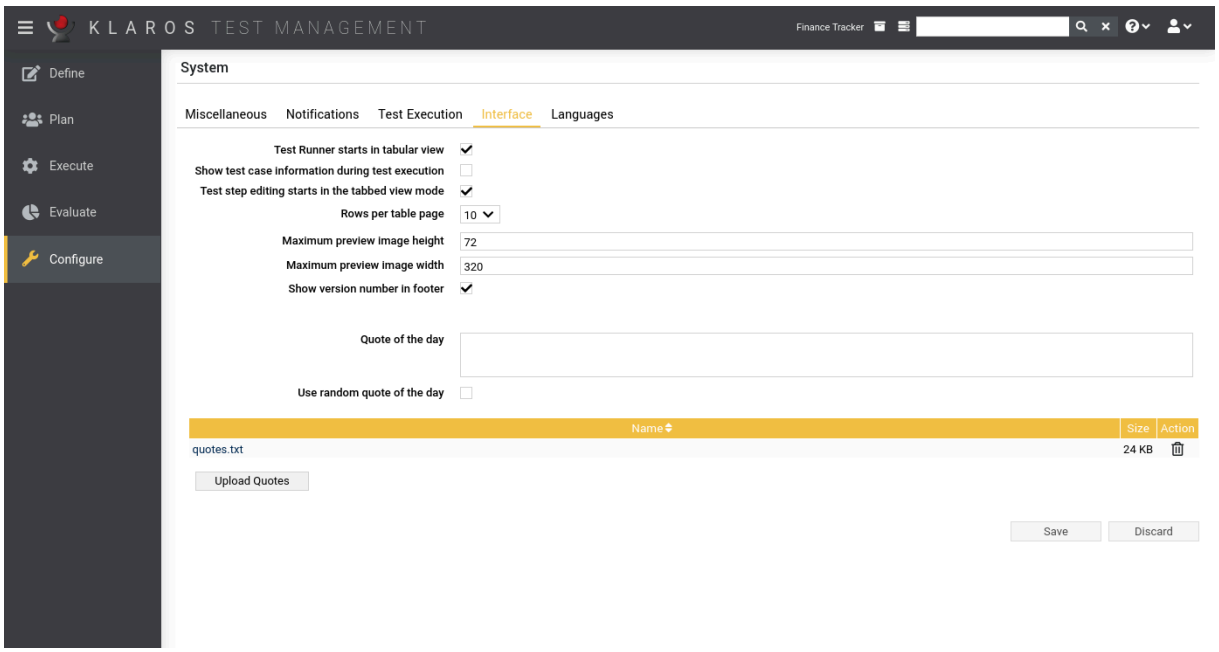


Figure 10.17. The “Interface” Tab

The following settings can be configured here:

Test Runner starts in tabular view — If enabled, the test runner will per default show the tabular view.

If enabled, show test case information during test execution — If enabled, the test case detail information panel is always expanded per default when launching or executing tests.

Test step editing starts in the tabbed view mode — If enabled, the default view in the *Steps* tab in the *Test Case Details* page will be the tabular view.

Rows per table page — This option presets the number of rows per table page.

Maximum preview image height — This option sets the maximum height of the preview images for embedded attachments during test case execution (see Figure 6.45, “An Attachment Reference Replaced by a Preview of the Attachment”).

Maximum preview image width — This option sets the maximum width of the preview images for embedded attachments during test case execution (see Figure 6.45, “An Attachment Reference Replaced by a Preview of the Attachment”).

Show version number in footer — If enabled, the version number of the application is displayed in the footer.

Quote of the day — A message to be displayed on the login screen.

Use random quote of the day — When selected, a random line from the file uploaded below will be displayed as the quote of the day.

Upload Quotes — By clicking the Upload Quotes button, a new text file with quotes can be selected and uploaded. The file must contain one quote per line. The uploaded file can be removed and replaced by the supplied file of the application using the icon.

Please refer to Section 4.2, “Quote of the Day” for detailed information on the file format.

10.4.5. Languages

Additional languages can be added to the user interface using language files.

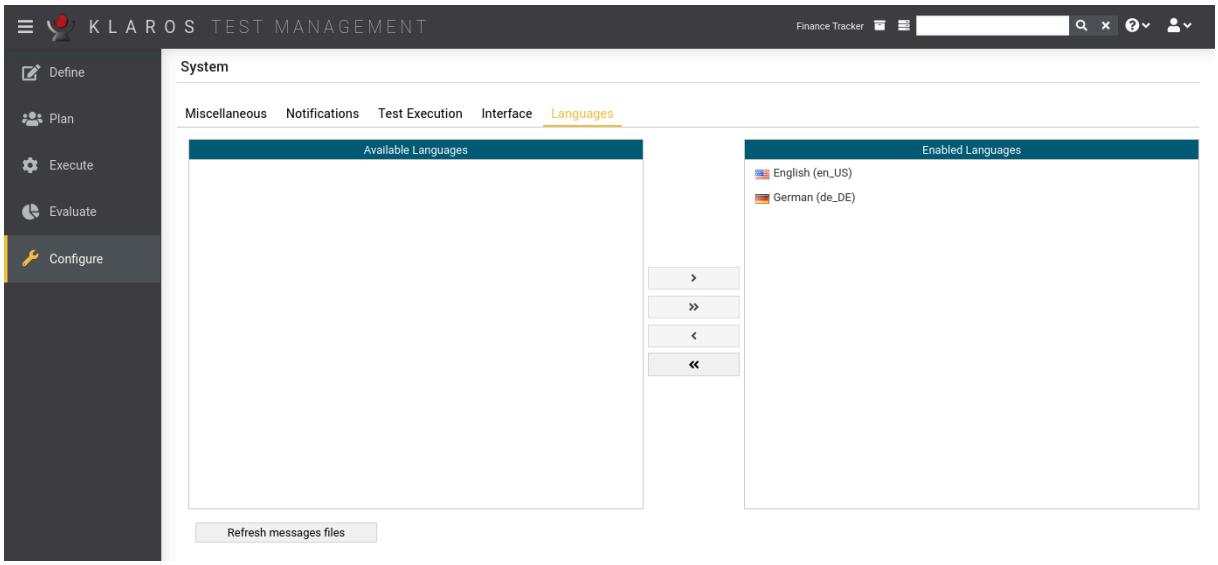


Figure 10.18. The “Languages” Tab

10.4.5.1. Enabling and Disabling Languages

Up to five different languages can be activated at the same time. Clicking the › icon adds the selected language, clicking the ‹ icon removes it. To add all available languages, click on the » or on « to remove all languages or use the drag and drop function.

File Format

Please refer to Section 4.1, “Languages” for detailed information on the file format.

Klaros-Testmanagement comes with language files in German and English. Support for additional languages can be added by using additional language files if needed. The corresponding language file must be stored in the `.klaros/resources/messages` directory for this.

Clicking the Refresh messages files button reloads the language files in the `.klaros/resources/messages` directory while the application is running and displays them in this view.

10.5. Integration

On the *Integration* page, external systems are connected and their settings are managed. It features six tabs, *Issue Management*, *Requirements Management*, *E-Mail*, *Network*, *LDAP* and *CAS*.

10.5.1. Issue Management

An issue management system (also called issue tracker or bug tracker) is an application for managing issues that occur during the testing process. Issues can contain information and reports about errors in a software system.

Issues can be created retrieved and assigned to failed test case results. It is possible to configure and use multiple issue management systems simultaneously.

Klaros-Testmanagement supports the following issue management systems:

Bugzilla, a free open source issue management system (see `http://www.bugzilla.org/`).

GitHub, a web-based Git repository hosting service offering both plans for private repositories or free accounts, which are usually used to host open-source software projects (see `http://github.com/`)

GitLab, a web-based DevOps lifecycle tool that provides a Git-repository manager providing wiki, issue-tracking and CI/CD pipeline features, using an open-source license (see `http://gitlab.com/`).

Jira, is a commercial issue management system produced by Atlassian Pty Ltd. (see `http://www.atlassian.com/`).

Mantis, a free open source issue management system (see `http://www.mantisbt.org/`).

Redmine, a free open source issue management system (see `http://www.redmine.org/`).

Trac, a free open source issue management system (see `http://trac.edgewall.org/`).

YouTrack, is a commercial issue management system produced by JetBrains (see `https://www.jetbrains.com/youtrack/`).

External Configuration Required

Some issue management systems need to be configured before they can be successfully connected to Klaros-Testmanagement. For more information, see Section 3.14, “Configuring External Issue-Management-Systems”.

Token based authentication

Some issue management systems like Jira, GitHub, GitLab, Redmine and YouTrack require access via token-based authentication. Authentication via username and password is not supported in this case.

Klaros-Testmanagement is supporting a token based authentication scheme as well. Just use the required username and the token as password.

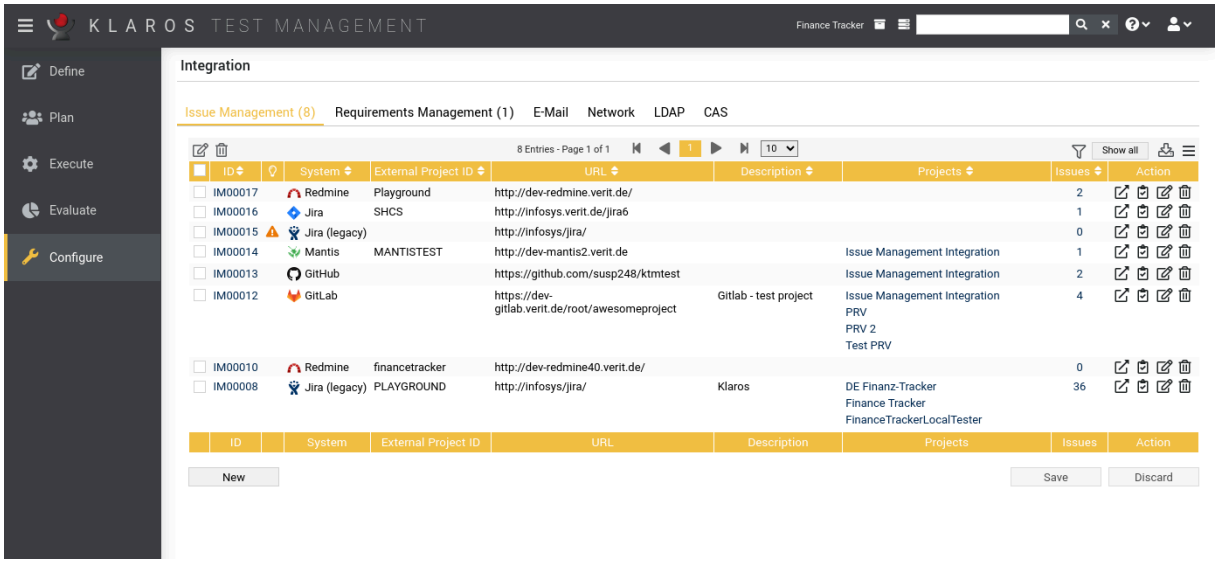


Figure 10.19. The “Issue Management” Tab

The table shows the following values:

ID — Assigned automatically.

Additional Information — A tooltip appears when the cursor is placed over the icon shown here.

⚠ no credentials for background synchronization has been provided.

System — The used issue management system (Bugzilla, Jira (Version 4.3 or higher), Jira Legacy (Version 4.2 or lower), GitHub, GitLab, Mantis, Redmine and Trac).

URL — The link to the issue management system.

External Project ID — Die Projekt-ID des Issue-Management-Systems. See Section 10.5.1.1, “External Project ID”.

Description — Additional information on the issue management system.

Projects — The projects in which the issue management system is used.

Actions — The executable actions.

10.5.1.1. External Project ID

If the issue management system organizes issues in projects, the project ID is required to address the specific project. Jira, Redmine, Mantis, Bugzilla and YouTrack use such a project id. With GitHub, GitLab, and Trac, the URL is used to address different projects.

The valid project ID values can be found in the issue management systems at the following locations:

10.5.1.1.1. Bugzilla

The Bugzilla project id consists of the `Product` field value.

Figure 10.20. The Bugzilla Project ID

10.5.1.1.2. Jira

The Jira project id consists of the `Key` field value.

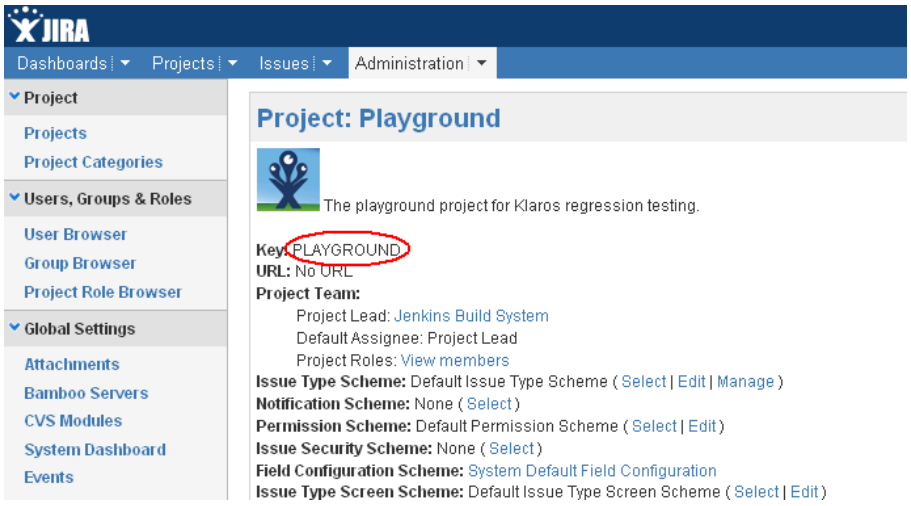


Figure 10.21. The Jira Project ID

10.5.1.1.3. Mantis

The Mantis project id consists of the `Project Name` field value.

Figure 10.22. The Mantis Project ID

10.5.1.1.4. Redmine

The Redmine project id consists of the `Identifier` field value.

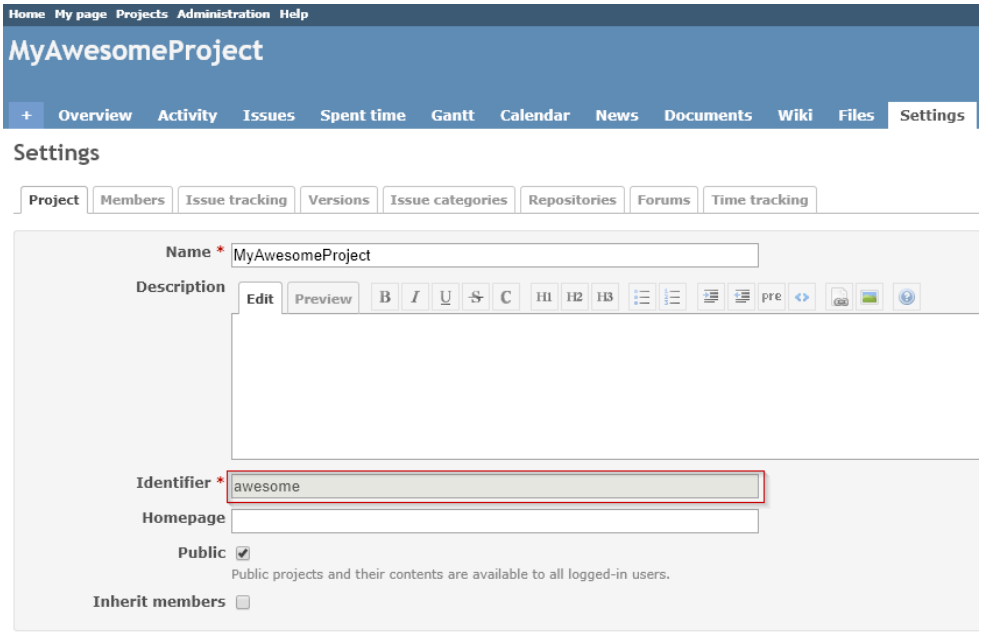


Figure 10.23. The Redmine Project ID

10.5.1.1.5. YouTrack

The YouTrack project id consists of the `ID` field value.

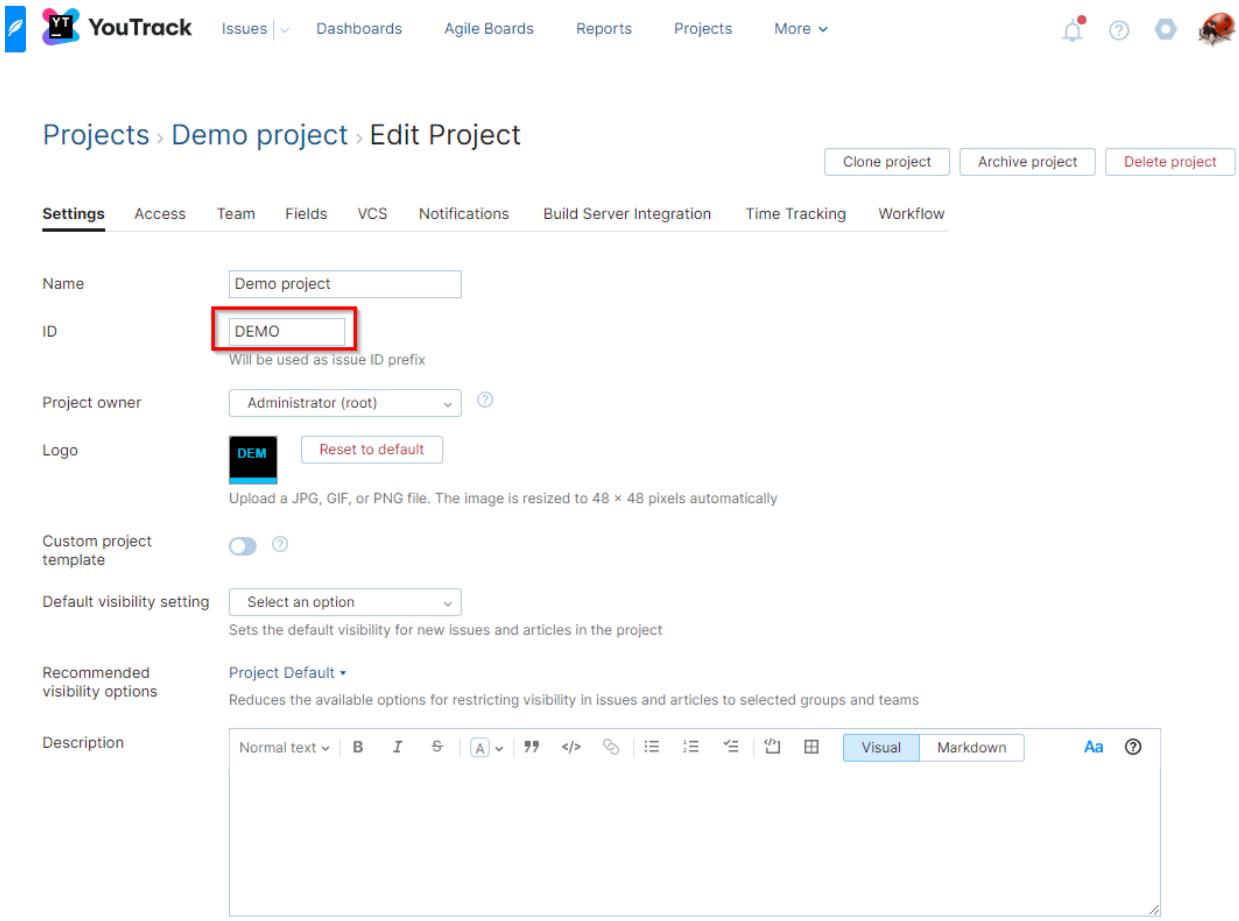


Figure 10.24. The YouTrack Project ID

10.5.1.2. Adding a new Issue-Management-System

By clicking on the button `New` a new empty table row will appear. Attributes of the issue management system can be defined here.

With `Save` the issue management system saved. The ID of the issue management system (IM00001) is automatically assigned by Klaros-Testmanagement.

With `New` several table rows can be created and edited at the same time. Only when clicking on `Save` the data is stored in the database.

With `Discard` all changes are undone.

Red IDs

All rows with red IDs have been changed and are not yet saved!

10.5.1.3. Actions

The following actions can be performed in the action column:

- Open external link to the issue Management System

- Validate the URL (see Section 10.5.1.5, “Editing an existing Issue-Management-System”)
- Edit (see Section 10.5.1.6, “Editing an existing Issue-Management-System”)
- Delete

10.5.1.4. Table Operationen

The following operations can be performed in the line above the table on the right:

- Filter / Sort

  Only active / Show all
- Export
- Column selection

All operations are described in detail in Section 5.2.3.1, “Overview Page”.

10.5.1.5. Editing an existing Issue-Management-System

In the *Action* column, click the icon to check whether a connection to the issue management system can be established.

Trailing Slashes of URLs

In case of an authentication error, first check whether the addition or deletion of a trailing slash to the URL resolves the problem.

10.5.1.6. Editing an existing Issue-Management-System

Each issue management system created has its own detail page with further subviews. By clicking on the ID of the respective issue management system or on the icon on the right side of the action column you will get to the last selected detail view. When called for the first time, this is the Properties tab.

The following tabs are available: *Properties*, *Projects*, *Attachments* and *Changes*.

10.5.1.7. Actions

On the detail pages there are more icons in the upper right corner. The following actions can be performed here:

Open external link	Shows the request in the external source.
Browse	Use the green arrows at the very top right to switch between the issue managements systems present on the previous page.

10.5.1.8. Properties

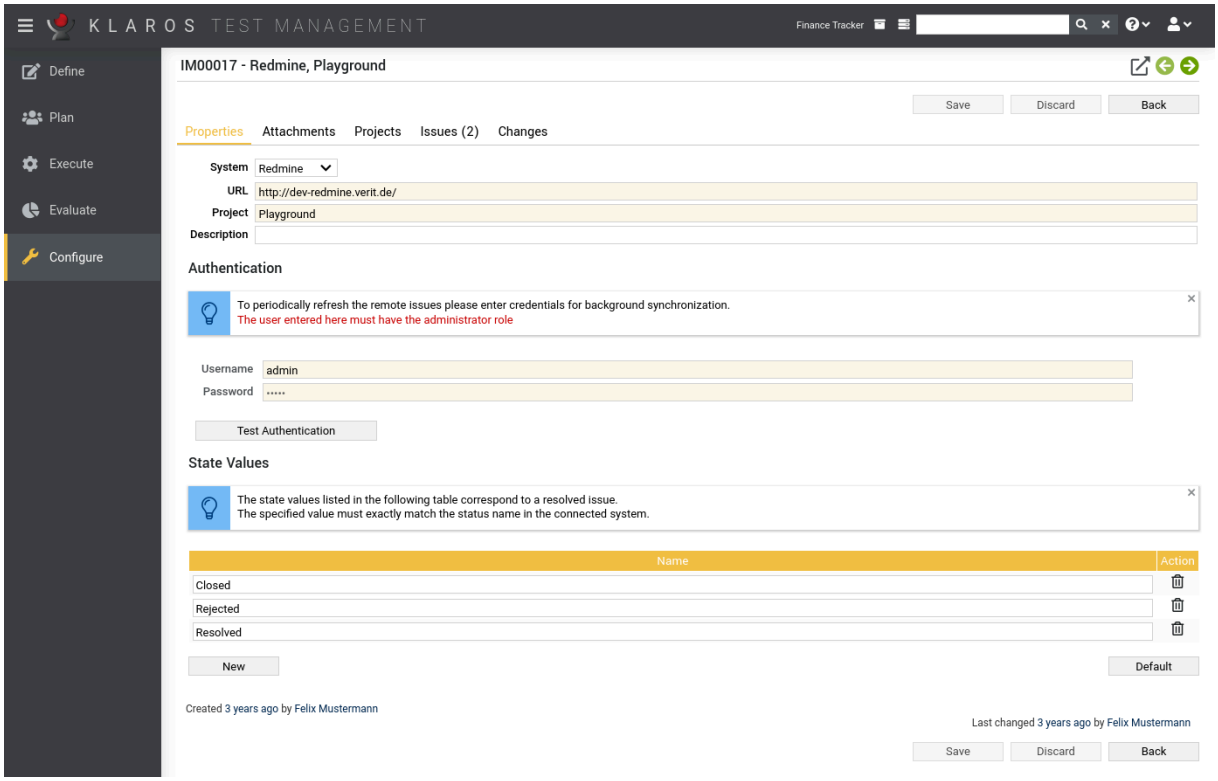


Figure 10.25. The “Properties” Tab

The *Properties* view is divided into three sections: Issue Management System, Authentication, Status Names

Issue-Management-System

System A drop-down menu can be used to select the desired system.

URL The URL where the system can be reached.

Project The name of the project.

Description An optional description of the system.

Below this is the "Authentication" section. Here you enter the login information for the background synchronization. The user entered here must have full access to the entered project.

Username The valid username.

Password The valid password.

The Test login button can be used to check whether the entries are correct.

Important

The credentials entered here will solely be used for background synchronization. If users creates or updates an issue, their personal credentials will be used to authenticate against the issue management system. If credentials are missing or not valid, then users are prompted to enter them.

The last section is "Status names". The status names listed in the following table correspond to a completed issue. The specified value must exactly match the status name in the connected system.

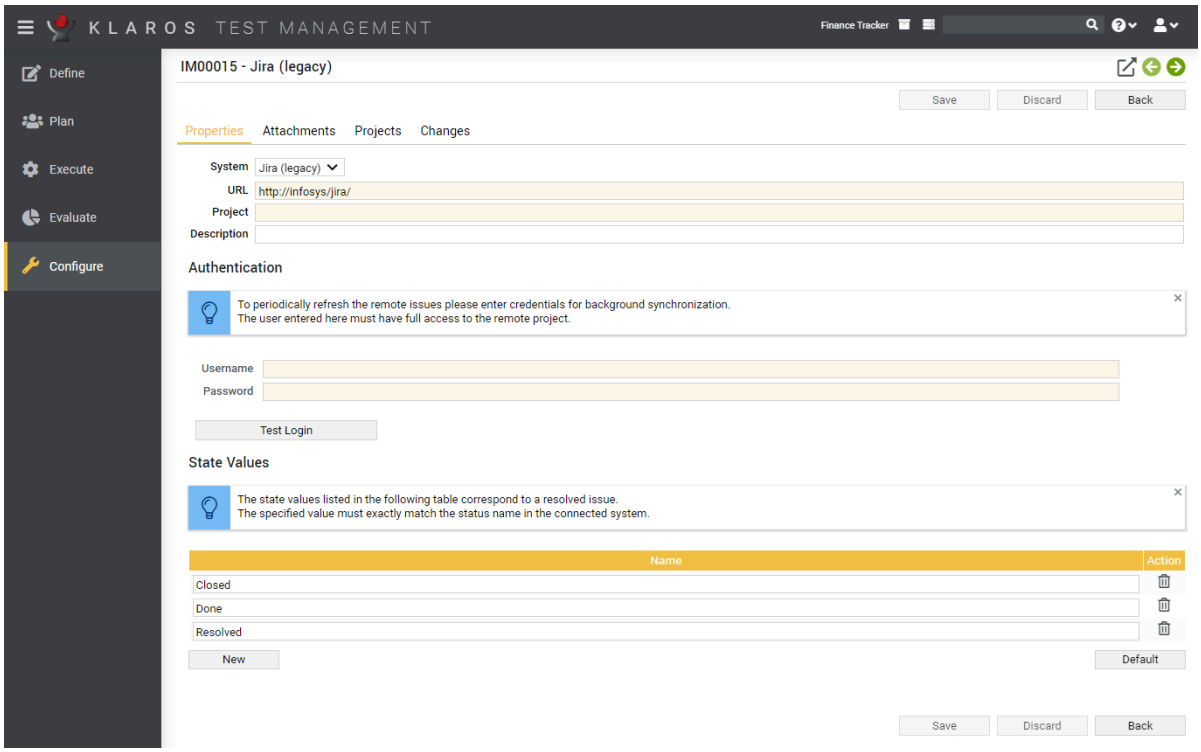


Figure 10.26. Edit the status mappings

The `New` button can be used to add a new status name.

The `Default` button resets the fields to the default values.

All entries are transferred to the database with the `Save` button. The `Discard` button deletes the entries. The button `Back` . takes you to the overview page with the created issue management systems.

10.5.1.9. Attachments

You can add any files as attachments to an issue management System. For more information, see Section 5.2.3.2.6, “Attachments”.

10.5.1.10. Projects

The *Projects* view displays all projects to which the Issue-Management-System has been assigned.

10.5.1.11. Changes

The tab *Changes* shows the change history of this issue management system.

For a detailed description of the *Changes* view, see Section 5.2.3.2.8, “Change History”.

10.5.1.12. Assignment of status values for completed issues

Each issue has a status field that documents its current state (e.g. "In progress" or "Fixed"). The individual statuses and their naming vary according to the issue management system used. Klaros-Testmanagement only distinguishes between open and resolved issues. An issue is considered closed when no further work needs to be done on it (e.g. "Closed", "Rejected"), otherwise it is still open.

Resolved issues are displayed in green, open ones in red. Section 9.6.5, “Details Page”

The resolved issue states must be named exactly as they are in the issue management system, otherwise they will not be properly identified and will be counted as open issues.

10.5.2. Requirements Management

Requirements can be managed either locally in Klaros-Testmanagement or in a remote system.

Local Requirements Management — By default, test managers can create, edit, and delete requirements for a project from within the project.

Remote Requirements Management — A project can also receive its requirements from an external requirement management system (RMS). If this is configured, the creation, editing and deletion of requirements can only be done via the connected RMS.

Currently Klaros-Testmanagement supports the following external requirement management systems:

Jira 4.3 or newer — A commercial issue management system produced by Atlassian Pty Ltd., see `http://www.atlassian.com/`.

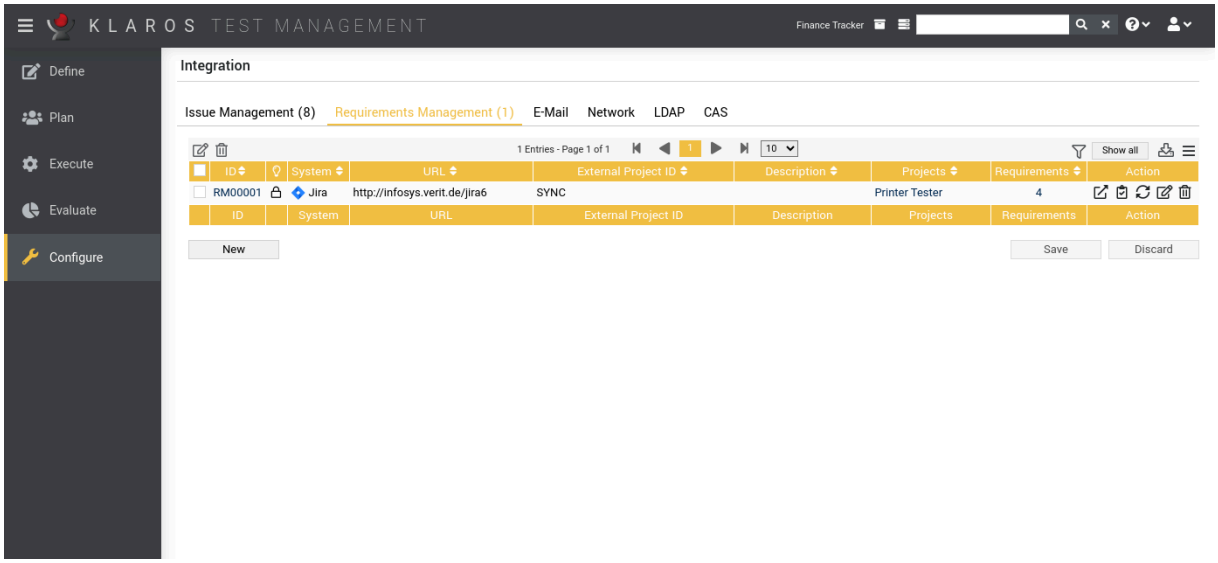


Figure 10.27. The “Requirements Management” Tab

The table shows the following values:

ID — Assigned automatically.

Additional Information	A tooltip appears when the cursor is placed over the icon shown here. 🔒 the requirement management system be deleted because it is connected with at least one project. ⚠ no credentials for background synchronization has been provided.
System	The used requirements management system (Jira Version 4.3 or higher).
URL	The link to the requirement management system.
External Project ID	See Section 10.5.1.1, "External Project ID".
Description	A short description of the requirement management system.
Projects	The projects in which the issue management system is used.
Actions	The executable actions.

10.5.2.1. Adding a new Requirements Management System

By clicking on the button `New` a new empty table row will appear. All attributes of the requirement management system can be defined here.

With `Save` the requirement management system is saved. The ID of the requirement management system (RM00001) is automatically assigned by Klaros-Testmanagement.

With `New` several table rows can be created and edited at the same time. Only when clicking on `Save` the data is stored in the database.

With `Discard` all changes are undone.

Red IDs

All rows with red IDs have been changed and are not yet saved!

10.5.2.2. Actions

The following actions can be performed in the action column:

- Validate the URL (see Section 10.5.1.5, "Editing an existing Issue-Management-System")
- Refresh requirements
- Edit (see Section 10.5.2.3, "Editing an existing Requirements Management System")
- Deactivate

Synchronizing Attachments

When connected to a Jira server, attachments from its requirements are automatically transferred to Klaros-Testmanagement during synchronization.

10.5.2.3. Editing an existing Requirements Management System

Clicking on the icon in the action column on the right opens a dialog. The credentials for the background synchronization can be provided here.

10.5.3. E-Mail

In the *Email* tab the e-mail server settings of the application are configured. These settings are required for sending notification e-mails (see Section 10.4.2, “Notifications”). The attributes that can be edited are shown in Figure 10.28.

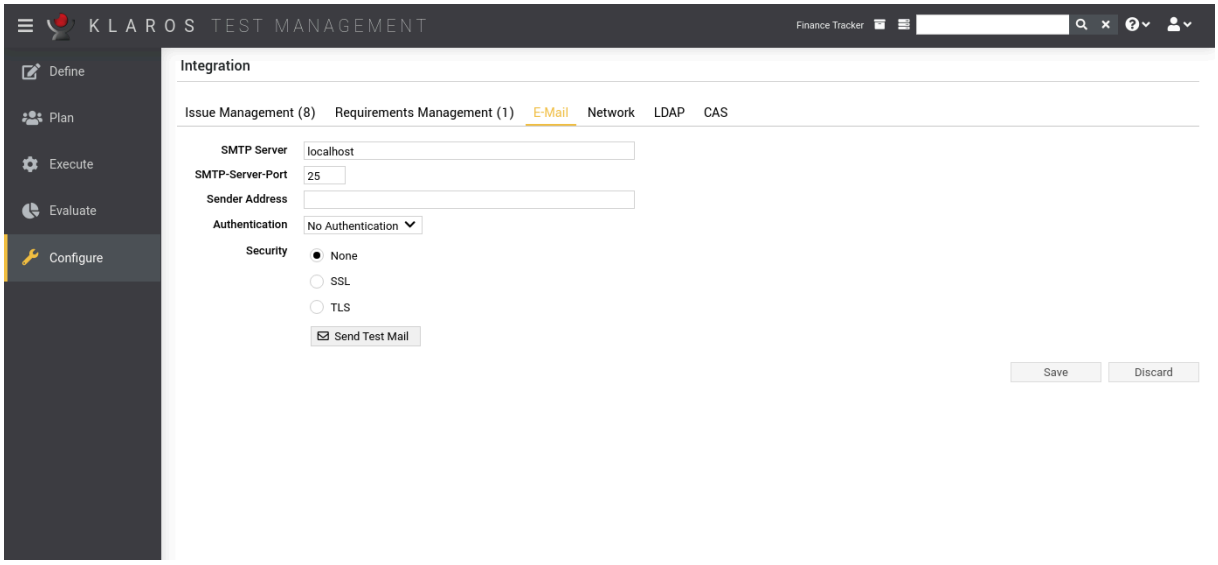


Figure 10.28. The “E-Mail” Tab

Note

When in doubt, please consult your system administrator about the values to enter here.

SMTP Server — The host name of your mail server.

SMTP Server Port — The port where your mail server is listening. This value usually depends on the transport security setting chosen below. The default value is 25 for no transport security.

Sender Address — The sender address of the generated e-mails (*From* field).

Username — The username needed to authenticate against the SMTP server. This field is only present if SMTP or POP authentication is selected in the *Authentication* field below.

Password — The password needed to authenticate against the SMTP server. This field is only present if SMTP or POP authentication is selected in the *Authentication* field below.

Authentication — To prevent or identify spam, some mail servers require users to authenticate themselves before they allow them to send mails.

This option selects whether no authentication is required, or a username / password combination via SMTP or POP is used. If one of the latter two is selected, the *User Name* and *Password* fields are shown in the user interface.

Security This setting defines the transport layer security used to access the SMTP mail server. Each transport security setting may require a different port value. The options are *None* (Port 25), *Secure Socket Layer / SSL* (Port 465) or *Transport Layer Security / TLS* aka STARTTLS (Port 25). Your local port settings may vary from the given defaults.

Send Test Mail This link sends a test email to verify that your e-mail server settings are valid. It is therefore necessary that a valid e-mail address is specified in your user account.

Click the *Send Test Mail* button to test if the fields are filled in properly.

10.5.4. Network

Klaros-Testmanagement supports the use of HTTP and SOCKS proxies for external network connections. Proxy settings are configured in the *Network* section.

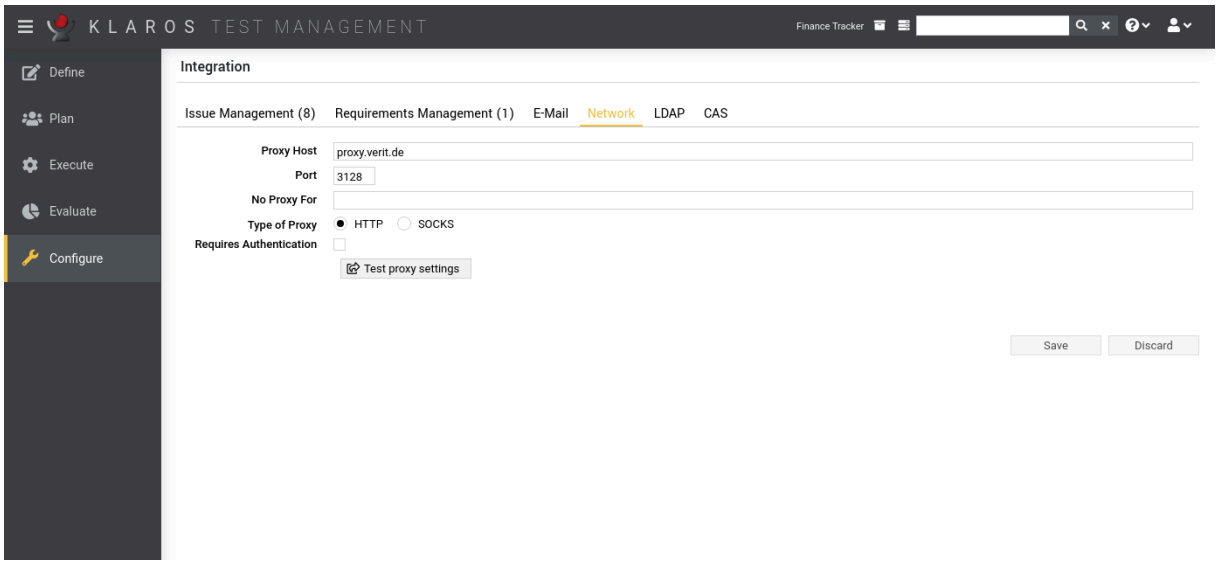


Figure 10.29. The “Network” Tab

Setting the application URL

If Klaros-Testmanagement is run behind a proxy, the application URL must be set accordingly in order for Klaros-Testmanagement functioning properly. The Application URL can be set in the *Miscellaneous* tab of *General Settings* (see Section 10.4.1, “Miscellaneous”).

The properties of the proxy settings are:

Proxy Host The network address or hostname of the proxy server.

Port — The port that the proxy server is active on.

No Proxy for — A |-separated list of domains which should bypass the proxy settings.

Type of Proxy — Possible settings are HTTP proxy or SOCKS proxy.

Requires Authentication — Specifies whether the proxy requires authentication or not.

Username — The username to use for the username/password authentication.

Password — The password to use for the username/password authentication.

The proxy settings can be tested by clicking the Test proxy settings button.

10.5.5. LDAP

Only available in Klaros-Testmanagement Enterprise Edition

Tip

To access an LDAP or Active Directory server, it is necessary to set numerous configuration parameters first. When in doubt, please consult your system administrator about the values to enter here.

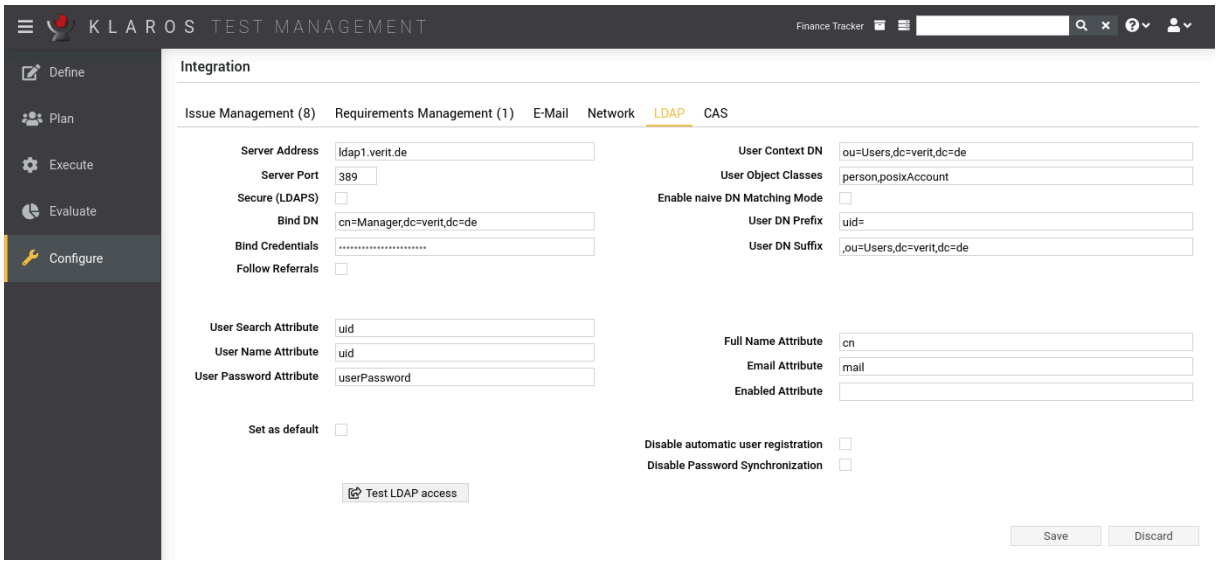


Figure 10.30. The “LDAP” Tab

Parameters required to contact the LDAP server:

Server Address — The URI under which the LDAP server resides (e.g. `ldap.acme.com`).

Server Port — The port on which the LDAP server is located (typically `389` for unencrypted connections or `686` for LDAPS).

Secure (LDAPS) — If this option is enabled, the encrypting LDAPS protocol is used.

Bind DN — The Distinguished Name of the user account that will log on to the server and execute the bind operation.

Bind Credentials — The password credential of the user account that will log on to the server and execute the bind operation.

Follow Referrals — If this option is enabled, searching a directory automatically follows any referrals the server might return. When disabled referrals will be ignored and other servers will not be contacted during the search.

Parameters needed to locate a user account:

User Context DN — The distinguished name of the path under which user accounts will be searched (e.g. `ou=Users,dc=verit,dc=de`).

User Object Classes — A comma separated list of the LDAP object classes a user account must match to be included in the search (e.g. `person,posixAccount`).

Enable naive DN Matching Mode — If this option is enabled, the returned DN of a user search is directly used to authenticate the user. If this option is disabled, the following two parameters will be used in conjunction with the *User Name Attribute* to build a DN to authenticate with.

User DN Prefix — The prefix of the distinguished name used to locate user accounts (e.g. `uid=`).

User DN Suffix — The suffix of the distinguished name used to locate the user account (e.g. `,ou=Users,dc=acme,dc=com`). When locating user accounts, the prefix, the account id and the suffix are concatenated to form the distinguished name of the user account.

Parameters describing the attributes of a user account:

User Search Attribute — The LDAP username attribute which corresponds to the Klaros-Testmanagement account name (e.g. `uid`).

User Name Attribute — The LDAP attribute which will be used in the DN bind action which authenticates the user (if naive DN Matching mode is deactivated). In a simple scenario this will match the *User Search Attribute*.

If your LDAP Server setup does not allow you to bind a user with the specified user search attribute you should specify the corresponding attribute here (e.g. `cn`) and use this in conjunction with the corresponding user DN prefix/suffix.

User Password Attribute — The LDAP password attribute which corresponds to the Klaros-Testmanagement account password (e.g. `userPassword`).

Full Name Attribute — The LDAP attribute containing the full name of the user (e.g `cn`). If specified, this will be automatically be transferred into the Klaros-Testmanagement database upon first successful login.

Email Attribute — The LDAP attribute containing the email address of the user account (e.g. `mail`). If specified, this will automatically be transferred into the Klaros-Testmanagement database upon the first successful login.

Enabled Attribute — If specified, this LDAP boolean attribute defines whether a user is allowed to log in. Not all directory servers provide such an attribute.

If the *Set as default* checkmark is activated, the login screen will default to LDAP authentication for all users. It is still possible for existing users to authenticate against the Klaros-Testmanagement user database if selected in the login screen.

Use the *Disable automatic user registration* option to prevent the automatic creation of a user in Klaros-Testmanagement upon the first successful login.

Upon the first successful login a matching password hash is created in the local user database so that users can also authenticate themselves against the local user database with their LDAP password. If the *Disable Password Synchronization* checkmark is activated, this synchronization will not be performed and users will only be able to log in locally when an administrator assigns them a local password interactively.

Clicking the *Test LDAP access* button tests whether the parameters entered on this page are correct. The test process in divided in two phases:

In the first phase, an attempt is made to log on to the server and a search for all available users is performed. If this is successful, a dialog is displayed listing the users found in the LDAP directory.

In the second phase, a username and password can be entered to test the LDAP login of a user. The result of the login attempt is logged in the log panel.

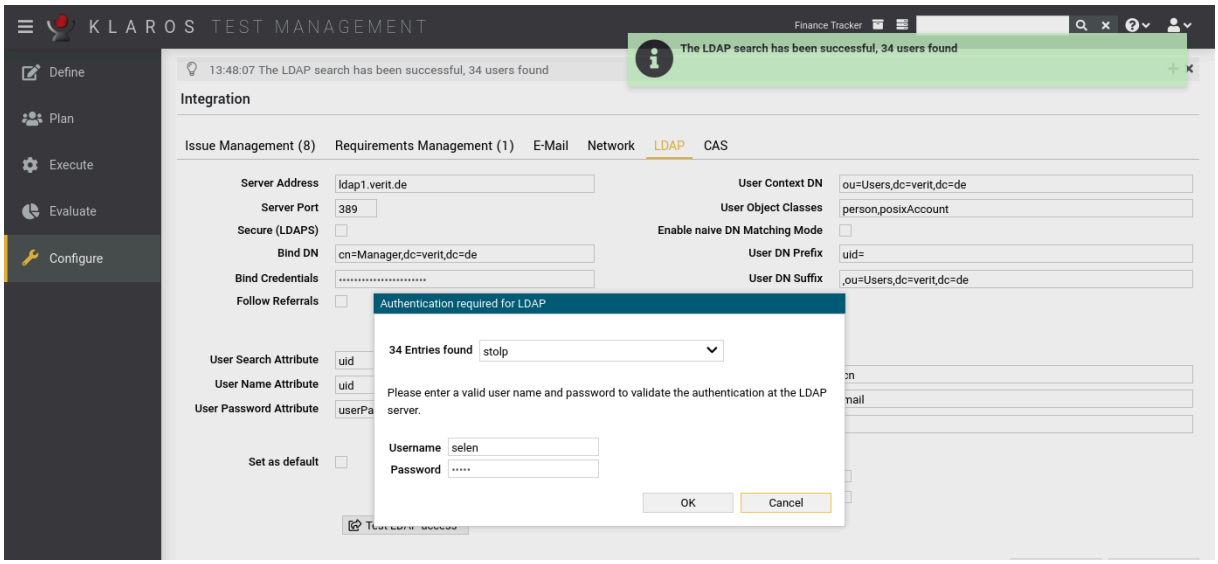


Figure 10.31. The “LDAP Authentication” Dialog

10.5.6. CAS

Only available in Klaros-Testmanagement Enterprise Edition

Klaros-Testmanagement supports Single sign-on Authentication (SSO) using a Central Authentication Service (CAS).

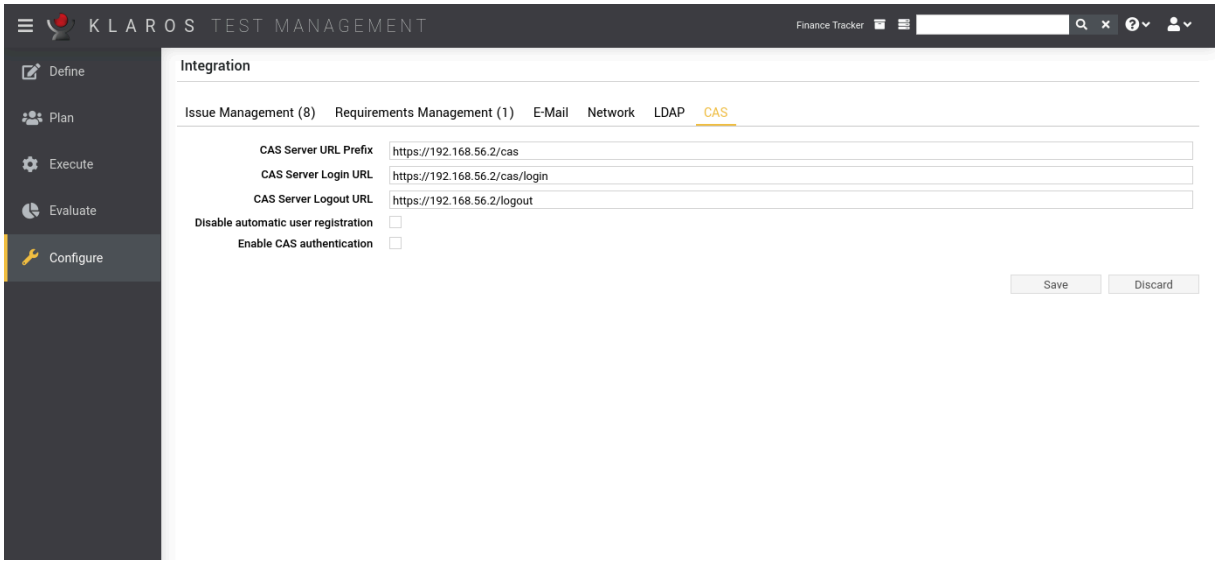


Figure 10.32. The “CAS” Tab

Parameters needed to contact the CAS server:

CAS Server URL Prefix — The URL under which the CAS server resides (e.g. `cas.acme.com`).

CAS Server Login URL — The URL of the CAS server for single sign-on logins. A user that has yet not been authenticated by CAS will be redirected to this URL.

CAS Server Logout URL — The URL of the CAS server for single sign-on logout. When a user logs out from Klaros-Testmanagement he is redirected to this URL.

Disable automatic user registration — When activated, a successful CAS authentication does not automatically create a guest user account in the system.

Enable CAS authentication — When activated, all authentication is delegated to the CAS Server and the default login window is no longer reachable. To complete the activation, a reboot of the application server is required.

CAS Single Sign-on Disables Local Authentication

Once CAS support is activated and Klaros-Testmanagement has been rebooted it is no longer possible to authenticate using the Klaros-Testmanagement login screen.

Please double-check that the account verification works before activating. In case of a broken setup please set the property `cas.enabled` in the configuration file `%KLAROS_HOME%/.klaros/klaros.properties` to `false` and reboot the application server. This restores the default behavior.

10.6. Backup

On the *Backup* page, complete projects can be backed up and restored in XML format. Only administrators and managers are authorized to import or export projects.

10.6.1. Export

In this tab one or more projects can be selected and exported by clicking the Export button (Figure 10.33).

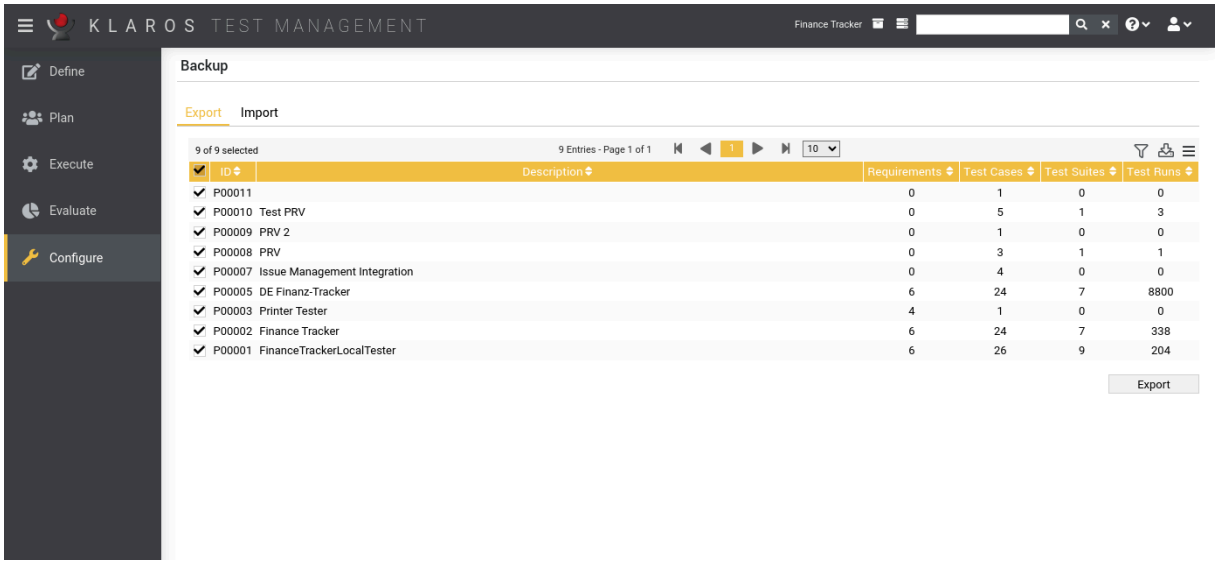


Figure 10.33. The “Export” Tab

Clicking the Export button exports the selected projects to an XML formatted output file.

Note

The backup files are marked with the respective database version that was exported. Backup files may only be imported if the database version of the Klaros installation matches the version of the backup file.

10.6.2. Import

In this tab, projects from backup files can be imported. By clicking the Upload Backup File button, a backup file can be uploaded. After uploading, the projects that are contained in the file are displayed. One or more of these projects can be selected (Figure 10.34).

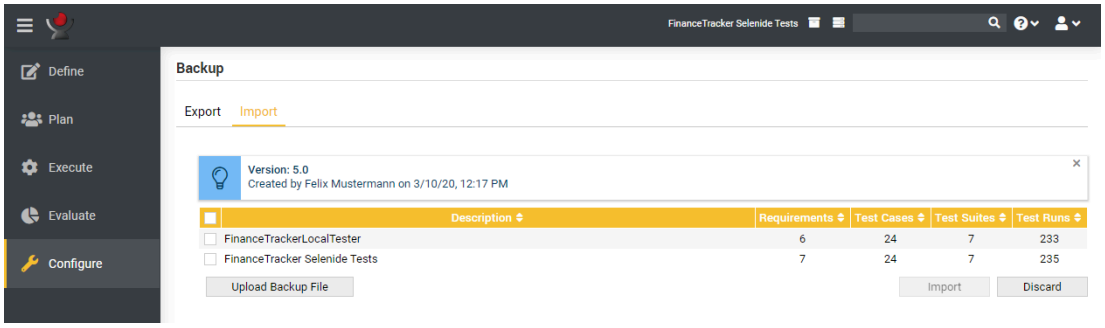


Figure 10.34. The “Import” Tab

Clicking the Import button imports the selected projects and clicking the Discard button cancels the restore.

Note

An import will never overwrite existing projects or other data therein.

Important

When trying to import a backup file from a previous Klaros database version, the import is rejected with a corresponding error message. You must first import this backup file with a version of Klaros-Testmanagement that matches the backup file, and then update the installation to your current version.

An export from that version is now compatible with your current release.

Chapter 11. Custom Reports

Only available in Klaros-Testmanagement Enterprise Edition

Although Klaros-Testmanagement already offers a large number of predefined reports, these are often not sufficient if there are special requirements regarding content or layout. For this reason, the Klaros-Testmanagement Enterprise Edition can also be used to create your own reports. Figure 11.1, “The Report Generation Process” gives an overview of the reporting process.

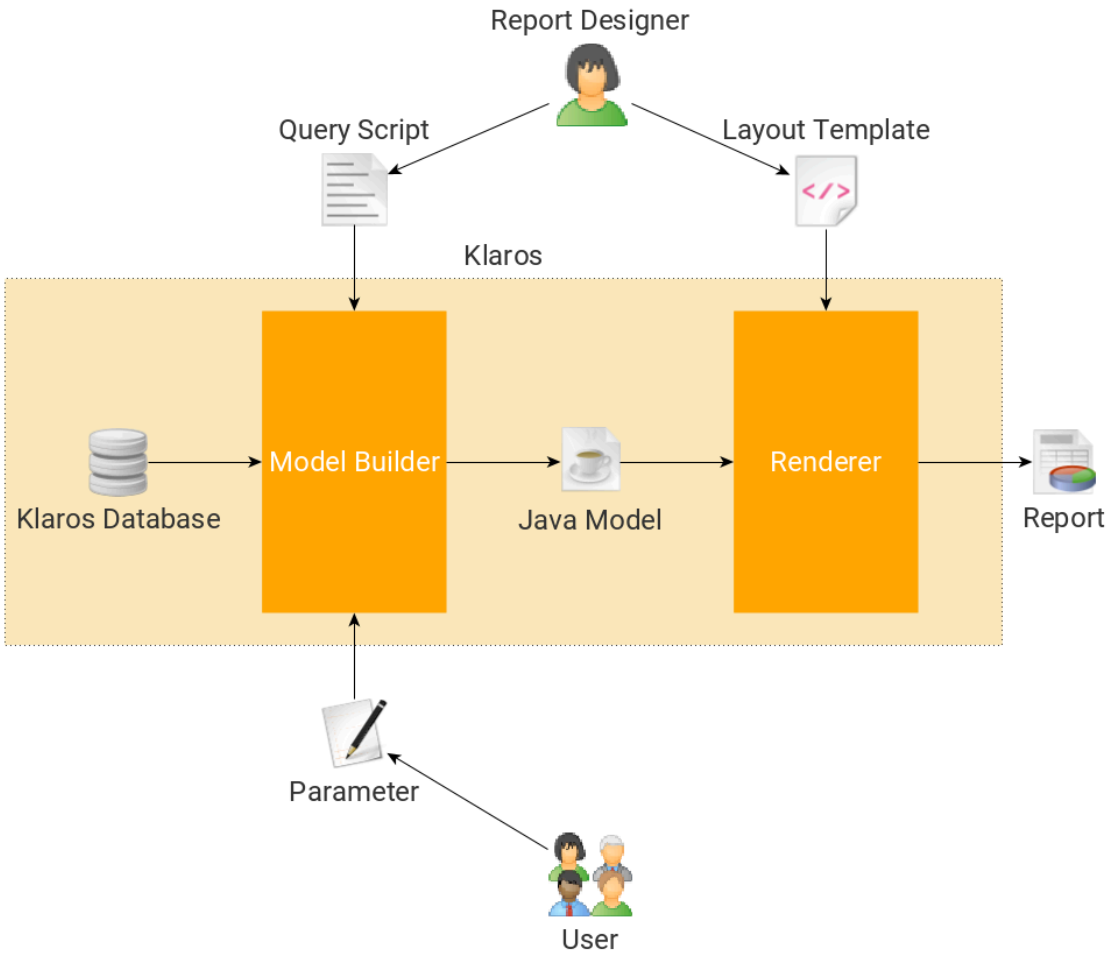


Figure 11.1. The Report Generation Process

Two user groups are involved in the generation of a report: `Report Designer` and `User`.

Programming Knowledge

To create custom reports, basic knowledge of the Java or Groovy programming languages and XML-based template creation with SeamPDF or SeamExcel is required.

- The **report designer** determines the parameters of a report and defines which data from the database should be used for this purpose. It also defines the layout and content of the report template.

- The **user** selects the parameters with which the report is to be created and retrieves the report as a PDF or Excel file.
- The **layout** template defines the layout and content of the report and is defined in an XML-based description language.
- The **report script** extracts the data from the database and prepares it for the use of the data in the report template. The report script retrieves and formats the data and is formulated in Groovy or Java. It is provided by a class that implements the Section E.2, "KlarosScript Interface". This interface defines a single method called `execute` and uses a Section E.2, "KlarosScript Interface" object as an input parameter.

This chapter covers the creation of user defined report templates so it primarily of interest to report designers. Section 9.2, "Reports" shows how users can generate reports from them.

11.1. The Context Object

The report script provides the data for the rendering of the layout template. It retrieves the data either directly from the object model de.verit.klaros.core.model or from the database via the HQL query language.

The resulting data for the layout template must then be stored in the Section E.2, "KlarosScript Interface" object which has been passed to the execute method.

If parameters are defined, they are available via the context object to both the report script and the layout template.

Predefined Objects in the Context

The Section E.2, "KlarosScript Interface" object already contains predefined objects. For a list please refer to Section E.1, "The Klaros Report Context".

11.2. Creating a Report Template

Go to the *Configure* section and select the *Report Templates* page.

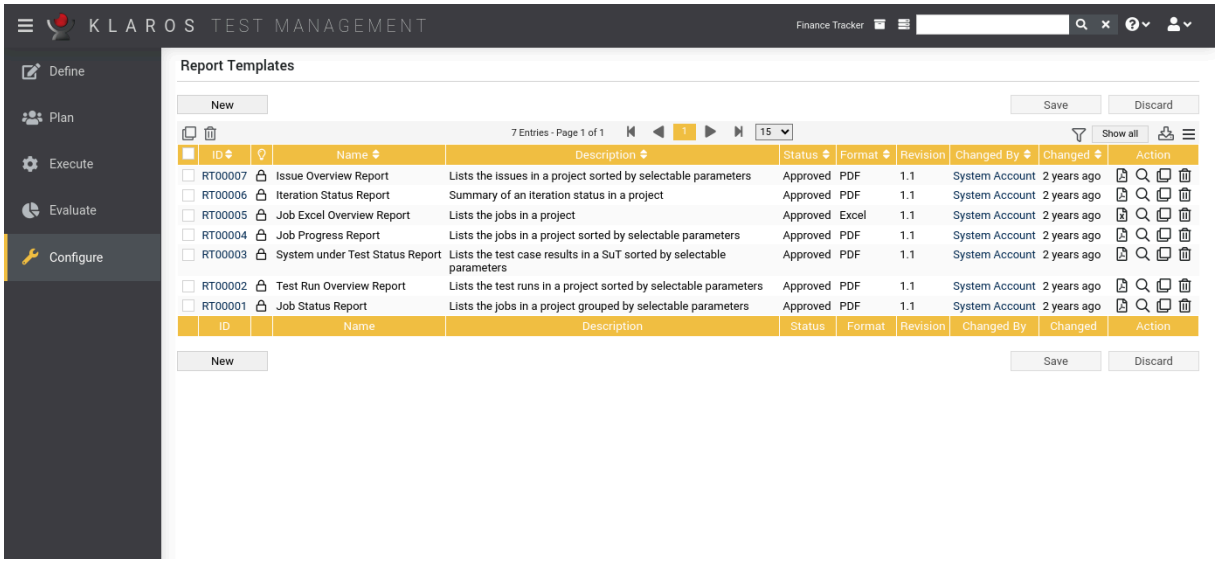


Figure 11.2. The “Report Templates” Page

A click on the button New creates a new report template. Here you can enter a *name* and a short *description* for the report.

Name The name of the report.

Description A description of the report.

Output Format The output format: PDF or Excel.

Status The status of the report: Draft or Approved.

Klaros-Testmanagement comes with several predefined report templates These are read-only, but can be duplicated with the icon. The duplicate can then be modified as desired.

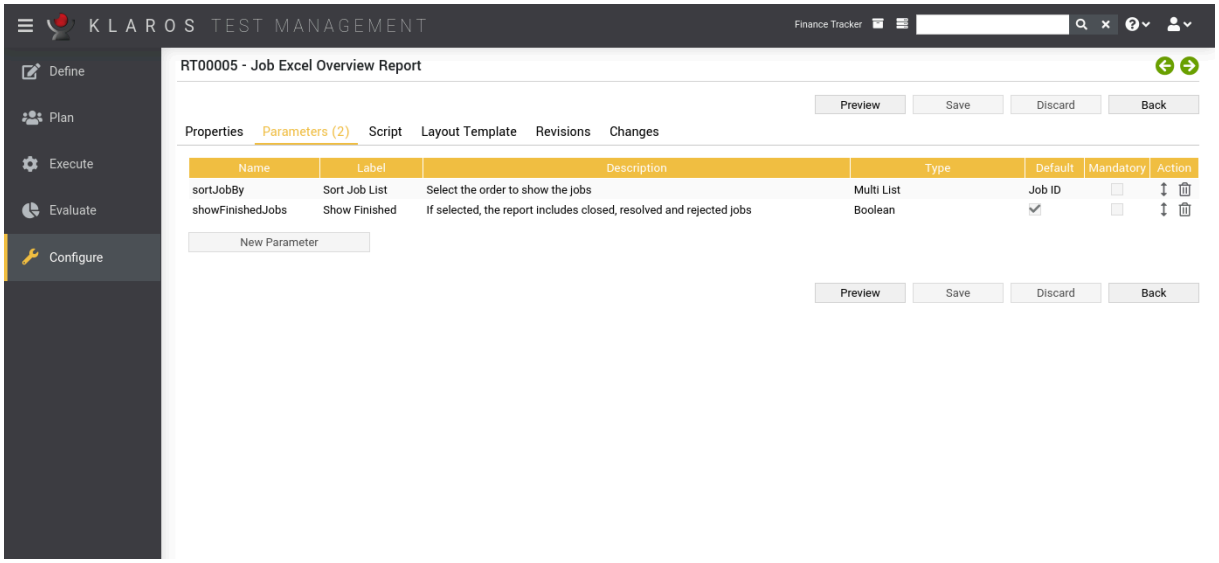


Figure 11.3. The “Report Templates Details” Page

- The ***Parameters*** view shows the parameters defined for this template. For newly created reports, this list is empty.

- The ***Script*** tab manages the report script. Newly created reports already have a predefined class with the necessary methods.
- The layout template is managed in the ***Template*** view. As with the report script, a basic layout already exists for new reports.

Note

For larger report templates it can be helpful to use an IDE (e.g. Eclipse) for the script and layout template. The created files can then be uploaded in Klaros-Test-management. To avoid errors, add the Klaros model libraries to the build path of your Eclipse project. The tutorial document has a chapter about creating report templates where this is explained in detail.

Instead of editing the report script and template directly in the application, they can also be imported from a file. Specify the file to be used by clicking the Browse button and selecting the file to be imported. A click on *OK* in the file dialog and then on Upload imports the selected file into the script or template field.

This page provides three actions to be executed:

Preview generates a preview of the report.

Save stores the report template.

Discard discard all changes to the report template.

To make reports more flexible, it is possible to pass parameters to the report script. These can be used, for example, to pass a time span for which data is determined.

To add arguments to the report, click the + icon button on the parameters tab.

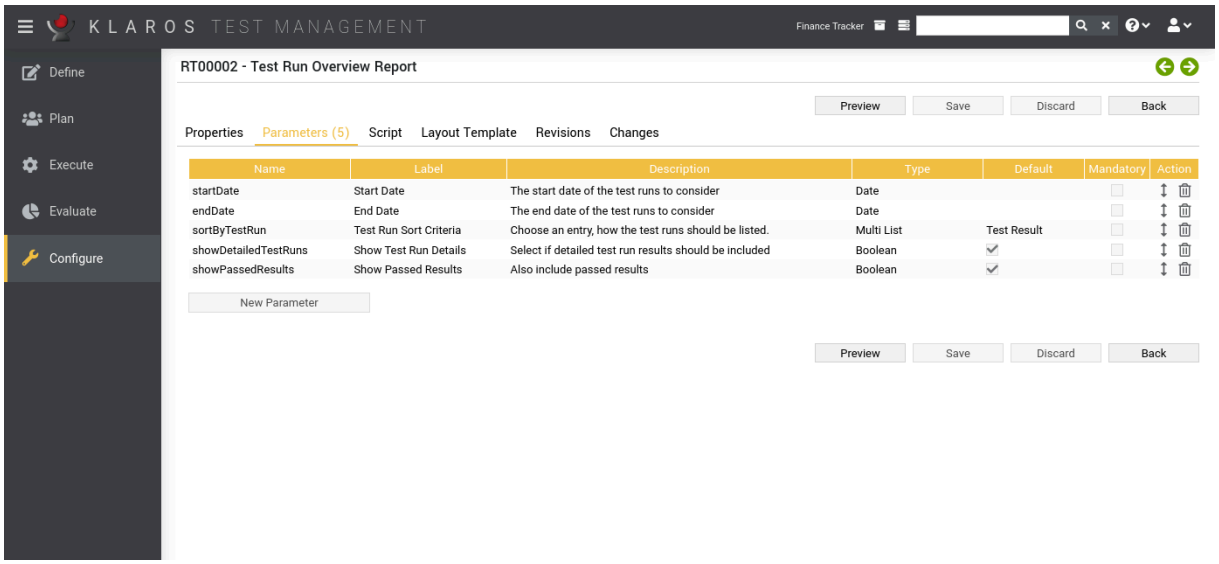


Figure 11.4. The “Parameters” Tab

The type of the parameters can be specified by selecting the appropriate value from the type list. Supported types are `Text`, `Number`, `Date`, `Boolean`, `Selection` and `Multiple Selection`. Default val-

ues can be specified and a mandatory flag can be set. Clicking the icon removes the parameter from the list.

The passed parameters can be accessed by the report script by either calling the `getParameterValue(String name)` or the `getParameter(String name)` method. These methods will return null if no parameter with the specified name could be found.

11.2.1. Supported Parameter Types

The following parameter types are supported:

Text

Number

Date

Selection

Multiple Selection

11.2.2. Dealing with Parameters

As mentioned before, the passed parameters can be retrieved from the context by the `getParameterValue(String name)` and `getParameter(String name)` methods.

The following code snippet shows how to access the parameters and how to use them in the report script:

```
StringBuffer query = new StringBuffer();
  query.append("select tcr from KlarosTestCaseResult tcr where tcr.executionTime < ");
  query.append(context.getParameterValue("executionTime"));
  List<?> tcr = context.executeQuery(query.toString());
```

This query will retrieve all objects of type `KlarosTestCaseResult` that have an execution time smaller than the value passed with the parameter `executionTime`.

Alternately, the parameters can be directly accessed in the query as shown in the following code snippet:

```
query.append("select tcr from KlarosTestCaseResult tcr where tcr.executionTime <:executionTime");
  List<?> tcr = context.executeParameterizedQuery(query.toString());
```

Parameters can be accessed from the layout template in the following way:

```
<p:text value="Test results for test runs with execution time &lt; #{executionTime} ms" />
```

Note

Make sure to escape characters like `&`, `<` or `>` when using them in XML attributes in the layout template. In our example `<` is the escape character for `<`.

11.3. Generating a Report

To generate a report, click on the *Evaluate* icon and select *Reports* from the menu on the left side. A list of available report templates will be shown.

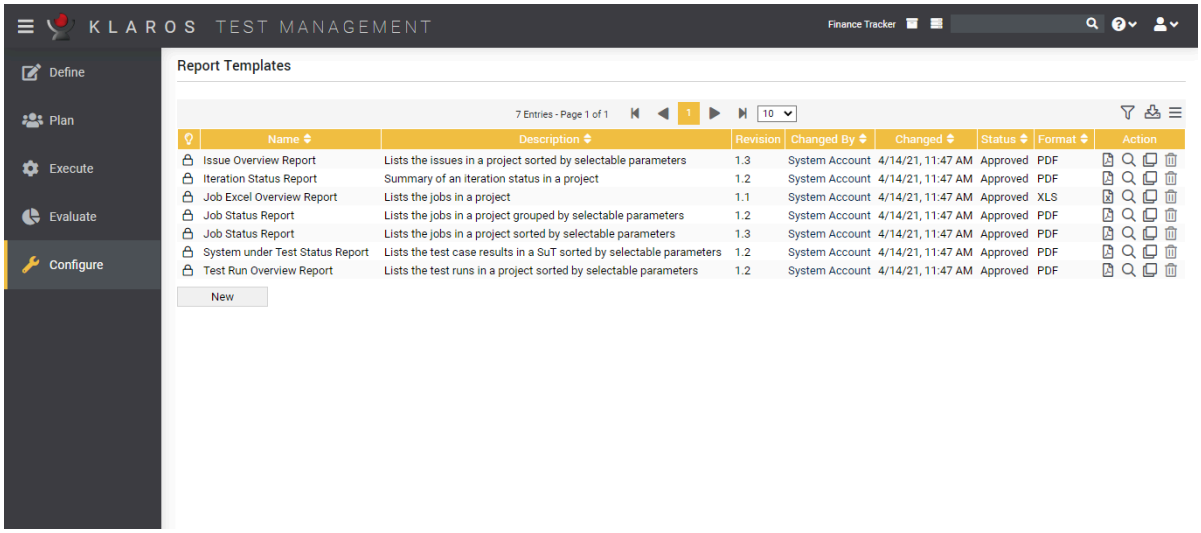


Figure 11.5. Generating a Report

Clicking the or icon of a report causes it to be rendered as a PDF or Excel file. If the report script was defined with parameters, a pop-up window will prompt the user to enter the defined parameters before the report is generated.

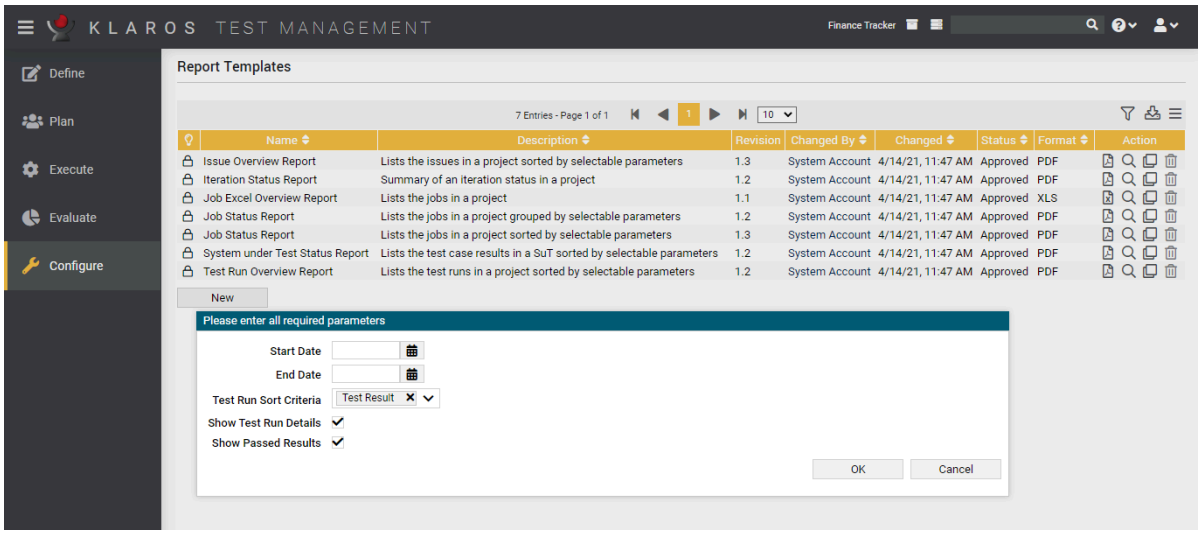


Figure 11.6. Entering Report Parameters

11.4. Example Report

This section provides an example report, which retrieves test case results and displays them depending on their status.

11.4.1. Creating the Report Script

The following code snippet shows the basic structure of a report script with all required imports. The code to retrieve the data will be implemented in the `execute` method. A more detailed description of the Klaros-Testmanagement API can be found in de.verit.klaros.core.model.

```
import de.verit.klaros.scripting.*;
import de.verit.klaros.core.model.*;
import java.util.*;

public class TestScript implements KlarosScript {

  public void execute(KlarosContext context) {...
  }
}
```

The next step is to retrieve the required data. The following code snippet shows how to build a query string to retrieve the data.

```
StringBuffer query = new StringBuffer();
query.append("select tcr from KlarosTestCaseResult tcr");
List<?> tcr = context.executeQuery(query.toString());
```

The data is returned in a `List` object. This list must be stored in the context so that it can be accessed from the layout template:

```
context.add("results", tcr);
```

Note

For more information on building queries please consult the HQL documentation.

The `List` object is stored in the context with the name `results` and can be accessed from the layout template by this name. If additional data is needed, execute the appropriate queries and store the processed data in the context using a different name.

11.4.2. Creating a PDF Report Template

The code snippets in this section show how to build a PDF layout template. For the official documentation on the layout language used, please refer to SeamPDF manual. More information regarding the object model can be found in de.verit.klaros.core.model.

The following code snippet shows the basic structure of a PDF layout template. Inside the `<p:document>` element, the content of the report is contained and grouped in chapters and sections.

```
<p:document xmlns:ui="http://java.sun.com/jsf/facelets" xmlns:f="http://java.sun.com/jsf/core"
 xmlns:p="http://jboss.org/schema/seam/pdf" title="Klaros-Testmanagement Test Suite Report"
 marginMirroring="true" author="#{user.name}" creator="#{user.name}" pageSize="A4">
   ...
</p:document>
```

Here the de.verit.klaros.core.model.KlarosUser parameter which is always present in the context is used via `#{user.name}` to specify the author of the PDF document.

The next code snippet shows how to define the report headers and footers. Here the current date and the de.verit.klaros.core.model.KlarosUser parameter in the context is used.

```
<f:facet name="header">
  <p:font size="8">
    <p:header borderWidthBottom="0.1" borderColorBottom="black" borderWidthTop="0" alignment="center">
      <p:text value="Example report - generated on #{date} by #{user.name}"/>
    </p:header>

    <p:footer borderWidthTop="0.1" borderColorTop="black" borderWidthBottom="0" alignment="center">
      <p:text value="Page " />
      <p:pageNumber />
    </p:footer>
  </p:font>
</f:facet>
```

Next, the front page for the report is defined, showing a short summary of the report. To keep this example short, only a fragment with the user's name and email address is shown. For the complete script see Section E.3, “Example Layout Template”.

```
<p:font style="bold" size="16">
  <p:paragraph alignment="center" spacingAfter="5">
    <p:text value="#{user.name} (#{user.email})"/>
  </p:paragraph>
</p:font>
```

11.5. Creating a Chart

Reports can be enhanced by providing a chart showing a graphical representation of the data retrieved. The data for these charts is also prepared by the report script and stored in the context. The layout template passed the data to the charting component of SeamPDF. This section explains how to create a pie chart as shown in Figure 11.7, “Pie Chart Example”.

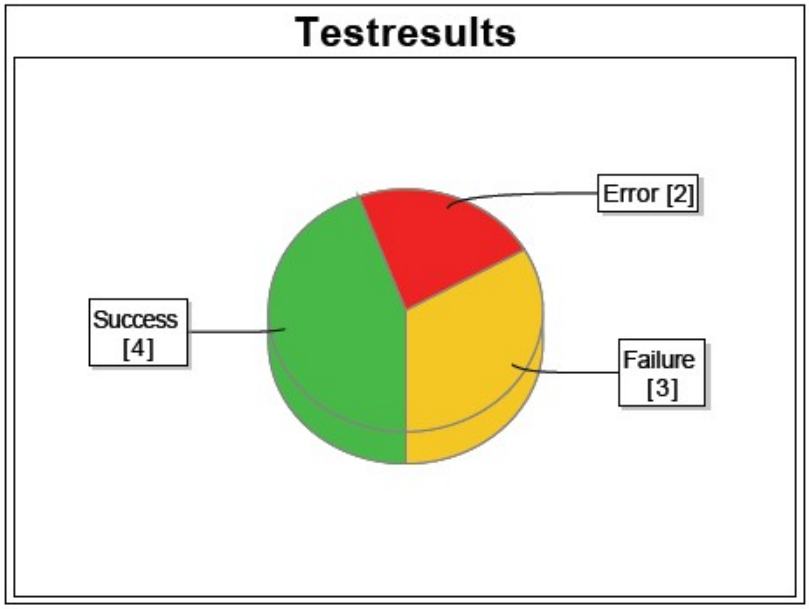


Figure 11.7. Pie Chart Example

11.5.1. Pie Chart Report Script

The report script is not only used to retrieve the data, but also be to prepare the data for the layout template. The following snippet shows a way to prepare the data for a pie chart. For each possi-

ble result type, an empty `List` object is created. Then the de.verit.klaros.core.model.KlarosTestCaseResult objects retrieved before are added to one of the lists depending on their result verdict. These lists can then later be accessed from the context by their corresponding key.

```
List<KlarosTestCaseResult> error = new ArrayList<KlarosTestCaseResult>();
List<KlarosTestCaseResult> failure = new ArrayList<KlarosTestCaseResult>();
List<KlarosTestCaseResult> success = new ArrayList<KlarosTestCaseResult>();
// Iterate over the results and retrieve the status
Iterator<KlarosTestCaseResult> iter = (Iterator<KlarosTestCaseResult>) tcr.iterator();
while (iter.hasNext()) {
  KlarosTestCaseResult result = iter.next();
    if (result.isError()) error.add(result);
    else if (result.isFailure()) failure.add(result);
    else if (result.isPassed()) success.add(result);
}
context.add("error", error);
context.add("failure", failure);
context.add("success", success);
```

11.5.2. Pie Chart Report Template

The following snippet shows the chart using the data that has been prepared by the report script in the previous section.

```
<p:paragraph horizontalAlignment="center">
  <p:piechart title="Test Results" direction="anticlockwise" circular="true"
    startAngle="30" labelGap="0.1" labelLinkPaint="#000000" plotBackgroundPaint="#ffffff"
    labelBackgroundPaint="#ffffff" is3D="true">
    <p:series key="results">
      <p:data key="Error [#{error.size}]" value="#{error.size}" sectionPaint="#FF0A0A" />
      <p:data key="Success [#{success.size}]" value="#{success.size}" sectionPaint="#33CC00"/>
      <p:data key="Failure [#{failure.size}]" value="#{failure.size}" sectionPaint="#FFCC00"/>
    </p:series>
  </p:piechart>
</p:paragraph>
```

The `piechart` element defines the presentation and contents of the chart. For a pie chart a `series` element is required. The data is retrieved from the lists formerly stored in the context by the script.

```
<p:data key="Error [#{error.size}]" value="#{error.size}" sectionPaint="#FF0A0A" />
```

This code retrieves the list containing the results from the context and calls its `size()` method to determine the number of test cases. The pie chart is then rendered, using the three data sections, as seen in Figure 11.7, “Pie Chart Example”.

Note

For detailed information on different chart types, please check the SeamPDF documentation.

11.5.3. Embedding Images

This section shows how to include an image into a report. To be accessible by the report template, the image file has to be stored in one of two possible locations:

- In a .jar file in the `.klaros/resources` folder

- On a web server, which can be accessed via a URL

When storing the image in a .jar file, it can be accessed using the following code snippet.

```
<p:image value="images/image.png"/>
```

The `value` attribute defines the image filename and the folder of the image location inside the jar file.

When providing an image via a URL it can be accessed using the following code snippet.

```
<p:html>
  <img src="http://www.verit.de/images/logo-klaros-160.png" />
</p:html>
```

The `src` attribute defines the URL of the image location.

11.5.4. Creating an Excel Report Template

This section shows how to build an Excel report template for Klaros-Testmanagement. For detailed information please refer to SeamExcel manual. More information on the Klaros-Testmanagement object model can be found in de.verit.klaros.core.model.

The following snippet shows a basic Excel report template. The <e:workbook> element, contains all other report details.

```
<e:workbook xmlns:e="http://jboss.org/schema/seam/excel"
  xmlns:f="http://java.sun.com/jsf/core" xmlns:ui="http://java.sun.com/jsf/facelets">
      <e:worksheet name="#{user.name}" value="#{testCases}" var="testCase">...
      </e:worksheet>
</e:workbook>
```

Note the usage of the de.verit.klaros.core.model.KlarosUser and the de.verit.klaros.core.model.KlarosTestCase parameter from the context: `#{user.name}` and `#{testCases}`.

A worksheet uses a list of objects to generate the rows and columns of the Excel sheet. This list must be passed to the `value` attribute of the `worksheet` element.

In the `var` attribute, the name of the current object can be defined. For example: To use the test case name of the current object inside the worksheet block, the statement `#{testCase.name}` is used.

It is also possible to use multiple `worksheet` elements in one template, though in this example only one worksheet is used, with the username as the worksheet name.

Chapter 12. Import/Export

Klaros-Testmanagement provides several interfaces to import data from other tools and export its data into several formats.

User authentication is mandatory for all import operations. The user role required varies with the type of data being imported. For importing test results, the *Tester* role is sufficient as for requirements and test cases the *Manager* role is required.

LDAP Authentication

If the setting *Login default* is enabled in the LDAP configuration, then authentication is done exclusively via the LDAP directory. Local authentication information is always ignored in this case.

12.1. Importing Test Cases from Excel

Klaros-Testmanagement allows importing of test cases from Excel sheets structured in a predefined format which will be further detailed in Section 12.1.1, “Import Format”.

In addition to the standard fields provided for test cases, users of the Klaros-Testmanagement Enterprise Edition are also able to import data for user-defined fields. For this it is necessary to prepare the project according to what is listed in Section 12.1.2, “Prerequisites”.

The import interface consists of a simple REST interface which can be accessed by command line tools as well as custom applications. This is further detailed in Section 12.1.3, “Execution”.

12.1.1. Import Format

The Excel sheet used to import test cases has to follow this strict format:

1. Each test case is located on a separate sheet.
2. Each sheet is divided into three sections: general properties, test case steps and custom properties.
3. The fields A:1 to A:22 are reserved for the standard attributes of a test case. The corresponding values are entered in the fields B:1 to B:22. It is not mandatory to fill all fields.
4. Column G is for the names of any number of user-defined properties. In column H the corresponding values are entered.
5. The `Step` field in cell A:25 **must** remain. Changing or deleting the A:25 field will cause the import of this table to be rejected. From here the definition of the test steps begins:
   - Column A for the test step number,
   - Column B for the action,
   - Column C for the precondition,
   - Column D for the postcondition and

- Column E for the expected result.

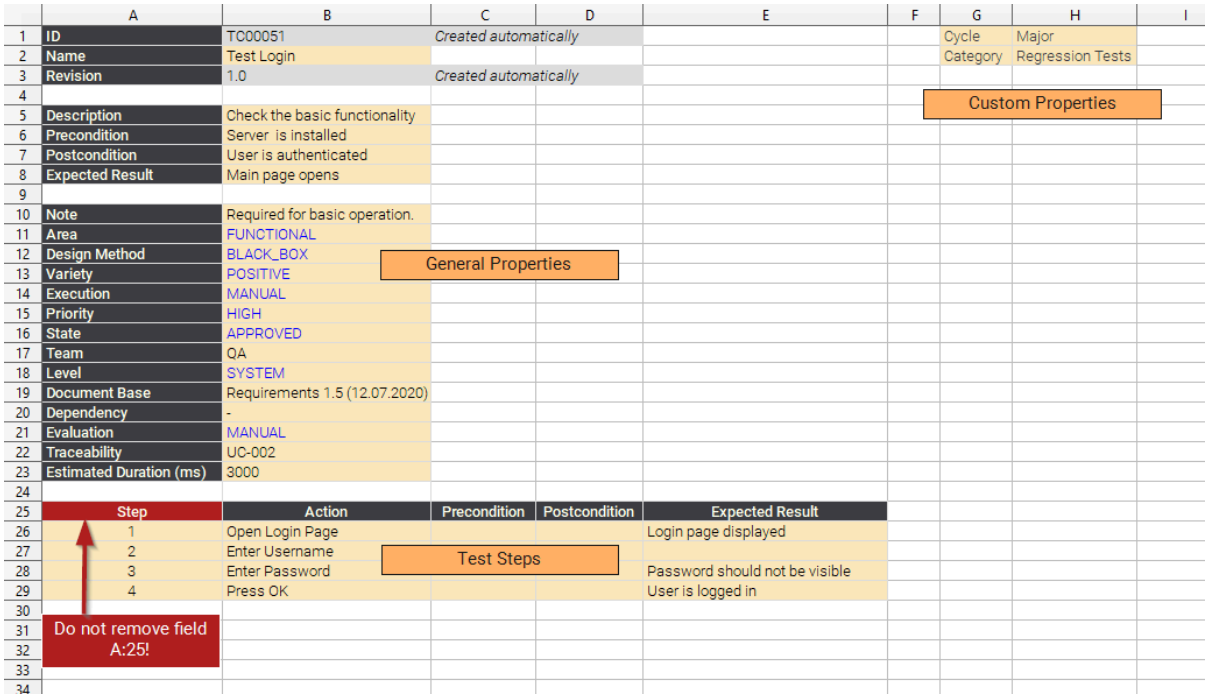


Figure 12.1. Test Case Excel Sheet Sample

12.1.1.1. General Properties

The table below lists the cell coordinates for the general test case properties. The test case ID does not have to be specified. The ID is generated during the import. None of these values are mandatory. Some fields can take only predefined values which are listed there.

Coordinate	Value
B2	Name
B3	Revision (only used in export)
B5	Description
B6	Precondition
B7	Postcondition
B8	Expected Result
B10	Note
B11	Area (`FUNCTIONAL`, `NON_FUNCTIONAL`, `STRUCTURAL`, `REGRESSION`, `RE_TEST`)
B12	Design Method (`BLACK_BOX`, `WHITE_BOX`)
B13	Variety (`POSITIVE`, `NEGATIVE`)
B14	Execution (`MANUAL`, `AUTO`)
B15	Priority (`LOW`, `MEDIUM`, `HIGH`)
B16	State (`NEW`, `APPROVED`, `LOCKED`, `INVISIBLE`)
B17	Team
B18	Level (`COMPONENT`, `INTEGRATION`, `SYSTEM`, `ACCEPTANCE`)

Coordinate	Value
B19	Document Base
B20	Dependency
B21	Evaluation
B22	Traceability
B23	Estimated Duration (ms)

Table 12.1. General Property Coordinates

12.1.1.2. Test Steps

Test cases may contain a variable number of test steps. These are listed one per row starting from cell A:26 downwards. The import parser will stop reading steps once it encounters an empty step number.

Coordinate	Value
A26 (ff)	Step Number
B26 (ff)	Action
C26 (ff)	Precondition
D26 (ff)	Postcondition
E26 (ff)	Expected Result

Table 12.2. Test Step Coordinates

12.1.1.3. Custom Properties

Only available in Klaros-Testmanagement Enterprise Edition

Test cases may contain custom properties. These may be listed as name/value pairs starting from cell G1 downwards.

For the import to succeed, the names of the listed properties must match with user defined properties defined for the project targeted by the import.

The value field for enumeration properties must exactly match the name of the enumeration property value definition to be parsed.

If an empty name cell is found, processing stops.

Coordinate	Value
G1 (ff)	Property name
H1 (ff)	Property value

Table 12.3. Custom Property Coordinates

12.1.2. Prerequisites

To import Excel based test cases the following prerequisites have to be met:

1. The file that is going to be imported has to be XLS or XLSX format.

2. The project that should contain the imported test cases has to be already created.
3. If user defined properties (only available in Klaros-Testmanagement Enterprise Edition) are to be imported they have to be created for the project prior to starting the import process.

12.1.3. Execution

To import Excel-based test cases a REST interface is available which allows you to upload your data from the command-line or other applications.

The following example shows how to import an Excel sheet containing test cases into the project named P00001 using the curl command line application. Curl should be available in every Linux distribution and as part of the Cygwin http://www.cygwin.com/ distribution or as a command line tool from http://curl.haxx.se/download.html for the Microsoft Windows operating system family.

Incompatible Windows Powershell curl Alias

Windows Powershell also defines an alias named **curl** which also can be used for this purpose but takes a completely different set of arguments. If you prefer to use the Linux compatible curl executable under Powershell, use **curl.exe** instead of **curl**.

```
curl -v -T TestCases.xls "<klaros-app-url>/seam/resource/rest/import/testcase/xls?config=P00001\
&username=user&password=secret"
```

Example 12.1. Excel Test Case Import via Command Line

ID Format

All objects which are referenced during import (like projects or test cases) contain five digits in their id.

For example, *P00001* is a valid project ID, while *P0001* and *P000001* are not.

12.2. Importing Test Cases from XML

Klaros-Testmanagement allows importing of test cases from XML Files. The import process will always create new instances of the supplied test cases for each invocation.

The format of the XML file is described in Appendix C, *Test Case Import File Specification*. The XML schema is available at the following URL: https://www.klaros-testmanagement.com/files/schema/klaros-testcases-1.0.xsd.

12.2.1. Prerequisites

To import XML based test cases the following prerequisites have to be met:

1. The file that is going to be imported has to in XML format.
2. The project that should contain the imported test cases has to be already created.
3. If user defined properties are to be imported, they have to be created in the project prior to starting the import process.

12.2.2. Execution

To import XML-based test cases a REST interface is available which allows you to upload your data from the command-line or other applications.

The following example shows how to import an XML file containing test cases into the project named P00001 using the curl command line application. Curl should be available in every decent Linux distribution and as part of the Cygwin http://www.cygwin.com/ distribution or as a command line tool from http://curl.haxx.se/download.html for the Microsoft Windows operating system family.

```
curl -v -T TestCases.xml "<klaros-app-url>/seam/resource/rest/import/testcase/xml?config=P00001\
&username=user&password=secret"
```

Example 12.2. XML Test Case Import via Command Line

12.3. Importing Requirements from Excel

In order to correctly import requirements from Excel spreadsheets, the data must exist in a specified format. This format is described in detail in the following section Section 12.1.1, "Import Format".

In addition to the standard fields provided for requirements, users of the Klaros-Testmanagement Enterprise Edition are also able to import data for user-defined fields. For this it is necessary to prepare the project accordingly. You can find more information about this at Section 12.1.2, "Prerequisites".

The import interface consists of a simple REST interface which can be accessed by command line tools as well as custom applications. This is further detailed in Section 12.1.3, "Execution".

12.3.1. Import Format

The Excel sheet used to import requirements has to follow this strict format:

1. Each requirement is located on a separate sheet.
2. Each sheet is divided into two sections: general properties and custom properties.
3. The fields A:1 to A:9 are reserved for the standard attributes of a requirement. The corresponding values are entered in the fields B:1 to B:9. It is not mandatory to fill all fields.
4. Column G is for the names of any number of user-defined properties. In column H the corresponding values are entered.

	A	B	C	D	E	F	G	H	I
1	ID	R00001	*Created automatically*				Department	Internal	
2	Name	Require authorization					Category	Security	
3	Revision	1.0	*Created automatically*				Phase	BETA	
4									
5	Summary	Unauthorized users must not be able to access services.					Custom Properties		
6	Description	During alpha development phase, it was possible to access services without providing a valid username/password combination. The server must now be secured against unauthorized access.							
7									
8	Priority	HIGH	General Properties						
9	State	APPROVED							
10									
11									

Figure 12.2. Requirement Excel Sheet Sample

12.3.1.1. General Properties

The table below lists the cell coordinates for the general requirement properties. None of these values are mandatory. Some fields can take only predefined values which are listed there.

Coordinate	Value
B1	ID (only used in export)
B2	Name
B3	Revision (only used in export)
B5	Summary
B6	Description
B8	Priority (`LOW`, `MEDIUM`, `HIGH`)
B9	State (`NEW`, `APPROVED`, `LOCKED`, `INVISIBLE`)

Table 12.4. General Property Coordinates

12.3.1.2. Custom Properties

Requirements may contain custom properties. These may be listed as name/value pairs starting from cell G1 downwards.

For the import to succeed, the names of the listed properties must match with user defined properties defined for the project targeted by the import.

The name in the name column must exactly match the name of a user defined property for a value to be parsed correctly. The parsing stops once an empty name cell has been found. The value field for enumeration properties must exactly match the name of the enumeration property value definition to be parsed.

Coordinate	Value
G1 (ff)	Property name
H1 (ff)	Property value

Table 12.5. Custom Property Coordinates

12.3.2. Prerequisites

To import Excel based requirements the following prerequisites have to be met:

1. The file that is going to be imported has to be in XLS or XLSX format.
2. The project that should contain the imported requirements has to be already created.
3. If user defined properties (only available in Klaros-Testmanagement Enterprise Edition) are to be imported they have to be created for the project prior to starting the import process.

12.3.3. Execution

To import Excel-based requirements a REST interface is available which allows you to upload your data from the command-line or other applications.

The following example shows how to import an Excel sheet containing requirements into the project named P00001 using the curl command line application. curl is available in almost any Linux distribution and for the Microsoft Windows operating system family as part of the Cygwin http://www.cygwin.com/ distribution or as a command line tool from http://curl.haxx.se/download.html.

```
curl -v -T Requirements.xls "<klaros-app-url>/seam/resource/rest/import/requirement/xls?config=P00001\
&username=user&password=secret"
```

Example 12.3. Excel Requirement Import via Command Line

12.4. Importing and Synchronizing Requirements from XML

Klaros-Testmanagement allows to both import and synchronize requirements from XML Files.

An import is limited to a one-time action that always creates new instances of the delivered requirements. Synchronization, on the other hand, allows updating existing requirements with the content of the XML file and creating new revisions of them as needed.

The format of the XML file is described in Appendix D, *Requirement Import File Specification*. The XML schema is available at the following URL: https://www.klaros-testmanagement.com/files/schema/klaros-requirements-1.0.xsd.

12.4.1. Prerequisites

To import XML based test cases the following prerequisites have to be met:

1. The file that is going to be imported has to be in *.XML format.
2. The project that should contain the imported requirement has to be already created.
3. If user defined properties are to be imported, they have to be created in the project prior to starting the import process.
4. If the imported requirements should be assigned to test cases via `externalTestCaseId`, Test cases with this external ID must already exist.

12.4.2. Importing

To import XML-based requirements a REST interface is available which allows you to upload your data from the command-line or other applications.

The following example shows how to import an XML file containing requirements into the project named P00001 using the curl command line application. curl is available in almost any Linux distribution and for the Microsoft Windows operating system family as part of the Cygwin http://www.cygwin.com/ distribution or as a command line tool from http://curl.haxx.se/download.html.

```
curl -v -T Requirements.xml "<klaros-app-url>/seam/resource/rest/import/requirement/xml?config=P00001\
&username=user&password=secret"
```

Example 12.4. XML Requirement Import via Command Line

12.4.3. Synchronization

To synchronize XML-based requirements a different REST interface is available (ending with `/sync/requirement/xml`). It understands the same import format as defined for importing requirements but requires additional elements to be present for correct operation.

In contrast to an import a synchronization action is intended to update the set of requirements stored in Klaros-Testmanagement from arbitrary external applications on a regular basis.

The synchronization supports creating and updating as well as revisioning of requirements. Changes to the requirements will be reflected in the Klaros-Testmanagement database on a per-field basis for each synchronization action.

Synchronization Overrides Local Changes!

Be aware that all data received via synchronization will overwrite any changes made to the requirements stored in Klaros-Testmanagement.

`externalId`

Supplying the `externalId` element (See Section D.11, “<externalId>”) will identify a requirement for subsequent synchronisation attempts. It is required field when using this interface. If you change this value, a new requirement object will be created instead of replacing the content of the previously created requirement.

`revision`

Supplying the `revision` element (See Section D.15, “<revision>”) will identify the version of a requirement for subsequent synchronisation attempts.

- If the value is empty, the currently selected revision of the requirement is updated.
- If a new revision is entered, a new version of the requirement is created and updated.
- If an older revision is entered, the entered revision will be updated.

The following example shows how to synchronize an XML file containing requirements into the project named P00001 using the curl command line application. curl is available in almost any Linux distribution and for the Microsoft Windows operating system family as part of the Cygwin http://www.cygwin.com/ distribution or as a command line tool from http://curl.haxx.se/download.html.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<r:container xmlns:r="http://klaros-testmanagement.com/export-requirements-1.0">
 <r:requirements>
  <r:requirement>
   <r:attributes/>
   <r:externalId>RTM-00001</r:externalId>
   <r:revision>1.0</r:revision>
   <r:priority>MEDIUM</r:priority>
```

```
   <r:shortname>Remote-controlled door panels / unlocking</r:shortname>
   <r:summary>Doors must me remote controllable.</r:summary>
  </r:requirement>
 </r:requirements>
</r:container>
```

Example 12.5. XML Requirement Synchronization via Command Line

```
curl -v -T Requirements.xml "<klaros-app-url>/seam/resource/rest/sync/requirement/xml?config=P00001\
&username=user&password=secret"
```

Example 12.6. XML Requirement Import via Command Line

12.5. Importing Test Results

Test automation tools typically generate test results in file form. These files can be imported via a REST interface and automatically converted into test results.

12.5.1. The JUnit XML++ format

JUnit XML++ is the generic format used internally by Klaros-Testmanagement for importing test case results from external sources. This format is upward compatible with the JUnit XML format.

By extending the JUnit XML format, the JUnit XML++ format enables you to specify individual test steps of a test, as well as to embed binary attachments in an XML document. In addition, with the result type `inconclusive` it allows to indicate ambiguous results.

12.5.1.1. The JUnit XML format

Unfortunately, a clear formal JUnit XML format specification does not exist, as JUnit itself does not create XML reports. XML reporting usually results from the processing of the JUnit invocation within a build tool such as the Ant JUnit task, the Maven Surefire plugin, or Gradle.

The JUnit XML format has established itself as a generic (quasi-)standard format for test case results across tool and programming language boundaries and is supported by a large number of automation tools. Despite the ambiguity of the format, there have been several attempts in the past to document the format in the form of an XML schema. Detailed commented schema definitions can be found, for example, here: https://llg.cubic.org/docs/junit/ and here: https://github.com/windyroad/Junit-Schema/blob/master/JUnit.xsd.

12.5.1.2. The JUnit XML++ format

The JUnit XML format unfortunately does not offer all the possibilities that are supported in Klaros-Testmanagement, especially it lacks:

Das JUnit XML Format bietet leider nicht alle Möglichkeiten, die in Klaros-Testmanagement unterstützt werden, insbesondere fehlt hier:

- Describing and evaluating tests at the test step level
- Adding binary attachments (images, documents, etc.) to the test results
- Support additional test ratings (unclear)

We have added this to the JUnit XML format without changing the existing syntax. In the following we call this new format JUnit XML++.

12.5.1.2.1. Test step results

Within a `testcase` element there can now occur any number of `teststep` elements. These must contain a `time` attribute which specifies the execution time of the test step in seconds.

As with test cases, `error`, `failure` or `skipped` elements within a test step control their rating.

Also likewise to the test cases, `system-out` as well as `system-err` elements are supported. If these contain content, they are automatically added as binary attachments to the test case result generated during import.

Optionally, the elements `action`, `precondition`, `postcondition` and `expectedResult` can be mapped to the test step result.

If these are available, they are transferred to the corresponding fields of a test step result in Kiros-Testmanagement. This information is not mandatory, but it is recommended at least for the `action` field in order to better identify individual test steps.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<testcase name="Test 1" time="0.008">
    <teststep time="0.006">
        <action>Textual description of the action taken in this step</action>
        <precondition>Precondition information</precondition>
        <postcondition>Postcondition information</postcondition>
        <expectedResult>Expected result of this test step</expectedResult>
    </teststep>
    <teststep time="0.002">
        <failure>
            Detailed failure message
        </failure>
        <system-out>Log message</system.out>
    </teststep>
</testcase>
```

Example 12.7. Sample Test Test-step-result.xml

12.5.1.2.2. Attachments

For the documentation of a test (step) result, it may be useful to supplement it with a file (screenshot, document, etc.) in addition to the textual description. To enable this, additional content can be defined in the system-out and system-err elements.

After successful extraction and saving, this content is not transferred to any additionally created system-out/system-err attachments. The content is defined according to the Jenkins plugin JUnit Attachments and may appear in three different forms:

Base64 Encoded Attachment

Syntax: `[[ATTACHMENT_DATA|«attached-file.xyz»|«encoded data»]]`

In “encoded data” a base64-encoded binary content is expected. After successful decoding, this is stored under the file name `attached-file.xyz` as an attachment to the test case result or test step result.

So this method saves the entire attachment as part of the result file. This requires more storage space, but allows access to the content at any time.

URL Attachment

Syntax: `[[ATTACHMENT_URL|/«url/to/attached-file.xyz»]]`

An attempt will be made to download the contents of the URL `url/to/attached-file.xyz` and to save it as an attachment under the filename `attached-file.xyz`.

This method expects the attachment to be available on the network. It is mandatory that the Klaros instance has access to this URL at import time for the extraction to succeed.

File Attachment

Syntax: `[[ATTACHMENT|/«path/to/attached-file.xyz»]]`

An attempt is made to download the contents of the file `path/to/attached-file.xyz` and save it as an attachment under the file name `attached-file.xyz`.

With this method it is expected that the attachment is provided in the local file system. It is mandatory that the Klaros instance has access to this file at the time of import for the extraction to succeed.

12.5.1.2.3. Test results

For test case results (`testcase`) as well as for test step results (`teststep`), it is possible to set `inconclusive` (unclear) as result type besides `error`, `failure` or `skipped`, since this is supported in Klaros-Testmanagement but not in JUnit.

12.5.2. Prerequisites

To import test results the following prerequisites have to be met:

1. The project, iteration, test environment and system under test that should contain the imported test case or test suite results have to be created.

12.5.3. Execution

The default URL of the import interface is located at `http://localhost:18080/klaros-web/seam/resource/rest/importer`. The content is transferred via an HTTP PUT request using the above URL and various URL query parameters.

The `<klaros-app-url>` Term

The `http://localhost:18080/klaros-web` URL shown above is the default klaros application url when accessing Klaros-Testmanagement from the host it is installed on and may vary with your setup. Throughout this chapter the term `<klaros-app-url>` will be used instead to reflect this.

The following parameters are supported:

config	The ID of the project to import the results into (e.g. P0001).
iteration	The ID of the iteration to relate the results to (e.g. ITR00001). This parameter is optional.
env	The ID of the test environment in which the tests have been run (e.g. ENV00001). Please make sure that this test environment already exists in the project before starting the import.
sut	The ID of the system under test in which the tests have been run (e.g. SUT00001). Please make sure that this system under test already exists in the project before starting the import.
type	The type of the import format. The following types are supported:

aunit	AUnit
boost	Boost Test
Check	Check
cpptest	CppTest
cppunit	CppUnit
ctest	ctest (CMake)
cunit	CUnit
fitnesse	Fitnesse
fpunit	Free Pascal Unit
gtester	GLib/gtester
googletest	GoogleTest
jbehave	JBehave
jmeter	JMeter
jsunit	JsUnit
jubula	Jubula / GUIdancer
junit	JUnit / Gauge / Robot Framework / Selenium / Squish / TestNG
mbunit	MBUnit
mstest	MSTest
nunit	NUnit
phpunit	PHPUnit

qftest	QF-Test
qtestlib	QTestLib
tessy	TESSY
testcomplete	TestComplete
tusar	TUSAR
uft	Unified Functional Testing (UFT) / QuickTest Professional (QTP)
cpptestunit	UnitTest++
valgrind	Valgrind
xUnit.net	xunitdotnet

job The optional ID of a job this test run should be assigned to (e.g. JOB00001). If given, the job must already exist in the project before starting the import.

time The time of the test execution. Please make sure the fixed format for the time is `dd.MM.yyyy_HH:mm`. This parameter can be used to override the test run date which is otherwise bound to the time of import.

createTestSuiteResults If set to `true` test suite results are automatically created for the corresponding test suite information contained in the result file. This information may vary with the import format. Additionally, a corresponding test suite is created for the test suite result if it did not yet exist.

autoCreateTestCases If set to `false` test cases are no longer automatically created for test case results for which no associated test case can be found. The default setting for this parameter is `true`.

username The username for the import.

password The password for the import.

A complete example for a QF-Test import URL would look like this:

```
http://localhost:18080/klaros-web/seam/resource/rest/importer?\
config=P00001&env=ENV00001&sut=SUT00001&type=qftest&time=01.03.2011_12:00&username=me&password=secret
```

Example 12.8. QF-Test import URL sample

The result file is contained in the HTTP request body.

The **curl** command line tool can be used on Linux or Windows/Cygwin to trigger an import in a single command line.

```
curl -v -H "Content-Type: text/xml" -T <test result file> \
```

```
"<klaros-app-url>/seam/resource/rest/importer?config=P00001&env=ENV00001&sut=SUT00001&type=junit\
&time=23.05.2011_14:55"
```

Example 12.9. Curl Command Line Example

Incompatible Windows Powershell curl alias

Windows Powershell also defines an alias named **curl** which also can be used for this purpose but takes a completely different set of arguments. If you prefer to use the Linux compatible curl executable under Powershell, use **curl.exe** instead of **curl**.

```
curl -Method put -InFile <test result file> -Uri \
"<klaros-app-url>/seam/resource/rest/importer?config=P00001&env=ENV00001&sut=SUT00001&type=junit\
&time=23.05.2011_14:55"
```

Example 12.10. Powershell Curl Alias Command Line Example

12.5.4. Jenkins Plugin

This plugin integrates a continuous integration server with Klaros-Testmanagement by publishing the test results of a Jenkins build to the Klaros-Testmanagement application. The test results will be stored in the Klaros-Testmanagement database for further evaluation and reporting purposes. You can find the installation and configuration guide for the plugin in the Jenkins GitHub page.

12.6. Exporting Test Results

Klaros-Testmanagement provides a JUnit XML export interface that enables the export of test results.

12.6.1. Execution

The default URL of the export interface is located at `http://localhost:18080/klaros-web/seam/resource/rest/export/result/junit`. The content is transferred via an HTTP GET request using the above URL and various URL query parameters.

The `<klaros-app-url>` Term

The `http://localhost:18080/klaros-web` URL shown above is the default klaros application url when accessing Klaros-Testmanagement from the host it is installed on and may vary with your setup. Throughout this chapter the term `<klaros-app-url>` will be used instead to reflect this.

The following parameters are supported:

config	The ID of the project to export the results from (e.g. P0001).
env	The ID of the test environment in which the tests have been run (e.g. ENV00001).
sut	The ID of the system under test in which the tests have been run (e.g. SUT00001).
job	The ID of the job to export the results of (e.g. JOB00001).

testRun The ID of the test run to export the results of (e.g. TRU0000001).

username The username for the export.

password The password for the export.

Both individual test results of a job or a test run and all test results of a system under test or a combination of system under test and test environment can be exported. This can be controlled by specifying the parameters.

The **curl** command line tool can be used on Linux or Windows/Cygwin to trigger an export in a single command line.

```
curl \
"<klaros-app-url>/seam/resource/rest/export/result/junit?config=P00001&job=JOB00001&user=me&password=secret"
```

In this example, the results of the job JOB00001 from the project P00001 are exported.

Example 12.11. Curl Command Line Example

Incompatible Windows Powershell curl alias

Windows Powershell also defines an alias named **curl** which also can be used for this purpose but takes a completely different set of arguments. If you prefer to use the Linux compatible curl executable under Powershell, use **curl.exe** instead of **curl**.

```
curl -Method get -Uri \
"<klaros-app-url>/seam/resource/rest/export/result/junit?config=P00001&testRun=TRU0000001&username=me&password=secret"
```

In this example, the results of the test run TRU0000001 from the project P00001 are exported.

Example 12.12. Powershell Curl Alias Command Line Example

12.7. Exporting Table Content

Only available in Klaros-Testmanagement Enterprise Edition

With Klaros-Testmanagement it is possible to export the content of all tables to an Excel file. The current filter and sort settings will be considered.

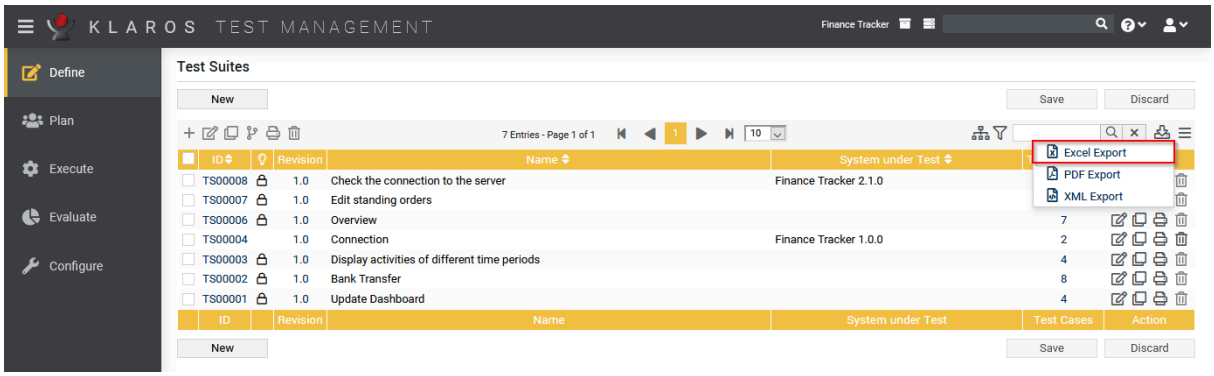


Figure 12.3. Export Table Content to Excel

12.8. Backup/Recovery

To move data between different database installations, or to selectively import data, Klaros-Testmanagement provides the functionality to import and export database content via XML files. Section 10.6, “Backup” explains the import and export functionality in detail.

12.8.1. Backup via REST

Klaros-Testmanagement Enterprise Edition contains a REST-based interface that grant access to back up files of single projects. The following URI for example returns the backup xml file of the project P00004: http://{host}:{port}/klaros-web/seam/resource/rest/io/backup/project/P00004.

Note

For more information on how to access REST-based interfaces, see Chapter 13, *The Remote API*.

Chapter 13. The Remote API

Only available in Klaros-Testmanagement Enterprise Edition

13.1. Overview

Klaros-Testmanagement Enterprise Edition comes with a REST-based interface which can be used to retrieve data like test cases, results or information about users. The data can be retrieved either in XML or JSON format. The API is accessible from the URL http://{host}:{port}/klaros-web/api/v1/resources/.

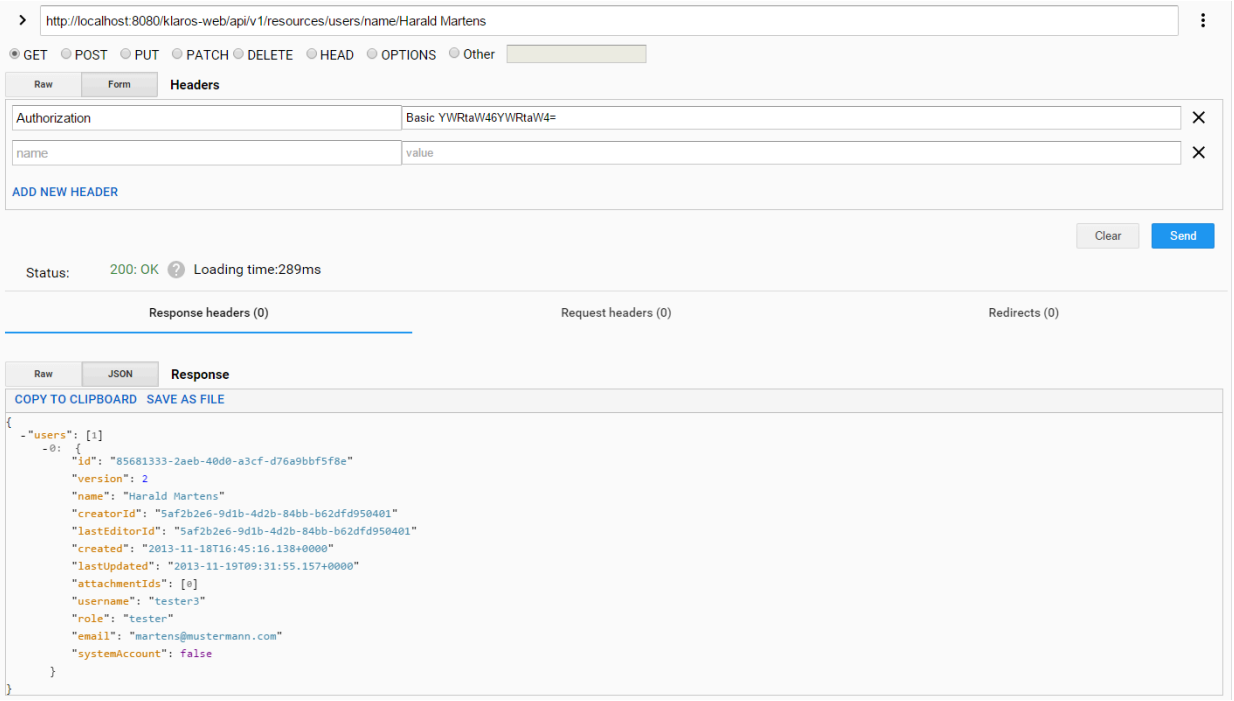


Figure 13.1. JSON Output after Searching for a User by its Name.

13.2. REST Clients

There are numerous ways to connect to the remote API. The following lists give an overview of some freely available clients.

Browser Plugins

- Advanced REST client plugin for Google Chrome

  Google Chrome Web Store

- RESTED for Firefox

  Firefox Add-Ons

- RestMan Add-On for Opera

  Opera Add-Ons

Standalone Clients

- Wget

  https://www.gnu.org/software/wget/

- cURL

  http://curl.haxx.se/

13.3. Documentation

Klaros-Testmanagement comes with an interactive documentation for the Remote API which can be accessed from a browser at http:/{host}:{port}/klaros-web/api-doc/resources/.

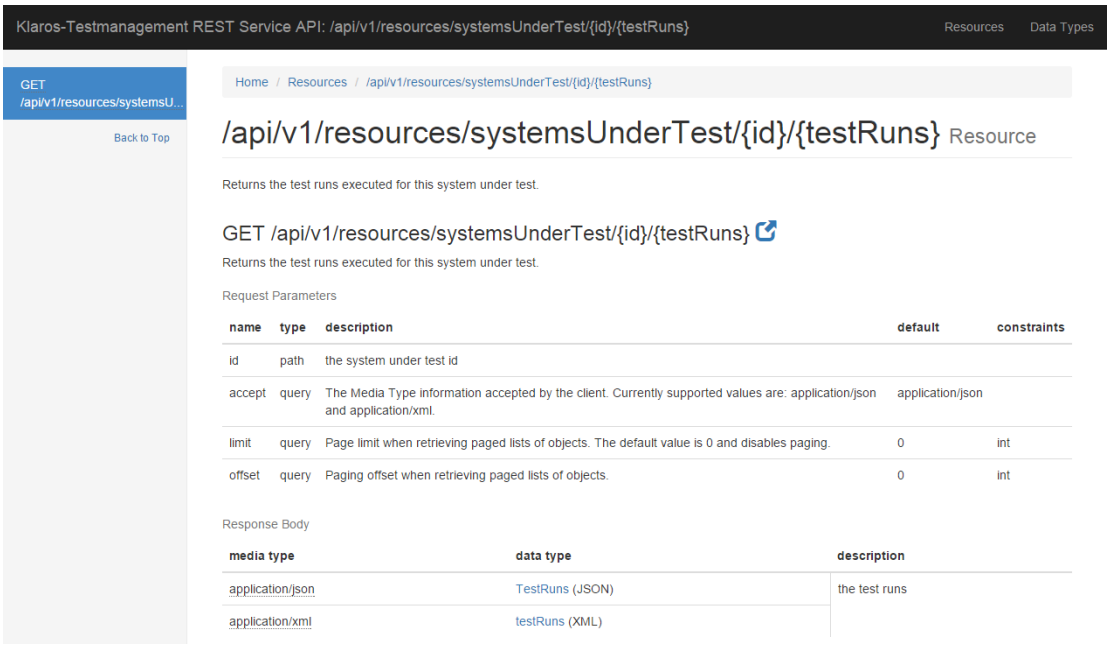


Figure 13.2. Remote API Documentation

Appendix A. Access Permissions

A.1. Role Permission Overview – Community Edition

Actions/Roles	Administrator	Manager	Tester	Guest
Execute test cases	✓	✓	✓	✗
Generate reports	✓	✓	✓	✓
View Test Results	✓	✓	✓	✓
Backup and Restore [a]	✓	✓	✗	✗
Create, edit and delete test objects	✓	✓	✗	✗
Create, edit and delete users with the role *Tester* [b]	✓	✓	✗	✗
Create, edit and delete users with the role *Administrator* or *Manager*	✓	✗	✗	✗
Edit system preferences [b]	✓	✗	✗	✗

[a] Global managers are able to restore any project. However, they may only export projects which have no project-specific roles defined or in which they have the role *Manager*.

[b] Not changed by project role.

Table A.1. Role Permission Overview – Community Edition

A.2. Role Permission Overview – Enterprise Edition

Actions/Roles	Administrator	Manager	Tester	Guest
Configure dashboard	✓	✓	✓	✓
Execute jobs	✓	✓	✓	✗
Export Excel tables	✓	✓	✓	✓
Generate custom reports	✓	✓	✓	✓
Generic full text search	✓	✓	✓	✓
Track test run execution	✓	✓	✓	✗
Create and assign jobs	✓	✓	✗	✗
Create Custom Reports [a]	✓	✓	✗	✗
Define custom fields	✓	✓	✗	✗
Configure LDAP authentication [a]	✓	✗	✗	✗

[a] Not changed by project role

Table A.2. Role Permission Overview – Enterprise Edition

Appendix B. The Klaros Object Model API Reference

B.1. Klaros Object Model API Reference

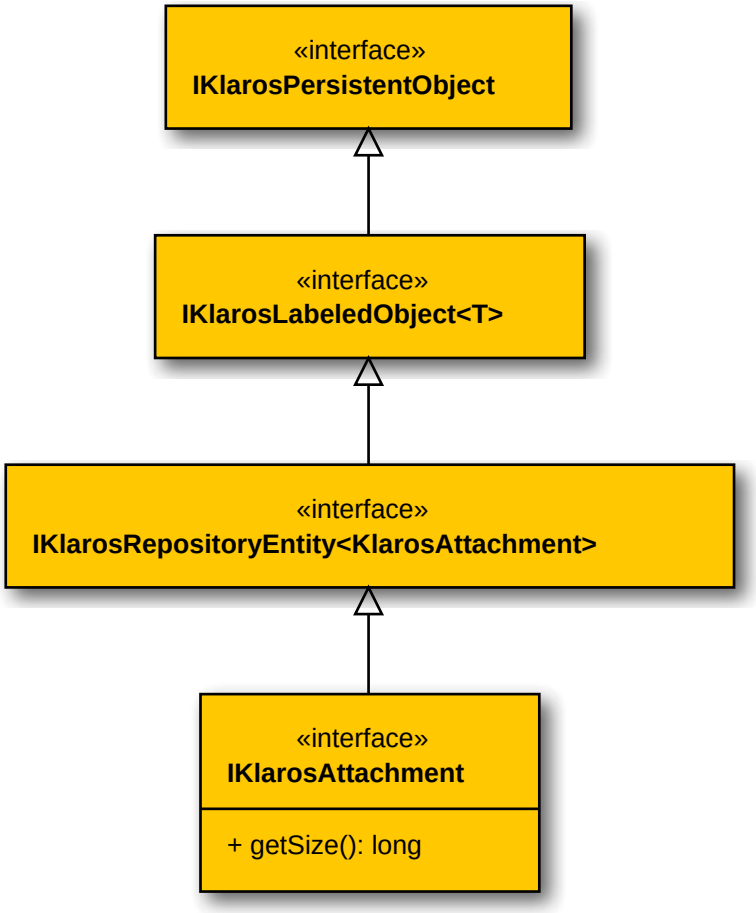
B.1.1. de.verit.klaros.core.model

B.1.1.1. IKlarosAttachment

This interface provides access to a binary attachment.

B.1.1.1.1. Synopsis

```
public interface IKlarosAttachment
 extends IKlarosRepositoryEntity<KlarosAttachment> {

  // Public Methods
  public abstract long getSize();
}
```



B.1.1.1.2. getSize()

```
public abstract long getSize();
```

Gets the size in bytes of this attachment.

Parameters

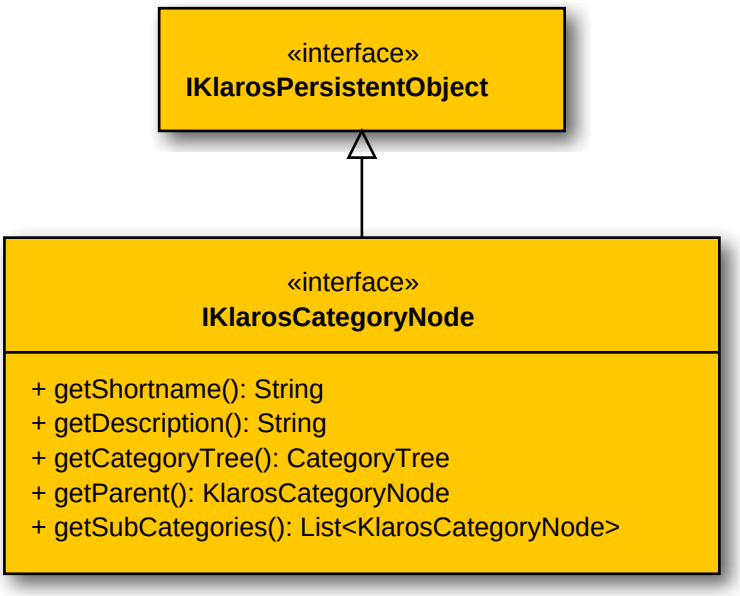
`return` the size

B.1.1.2. IKlarosCategoryNode

A Klaros Category Node.

B.1.1.2.1. Synopsis

```
public interface IKlarosCategoryNode extends IKlarosPersistentObject {

  // Public Methods
  public abstract CategoryTree getCategoryTree();
  public abstract String getDescription();
  public abstract KlarosCategoryNode getParent();
  public abstract String getShortname();
  public abstract List<KlarosCategoryNode> getSubCategories();
}
```



B.1.1.2.2. getCategoryTree()

```
public abstract CategoryTree getCategoryTree();
```

Gets the category tree.

Parameters

`return` the category tree

B.1.1.2.3. getDescription()

```
public abstract String getDescription();
```

Gets the description.

Parameters

`return` the description

B.1.1.2.4. getParent()

```
public abstract KlarosCategoryNode getParent();
```

Gets the parent category node.

Parameters

`return` the parent

B.1.1.2.5. getShortname()

```
public abstract String getShortname();
```

Gets the shortname.

Parameters

`return` the shortname

B.1.1.2.6. getSubCategories()

```
public abstract List<KlarosCategoryNode> getSubCategories();
```

Gets the sub categories.

Parameters

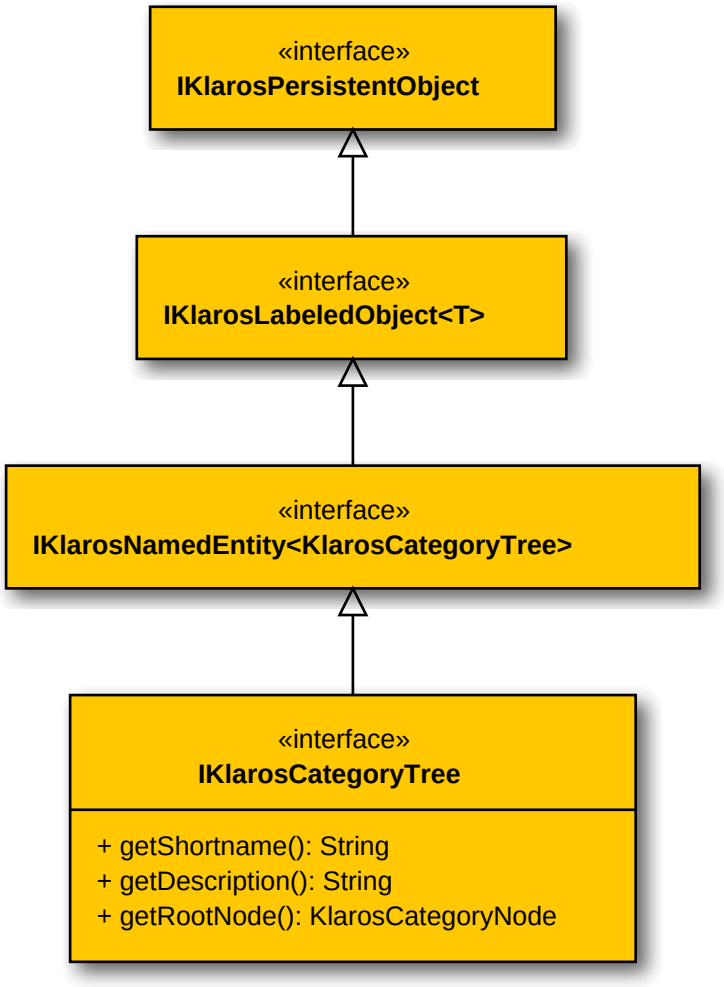
`return` the sub categories

B.1.1.3. IKlarosCategoryTree

A category tree.

B.1.1.3.1. Synopsis

```
public interface IKlarosCategoryTree extends IKlarosNamedEntity<KlarosCategoryTree> {

  // Public Methods
  public abstract String getDescription();
  public abstract KlarosCategoryNode getRootNode();
  public abstract String getShortname();
}
```



B.1.1.3.2. getDescription()

```
public abstract String getDescription();
```

Gets the description.

Parameters

`return`	the description

B.1.1.3.3. getRootNode()

```
public abstract KlarosCategoryNode getRootNode();
```

Gets the root category node.

Parameters

`return`	the root

B.1.1.3.4. getShortname()

```
public abstract String getShortname();
```

Gets the shortname.

Parameters

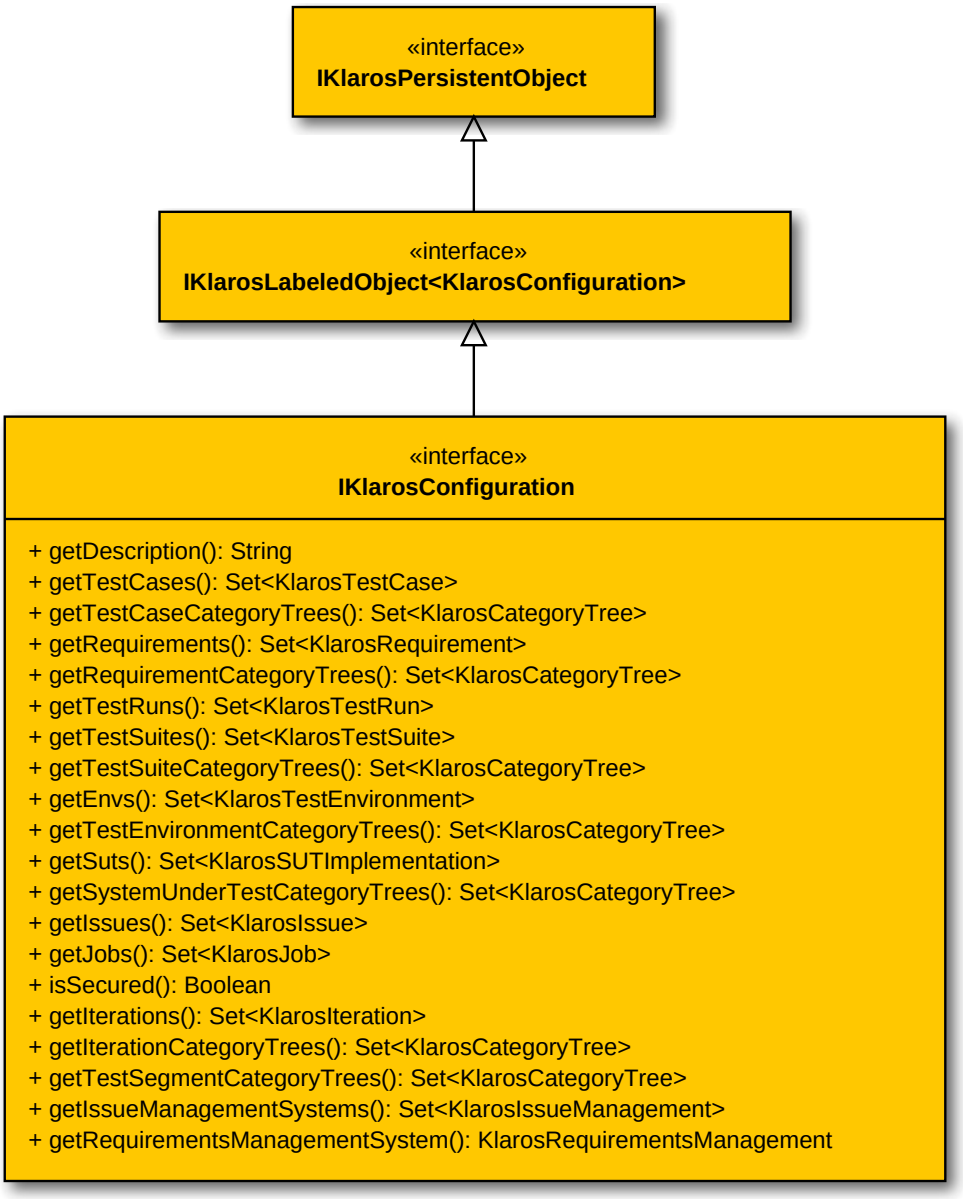
`return` the shortname

B.1.1.4. IKlarosConfiguration

This interface provides access to data of a test project.

B.1.1.4.1. Synopsis

```
public interface IKlarosConfiguration
 extends IKlarosLabeledObject<KlarosConfiguration> {

  // Public Methods
  public abstract String getDescription();
  public abstract Set<KlarosTestEnvironment> getEnvs();
  public abstract Set<KlarosIssueManagement> getIssueManagementSystems();
  public abstract Set<KlarosIssue> getIssues();
  public abstract Set<KlarosCategoryTree> getIterationCategoryTrees();
  public abstract Set<KlarosIteration> getIterations();
  public abstract Set<KlarosJob> getJobs();
  public abstract Set<KlarosCategoryTree> getRequirementCategoryTrees();
  public abstract Set<KlarosRequirement> getRequirements();
  public abstract KlarosRequirementsManagement getRequirementsManagementSystem();
  public abstract Set<KlarosSUTImplementation> getSuts();
  public abstract Set<KlarosCategoryTree> getSystemUnderTestCategoryTrees();
  public abstract Set<KlarosCategoryTree> getTestCaseCategoryTrees();
  public abstract Set<KlarosTestCase> getTestCases();
  public abstract Set<KlarosCategoryTree> getTestEnvironmentCategoryTrees();
  public abstract Set<KlarosTestRun> getTestRuns();
  public abstract Set<KlarosCategoryTree> getTestSegmentCategoryTrees();
  public abstract Set<KlarosCategoryTree> getTestSuiteCategoryTrees();
  public abstract Set<KlarosTestSuite> getTestSuites();
  public abstract Boolean isSecured();
}
```



B.1.1.4.2. getDescription()

```
public abstract String getDescription();
```

Returns the project description.

Parameters

`return`	The description of the project.

B.1.1.4.3. getEnvs()

```
public abstract Set<KlarosTestEnvironment> getEnvs();
```

Returns the project test environments.

Parameters

`return`	Set containing the test environments of the project.

B.1.1.4.4. getIssueManagementSystems()

```
public abstract Set<KlarosIssueManagement> getIssueManagementSystems();
```

Returns the related issue management systems.

Parameters

`return`	Set containing the related issue management systems of the project.

B.1.1.4.5. getIssues()

```
public abstract Set<KlarosIssue> getIssues();
```

Returns the issues.

Parameters

`return`	KlarosSet containing the issue objects of the project.

B.1.1.4.6. getIterationCategoryTrees()

```
public abstract Set<KlarosCategoryTree> getIterationCategoryTrees();
```

Returns the project iteration category trees.

Parameters

`return`	Set containing the iteration category trees of the project.

B.1.1.4.7. getIterations()

```
public abstract Set<KlarosIteration> getIterations();
```

Returns the project iteration.

Parameters

`return`	KlarosSet containing the iteration objects of the project.

B.1.1.4.8. getJobs()

```
public abstract Set<KlarosJob> getJobs();
```

Returns the project jobs.

Parameters

`return`	KlarosSet containing the job objects of the project.

B.1.1.4.9. getRequirementCategoryTrees()

```
public abstract Set<KlarosCategoryTree> getRequirementCategoryTrees();
```

Returns the project requirement category trees.

Parameters

`return`	Set containing the requirement category trees of the project.

B.1.1.4.10. getRequirements()

```
public abstract Set<KlarosRequirement> getRequirements();
```

Returns the project requirements.

Parameters

`return`	Set containing the requirement objects of the project.

B.1.1.4.11. getRequirementsManagementSystem()

```
public abstract KlarosRequirementsManagement getRequirementsManagementSystem();
```

Returns the related requirement management system.

Parameters

`return`	Set containing the related requirement management system of the project.

B.1.1.4.12. getSuts()

```
public abstract Set<KlarosSUTImplementation> getSuts();
```

Returns the project SUTs (systems under test).

Parameters

`return`	KlarosSet containing the SUT objects of the project.

B.1.1.4.13. getSystemUnderTestCategoryTrees()

```
public abstract Set<KlarosCategoryTree> getSystemUnderTestCategoryTrees();
```

Returns the project system under test category trees.

Parameters

`return`	Set containing the system under test category trees of the project.

B.1.1.4.14. getTestCaseCategoryTrees()

```
public abstract Set<KlarosCategoryTree> getTestCaseCategoryTrees();
```

Returns the project test case category trees.

Parameters

`return`	Set containing the test case category trees of the project.

B.1.1.4.15. getTestCases()

```
public abstract Set<KlarosTestCase> getTestCases();
```

Returns the project test cases.

Parameters

`return`	Set containing the test case objects of the project.

B.1.1.4.16. getTestEnvironmentCategoryTrees()

```
public abstract Set<KlarosCategoryTree> getTestEnvironmentCategoryTrees();
```

Returns the project test environment category trees.

Parameters

`return`	Set containing the test environment category trees of the project.

B.1.1.4.17. getTestRuns()

```
public abstract Set<KlarosTestRun> getTestRuns();
```

Returns the project test runs.

Parameters

`return`	Set containing the test run objects of the project.

B.1.1.4.18. getTestSegmentCategoryTrees()

```
public abstract Set<KlarosCategoryTree> getTestSegmentCategoryTrees();
```

Returns the project test segment category trees.

Parameters

`return`	Set containing the test segment category trees of the project.

B.1.1.4.19. getTestSuiteCategoryTrees()

```
public abstract Set<KlarosCategoryTree> getTestSuiteCategoryTrees();
```

Returns the project test suite category trees.

Parameters

`return`	Set containing the test suite category trees of the project.

B.1.1.4.20. getTestSuites()

```
public abstract Set<KlarosTestSuite> getTestSuites();
```

Returns the project test suites.

Parameters

`return`	Set containing the test suite objects of the project.

B.1.1.4.21. isSecured()

```
public abstract Boolean isSecured();
```

Returns whether this project is secured or not.

Parameters

`return`	true if this project is secured, false if not.

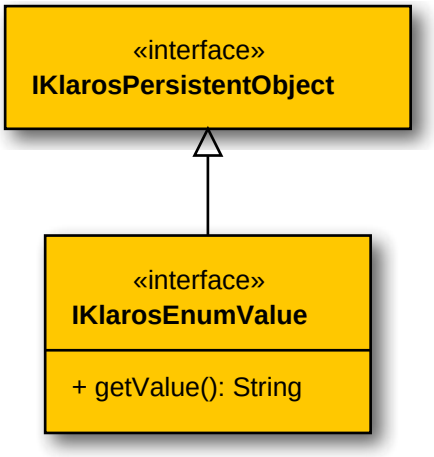
B.1.1.5. IKlarosEnumValue

A Custom property enumeration value.

B.1.1.5.1. Synopsis

```
public interface IKlarosEnumValue extends IKlarosPersistentObject {

  // Public Methods
```

```
  public abstract String getValue();
}
```



B.1.1.5.2. getValue()

```
  public abstract String getValue();
```

Gets the enumeration value.

Parameters

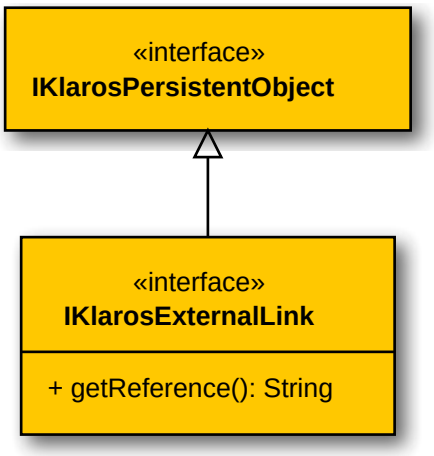
`return`	the value

B.1.1.6. IKlarosExternalLink

This interface provides access to externally stored information about an object.

B.1.1.6.1. Synopsis

```
public interface IKlarosExternalLink extends IKlarosPersistentObject {

  // Public Methods
  public abstract String getReference();
}
```



B.1.1.6.2. getReference()

```
public abstract String getReference();
```

Get the reference to the externally stored information.

Parameters

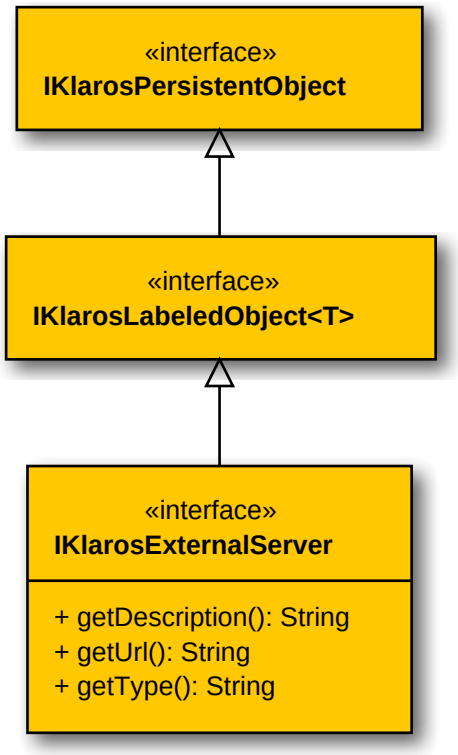
return	The reference to the information.

B.1.1.7. IKlarosExternalServer

This interface provides access to external systems.

B.1.1.7.1. Synopsis

```
public interface IKlarosExternalServer<T> extends IKlarosLabeledObject<T> {

  // Public Methods
  public abstract String getDescription();
  public abstract String getType();
  public abstract String getUrl();
}
```



B.1.1.7.2. getDescription()

```
public abstract String getDescription();
```

Get the description of the external system.

Parameters

`return`	The description of the external system.

B.1.1.7.3. getType()

```
public abstract String getType();
```

Get the type of this system.

Parameters

`return`	The type

B.1.1.7.4. getUrl()

```
public abstract String getUrl();
```

Get the base url of this system.

Parameters

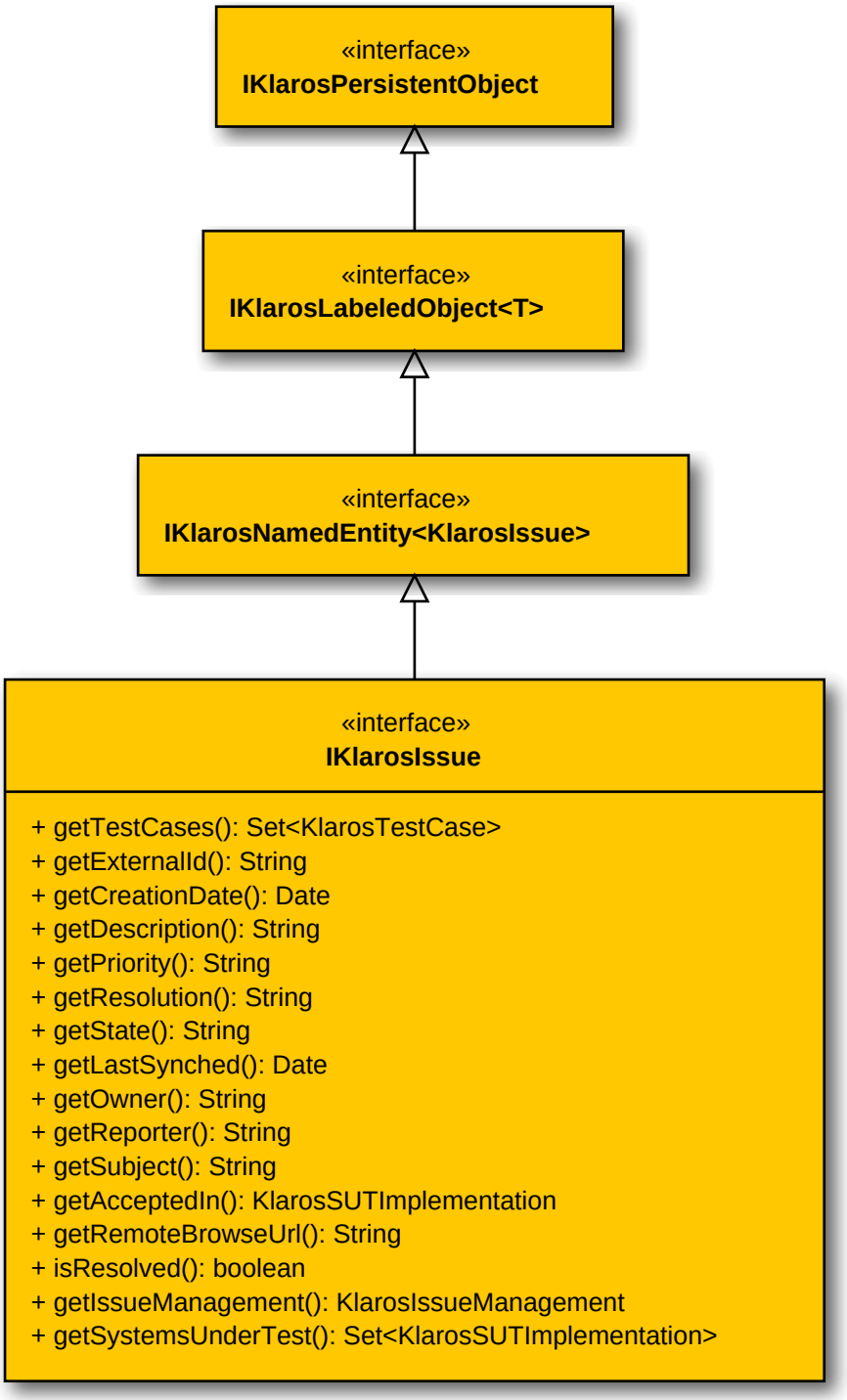
`return`	The url

B.1.1.8. IKlarosIssue

This interface provides access to a software issue.

B.1.1.8.1. Synopsis

```
public interface IKlarosIssue extends IKlarosNamedEntity<KlarosIssue> {

  // Public Methods
  @Deprecated(since="5.3.3")
  public abstract KlarosSUTImplementation getAcceptedIn();
  public abstract Date getCreationDate();
  public abstract String getDescription();
  public abstract String getExternalId();
  public abstract KlarosIssueManagement getIssueManagement();
  public abstract Date getLastSynched();
  public abstract String getOwner();
  public abstract String getPriority();
  public abstract String getRemoteBrowseUrl();
  public abstract String getReporter();
  public abstract String getResolution();
  public abstract String getState();
  public abstract String getSubject();
  public abstract Set<KlarosSUTImplementation> getSystemsUnderTest();
  public abstract Set<KlarosTestCase> getTestCases();
  public abstract boolean isResolved();
}
```



B.1.1.8.2. getAcceptedIn()

```
@Deprecated(since="5.3.3")
public abstract KlarosSUTImplementation getAcceptedIn();
```

Gets the system under test this issue has been accepted/detected in.

Parameters

`return`	the system under test

B.1.1.8.3. getCreationDate()

```
public abstract Date getCreationDate();
```

Gets the creation date of this issue.

Parameters

`return` the creation date of this issue.

B.1.1.8.4. getDescription()

```
public abstract String getDescription();
```

Get the description.

Parameters

`return` The description of this issue.

B.1.1.8.5. getExternalId()

```
public abstract String getExternalId();
```

Gets the external id of this issue, if available. This id is generated by the external issue management system upon creation of the issue.

Parameters

`return` the external id

B.1.1.8.6. getIssueManagement()

```
public abstract KlarosIssueManagement getIssueManagement();
```

Gets the issue management this issue belongs to.

Parameters

`return` the issue management

B.1.1.8.7. getLastSynched()

```
public abstract Date getLastSynched();
```

Gets the last synchronization date of this issue.

Parameters

`return` the last synchronization date of this issue.

B.1.1.8.8. getOwner()

```
public abstract String getOwner();
```

Gets the name of the owner/assignee of this issue.

Parameters

`return` the owner name of this issue.

B.1.1.8.9. getPriority()

```
public abstract String getPriority();
```

Gets the priority of this issue. The format and content of this value varies with the issue management system and its configuration.

Parameters

return	the priority of this issue.

B.1.1.8.10. getRemoteBrowseUrl()

```
public abstract String getRemoteBrowseUrl();
```

Returns an url to browse the issue in the remote issue management system.

Parameters

return	the remote browse url

B.1.1.8.11. getReporter()

```
public abstract String getReporter();
```

Gets the name of the reporter of this issue.

Parameters

return	the reporter name of this issue.

B.1.1.8.12. getResolution()

```
public abstract String getResolution();
```

Gets the resolution of this issue. The format and content of this value varies with the issue management system and its configuration.

Parameters

return	the resolution of this issue.

B.1.1.8.13. getState()

```
public abstract String getState();
```

Gets the state of this issue. The format and content of this value varies with the issue management system and its configuration.

Parameters

return	the state string of this issue.

B.1.1.8.14. getSubject()

```
public abstract String getSubject();
```

Gets the subject of this issue.

Parameters

return	the subject string of this issue.

B.1.1.8.15. getSystemsUnderTest()

```
public abstract Set<KlarosSUTImplementation> getSystemsUnderTest();
```

Get the related systems under test.

Parameters

`return`	`Set` of systems under test which have detected this issue.

B.1.1.8.16. getTestCases()

```
public abstract Set<KlarosTestCase> getTestCases();
```

Get the related test cases.

Parameters

`return`	`Set` of test cases which have detected this issue.

B.1.1.8.17. isResolved()

```
public abstract boolean isResolved();
```

Checks if this issue is resolved.

Parameters

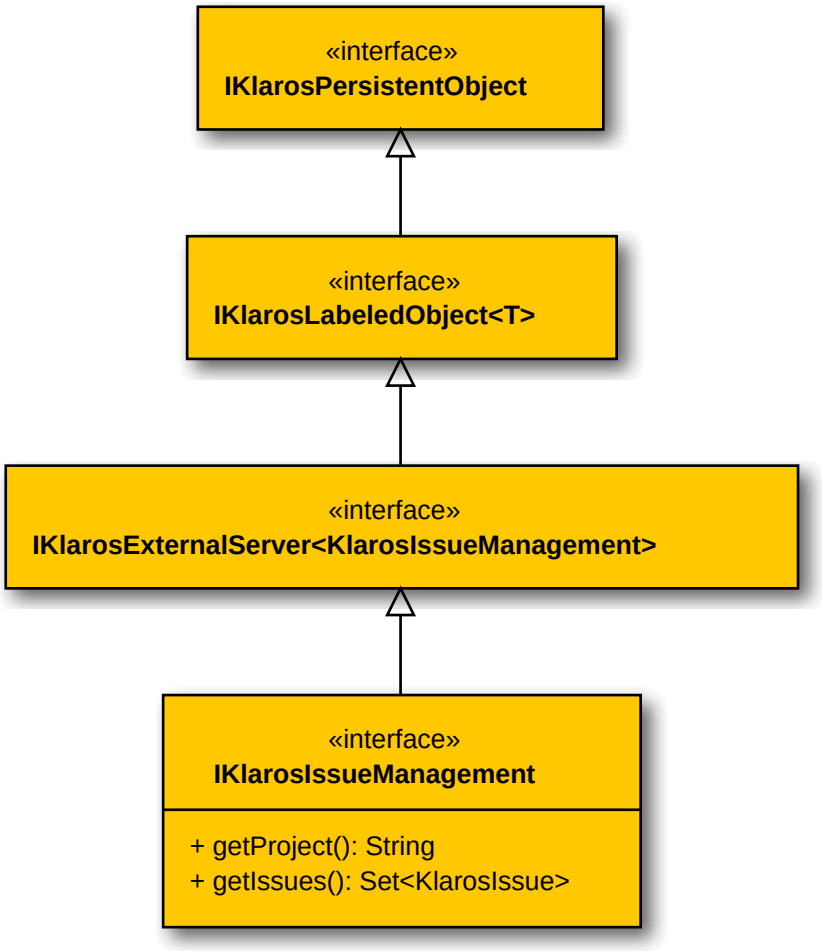
`return`	true, if resolved

B.1.1.9. IKlarosIssueManagement

The interface for accessing issue management systems.

B.1.1.9.1. Synopsis

```
public interface IKlarosIssueManagement
 extends IKlarosExternalServer<KlarosIssueManagement> {

  // Public Methods
  public abstract Set<KlarosIssue> getIssues();
  public abstract String getProject();
}
```



B.1.1.9.2. getIssues()

```
public abstract Set<KlarosIssue> getIssues();
```

Gets the referenced issues of this system.

Parameters

`return`	the issues

B.1.1.9.3. getProject()

```
public abstract String getProject();
```

Get the optional project of this system.

Parameters

`return`	The project

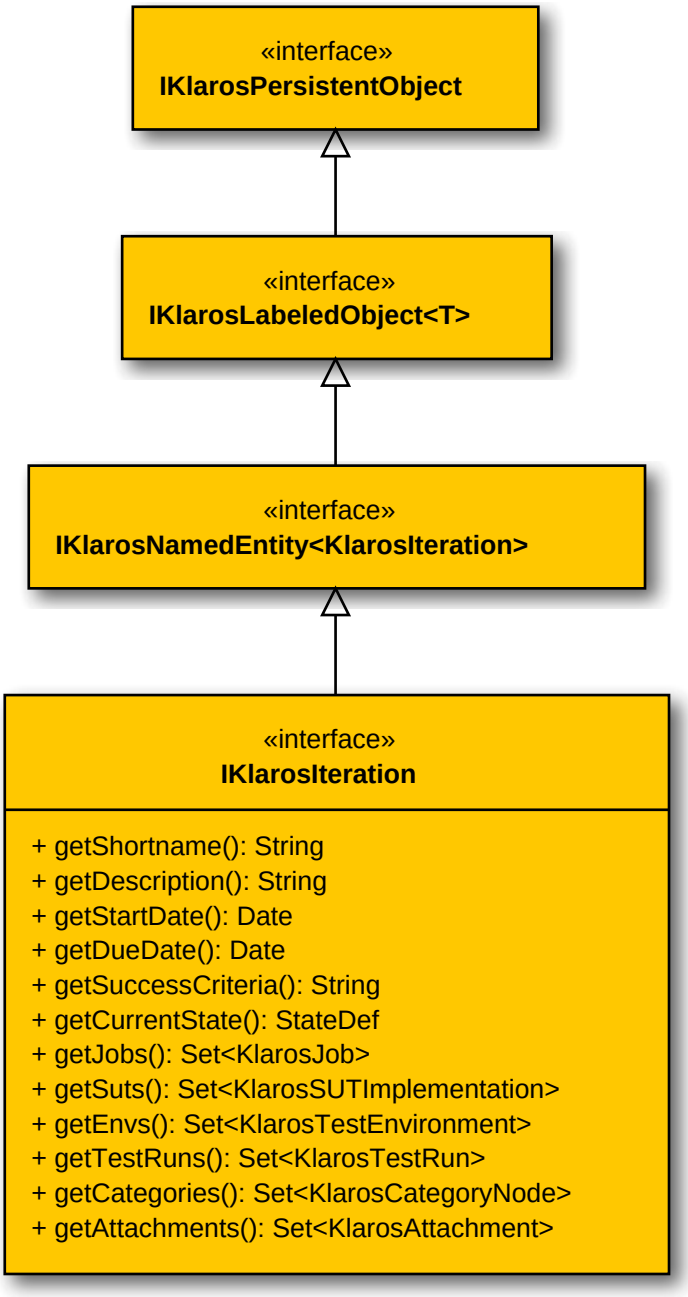
B.1.1.10. IKlarosIteration

An iteration in a project.

B.1.1.10.1. Synopsis

```
public interface IKlarosIteration extends IKlarosNamedEntity<KlarosIteration> {
```

```
  // Public Methods
  public abstract Set<KlarosAttachment> getAttachments();
  public abstract Set<KlarosCategoryNode> getCategories();
  public abstract StateDef getCurrentState();
  public abstract String getDescription();
  public abstract Date getDueDate();
  public abstract Set<KlarosTestEnvironment> getEnvs();
  public abstract Set<KlarosJob> getJobs();
  public abstract String getShortname();
  public abstract Date getStartDate();
  public abstract String getSuccessCriteria();
  public abstract Set<KlarosSUTImplementation> getSuts();
  public abstract Set<KlarosTestRun> getTestRuns();
}
```



B.1.1.10.2. getAttachments()

```
public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this iteration.

Parameters

`return` the attachments

B.1.1.10.3. getCategories()

```
public abstract Set<KlarosCategoryNode> getCategories();
```

Gets the categories this object belongs to. Each category node will belong to a different category tree.

Parameters

`return` the category nodes

B.1.1.10.4. getCurrentState()

```
public abstract StateDef getCurrentState();
```

Gets the current state.

Parameters

`return` the current state

B.1.1.10.5. getDescription()

```
public abstract String getDescription();
```

Gets the description.

Parameters

`return` the description

B.1.1.10.6. getDueDate()

```
public abstract Date getDueDate();
```

Gets the due date.

Parameters

`return` the due date

B.1.1.10.7. getEnvs()

```
public abstract Set<KlarosTestEnvironment> getEnvs();
```

Gets the test environments associated with this iteration.

Parameters

`return` the test environments

B.1.1.10.8. getJobs()

```
public abstract Set<KlarosJob> getJobs();
```

Gets the jobs associated with this iteration.

Parameters

`return` the jobs

B.1.1.10.9. getShortname()

```
public abstract String getShortname();
```

Gets the short name.

Parameters

`return` the short name

B.1.1.10.10. getStartDate()

```
public abstract Date getStartDate();
```

Gets the start date.

Parameters

`return` the start date

B.1.1.10.11. getSuccessCriteria()

```
public abstract String getSuccessCriteria();
```

Gets the success criteria.

Parameters

`return` the success criteria

B.1.1.10.12. getSuts()

```
public abstract Set<KlarosSUTImplementation> getSuts();
```

Gets the systems under test associated with this iteration.

Parameters

`return` the systems under test

B.1.1.10.13. getTestRuns()

```
public abstract Set<KlarosTestRun> getTestRuns();
```

Gets the test runs associated with this iteration.

Parameters

`return` the test runs

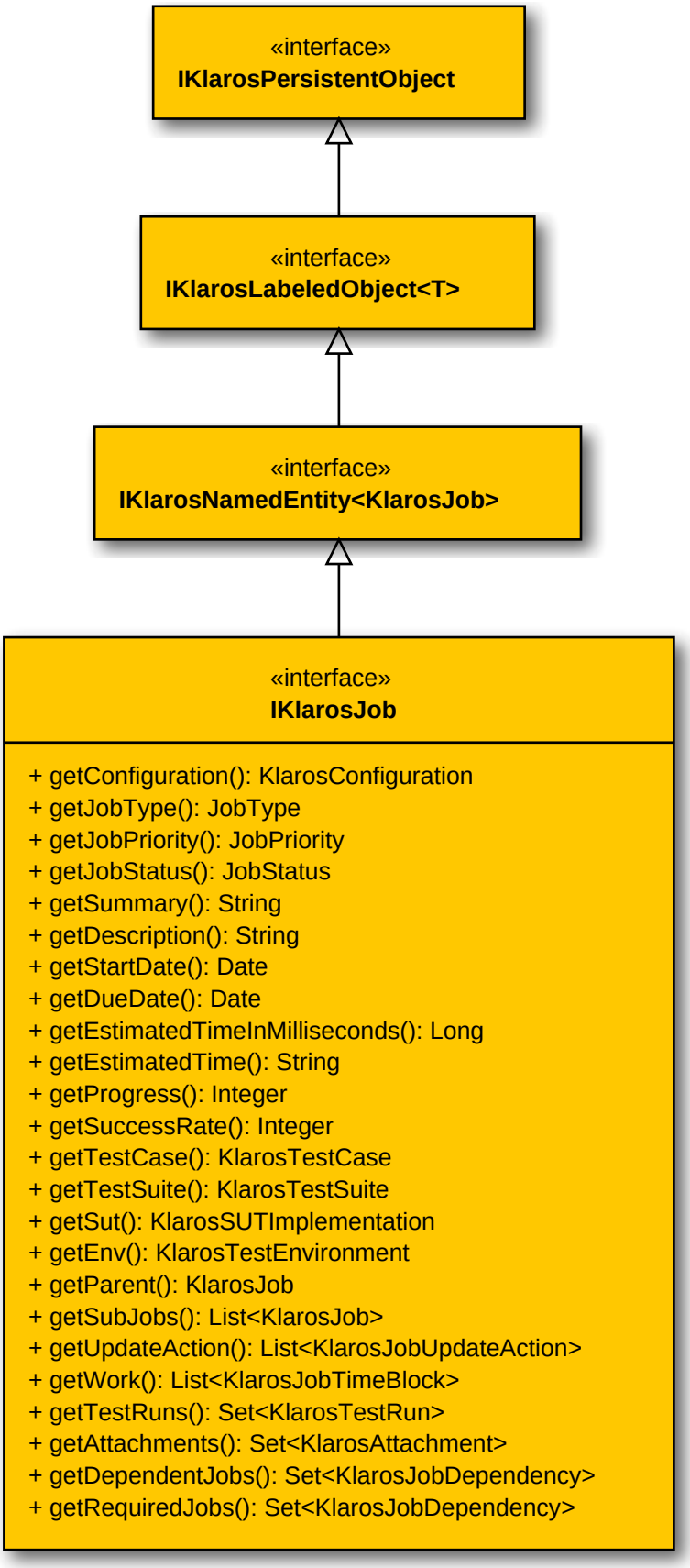
B.1.1.11. IKlarosJob

This interface provides access to the data of a job.

B.1.1.11.1. Synopsis

```
public interface IKlarosJob extends IKlarosNamedEntity<KlarosJob> {

  // Public Methods
  public abstract Set<KlarosAttachment> getAttachments();
  public abstract KlarosConfiguration getConfiguration();
  public abstract Set<KlarosJobDependency> getDependentJobs();
  public abstract String getDescription();
  public abstract Date getDueDate();
  public abstract KlarosTestEnvironment getEnv();
  public abstract String getEstimatedTime();
  public abstract Long getEstimatedTimeInMilliseconds();
  public abstract JobPriority getJobPriority();
  public abstract JobStatus getJobStatus();
  public abstract JobType getJobType();
  public abstract KlarosJob getParent();
  public abstract Integer getProgress();
```

```
  public abstract Set<KlarosJobDependency> getRequiredJobs();
  public abstract Date getStartDate();
  public abstract List<KlarosJob> getSubJobs();
  public abstract Integer getSuccessRate();
  public abstract String getSummary();
  public abstract KlarosSUTImplementation getSut();
  public abstract KlarosTestCase getTestCase();
  public abstract Set<KlarosTestRun> getTestRuns();
  public abstract KlarosTestSuite getTestSuite();
  public abstract List<KlarosJobUpdateAction> getUpdateAction();
  public abstract List<KlarosJobTimeBlock> getWork();
}
```


«interface»
IKlarosPersistentObject
«interface»
IKlarosLabeledObject<T>
«interface»
IKlarosNamedEntity<KlarosJob>
«interface»
IKlarosJob
+ getConfiguration(): KlarosConfiguration
+ getJobType(): JobType
+ getJobPriority(): JobPriority
+ getJobStatus(): JobStatus
+ getSummary(): String
+ getDescription(): String
+ getStartDate(): Date
+ getDueDate(): Date
+ getEstimatedTimeInMilliseconds(): Long
+ getEstimatedTime(): String
+ getProgress(): Integer
+ getSuccessRate(): Integer
+ getTestCase(): KlarosTestCase
+ getTestSuite(): KlarosTestSuite
+ getSut(): KlarosSUTImplementation
+ getEnv(): KlarosTestEnvironment
+ getParent(): KlarosJob
+ getSubJobs(): List<KlarosJob>
+ getUpdateAction(): List<KlarosJobUpdateAction>
+ getWork(): List<KlarosJobTimeBlock>
+ getTestRuns(): Set<KlarosTestRun>
+ getAttachments(): Set<KlarosAttachment>
+ getDependentJobs(): Set<KlarosJobDependency>
+ getRequiredJobs(): Set<KlarosJobDependency>

B.1.1.11.2. getAttachments()

```
public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this job.

Parameters

`return` the attachments

B.1.1.11.3. getDependentJobs()

```
public abstract Set<KlarosJobDependency> getDependentJobs();
```

Gets the jobs depending on this jobs state.

Parameters

`return` the dependent jobs

B.1.1.11.4. getRequiredJobs()

```
public abstract Set<KlarosJobDependency> getRequiredJobs();
```

Gets the jobs on which states this job is depending on.

Parameters

`return` the required jobs

B.1.1.11.5. getTestRuns()

```
public abstract Set<KlarosTestRun> getTestRuns();
```

Gets the test runs executed with this job.

Parameters

`return` the test runs

B.1.1.11.6. getUpdateAction()

```
public abstract List<KlarosJobUpdateAction> getUpdateAction();
```

Gets the update action list.

Parameters

`return` the update action

B.1.1.11.7. getWork()

```
public abstract List<KlarosJobTimeBlock> getWork();
```

Gets the list of work items done on this job.

Parameters

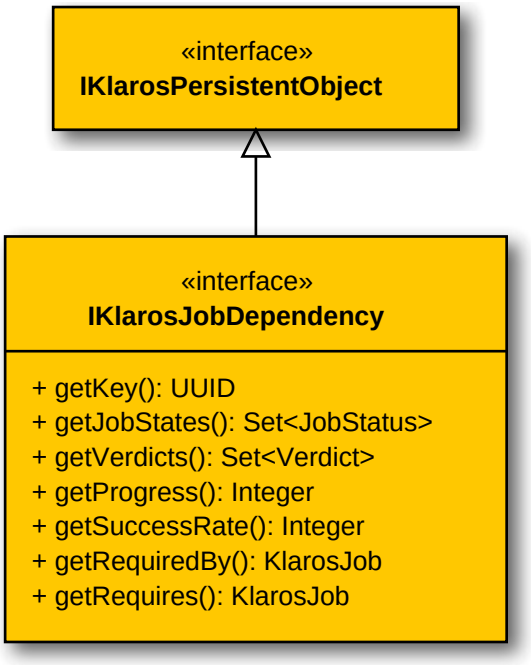
`return` the work items

B.1.1.12. IKlarosJobDependency

This interface provides access to the data of a job.

B.1.1.12.1. Synopsis

```
public interface IKlarosJobDependency extends IKlarosPersistentObject {

  // Public Methods
  public abstract Set<JobStatus> getJobStates();
  public abstract UUID getKey();
  public abstract Integer getProgress();
  public abstract KlarosJob getRequiredBy();
  public abstract KlarosJob getRequires();
  public abstract Integer getSuccessRate();
  public abstract Set<Verdict> getVerdicts();
}
```



B.1.1.12.2. getJobStates()

```
public abstract Set<JobStatus> getJobStates();
```

Gets the job states.

Parameters

`return` the job states

B.1.1.12.3. getKey()

```
public abstract UUID getKey();
```

Get the internal key of the object. This key is globally unique.

Parameters

`return` The internal key.

B.1.1.12.4. getRequiredBy()

```
public abstract KlarosJob getRequiredBy();
```

Gets the job this dependency is required by.

Parameters

`return` the job this dependency is required by

B.1.1.12.5. getRequires()

```
public abstract KlarosJob getRequires();
```

Returns the job this dependency requires for evaluation.

Parameters

`return` the job this dependency requires

B.1.1.12.6. getVerdicts()

```
public abstract Set<Verdict> getVerdicts();
```

Gets the verdicts.

Parameters

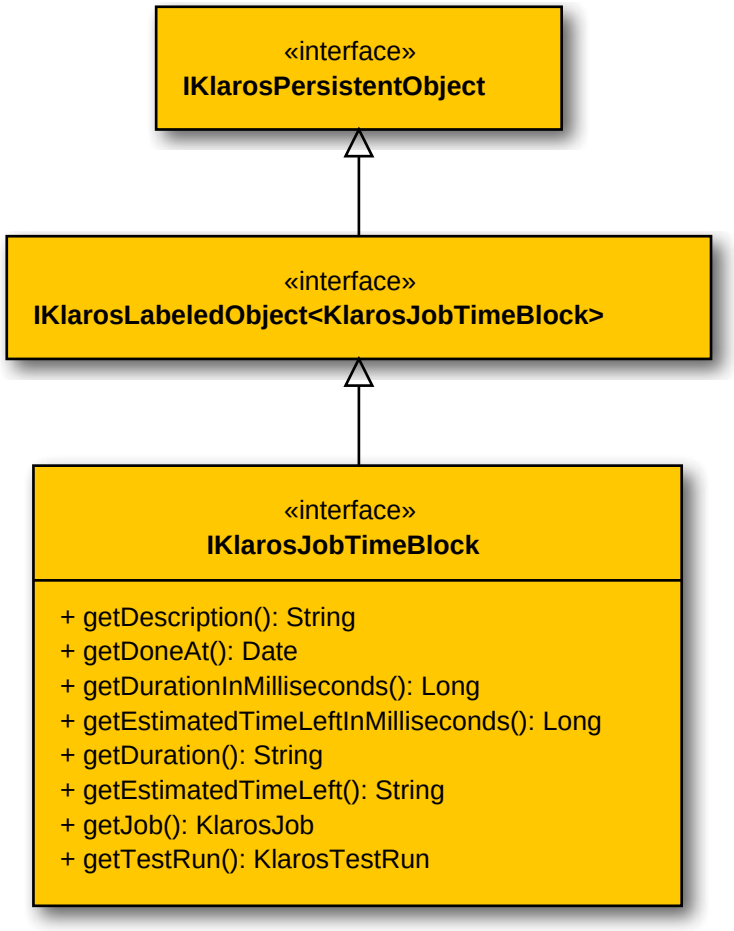
`return` the verdicts

B.1.1.13. IKlarosJobTimeBlock

The job time block defines a certain amount of time the job has been worked on.

B.1.1.13.1. Synopsis

```
public interface IKlarosJobTimeBlock
 extends IKlarosLabeledObject<KlarosJobTimeBlock> {

  // Public Methods
  public abstract String getDescription();
  public abstract Date getDoneAt();
  public abstract String getDuration();
  public abstract Long getDurationInMilliseconds();
  public abstract String getEstimatedTimeLeft();
  public abstract Long getEstimatedTimeLeftInMilliseconds();
  public abstract KlarosJob getJob();
  public abstract KlarosTestRun getTestRun();
}
```



B.1.1.13.2. getDescription()

```
public abstract String getDescription();
```

Gets the work description.

Parameters

`return` the work description

B.1.1.13.3. getDoneAt()

```
public abstract Date getDoneAt();
```

Gets the date this work was done at.

Parameters

`return` the start date

B.1.1.13.4. getDuration()

```
public abstract String getDuration();
```

Gets the work duration as a String.

Parameters

`return` the duration

B.1.1.13.5. getDurationInMilliseconds()

```
public abstract Long getDurationInMilliseconds();
```

Gets the work duration in milliseconds.

Parameters

`return` the work duration in hours

B.1.1.13.6. getEstimatedTimeLeft()

```
public abstract String getEstimatedTimeLeft();
```

Gets the estimated time left for this job as a String.

Parameters

`return` the estimated time left

B.1.1.13.7. getEstimatedTimeLeftInMilliseconds()

```
public abstract Long getEstimatedTimeLeftInMilliseconds();
```

Gets the estimated time left for this job in milliseconds.

Parameters

`return` the estimated time left in hours

B.1.1.13.8. getJob()

```
public abstract KlarosJob getJob();
```

Gets the job this job time block belongs to.

Parameters

`return` the job

B.1.1.13.9. getTestRun()

```
public abstract KlarosTestRun getTestRun();
```

Gets the test run this job time block belongs to.

Parameters

`return` the test run

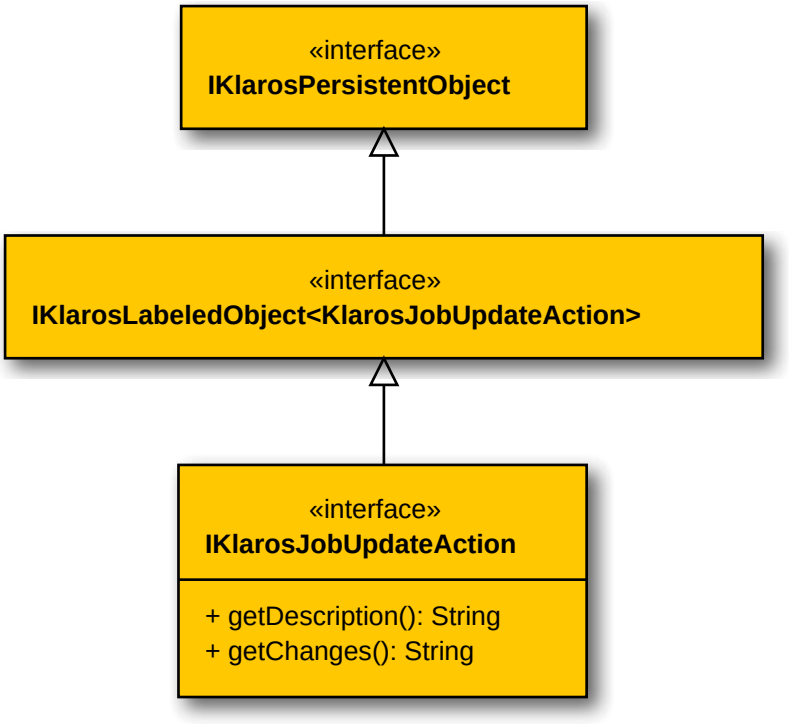
B.1.1.14. IKlarosJobUpdateAction

This interface provides access to the job update action.

B.1.1.14.1. Synopsis

```
public interface IKlarosJobUpdateAction
 extends IKlarosLabeledObject<KlarosJobUpdateAction> {

  // Public Methods
```

```
  public abstract String getChanges();
  public abstract String getDescription();
}
```



B.1.1.14.2. getChanges()

```
public abstract String getChanges();
```

Gets the changes.

Parameters

`return`	the changes

B.1.1.14.3. getDescription()

```
public abstract String getDescription();
```

Gets the description.

Parameters

`return`	the description

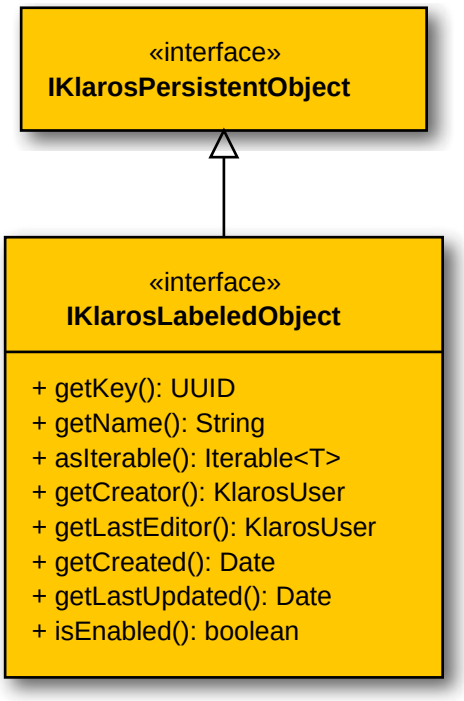
B.1.1.15. IKlarosLabeledObject

This interface provides access to the fields of a labeled object.

B.1.1.15.1. Synopsis

```
public interface IKlarosLabeledObject<T> extends IKlarosPersistentObject {
```

```
  // Public Methods
  public abstract Iterable<T> asIterable();
  public abstract Date getCreated();
  public abstract KlarosUser getCreator();
  public abstract UUID getKey();
  public abstract KlarosUser getLastEditor();
  public abstract Date getLastUpdated();
  public abstract String getName();
  public abstract boolean isEnabled();
}
```



B.1.1.15.2. asIterable()

```
public abstract Iterable<T> asIterable();
```

Return this object as an iterable containing just this object.

Parameters

return	the iterable container

B.1.1.15.3. getKey()

```
public abstract UUID getKey();
```

Get the internal key of the object. This key is globally unique.

Parameters

return	The internal key.

B.1.1.15.4. getName()

```
public abstract String getName();
```

Get the name of the object. This matches the id field visible in the UI.

Parameters

`return` The name of the object.

B.1.1.15.5. isEnabled()

```
public abstract boolean isEnabled();
```

Returns the value of the enabled flag of this object.

Parameters

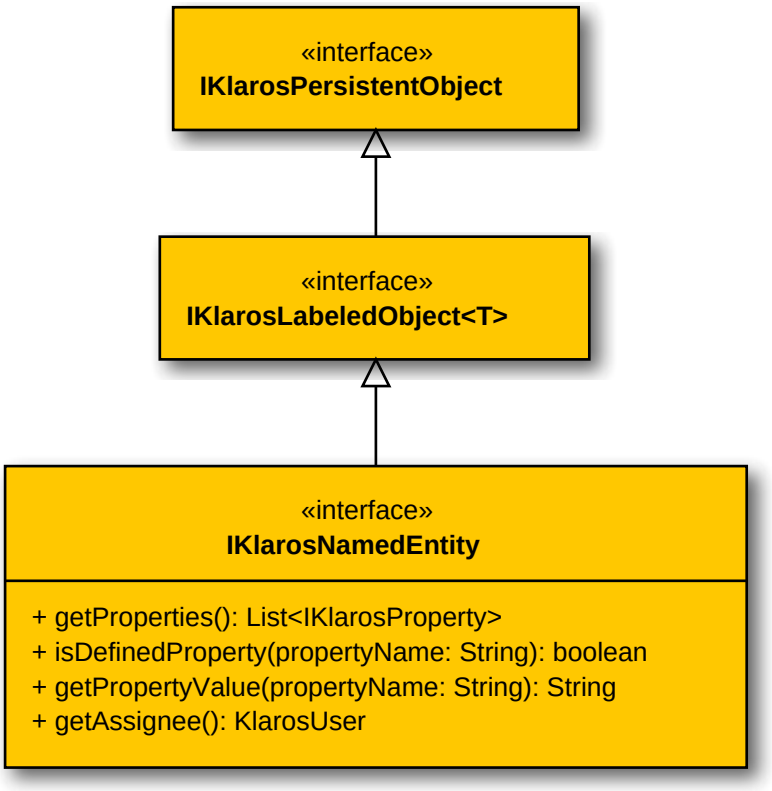
`return` `true` if the entity is enabled, else `false`

B.1.1.16. IKlarosNamedEntity

This interface provides access to data of a properties owner.

B.1.1.16.1. Synopsis

```
public interface IKlarosNamedEntity<T> extends IKlarosLabeledObject<T> {

  // Public Methods
  public abstract KlarosUser getAssignee();
  public abstract List<IKlarosProperty> getProperties();
  public abstract String getPropertyValue(String propertyName);
  public abstract boolean isDefinedProperty(String propertyName);
}
```



B.1.1.16.2. getProperties()

```
public abstract List<IKlarosProperty> getProperties();
```

Gets the list of properties.

Parameters

`return`	the properties

B.1.1.16.3. getPropertyValue(String)

```
public abstract String getPropertyValue(String propertyName);
```

Gets the property value for the given property name.

Parameters

`propertyName`	the property name
`return`	the property value or null if not name is not present

B.1.1.16.4. isDefinedProperty(String)

```
public abstract boolean isDefinedProperty(String propertyName);
```

Check if a property identified by given name is a defined property.

Parameters

`propertyName`	The name of the property to check.
`return`	`true` if the identified property is a defined property, `false` else.

B.1.1.17. IKlarosPersistentObject

Base interface.

B.1.1.17.1. Synopsis

```
public interface IKlarosPersistentObject {
}
```

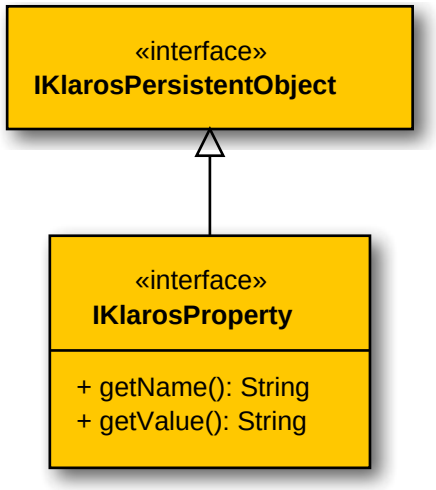
B.1.1.18. IKlarosProperty

A user defined property.

B.1.1.18.1. Synopsis

```
public interface IKlarosProperty extends IKlarosPersistentObject {
```

```
  // Public Methods
  public abstract String getName();
  public abstract String getValue();
}
```



B.1.1.18.2. getName()

```
public abstract String getName();
```

Gets the property name.

Parameters

return	the property name

B.1.1.18.3. getValue()

```
public abstract String getValue();
```

Gets the property value.

Parameters

return	the property value

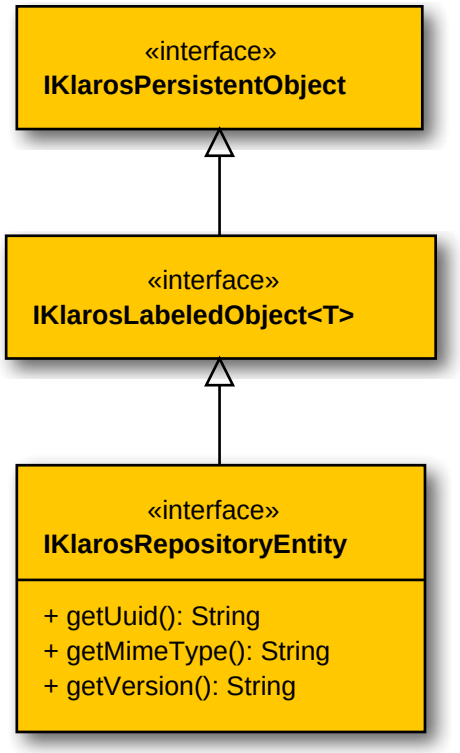
B.1.1.19. IKlarosRepositoryEntity

This interface provides access to binary content.

B.1.1.19.1. Synopsis

```
public interface IKlarosRepositoryEntity<T> extends IKlarosLabeledObject<T> {

  // Public Methods
  public abstract String getMimeType();
  public abstract String getUuid();
  public abstract String getVersion();
```

```
}
```



B.1.1.19.2. getMimeType()

```
public abstract String getMimeType();
```

Gets the mime type of the attachment.

Parameters

`return` the mime type

B.1.1.19.3. getUuid()

```
public abstract String getUuid();
```

Get the unique id of this attachment. This may be resolved by calling KlarosContext.-getAttachmentURL() to an URL pointing to the attachment data.

Parameters

`return` The uuid of this attachment.

B.1.1.19.4. getVersion()

```
public abstract String getVersion();
```

Gets the version of this attachment.

Parameters

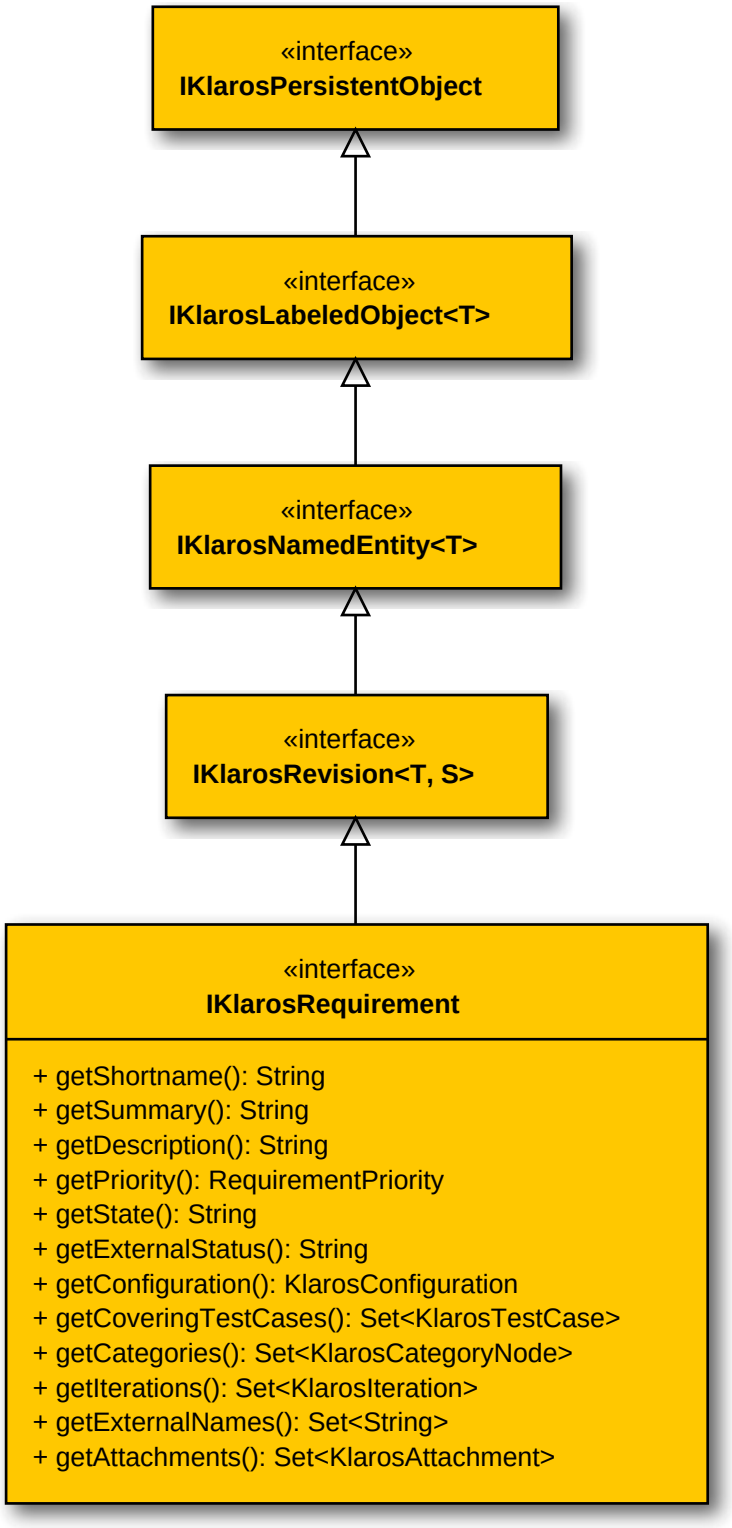
`return` the version

B.1.1.20. IKlarosRequirement

This interface provides access to data of a requirement.

B.1.1.20.1. Synopsis

```
public interface IKlarosRequirement<T,S extends Revision<?>>
 extends IKlarosRevision<T, S> {

  // Public Methods
  public abstract Set<KlarosAttachment> getAttachments();
  public abstract Set<KlarosCategoryNode> getCategories();
  public abstract KlarosConfiguration getConfiguration();
  public abstract Set<KlarosTestCase> getCoveringTestCases();
  public abstract String getDescription();
  public abstract Set<String> getExternalNames();
  public abstract String getExternalStatus();
  public abstract Set<KlarosIteration> getIterations();
  public abstract RequirementPriority getPriority();
  public abstract String getShortname();
  public abstract String getState();
  public abstract String getSummary();
}
```



B.1.1.20.2. getAttachments()

```
public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this requirement.

Parameters

`return` the attachments

B.1.1.20.3. getCategories()

```
public abstract Set<KlarosCategoryNode> getCategories();
```

Gets the categories this object belongs to. Each category node will belong to a different category tree.

Parameters

`return` the category nodes

B.1.1.20.4. getConfiguration()

```
public abstract KlarosConfiguration getConfiguration();
```

Get configuration.

Parameters

`return` The related configuration.

B.1.1.20.5. getCoveringTestCases()

```
public abstract Set<KlarosTestCase> getCoveringTestCases();
```

Get test cases covering the requirement.

Parameters

`return` `Set` of test cases which cover this test requirement.

B.1.1.20.6. getDescription()

```
public abstract String getDescription();
```

Get the description.

Parameters

`return` the string

B.1.1.20.7. getExternalNames()

```
public abstract Set<String> getExternalNames();
```

Gets the external requirement names this requirement is associated with. External requirement names are used to match imported requirements to existing requirements.

Parameters

`return` the external names

B.1.1.20.8. getExternalStatus()

```
public abstract String getExternalStatus();
```

Gets the external status.

Parameters

`return` the external status

B.1.1.20.9. getIterations()

```
public abstract Set<KlarosIteration> getIterations();
```

Get the iterations this requirement is assigned to.

Parameters

`return` `Set` of iterations this requirement is assigned to.

B.1.1.20.10. getPriority()

```
public abstract RequirementPriority getPriority();
```

Gets the priority.

Parameters

`return` the priority

B.1.1.20.11. getShortname()

```
public abstract String getShortname();
```

Gets the shortname.

Parameters

`return` the shortname

B.1.1.20.12. getState()

```
public abstract String getState();
```

Gets the state.

Parameters

`return` the state

B.1.1.20.13. getSummary()

```
public abstract String getSummary();
```

Gets the summary.

Parameters

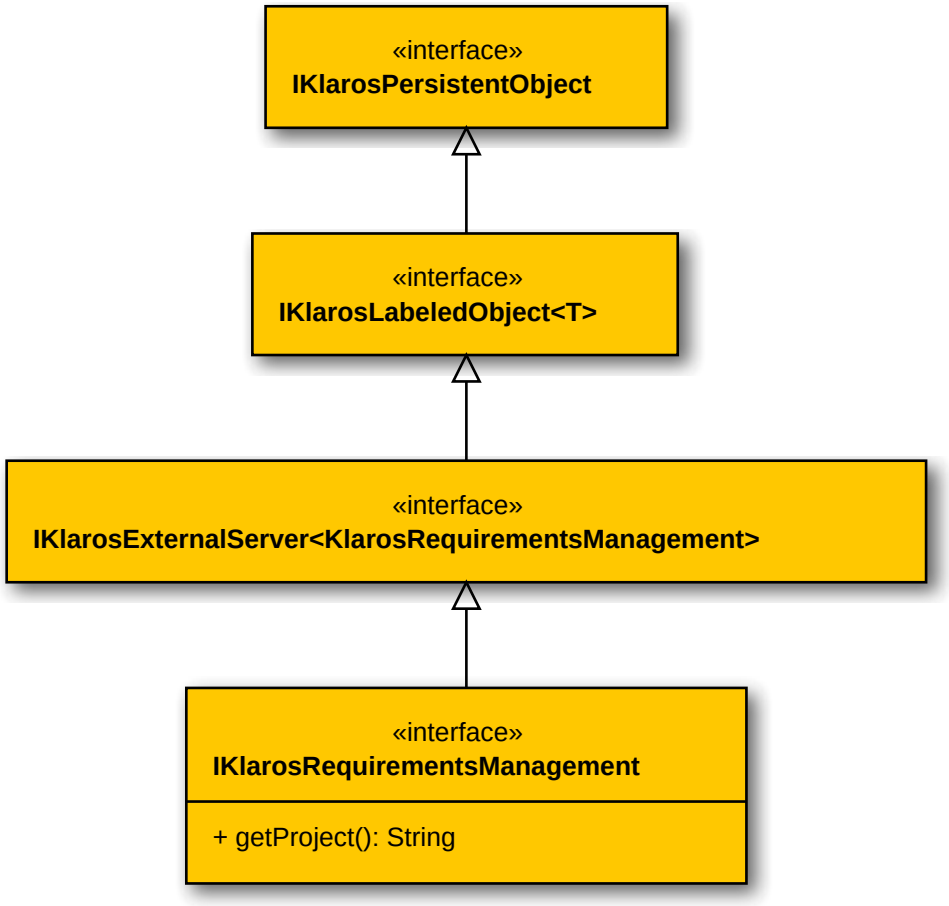
`return` the summary

B.1.1.21. IKlarosRequirementsManagement

The interface for accessing requirements management systems.

B.1.1.21.1. Synopsis

```
public interface IKlarosRequirementsManagement
 extends IKlarosExternalServer<KlarosRequirementsManagement> {

  // Public Methods
  public abstract String getProject();
}
```



B.1.1.21.2. getProject()

```
public abstract String getProject();
```

Get the optional project of this system.

Parameters

`return` The project

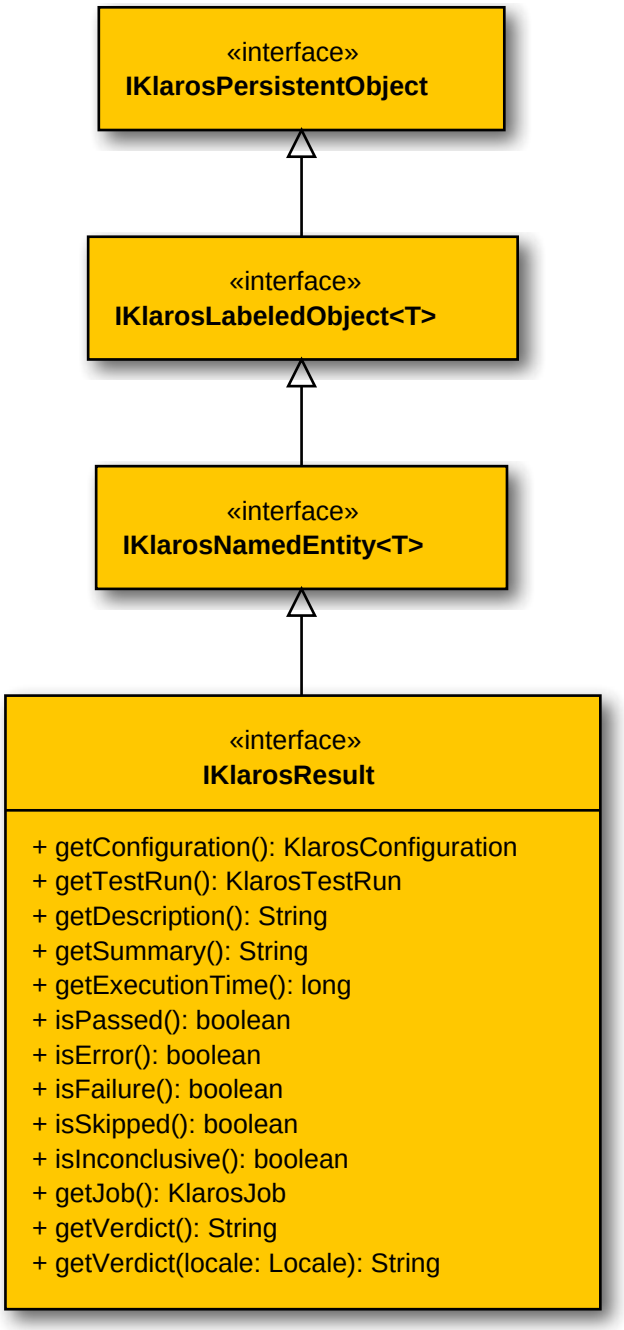
B.1.1.22. IKlarosResult

This interface provides access to data of generic result.

B.1.1.22.1. Synopsis

```
public interface IKlarosResult<T> extends IKlarosNamedEntity<T> {

  // Public Methods
  public abstract KlarosConfiguration getConfiguration();
  public abstract String getDescription();
  public abstract long getExecutionTime();
  public abstract KlarosJob getJob();
  public abstract String getSummary();
  public abstract KlarosTestRun getTestRun();
  public abstract String getVerdict();
```

```
  public abstract String getVerdict(Locale locale);
  public abstract boolean isError();
  public abstract boolean isFailure();
  public abstract boolean isInconclusive();
  public abstract boolean isPassed();
  public abstract boolean isSkipped();
}
```



B.1.1.22.2. getConfiguration()

```
  public abstract KlarosConfiguration getConfiguration();
```

Get configuration.

Parameters

`return` The related configuration

B.1.1.22.3. getDescription()

```
public abstract String getDescription();
```

Get the test result description. This is usually set for failed/error status results.

Parameters

`return` The test result description.

B.1.1.22.4. getExecutionTime()

```
public abstract long getExecutionTime();
```

Get the test execution time in ms.

Parameters

`return` The test execution time.

B.1.1.22.5. getJob()

```
public abstract KlarosJob getJob();
```

Gets the job this result belongs if one exists.

Parameters

`return` the associated job

B.1.1.22.6. getSummary()

```
public abstract String getSummary();
```

Get the test result summary. This is usually set for failed/error status results.

Parameters

`return` The test result summary.

B.1.1.22.7. getTestRun()

```
public abstract KlarosTestRun getTestRun();
```

Get the associated test run.

Parameters

`return` The test run that created this result.

B.1.1.22.8. getVerdict()

```
public abstract String getVerdict();
```

Returns the Test Case Result verdict as String. P = Passed U = Unknown S = Skipped E = Error F = Failed

Parameters

`return` The Test Case Result verdict as String.

B.1.1.22.9. getVerdict(Locale)

```
public abstract String getVerdict(Locale locale);
```

Returns the Test Case Result verdict as String with localization. P = Passed U = Unknown S = Skipped E = Error F = Failed

Parameters

`locale`	set location Information
`return`	The Test Case Result verdict as String.

B.1.1.22.10. isError()

```
public abstract boolean isError();
```

Check if this is an error result. It is assumed, that error results have a property 'type' with the value 'E' or 'error'.

Parameters

`return`	`true` if this results represents an error.

B.1.1.22.11. isFailure()

```
public abstract boolean isFailure();
```

Check if this is a failure result. It is assumed, that failure results have a property 'type' with the value 'F' or 'failure'.

Parameters

`return`	`true` if this results represents a failure.

B.1.1.22.12. isInconclusive()

```
public abstract boolean isInconclusive();
```

Check if this is an inconclusive result. It is assumed, that inconclusive results have a property 'type' with the value 'I' or 'inconclusive'.

Parameters

`return`	`true` if this results represents an inconclusive result.

B.1.1.22.13. isPassed()

```
public abstract boolean isPassed();
```

Check if this is a result of a passed test case. It is assumed, that passed results have a property 'type' with the value 'P' or 'error'.

Parameters

`return`	`true` if this results represents an error.

B.1.1.22.14. isSkipped()

```
public abstract boolean isSkipped();
```

Check if this is a skipped result. It is assumed, that skipped results have a property 'type' with the value 'S' or 'skipped'.

Parameters

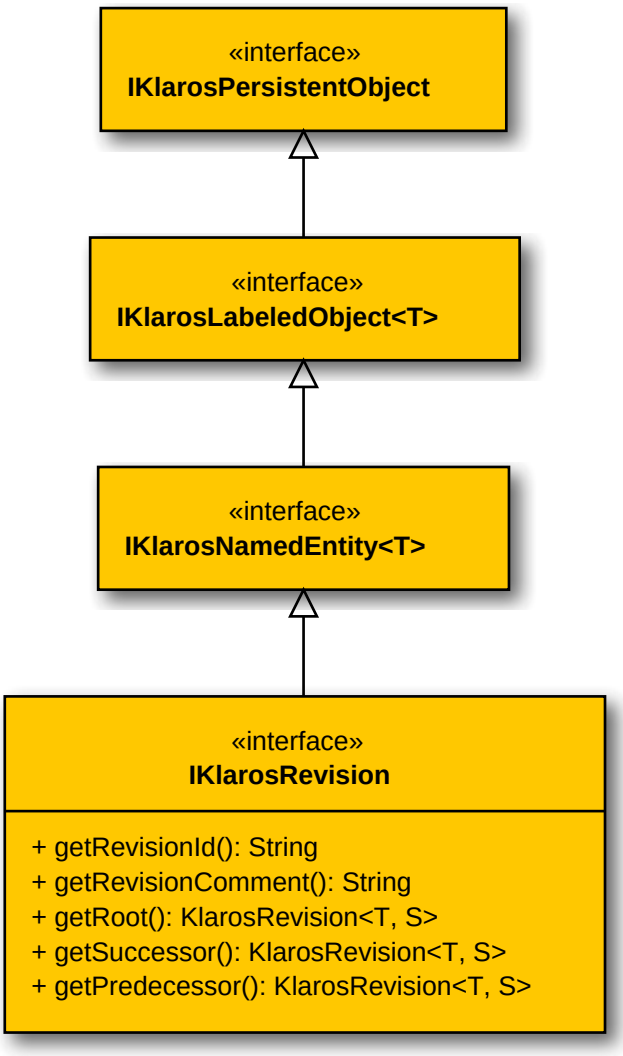
`return` `true` if this results represents a skipped result.

B.1.1.23. IKlarosRevision

This interface provides access to a revisionable Klaros object.

B.1.1.23.1. Synopsis

```
public interface IKlarosRevision<T,S extends Revision<?>>
 extends IKlarosNamedEntity<T> {

  // Public Methods
  public abstract KlarosRevision<T, S> getPredecessor();
  public abstract String getRevisionComment();
  public abstract String getRevisionId();
  public abstract KlarosRevision<T, S> getRoot();
  public abstract KlarosRevision<T, S> getSuccessor();
}
```



B.1.1.23.2. getPredecessor()

```
public abstract KlarosRevision<T, S> getPredecessor();
```

Get the predecessor of the revision.

Parameters

`return`	The revision object that is the predecessor of this revision.

B.1.1.23.3. getRevisionComment()

```
public abstract String getRevisionComment();
```

Get comment.

Parameters

`return`	The comment of the revision.

B.1.1.23.4. getRevisionId()

```
public abstract String getRevisionId();
```

Get the revision id.

Parameters

`return`	The revision id.

B.1.1.23.5. getRoot()

```
public abstract KlarosRevision<T, S> getRoot();
```

Get the root of the revision hierarchy.

Parameters

`return`	The root revision object.

B.1.1.23.6. getSuccessor()

```
public abstract KlarosRevision<T, S> getSuccessor();
```

Get the successor of the revision.

Parameters

`return`	The revision object that is the successor of this revision.

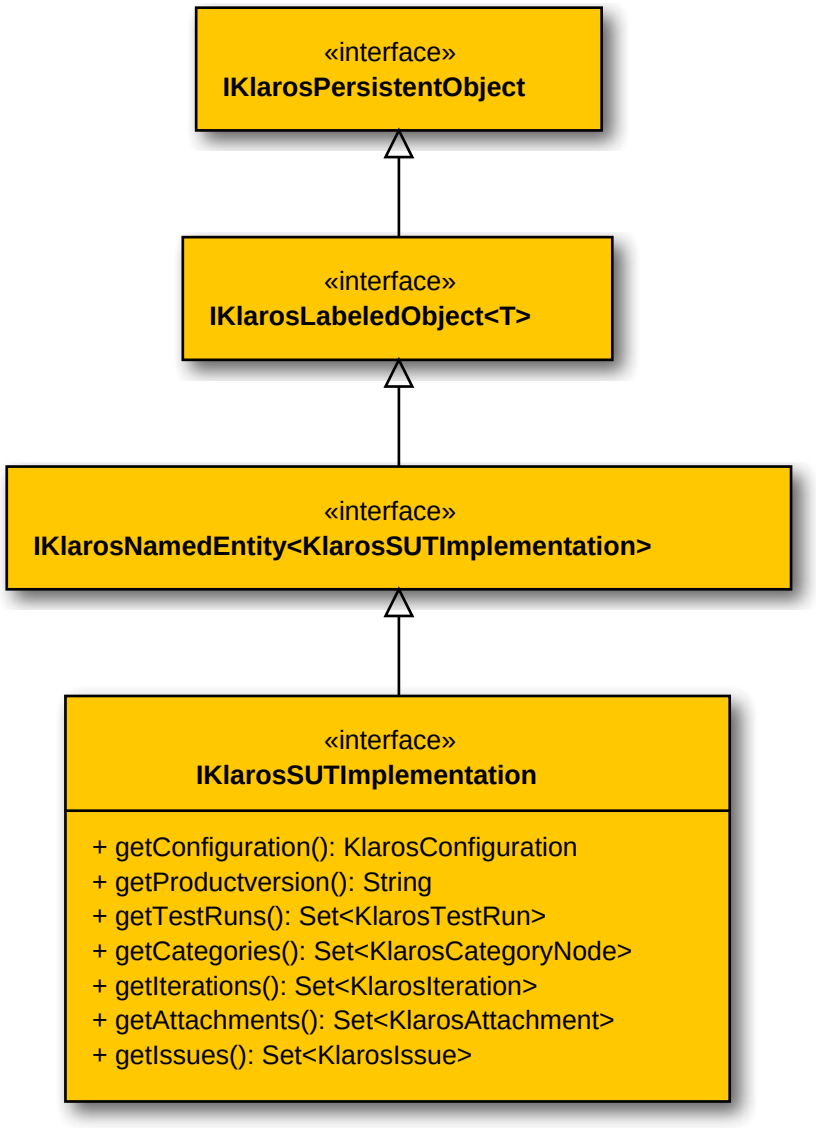
B.1.1.24. IKlarosSUTImplementation

This interface provides access to data of a system under test version.

B.1.1.24.1. Synopsis

```
public interface IKlarosSUTImplementation
 extends IKlarosNamedEntity<KlarosSUTImplementation> {
```

```
  // Public Methods
  public abstract Set<KlarosAttachment> getAttachments();
  public abstract Set<KlarosCategoryNode> getCategories();
  public abstract KlarosConfiguration getConfiguration();
  public abstract Set<KlarosIssue> getIssues();
  public abstract Set<KlarosIteration> getIterations();
  public abstract String getProductversion();
  public abstract Set<KlarosTestRun> getTestRuns();
}
```



B.1.1.24.2. getAttachments()

```
  public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this system under test.

Parameters

`return` the attachments

B.1.1.24.3. getCategories()

```
public abstract Set<KlarosCategoryNode> getCategories();
```

Gets the categories this system under test belongs to. Each category node will belong to a different category tree.

Parameters

`return` the category nodes

B.1.1.24.4. getConfiguration()

```
public abstract KlarosConfiguration getConfiguration();
```

Get configuration.

Parameters

`return` The related configuration.

B.1.1.24.5. getIssues()

```
public abstract Set<KlarosIssue> getIssues();
```

Gets the issues related to this system under test.

Parameters

`return` the issues

B.1.1.24.6. getIterations()

```
public abstract Set<KlarosIteration> getIterations();
```

Get the iterations this system under test is assigned to.

Parameters

`return` `Set` of iterations this system under test is assigned to.

B.1.1.24.7. getProductversion()

```
public abstract String getProductversion();
```

Get product version.

Parameters

`return` The version id of the system under test.

B.1.1.24.8. getTestRuns()

```
public abstract Set<KlarosTestRun> getTestRuns();
```

Get the test runs performed with this system under test.

Parameters

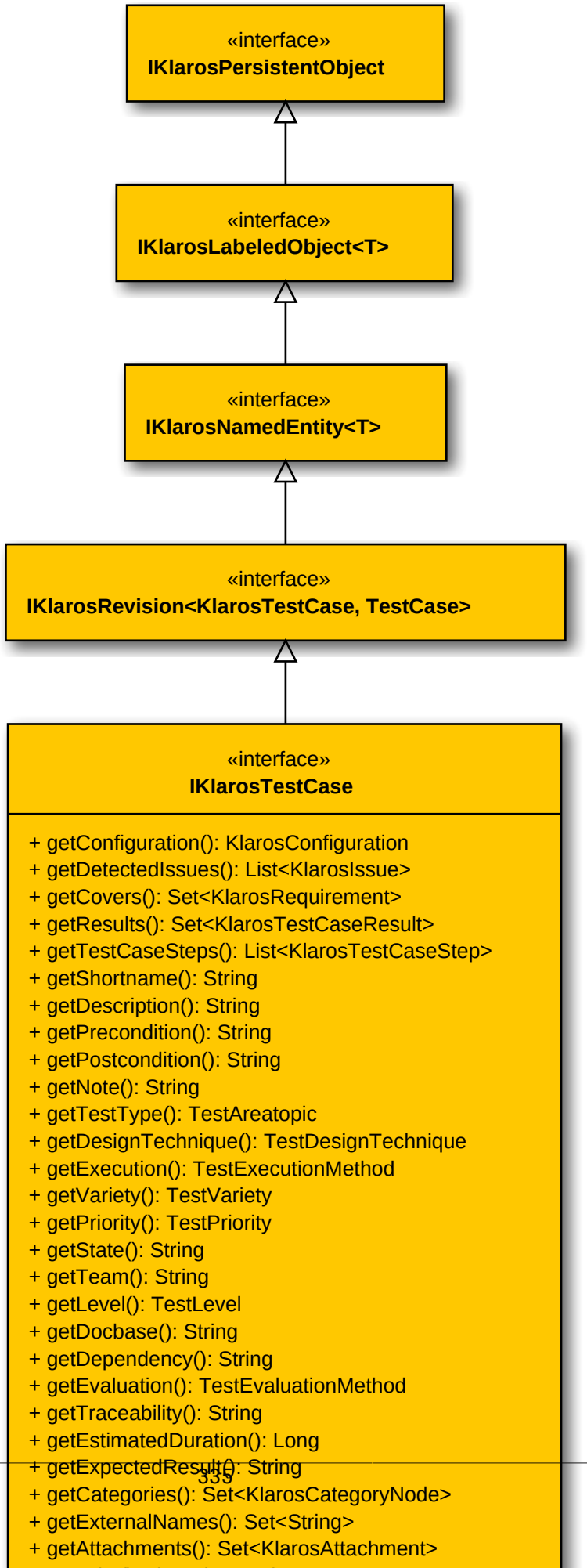
`return` `Set` of test runs performed for this system under test.

B.1.1.25. IKlarosTestCase

This class provides access to data of a test case.

B.1.1.25.1. Synopsis

```
public interface IKlarosTestCase extends IKlarosRevision<KlarosTestCase, TestCase> {

  // Public Methods
  public abstract Set<KlarosAttachment> getAttachments();
  public abstract Set<KlarosCategoryNode> getCategories();
  public abstract KlarosConfiguration getConfiguration();
  public abstract Set<KlarosRequirement> getCovers();
  public abstract String getDependency();
  public abstract String getDescription();
  public abstract TestDesignTechnique getDesignTechnique();
  public abstract List<KlarosIssue> getDetectedIssues();
  public abstract String getDocbase();
  public abstract Long getEstimatedDuration();
  public abstract TestEvaluationMethod getEvaluation();
  public abstract TestExecutionMethod getExecution();
  public abstract String getExpectedResult();
  public abstract Set<String> getExternalNames();
  public abstract List<KlarosJob> getJobs();
  public abstract TestLevel getLevel();
  public abstract String getNote();
  public abstract String getPostcondition();
  public abstract String getPrecondition();
  public abstract TestPriority getPriority();
  public abstract Set<KlarosTestCaseResult> getResults();
  public abstract String getShortname();
  public abstract String getState();
  public abstract String getTeam();
  public abstract List<KlarosTestCaseStep> getTestCaseSteps();
  public abstract TestAreatopic getTestType();
  public abstract String getTraceability();
  public abstract TestVariety getVariety();
}
```


«interface»
IKlarosPersistentObject
«interface»
IKlarosLabeledObject<T>
«interface»
IKlarosNamedEntity<T>
«interface»
IKlarosRevision<KlarosTestCase, TestCase>
«interface»
IKlarosTestCase
+ getConfiguration(): KlarosConfiguration
+ getDetectedIssues(): List<KlarosIssue>
+ getCovers(): Set<KlarosRequirement>
+ getResults(): Set<KlarosTestCaseResult>
+ getTestCaseSteps(): List<KlarosTestCaseStep>
+ getShortname(): String
+ getDescription(): String
+ getPrecondition(): String
+ getPostcondition(): String
+ getNote(): String
+ getTestType(): TestAreatopic
+ getDesignTechnique(): TestDesignTechnique
+ getExecution(): TestExecutionMethod
+ getVariety(): TestVariety
+ getPriority(): TestPriority
+ getState(): String
+ getTeam(): String
+ getLevel(): TestLevel
+ getDocbase(): String
+ getDependency(): String
+ getEvaluation(): TestEvaluationMethod
+ getTraceability(): String
+ getEstimatedDuration(): Long
+ getExpectedResult(): String
+ getCategories(): Set<KlarosCategoryNode>
+ getExternalNames(): Set<String>
+ getAttachments(): Set<KlarosAttachment>

B.1.1.25.2. getAttachments()

```
public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this test case.

Parameters

`return` the attachments

B.1.1.25.3. getCategories()

```
public abstract Set<KlarosCategoryNode> getCategories();
```

Gets the categories this object belongs to. Each category node will belong to a different category tree.

Parameters

`return` the category nodes

B.1.1.25.4. getConfiguration()

```
public abstract KlarosConfiguration getConfiguration();
```

Get the project configuration this test case revision belongs to.

Parameters

`return` The related configuration.

B.1.1.25.5. getCovers()

```
public abstract Set<KlarosRequirement> getCovers();
```

Get covered requirements.

Parameters

`return` `Set` of requirements which are covered by this test case.

B.1.1.25.6. getDependency()

```
public abstract String getDependency();
```

The dependency of this test case.

Parameters

`return` The dependency.

B.1.1.25.7. getDescription()

```
public abstract String getDescription();
```

The description of this test case.

Parameters

`return` The description.

B.1.1.25.8. getDesignTechnique()

```
public abstract TestDesignTechnique getDesignTechnique();
```

The design technique of this test case.

Parameters

`return`	The type.

B.1.1.25.9. getDetectedIssues()

```
public abstract List<KlarosIssue> getDetectedIssues();
```

Get detected issues.

Parameters

`return`	`List` of issues which have been detected by this test case.

B.1.1.25.10. getDocbase()

```
public abstract String getDocbase();
```

The docbase of this test case.

Parameters

`return`	The docbase.

B.1.1.25.11. getEstimatedDuration()

```
public abstract Long getEstimatedDuration();
```

The estimated duration of this test case.

Parameters

`return`	The estimatedDuration.

B.1.1.25.12. getEvaluation()

```
public abstract TestEvaluationMethod getEvaluation();
```

The evaluation of this test case.

Parameters

`return`	The evaluation.

B.1.1.25.13. getExecution()

```
public abstract TestExecutionMethod getExecution();
```

The execution method of this test case.

Parameters

`return`	The execution method.

B.1.1.25.14. getExpectedResult()

```
public abstract String getExpectedResult();
```

The expected result of this test case.

Parameters

`return` The expected result.

B.1.1.25.15. getExternalNames()

```
public abstract Set<String> getExternalNames();
```

Gets the external test case names this test case is associated with. External test case names are used to match imported test case result to existing test cases.

Parameters

`return` the external names

B.1.1.25.16. getJobs()

```
public abstract List<KlarosJob> getJobs();
```

Gets the jobs associated with this test case.

Parameters

`return` the jobs

B.1.1.25.17. getLevel()

```
public abstract TestLevel getLevel();
```

The level of this test case.

Parameters

`return` The level.

B.1.1.25.18. getNote()

```
public abstract String getNote();
```

The note of this test case.

Parameters

`return` The note.

B.1.1.25.19. getPostcondition()

```
public abstract String getPostcondition();
```

The postcondition of this test case.

Parameters

`return` The postcondition.

B.1.1.25.20. getPrecondition()

```
public abstract String getPrecondition();
```

The precondition of this test case.

Parameters

`return` The precondition.

B.1.1.25.21. getPriority()

```
public abstract TestPriority getPriority();
```

The priority of this test case.

Parameters

`return` The priority.

B.1.1.25.22. getResults()

```
public abstract Set<KlarosTestCaseResult> getResults();
```

Get test case results.

Parameters

`return` `Set` of results of executions of this test case.

B.1.1.25.23. getShortname()

```
public abstract String getShortname();
```

The short name (title) of this test case.

Parameters

`return` The short name.

B.1.1.25.24. getState()

```
public abstract String getState();
```

The type of this test case.

Parameters

`return` The type.

B.1.1.25.25. getTeam()

```
public abstract String getTeam();
```

The team responsible for this test case.

Parameters

`return` The team.

B.1.1.25.26. getTestCaseSteps()

```
public abstract List<KlarosTestCaseStep> getTestCaseSteps();
```

Get test case steps.

Parameters

return	list of steps of this test case.

B.1.1.25.27. getTestType()

```
public abstract TestAreatopic getTestType();
```

The test type of this test case.

Parameters

return	The test type.

B.1.1.25.28. getTraceability()

```
public abstract String getTraceability();
```

The traceability of this test case.

Parameters

return	The traceability.

B.1.1.25.29. getVariety()

```
public abstract TestVariety getVariety();
```

The variety of this test case.

Parameters

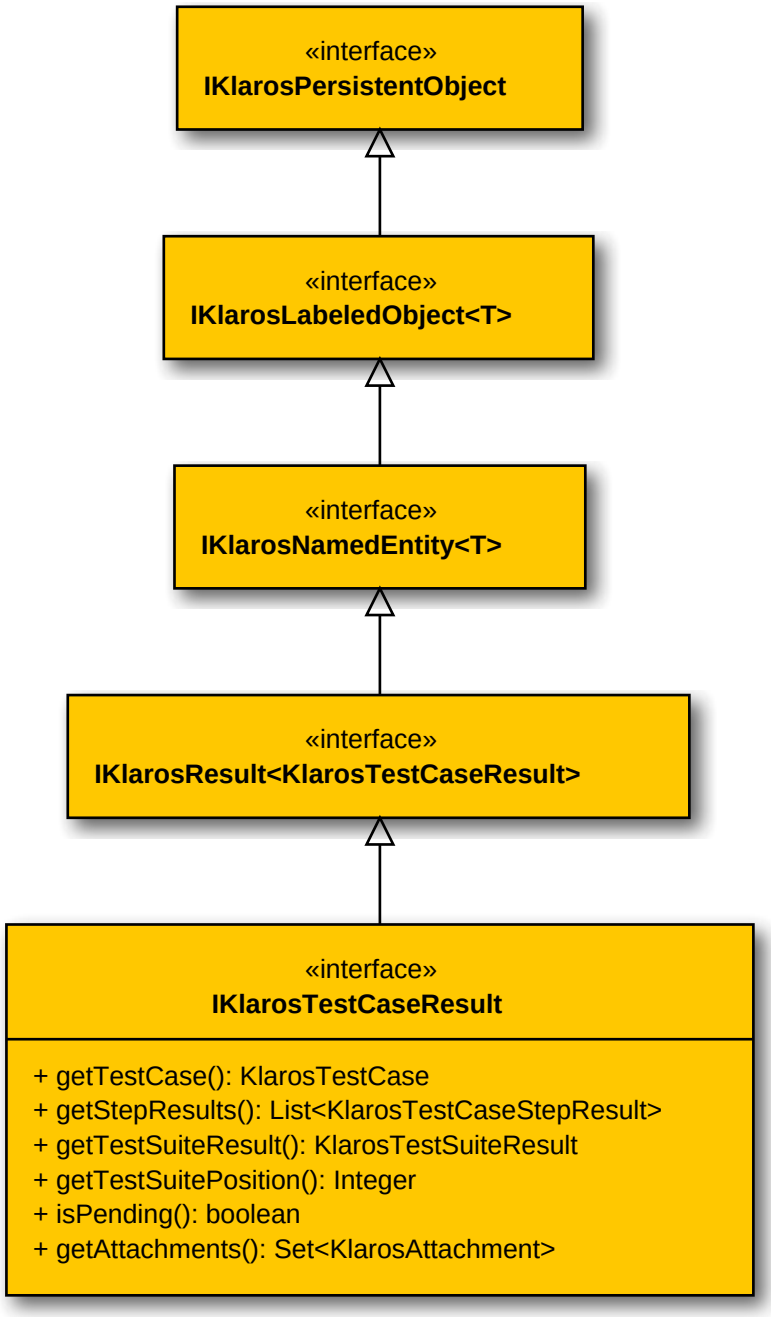
return	The variety.

B.1.1.26. IKlarosTestCaseResult

This interface provides access to data of a test case result.

B.1.1.26.1. Synopsis

```
public interface IKlarosTestCaseResult extends IKlarosResult<KlarosTestCaseResult> {

  // Public Methods
  public abstract Set<KlarosAttachment> getAttachments();
  public abstract List<KlarosTestCaseStepResult> getStepResults();
  public abstract KlarosTestCase getTestCase();
  public abstract Integer getTestSuitePosition();
  public abstract KlarosTestSuiteResult getTestSuiteResult();
  public abstract boolean isPending();
}
```



B.1.1.26.2. getAttachments()

```
public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this result.

Parameters

`return` the attachments

B.1.1.26.3. getStepResults()

```
public abstract List<KlarosTestCaseStepResult> getStepResults();
```

Get the associated step results.

Parameters

`return` `List` of test case step results.

B.1.1.26.4. getTestCase()

```
public abstract KlarosTestCase getTestCase();
```

Get the associated test case.

Parameters

`return` The test case that has been executed to get this result.

B.1.1.26.5. getTestSuitePosition()

```
public abstract Integer getTestSuitePosition();
```

Return the position in the test suite result this test case result belongs to or null if this test case has not been executed by executing a test suite.

Parameters

`return` the position of the test case in the test suite, or null

B.1.1.26.6. getTestSuiteResult()

```
public abstract KlarosTestSuiteResult getTestSuiteResult();
```

Return the test suite result this test case result belongs to or null if this test case has not been executed by executing a test suite.

Parameters

`return` the test suite result, or null

B.1.1.26.7. isPending()

```
public abstract boolean isPending();
```

Checks if the result is pending. This means that it has not yet been completely executed.

Parameters

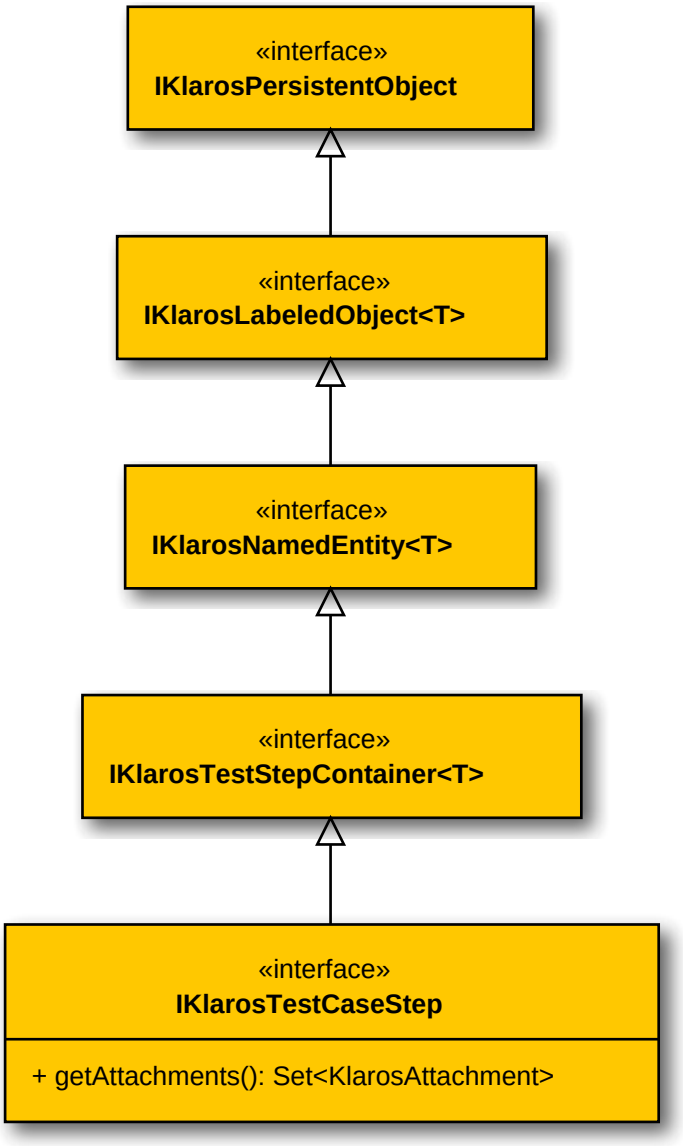
`return` true, if the result is pending

B.1.1.27. IKlarosTestCaseStep

This interface provides access to a test case step.

B.1.1.27.1. Synopsis

```
public interface IKlarosTestCaseStep<T> extends IKlarosTestStepContainer<T> {

  // Public Methods
  public abstract Set<KlarosAttachment> getAttachments();
}
```



B.1.1.27.2. getAttachments()

```
public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this test case.

Parameters

`return` the attachments

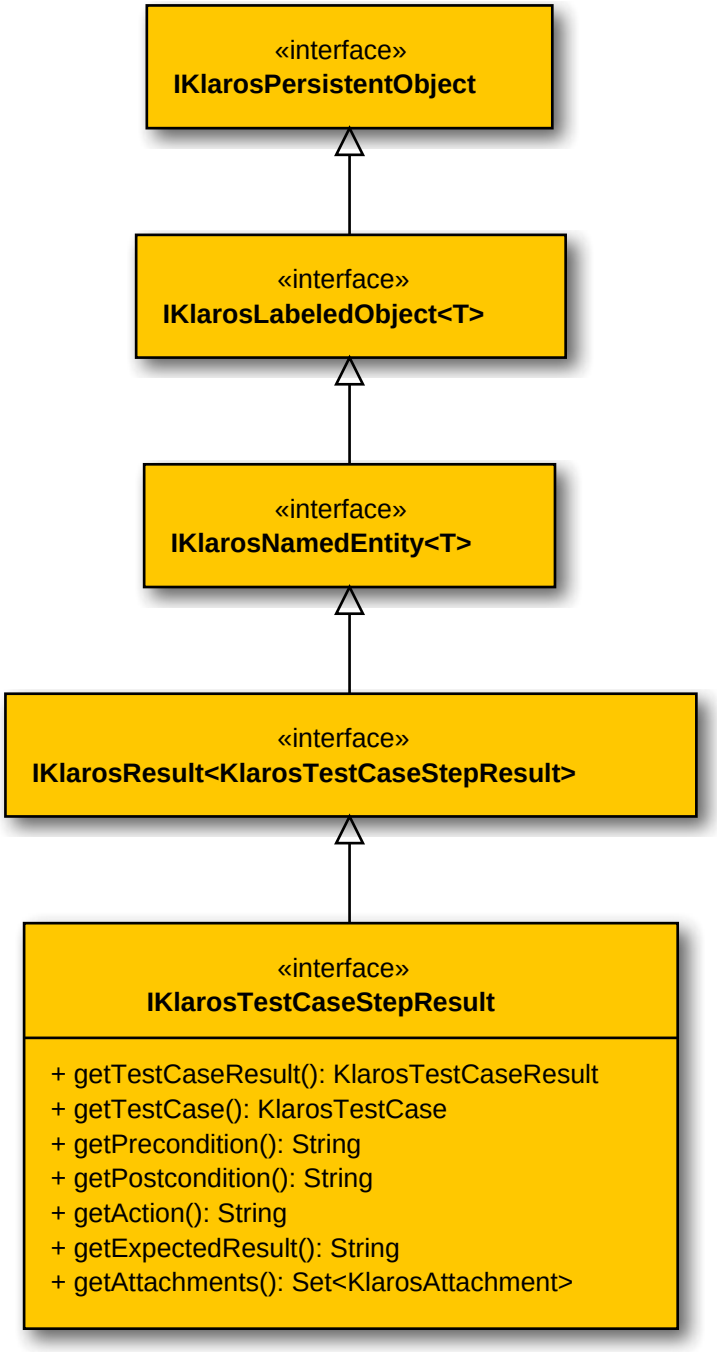
B.1.1.28. IKlarosTestCaseStepResult

This interface provides access to data of a test case step result.

B.1.1.28.1. Synopsis

```
public interface IKlarosTestCaseStepResult
 extends IKlarosResult<KlarosTestCaseStepResult> {
```

```
  // Public Methods
  public abstract String getAction();
  public abstract Set<KlarosAttachment> getAttachments();
  public abstract String getExpectedResult();
  public abstract String getPostcondition();
  public abstract String getPrecondition();
  public abstract KlarosTestCase getTestCase();
  public abstract KlarosTestCaseResult getTestCaseResult();
}
```



B.1.1.28.2. getAction()

```
public abstract String getAction();
```

Get the action.

Parameters

`return` The action of this test case step.

B.1.1.28.3. getAttachments()

```
public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this test case.

Parameters

`return` the attachments

B.1.1.28.4. getExpectedResult()

```
public abstract String getExpectedResult();
```

Get the expected result.

Parameters

`return` The expected result of this test case step.

B.1.1.28.5. getPostcondition()

```
public abstract String getPostcondition();
```

Get the postcondition.

Parameters

`return` The postcondition of this test case step.

B.1.1.28.6. getPrecondition()

```
public abstract String getPrecondition();
```

Get the precondition.

Parameters

`return` The precondition of this test case step.

B.1.1.28.7. getTestCase()

```
public abstract KlarosTestCase getTestCase();
```

Get test case.

Parameters

`return` The test case that has been executed to get this result.

B.1.1.28.8. getTestCaseResult()

```
public abstract KlarosTestCaseResult getTestCaseResult();
```

Get test case result.

Parameters

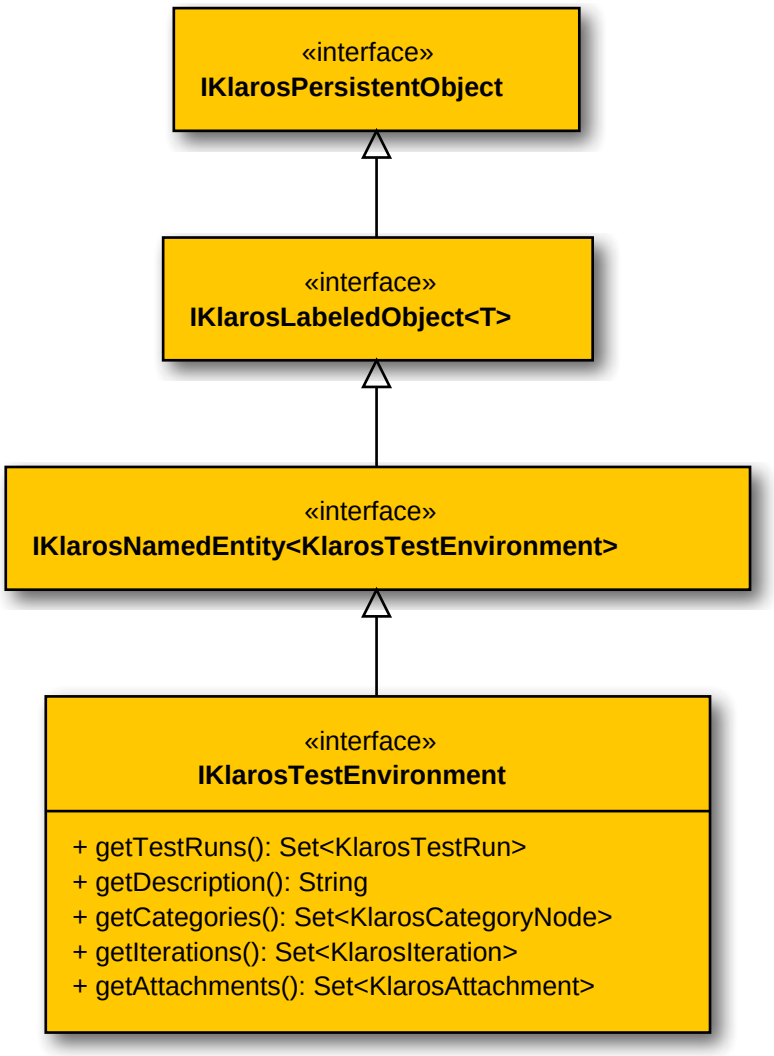
`return`	The test case result that has been executed to get this result.

B.1.1.29. IKlarosTestEnvironment

This interface provides access to data of a test environment.

B.1.1.29.1. Synopsis

```
public interface IKlarosTestEnvironment
 extends IKlarosNamedEntity<KlarosTestEnvironment> {

  // Public Methods
  public abstract Set<KlarosAttachment> getAttachments();
  public abstract Set<KlarosCategoryNode> getCategories();
  public abstract String getDescription();
  public abstract Set<KlarosIteration> getIterations();
  public abstract Set<KlarosTestRun> getTestRuns();
}
```



B.1.1.29.2. getAttachments()

```
public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this test environment.

Parameters

`return`	the attachments

B.1.1.29.3. getCategories()

```
public abstract Set<KlarosCategoryNode> getCategories();
```

Gets the categories this test environment belongs to. Each category node will belong to a different category tree.

Parameters

`return`	the category nodes

B.1.1.29.4. getIterations()

```
public abstract Set<KlarosIteration> getIterations();
```

Get the iterations this test environment is assigned to.

Parameters

`return`	`Set` of iterations this test environment is assigned to.

B.1.1.29.5. getTestRuns()

```
public abstract Set<KlarosTestRun> getTestRuns();
```

Gets the test runs associated with this test environment.

Parameters

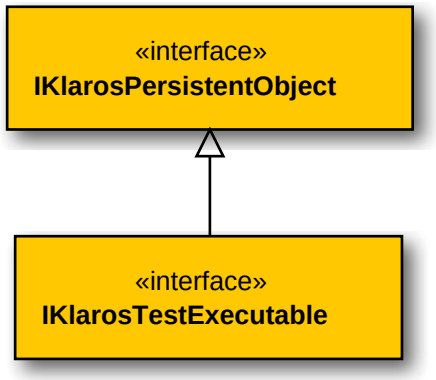
`return`	`Collection` of test runs executed in the environment.

B.1.1.30. IKlarosTestExecutable

This class provides access to a test executable. This may wrap a test case or a test suite.

B.1.1.30.1. Synopsis

```
public interface IKlarosTestExecutable extends IKlarosPersistentObject {
}
```

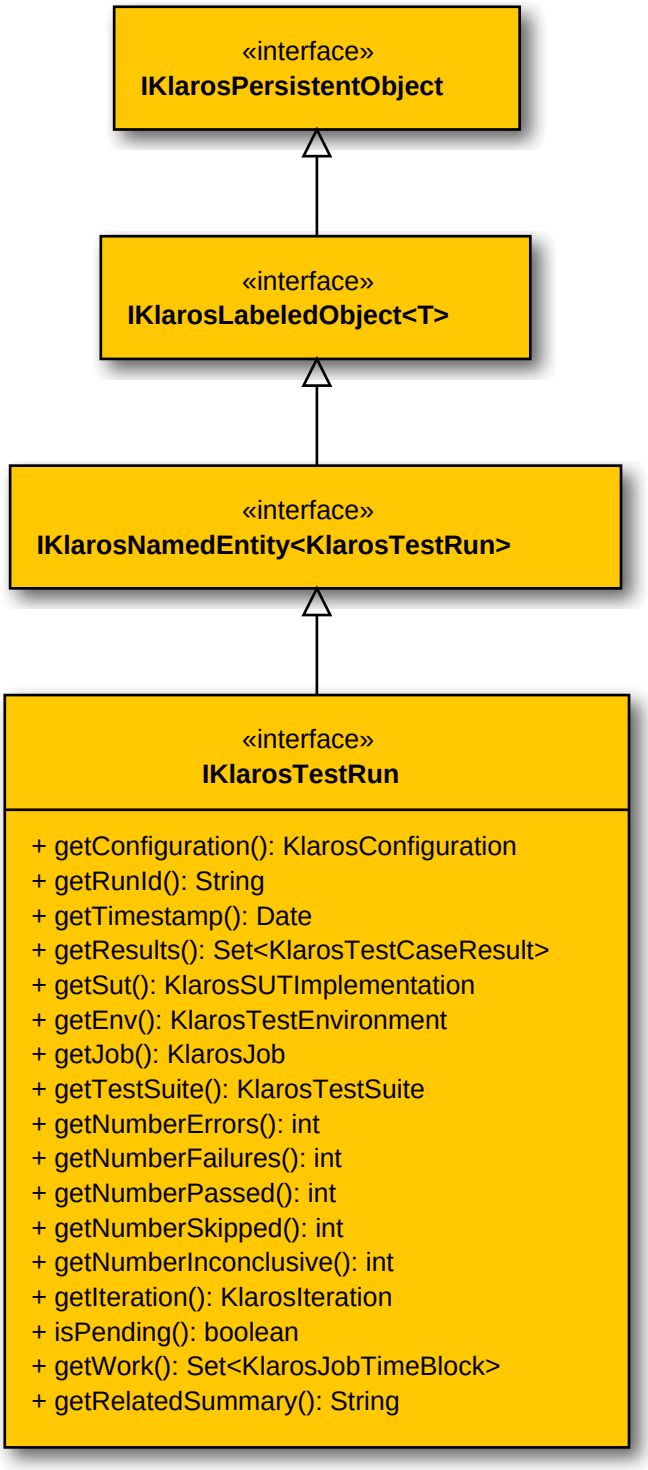


B.1.1.31. IKlarosTestRun

This interface provides access to data of a test run.

B.1.1.31.1. Synopsis

```
public interface IKlarosTestRun extends IKlarosNamedEntity<KlarosTestRun> {

  // Public Methods
  public abstract KlarosConfiguration getConfiguration();
  public abstract KlarosTestEnvironment getEnv();
  public abstract KlarosIteration getIteration();
  public abstract KlarosJob getJob();
  public abstract int getNumberErrors();
  public abstract int getNumberFailures();
  public abstract int getNumberInconclusive();
  public abstract int getNumberPassed();
  public abstract int getNumberSkipped();
  public abstract String getRelatedSummary();
  public abstract Set<KlarosTestCaseResult> getResults();
  public abstract String getRunId();
  public abstract KlarosSUTImplementation getSut();
  public abstract KlarosTestSuite getTestSuite();
  public abstract Date getTimestamp();
  public abstract Set<KlarosJobTimeBlock> getWork();
  public abstract boolean isPending();
}
```



B.1.1.31.2. getConfiguration()

```
public abstract KlarosConfiguration getConfiguration();
```

Get configuration.

Parameters

`return`	The related configuration

B.1.1.31.3. getEnv()

```
public abstract KlarosTestEnvironment getEnv();
```

Get test environment.

Parameters

return	The test environment in which the test cases have been executed.

B.1.1.31.4. getIteration()

```
public abstract KlarosIteration getIteration();
```

Get the iteration in which the test run has been created, if available.

Parameters

return	The iteration in which the test run has been created

B.1.1.31.5. getJob()

```
public abstract KlarosJob getJob();
```

Get the job that initiated the test run, if available.

Parameters

return	The job that initiated the test run

B.1.1.31.6. getNumberErrors()

```
public abstract int getNumberErrors();
```

Get number of test cases with errors of this test run. It is assumed, that error results have a property 'type' with the value 'E'.

Parameters

return	The number of error test cases

B.1.1.31.7. getNumberFailures()

```
public abstract int getNumberFailures();
```

Get number of failed test cases of this test run. It is assumed, that failed results have a property 'type' with the value 'F'.

Parameters

return	The number of failed test cases

B.1.1.31.8. getNumberInconclusive()

```
public abstract int getNumberInconclusive();
```

Get number of inconclusive test cases of this test run. It is assumed, that inconclusive results have a property 'type' with the value 'I'.

Parameters

`return` The number of skipped test cases

B.1.1.31.9. getNumberPassed()

```
public abstract int getNumberPassed();
```

Get number of passed test cases of this test run. It is assumed, that passed results have a property 'testCasePassed' with value 'true'.

Parameters

`return` The number of passed test cases

B.1.1.31.10. getNumberSkipped()

```
public abstract int getNumberSkipped();
```

Get number of skipped test cases of this test run. It is assumed, that skipped results have a property 'type' with the value 'S'.

Parameters

`return` The number of skipped test cases

B.1.1.31.11. getRelatedSummary()

```
public abstract String getRelatedSummary();
```

Gets the summary of the job, test suite or test case related to this test run.

Parameters

`return` the related summary

B.1.1.31.12. getResults()

```
public abstract Set<KlarosTestCaseResult> getResults();
```

Get results.

Parameters

`return` `Set` of results of test case executions.

B.1.1.31.13. getRunId()

```
public abstract String getRunId();
```

Get id of test run.

Parameters

`return` The id of the test run

B.1.1.31.14. getSut()

```
public abstract KlarosSUTImplementation getSut();
```

Get the tested system version.

Parameters

`return` The SUT version which has been tested

B.1.1.31.15. getTestSuite()

```
public abstract KlarosTestSuite getTestSuite();
```

Get the test suite executed with this test run, if available.

Parameters

`return` The test suite executed with this test run

B.1.1.31.16. getTimestamp()

```
public abstract Date getTimestamp();
```

Get timestamp.

Parameters

`return` The time the test run has been executed as `Date` object

B.1.1.31.17. getWork()

```
public abstract Set<KlarosJobTimeBlock> getWork();
```

Get the job time blocks of this test run.

Parameters

`return` `Set` of job time blocks.

B.1.1.31.18. isPending()

```
public abstract boolean isPending();
```

Checks if the test run is pending. This means that it has not yet been completely executed.

Parameters

`return` true, if the test run is pending

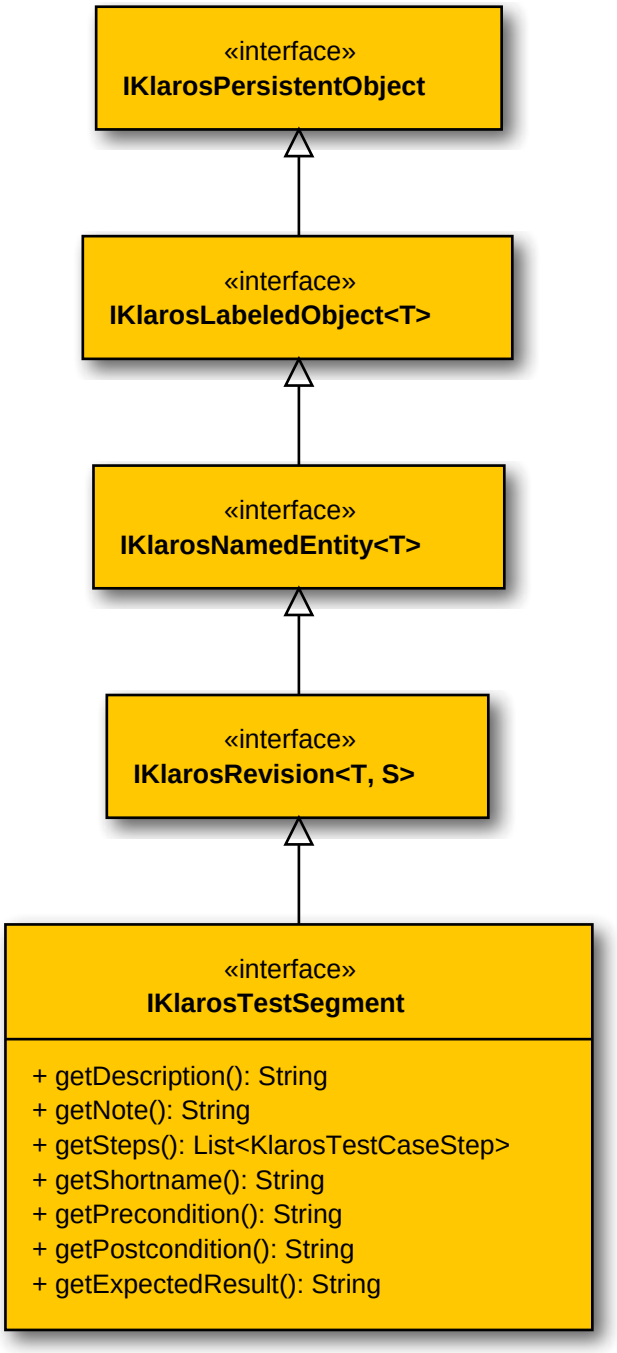
B.1.1.32. IKlarosTestSegment

This interface provides access to a test segment.

B.1.1.32.1. Synopsis

```
public interface IKlarosTestSegment<T,S extends Revision<S>>
 extends IKlarosRevision<T, S> {

  // Public Methods
  public abstract String getDescription();
  public abstract String getExpectedResult();
```

```
  public abstract String getNote();
  public abstract String getPostcondition();
  public abstract String getPrecondition();
  public abstract String getShortname();
  public abstract List<KlarosTestCaseStep> getSteps();
}
```



B.1.1.32.2. getDescription()

```
  public abstract String getDescription();
```

Get the description.

Parameters

`return` The description of this test segment.

B.1.1.32.3. getExpectedResult()

```
public abstract String getExpectedResult();
```

Get the expected result.

Parameters

`return` The expected result of this test case step.

B.1.1.32.4. getNote()

```
public abstract String getNote();
```

Get the note.

Parameters

`return` The note of this test segment.

B.1.1.32.5. getPostcondition()

```
public abstract String getPostcondition();
```

Get the postcondition.

Parameters

`return` The postcondition of this test case step.

B.1.1.32.6. getPrecondition()

```
public abstract String getPrecondition();
```

Get the precondition.

Parameters

`return` The precondition of this test case step.

B.1.1.32.7. getShortname()

```
public abstract String getShortname();
```

Get the shortname.

Parameters

`return` The shortname of this test case segment.

B.1.1.32.8. getSteps()

```
public abstract List<KlarosTestCaseStep> getSteps();
```

Get the test case steps.

Parameters

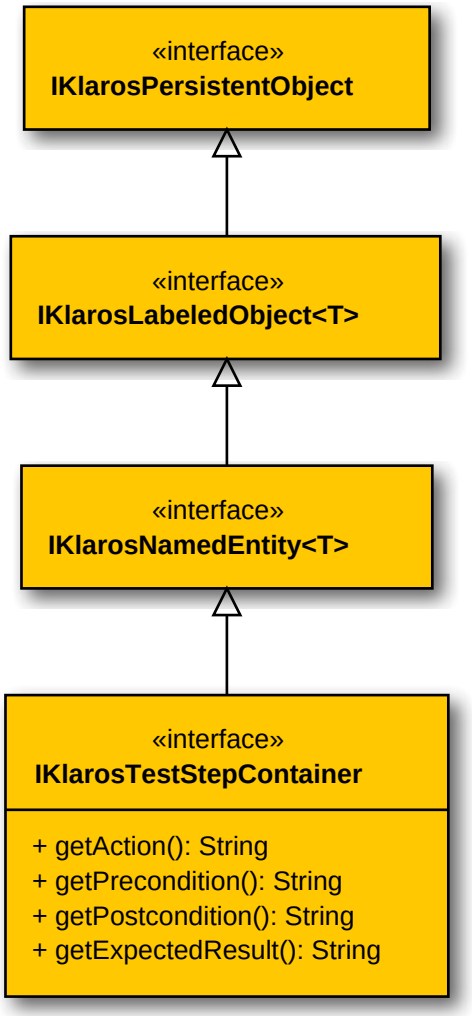
`return` `list` of steps of this test segment.

B.1.1.33. IKlarosTestStepContainer

This interface provides access to a test step containers.

B.1.1.33.1. Synopsis

```
public interface IKlarosTestStepContainer<T> extends IKlarosNamedEntity<T> {

  // Public Methods
  public abstract String getAction();
  public abstract String getExpectedResult();
  public abstract String getPostcondition();
  public abstract String getPrecondition();
}
```



B.1.1.33.2. getAction()

```
public abstract String getAction();
```

Get the action.

Parameters

`return` The action of this test case step.

B.1.1.33.3. getExpectedResult()

```
public abstract String getExpectedResult();
```

Get the expected result.

Parameters

`return` The expected result of this test case step.

B.1.1.33.4. getPostcondition()

```
public abstract String getPostcondition();
```

Get the postcondition.

Parameters

`return` The postcondition of this test case step.

B.1.1.33.5. getPrecondition()

```
public abstract String getPrecondition();
```

Get the precondition.

Parameters

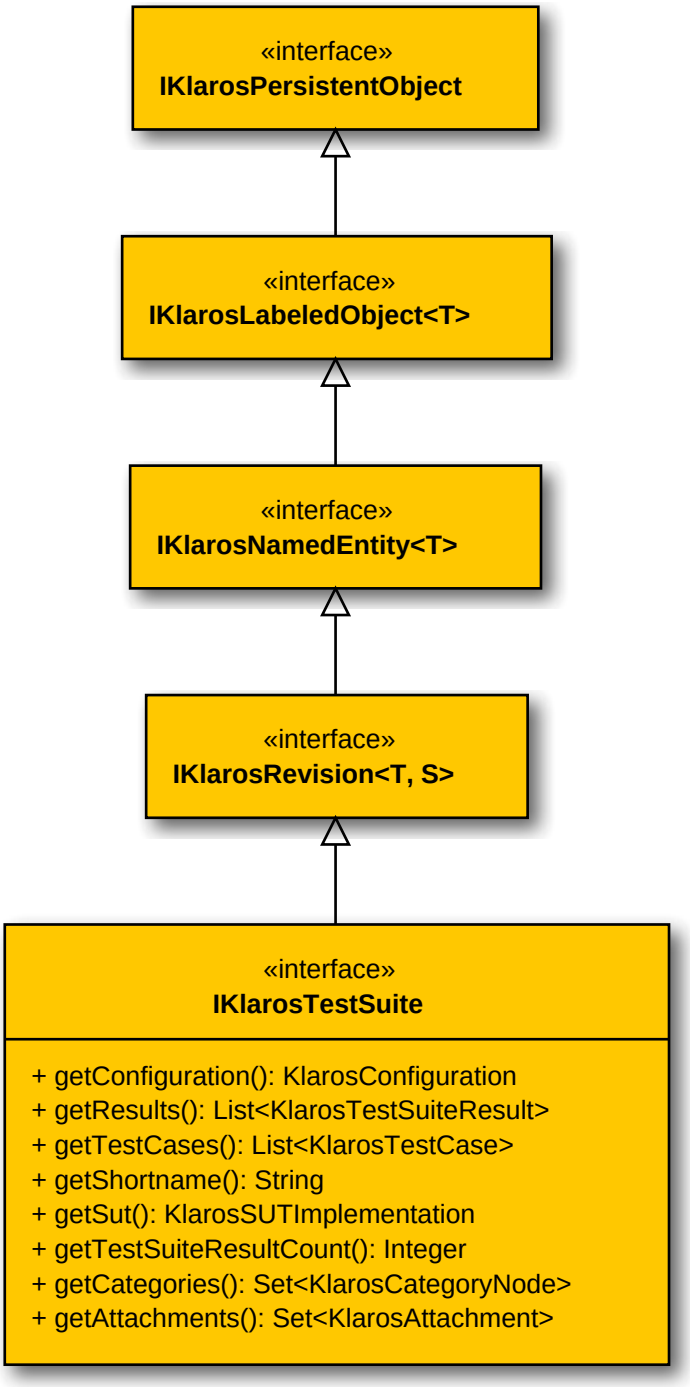
`return` The precondition of this test case step.

B.1.1.34. IKlarosTestSuite

This interface provides access to data of a test suite.

B.1.1.34.1. Synopsis

```
public interface IKlarosTestSuite<T,S extends Revision<?>> extends IKlarosRevision<T,
 S> {

  // Public Methods
  public abstract Set<KlarosAttachment> getAttachments();
  public abstract Set<KlarosCategoryNode> getCategories();
  public abstract KlarosConfiguration getConfiguration();
  public abstract List<KlarosTestSuiteResult> getResults();
  public abstract String getShortname();
  public abstract KlarosSUTImplementation getSut();
  public abstract List<KlarosTestCase> getTestCases();
  public abstract Integer getTestSuiteResultCount();
}
```



B.1.1.34.2. getAttachments()

```
public abstract Set<KlarosAttachment> getAttachments();
```

Gets the attachments associated with this test suite.

Parameters

`return` the attachments

B.1.1.34.3. getCategories()

```
public abstract Set<KlarosCategoryNode> getCategories();
```

Gets the categories this object belongs to. Each category node will belong to a different category tree.

Parameters

`return` the category nodes

B.1.1.34.4. getConfiguration()

```
public abstract KlarosConfiguration getConfiguration();
```

Get configuration.

Parameters

`return` The related configuration.

B.1.1.34.5. getResults()

```
public abstract List<KlarosTestSuiteResult> getResults();
```

Get test suite results.

Parameters

`return` `Set` of results of executions of this test suite.

B.1.1.34.6. getShortname()

```
public abstract String getShortname();
```

Get the short name of the test suite.

Parameters

`return` `String` of the description of the test suite.

B.1.1.34.7. getSut()

```
public abstract KlarosSUTImplementation getSut();
```

Get the description of the test suite.

Parameters

`return` `String` of the description of the test suite.

B.1.1.34.8. getTestCases()

```
public abstract List<KlarosTestCase> getTestCases();
```

Get the executables of this test suite.

Parameters

`return` `Set` of test cases of this test suite.

B.1.1.34.9. getTestSuiteResultCount()

```
public abstract Integer getTestSuiteResultCount();
```

Return the number of test suite results in this test suite.

Parameters

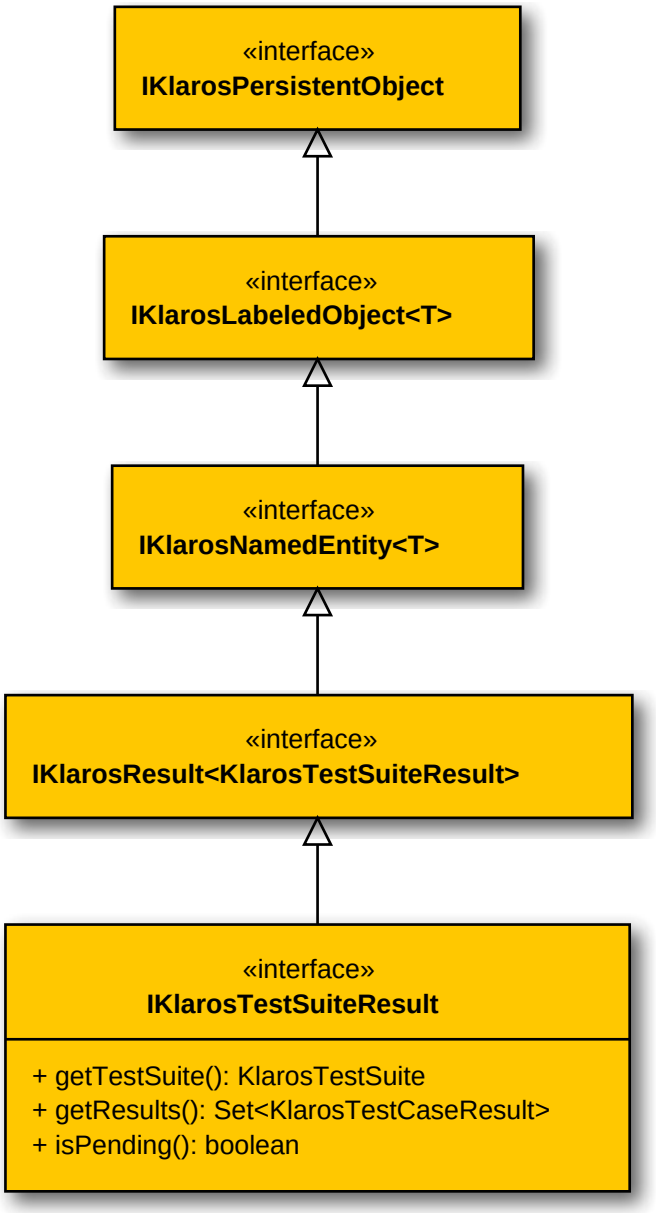
`return` The number of test suite results in this test suite.

B.1.1.35. IKlarosTestSuiteResult

This interface provides access to data of a test suite result.

B.1.1.35.1. Synopsis

```
public interface IKlarosTestSuiteResult
 extends IKlarosResult<KlarosTestSuiteResult> {

  // Public Methods
  public abstract Set<KlarosTestCaseResult> getResults();
  public abstract KlarosTestSuite getTestSuite();
  public abstract boolean isPending();
}
```



B.1.1.35.2. getResults()

```
public abstract Set<KlarosTestCaseResult> getResults();
```

Gets the test case results for this test suite result. The list is sorted by the position in which the test cases appear in the test suite.

Parameters

`return` the results

B.1.1.35.3. getTestSuite()

```
public abstract KlarosTestSuite getTestSuite();
```

Get the related test suite.

Parameters

`return` The test suite that has been executed to get this result.

B.1.1.35.4. isPending()

```
public abstract boolean isPending();
```

Checks if the result is pending. This means that it has not yet been completely executed.

Parameters

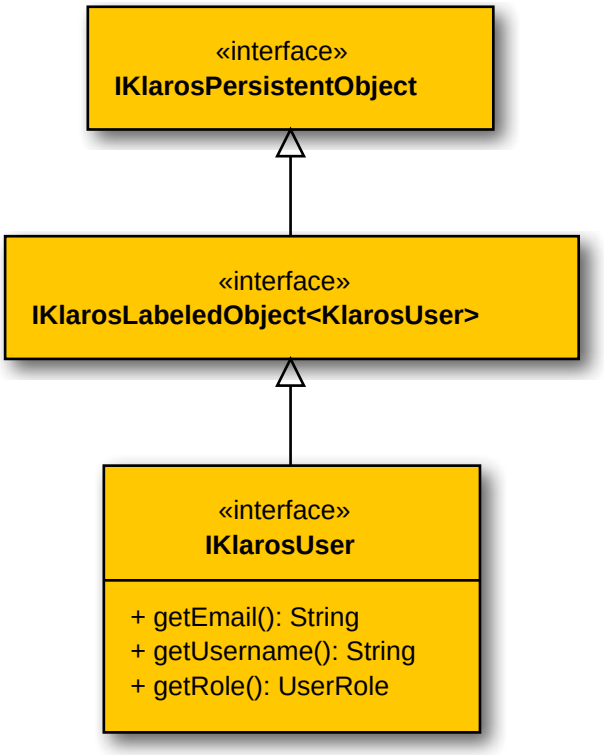
`return` true, if the result is pending

B.1.1.36. IKlarosUser

The user object.

B.1.1.36.1. Synopsis

```
public interface IKlarosUser extends IKlarosLabeledObject<KlarosUser> {

  // Public Methods
  public abstract String getEmail();
  public abstract UserRole getRole();
  public abstract String getUsername();
}
```



B.1.1.36.2. getEmail()

```
public abstract String getEmail();
```

The email address of this user.

Parameters

`return` the email address

B.1.1.36.3. getRole()

```
public abstract UserRole getRole();
```

The role name of this user.

Parameters

`return` the role name

B.1.1.36.4. getUsername()

```
public abstract String getUsername();
```

The username of this user as used when logging in.

Parameters

`return` the username

B.1.1.37. KlarosAttachment

This class provides access to binary attachment.

B.1.1.37.1. Synopsis

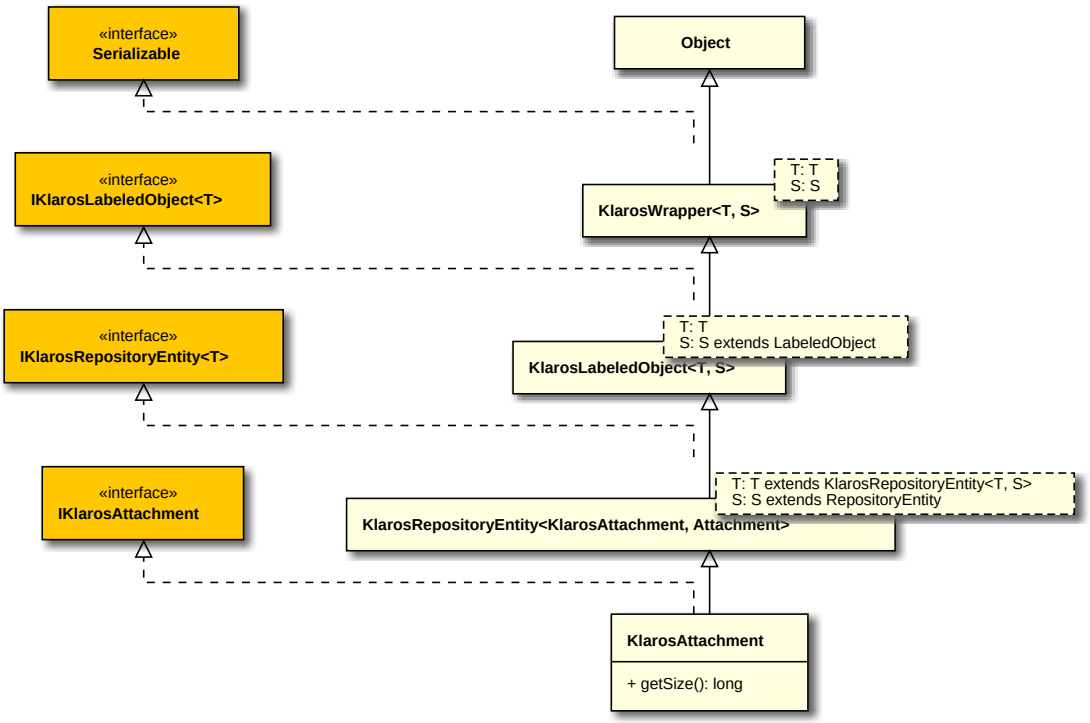
```
public final class KlarosAttachment extends
                                KlarosRepositoryEntity<KlarosAttachment, Attachment>

   implements IKlarosAttachment {

  // Public Methods
  public long getSize();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosRepositoryEntity: `getMimeType` , `getName` , `getUuid` , `getVersion`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.38. KlarosCategoryNode

A generic category node.

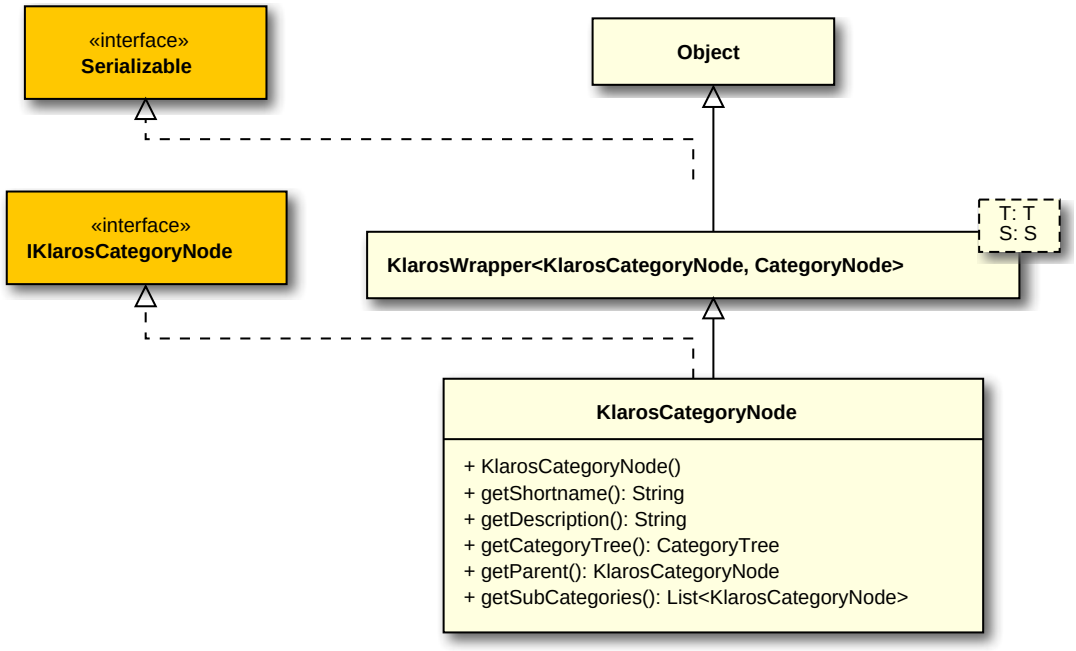
B.1.1.38.1. Synopsis

```
public class KlarosCategoryNode extends KlarosWrapper<KlarosCategoryNode,
 CategoryNode>
```

```
  implements IKlarosCategoryNode {

 // Public Constructors
 public KlarosCategoryNode();

 // Public Methods
 public CategoryTree getCategoryTree();
 public String getDescription();
 public KlarosCategoryNode getParent();
 public String getShortname();
 public List<KlarosCategoryNode> getSubCategories();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals`, `getWrapped`, `hashCode`

Methods inherited from java.lang.Object: `getClass`, `notify`, `notifyAll`, `toString`, `wait`



B.1.1.38.2. KlarosCategoryNode()

```
public KlarosCategoryNode();
```

Constructor.

B.1.1.39. KlarosCategoryTree

A category tree.

B.1.1.39.1. Synopsis

```
public class KlarosCategoryTree extends
                                KlarosNamedEntity<KlarosCategoryTree, CategoryTree>
```

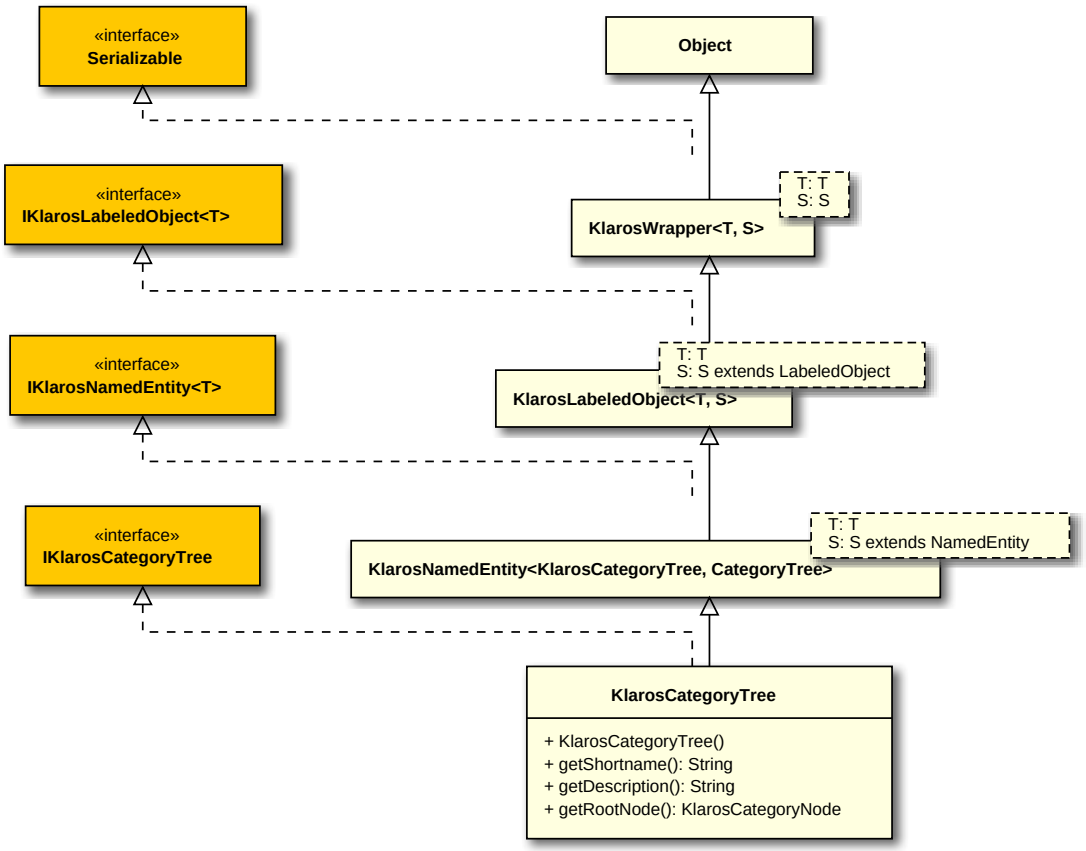
```
 implements IKlarosCategoryTree {

  // Public Constructors
  public KlarosCategoryTree();

  // Public Methods
  public String getDescription();
  public KlarosCategoryNode getRootNode();
  public String getShortname();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `getPropertyValue` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.39.2. KlarosCategoryTree()

```
public KlarosCategoryTree();
```

Serialization constructor.

B.1.1.40. KlarosConfiguration

This class provides access to the information stored for project's configuration.

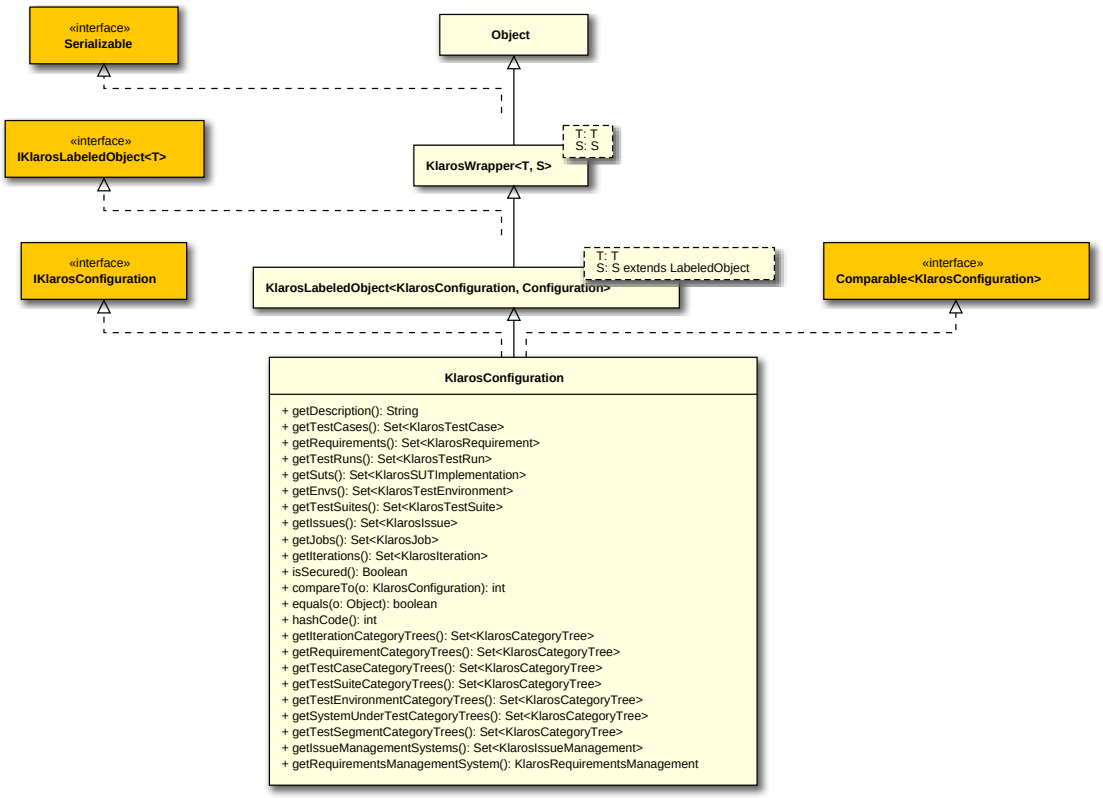
B.1.1.40.1. Synopsis

```
public class KlarosConfiguration extends
                                KlarosLabeledObject<KlarosConfiguration,
 Configuration>

  implements IKlarosConfiguration,
          Comparable<KlarosConfiguration> {

  // Public Methods
  public int compareTo(KlarosConfiguration o);
  public boolean equals(Object o);
  public String getDescription();
  public Set<KlarosTestEnvironment> getEnvs();
  public Set<KlarosIssueManagement> getIssueManagementSystems();
  public Set<KlarosIssue> getIssues();
  public Set<KlarosCategoryTree> getIterationCategoryTrees();
  public Set<KlarosIteration> getIterations();
  public Set<KlarosJob> getJobs();
  public Set<KlarosCategoryTree> getRequirementCategoryTrees();
  public Set<KlarosRequirement> getRequirements();
  public KlarosRequirementsManagement getRequirementsManagementSystem();
  public Set<KlarosSUTImplementation> getSuts();
  public Set<KlarosCategoryTree> getSystemUnderTestCategoryTrees();
  public Set<KlarosCategoryTree> getTestCaseCategoryTrees();
  public Set<KlarosTestCase> getTestCases();
  public Set<KlarosCategoryTree> getTestEnvironmentCategoryTrees();
  public Set<KlarosTestRun> getTestRuns();
  public Set<KlarosCategoryTree> getTestSegmentCategoryTrees();
  public Set<KlarosCategoryTree> getTestSuiteCategoryTrees();
  public Set<KlarosTestSuite> getTestSuites();
  public int hashCode();
  public Boolean isSecured();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.41. KlarosEnumValue

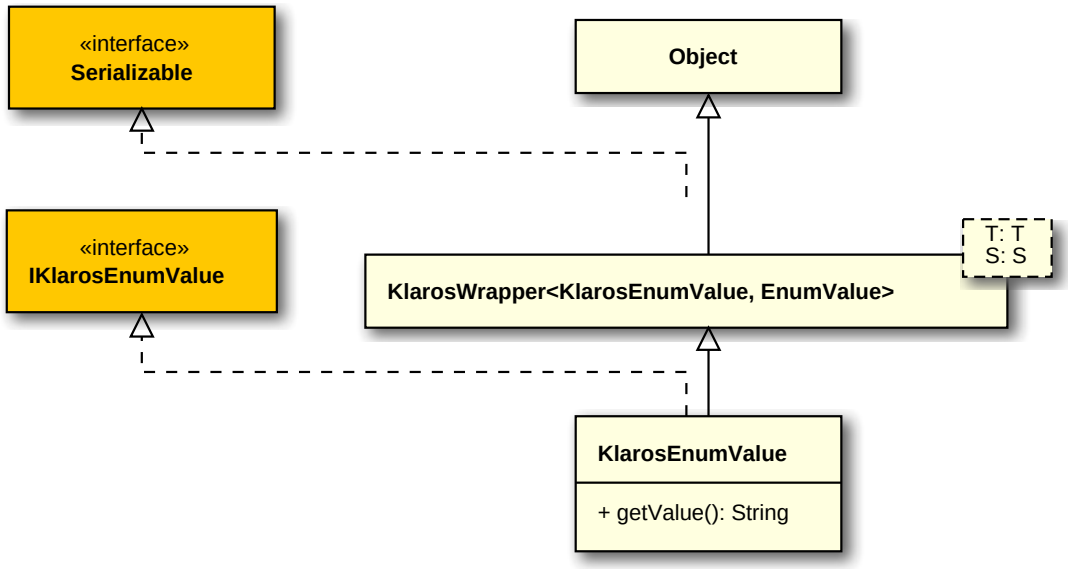
A Custom property enumeration value.

B.1.1.41.1. Synopsis

```
public final class KlarosEnumValue extends KlarosWrapper<KlarosEnumValue, EnumValue>
   implements IKlarosEnumValue {

  // Public Methods
  public String getValue();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.42. KlarosExternalServer

This class encapsulates the (dynamic) properties of a klaros object.

B.1.1.42.1. Synopsis

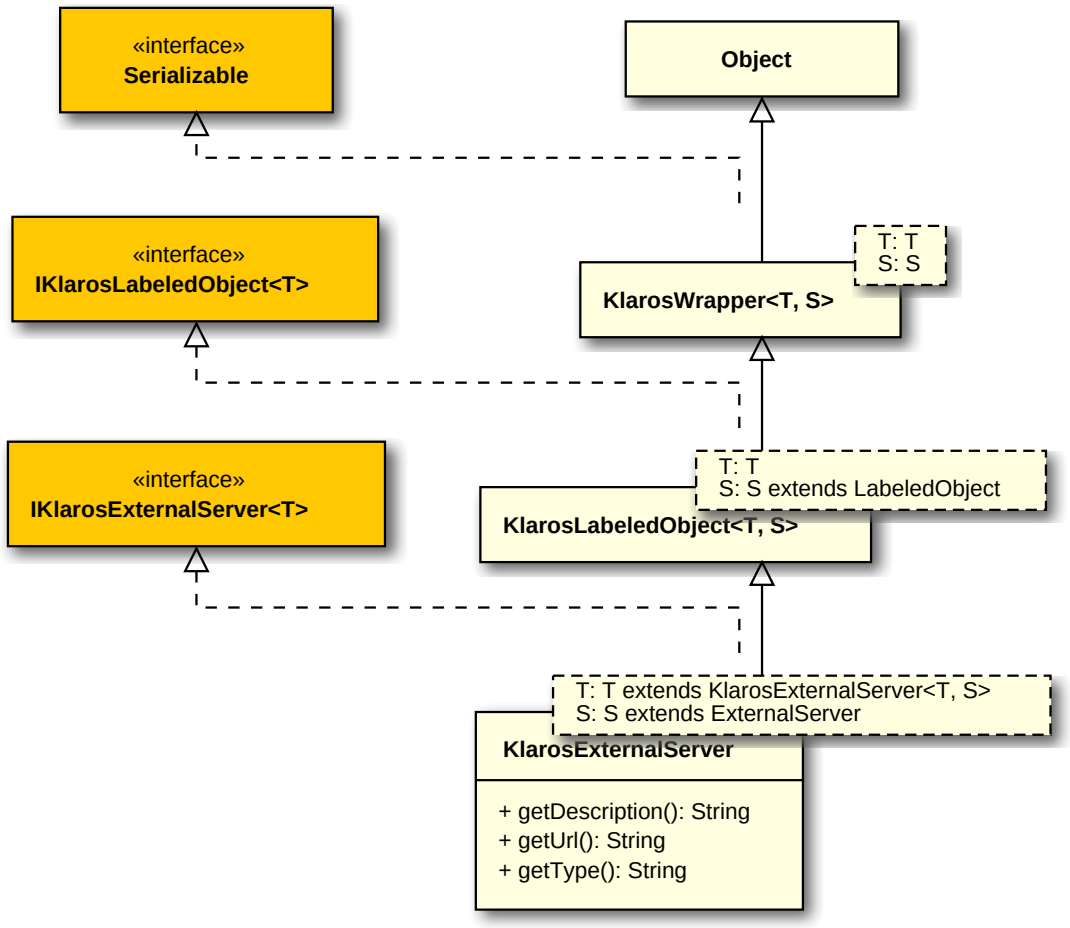
```
public abstract class KlarosExternalServer<T extends KlarosExternalServer<T, S>,S
 extends ExternalServer> extends
                                KlarosLabeledObject<T, S>

  implements IKlarosExternalServer<T> {

 // Public Methods
 public String getDescription();
 public String getType();
 public String getUrl();
}
```

Direct known subclasses: `de.verit.klaros.core.model.KlarosIssueManagement` , `de.verit.klaros.core.model.KlarosRequirementsManagement`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.43. KlarosIssue

This class provides access to the information stored for detected issues.

B.1.1.43.1. Synopsis

```
public final class KlarosIssue extends
                                KlarosNamedEntity<KlarosIssue, Issue>

  implements IKlarosIssue {

 // Public Methods
 public KlarosSUTImplementation getAcceptedIn();
 public Date getCreationDate();
 public String getDescription();
 public String getExternalId();
 public KlarosIssueManagement getIssueManagement();
 public Date getLastSynched();
 public String getOwner();
 public String getPriority();
 public String getRemoteBrowseUrl();
 public String getReporter();
 public String getResolution();
 public String getState();
```

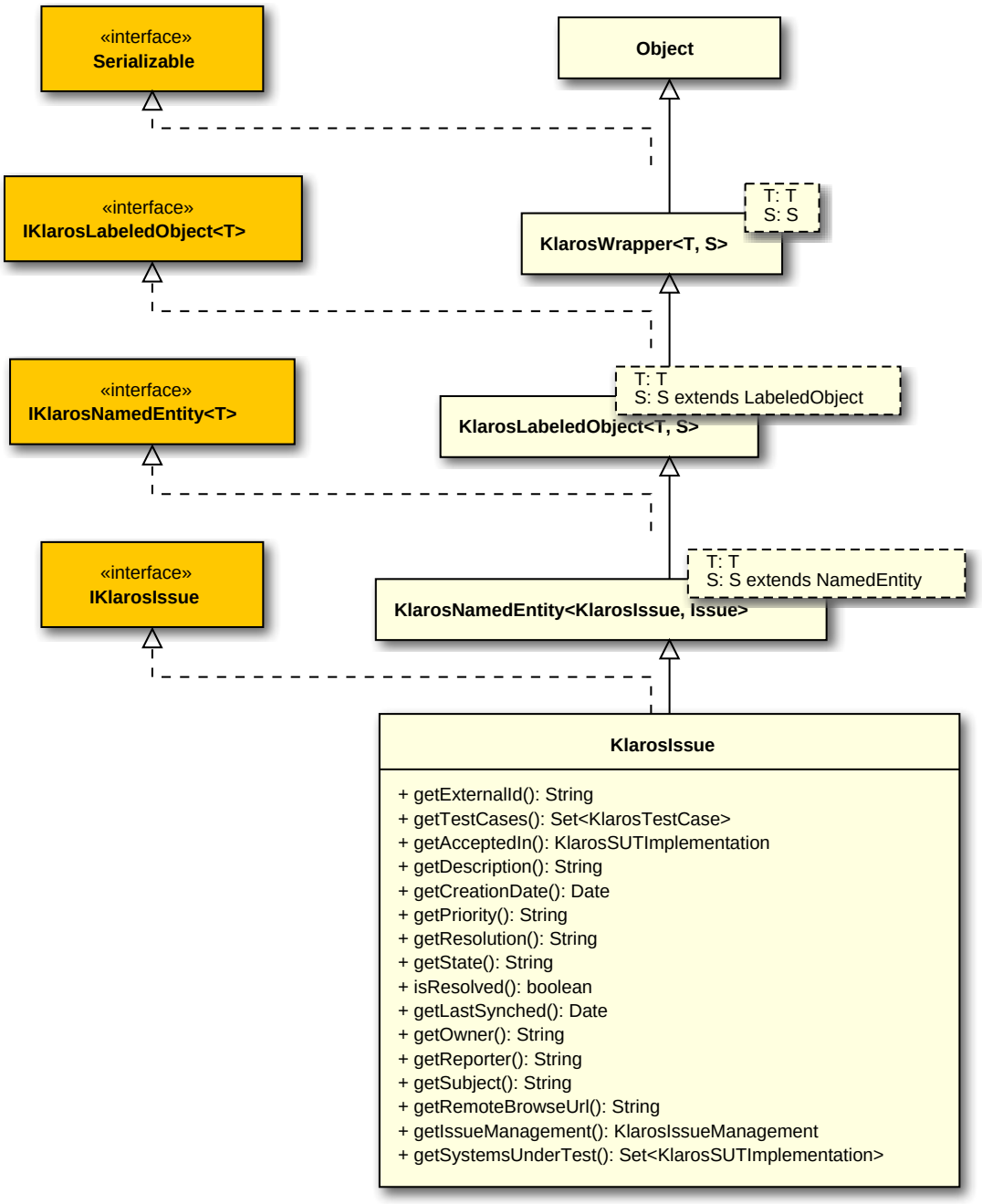
```
  public String getSubject();
  public Set<KlarosSUTImplementation> getSystemsUnderTest();
  public Set<KlarosTestCase> getTestCases();
  public boolean isResolved();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: getAssignee , getProperties , getPropertyValue , isDefinedProperty

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: asIterable , getCreated , getCreator , getKey , getLastEditor , getLastUpdated , getName , isEnabled

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: equals , getWrapped , hashCode

Methods inherited from java.lang.Object: getClass , notify , notifyAll , toString , wait



B.1.1.44. KlarosIssueManagement

An issue management system.

B.1.1.44.1. Synopsis

```
public class KlarosIssueManagement extends
                                KlarosExternalServer<KlarosIssueManagement,
 IssueManagement>

  implements IKlarosIssueManagement {

 // Public Methods
```

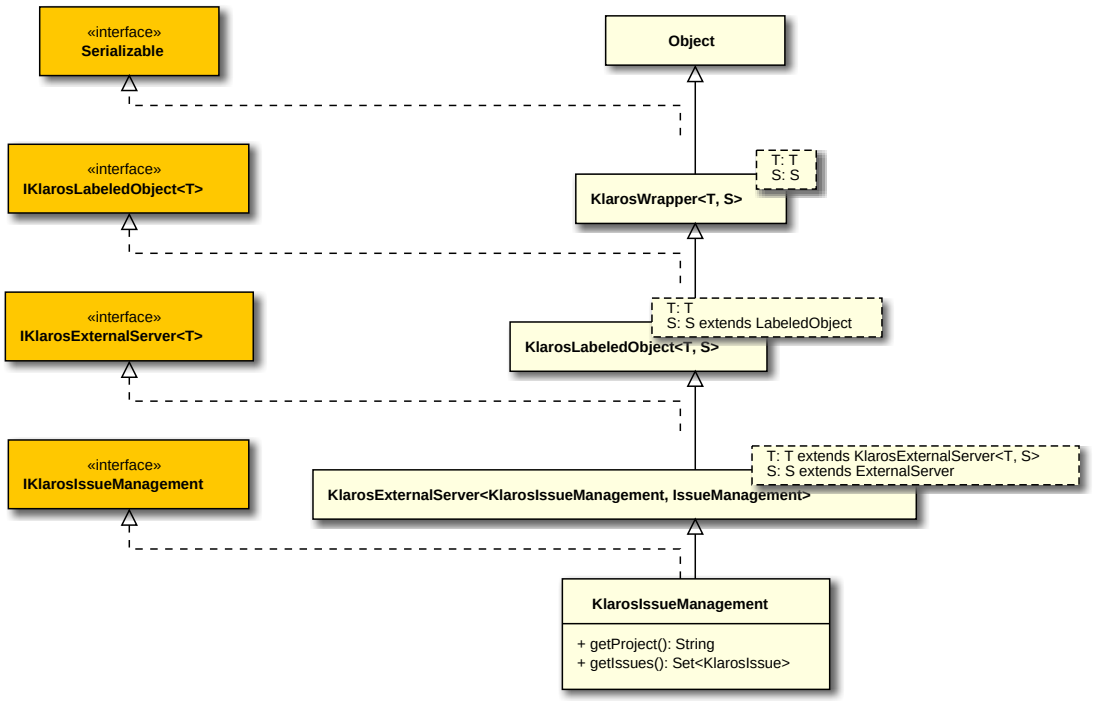
```
  public Set<KlarosIssue> getIssues();
  public String getProject();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosExternalServer: `getDescription` , `getType` , `getUrl`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.45. KlarosIteration

An iteration in a project.

B.1.1.45.1. Synopsis

```
public final class KlarosIteration extends
                                KlarosNamedEntity<KlarosIteration, Iteration>

  implements IKlarosIteration,
         Comparable<KlarosIteration> {

  // Public Methods
  public int compareTo(KlarosIteration o);
  public boolean equals(Object o);
  public Set<KlarosAttachment> getAttachments();
```

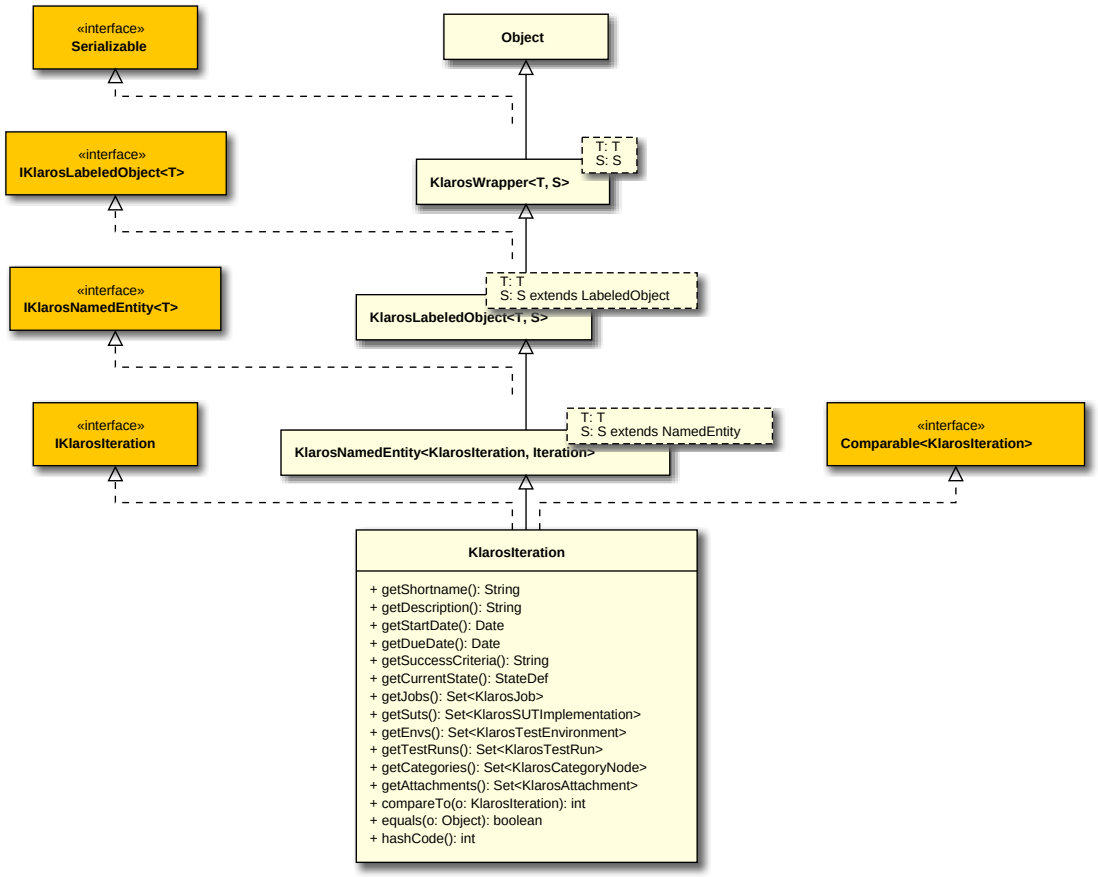
```
  public Set<KlarosCategoryNode> getCategories();
  public StateDef getCurrentState();
  public String getDescription();
  public Date getDueDate();
  public Set<KlarosTestEnvironment> getEnvs();
  public Set<KlarosJob> getJobs();
  public String getShortname();
  public Date getStartDate();
  public String getSuccessCriteria();
  public Set<KlarosSUTImplementation> getSuts();
  public Set<KlarosTestRun> getTestRuns();
  public int hashCode();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: getAssignee , getProperties , getPropertyValue , isDefinedProperty

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: asIterable , getCreated , getCreator , getKey , getLastEditor , getLastUpdated , getName , isEnabled

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: equals , getWrapped , hashCode

Methods inherited from java.lang.Object: getClass , notify , notifyAll , toString , wait



B.1.1.46. KlarosJob

This class provides access to the information stored for jobs.

B.1.1.46.1. Synopsis

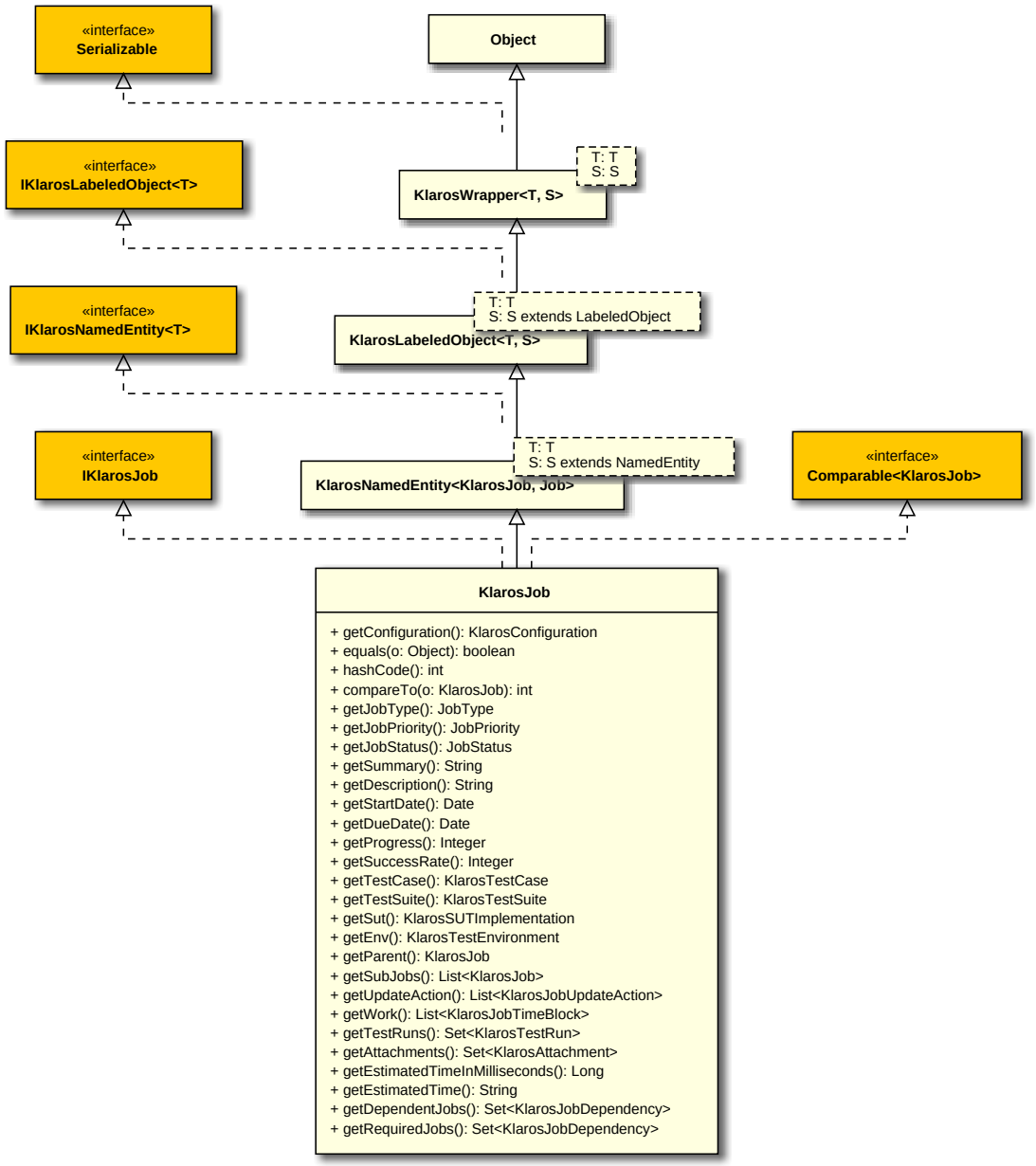
```
public final class KlarosJob extends
                                KlarosNamedEntity<KlarosJob, Job>

   implements IKlarosJob,
           Comparable<KlarosJob> {

  // Public Methods
  public int compareTo(KlarosJob o);
  public boolean equals(Object o);
  public Set<KlarosAttachment> getAttachments();
  public KlarosConfiguration getConfiguration();
  public Set<KlarosJobDependency> getDependentJobs();
  public String getDescription();
  public Date getDueDate();
  public KlarosTestEnvironment getEnv();
  public String getEstimatedTime();
  public Long getEstimatedTimeInMilliseconds();
  public JobPriority getJobPriority();
  public JobStatus getJobStatus();
  public JobType getJobType();
  public KlarosJob getParent();
  public Integer getProgress();
  public Set<KlarosJobDependency> getRequiredJobs();
  public Date getStartDate();
  public List<KlarosJob> getSubJobs();
  public Integer getSuccessRate();
  public String getSummary();
  public KlarosSUTImplementation getSut();
  public KlarosTestCase getTestCase();
  public Set<KlarosTestRun> getTestRuns();
  public KlarosTestSuite getTestSuite();
  public List<KlarosJobUpdateAction> getUpdateAction();
  public List<KlarosJobTimeBlock> getWork();
  public int hashCode();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: getAssignee , getProperties , getPropertyValue , isDefinedProperty

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: asIterable , getCreated , getCreator , getKey , getLastEditor , getLastUpdated , getName , isEnabled

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: equals , getWrapped , hashCode

Methods inherited from java.lang.Object: getClass , notify , notifyAll , toString , wait



B.1.1.47. KlarosJobDependency

This class provides access to the information stored for jobs.

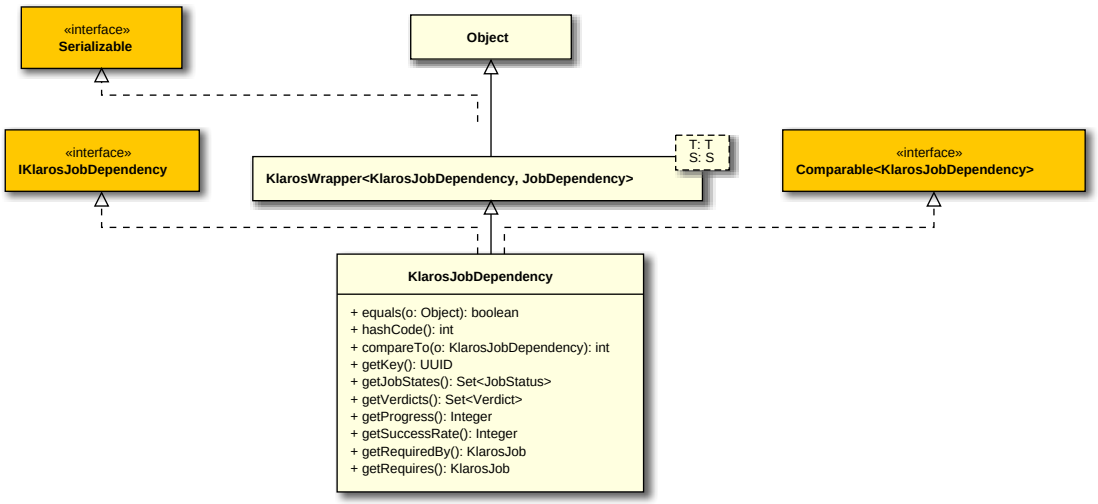
B.1.1.47.1. Synopsis

```
public final class KlarosJobDependency extends KlarosWrapper<KlarosJobDependency,
 JobDependency>
  implements IKlarosJobDependency,
         Comparable<KlarosJobDependency> {

 // Public Methods
 public int compareTo(KlarosJobDependency o);
 public boolean equals(Object o);
 public Set<JobStatus> getJobStates();
 public UUID getKey();
```

```
  public Integer getProgress();
  public KlarosJob getRequiredBy();
  public KlarosJob getRequires();
  public Integer getSuccessRate();
  public Set<Verdict> getVerdicts();
  public int hashCode();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.48. KlarosJobTimeBlock

The implementation of a job time block, measuring a unit of work done for a job.

B.1.1.48.1. Synopsis

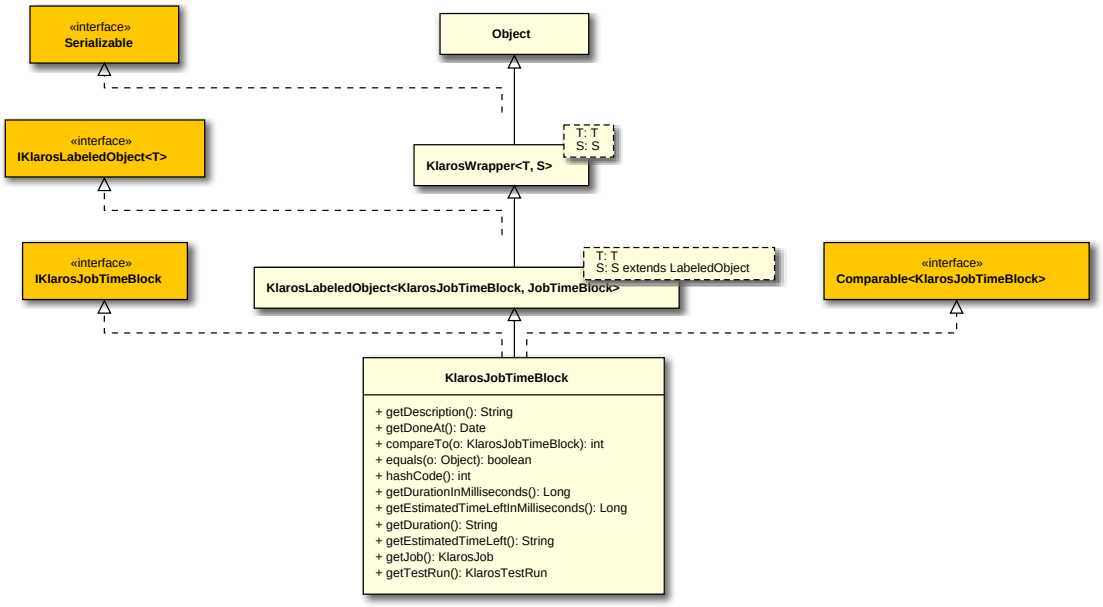
```
public final class KlarosJobTimeBlock extends
                                KlarosLabeledObject<KlarosJobTimeBlock,
 JobTimeBlock>

  implements IKlarosJobTimeBlock,
         Comparable<KlarosJobTimeBlock> {

  // Public Methods
  public int compareTo(KlarosJobTimeBlock o);
  public boolean equals(Object o);
  public String getDescription();
  public Date getDoneAt();
  public String getDuration();
  public Long getDurationInMilliseconds();
  public String getEstimatedTimeLeft();
  public Long getEstimatedTimeLeftInMilliseconds();
  public KlarosJob getJob();
  public KlarosTestRun getTestRun();
```

```
  public int hashCode();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: asIterable , getCreated , getCreator , getKey , getLastEditor , getLastUpdated , getName , isEnabled

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: equals , getWrapped , hashCode

Methods inherited from java.lang.Object: getClass , notify , notifyAll , toString , wait



B.1.1.49. KlarosJobUpdateAction

The model of a job comment.

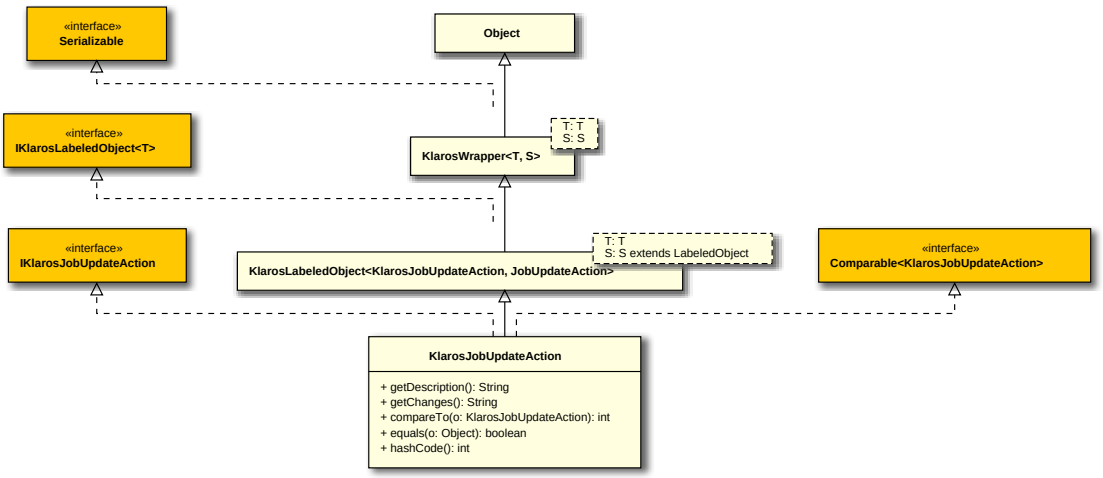
B.1.1.49.1. Synopsis

```
public final class KlarosJobUpdateAction extends
                                KlarosLabeledObject<KlarosJobUpdateAction,
 JobUpdateAction>

  implements IKlarosJobUpdateAction,
         Comparable<KlarosJobUpdateAction> {

  // Public Methods
  public int compareTo(KlarosJobUpdateAction o);
  public boolean equals(Object o);
  public String getChanges();
  public String getDescription();
  public int hashCode();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: asIterable , getCreated , getCreator , getKey , getLastEditor , getLastUpdated , getName , isEnabled

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.50. KlarosLabeledObject

This class encapsulates the (dynamic) properties of a klaros object.

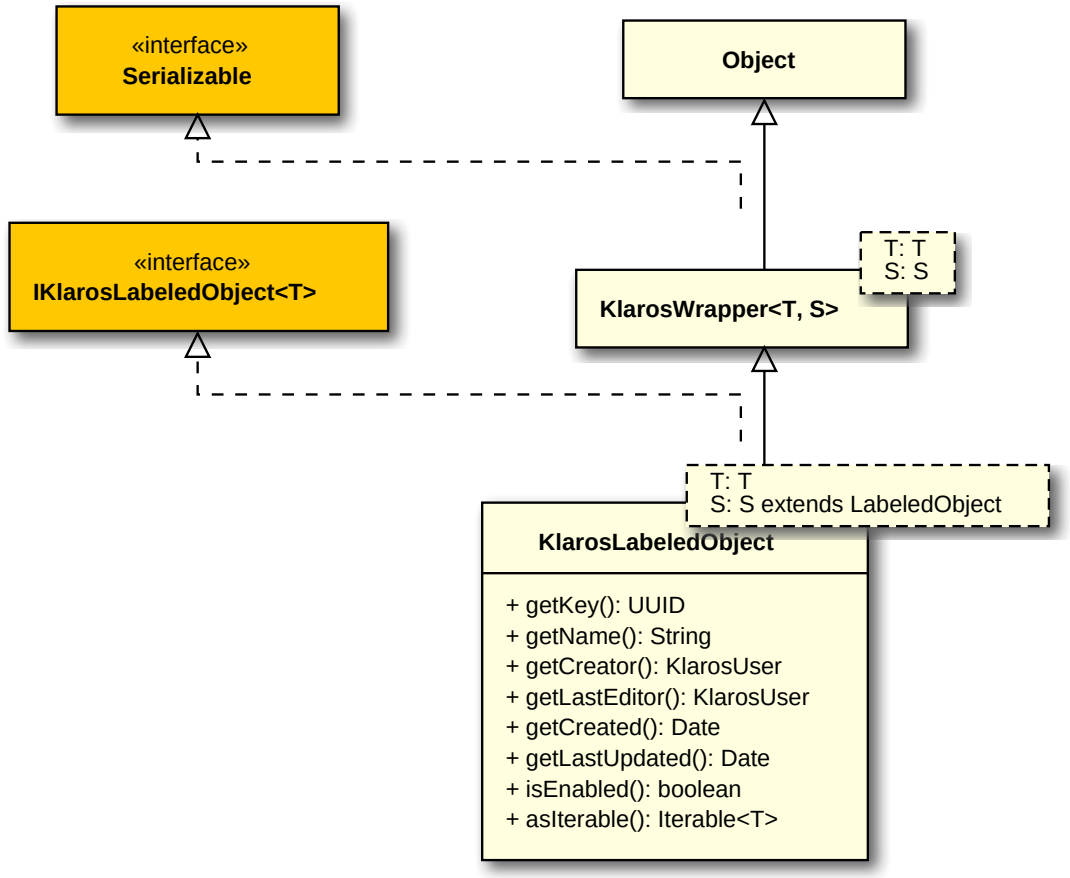
B.1.1.50.1. Synopsis

```
public abstract class KlarosLabeledObject<T,S extends LabeledObject>
 extends KlarosWrapper<T, S>
  implements IKlarosLabeledObject<T> {

  // Public Methods
  public Iterable<T> asIterable();
  public Date getCreated();
  public KlarosUser getCreator();
  public UUID getKey();
  public KlarosUser getLastEditor();
  public Date getLastUpdated();
  public String getName();
  public boolean isEnabled();
}
```

Direct known subclasses: `de.verit.klaros.core.model.KlarosConfiguration` , `de.verit.klaros.core.model.KlarosExternalServer` , `de.verit.klaros.core.model.KlarosJobTimeBlock` , `de.verit.klaros.core.model.KlarosJobUpdateAction` , `de.verit.klaros.core.model.KlarosNamedEntity` , `de.verit.klaros.core.model.KlarosRepositoryEntity` , `de.verit.klaros.core.model.KlarosUser`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.51. KlarosNamedEntity

This class encapsulates the (dynamic) properties of a klaros object.

B.1.1.51.1. Synopsis

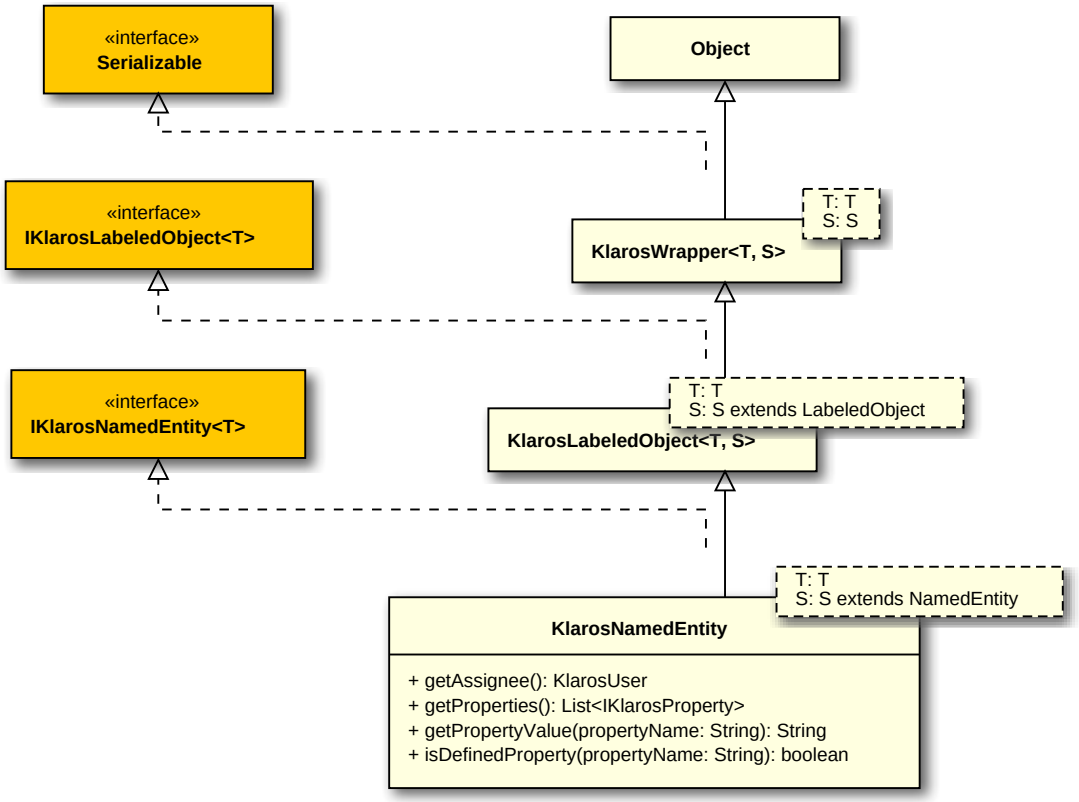
```
public abstract class KlarosNamedEntity<T,S extends NamedEntity> extends
                                KlarosLabeledObject<T, S>

   implements IKlarosNamedEntity<T> {

  // Public Methods
  public KlarosUser getAssignee();
  public List<IKlarosProperty> getProperties();
  public String getPropertyValue(String propertyName);
  public boolean isDefinedProperty(String propertyName);
}
```

Direct known subclasses: de.verit.klaros.core.model.KlarosCategoryTree , de.verit.-klaros.core.model.KlarosIssue , de.verit.klaros.core.model.KlarosIteration , de.verit.klaros.core.model.KlarosJob , de.verit.klaros.core.model.KlarosResult , de.verit.-klaros.core.model.KlarosRevision , de.verit.klaros.core.model.KlarosSUTImplementation , de.verit.klaros.core.model.KlarosTestEnvironment , de.verit.klaros.core.model.Klaros-TestRun , de.verit.klaros.core.model.KlarosTestStepContainer

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.52. KlarosProperty

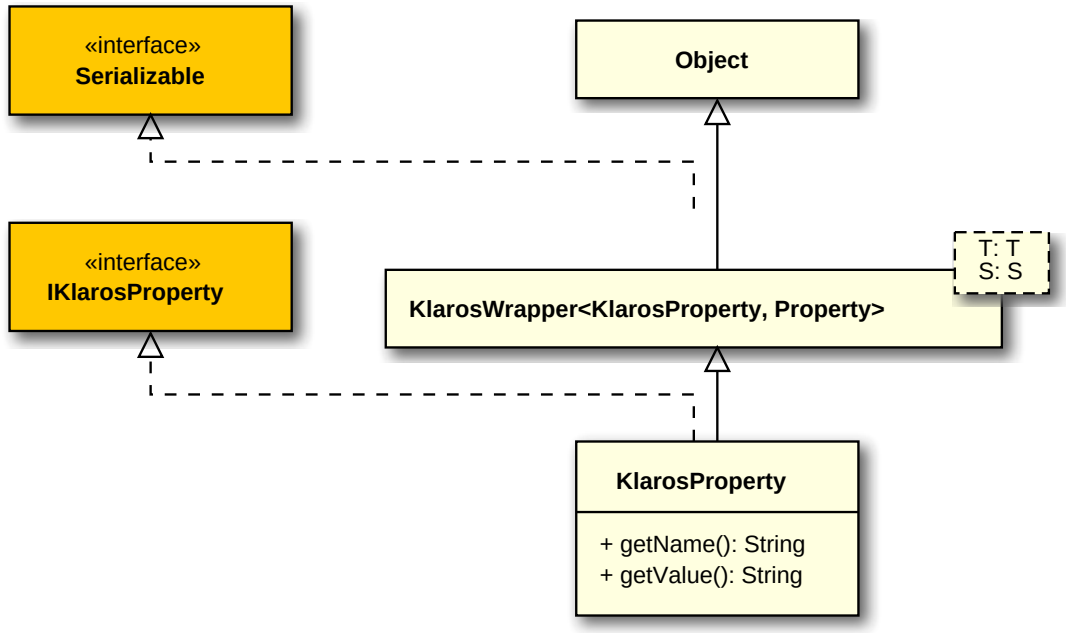
The Klaros property class.

B.1.1.52.1. Synopsis

```
public final class KlarosProperty extends KlarosWrapper<KlarosProperty, Property>
   implements IKlarosProperty {

  // Public Methods
  public String getName();
  public String getValue();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.53. KlarosRepositoryEntity

This class encapsulates the (dynamic) properties of a klaros object.

B.1.1.53.1. Synopsis

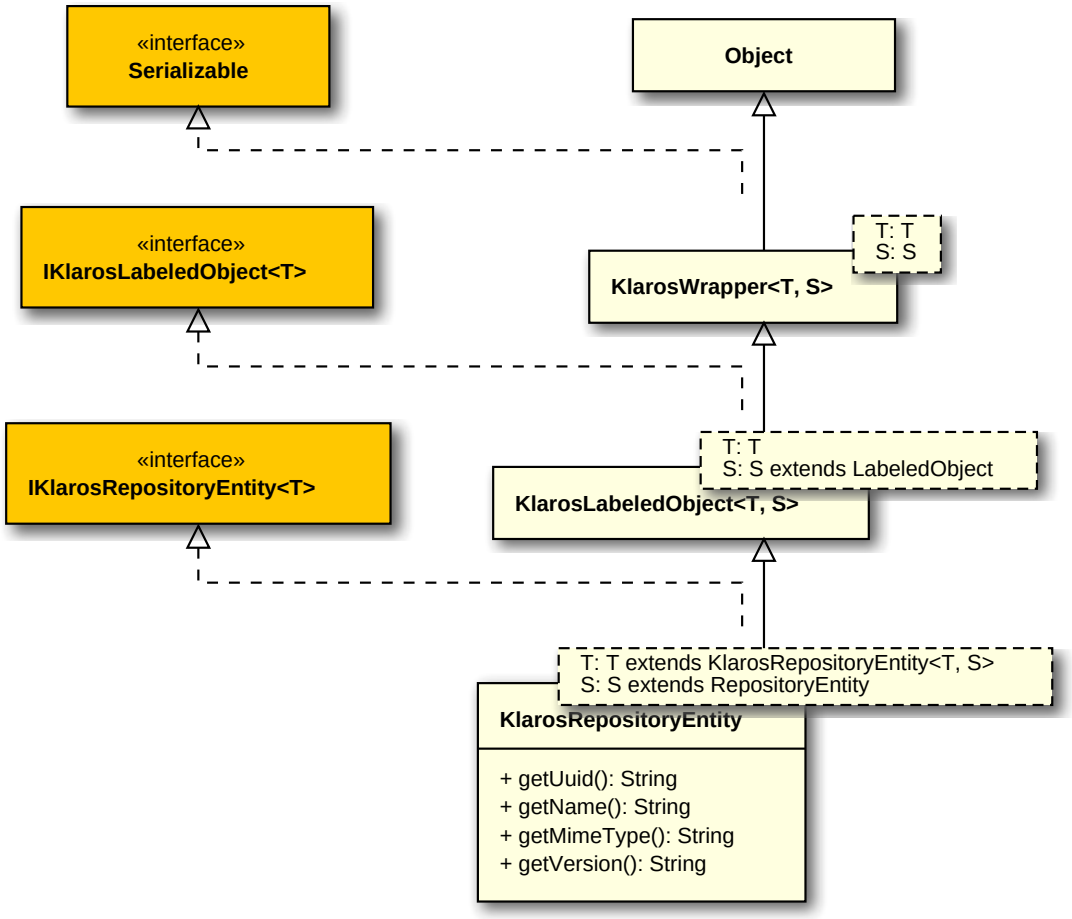
```
public abstract class KlarosRepositoryEntity<T extends KlarosRepositoryEntity<T, S>,S
 extends RepositoryEntity> extends
                                KlarosLabeledObject<T, S>

   implements IKlarosRepositoryEntity<T> {

  // Public Methods
  public String getMimeType();
  public String getName();
  public String getUuid();
  public String getVersion();
}
```

Direct known subclasses: `de.verit.klaros.core.model.KlarosAttachment`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.54. KlarosRequirement

This class provides access to the information stored for requirements.

B.1.1.54.1. Synopsis

```
public final class KlarosRequirement extends
                                KlarosRevision<KlarosRequirement, Requirement>

  implements IKlarosRequirement<KlarosRequirement, Requirement> {

 // Public Methods
 public Set<KlarosAttachment> getAttachments();
 public Set<KlarosCategoryNode> getCategories();
 public KlarosConfiguration getConfiguration();
 public Set<KlarosTestCase> getCoveringTestCases();
 public String getDescription();
 public Set<String> getExternalNames();
 public String getExternalStatus();
 public Set<KlarosIteration> getIterations();
 public KlarosRequirement getPredecessor();
 public RequirementPriority getPriority();
 public KlarosRequirement getRoot();
 public String getShortname();
```

```
  public String getState();
  public KlarosRequirement getSuccessor();
  public String getSummary();
}
```

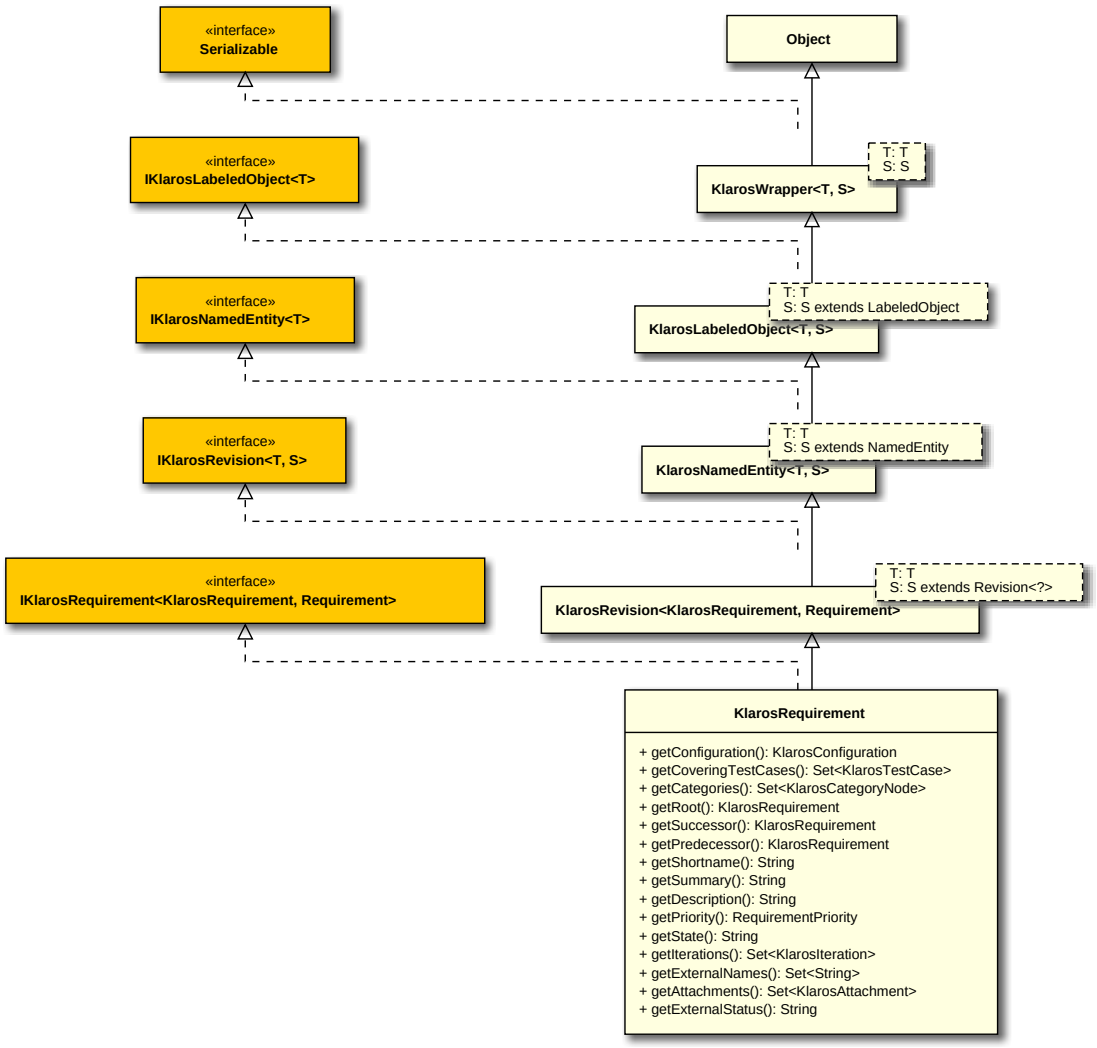
Methods inherited from de.verit.klaros.core.model.KlarosRevision: `getRevisionComment` , `getRevisionId`

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `getPropertyValue` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.54.2. getExternalStatus()

```
public String getExternalStatus();
```

The external status of this requirement.

Parameters

`return` the external status

B.1.1.55. KlarosRequirementsManagement

A requirement management system.

B.1.1.55.1. Synopsis

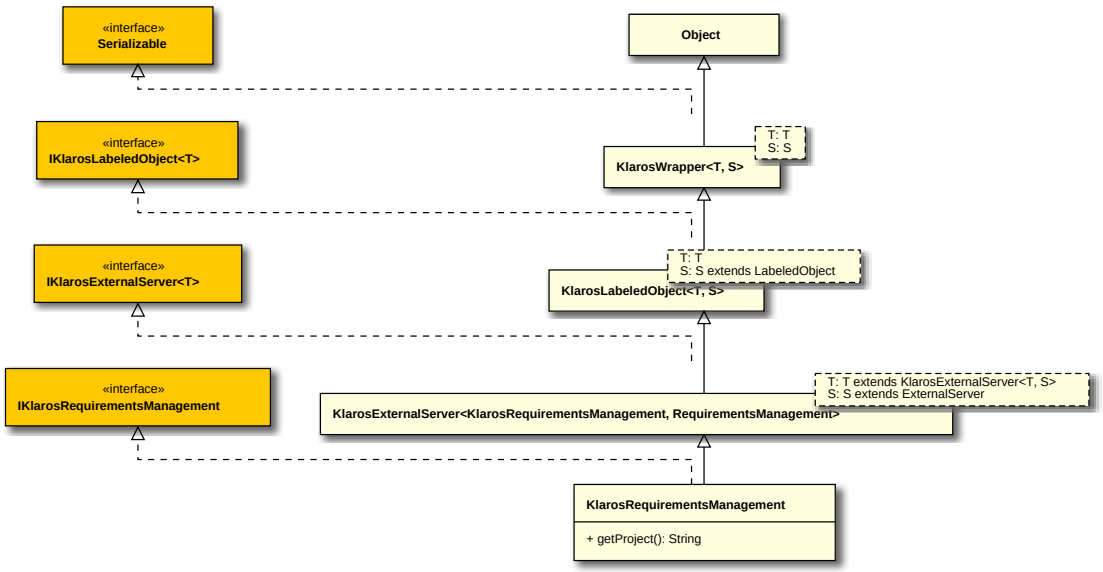
```
public class KlarosRequirementsManagement extends
                                KlarosExternalServer<KlarosRequirementsManagement,
 RequirementsManagement>

  implements IKlarosRequirementsManagement {

 // Public Methods
 public String getProject();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosExternalServer: `getDescription` , `getType` , `getUrl`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.56. KlarosResult

This class provides access to the information stored for generic results.

B.1.1.56.1. Synopsis

```
public abstract class KlarosResult<T,S extends Result> extends
                                KlarosNamedEntity<T, S>

   implements IKlarosResult<T> {

  // Public Methods
  public KlarosConfiguration getConfiguration();
  public String getDescription();
  public long getExecutionTime();
  public KlarosJob getJob();
  public String getSummary();
  public String getVerdict();
  public String getVerdict(Locale locale);
  public boolean isError();
  public boolean isFailure();
  public boolean isInconclusive();
  public boolean isPassed();
  public boolean isSkipped();
}
```

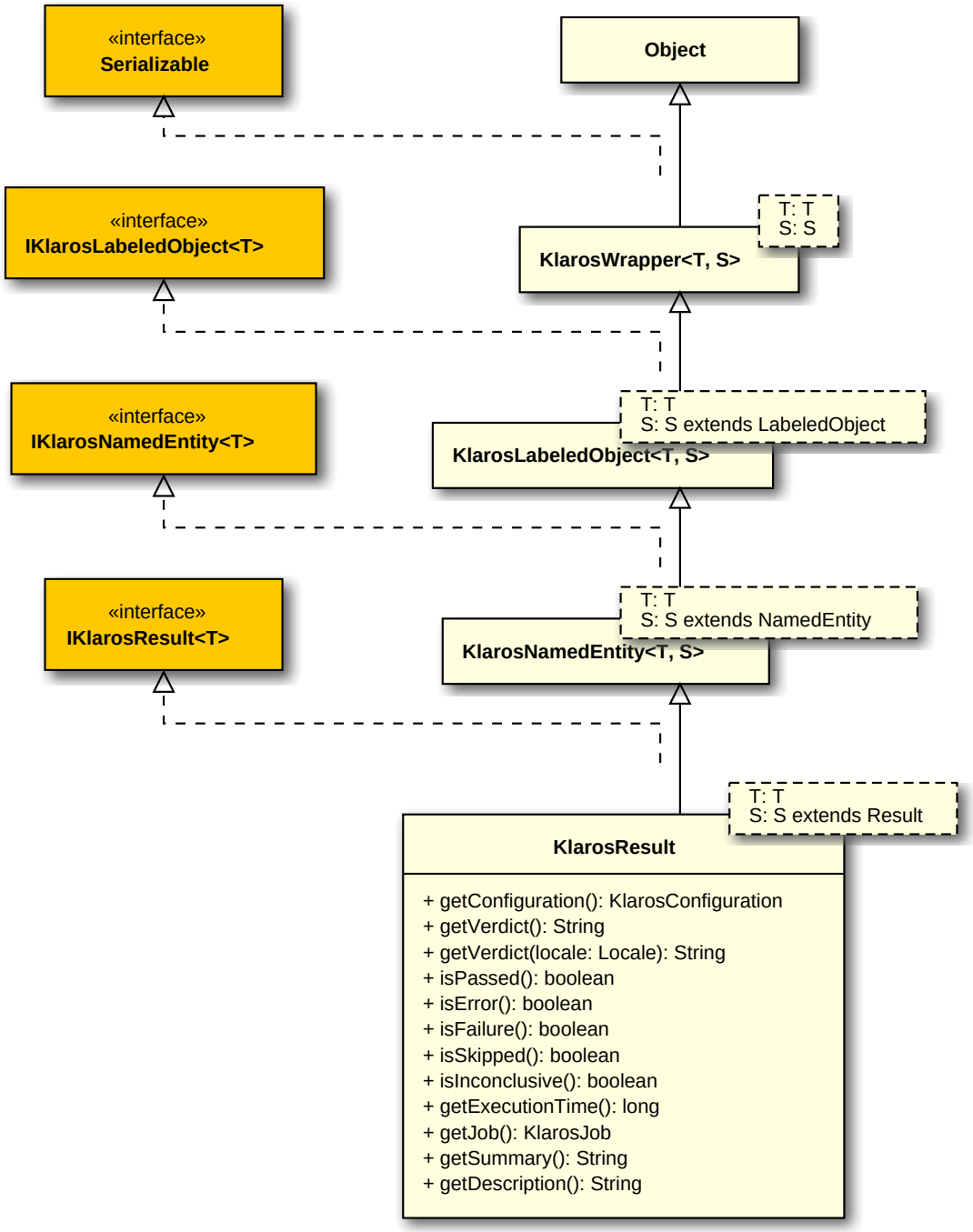
Direct known subclasses: de.verit.klaros.core.model.KlarosTestCaseResult , de.verit.klaros.core.model.KlarosTestCaseStepResult , de.verit.klaros.core.model.KlarosTestSuiteResult

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: getAssignee , getProperties , getPropertyValue , isDefinedProperty

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: asIterable , getCreated , getCreator , getKey , getLastEditor , getLastUpdated , getName , isEnabled

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: equals , getWrapped , hashCode

Methods inherited from java.lang.Object: getClass , notify , notifyAll , toString , wait



B.1.1.57. KlarosRevision

This class encapsulates the revision related information of a klaros object.

B.1.1.57.1. Synopsis

```
public abstract class KlarosRevision<T,S extends Revision<?>> extends
                                KlarosNamedEntity<T, S>

  implements IKlarosRevision<T, S> {

 // Public Methods
```

```
  public final String getRevisionComment();
  public final String getRevisionId();
}
```

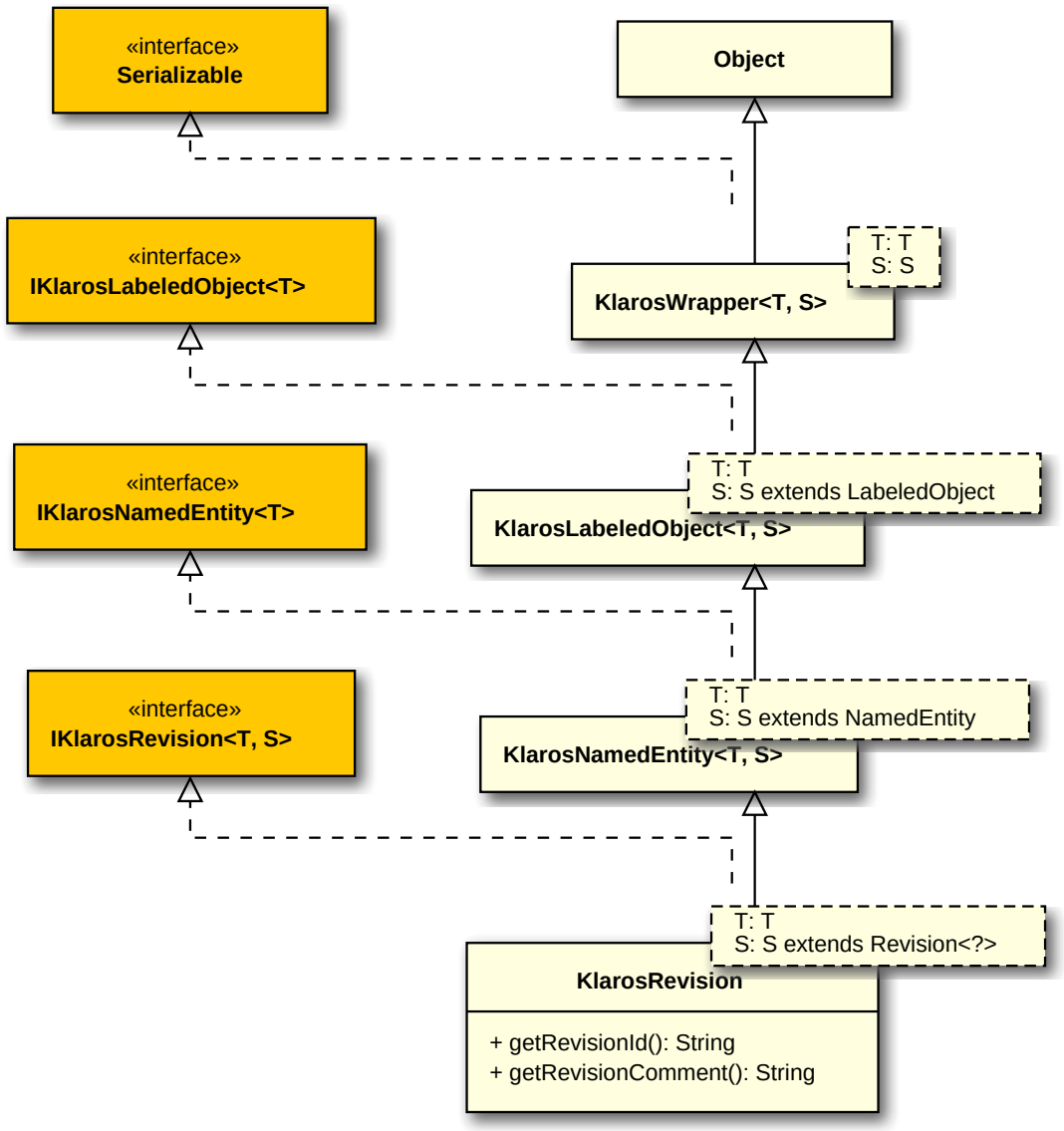
Direct known subclasses: de.verit.klaros.core.model.KlarosRequirement , de.verit.klaros.core.model.KlarosTestCase , de.verit.klaros.core.model.KlarosTestSegment , de.verit.klaros.core.model.KlarosTestSuite

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: getAssignee , getProperties , getPropertyValue , isDefinedProperty

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: asIterable , getCreated , getCreator , getKey , getLastEditor , getLastUpdated , getName , isEnabled

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: equals , getWrapped , hashCode

Methods inherited from java.lang.Object: getClass , notify , notifyAll , toString , wait



B.1.1.58. KlarosSUTImplementation

This class provides access to the information stored for systems under tests (SUT).

B.1.1.58.1. Synopsis

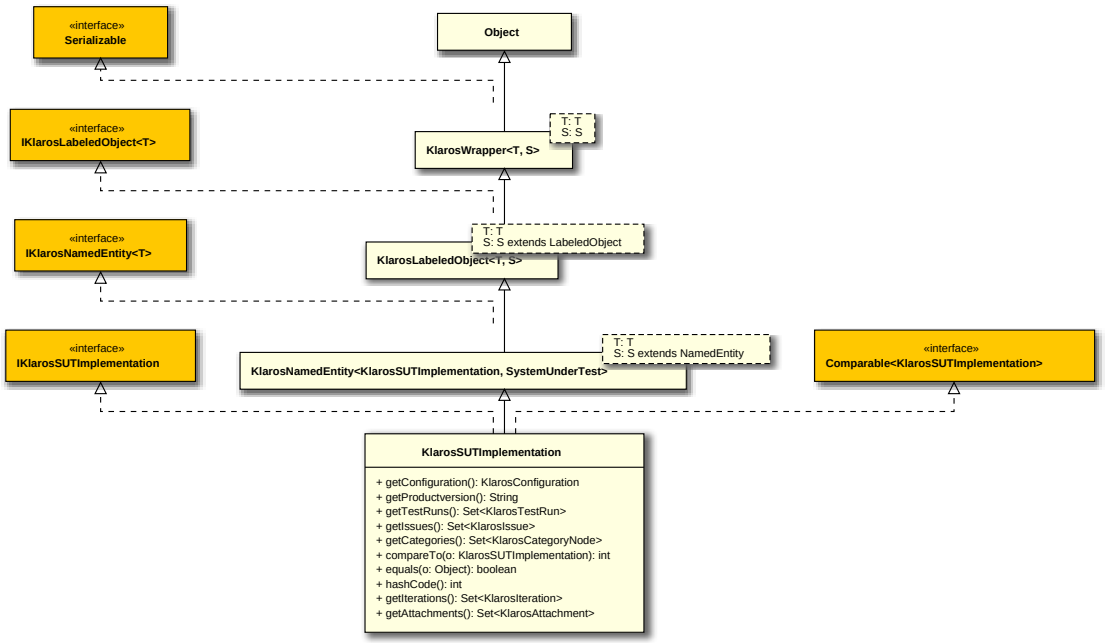
```
public final class KlarosSUTImplementation extends
                                KlarosNamedEntity<KlarosSUTImplementation,
 SystemUnderTest>

  implements IKlarosSUTImplementation,
          Comparable<KlarosSUTImplementation> {

 // Public Methods
 public int compareTo(KlarosSUTImplementation o);
 public boolean equals(Object o);
 public Set<KlarosAttachment> getAttachments();
 public Set<KlarosCategoryNode> getCategories();
 public KlarosConfiguration getConfiguration();
 public Set<KlarosIssue> getIssues();
 public Set<KlarosIteration> getIterations();
 public String getProductversion();
 public Set<KlarosTestRun> getTestRuns();
 public int hashCode();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `getPropertyValue` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.59. KlarosTestCase

This class provides access to the information stored for a test case.

B.1.1.59.1. Synopsis

```
public final class KlarosTestCase extends
                                KlarosRevision<KlarosTestCase, TestCase>

  implements IKlarosTestCase,
         Comparable<KlarosTestCase> {

 // Public Methods
 public int compareTo(KlarosTestCase o);
 public boolean equals(Object o);
 public Set<KlarosAttachment> getAttachments();
 public Set<KlarosCategoryNode> getCategories();
 public KlarosConfiguration getConfiguration();
 public Set<KlarosRequirement> getCovers();
 public String getDependency();
 public String getDescription();
 public TestDesignTechnique getDesignTechnique();
 public List<KlarosIssue> getDetectedIssues();
 public String getDocbase();
 public Long getEstimatedDuration();
 public TestEvaluationMethod getEvaluation();
 public TestExecutionMethod getExecution();
 public String getExpectedResult();
 public Set<String> getExternalNames();
 public List<KlarosJob> getJobs();
 public TestLevel getLevel();
 public String getNote();
 public String getPostcondition();
```

```
  public String getPrecondition();
  public KlarosTestCase getPredecessor();
  public TestPriority getPriority();
  public Set<KlarosTestCaseResult> getResults();
  public KlarosTestCase getRoot();
  public String getShortname();
  public String getState();
  public KlarosTestCase getSuccessor();
  public String getTeam();
  public List<KlarosTestCaseStep> getTestCaseSteps();
  public TestAreatopic getTestType();
  public String getTraceability();
  public TestVariety getVariety();
  public int hashCode();
}
```

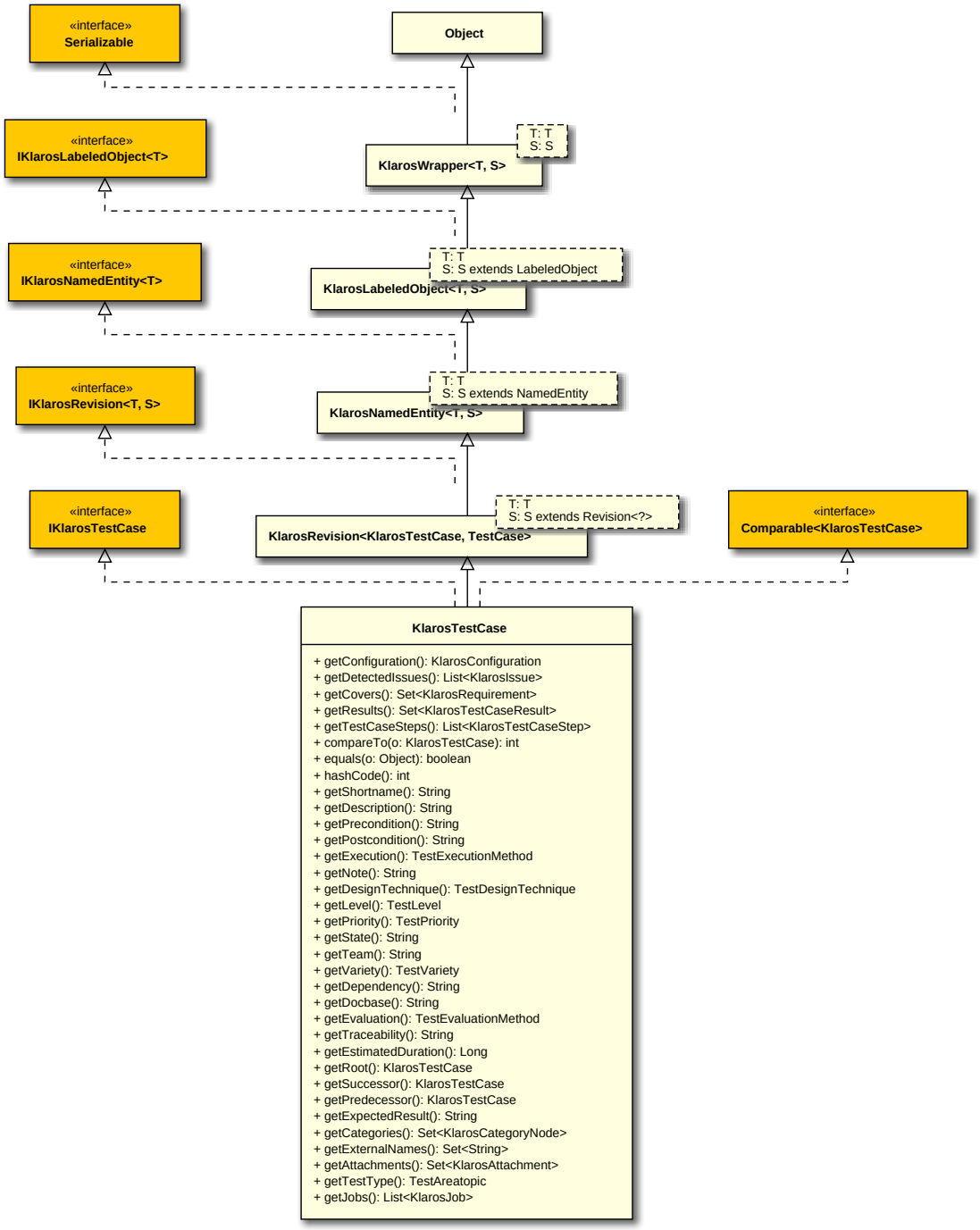
Methods inherited from de.verit.klaros.core.model.KlarosRevision: getRevisionComment , getRevisionId

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: getAssignee , getProperties , getPropertyValue , isDefinedProperty

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: asIterable , getCreated , getCreator , getKey , getLastEditor , getLastUpdated , getName , isEnabled

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: equals , getWrapped , hashCode

Methods inherited from java.lang.Object: getClass , notify , notifyAll , toString , wait



B.1.1.60. KlarosTestCaseResult

This class provides access to the information stored for test case results.

B.1.1.60.1. Synopsis

```
public final class KlarosTestCaseResult extends
                              KlarosResult<KlarosTestCaseResult, TestCaseResult>

  implements IKlarosTestCaseResult,
         Comparable<KlarosTestCaseResult> {
```

```
  // Public Methods
  public int compareTo(KlarosTestCaseResult o);
  public boolean equals(Object o);
  public Set<KlarosAttachment> getAttachments();
  public List<KlarosTestCaseStepResult> getStepResults();
  public KlarosTestCase getTestCase();
  public KlarosTestRun getTestRun();
  public Integer getTestSuitePosition();
  public KlarosTestSuiteResult getTestSuiteResult();
  public int hashCode();
  public boolean isPending();
}
```

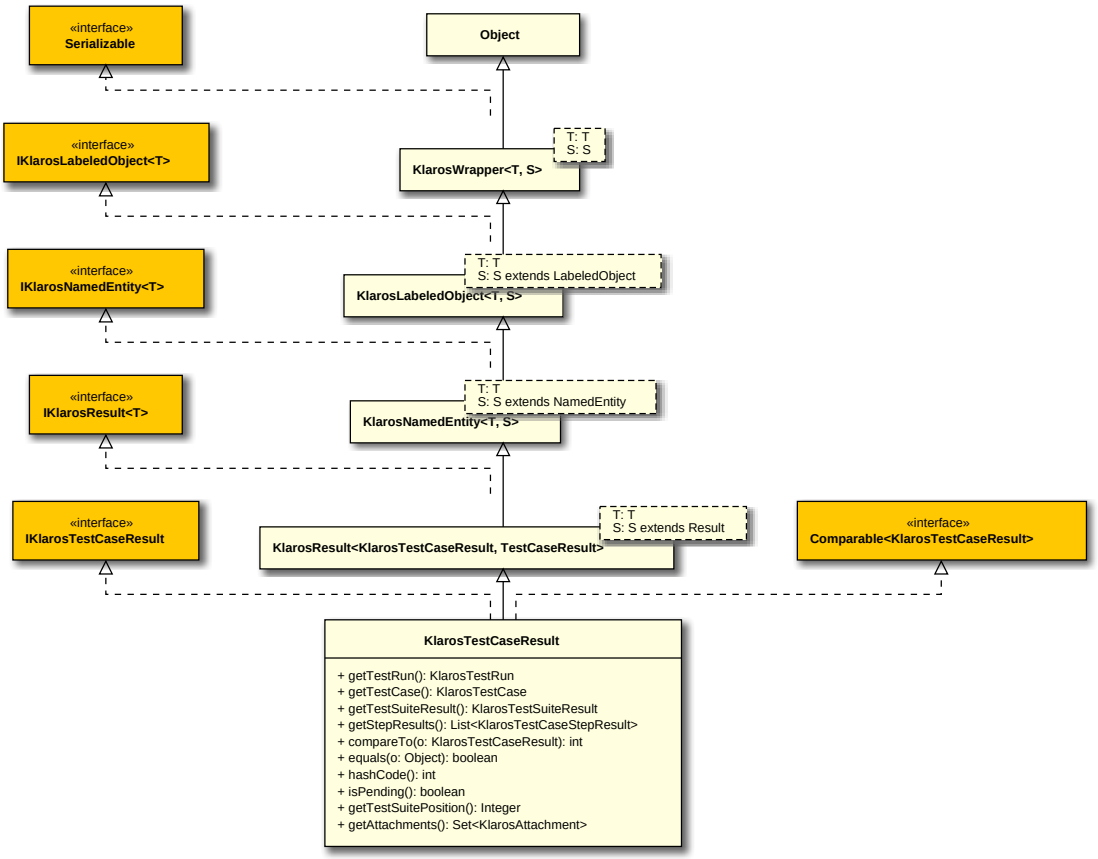
Methods inherited from de.verit.klaros.core.model.KlarosResult: `getConfiguration` , `getDescription` , `getExecutionTime` , `getJob` , `getSummary` , `getVerdict` , `isError` , `isFailure` , `isInconclusive` , `isPassed` , `isSkipped`

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `getPropertyValue` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.61. KlarosTestCaseStep

This class provides access to the information stored for test case steps.

B.1.1.61.1. Synopsis

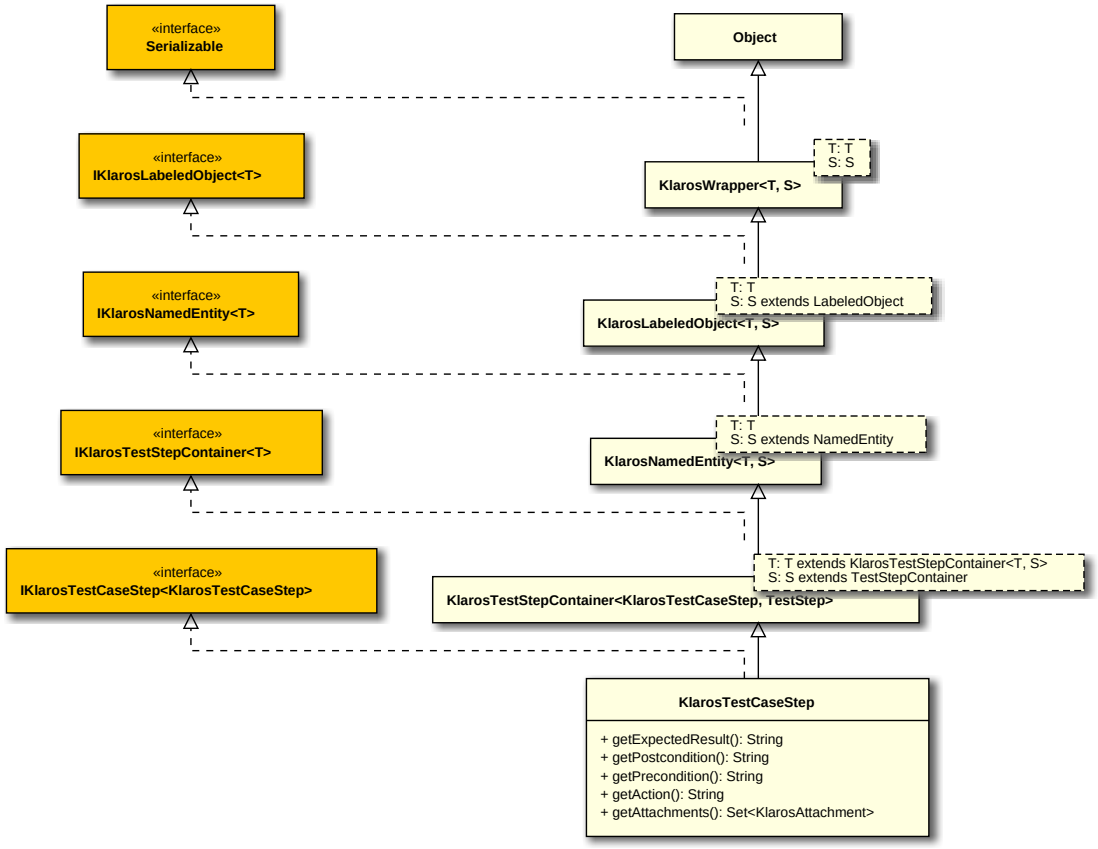
```
public final class KlarosTestCaseStep extends
                                KlarosTestStepContainer<KlarosTestCaseStep,
 TestStep>

  implements IKlarosTestCaseStep<KlarosTestCaseStep> {

 // Public Methods
 public String getAction();
 public Set<KlarosAttachment> getAttachments();
 public String getExpectedResult();
 public String getPostcondition();
 public String getPrecondition();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `getPropertyValue` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.62. KlarosTestCaseStepResult

This class provides access to the information stored for test case step results.

B.1.1.62.1. Synopsis

```
public final class KlarosTestCaseStepResult extends
                                KlarosResult<KlarosTestCaseStepResult,
 TestCaseStepResult>

  implements IKlarosTestCaseStepResult,
         Comparable<KlarosTestCaseStepResult> {

  // Public Methods
  public int compareTo(KlarosTestCaseStepResult o);
  public boolean equals(Object o);
  public String getAction();
  public Set<KlarosAttachment> getAttachments();
  public String getExpectedResult();
  public String getPostcondition();
  public String getPrecondition();
  public KlarosTestCase getTestCase();
  public KlarosTestCaseResult getTestCaseResult();
  public KlarosTestRun getTestRun();
  public int hashCode();
}
```

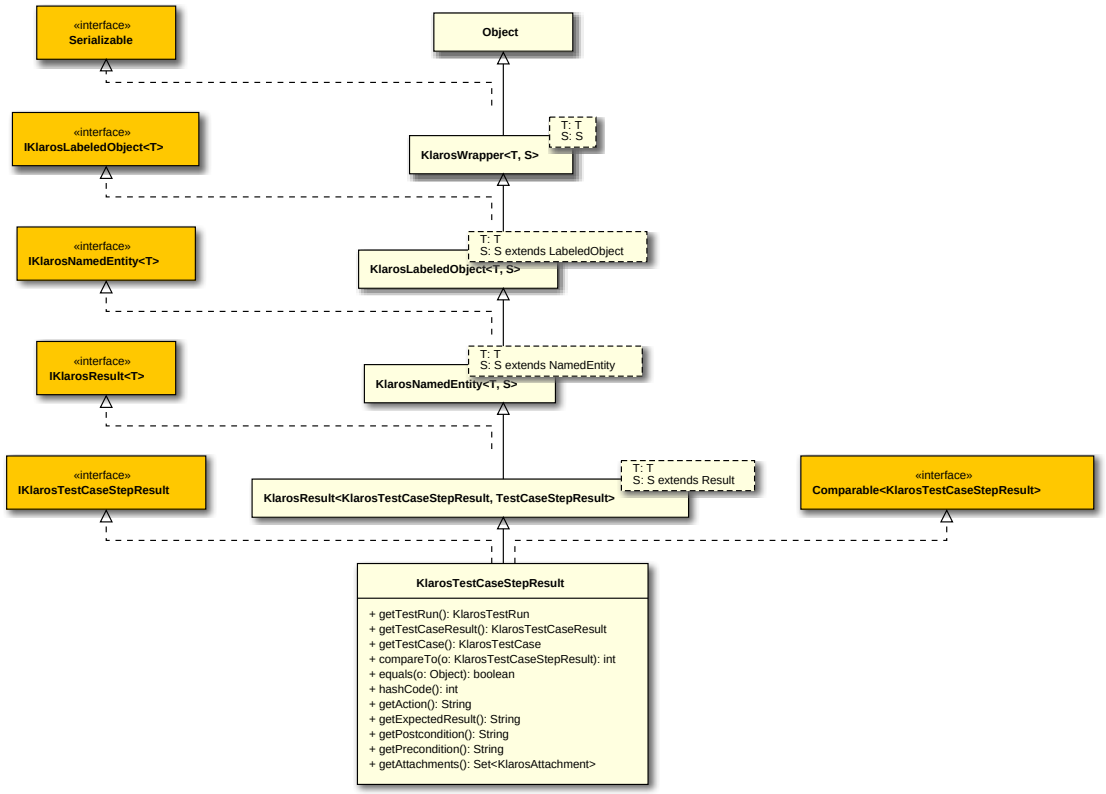
Methods inherited from de.verit.klaros.core.model.KlarosResult: getConfiguration , getDescription , getExecutionTime , getJob , getSummary , getVerdict , isError , isFailure , isInconclusive , isPassed , isSkipped

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: getAssignee , getProperties , getPropertyValue , isDefinedProperty

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: asIterable , getCreated , getCreator , getKey , getLastEditor , getLastUpdated , getName , isEnabled

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: equals , getWrapped , hashCode

Methods inherited from java.lang.Object: getClass , notify , notifyAll , toString , wait



B.1.1.63. KlarosTestEnvironment

This class provides access to the information stored for test environments.

B.1.1.63.1. Synopsis

```
public final class KlarosTestEnvironment extends
                                KlarosNamedEntity<KlarosTestEnvironment,
 TestEnvironment>

  implements IKlarosTestEnvironment,
         Comparable<KlarosTestEnvironment> {

 // Public Methods
```

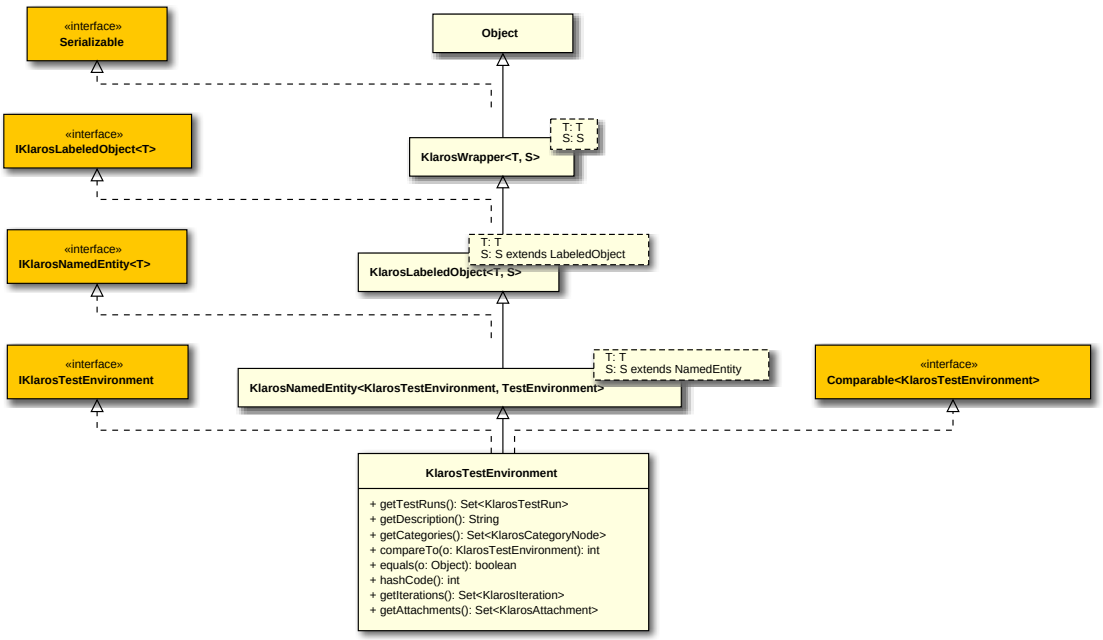
```
  public int compareTo(KlarosTestEnvironment o);
  public boolean equals(Object o);
  public Set<KlarosAttachment> getAttachments();
  public Set<KlarosCategoryNode> getCategories();
  public String getDescription();
  public Set<KlarosIteration> getIterations();
  public Set<KlarosTestRun> getTestRuns();
  public int hashCode();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: getAssignee , getProperties , getPropertyValue , isDefinedProperty

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: asIterable , getCreated , getCreator , getKey , getLastEditor , getLastUpdated , getName , isEnabled

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: equals , getWrapped , hashCode

Methods inherited from java.lang.Object: getClass , notify , notifyAll , toString , wait



B.1.1.64. KlarosTestExecutable

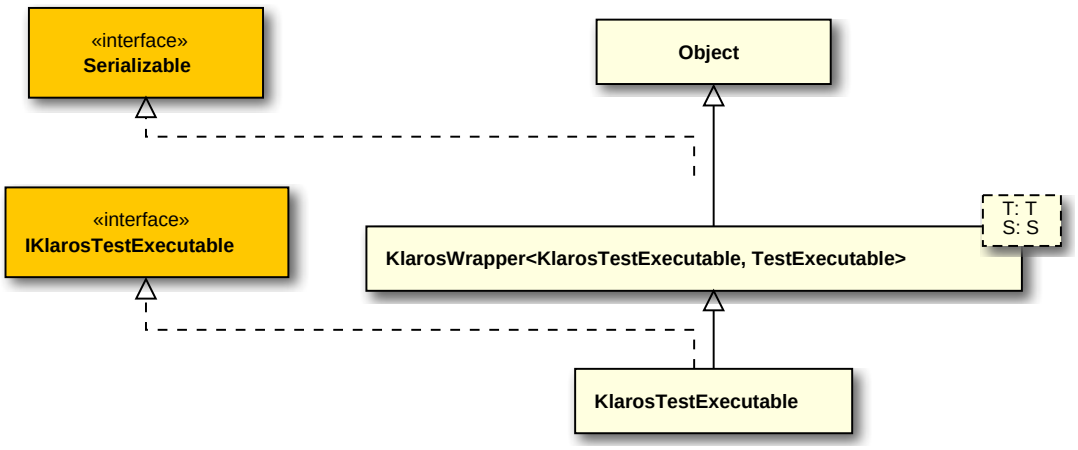
This class provides access to the information stored for test cases.

B.1.1.64.1. Synopsis

```
public final class KlarosTestExecutable extends KlarosWrapper<KlarosTestExecutable,
 TestExecutable>
  implements IKlarosTestExecutable {
}
```

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: equals , getWrapped , hashCode

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.65. KlarosTestRun

This class provides access to the information stored for test runs.

B.1.1.65.1. Synopsis

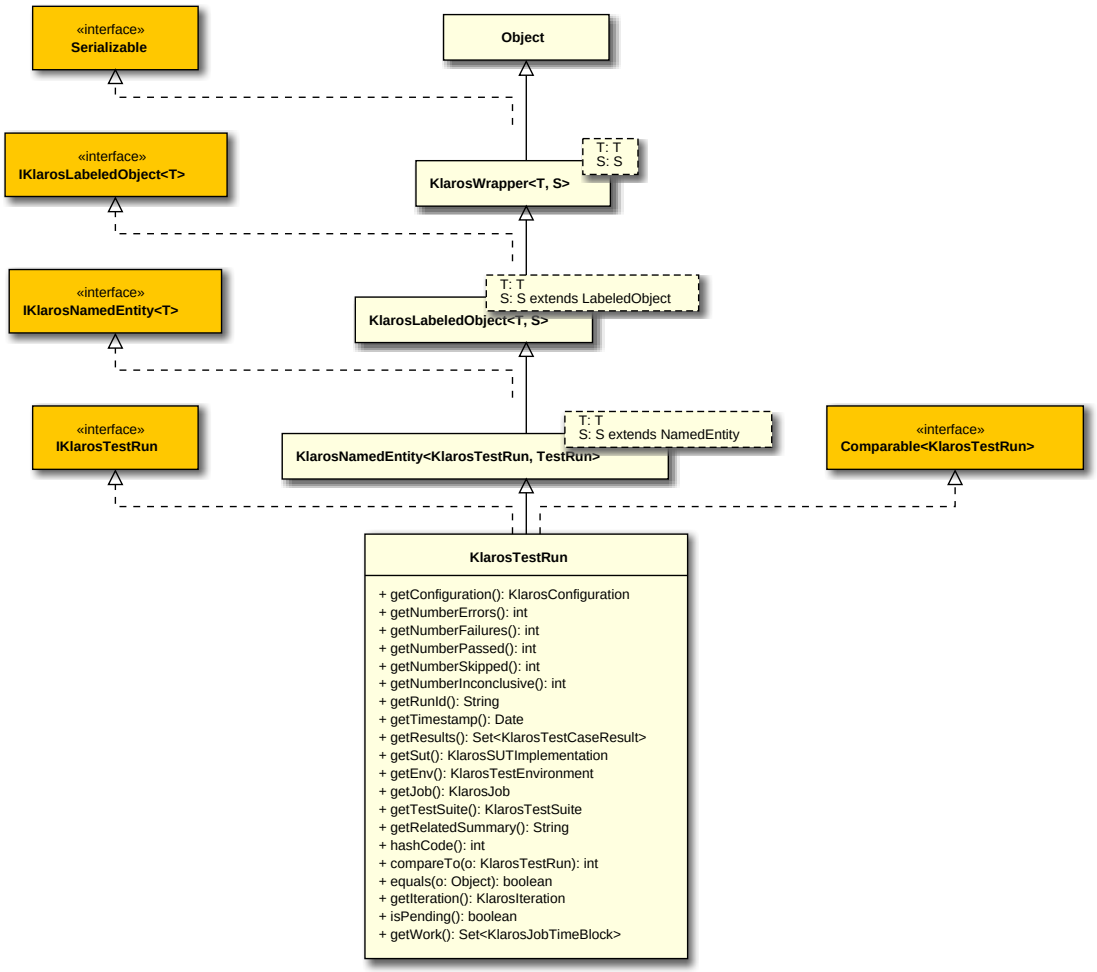
```
public final class KlarosTestRun extends
                                KlarosNamedEntity<KlarosTestRun, TestRun>

  implements IKlarosTestRun,
          Comparable<KlarosTestRun> {

  // Public Methods
  public int compareTo(KlarosTestRun o);
  public boolean equals(Object o);
  public KlarosConfiguration getConfiguration();
  public KlarosTestEnvironment getEnv();
  public KlarosIteration getIteration();
  public KlarosJob getJob();
  public int getNumberErrors();
  public int getNumberFailures();
  public int getNumberInconclusive();
  public int getNumberPassed();
  public int getNumberSkipped();
  public String getRelatedSummary();
  public Set<KlarosTestCaseResult> getResults();
  public String getRunId();
  public KlarosSUTImplementation getSut();
  public KlarosTestSuite getTestSuite();
  public Date getTimestamp();
  public Set<KlarosJobTimeBlock> getWork();
  public int hashCode();
  public boolean isPending();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `getPropertyValue` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.66. KlarosTestSegment

This class provides access to the information stored for test segments.

B.1.1.66.1. Synopsis

```
public final class KlarosTestSegment extends
                                KlarosRevision<KlarosTestSegment, TestSegment>

  implements IKlarosTestSegment<KlarosTestSegment, TestSegment> {

 // Public Methods
 public String getDescription();
 public String getExpectedResult();
 public String getNote();
```

```
  public String getPostcondition();
  public String getPrecondition();
  public KlarosRevision<KlarosTestSegment, TestSegment> getPredecessor();
  public KlarosRevision<KlarosTestSegment, TestSegment> getRoot();
  public String getShortname();
  public List<KlarosTestCaseStep> getSteps();
  public KlarosRevision<KlarosTestSegment, TestSegment> getSuccessor();
}
```

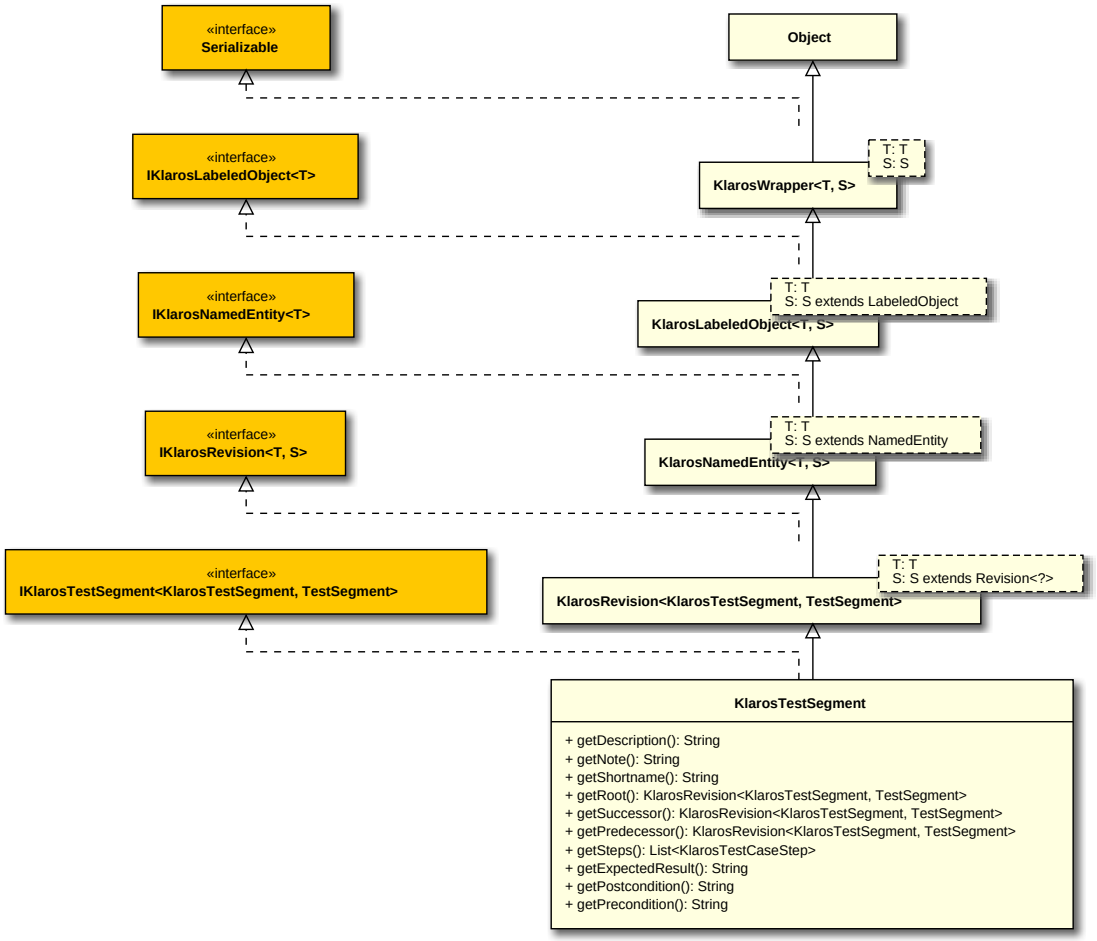
Methods inherited from de.verit.klaros.core.model.KlarosRevision: `getRevisionComment` , `getRevisionId`

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `getPropertyValue` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.67. KlarosTestStepContainer

This class encapsulates the (dynamic) properties of a klaros object.

B.1.1.67.1. Synopsis

```
public abstract class KlarosTestStepContainer<T extends KlarosTestStepContainer<T,
 S>,S extends TestStepContainer> extends
                                KlarosNamedEntity<T, S>

   implements IKlarosTestStepContainer<T> {
}
```

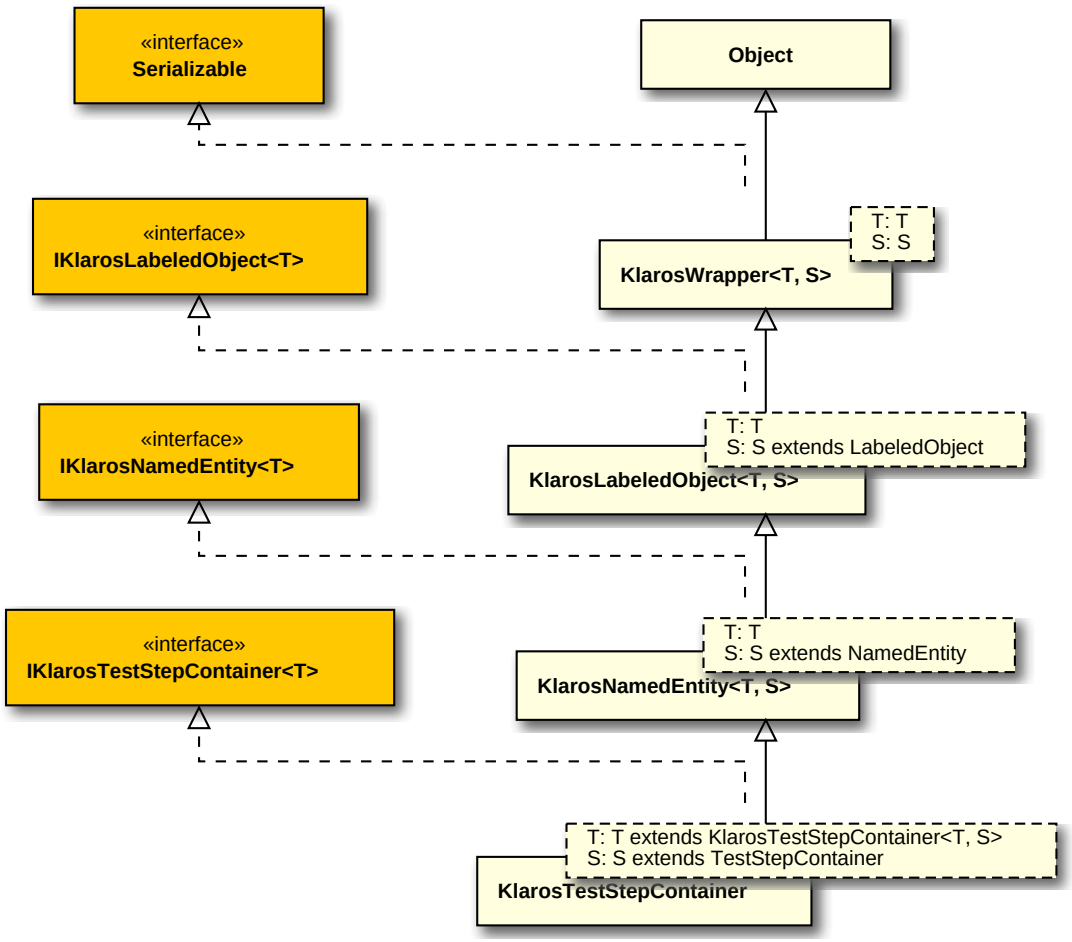
Direct known subclasses: `de.verit.klaros.core.model.KlarosTestCaseStep`

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `getPropertyValue` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.68. KlarosTestSuite

This class provides access to the information stored for test suites.

B.1.1.68.1. Synopsis

```
public final class KlarosTestSuite extends
                                KlarosRevision<KlarosTestSuite, TestSuite>

  implements IKlarosTestSuite<KlarosTestSuite, TestSuite> {

 // Public Methods
 public Set<KlarosAttachment> getAttachments();
 public Set<KlarosCategoryNode> getCategories();
 public KlarosConfiguration getConfiguration();
 public KlarosTestSuite getPredecessor();
 public List<KlarosTestSuiteResult> getResults();
 public KlarosTestSuite getRoot();
 public String getShortname();
 public KlarosTestSuite getSuccessor();
 public KlarosSUTImplementation getSut();
 public List<KlarosTestCase> getTestCases();
 public Integer getTestSuiteResultCount();
}
```

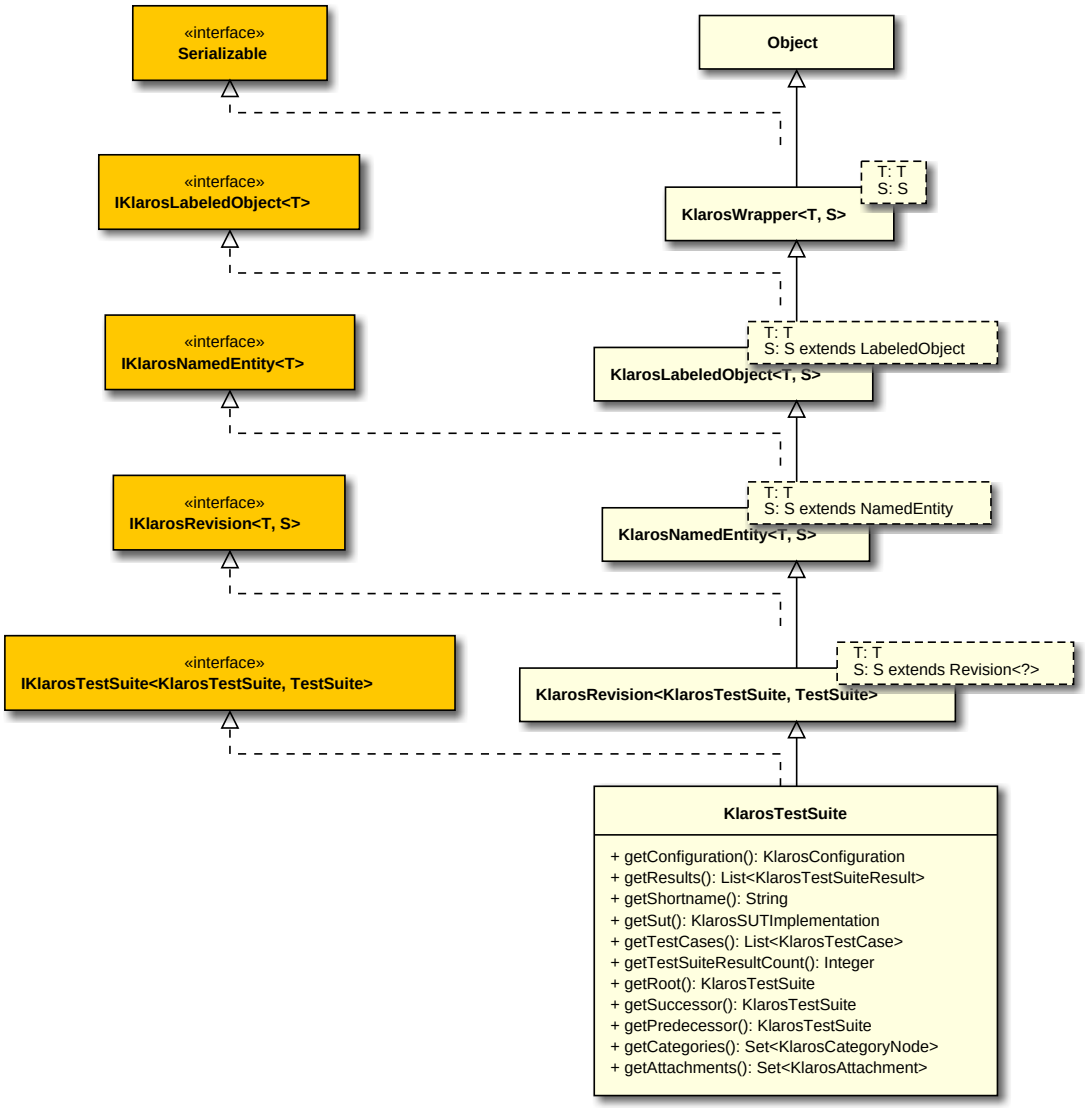
Methods inherited from de.verit.klaros.core.model.KlarosRevision: `getRevisionComment` , `getRevisionId`

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `getPropertyValue` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.69. KlarosTestSuiteResult

This class provides access to the information stored for test suite results.

B.1.1.69.1. Synopsis

```
public final class KlarosTestSuiteResult extends
                                KlarosResult<KlarosTestSuiteResult, TestSuiteResult>

  implements IKlarosTestSuiteResult,
         Comparable<KlarosTestSuiteResult> {

 // Public Methods
 public int compareTo(KlarosTestSuiteResult o);
 public boolean equals(Object o);
 public Set<KlarosTestCaseResult> getResults();
 public KlarosTestRun getTestRun();
 public KlarosTestSuite getTestSuite();
 public int hashCode();
```

```
  public boolean isPending();
}
```

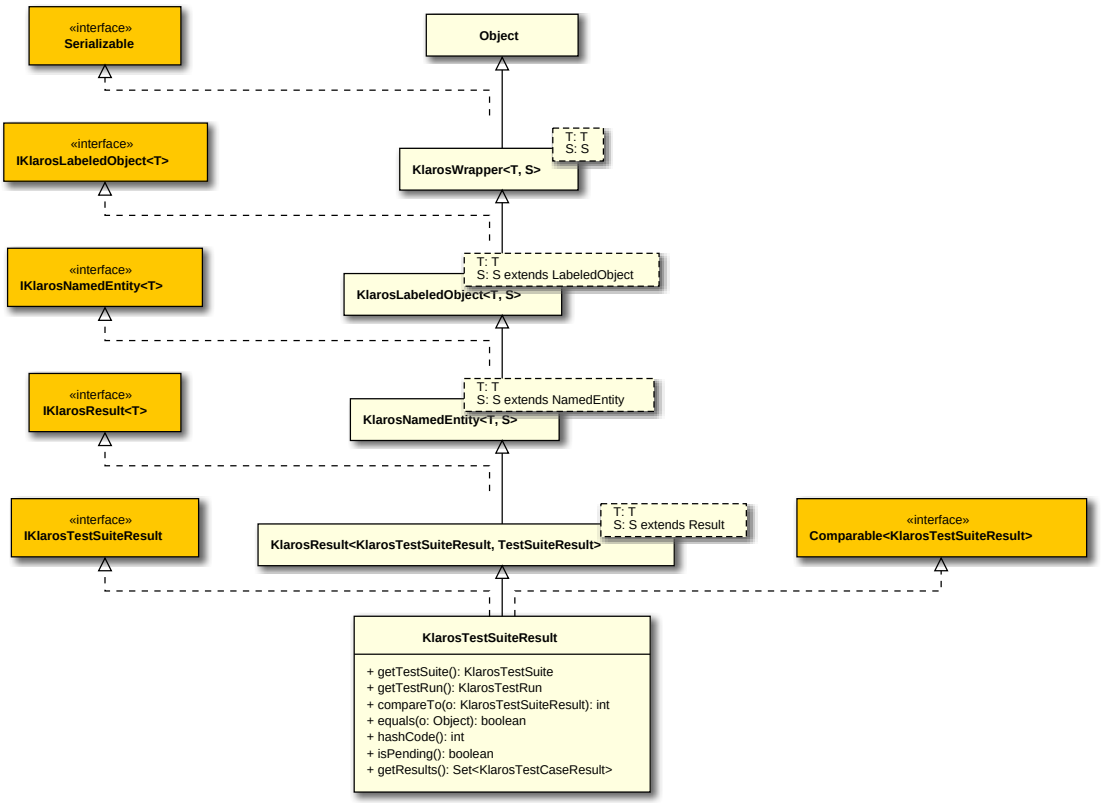
Methods inherited from de.verit.klaros.core.model.KlarosResult: `getConfiguration` , `getDescription` , `getExecutionTime` , `getJob` , `getSummary` , `getVerdict` , `isError` , `isFailure` , `isInconclusive` , `isPassed` , `isSkipped`

Methods inherited from de.verit.klaros.core.model.KlarosNamedEntity: `getAssignee` , `getProperties` , `getPropertyValue` , `isDefinedProperty`

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.1.70. KlarosUser

The user object.

B.1.1.70.1. Synopsis

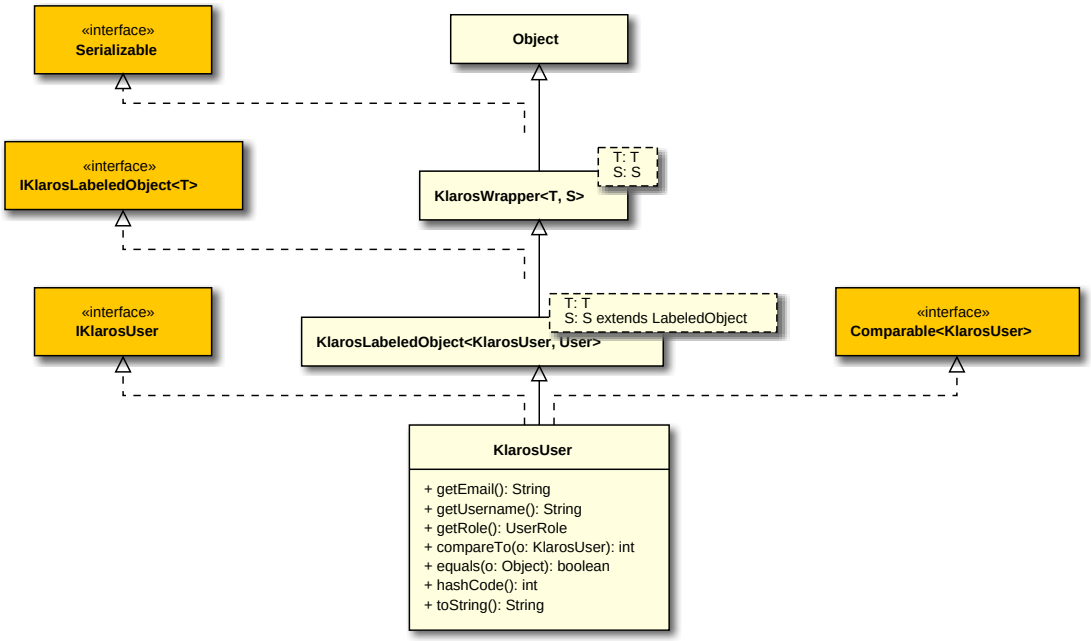
```
public class KlarosUser extends
                              KlarosLabeledObject<KlarosUser, User>

  implements IKlarosUser,
         Comparable<KlarosUser> {
```

```
  // Public Methods
  public int compareTo(KlarosUser o);
  public boolean equals(Object o);
  public String getEmail();
  public UserRole getRole();
  public String getUsername();
  public int hashCode();
  public String toString();
}
```

Methods inherited from de.verit.klaros.core.model.KlarosLabeledObject: `asIterable` , `getCreated` , `getCreator` , `getKey` , `getLastEditor` , `getLastUpdated` , `getName` , `isEnabled`

Methods inherited from de.verit.klaros.core.model.KlarosWrapper: `equals` , `getWrapped` , `hashCode`

Methods inherited from java.lang.Object: `getClass` , `notify` , `notifyAll` , `toString` , `wait`



B.1.2. Deprecated API

B.1.2.1. Deprecated Methods

de.verit.klaros.core.model.IKlarosIssue.getAcceptedIn()	use getSystemsUnderTest

B.2. Scripting API Reference

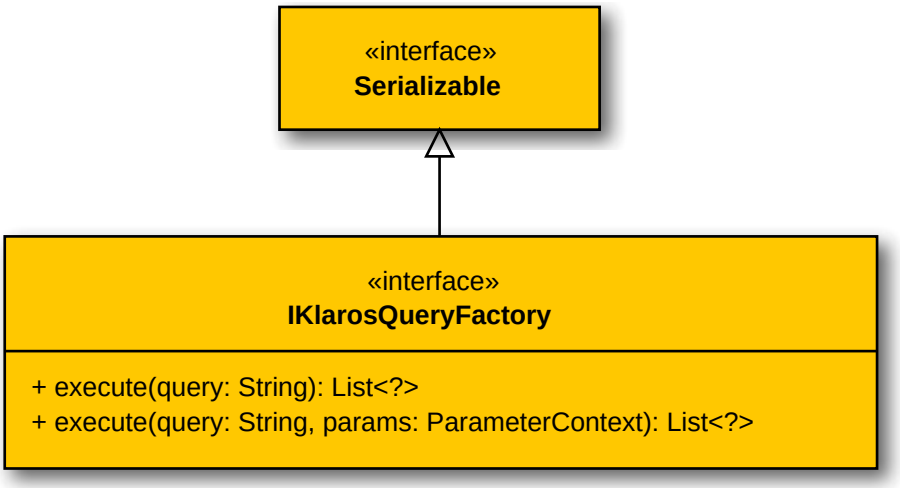
B.2.1. de.verit.klaros.scripting

B.2.1.1. IKlarosQueryFactory

A factory for executing Klaros queries.

B.2.1.1.1. Synopsis

```
public interface IKlarosQueryFactory extends Serializable {

  // Public Methods
  public abstract List<?> execute(String query);
  public abstract List<?> execute(String query, ParameterContext params);
}
```



B.2.1.1.2. execute(String)

```
public abstract List<?> execute(String query);
```

Prepares the given query string with our KlarosOdaQuery and returns a KlarosList with the results of the query.

Parameters

`query`	the query
`return`	the list of query results

B.2.1.1.3. execute(String, ParameterContext)

```
public abstract List<?> execute(String query, ParameterContext params);
```

Prepares the given query string with our KlarosOdaQuery and returns a KlarosList with the results of the query.

Parameters

`query`	the query
`params`	the parameters

`return` the list

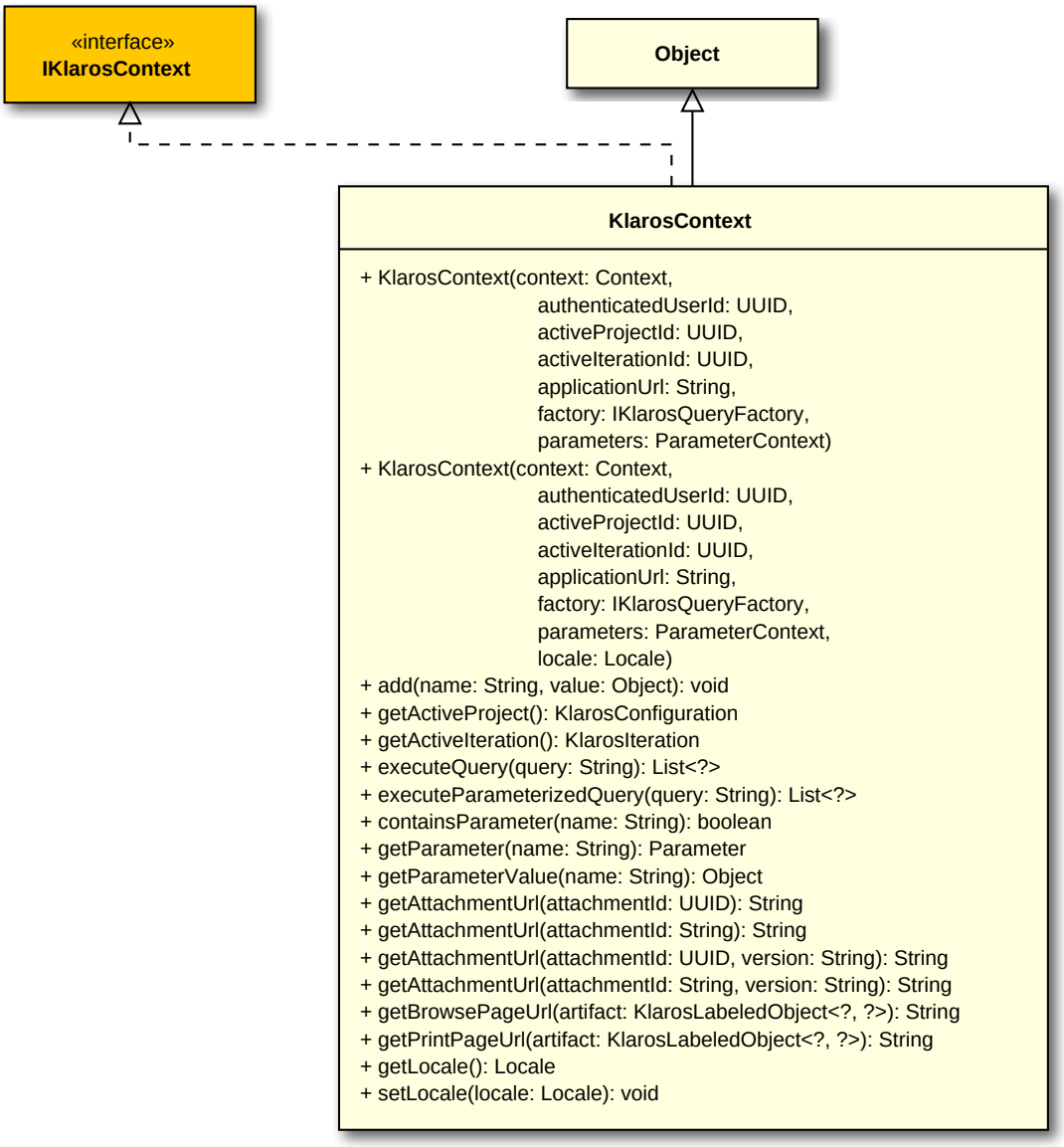
B.2.1.2. KlarosContext

Context to provide all methods to the user to add own objects to the event context.

B.2.1.2.1. Synopsis

```
public class KlarosContext implements IKlarosContext {

  // Public Constructors
  public KlarosContext(Context context,
                  UUID authenticatedUserId,
                  UUID activeProjectId,
                  UUID activeIterationId,
                  String applicationUrl,
                  IKlarosQueryFactory factory,
                  ParameterContext parameters);
  public KlarosContext(Context context,
                  UUID authenticatedUserId,
                  UUID activeProjectId,
                  UUID activeIterationId,
                  String applicationUrl,
                  IKlarosQueryFactory factory,
                  ParameterContext parameters,
                  Locale locale);

  // Public Methods
  public void add(String name, Object value);
  public boolean containsParameter(String name);
  public List<?> executeParameterizedQuery(String query);
  public List<?> executeQuery(String query);
  public KlarosIteration getActiveIteration();
  public KlarosConfiguration getActiveProject();
  public String getAttachmentUrl(String attachmentId);
  public String getAttachmentUrl(String attachmentId, String version);
  public String getAttachmentUrl(UUID attachmentId);
  public String getAttachmentUrl(UUID attachmentId, String version);
  public String getBrowsePageUrl(KlarosLabeledObject<?, ?> artifact);
  public Locale getLocale();
  public Parameter getParameter(String name);
  public Object getParameterValue(String name);
  public String getPrintPageUrl(KlarosLabeledObject<?, ?> artifact);
  public void setLocale(Locale locale);
}
```

Methods inherited from java.lang.Object: `equals` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`



B.2.1.2.2. KlarosContext(Context, UUID, UUID, UUID, String, IKlarosQueryFactory, ParameterContext)

```
public KlarosContext(Context context,
                     UUID authenticatedUserId,
                     UUID activeProjectId,
                     UUID activeIterationId,
                     String applicationUrl,
                     IKlarosQueryFactory factory,
                     ParameterContext parameters);
```

Create a KlarosContext.

Parameters

`context`	Gets passed by the relating servlet.
`authenticatedUserId`	the authenticated user id

activeProjectId	the active project id
activeIterationId	the active iteration id if available
applicationUrl	the application url
factory	the factory
parameters	the parameters

B.2.1.2.3. KlarosContext(Context, UUID, UUID, UUID, String, IKlarosQueryFactory, ParameterContext, Locale)

```
public KlarosContext(Context context,
                     UUID authenticatedUserId,
                     UUID activeProjectId,
                     UUID activeIterationId,
                     String applicationUrl,
                     IKlarosQueryFactory factory,
                     ParameterContext parameters,
                     Locale locale);
```

Create a KlarosContext.

Parameters

context	Gets passed by the relating servlet.
authenticatedUserId	the authenticated user id
activeProjectId	the active project id
activeIterationId	the active iteration id
applicationUrl	the application url
factory	the factory
parameters	the parameters
locale	the locale

B.2.2. de.verit.klaros.scripting.context

B.2.2.1. IKlarosContext

The context object encapsulates the input/output interface a report script can access.
<orderedlist>
<listitem>

The parameters given before starting the report.
</listitem>
<listitem>

An HQL query interface for retrieving model objects from the database.
</listitem>
<listitem>

A writable context for storing values for template access.
</listitem>
<listitem>

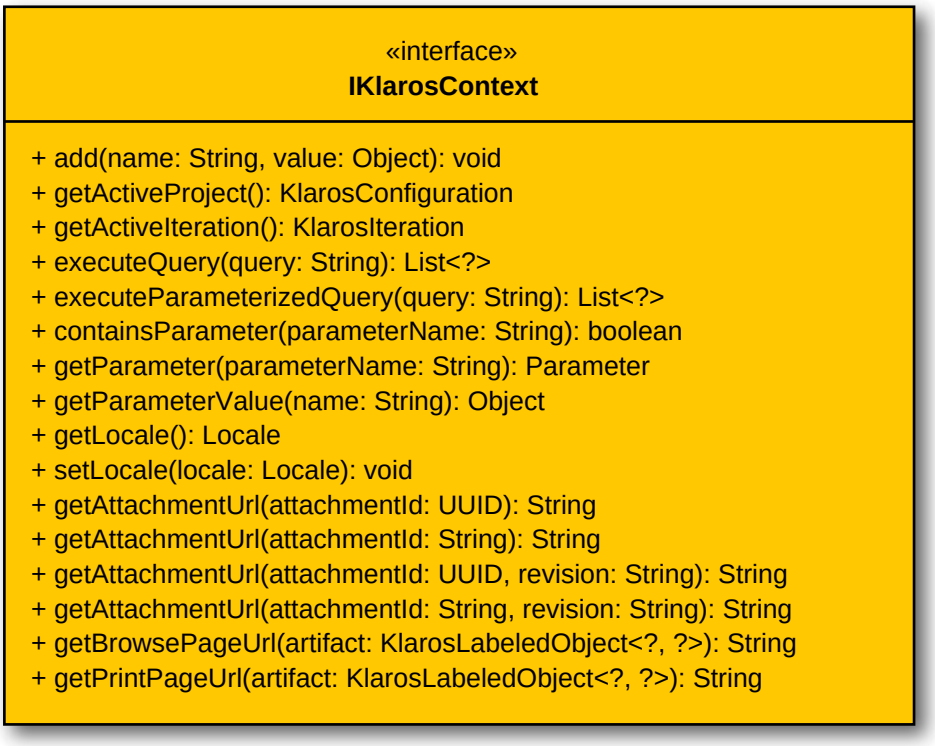
Access to current Klaros settings (current project, current iteration, locale settings)
</listitem>
<listitem>

URL conversions for generating URLs to attachments, page and print page objects inside the application.
</listitem>
</orderedlist>

The following entries are available

B.2.2.1.1. Synopsis

```
public interface IKlarosContext {

  // Public Methods
  public abstract void add(String name, Object value);
  public abstract boolean containsParameter(String parameterName);
  public abstract List<?> executeParameterizedQuery(String query);
  public abstract List<?> executeQuery(String query);
  public abstract KlarosIteration getActiveIteration();
  public abstract KlarosConfiguration getActiveProject();
  public abstract String getAttachmentUrl(String attachmentId);
  public abstract String getAttachmentUrl(String attachmentId, String revision);
  public abstract String getAttachmentUrl(UUID attachmentId);
  public abstract String getAttachmentUrl(UUID attachmentId, String revision);
  public abstract String getBrowsePageUrl(KlarosLabeledObject<?, ?> artifact);
  public abstract Locale getLocale();
  public abstract Parameter getParameter(String parameterName);
  public abstract Object getParameterValue(String name);
  public abstract String getPrintPageUrl(KlarosLabeledObject<?, ?> artifact);
  public abstract void setLocale(Locale locale);
}
```



B.2.2.1.2. add(String, Object)

```
public abstract void add(String name, Object value);
```

Add a new object with the give key to the event context.

Parameters

`name`	the name
`value`	the value

B.2.2.1.3. containsParameter(String)

```
public abstract boolean containsParameter(String parameterName);
```

Checks if the parameter is available in the context.

Parameters

`parameterName`	the parameter name
`return`	true, if successful

B.2.2.1.4. executeParameterizedQuery(String)

```
public abstract List<?> executeParameterizedQuery(String query);
```

Execute the given query with the Parameters from the ParameterContext.

Parameters

`query`	The HQL query to execute.

`return`	A KlarosList of the selected objects.

B.2.2.1.5. executeQuery(String)

```
public abstract List<?> executeQuery(String query);
```

Execute the given query.

Parameters

`query`	The HQL query to execute.
`return`	A KlarosList of the selected objects.

B.2.2.1.6. getActiveIteration()

```
public abstract KlarosIteration getActiveIteration();
```

Gets the active iteration.

Parameters

`return`	the active iteration

B.2.2.1.7. getActiveProject()

```
public abstract KlarosConfiguration getActiveProject();
```

Gets the active project.

Parameters

`return`	the active project

B.2.2.1.8. getAttachmentUrl(String)

```
public abstract String getAttachmentUrl(String attachmentId);
```

Return an URL string to retrieve the attachment with given attachment id.

Parameters

`attachmentId`	the attachment id
`return`	the attachment url

B.2.2.1.9. getAttachmentUrl(String, String)

```
public abstract String getAttachmentUrl(String attachmentId, String revision);
```

Return an URL string to retrieve the attachment with given attachment id string and revision.

Parameters

`attachmentId`	the attachment id
`revision`	the revision
`return`	the attachment url

B.2.2.1.10. getAttachmentUrl(UUID)

```
public abstract String getAttachmentUrl(UUID attachmentId);
```

Return an URL string to retrieve the attachment with given attachment UUID.

Parameters

`attachmentId`	the attachment id
`return`	the attachment url

B.2.2.1.11. getAttachmentUrl(UUID, String)

```
public abstract String getAttachmentUrl(UUID attachmentId, String revision);
```

Return an URL string to retrieve the attachment with given attachment UUID and revision.

Parameters

`attachmentId`	the attachment id
`revision`	the revision string
`return`	the attachment url

B.2.2.1.12. getBrowsePageUrl(KlarosLabeledObject<?, ?>)

```
public abstract String getBrowsePageUrl(KlarosLabeledObject<?, ?> artifact);
```

Gets the browse page url to the given klaros artifact. Will return null if the artifact type is not supported.

Parameters

`artifact`	the artifact
`return`	the browse page url

B.2.2.1.13. getLocale()

```
public abstract Locale getLocale();
```

/** Returns the active Java Locale setting.

Parameters

`return`	the Java Locale

B.2.2.1.14. getParameter(String)

```
public abstract Parameter getParameter(String parameterName);
```

Gets the parameter.

Parameters

`parameterName`	the parameter name
`return`	The parameter with the given name

Exceptions

`Klaros-Illegal-`	if the parameter is not available.

Argument-
Exception

B.2.2.1.15. getParameterValue(String)

```
public abstract Object getParameterValue(String name);
```

Gets the parameter value.

Parameters

name	the name
return	The parameter with the given name

Exceptions

Klaros-Illegal-Argument-Exception	if the parameter is not available.

B.2.2.1.16. getPrintPageUrl(KlarosLabeledObject<?, ?>)

```
public abstract String getPrintPageUrl(KlarosLabeledObject<?, ?> artifact);
```

Gets the print page of the given klaros artifact. Will return null if the artifact type is not supported.

Parameters

artifact	the artifact
return	the print page url

B.2.2.1.17. setLocale(Locale)

```
public abstract void setLocale(Locale locale);
```

Sets the active Java Locale setting.

Parameters

locale	the new locale

B.2.3. de.verit.klaros.scripting.model

B.2.3.1. Parameter

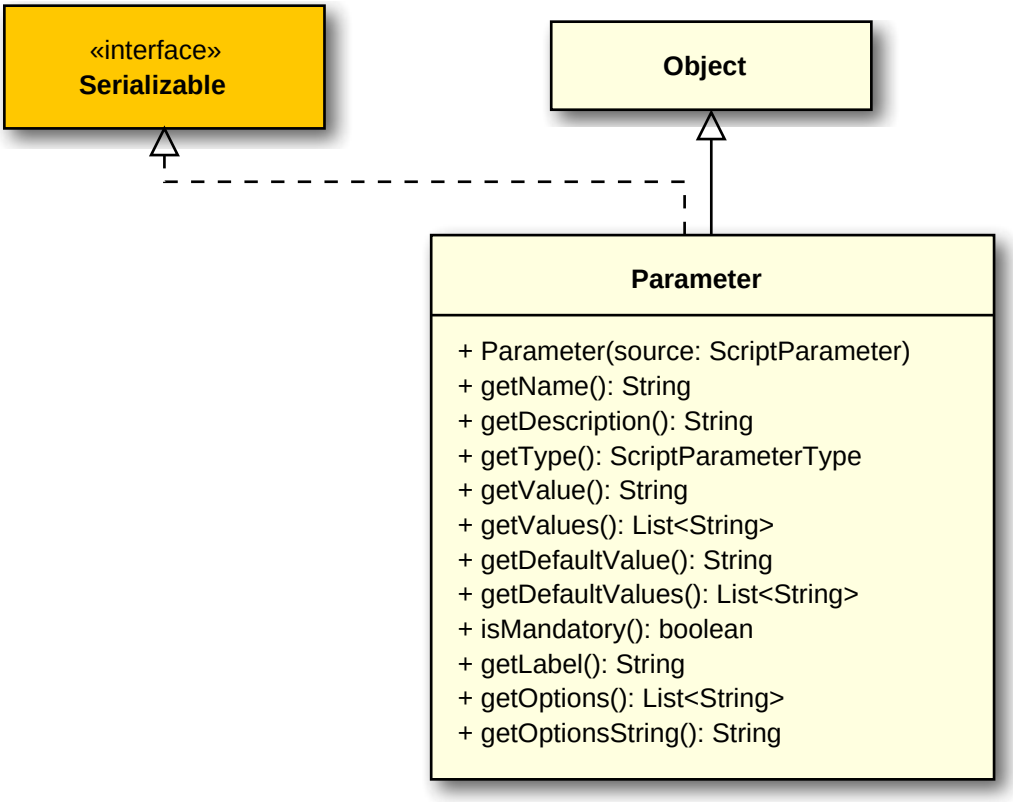
The Parameter class encapsulates a report query parameter.

B.2.3.1.1. Synopsis

```
public class Parameter implements Serializable {

  // Public Constructors
  public Parameter(ScriptParameter source);

  // Public Methods
  public String getDefaultValue();
```

```
  public List<String> getDefaultValues();
  public String getDescription();
  public String getLabel();
  public String getName();
  public List<String> getOptions();
  public String getOptionsString();
  public ScriptParameterType getType();
  public String getValue();
  public List<String> getValues();
  public boolean isMandatory();
}
```

Methods inherited from java.lang.Object: `equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`



B.2.3.1.2. Parameter(ScriptParameter)

```
public Parameter(ScriptParameter source);
```

Instantiates a new parameter.

Parameters

`source`	the source script parameter

B.2.3.1.3. getDefaultValue()

```
public String getDefaultValue();
```

Gets the default value.

Parameters

`return` the default value

B.2.3.1.4. getDefaultValues()

```
public List<String> getDefaultValues();
```

Gets the default values.

Parameters

`return` the default values

B.2.3.1.5. getDescription()

```
public String getDescription();
```

Gets the description.

Parameters

`return` the description

B.2.3.1.6. getLabel()

```
public String getLabel();
```

Gets the label.

Parameters

`return` the label

B.2.3.1.7. getName()

```
public String getName();
```

Gets the name.

Parameters

`return` the name

B.2.3.1.8. getOptions()

```
public List<String> getOptions();
```

Gets the options.

Parameters

`return` the options

B.2.3.1.9. getOptionsString()

```
public String getOptionsString();
```

Gets the options string.

Parameters

`return` the options string

B.2.3.1.10. getType()

```
public ScriptParameterType getType();
```

Gets the type.

Parameters

return	the type

B.2.3.1.11. getValue()

```
public String getValue();
```

Gets the value.

Parameters

return	the value

B.2.3.1.12. getValues()

```
public List<String> getValues();
```

Gets the value.

Parameters

return	the value

B.2.3.1.13. isMandatory()

```
public boolean isMandatory();
```

Checks if is mandatory.

Parameters

return	true, if is mandatory

Appendix C. Test Case Import File Specification

The following table gives a short overview of the available elements:

Name	Description
<areatopic>	(Test Type) A group of test activities based on specific test objectives with the purpose of testing a component or system for specific characteristics (FUNCTIONAL, NON_FUNCTIONAL, STRUCTURAL, REGRESSION or RE_TEST).
<attachment>	An attachment.
<attachments>	The list of attachments.
<attribute>	A custom field.
<attributes>	The list of user-defined custom fields.
<category>	A node of a category tree.
<categoryDescription>	The description of the category.
<categoryName>	The name of the category.
<categoryTree>	A category tree.
<container>	The root element of this schema, which contains all the test cases and optionally a test suite/category tree.
<content>	The base64 encoded content of the attachment.
<depends>	The dependencies of the test case in text form.
<description>	The textual description of the test case.
<docbase>	The reference document containing the requirement on which the test case is based.
<estimatedDuration>	The estimated duration in milli seconds.
<evaluation>	The evaluation method of this test case (MANUAL or AUTO).
<execution>	The execution type of this test case (MANUAL or AUTO).
<expectedResult>	The expected result.
<externalId>	The external id of this test case.
<level>	The test level of this test case (COMPONENT, INTEGRATION, SYSTEM or ACCEPTANCE).
<method>	(Design Technique) The test design view on the system under test (BLACK_BOX or WHITE_BOX).
<name>	The name of the user-defined custom field.
<note>	A placeholder for any short notes about the test case.
<postcondition>	A text describing the postcondition.
<precondition>	A text describing the precondition.
<priority>	The priority of this test case (LOW, MEDIUM or HIGH).
<revision>	The revision of this test case.

Name	Description
<shortname>	The name of the test case.
<state>	The test case state (not used, reserved for future use).
<step>	A test case step.
<steps>	The list of test case steps.
<team>	The team that is responsible for this test case.
<testcase>	A test case.
<testcases>	The list of test cases.
<testsuite>	If this element is present, a test suite with the name given here will be generated upon import.
<traceability>	A reference to the associated requirement, use case, or work package.
<value>	The value of the user-defined custom field.
<variety>	The result expectation of the test (POSITIVE or NEGATIVE).

Table C.1. Element summary

C.1. <step>

A test case step.

This element may occur inside the following elements: <steps> .

Name	Multiplicity	Description
<description>	0..1	The action to take in this test case step.
<expectedResult>	0..1	The expected result.
<postcondition>	0..1	The postcondition.
<precondition>	0..1	The precondition.

Table C.2. <step> elements

C.2. <steps>

The list of test case steps.

This element may occur inside the following elements: <testcase> .

Name	Multiplicity	Description
<step>	0..n	A test step.

Table C.3. <steps> elements

C.3. <attachments>

The list of attachments.

This element may occur inside the following elements: <testcase> .

Name	Multiplicity	Description
<attachment>	0..n	A binary attachment.

Table C.4. <attachments> elements

C.4. <attachment>

An attachment.

This element may occur inside the following elements: <attachments> .

Name	Multiplicity	Description
<name>	1..1	The name of the attachment.
<content>	1..1	The attachment binary content, encoded in - Base-64 format.

Table C.5. <attachment> elements

C.5. <attribute>

A custom field.

This element may occur inside the following elements: <attributes> .

Name	Multiplicity	Description
<name>	1..1	The name of the custom field.
<value>	1..1	The value of the custom field.

Table C.6. <attribute> elements

C.6. <attributes>

The list of user-defined custom fields.

This element may occur inside the following elements: <testcase> .

Name	Multiplicity	Description
<attribute>	0..n	

Table C.7. <attributes> elements

C.7. <category>

A node of a category tree. It can contain further subcategories.

This element may occur inside the following elements: <category> , <categoryTree> , <testcase> .

Name	Multiplicity	Description
<categoryName>	1..1	The name of the category node.
<categoryDescription>	1..1	The description of the category node.
<category>	0..1	The parent category node.

Table C.8. <category> elements

C.8. <categoryTree>

A category tree. It contains the root category node with an arbitrary nomber of sub categories.

This element may occur inside the following elements: <container> .

Name	Multiplicity	Description
<categoryName>	1..1	The name of the category tree.
<category>	1..1	The root category tree node.

Table C.9. <categoryTree> elements

C.9. <container>

The root element of this schema, which contains all the test cases and optionally a test suite/ category tree.

Name	Multiplicity	Description
<testsuite>	0..1	An optional test suite containing the listed test cases.
<testcases>	0..1	The list of test cases.
<categoryTree>	0..1	The optional category tree of these requirements.

Table C.10. <container> elements

C.10. <testcases>

The list of test cases.

This element may occur inside the following elements: <container> .

Name	Multiplicity	Description
<testcase>	0..n	A test case.

Table C.11. <testcases> elements

C.11. <testcase>

A test case.

This element may occur inside the following elements: <testcases> .

Name	Multiplicity	Description
<areatopic>	0..1	The test case type (FUNCTIONAL, NON_FUNCTIONAL, STRUCTURAL, REGRESSION or RE_TEST).
<attachments>	0..1	The list of binary attachments.
<attributes>	0..1	The custom fields this test case has.
<category>	0..1	The category node to which this test case is assigned.
<depends>	0..1	The dependencies of the test case in text form.
<description>	0..1	The textual description of the test case.
<docbase>	0..1	The reference document containing the requirement on which the test case is based.
<estimatedDuration>	0..1	The estimated duration in milli seconds for executing this test case.

Name	Multiplicity	Description
<evaluation>	0..1	The evaluation method of this test case (MANUAL or AUTO).
<execution>	0..1	The execution type of this test case (MANUAL or AUTO).
<expectedResult>	0..1	The expected result.
<externalId>	0..1	The external id of this test case. This is required when referencing test cases in requirement imports.
<level>	0..1	The test level of this test case (COMPONENT, INTEGRATION, SYSTEM or ACCEPTANCE).
<method>	0..1	(Design Technique) The test design view on the system under test (BLACK_BOX or WHITE_BOX).
<note>	0..1	Any short note about the test case.
<postcondition>	0..1	The postcondition.
<precondition>	0..1	The precondition.
<priority>	0..1	The priority of this test case (LOW, MEDIUM or HIGH).
<revision>	0..1	The revision of this test case.
<shortname>	0..1	The name of the test case.
<state>	0..1	The test case state (not used, reserved for future use).
<steps>	0..1	The list of test case steps.
<team>	0..1	The team that is responsible for this test case.
<traceability>	0..1	A reference to the associated requirement, use case, or work package.
<variety>	0..1	The result expectation of the test (POSITIVE or NEGATIVE).

Table C.12. <testcase> elements

C.12. <testsuite>

If this element is present, a test suite with the name given here will be generated upon import. This test suite will contain all the test cases that are part of this import container.

This element may occur inside the following elements: <container> .

C.13. <externalId>

The external id of this test case. This is required when referencing test cases in requirement imports.

This element may occur inside the following elements: <testcase> .

C.14. <areatopic>

(Test Type) A group of test activities based on specific test objectives with the purpose of testing a component or system for specific characteristics (FUNCTIONAL, NON_FUNCTIONAL, STRUCTURAL, REGRESSION or RE_TEST).

This element may occur inside the following elements: <testcase> .

C.15. <depends>

The dependencies of the test case in text form.

This element may occur inside the following elements: <testcase> .

C.16. <description>

The textual description of the test case.

This element may occur inside the following elements: <step> , <testcase> .

C.17. <docbase>

The reference document containing the requirement on which the test case is based.

This element may occur inside the following elements: <testcase> .

C.18. <estimatedDuration>

The estimated duration in milli seconds.

This element may occur inside the following elements: <testcase> .

C.19. <evaluation>

The evaluation method of this test case (MANUAL or AUTO).

This element may occur inside the following elements: <testcase> .

C.20. <execution>

The execution type of this test case (MANUAL or AUTO).

This element may occur inside the following elements: <testcase> .

C.21. <expectedResult>

The expected result.

This element may occur inside the following elements: <step> , <testcase> .

C.22. <level>

The test level of this test case (COMPONENT, INTEGRATION, SYSTEM or ACCEPTANCE).

This element may occur inside the following elements: <testcase> .

C.23. <method>

(Design Technique) The test design view on the system under test (BLACK_BOX or WHITE_BOX).

This element may occur inside the following elements: <testcase> .

C.24. <note>

A placeholder for any short notes about the test case.

This element may occur inside the following elements: <testcase> .

C.25. <postcondition>

A text describing the postcondition.

This element may occur inside the following elements: <step> , <testcase> .

C.26. <precondition>

A text describing the precondition.

This element may occur inside the following elements: <step> , <testcase> .

C.27. <revision>

The revision of this test case. This element is used only when exporting test cases, imported test cases always get the fixed revision Id "1.0".

This element may occur inside the following elements: <testcase> .

C.28. <priority>

The priority of this test case (LOW, MEDIUM or HIGH).

This element may occur inside the following elements: <testcase> .

C.29. <shortname>

The name of the test case.

This element may occur inside the following elements: <testcase> .

C.30. <state>

The test case state (not used, reserved for future use).

This element may occur inside the following elements: <testcase> .

C.31. <team>

The team that is responsible for this test case.

This element may occur inside the following elements: <testcase> .

C.32. <traceability>

A reference to the associated requirement, use case, or work package.

This element may occur inside the following elements: <testcase> .

C.33. <variety>

The result expectation of the test (POSITIVE or NEGATIVE).

This element may occur inside the following elements: <testcase> .

C.34. <name>

The name of the user-defined custom field.

This element may occur inside the following elements: <attachment> , <attribute> .

C.35. <value>

The value of the user-defined custom field.

This element may occur inside the following elements: <attribute> .

C.36. <content>

The base64 encoded content of the attachment.

This element may occur inside the following elements: <attachment> .

C.37. <categoryName>

The name of the category.

This element may occur inside the following elements: <category> , <categoryTree> .

C.38. <categoryDescription>

The description of the category.

This element may occur inside the following elements: <category> .

Appendix D. Requirement Import File Specification

The following table gives a short overview of the available elements:

Name	Description
<attachment>	An attachment.
<attachments>	The list of binary attachments.
<attribute>	A custom field.
<attributes>	The list of custom fields.
<category>	A node of a category tree.
<categoryDescription>	The description of the category.
<categoryName>	The name of the category.
<categoryTree>	A category tree.
<container>	The root element of this schema, which contains all requirements.
<content>	The base64 encoded content of the attachment.
<description>	The textual description of the requirement.
<externalId>	The external id of this requirement, the format of which can be freely chosen.
<externalRevision>	The external revision of this requirement (not used, reserved for future use).
<externalTestCaseIds>	The list of external test case ids covering this requirement.
<name>	The name of a field.
<priority>	The priority of this requirement (LOW, MEDIUM or HIGH).
<requirement>	A requirement.
<requirements>	The list of requirements.
<revision>	The revision of this requirement.
<shortname>	The name of the requirement.
<summary>	A short summary of the requirement.
<value>	The value of a field.

Table D.1. Element summary

D.1. <attachments>

The list of binary attachments.

This element may occur inside the following elements: <requirement> .

Name	Multiplicity	Description
<attachment>	0..n	An attachment.

Table D.2. <attachments> elements

D.2. <attachment>

An attachment.

This element may occur inside the following elements: <attachments> .

Name	Multiplicity	Description
<name>	1..1	The name of the attachment.
<content>	1..1	The attachment binary content, encoded in - Base-64 format.

Table D.3. <attachment> elements

D.3. <attributes>

The list of custom fields.

This element may occur inside the following elements: <requirement> .

Name	Multiplicity	Description
<attribute>	0..n	A custom field.

Table D.4. <attributes> elements

D.4. <attribute>

A custom field.

This element may occur inside the following elements: <attributes> .

Name	Multiplicity	Description
<name>	1..1	The name of the custom field.
<value>	1..1	The value of the custom field.

Table D.5. <attribute> elements

D.5. <category>

A node of a category tree. It can contain further subcategories.

This element may occur inside the following elements: <category> , <categoryTree> , <requirement> .

Name	Multiplicity	Description
<categoryName>	1..1	The name of the category node.
<categoryDescription>	1..1	The description of the category node.
<category>	0..1	The parent category node.

Table D.6. <category> elements

D.6. <categoryTree>

A category tree. It contains the root category node with an arbitrary nomber of sub categories.

This element may occur inside the following elements: <container> .

Name	Multiplicity	Description
<categoryName>	1..1	The name of the category tree.
<category>	1..1	The root category tree node.

Table D.7. <categoryTree> elements

D.7. <container>

The root element of this schema, which contains all requirements.

Name	Multiplicity	Description
<requirements>	0..1	The list of requirements.
<categoryTree>	0..1	The optional category tree of these requirements.

Table D.8. <container> elements

D.8. <requirements>

The list of requirements.

This element may occur inside the following elements: <container> .

Name	Multiplicity	Description
<requirement>	0..n	A requirement.

Table D.9. <requirements> elements

D.9. <requirement>

A requirement.

This element may occur inside the following elements: <requirements> .

Name	Multiplicity	Description
<attributes>	0..1	The custom fields this requirement has.
<category>	0..1	The category node to which this requirement is assigned.
<description>	0..1	The detailed description of the requirement in text form.
<externalId>	0..1	The external id of this requirement.
<priority>	0..1	The priority of this requirement.
<revision>	0..1	The revision of this requirement.
<shortname>	0..1	The name of the requirement.
<summary>	0..1	A short summary of the requirement.
<attachments>	0..1	The list of binary attachments.
<externalTestCaseIds>	0..1	The list of external test case ids covering this requirement.

Table D.10. <requirement> elements

D.10. <externalTestCaseIds>

The list of external test case ids covering this requirement.

This element may occur inside the following elements: <requirement> .

Name	Multiplicity	Description
<externalId>	0..n	An id of an external test case covering the requirement.

Table D.11. <externalTestCaseIds> elements

D.11. <externalId>

The external id of this requirement, the format of which can be freely chosen. This element is used only for requirement synchronization and is ignored during import. If the id of a requirement is changed before a synchronization, a new requirement with this id will be created in Klaros-Testmanagement. Otherwise, the existing requirement is overwritten with the data from this XML file.

This element may occur inside the following elements: <requirement> .

D.12. <externalRevision>

The external revision of this requirement (not used, reserved for future use).

D.13. <description>

The textual description of the requirement.

This element may occur inside the following elements: <requirement> .

D.14. <priority>

The priority of this requirement (LOW, MEDIUM or HIGH).

This element may occur inside the following elements: <requirement> .

D.15. <revision>

The revision of this requirement. This element is used only when exporting and synchronising requirements, imported requirements always get the fixed revision Id "1.0".

This element may occur inside the following elements: <requirement> .

D.16. <shortname>

The name of the requirement.

This element may occur inside the following elements: <requirement> .

D.17. <summary>

A short summary of the requirement.

This element may occur inside the following elements: <requirement> .

D.18. <name>

The name of a field.

This element may occur inside the following elements: <attachment> , <attribute> .

D.19. <value>

The value of a field.

This element may occur inside the following elements: <attribute> .

D.20. <content>

The base64 encoded content of the attachment.

This element may occur inside the following elements: <attachment> .

D.21. <categoryName>

The name of the category.

This element may occur inside the following elements: <category> , <categoryTree> .

D.22. <categoryDescription>

The description of the category.

This element may occur inside the following elements: <category> .

Appendix E. The Reporting Context API

E.1. The Klaros Report Context

Name	Description	Typ
date	The time of report execution.	`java.util.Date`
locale	The localization selected by the user.	`java.util.Locale`
activeProject	The currently selected project or `null` if no project is selected.	`de.verit.klaros.model.KlarosConfiguration`
activeIteration	The currently selected iteration or `null` if no iteration is selected.	`de.verit.klaros.model.KlarosIteration`
user	The current user who creates the report.	`de.verit.klaros.model.KlarosUser`
parameters	A map of parameter names to parameter objects containing the user defined parameters entered when running the report.	`java.util.Map java.lang.String`, `de.verit.klaros.scripting.model.Parameter`

Table E.1. Context Variables

The context variables can be accessed via SeamPDF by e.g.:

```
<p:text value="#{date}" />
```

Note

#{user.name} und #{user.username} contain different values. The first provides the user's real name, while the latter provides the name the user is logged in with.

E.2. KlarosScript Interface

```
package de.verit.klaros.scripting;
/**
 * Public interface for generating user defined reports.
 */
public interface KlarosScript {

    /**
     * This method starts the report generation.
     *
     * @param context
     *            The event context to provide all needed functions, properties
     *            and objects.
     * @throws KlarosScriptingException if executing the script fails
```

```
     */
    void execute(KlarosContext context) throws KlarosScriptingException;
}
```

E.3. Example Layout Template

```
<p:document xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:f="http://java.sun.com/jsf/core" xmlns:p="http://jboss.org/schema/seam/pdf"
  title="Klaros-Testmanagement Test Suite Report" marginMirroring="true"
  author="#{user.name}" creator="#{user.name}" pageSize="A4">

  <f:facet name="header">
    <p:font size="8">
      <p:header borderWidthBottom="0.1" borderColorBottom="black" borderWidthTop="0" alignment="center">
        <p:text value="Example report - generated #{date} by #{user.name}"/>
      </p:header>
      <p:footer borderWidthTop="0.1" borderColorTop="black" borderWidthBottom="0" alignment="center">
        <p:text value="Page " />
        <p:pageNumber />
      </p:footer>
    </p:font>
  </f:facet>

  <!-- print the frontpage -->
  <p:paragraph alignment="center" spacingAfter="100">
    <p:text value="" />
  </p:paragraph>

  <p:font style="bold" size="32">
    <p:paragraph alignment="center" spacingAfter="75">
      <p:text value="Test Case Report" />
    </p:paragraph>
  </p:font>

  <p:font style="normal" size="12">
    <p:paragraph alignment="center" spacingAfter="5">
      <p:text value="Created by" />
    </p:paragraph>
  </p:font>

  <p:font style="bold" size="16">
    <p:paragraph alignment="center" spacingAfter="5">
      <p:text value="#{user.name} (#{user.email})"/>
    </p:paragraph>
  </p:font>

  <p:font style="normal" size="12">
    <p:paragraph alignment="center" spacingAfter="5">
      <p:text value="at" />
    </p:paragraph>
  </p:font>

  <p:font style="bold" size="16">
    <p:paragraph alignment="center" spacingAfter="75">
      <p:text value="#{date}" />
    </p:paragraph>
  </p:font>
  <p:newPage/>

  <ui:fragment rendered="#{results != null}">
    <p:font style="normal" size="12">
```

```
    <p:paragraph alignment="left" spacingAfter="10">
      <p:text value="The test results for " />
    </p:paragraph>
  </p:font>
</ui:fragment>

<!-- Test result table -->
<p:table columns="4" widths="1 1 3 3">
  <!-- create the headline with bold characters -->
  <p:font size="10" style="bold">

    <p:cell horizontalAlignment="center" verticalAlignment="top">
      <p:paragraph>
        <p:text value="Name" />
      </p:paragraph>
    </p:cell>

    <p:cell horizontalAlignment="center" verticalAlignment="top">
      <p:paragraph>
        <p:text value="Result" />
      </p:paragraph>
    </p:cell>

    <p:cell horizontalAlignment="center" verticalAlignment="top">
      <p:paragraph>
        <p:text value="Test run description" />
      </p:paragraph>
    </p:cell>

    <p:cell horizontalAlignment="center" verticalAlignment="top">
      <p:paragraph>
        <p:text value="Summary" />
      </p:paragraph>
    </p:cell>
  </p:font>

  <!-- display the attributes of the test results -->
  <p:font size="8">
    <ui:repeat value="#{results}" var="tcr">

      <p:cell verticalAlignment="top" horizontalAlignment="left">
        <p:paragraph>
          <p:text value="#{tcr.testCase.name}" />
        </p:paragraph>
      </p:cell>

      <!-- decide which color has to be displayed, based on the test result -->
      <ui:fragment rendered="#{tcr.error}">
        <p:cell backgroundColor="rgb(255,0,0)" verticalAlignment="top" horizontalAlignment="center">
          <p:paragraph>
            <p:text value="error" />
          </p:paragraph>
        </p:cell>
      </ui:fragment>

      <ui:fragment rendered="#{tcr.failure}">
        <p:cell backgroundColor="rgb(255,215,0)" verticalAlignment="top" horizontalAlignment="center">
          <p:paragraph>
            <p:text value="failure" />
          </p:paragraph>
        </p:cell>
      </ui:fragment>
```

```
      <ui:fragment rendered="#{tcr.passed}">
        <p:cell backgroundColor="rgb(0,255,0)" verticalAlignment="top" horizontalAlignment="center">
          <p:paragraph>
            <p:text value="passed" />
          </p:paragraph>
        </p:cell>
      </ui:fragment>

      <p:cell verticalAlignment="top" horizontalAlignment="left">
        <p:paragraph>
          <p:text value="#{tcr.description}" />
        </p:paragraph>
      </p:cell>

      <p:cell verticalAlignment="top" horizontalAlignment="left">
        <p:paragraph>
          <p:text value="#{tcr.summary}" />
        </p:paragraph>
      </p:cell>

      <!-- Print the test case description below the result row.
           To differ from the next row use a bigger border for the bottom. -->
      <p:cell colspan="4" verticalAlignment="top" horizontalAlignment="left"
              borderWidthBottom="1" paddingBottom="3">
        <p:paragraph>
          <p:font size="6" style="bold">
            <p:text value="Testcase description:" />
          </p:font>
          <p:font size="6">
            <p:text value="#{tcr.testCase.description}" />
          </p:font>
        </p:paragraph>
      </p:cell>
    </ui:repeat>
  </p:font>
 </p:table>
</p:document>
```

Appendix F. Icon Index

F.1. Sections

Icon	Description
	Define
	Plan
	Execute
	Evaluate
	Configure

Table F.1. Sections

F.2. Actions

F.2.1. Actions

Icon	Description
	Activate object
	View object
	Edit object
	Duplicate object
	Delete object
	Restore object
	Purge object
	Print object
	Refresh object
	Create revision
	Add item
	Remove item
	Clear item
	Assign to category
	Limit display to active iteration
	Open external link
	Connection check
	URL validation
	Change the role of the user in a project
	Authentication check
	Bookmark the page

Icon	Description
	Start a download
	Send Email
	Open the filter section
	Merge objects
	Remove duplicate objects
	Assign a test case or test suite to a job
	Show all entries
	Show only already executed entries
	Show only entries that have not yet been executed
	Show only the most recently executed entries

Table F.2. Common Actions

F.2.2. Execution Actions

Icon	Description
	Execute object
	Manually execute object
	Execution pending
	Execute object review
	Reopen or update the object
	Import test result

Table F.3. Execution Actions

F.2.3. Test Execution Actions

Icon	Description
	Previous step
	Skip step
	Skip all remaining steps and finish test
	Pause execution
	Edit step result
	Request test review
	Create new issue
	Link existing issue
	Step passed
	Step failed
	Error during step execution
	Step result inconclusive

Icon	Description
	Step skipped

Table F.4. Test Execution Actions

F.2.4. Arranging Actions

Icon	Description
	Move objects up on the same hierarchy level
	Move objects down on the same hierarchy level
	Move objects to a selected position on the same hierarchy level
	Move objects one hierarchy level down
	Move objects one hierarchy level up
	Inserts a test segment before test case step
	Inserts a test segment after test case step
	Expand Window
	Compress Window
	Expand items
	Collapse items
	Insert a test step to the right
	Insert a test step to the left
	Insert a test segment to the right
	Insert a test segment to the left

Table F.5. Arranging Actions

F.3. Table Operations

Icon	Description
	Opens the category section
	Filter / Sort
	Apply quick filter
	Export table
	Configure columns

Table F.6. Table Operations

F.4. Information

Icon	Description
	Information
	Warning

Icon	Description
	Error
	Delete disabled
	Linked with other objects
	Issues exist for this object
	Object is synchronized from external source
	Synchronized object has pending updates
	Hint
	Job is blocked
	Job has a pending dependency

Table F.7. Information

F.5. Properties

F.5.1. Results

Icon	Description
	Passed
	Failed
	Error
	Step result inconclusive
	Skipped

Table F.8. Results

F.5.2. Priorities

Icon	Description
	Blocker
	Critical
	High
	Low
	Trivial

Table F.9. Priorities

F.6. Document Formats

Icon	Description
	CSV
	Excel

Icon	Description
	HTML
	PDF
	XML

Table F.10. Document Formats

Glossary

A

Administrator	A user role that has full access to all functions. This includes installation and maintenance of the software, updates, system settings, user administration, backups and all rights to create, execute and evaluate tests.
Authentication	Users are usually authenticated via user name and password. In Klaros-Testmanagement the user authentication can be done directly in the software or via an external LDAP or Active Directory directory.
Automated Test Cases	The test results from automation tools are imported via result files into Klaros-Testmanagement. If both manual and automated automated test results are available, they can be evaluated together.

B

Bugzilla	Bugzilla is an open source issue management system. Detailed information can be found on the Bugzilla Web Site. Klaros-Testmanagement provides an interface to Bugzilla.

C

Change Tracking	Complete tracking of all changes to managed objects through automatic logging of modifications. In Klaros-Testmanagement, all changes are highlighted in color on the detail page of the corresponding Object over time.
Category	All objects can be arranged in categories for better overview. Grouping of objects (requirements, iterations, test systems, test environments, test cases or test suites) into a category tree can be done according to self-selected criteria. Multiple categorizations are also possible.
Community Edition	The free edition of Klaros test management. It has a limited feature set, but can also be used freely for commercial purposes. See also Enterprise">Enterprise Edition.
Compliance	The capability of the software product to adhere to standards, conventions or regulations in laws and similar prescriptions [ISO9126].
Compliance Rate	In Klaros Test Management, the conformance rate shows how many test cases assigned to a requirement have been executed with the result "Passed".

Continuous Integration	Continuous Integration (CI) is the automated integration of code changes from multiple contributors into a single software project. It is an important DevOps best practice that enables developers to regularly merge code changes into a central repository where builds and tests are then executed. A plug-in for the Jenkins Continuous Integration Server automatically transfers automatically transfers the test results of a build to Klaros Test Management, where they can be are evaluated.
Coverage, Requirements Coverage	The requirement coverage represents the ratio of requirements covered by tests to the total set of defined requirements. In Klaros Test Management, requirements can be directly entered and assigned to test cases. assigned to test cases. Individual requirements can be covered by several tests and one test can cover more than one requirement.

D

Dashboard	A dashboard displays a collection of information, typically in the form of charts. In Klaros-Testmanagement dashboards display the evaluations for a specific test project. The displayed diagrams can be individually combined to form a dashboard. Each user can compile as many dashboards as desired. Dashboards can be private (visible only to the creator) or public. Administrators can create default dashboards.
Data Base	A database is a collection of information organized in interrelated tables of data and specifications of data objects. Klaros-Testmanagement requires a database system to store the objects and is delivered with a preconfigured Apache Derby database. Other supported database systems are MariaDB, Microsoft SQLServer, MySQL and PostgresSQL.
Details Page	Each object (such as test cases, requirements, iterations, etc.) has its own detail page with various other subviews. These vary depending on the object and show detailed information for example “ Properties”, “Attachments”, “Changes” and “Results” an.
Docker	Docker is free software for isolating applications through container virtualization. The containers contain all necessary packages and can thus be easily transported and installed as files. Klaros Test Management can also be run as a container within a Docker-based environment. Ready-to-use Docker images for various databases are already available. already available. Detailed documentation is available at GitHub.

E

Enterprise Edition	The commercial edition with full support is licensed per user. The minimum installation supports 3 users. A free 30-day demo can be requested Demo.
Environment Variables	Environment variables are dynamically named values that can affect the behavior of running processes on a computer. They are part of the environment in which a process runs. By setting the environment variable KLAROS-HOME before the start of the application, the destination of the Klaros home directory can be moved to another location.
Error (Verdict)	A test execution error occurs when the system cannot execute the test correctly. It should not be confused with a failure.
Evaluation	The type of test result evaluation: manual or automated.

F

Failure (Verdict)	A test is deemed to fail if its actual result does not match its expected result. (ISTQB). It should not be be confused with an error.
Functional Test Design Technique	Procedure to derive and select test cases based on an analysis of the specification of the functionality of a component or system without reference to its internal structure. (ISTQB), also known as black box and white box tests.

G

Guest	A user role with full, but read-only access to all data. A guest can view and save reports in various formats. This role is intended for e.g. project managers, customers or reviewers/reviewers.

I

ID	The ID of an object is created automatically when it is created and cannot be changed later. All IDs consist of an abbreviation for the respective object (e.g. TC for test case, ITR for iteration) and an ascending number. number. “SEG00025” would therefore be the test segment 25.
Inconclusive (Verdict)	A test is inconclusive if it is unclear, if its actual result matches its expected result.
Integration	Integration refers to the connection of external systems. Klaros-Testmanagement has interfaces for integration with issue and requirements management systems, test automation and load testing tools, and and continuous integration servers. There are also interfaces to authentication and e-mail servers.

Issue

An issue is a partial aspect that is required to complete a piece of work. This can be a bug, a requested feature, a task, missing documentation or something similar. An issue is tracked in an issue management system.

Klaros-Testmanagement has interfaces to several issue management systems. This means that errors found can be transferred directly to the connected IMS, without having to leave the application.

Iteration

An iteration is a complete development loop resulting in a release (internal or external) of an executable product, a subset of the final product under development, which grows from iteration to iteration to become the final product. (ISTQB)

In Klaros test management supports agile development principles and manages iterations as dedicated objects.

J

Java

In order to run Klaros-Testmanagement, a Java runtime environment is required. This is delivered with the installation file and used automatically.

JavaScript

JavaScript is a programming language used in web browsers and is required for running Klaros test management. When using JavaScript blockers such as NoScript or similar. When using JavaScript blockers such as NoScript or similar, the execution of JavaScript for Klaros-Testmanagement must be allowed as an exception.

JIRA

JIRA is a popular application by the company Atlassian Software for defect management, troubleshooting, and project management.

Klaros-Testmanagement has an integration with JIRA.

Job

Test activities are planned and logged by means of jobs. The execution and results of the executed jobs are automatically logged and used for evaluation.

L

Latest Success Rate, Report Diagram

The “Last Success Rate” report displays the latest results of executed test cases for the selected combination of test system (one or more) and test environment (one or more) as a chart on the dashboard.

Layout Template

Besides the script, the layout template is the other elementary part of a report template. It describes the structure of the generated report document in an XML-based description language. Layout templates can be adapted to individual requirements.

Log Panel

The log panel displays status messages such as warnings or information are displayed. By default, only the last status message is displayed. Clicking on the + icon opens the history.

M

Mantis

Mantis (MantisBT, Mantis Bug Tracker). is an open source application for defect management and issue tracking in software development.

Klaros test management provides integration with Mantis.

N

Notification Event

Notification events may trigger an automatic e-mail notification to specific users. These are sent out as soon as the designated event has has occurred. Possible notification events are, “Job assigned”, “User account created”, “Test execution failed”, “User account password changed” and “Job ready for execution”.

O

Object

The objects defined in Klaros-Testmanagement are project, test case, test step, test segment, test step result, test case result, test suite, test suite result, test environment, system under test, test run, requirement, iteration and job.

Overview Page

Each object type has an overview page. There the individual objects, such as test cases or iterations, are displayed in tabular form. New objects are also created here created.

P

Passed (Verdict)

A test is passed if its actual result matches its expected result. (ISTQB)

Precondition

Environmental and state conditions that must be fulfilled before the component or system can be executed with a particular test or test procedure. (ISTQB).

Postcondition

Environmental and state conditions that must be fulfilled after the execution of a test or test procedure. (ISTQB)

Print View

All objects can be rendered in a printout-optimized representation. The level of detail can be specified with various parameters.

Project

A project is the main object in Klaros Test Management and contains all other objects assigned to a test project.

Project Overview, Report Diagram

The “Project Overview” report displays the number of executed/not executed test cases or execution definitions and the number of scheduled/executed jobs as a chart on the dashboard.

Q

QF-Test

QF-Test from Quality First Software is a cross-platform GUI test automation tool.

Klaros-Testmanagement integrates with QF-Test and can import the automatically generated test results for further processing.

R

Redmine

Redmine is an open source application for defect management and issue tracking in software development.

Klaros-Testmanagement integrates with Redmine.

Reports

Reports provide information on the status of testing activities, the condition of the software, indications of critical components and enable forward planning. For this purpose, the data from the database are sorted according to certain criteria and prepared in textual and graphical overviews. Examples of reports are “Project overview” and “Test activity” as well as overviews of test progress, test results and test runs.

Klaros-Testmanagement contains numerous predefined, basic reports. Individual reports can be created by the user. All reports are exportable to various formats such as PDF or Excel (also for further editing).

Report Template

A report template predefines a report that can take report parameters and be customized by the user as needed. It consists of a *Script* and a *Layout Template*.

With report templates, ready-made reports are available which can be customized by the user as needed.

REST (Representational State Transfer)

Representational State Transfer (REST) is a software architecture style that uses a subset of HTTP. It is commonly used to provide web services.

Klaros-Testmanagement offers several integration interfaces based on REST. These include read access to all stored information as well as import interfaces for test results, test cases and requirements.

Requirement

A user-required property or capability that software must meet or possess in order to Comply with a contract, standard, specification, or other formal document. [According to IEEE 610].

Requirements can be managed either directly in Klaros Test Management or or synchronized with external sources such as JIRA. A reference to the corresponding requirement, e.g. to a document, can be added to test cases and test suites (traceability).

Result, expected

The behavior predicted by the specification, or another source, of a component or system under specified conditions (ISTQB).

Revision

If a major change is made to an object and the previous version is still needed, a new revision should be created. Only selected objects such as test cases or requirements support this mechanism.

Role	A user role defines the rights of the user. Roles can be defined globally and project project-specific. The following user roles are provided: “Administrator”, “Test Manager”, “Tester” and “Guest”.

S

Selenium	Selenium is a web browser automation tool and is mainly used for testing web apps. Selenium produces JUnit XML compliant test results that can be imported into Klaros test management for further processing.
Script	The script is an elementary component of a report template along with the layout template. It extracts the data intended for the report and is available as a Java class. Scripts can be adapted to individual requirements.
Skipped, Verdict	A test case or a test step was omitted during execution.
System Account	A system access reserved for automated operations such as importing test results. A login via the login page is not possible with this.

T

Test Activity, Report Diagram	The "Test Activity" report shows the test case results for a selected combination of one or more test systems and test environments during a selected in a selected time period as a graph on the dashboard.
Test Case	A test case comprises a set of input values, execution conditions, expected results, and postconditions defined for a specific system or test object under test. It is developed with respect to a specific goal or test condition, such as conformance to specific requirements.
Test Case History, Report Diagram	The “Test History - Overview” report displays the rate of defined test cases versus executed test cases of a project for one or more test environments and one or more test systems in a given time period as a graph on the dashboard.
Test Case Priority	The priority of the test case: “Low”, “Medium” or “High”.
Test Case Progress, Report Diagram	The Test Progress - Overview report shows the rate of the executed versus the defined test cases of a project project for a given set of test environments and test systems as a graph on the dashboard.
Test Case Status	The status of the test case. Possible values are “Locked”, “Approved”, “Draft” and “Omitted”.
Test Step	A test step in Klaros-Testmanagement is the smallest action that is processed within a test case. It contains execution preconditions, ex-

	pected results, and execution postconditions for each individual action during test execution.
Test Case Result	The result of the execution of a test case including a verdict such as “Skipped”, “ Passed”, “Failed” or “Failed”.
Test Environment	A test environment represents the external circumstances that can influence the test result. Examples for test environments are e.g. the operating system or an application server (e.g. Tomcat 10 with Ubuntu 20.04.2) or also parameters such as temperature or speed.
Test Environment Overview, Report Diagram	The “Test Environment - Overview” report displays the success and progress rate of test systems under a single test environment in a radar chart on the dashboard. If less than three test systems are configured, a bar chart is displayed instead.
Tester	A user role that is allowed to view objects and execute test cases and test suites.
Test Execution	The process of running a test on the component or system under test, producing actual result(s). (ISTQB) Klaros-Testmanagement automatically guides and logs test execution, see Testrunner.
Test Manager	A test manager has full read and write access to all data of a project. He can create, modify and delete all types of objects and manage project settings.
Test Run	A test run contains the results of the execution of a single test case or an entire test suite. It always refers to exactly one test system and exactly one test environment.
Testrunner	Klaros-Testmanagement contains a web-based client for the execution of manual tests. This guides the tester through the test steps, gives him the possibility to create annotations and attachments and automatically logs the test progress and results.
Test Step	A test step in Klaros Test Management is the smallest action that is processed within a test case. It contains execution preconditions, expected results and execution postconditions for each individual action during test execution.
testSegment	A test segment in Klaros-Testmanagement is defined sequence of individual test steps that can be inserted into a test case. A test segment can be used in several test cases at once. This makes it possible to realize a module concept based on test steps.
Test Suite	A test suite is a set of test cases that can be executed together in a group.
Test system, System under test, Unit under test, Test object	In Klaros-Testmanagement, a test system represents the component or system that is to be tested. Depending on the business sector, the

	terms system under test, SUT, DUT or test object are also commonly used.
System under Test Overview, Report Diagram	The Test System Overview report shows the success and progress rate of test progress rate of test environments under a single test system in a radar chart on the dashboard. If less than three test environments are configured, a bar chart is displayed instead.
Test Type	A group of test activities based on specific test objectives with the purpose of testing a component or system for specific properties (Functional, Non-Functional, Structural, Regression, Retest).
Trac	Trac is an open source application for defect management and issue tracking in software development. Klaros Test Management integrates with Trac.

V

Variety	The expected outcome of the test: positive or negative.
Verdict	A rating is the result of the executed test case, test step or test suite. Possible ratings are “Passed”, “Failed”, “Error”, “Inconclusive”, “Skipped”.

W

Web Browser	Klaros-Testmanagement is a web application and runs within a web browser. Supported browsers are Apple Safari, Google Chrome, Microsoft Edge and Mozilla Firefox.
Work Log	The work log shows the chronological progression and duration of the activities for processing a job. The time spent for the entire test execution results from the sum of the execution times. the sum of the execution times.

Y

YouTrack	YouTrack is a popular application by the company JetBrains for defect management, troubleshooting, and project management.

Index

S

T

U