Klaros-Testmanagement Tutorial
by Sabrina Gidley, Fabian Klaffke, Claudia Könnecke, Klaus Mandola, Patrick Reilly, and Torsten Stolpman
Version 5.0.9
Publication date May 04 2021
Copyright © 2009-2021 verit Informationssysteme GmbH

Abstract

This document serves as a tutorial for the Klaros-Testmanagement application. It gives an example tour through the application and the provided functionality.

Legal Notice . Copyright 2009-2021 verit Informationssysteme GmbH, Europaallee 10, 67657 Kaiserslautern, Germany. All rights reserved. This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or documentation may be reproduced in any form by any means without prior written authorization of verit Informationssysteme GmbH and its licensors, if any.

Trademarks

Oracle™, Java™ and Solaris™ are registered trademarks of Oracle and/or their affiliates.

Windows® is a registered trademark of Microsoft Corporation in the United States and other countries.

JIRA® is a registered trademark of Atlassian Pty Ltd.

Other names may be trademarks of their respective owners.
List of Figures

1.1. Navigation in Klaros-Testmanagement ................................................................. 1
2.1. The >Projects page ............................................................................................. 2
2.2. The Systems under Test page ............................................................................. 3
2.3. The Test Environments page ............................................................................... 4
2.4. The Test Cases page ........................................................................................... 4
2.5. The Test Case Step tab in the Test Cases page .................................................. 5
2.6. The Test Suites page .......................................................................................... 6
2.7. The Test Suite page ........................................................................................... 6
2.8. The Run Test Case page .................................................................................... 7
2.9. The Run Test Case page .................................................................................... 8
2.10. The Test Case Runner ...................................................................................... 9
2.11. The Results Overview in the Test Runner ....................................................... 10
2.12. The Run Test Suite page ................................................................................ 11
2.13. The Run Test Suite page ................................................................................ 11
2.14. The Test Suite Runner .................................................................................... 12
2.15. The Dashboard ............................................................................................... 13
2.16. The Test Case Results page .......................................................................... 13
2.17. The Test Case Result page ............................................................................. 14
2.18. The Test Case Result page ............................................................................. 14
3.1. The Access tab ................................................................................................. 16
3.2. A Project with Limited Access ......................................................................... 16
3.3. Creating Categories ......................................................................................... 17
3.4. Assigning Test Environments to Categories .................................................... 18
3.5. The Test Environment Categories .................................................................... 19
3.6. Adding User Defined Properties ...................................................................... 19
3.7. The User-Defined / System under Test page .................................................. 20
3.8. Continuing a Test Run ...................................................................................... 21
3.9. The Issue Management page ........................................................................... 22
3.10. Linking Issues ............................................................................................... 23
3.11. Creating Issues .............................................................................................. 24
3.12. The Requirements Page ................................................................................ 26
3.13. The Requirements Details Page ..................................................................... 27
3.14. The Iterations Page ....................................................................................... 28
3.15. The Selected Iteration in the Topbar ............................................................... 28
3.16. The Iteration Details Page ............................................................................. 29
4.1. The Wizard Selection Screen .......................................................................... 30
4.2. The New Project Wizard ................................................................................ 31
4.3. The Package Explorer ..................................................................................... 32
4.4. Creating a New Folder ................................................................................... 33
4.5. The Build Path Setting Screen ...................................................................... 34
4.6. Selecting an External Jar File ......................................................................... 34
4.7. The Package Explorer ................................................................................... 35
4.8. Creating a Class ............................................................................................. 35
4.9. The Created Class .......................................................................................... 36
4.10. A Sample Front Page of a Report ................................................................. 39
4.11. A Generated Pie Chart ................................................................................ 40
4.12. The Header of a Test Case Report ................................................................ 41
List of Tables

3.1. User Roles ................................................................................................................. 22
Chapter 1. Introduction

Klaros-Testmanagement is an easy to use web based application which helps to organize test projects. It manages test cases, test suites, information about systems under test and test environments in which tests have been run. When a test case or test suite has been executed, its result, including the information about the system under test and its test environment, is stored in a database. This enables full traceability of all test runs. Multiple reports enable a detailed overview of the progress of the project at any time.

1.1. Navigation

Throughout this tutorial you will find instructions for navigating through Klaros-Testmanagement. Figure 1.1 shows a typical page with the navigation areas numbered.

The User Interface of Klaros-Testmanagement

1. The top menu bar. Here you can access functions for selecting the current project, for keyword search, help, language selection and logging out.

2. The sidebar shows the five main sections. If you hover over a specific section in the sidebar, the menu entries for this section are displayed.

3. The log panel displays various information, warnings and error messages.

4. Most tables in Klaros-Testmanagement have an action column like the one shown here. A click on the respective icons in this column executes actions for the objects in the corresponding row.

5. Many pages in Klaros-Testmanagement have buttons like this to perform actions like creating new objects, or saving or discarding changes. These buttons are always located above or below the table.
Chapter 2. Quick Start Guide

Our quick start guide shows how to create a project containing test environments, test systems and test cases and a test suite. The test cases contain test steps, and user-defined fields and categories are created. Then a single test case and a test suite with three test cases will be executed and finally the results will be displayed.

The system under test is a printer for which both hardware and software (drivers) should be tested. The driver software is available in different versions and for different operating systems, here Windows and Linux.

In addition, there are two separate development teams: one for the hardware, the other for the software. The hardware team uses JIRA as system for defect management, the software team uses Mantis.

2.1. Setup

First the test setup is created. This includes the project, the test environments, the systems under test, test cases and a test suite.

2.1.1. Creating a Project

Projects are the top-level objects in Klaros-Testmanagement. All other objects such as test cases, iterations, requirements, or test suites are part of the project. Therefore, creating the project is always the first step when working with Klaros-Testmanagement:

1. Press the New button to create a new project.

2. Enter Printer as the project Description.

3. Press the Save button. A unique project ID is assigned automatically.

Activate the new project with the radio button to the left of the project ID or to the right in the action column. Now the other menu items in the side menu bar can also be selected. The new project can now be edited with the Edit icon.
2.2. Creating Systems under Test

Next we create our systems under test. In Klaros-Testmanagement, a system under test represents the versions of the product or software system to be tested. In our example these are printers.

1. Under Define, select Systems under Test from the side menu bar.

2. Press the **New** button.

3. Enter the text *Printer Model 1* in the **Version** field.

4. Press the **Save** button.

5. Repeat the process for three more systems under test:
   - *Printer Model 2*
   - *Printer Model 3*
   - *Printer Model 4*

   Clicking in the ID column header changes the order of the displayed test systems.

2.3. Creating Test Environments

The next step in setting up the project is to define the test environments in which the tests will take place. In Klaros-Testmanagement the test environment defines the external influences that can influence the test result. This could be the operating system on which the printer driver is installed or the physical environment in which the printer is operated.

For this tutorial, we will create two software-related and two hardware-related test environments.

1. Under Define, select Test Environments from the side menu bar.

2. Press the **New** button.
3. Enter **Ubuntu 20.04** in the *Description* text field and click the *Save* button.

4. Create three more test environments with the following values in the field *Description*:
   - **Windows 10**
   - **Average Room Temperature**
   - **Maximum Operating Temperature**

5. Press the *Save* button.

### 2.4. Creating Test Cases

Now follows the creation of the test cases. We will create test cases for both the hardware and the software team. The first test for the hardware team is to verify that the printer meets the specifications regarding the minimum number of pages printed per minute.

1. **Under Define**, select *Test Cases* from the side menu bar.

2. **Press the** *New* button.
3. Enter **Test if the printer prints at least 10 page per minute** in the Name field. The name provides information about the function/task of the test.

4. Press the **Save** button.

5. Click on the test case ID or on the **Edit** icon in the action column.

6. Select the **Steps** tab.

![Figure 2.5. The Test Case Step tab in the Test Cases page](image)

7. Create the following steps:

<table>
<thead>
<tr>
<th>Description</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start printing the test document</td>
<td></td>
</tr>
<tr>
<td>Start the timer when the first page lands in the tray</td>
<td></td>
</tr>
<tr>
<td>Stop timer when the last page lands in the tray</td>
<td></td>
</tr>
</tbody>
</table>

8. Press the **Save** button.

Next, we create the following test cases, each with a single test case step. First click on the **Back** button to go back to the list of test cases.

<table>
<thead>
<tr>
<th>Name</th>
<th>Execution</th>
<th>Step Description</th>
<th>Step Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print test page</td>
<td>Manual</td>
<td>Print the test page using the OS UI</td>
<td>The page is being printed</td>
</tr>
<tr>
<td>Detect empty cartridge</td>
<td>Manual</td>
<td>Insert empty cartridge</td>
<td>Printer display shows the Cartridge empty warning</td>
</tr>
</tbody>
</table>
You should now see three created test cases, one with two test steps and one with three test steps.

2.5. Creating Test Suites

The final step before testing can commence is the definition of test suites. In Klaros-Testmanagement test cases are assigned to a test suite in order to execute them jointly later. A test suite can contain one or more test cases and test cases can occur in multiple test suites.

1. Under Define, select Test Suites from the side menu bar.
2. Press the New button.
3. Enter Tutorial Hardware Suite in the Name field.
4. Press the Save button.
5. Press the icon.
6. Click on the Properties tab.
7. Press the + icon on the Detects empty and the Test if the printer prints at least 10 pages per minute test cases.
8. Press the **Save** button

Our project is now complete, and we can start to execute the test cases.

2.6. Executing Tests

Now that we have created some test cases and a test suite, we are ready for the first test execution.

2.6.1. Executing Single Test Cases

In this section, we will execute a single test case multiple times with different parameters.

1. Under **Execute**, select **Run Test Case** from the side menu bar.

2. Now the page **Run Test Cases** is displayed. For all existing test cases it displays an execution icon in the column **Action**. The appearance of the icon can vary, for manually executable tests the 🔄 is displayed here.

![Figure 2.8. The Run Test Case page](image)

Click on the icon 🔄 for the test case **Test if the printer prints at least 10 pages per minute**.

3. On the next screen, select **Average Room Temperature** as the **Test Environment**
4. Select **Printer Model 1** as the **System Under Test**.

5. Click the **Execute** button.

6. A new browser window will open after a message about the current execution. If this does not happen, please check if your browser uses a popup blocker and add the Klaros-Testmanagement web URL to the exceptions if necessary.

   In the new window the tester is guided step by step through the test procedure. On the first page you can see an overview of the test case. The previous browser window is not required for the test execution and can be minimized.
7. The manual test runner will now show the first step. *Action* shows what we as a tester are supposed to do. Let’s assume our printer will start printing the test document with no problems, so the first step is passed.

Press the ✔.

8. Now the second step is displayed. Also mark the further steps as ✔ passed. Another dialog will appear in which you confirm that you want to end the test execution. Click the OK button.

9. Now the overview of the completed *Test Run* appears. Since all steps passed without complications, it is not necessary to create an issue/defect.

Click the EXIT button. Now return to the overview page with the three test cases.
Now we will execute the same test case using a different system under test and test environment.

1. Again click the icon on the **Test if the printer prints at least 10 pages per minute** test case.

2. On the next screen, select **Maximum Operating Temperature** as the **Test Environment**.

3. Select **Printer Model 2** as the **System Under Test**.

4. Press the **Execute** button.

5. Press the green ✔️ icon.

6. Press the green ✔️ icon.

7. Printer Model 2 uses the Model 2 printerheads, which have an overheating problem, so this test step will fail in an environment with a high ambient temperature.

   Press the orange ⚠️ icon.

8. Enter **Too many pause cycles due to overheating** in the **Summary** field.

   These comments will help reproducing the failure.

9. Click on the **Finish Test** button. A dialog appears in which you confirm the completion of the test case execution with **OK**.

10. If you have set up an issue management system, you can create or link an issue from this page. This process is described in Section 3.6, "Issues".

11. Press the **Finish** button.
2.6.2. Executing Test Suites

Next we will execute our test suite. The process is very similar to executing a test case. Additionally, an overview screen will appear at the beginning.

1. Select the entry Run Test Suites under Execute in the side menu.

2. Click on the icon for the test suite Tutorial Hardware Suite.

3. Select Average Room Temperature as the Test Environment

4. Select Printer Model 1 as the System under Test.

5. Press the Execute button.

6. Press the Start button in the manual test runner.
7. Click the green icon for all steps.

8. Confirm the following dialog by clicking the OK button.

9. Repeat for the second test case.

10. Press the Close button.

2.7. Results and Reports

Now we can take a look at the test results. has a comprehensive and detailed reporting system.

1. Log in using the manager account.

2. Select the Printer project if it is not already selected.

3. Select Evaluate in the side bar.

4. Now the dashboard will be displayed, showing some of the default reports for the Printer project. These show relevant information like the number of test cases and test suites in the Project, the overall numbers of passed and failed test runs, and the testing activity in the last 30 days.
5. The "Latest Success Rate" report can be changed by clicking on the ☐ icon. There you can select the test system and the test environment to be displayed.

6. You can also view individual test results for test cases and test suites in the Evaluate section. Under Evaluate, select Test Case Results from the side menu bar.

7. Here you will see the test cases which have been executed in this project, along with a count of test runs with their results.

8. Press the ☐ icon for the Test if the printer prints at least 10 pages per minute test case.

9. You will now see a screen summarizing each of the test runs for this test case. Press the ☐ icon for one of the test runs.
10. This screen shows a breakdown of the results of each step in the test case, as well as the summary and description the tester has entered for each of them.

Under Evaluate, select Test Suite Results from the side menu bar.

This page shows results of all test cases which have been run in a similar style to the Test Case Results page. Have a look at the results of the Tutorial Hardware suite.

This concludes our quick introduction. The following chapters contain more detailed sections that explain the possibilities of Klaros-Testmanagement in greater detail.
Chapter 3. How-Tos

This section of the tutorial consists of how-to guides for certain actions in Klaros-Testmanagement. We will continue to use the printer example project from the quick start guide.

It is recommended to complete the quick-start guide first if you wish to follow these guides step-by-step.

3.1. User Role Management

In this section we will learn how project access works and how to limit the access to our example project to specific users.

The default installation of Klaros-Testmanagement contains the three users admin, manager and tester (see the Klaros-Testmanagement documentation here and here for more information about user roles). If you create a new project (see Section 2.1.1, “Creating a Project”) all users of Klaros-Testmanagement will have access to this project. To limit the access to this project to a single users or several specific users, you can use project specific roles, which are part of Klaros-Testmanagement Enterprise Edition.

3.1.1. Limiting Project Access

Feature only available in Klaros-Testmanagement Enterprise Edition

In this section we will limit access to the Printer project. See here for more information about project specific roles.

Assigning project specific roles

Project specific roles do not need to match the role of the user. For example, a user with the role test manager can be a tester in one project and a test manager in another project.

1. Login to Klaros-Testmanagement using the Manager account.
2. Select the Define section in the sidebar.
3. Select the project P00001, or the "Printer Test" project if you already created other projects.
4. Select the Access tab.
5. **Figure 3.1. The Access tab**

6. Click on the **Assign** button.

7. In the following dialog select the checkbox in front of Max Mustermann and select **Test Manager** from the **Project Role** dropdown list (if not already selected).

8. Confirm the changes by clicking on the **OK** button.

The only user allowed to view the **Printer** project now is the user named **manager**. Administrators can still view the project, since they are excluded from project access rules. On the projects page, you can see which projects have access rules defined by the **icon in the **Additional information** column.

1. **Figure 3.2. A Project with Limited Access**

   1. Login to Klaros-Testmanagement using the **Manager** account.

   2. Select the **Define** section in the side bar.

   3. Select the project “**Printer Test**”.
4. Select the Access tab.

5. First we need to make sure, that at least one manager is assigned to the project. Select checkbox in the row containing Max Mustermann.

Click the OK button. Max Mustermann is now the only user with access to this project.

**Important**

If you use project specific roles, you need to assign at least one test manager per project.

### 3.2. Categorization

Feature only available in Klaros-Testmanagement Enterprise Edition

**Important**

In order to use the examples in this guide it is recommended that you complete Section 2.3, “Creating Test Environments” of the quick-start guide.

In our example project, we have separate test environments and test cases for hardware- and software-related criteria. A useful way to separate these within in Klaros-Testmanagement Enterprise Edition is to use the categorization feature.

In Klaros-Testmanagement objects such as test cases, test environments, etc. can be assigned to user-defined categories. This works similar to storing files in folders of a file system. We will now create two categories and assign the created test environments to the respective teams.

1. Click on the icon directly above the table in the workspace to open the Categories panel. Select Click here to edit categories.

![Figure 3.3. Creating Categories](image-url)
How-Tos

2. Press the + icon to create a new category group.

3. Insert Printer Test into the field Name.

4. Click the Save button and select the category Printer Test from the drop-down menu Category Group.

5. Next, rename the field ROOT to Team.

6. Now click on the + icon in the field Team and enter Hardware in the text field of the new subcategory. Repeat the step with the entry Software.

7. Click the Save button.

After the categories have been successfully created, the test environments can now be assigned to the individual teams.

1. To do this, use the icon right above the table to change the view.

2. Select the checkboxes to the left of the software-related test environments (Windows 10 and Ubuntu 20.04).

3. Click on the icon left above the table.

4. Select the entry Software in the displayed dialog.

![Klaros Test Management](image)

Figure 3.4. Assigning Test Environments to Categories

5. Click the OK button. The two entries now appear in the category Software.

6. Now select the checkboxes next to the hardware-related test environments.

7. Click the icon.

8. Now click on the entry Hardware in the dialog.

9. Click the OK button.
If the categories panel is closed, all existing test environments are displayed. If the panel is open, only the test environments of the selected category are displayed.

### 3.3. User Defined Properties

User defined properties are customizable fields which can be added to the various artifacts of a project e.g. to make it easier to find specific elements or to reference external systems or documents. In this how-to, we will set up some user defined properties for our systems under test (or SUTs).

Since the SUTs in our example are printers, there are a number of properties that will make searching through them easier. One of these is the firmware version, an alpha-numeric value which is changed relatively often.

The printers used in our example have different properties. These properties can be customized in the tab **User Defined**. You will find a detailed description in *Section 3.3, "User Defined Properties"*. Here we define the firmware version and the printer head models.

1. Click in the project ID or in the icon in the action column.
2. Select the tab **User defined** and **System under Test** below.
3. Click the **New** button.
4. Add the entry **Firmware Version** in the field **Name**. The **Type** is already preset with the desired value **Text**.

5. Click the **Save** button.

Next, we create 2 printer head models and define an enumeration type for them.

1. Click the **New** button.

2. Enter **Printerhead Model** in the **Name** text field.

3. Select **Enumeration** in the **Type** column. Click on the ** gore** icon that appears in the **Values** column. A dialog opens.

4. Click the **New** button and enter the entry **Model 1**.

5. Again click the **New** button and enter the entry **Model 2**.

6. Click the **Ok** button and then click **Save** on the main screen.

1. Click on the ID for **Printer Model 1** or select the ** gore** icon in its action column

2. Select the **User Defined** tab.

3. Enter **V23.41.06** in the **Firmware Version** field.

4. Select **Model 1** in the **Printerhead Model** dropdown list.

5. Press the **Save** button.

6. Now click on the green arrow in the upper right corner to switch to the next element. Now printer model 2 is displayed. Repeat the process for the remaining three systems under test.

<table>
<thead>
<tr>
<th>Version</th>
<th>Firmware Version</th>
<th>Printerhead Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Printer Model 2</td>
<td>V23.41.07B</td>
<td>Model 2</td>
</tr>
<tr>
<td>Printer Model 3</td>
<td>V23.41.07B</td>
<td>Model 1</td>
</tr>
<tr>
<td>Printer Model 4</td>
<td>V23.41.05</td>
<td>Model 2</td>
</tr>
</tbody>
</table>

Figure 3.7. The User-Defined / System under Test page

20
3.4. Pausing and Resuming Test Cases and Test Runs

**Important**

In order to use the examples in this guide it is recommended that you complete Section 2.5, “Creating Test Suites” of the quick-start guide.

Klaros-Testmanagement supports pausing and resuming of test cases and test runs. If the test runner is cancelled after at least one test step has been completed, the test run is automatically saved in the *Continue Test Runs* section.

1. Login to Klaros-Testmanagement using the Tester account.
2. Select the *Printer* project.
3. Under *Execute*, select *Run Test Suite* from the side menu bar.
4. Select the *Tutorial Hardware* test suite.
5. Complete the first test case as explained in Section 2.6.2, “Executing Test Suites”.
6. Press the *Cancel* button.

The manual test runner will now close, returning you to the main Klaros-Testmanagement window.

7. Select the *Continue Test Run* menu entry in the sidebar.

The interrupted test suite run will now be displayed, showing that one of the two test cases in the test suite have been completed.

8. You can click the ⬤ icon to continue executing the test suite or the ⌿ icon to delete this test suite run and the associated results.

![Figure 3.8. Continuing a Test Run](image)

3.5. Configuring Issue Management Systems

In order to set up Klaros-Testmanagement to use issue management systems, you must log in as an administrator.
Three user accounts are set up by default during the installation of Klaros-Testmanagement an administrator account, a manager account and a tester account.

<table>
<thead>
<tr>
<th>Username</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>admin</td>
</tr>
<tr>
<td>manager</td>
<td>manager</td>
</tr>
<tr>
<td>tester</td>
<td>tester</td>
</tr>
</tbody>
</table>

Table 3.1. User Roles

1. Log in as administrator and navigate to **Configure - Integration - Issue Management**.

2. Press the **New** button.

3. Select JIRA in the **System** drop-down list.

4. Enter **PRINTER** in the **Project** text field (or the name of a test project, e.g. **PLAYGROUND**)

5. Enter **Hardware Team** in the **Description** text field.

6. (Optional) If you have a JIRA system to test with, enter the address in the **URL** text field and press the **go** icon.

7. Press the **Save** button to save your new issue management system.

Repeat the process, select Mantis instead of JIRA and use **Software Team** as the description.

Now these two issue management systems can be assigned to projects within Klaros-Testmanagement.

This concludes the role of the administrator account in our quick start guide. Log out of Klaros-Testmanagement now and log in using a manager account.
3.6. Issues

**Important**

In order to use the examples in this guide it is recommended that you complete Section 2.6.1, “Executing Single Test Cases” of the quick-start guide.

Klaros-Testmanagement supports the creation of issues in issue management systems (IMS) (see Section 3.5, “Configuring Issue Management Systems”) as well as linking them with test cases. This can be achieved either by creating an issue in the IMS from within Klaros-Testmanagement, or by using the **Link Issues** feature to link pre-existing issues and test cases.

Both of these actions can be carried out from within the manual test runner or in the Evaluate - Issues section of Klaros-Testmanagement. Each Results Overview page in the manual test runner has the Link Issue and Create Issue buttons.

3.6.1. Linking Issues

![Figure 3.10. Linking Issues](image)

1. Login to Klaros-Testmanagement using the **Tester** account.
2. Under Evaluate, select Issues from the side menu bar.
3. Press the ![icon](image) icon in the action column of the test case **Test if the printer prints at least 10 pages per minute**.
4. Press the **New** button.
5. Select the **Hardware Team** issue management system from the Issue Management System dropdown list.
6. Enter a valid issue id in the **ID** text field (this id has to match an existing issue in the IMS).
7. Press the ![icon](image) icon next to the **ID** field (Please wait a few moments until the issue has been retrieved from the IMS).
8. Select the system under test **Printer Model 1** in the **System under Test** dropdown list.

9. Press the **Link** button.

The list of assigned issues should now contain a single entry.

### 3.6.2. Creating Issues

![Image of Klaros Test Management interface](image)

**Figure 3.11. Creating Issues**

1. **Login** to Klaros-Testmanagement using the **Tester** account.

2. Press the **Evaluate** link in the sidebar.

3. Select the **Issues** menu entry in the sidebar.

4. Press the **New** button.

5. Select the **Software Team** issue management system from the **Issue Management System** dropdown list.

6. Enter **The printer Model 1 cannot print documents larger than 50 Mb** into the **Summary** field.

7. In the **Description** field, enter **The Model 1 version of the printer cannot print documents which are larger than 50 Mb in size. This is regardless of the document type (e.g. pdf or txt).**

8. Select the test case **Test if the printer prints at least 10 pages per minute** in the **Test Case** dropdown list.
9. Press the Save button.

You should now see a message *The issue was successfully created in Mantis with ID XYZ.* in the log panel on top of the screen.

### 3.7. Revisions

**Important**

In order to use the examples in this guide it is recommended that you complete Section 2.4, "Creating Test Cases" of the quick-start guide.

Test requirements can change during the lifecycle of a project, due to various internal or external changes. Let's assume that in our *Printer* project for example, 20 pages per minute instead of 10 are the new status quo for printers. Instead of copying and editing the test case *Test if the printer prints at least 10 page per minute* you can simply create a new revision for that test case.

**Note**

With Klaros-Testmanagement you can create Revisions for test cases, test segments, test suites and requirements.

1. Login to Klaros-Testmanagement using the *Manager* account.
2. Select the *Printer* project.
3. Press the *Define* link in the sidebar.
4. Select the *Test Cases* menu entry in the sidebar.
5. Press the icon on the test case *Test if the printer prints at least 10 page per minute*.
6. Press the *Revisions* tab.
7. Press the *New Revision* button.
8. Enter *requirements have changed* in the *Comments* text field.
9. Press the *OK* button.
10. Press the *Properties* tab.
11. Change the *Description* of the test case to *Test if the printer prints at least 20 page per minute*.

You have successfully created a new revision of the test case.

**Note**

When creating a new revision, this revision gets selected as the active revision for that test case. You can, at any time, switch back to an older revision via the revision tab.
3.8. Requirements

Requirements are conditions which must be met in order for the product to be considered ready. A requirement can be linked to several test cases, which must be executed in order to fulfill this requirement.

In the course of this section, we will create a requirement, link it to a test case and execute this test case.

![Figure 3.12. The Requirements Page](image)

1. Select the **Define** section in the sidebar.
2. Select the **Requirements** menu entry in the sidebar.
3. Press the **New** button.
4. Write **The printer prints at least 10 pages per minute** in the **Name** text field.
5. Select **High** from the **Priority** dropdown list.
6. Press the **Save** button.
7. Press the ![checkmark] icon of the requirement **The printer prints at least 10 pages per minute**.
8. Select the **Test Cases** tab.
9. Press the **Neu** button.
10. Select the test case **Test if the printer prints at least 10 page per minute**.
11. Press the **Assign** button.

### Multiple Requirements

You can assign multiple test cases to the same requirement.

You can also assign test cases to multiple requirements!
Now we will execute the test case with which the requirement is assigned to.

1. Switch to the **Execute** section in the sidebar.
2. Select the **Run Test Case** menu entry in the sidebar.
3. Press the icon.
4. Press the **Execute** button.
5. Press the green icon.
6. Press the **OK** button.

You have now executed a test case that is assigned to a requirement. Let's see the results of this test case on the requirements page.

1. Select the **Define** section in the sidebar.
2. Select the **Requirements** menu entry in the sidebar.
3. Press the icon of the requirement *The printer prints at least 10 pages per minute*.
4. Select the **Results** tab.

In the results tab, you can see the results of all test cases that are assigned to this requirement. Since we've executed a single test case, there is only one element in this table. Pressing the icon shows you the details of this test case result.

### 3.9. Iterations

An iteration represents a single test cycle in a project. This helps to synchronize the test process with an agile development process (e.g. Scrum). An iteration can represent a milestone for a
project. In a software development project for example, an iteration could be *The core functionality is working*.

In this section, we will create an iteration, learn how to activate or deactivate iterations and how to assign requirements to a specific iteration.

![Figure 3.14. The Iterations Page](image)

1. Select the **Define** section in the sidebar.
2. Select the **Iterations** menu entry in the sidebar.
3. Press the **New** button.
4. Enter **Sprint 01 - Alpha stage** in the **Name** text field.
5. Press the **✓** icon.
6. Select the **Properties** tab.
7. Enter **All printer models can print a test page without an error.** and **All printers print at least 10 pages per minute.** in the **Success Criteria** text field.
8. Press the **Save** button.

You have now successfully created a new iteration. Activate the new iteration with the radio button to the left of the iteration ID or to the right in the action column. You can see the active iteration in the topbar (see Section 3.9, "Iterations").

![Figure 3.15. The Selected Iteration in the Topbar](image)

**Using Iterations with Requirements**

Iterations are more useful when combined with requirements (see Section 3.8, "Requirements"). For example, the iteration "Alpha release" of a project could contain the requirements *All database unit tests pass* and *The setting menu is working*. 
1. Select the **Requirements** tab.

2. Click on the **Assign** button.

3. Select the requirement *The printer prints at least 10 pages per minute*.

4. Click on the **Assign** button.

You have now successfully linked a requirement to an iteration.

If an iteration is active, only results and artefacts for this iteration are displayed. To deselect the current iteration or switch to a different one, follow the next steps.

1. Click in the [ ] icon in the topbar.

2. Select the empty entry in the iteration dropdown box.
Chapter 4. Report Tutorial

This tutorial will show how to create reports for Klaros-Testmanagement. Please note that custom reports are only available in Klaros-Testmanagement Enterprise Edition. Basic knowledge in Java programming and XML is required to follow this tutorial. An example of the resulting report can be found here. The complete code can be downloaded here, feel free to modify it to your needs.

To create custom reports navigate via the top menu Configure and then via the side menu Report Templates to the custom report section of Klaros-Testmanagement Enterprise Edition. You can create a new report by pressing the New button. You can later use this report via the top menu Evaluate and the side menu Report Templates.

4.1. Development Environment

The first step in this tutorial will describe how to set up a development environment to help with creating reports. We recommend using Eclipse as the development framework.

Create a project to contain your reports

• Open Eclipse and right click into the Project Explorer
• In the menu then select New -> Other.
• In the dialog select Java Project as shown in the picture then click on Next.

Figure 4.1. The Wizard Selection Screen
• In the next dialog enter a title for your project (e.g. `ReportTutorial`) and press *Finish*.

![New Java Project dialog](image)

**Figure 4.2. The New Project Wizard**

• The *Package Explorer* in Eclipse should now show your new project.
Set up the Project

- First, create a subfolder to hold your XML files. Right click on your project and select New -> Folder. In the dialog enter xml in the Folder name field and click on Finish.
• Add the Klaros-Testmanagement scripting library to your build path. Right click on our project and select Properties. In the dialog click on Java Build Path(1), select the Libraries(2) tab and then click on Add External JARs...(3).

Figure 4.4. Creating a New Folder
• In the following dialog navigate to your Klaros-Testmanagement installation folder. From there navigate to the webapps/klaros-web/WEB-INF/lib folder and select the klaros-core-x.y.z, klaros-model-x.y.z.jar and klaros-scripting-x.y.z.jar, then click the Open button.

• Finally click on the OK button in the Build Path dialog. You now have an additional entry in your project named Referenced Libraries.
4.2. Preparing the Data for the Report

The next step of this tutorial will show how to gather and prepare data for the report. The report we are going to create will show a summary of all test runs for all test cases in a test suite together with a detailed description of all executed test steps, listed per test case per test run. To provide the data for the report the API described in the Klaros API documentation can be used to retrieve the required data.

The following picture shows the steps to perform to create a KlarosScript class.
• First a class implementing the KlarosScript interface must be created. Right click on the src folder (1) in your project and select New -> Class.

• In the new dialog enter a name for your script class (2).

• Then click on the Add button (3) and enter KlarosScript in the new dialog to provide the KlarosScript interface for the class. Then press the ok button to close the interface dialog.

• Finally press the Finish button (4). After a short while the class should be created and Eclipse should present you the following view.

![Figure 4.9. The Created Class](image)

• Before entering any code, the import statements must be provided.

```java
import de.verit.klaros.scripting.*;
import de.verit.klaros.core.model.*;
import java.util.*;
```

• The code to be executed must be entered into the execute method of the KlarosScript class. To provide a certain level of flexibility, the test suite to generate the report for should be passed as a parameter to the KlarosScript class. We will see later in this tutorial how this works in detail. It is achieved by the following line of code:

```java
String suiteName = (String)context.getParameterValue("suiteName");
```

• Next the database is accessed to retrieve the data for the report. The following snippet shows how to access the database. This code selects all entries from the KlarosTestSuite table whose ids matches the test suite id passed to the KlarosScript and that are present in the currently active project. The result is returned as a list, though only one single entry is expected.

```java
String suiteQuery = "select g from KlarosTestSuite g where g.name='" + suiteName + "'";
if (context.getActiveProject() != null) {
    suiteQuery += " and g.configuration.name='";
    suiteQuery += context.getActiveProject().getName();
    suiteQuery += "'");
}
List testSuites = context.executeQuery(suiteQuery);
```

• To avoid any errors when executing the KlarosScript the following code is contained in an if-clause if no matching test suite was found.

```java
if (!testSuites.isEmpty()) {
    testSuite = (IKlarosTestSuite) testSuites.get(0);
    ...
}
```

• Add the test suite to the context so it can later be retrieved from the template.
Now the preparations for the pie chart in the report can begin. The report should present a pie chart showing the amount of test case results in the states error, skipped, failure and success. Therefore a list of KlarosTestCaseResult is required for each result type. This code is also placed inside the if-clause.

```java
List<KlarosTestCaseResult> error = new ArrayList<>();
List<KlarosTestCaseResult> failure = new ArrayList<>();
List<KlarosTestCaseResult> success = new ArrayList<>();
List<KlarosTestCaseResult> skipped = new ArrayList<>();
```

Now the lists get filled. First we step through the list of test cases belonging to the test suite.

```java
for (KlarosTestCase klarosTestCase : testSuite.getTestCases()) {
    ...
}
```

Inside this loop we iterate over the test results for each test case. This can be more than one since a test case can have several test runs. The following code must be placed in the for loop from the step before.

```java
for (KlarosTestCaseResult testResult: klarosTestCase.getResults()) {
    ...
}
```

We now have the test results at hands and can distribute them to the four lists defined earlier, depending on the state of the test result. This code must be placed inside the second for loop defined one step before.

```java
if (testResult.isError()) error.add(testResult);
else if (testResult.isFailure()) failure.add(testResult);
else if (testResult.isPassed()) success.add(testResult);
else if (testResult.isSkipped()) skipped.add(testResult);
```

We are almost done. The last step is to store the four lists containing the KlarosTestCaseResults in the context to access it later from the template. This code is placed inside the outermost if-clause and outside of the two for loops.

```java
context.add("error", error);
context.add("failure", failure);
context.add("success", success);
context.add("skipped", skipped);
```

The complete KlarosScript class can be found in the archive here.

### 4.3. Create the Layout of the Report

Klaros-Testmanagement uses the JBoss Seam PDF framework to render the report and fill the retrieved data into the layout template.

1. We will start with an empty document that contains a header and footer definition.

```xml
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:p="http://jboss.org/schema/seam/pdf"
    title="Klaros-Testmanagement Test Plan Report" marginMirroring="true"/>
```
This snippet creates a header that is displayed on every page. It contains the date and the name of the Klaros-Testmanager user who created the report. The footer contains the page number and an informative text that the report was created with Klaros-Testmanagement. Note that for the header the `borderWidthBottom` attribute was set to provide a separation from the following text, while for the footer the `borderWidthTop` attribute was used to create a single line for separation.

2. Next we create a front page accumulating the main information about the report.

The layout of the front page is done using `<p:paragraph>` elements for formatting. These elements use alignment and spacing to define the position where the text will be placed. To change the font of the paragraphs, you can use the `<p:font>` element and use its style and size attributes to highlight certain parts.
An image of a sample front page can be found below.

Test Run Report

Created by
Felix Mustermann (admin@verit.de)
on
9/1/2020

Testsuite TS00001 - Tutorial Hardware Suite - revision 1.0

SUT: SUT00001 - Printer Model 1

Figure 4.10. A Sample Front Page of a Report

As you might have noticed from the header and footer definition, values stored in the context can be accessed by preceding the context variable enclosed in curly brackets with a #, e.g. #{user.name}. The context variable for the user is automatically inserted into the context by Klaros-Testmanagement. You might remember the part where we prepared the data for this report and added the test suite to the context.

context.add("testSuite", testSuite);
For the front page we access this context variable to retrieve the data that came with this test suite, e.g. #{testSuite.name}, #{testSuite.shortname}, and #{testSuite.revisionId}. You can find the accessible attributes in the Klaros-Testmanagement API Documentation reference in the online documentation. All attributes that have a getter can be accessed, e.g. for #{testSuite.name} see the interface IKlarosTestSuite where you will find the method getName in its IKlarosLabeledObject parent interface. To add a page break to the document, you can use

3. Now that we have a front page we can start to fill in the data we prepared and stored in the context. To give a quick overview over the test results we will start with a pie chart displaying the amount of successful, failed, skipped, and erroneous test case results in this test suite.

An example of a generated pie chart can be found below.
For a detailed description about how to layout the pie chart and many other charts, please check the seam-pdf documentation. Again you might remember that we added four List<KlarosTestCaseResult> objects to the context:

```java
context.add("error", error);
context.add("failure", failure);
context.add("success", success);
context.add("skipped", skipped);
```

These four objects represent the results of our test suite. We can simply use the size of the collections representing the data for the pie chart:

```html
<p:data key="Error [#{error.size}]" value="#{error.size}" sectionPaint="#FF0A0A" />
```

4. Next we want to print a page for each Test Case contained in the test suite and present its data in a table.

As this might get a bit lengthy, only fragments will be presented here. An example can be found below.

![Figure 4.12. The Header of a Test Case Report](image)

First we need to loop over all test cases of the test suite:

```xml
<ui:repeat value="#{testSuite.testCases}" var="testCase">
...
</ui:repeat>
```

A loop can be done using the `<ui:repeat>` element. The attribute value gets a collection and the attribute var gets the name of a variable that can be used for further processing, e.g. testCase. With every step through the loop, the variable testCase will get the next element from the collection.

Now for the table. We need a two column table and the width of the second column should be three times the size of the first. The code is as follows:

```html
<p:table columns="2" widths="1 3">
...
</p:table>
```

Let's get some data into the table. Therefore we have to describe what each table cell will look like. This code goes inside the `<p:table>` element.
Since we defined that the table should be two columns wide, two `<p:cell>` elements build one row in the table. In the code above the first row holds the name of the project while the second line holds the name of the user who created the test case. A cell can also contain text formatting information as you can see from the `<p:font>` elements.

5. Next we need the test case results for the test case being displayed. This is where another element comes in. If there is no test result for a test case contained in the test suite an informative text should be displayed instead of just leaving a blank space.

```xml
<ui:fragment rendered="#{testCase.results.isEmpty()}">
  <p:font style="normal" size="14">
    <p:paragraph alignment="left" spacingAfter="15" indentationLeft="10">
      <p:text value="No test runs found for this Test Case." />
    </p:paragraph>
  </p:font>
</ui:fragment>
```

Notice the `<ui:fragment rendered="..."/>` element. This part of the document is only integrated into the pdf if the condition defined in the rendered attribute evaluates to `true`. This could be compared to an if-clause in modern programming languages. In the expression

```java
#{testCase.results.isEmpty()}
```

the `isEmpty()` method of a the test case result list is called which evaluates to a boolean value. Next we have to define the block that displays the data in case there are test results present for the test case:

```xml
<ui:fragment rendered="#{!testCase.results.isEmpty()}">
  ...
</ui:fragment>
```

Note the `!` which is used to negate the boolean expression we used in the block before. The code inside this expression is called if there is at least one test case result.
6. And now we loop over the results of the test cases, which will include each test run for a test case. An example of a Test Run can be found below.

```
<ui:repeat value="#{testCase.results.toArray()}
  var = "testResult">
  <!-- Start Test Result summary (equivalent to Test Run) -->
  ...
</ui:repeat>
```

This will provide us with the test results of a test case in the variable `testResult`. The next pieces of code include the displaying of a summary of the test result which is quite similar to the code used displaying the test case summary. You can have a look at it in the attached source file. The next thing to display are the test case steps and their results for the current test run.

7. Therefore another loop is added inside the Test Result loop.

```
<ui:fragment rendered="#{testResult.stepResults.isEmpty()}">
  <p:font style="normal" size="18">
    <p:paragraph alignment="left" spacingAfter="35"
    indentationLeft="25">
      <p:text value="No test step results found. "/>
    </p:paragraph>
  </p:font>
</ui:fragment>
```

Inside the `<ui:repeat>` block a table is created. This table should be displayed if there are any results for the test case step. The table displays the precondition, action, and postcondition of the test case step and also the test case step result, the test case step summary, and the test case step description. The test case step result cell of the table will get
coloured according to the result, e.g. green if the step passed. Each table will have the number of the step as a headline.

8. Displaying the step number.

```xml
<ui:fragment rendered="#{testStepResult.isPassed()}">
  <p:cell horizontalAlignment="center">
    <p:font style="normal" size="8">
      <p:paragraph>
        <p:text value="Passed" />
      </p:paragraph>
    </p:font>
  </p:cell>
</ui:fragment>
```

Here you can see how to retrieve the index of the test case step result from the list of test results. testStepResult is the variable from the inner loop, while testResult is the variable from the outer loop. As counting starts at zero we have to increment the retrieved value, otherwise the first step would be step 0.

9. Displaying the table if the test case has at least one test case step defined.

```xml
<ui:fragment rendered="#{testCase.testCaseSteps!=null and testCase.testCaseSteps.size() > 0}">
  <p:table columns="3" widths="3 3 3" spacingBefore="5">
    ...  
  </p:table>
</ui:fragment>
```

10. Retrieving the test case step properties (precondition, action, postcondition)

```xml
<ui:fragment rendered="#{testCase.testCaseSteps!=null and testCase.testCaseSteps.size() > 0}">
  <p:cell horizontalAlignment="left">
    <p:font style="normal" size="8">
      <p:paragraph>
        <p:text value="#{testCase.testCaseSteps.get(testStepResult.precondition}" />
      </p:paragraph>
    </p:font>
  </p:cell>
</ui:fragment>
```

To get to the test case step properties we have to access the test case to retrieve the data. From the list of test case steps we retrieve the test case step via the index in the list of the test case step results and then the precondition can be accessed. The same applies to the action and the postcondition of the test case step.

11. Colouring a cell depending on the test step result.

```xml
<ui:fragment rendered="#{testStepResult.isPassed()}">
  <p:cell backgroundColor="rgb(0,255,0)" horizontalAlignment="center">
    <p:font style="normal" size="8">
      <p:paragraph>
        <p:text value="Passed" />
      </p:paragraph>
    </p:font>
  </p:cell>
</ui:fragment>
```

```xml
<ui:fragment rendered="#{testStepResult.isError()}">
  <p:cell backgroundColor="rgb(255,0,0)" horizontalAlignment="center">
    <p:font style="normal" size="8">
      <p:paragraph>
        <p:text value="Error" />
      </p:paragraph>
    </p:font>
  </p:cell>
</ui:fragment>
```
The cells are rendered depending on the status of the test case step, therefore the methods isError(), isFailure() and so on are called for the test result object under processing. The cell gets its colour by setting the backgroundColor attribute to the desired rgb value.
# Glossary

## A

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin</td>
<td>See Administrator.</td>
</tr>
<tr>
<td>Administrator</td>
<td>User role that has access to all functionalities in Klaros-Testmanagement.</td>
</tr>
<tr>
<td>Artifact</td>
<td>An Artifact is a definable object like a Project, Iteration, Requirement, Test Environment, System under Test, Job, Test Case, Test Suite or Test Case.</td>
</tr>
</tbody>
</table>

## B

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bugzilla</td>
<td>Bugzilla is an open source bug tracking and testing tool.</td>
</tr>
</tbody>
</table>

## C

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>Artifacts can be assigned to any number of user-defined categories. Users can group related Artifacts together.</td>
</tr>
<tr>
<td>Coverage</td>
<td>A database is a collection of information organized into interrelated tables of data and specifications of data objects.</td>
</tr>
<tr>
<td>Compliance</td>
<td>nix drin</td>
</tr>
</tbody>
</table>

## D

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>A database is a collection of information organized into interrelated tables of data and specifications of data objects.</td>
</tr>
<tr>
<td>Defect</td>
<td>See Issue.</td>
</tr>
<tr>
<td>Defect Management System</td>
<td>See Issue Management System.</td>
</tr>
</tbody>
</table>

## E

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-Mail</td>
<td>Electronic mail, often abbreviated as e-mail, is any method of creating, transmitting, or storing primarily text-based communications with digital communications systems. E-Mail notifications can now be sent if a test execution fails or if a job dependency has been resolved.</td>
</tr>
<tr>
<td>Error</td>
<td>An error is the inability of the system to perform a test correctly. Not to be confused with Failure.</td>
</tr>
</tbody>
</table>
F

Failure  A failure is a discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. Not to be confused with Error.

G

Guest  A user role that can display artifacts and generate reports, but may not change any data.

GUIdancer  GUIdancer is an Eclipse-based tool for automated functional testing through the Graphical User Interface (GUI).

I

Incident  See Issue.

Incident Management System  See Issue Management System.

Issue  The term issue is a unit of work to accomplish an improvement in a system. An Issue could be a bug, a requested feature, task, missing documentation, and likewise.

Issue Management System  An Issue Management System (Issue Tracking System) is a software to manage Issues.


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 glossary).

J

Java  Java is a programming language. Most often, Java is used as an abbreviation for Java Runtime Environment, which needs to be installed in order to run Klaros-Testmanagement.

Java Runtime Environment  The Java Runtime environment needs to be installed in order to execute applications programmed in the Java programming language.

JavaScript  JavaScript is a scripting language most often used to add functionality to web pages. Most newer Web browsers can process JavaScript generated code.

Java Runtime Environment  See Java Runtime Environment.

JIRA  JIRA is a bug tracking, Issue tracking, and project management system by Atlassian Software.
<table>
<thead>
<tr>
<th><strong>Glossary</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Job</strong></td>
</tr>
<tr>
<td><strong>Jubula</strong></td>
</tr>
<tr>
<td><strong>JUnit</strong></td>
</tr>
<tr>
<td><strong>Manager</strong></td>
</tr>
<tr>
<td><strong>Mantis</strong></td>
</tr>
<tr>
<td><strong>Operating System</strong></td>
</tr>
<tr>
<td><strong>Postcondition</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td><strong>Project</strong></td>
</tr>
<tr>
<td><strong>QFTest</strong></td>
</tr>
<tr>
<td><strong>Redmine</strong></td>
</tr>
<tr>
<td><strong>Glossary</strong></td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td><strong>Requirement</strong></td>
</tr>
<tr>
<td><strong>Role</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>S</strong></td>
</tr>
<tr>
<td><strong>Selenium</strong></td>
</tr>
<tr>
<td><strong>SUT</strong></td>
</tr>
<tr>
<td><strong>System Account</strong></td>
</tr>
<tr>
<td><strong>System under Test</strong></td>
</tr>
<tr>
<td><strong>T</strong></td>
</tr>
<tr>
<td><strong>Test Case</strong></td>
</tr>
<tr>
<td><strong>Test Case Result</strong></td>
</tr>
<tr>
<td><strong>Test Case Step</strong></td>
</tr>
<tr>
<td><strong>Test Environment</strong></td>
</tr>
<tr>
<td><strong>Tester</strong></td>
</tr>
<tr>
<td><strong>Test Execution</strong></td>
</tr>
<tr>
<td><strong>Test Run</strong></td>
</tr>
<tr>
<td><strong>Test Runner</strong></td>
</tr>
<tr>
<td><strong>Test Suite</strong></td>
</tr>
<tr>
<td><strong>Trac</strong></td>
</tr>
<tr>
<td><strong>URL</strong></td>
</tr>
<tr>
<td><strong>Web browser</strong></td>
</tr>
<tr>
<td><strong>Windows</strong></td>
</tr>
</tbody>
</table>