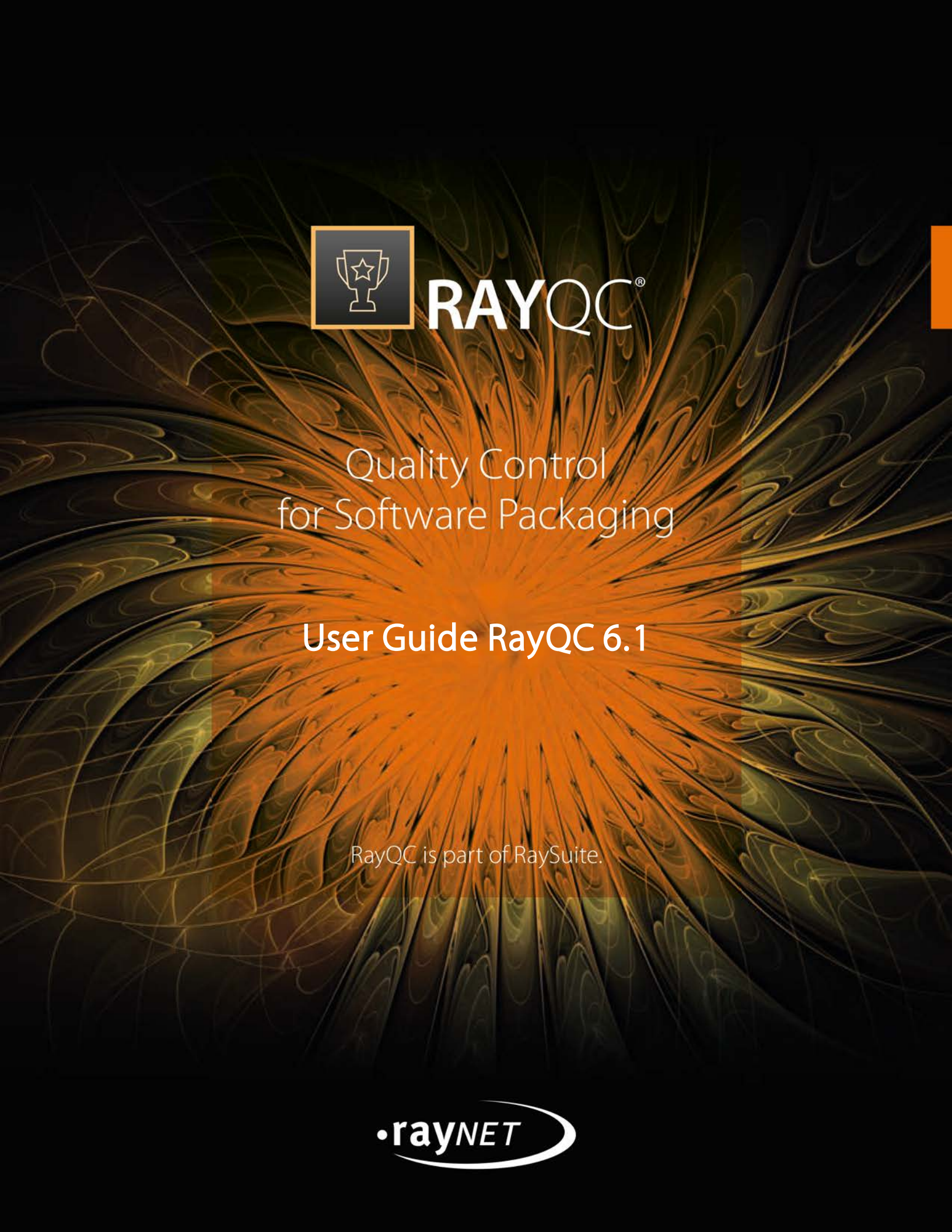

RAYQC®
Quality Control
for Software Packaging
User Guide RayQC 6.1
RayQC is part of RaySuite.
•rayNET

Copyright © Raynet GmbH (Germany, Paderborn HRB 3524). All rights reserved.
Complete or partial reproduction, adaptation, or translation without prior written permission is prohibited.

RayQC 6.1 User Guide RayQC

Raynet and RayFlow are trademarks or registered trademarks of Raynet GmbH protected by patents in European Union, USA and Australia, other patents pending. Other company names and product names are trademarks of their respective owners and are used to their credit.

The content of this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Raynet GmbH. Raynet GmbH assumes no responsibility or liability for any errors or inaccuracies that may appear in this document. All names and data used in examples are fictitious unless otherwise noted.

Any type of software or data file can be packaged for software management using packaging tools from Raynet or those publicly purchasable in the market. The resulting package is referred to as a Raynet package. Copyright for any third party software and/or data described in a Raynet package remains the property of the relevant software vendor and/or developer. Raynet GmbH does not accept any liability arising from the distribution and/or use of third party software and/or data described in Raynet packages. Please refer to your Raynet license agreement for complete warranty and liability information.

Raynet GmbH Germany
See our website for locations.

www.raynet.de

Table of Contents

Introduction 7
- RayQC Features at a Glance 7
- RayQC is a part of RaySuite 8
- Development Roadmap 9
- Additional Resources 9

System Requirements 10
- Hardware Requirements 10
- Supported OS 10
- Prerequisite Software 10
- Migration 12

Installing RayQC 16
- Product Activation 24
  - License wizard 24
    - Use order number 30
    - Use license file 34
    - Use floating license server 36
    - I do not have a license or order number 37
    - I want to take my activation back 37

Working with RayQC 38
- Objects 41
- Typical Workflows 44
- The Home Screen 45
- The Checklist Viewer 47
- The Checklist Editor 53
- Shortcuts 56
- Communication with RayFlow 57
  - From RayFlow to RayQC 57
  - From RayFlow to RayQC and back 59
- The RayQC Command Line Interface 61
- The FILE menu 63

Settings 66
- Interface 69
- Behavior 71
- Signing 72
- Plug-Ins 76
- Report profiles 80
- RayFlow 85

Virtual Machines ... 86
Snapshot Selector ... 89
Preparing Virtual Machines ... 90
Advanced Configuration Options ... 93
About ... 96
Get Started ... 96
License and Edition ... 99
Troubleshooting ... 102
Checklist Structures ... 104
Basic Checklist Properties ... 104
Steps and Actions ... 105
Groups ... 108
Elements ... 111
Element Types ... 112
Element Options ... 117
Element Controls ... 128
Properties ... 136
Supporting Files ... 138
Plug-Ins ... 141
Post Processing ... 143
Checklists on the File System ... 146
Formatting Markup Options ... 146
Standard Checklist Procedures ... 148
Create Checklist Templates ... 148
Evaluate Checklist Projects ... 149
Create Checklist Evaluation Reports ... 150
Edit Checklist Templates ... 152
Delete Checklists ... 153
Plug-Ins ... 155
Plug-In Types ... 155
Using Plug-Ins in Checklists ... 156
Configuration during Checklist Creation ... 158
Execution during Checklist Evaluation ... 161
Internal Plug-Ins ... 165
Command Plug-In ... 166
File Plug-In ... 168
Folder Plug-In ... 174
IniFile Plug-In ... 179
Local System Plug-In ... 183

Logic Plug-In ... 185
Regular Expressions ... 195
MSI Plug-In ... 198
RayFlow Plug-In ... 203
Registry Plug-In ... 207
Advanced Plug-In ... 211
External Plug-Ins ... 213
Structure of a Plug-In ... 213
The Manifest ... 215
The Plug-In Logic Script ... 217
Local and Global Plug-Ins ... 218
Using Virtual Machines ... 222
Working with RayFlow ... 225
Introduction ... 225
Signing-in to RayFlow ... 225
Signing out from RayFlow ... 228
Troubleshooting ... 229
License Activation Tool is Shown at RayQC Launch ... 229
Missing Item Numbers in a Checklist Group ... 230
Missing Plug-Ins ... 230
Logging RayQC Activity Fails ... 232
Connections to Virtual Machines ... 234
Additional information ... 236
Help & Support ... 236
Appendices ... 237
Basic Checklist Structure ... 237
Checklist Example ... 238
Basic External Plug-In Structure ... 239
External Plug-In Example ... 240

Introduction

RayQC is a rule-based tool to create and execute proof-criteria in one or more checklists. It offers various modules to check the quality of applications and software packages all along the Application Lifecycle Management process and also allows you to integrate your own test criteria. The execution of the various test phases is typically done via a combination of manual (i.e. are there available and sufficient licenses for the application?) and automated tasks (i.e. are the MSI properties set according to the packaging guidelines?).

More than the ease-of-use of the checklist viewers in RayQC, the rule-based interpretation of the individual test results takes quality control to a new level. It is for example possible that the overall quality control phase does not fail, even when one or more test criteria are set as "not achieved", thanks to predefined interpretation rules. All the control steps are also summarized in a final quality report which is intended to be handed over to the next packaging process phase.

RayQC therefore meets the requirements for an enterprise level software packaging quality control tool. With RayQC from Raynet, you can rapidly see a reduction in errors, and in the long-term secure the quality of business processes, which in turn drastically relieves IT and Helpdesk alike. Not to mention the significant positive influence on the incurring expenses for Application Lifecycle TCO in general.

Tip:
Various online references are used throughout this document. Although an online connection is not required for this help file, further reading and reference material is available if you do have an internet connection for any of the external links present in this document.

RayQC Features at a Glance

The core features of RayQC cover the following quality assurance requirements:

- Standardization of the quality control process and tasks
- Reduced quality control cycle times
- Improved quality through results certified by RayQC
- Ability to add or change rules according to requirements
- Combination of manual and automated test phases
- Flexible implementation of in-house test scripts in the checklist
- Documentation of the results in the quality report
- Integral part of RaySuite and fully integrated in RayFlow

Standardization

- Easy creation of XML-based checklists with the Checklist Editor.
- Create your own test criteria for each individual quality control phase in the software packaging lifecycle
- Define how to proceed with each test result. Does a "failed" test lead to a rejected package? Or does the tester simply need to document the reason for failure to move on to the next test phase?
- Export of the test results as HTML or DOCX files

Automation

- Automating test routines („Runbook") is available by the integration of standard or custom plug-in functionality
- Start tests directly from RayFlow and integrate the overall process management with RayFlow
- Reporting to RayFlow may be configured to be an automated part of the overall QA workflow procedures

RayQC is a part of RaySuite

"Best-practice workflow" for Enterprise Application Lifecycle Management

RaySuite offers product solutions for the creation, operation and control of individually customized Enterprise Application Lifecycle Management projects in automatically administered environments.

Designed as part of the RaySuite, Raynet's product suite for Enterprise Application Lifecycle Management, RayQC is a packaging process related quality assurance tool. However, due to its flexible design RayQC actually is a multi-purpose application for any kind of checklist-based validation procedure. It may be used either way - standalone or as integrated part of RaySuite.

When RayQC is used as part of the RaySuite, it is connected to RayFlow, the workflow management component of the RaySuite. In this case there are some additional features, such as directly exporting checklist results to RayFlow, which will not be available in a standalone instance of RayQC. Please refer to the RaySuite website for further information regarding the benefits this product family offers for your daily business tasks.

Development Roadmap

Upcoming releases will introduce additional new components and features, resulting in boosted productivity, time and resource savings and improvements regarding the product experience. As the development of RayQC is customer oriented, please let us know if you have any ideas or suggestions of how you see your ideal quality assurance and checklist execution tool. Our sales team will be happy to help you, also make sure to check our website http://raynet.de to never miss our next releases, announcements, special offers and product trainings.

Additional Resources

Further information regarding RayQC can be found in several resources.

- The RayQC 6.1 Release Notes provide an overview about the changes and new features that are part of this version of RayQC.
- The RayQC 6.1 Installation Guide provides IT professionals with a guide on how to install RayQC for the first time.
- The RayQC 6.1 Operations Supplement provides information about third-party software packages and libraries redistributed with RayQC.
- The product website www.rayqc.de provides information about the product, news, and support.
- Raynet and its partners offer a range of training courses that can also be customized to meet your requirements. For more information on these courses, speak with your Raynet consultant or contact the Raynet Sales department via sales@raynet.de.

System Requirements

The given requirements name prerequisites for devices running the RayQC application.

Hardware Requirements

Minimal

- CPU: Intel Core i5
- Screen resolution: 1024 x 768 pixels
- RAM: 4GB
- Disk space: 10GB

Recommended

- CPU: Intel Core i7
- Screen resolution: 1280 x 1024 pixels
- RAM: 16GB or higher
- Disk space: 100GB or more

Note:
The installation of the RayQC framework itself requires about 70MB of disk space. The amount of additional space required depends on the amount of packaging material and the location of your data store.

Supported OS

The following represents the list of supported operating systems at the time of release.

- Windows Vista SP2
- Windows 7 SP1
- Windows 8
- Windows 8.1
- Windows 10
- Windows Server 2008 R2
- Windows Server 2008 SP1
- Windows Server 2012
- Windows Server 2012 R2
- Windows Server 2016
- Windows Server 2019

Prerequisite Software

General

- .NET 4.5 Client & Full for Windows Vista up to Windows 8 systems (both 32-bit and 64-bit).

- Windows Management Framework (including Windows PowerShell 3.0, WinRM 2.0, and BITS 4.0).
  Please verify if the named or later versions are available on your device before using internal or external plug-ins in checklists.
  For further details and download resources visit http://support.microsoft.com/en-us/kb/968929.

Be aware:
In order to be able to use external plug-ins with RayQC, it has to be ensured that the PowerShell version supported by the device that hosts the application matches the PowerShell version of the actual plug-in script. It is highly recommended to synchronize the PowerShell version among all devices that are assigned for QA execution to prevent compatibility issues in the first place.

RayFlow

In order to use RayFlow functionality directly from RayQC, a running RayFlow server has to be accessible.

Hyper-V integration

- RayQC requires that RayPack 6.1 is installed on the same machine in order to use Hyper-V functions..
- Both host and guest machine must have PowerShell 3.0 or newer installed.
- Windows Remote Management
- RayPack Studio Tools for Hyper-V need to be installed on the guest machine.

The tools can be installed from a Windows Installer package that is present in the RayQC subfolder `Tools \HyperVTools\Packaging Suite Tools for Hyper-V.msi`.
The installation of the tools is required, so that the user can see interactive prompts and windows on Hyper-V machines. It is recommended to install the tools as a part of the base snapshot.

VMware Workstation / ESXi5.5 - 6.0

RayQC requires that RayPack 6.1 is installed on the same machine in order to use ESXi/Workstation functions.
RayQC supports the following products:

- VMware vSphere 5.5-6.0
- VMware Workstation 10 and newer
- VMware Workstation 7, 8, 9 and for VMware vSphere 4.x, 5 and 5.1 are experimentally supported.

To use any of VMware Workstation / ESX machines, one of the following must be installed in an appropriate version:

- VMware Workstation
- VMware VIX API (https://my.vmware.com/web/vmware/details?productId=26&downloadGroup=VIX-API-162)
- vSphere

The required VIX API version depends on the systems that it needs to connect to. The below table presents the supported versions of VMware products depending on the installed VIX API version.

VIX API Version	VMware Platform Products	Library Location
1.11	vSphere 5, Workstation 8 or earlier	Workstation-8.0.0-and-vSphere-5.0.0
1.12	vSphere 5.1, Workstation 9 or earlier	Workstation-9.0.0-and-vSphere-5.1.0
1.13	vSphere 5.5, Workstation 10 or earlier	Workstation-10.0.0-and-vSphere-5.5.0
1.14	Workstation 11 or earlier	Workstation-11.0.0
1.15.0	Workstation 12 or earlier	Workstation-12.0.0

ESXi 6.5 and newer

RayQC requires that RayPack 6.1 is installed on the same machine in order to use ESXi/Workstation functions..

To make use of ESXi 6.5+ servers, the following prerequisites must be met:

- PowerShell 3.0
- PowerShell Execution Policy set to Unrestricted or RemoteSigned
- PowerCLI installer (https://www.powershellgallery.com/packages/VMware.PowerCLI/11.2.0.12483598)
- VMware Tools installed on the VM
- **Guest operations** and **System** permissions granted to the user executing the product.

Combination of supported versions is presented in the following table:

More information about PowerCLI:

- https://pubs.vmware.com/vsphere-51/index.jsp?topic=%2Fcom.vmware.powercli.cmdletref.doc%2FGet-VMGuest.html
- https://pubs.vmware.com/vsphere-51/topic/com.vmware.powercli.cmdletref.doc/Invoke-VMScript.html
- https://pubs.vmware.com/vsphere-50/index.jsp?topic=%2Fcom.vmware.wssdk.pg.doc_50%2FPG_ChD_Privileges_Reference.22.3.html

Migration

Upgrading the RayQC Application

General upgrade preparations

RayQC 6.1 is delivered as an MSI software package. In order to install it safely:

1. Download the MSI package for RayQC 6.1 from the Raynet resource repositories.
   (If you have not already received credentials, please contact the Raynet support team via our Support Panel).

2. Copy all files that need to be kept for later reuse or look-up (such as resources of global external plug-ins, log, settings and config files, the `*.license` file, etc.) to a temporary transfer directory outside the RayQC application directory (where they usually reside).

3. Make a backup of your SQL Server database which is used by RayQC Advanced Module.

4. Execute the RayQC 6.1 MSI package and work yourself through the setup routine. The installation of RayQC 6.1 is described in the RayQC 6.1 Installation Guide.

Note:
If upgrading RayQC Advanced, ensure that a **running** SQL server is available before starting the migration/installation.

If an older version then RayQC 2.1 is installed on the target machine

If an older version then RayQC 2.1 is already installed on the target machine there are two different ways to migrate to the new RayQC 6.1.

- Install RayQC 6.1 and keep the installation of RayQC 1.5 or RayQC 2.0. They will remain untouched by the installation of RayQC 6.1.

General upgrade preparations

RayQC 2.1 is delivered as an MSI software package. In order to install it safely:

1. Download the MSI package for RayQC 2.1 from the Raynet resource repositories.
   (If you have not already received credentials, please contact the Raynet support team via our Support Panel).

2. Copy all files that need to be kept for later reuse or look-up (such as resources of global external plug-ins, log, settings and config files, the `*.license` file, etc.) to a temporary transfer directory outside the RayQC application directory (where they usually reside).

3. Remove the old RayQC installation manually.

4. Execute the RayQC 6.1 MSI package and work yourself through the setup routine. The installation of RayQC 6.1 is described in the RayQC 6.1 Installation Guide.

Adjusting the newly installed RayQC instance

1. Launch RayQC.

2. Define config files and settings according to the old system state.

3. Test the new settings and configurations by creating and evaluating checklists, communicating with RayFlow, reviewing log files, etc.

4. If there are issues regarding broken or missing functionality, please feel free to contact the Raynet support team via our Support Panel.

RayUpdater

During the migration, if database changes are necessary, these will be done automatically. While executing the setup routine, RayUpdater, the tool used for a safe migration of the data, will be launched automatically and perform all necessary steps without the necessity of any user input.

Upgrading RayQC Files

The file formats RQCT and RQCP Raynet introduced in RayQC 1.5 and have been massively reworked to match the needs of the modernized application logic. Therefore, it is not possible to simply re-use templates and projects that have been generated with RayQC 1.5 in the current version 6.1.

The RQCT files used in RayQC 6.1 are no longer XML structures, but ZIP containers that contain the XML checklist file (`checklist.xml`) as well as all other resources required to run the checklist on RayQC: plug-ins, help files, images, etc. are stored within dedicated directories wrapped in the ZIP container.

Additional files that represent the current project status of a checklist evaluation (`state.xml`), post-processing settings and signature information, are added when a template is saved as project file RQCP.

Knowing about these changes makes it quite obvious that there must be some manual steps in any kind of checklist transition from version 1.5 to 6.1. Once this is done, the following standard procedure is a valid option for their transition to 6.1:

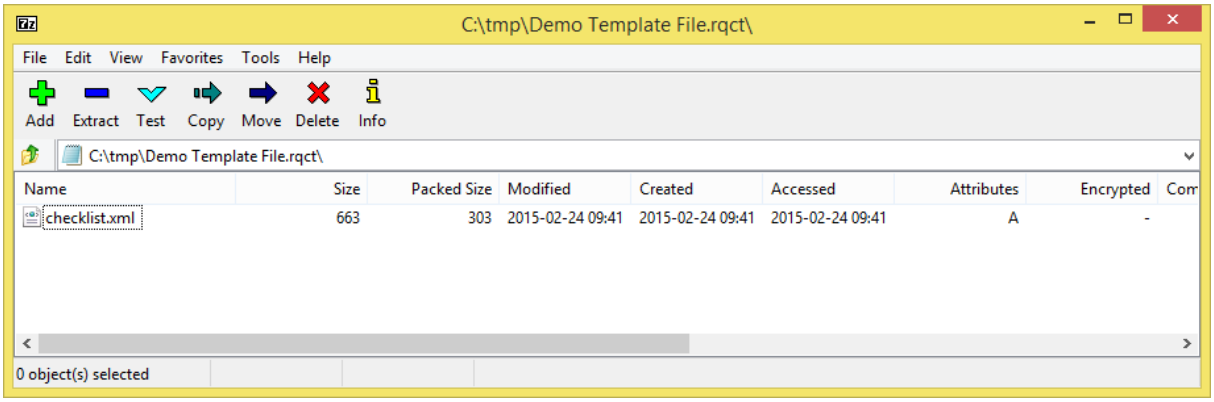
To transfer a RayQC 1.5 RQCT to the current 6.1 format, users have to run the following procedure:

1. Copy the original RQCT file to a temporary working directory.

2. Change the file
   a. name to `checklist`
   b. extension from `.rqct` to `.xml`

3. Create a new ZIP that contains the `checklist.xml` file. Name the ZIP container according to the old checklist file name, and set the file extension to RQCT.

4. The result of steps 1-4 has to be a zip container with the file extension `*.rqct`, that contains a `checklist.xml` file with the original checklist structure.



5. Open this file in RayQC 6.1.

6. It is most likely, that the validation procedure run during checklist loading states issues with the XML source structure. In this case, a dialog is displayed, revealing details about invalid areas with a click on the more button.

   Open the `checklist.xml` file from within the RQCT container, and correct all mentioned issues to establish an XML file that is valid according to the `ChecklistSchema.xsd` demanded by RayQC 6.1.

7. Save the changes to the `checklist.xml` file, and re-try to open the RQCT container with RayQC.

8. Repeat steps 6 & 7 until the checklist is successfully validated and opened by the application.

   Once this level is achieved, all upcoming changes may be executed directly within the checklist editor. Please refer to the User Guide section about editing checklist templates for further instructions.

Be aware:
Checklists with extended plug-in and condition usage may be quite difficult to upgrade manually, since both parts of the system logic have undergone revolutionary changes during the development of RayQC 6.1. Therefore, these checklists are recommended to be recreated from scratch.
Also be aware:
There is no direct upgrade path for RayQC projects from product versions prior to 1.5.

Please contact your RayQC service consultant, or the Raynet support team to get information about possible forms of assistance for any required upgrading measures.

Installing RayQC

The RayQC installation resources come along as a MSI file with additional documents, such as the current Release Notes and a User Guide. Usually, these documents are delivered via FTP server as provided by our support which you can contact via our Support Panel.

Before the application is installed on a device some preparations are needed:

1. Please make sure, that you have your order number or license file at hand. Either one will be provided by your Raynet sales representative or our support team which you can contact via our Support Panel.

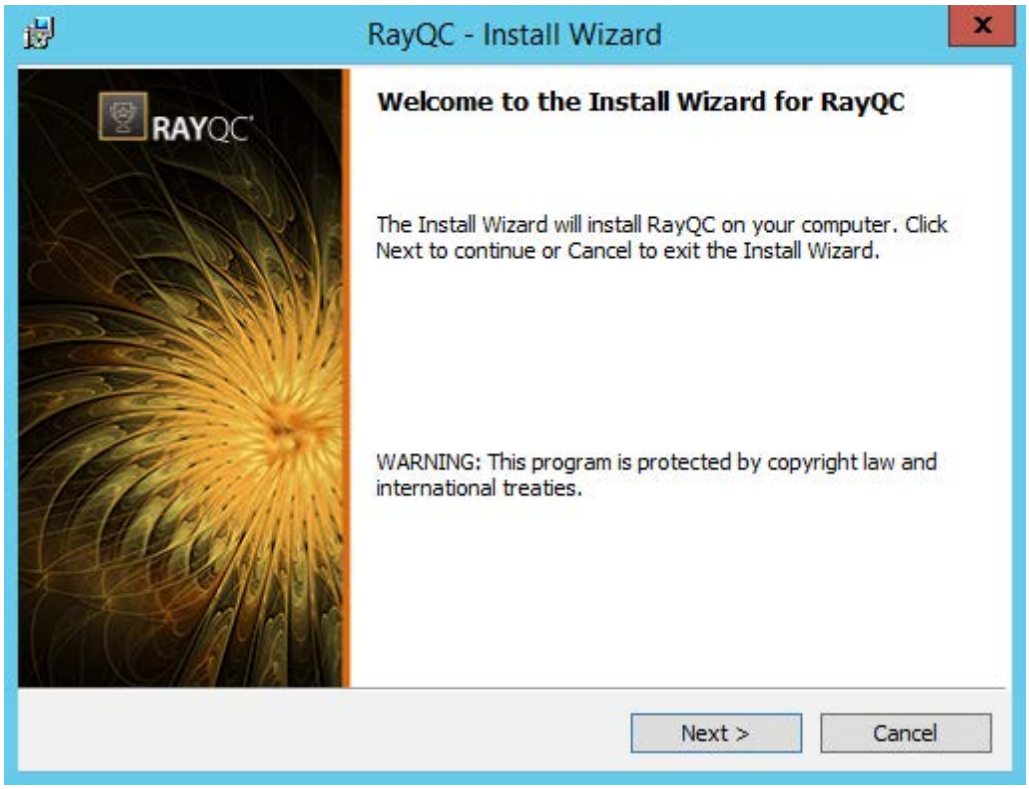
2. The target system needs to meet the system requirements described within the *System Requirements* chapter.

3. A Windows User with sufficient rights for installations has to be logged in

4. Close all dispensable applications during the setup routine execution.

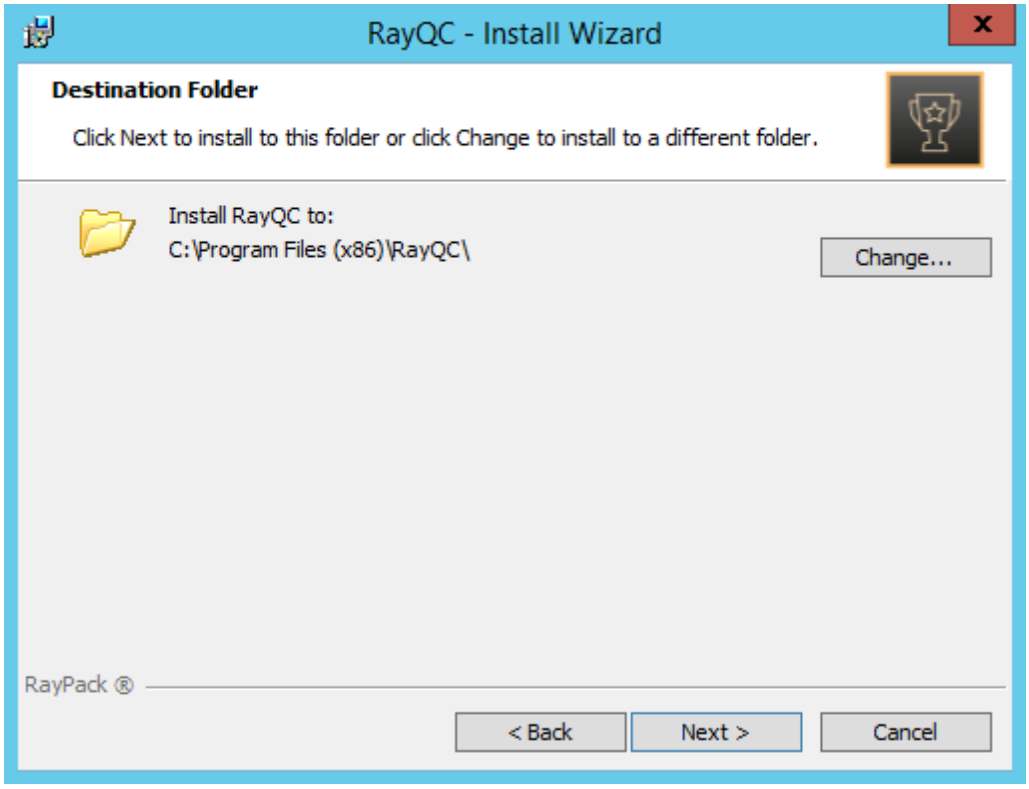
Installing RayQC

Launch the RayQC setup with a double-click on the MSI file and wait for the **Welcome Screen** to be prepared.

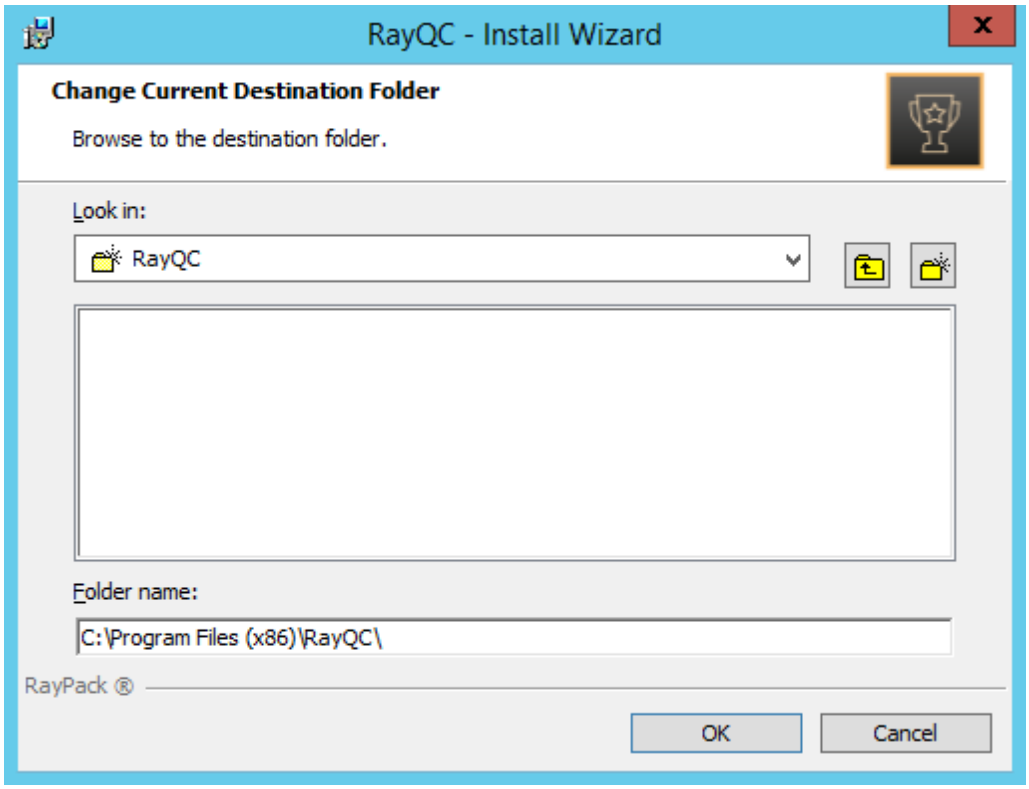


Click on the **Next >** button to proceed with the installation.

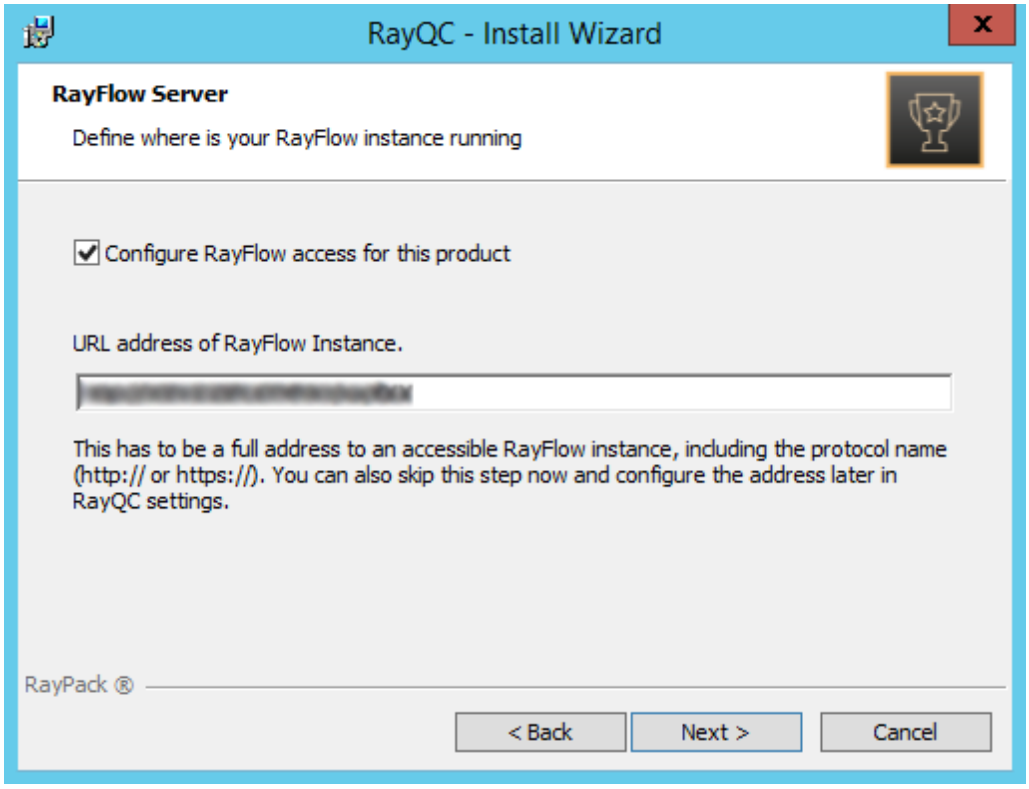
The **End User License Agreement** dialog appears. In order to install RayQC, the End User License Agreement has to be accepted. To proceed, read the End User License Agreement, select the **I accept the license agreement** option, and click on the **Next >** button.



Choose the destination folder by either keeping the suggested default or by clicking on the **Change...** button to select another target directory.

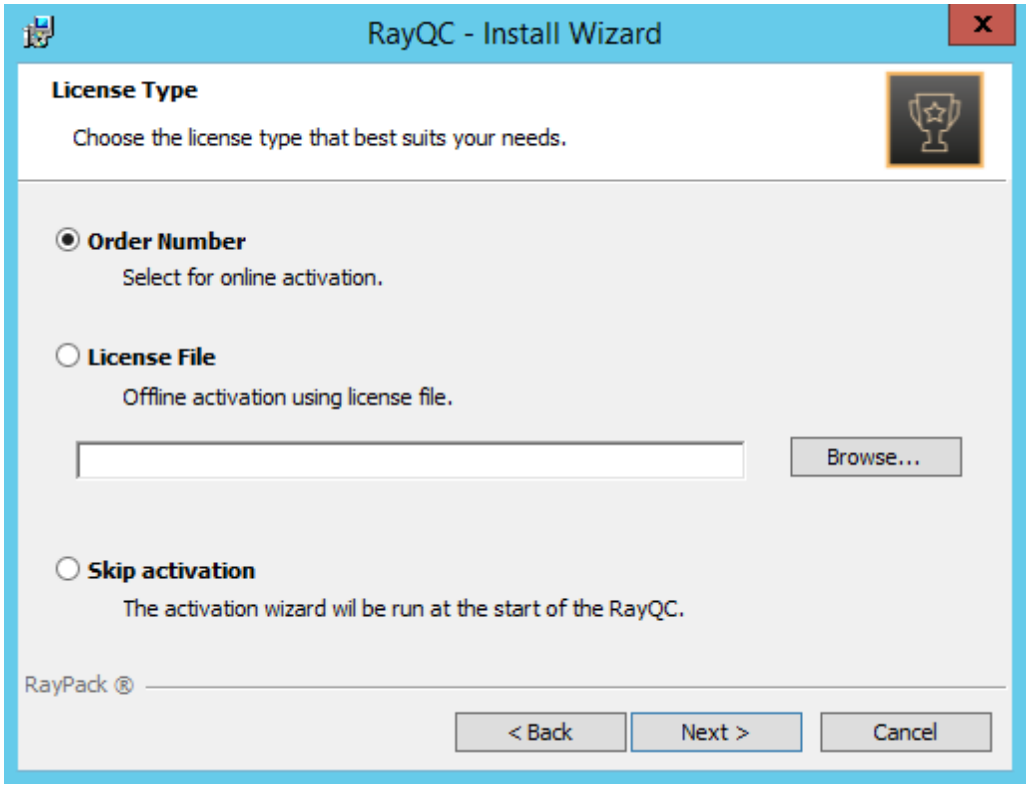


If a custom destination folder has to be defined, use the icons in the dialog above to navigate to the desired installation directory or to create a new one. After this, click on the **OK** button to return to the Destination Folder dialog. The path will be updated to display the custom destination recently selected. Click the **Next >** button in the **Destination Folder** dialog to proceed.



The RayFlow access for this product needs to be configured. If the **Configure RayFlow access for this product** checkbox is checked, the RayFlow server can be configured. The full address of the RayFlow instance that is about to be used for RayQC needs to be entered in the text field. If the **Configure RayFlow access for this product checkbox** is left unchecked, RayFlow access can still be configured in the **Settings** page of RayQC after the installation has been successfully completed.

Click the **Next >** button in the **RayFlow Server** dialog to proceed.

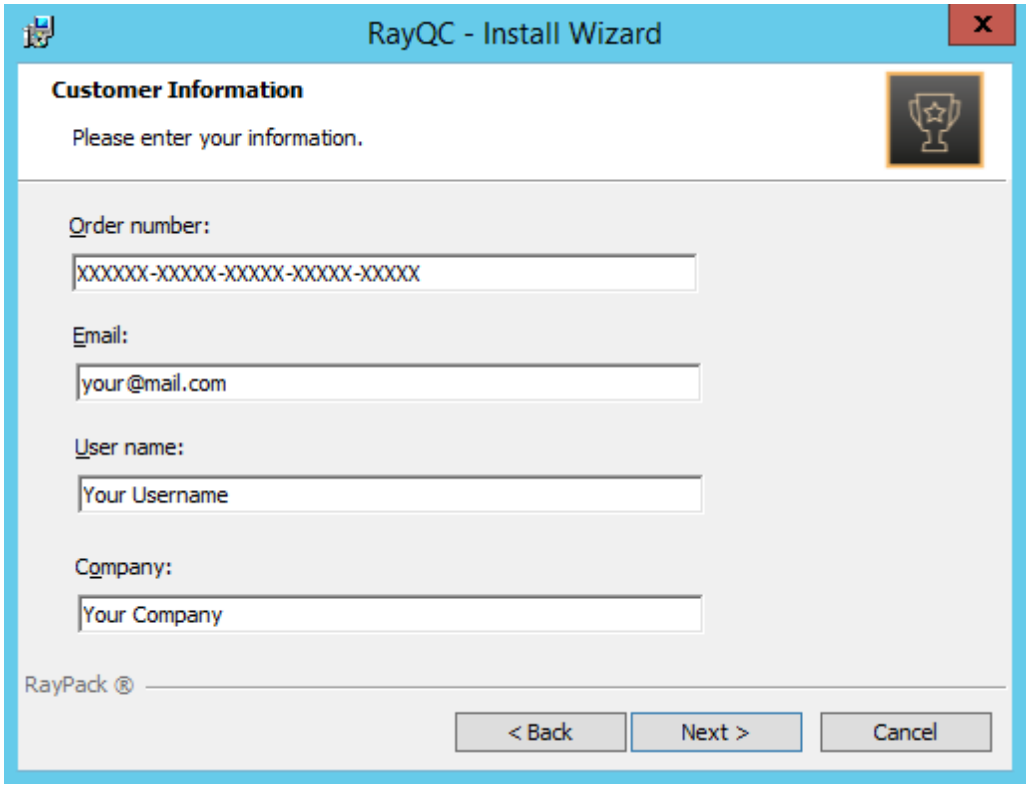


The **License Type** screen provides options to either activate the RayQC instance via order number and online activation service, by using an already prepared file (`*.license`), or to skip activation for now. If the activation is skipped, it will need to be performed later when RayQC is launched. For more information on product activation at the first launch of RayQC read this *section*.

To use an already existing license file, which most likely has been provided by the Raynet support team, the **Browse...** button has to be clicked. Use the controls of the system browser dialog to navigate to the `*.license` file and select it with a click on the **Open** button.

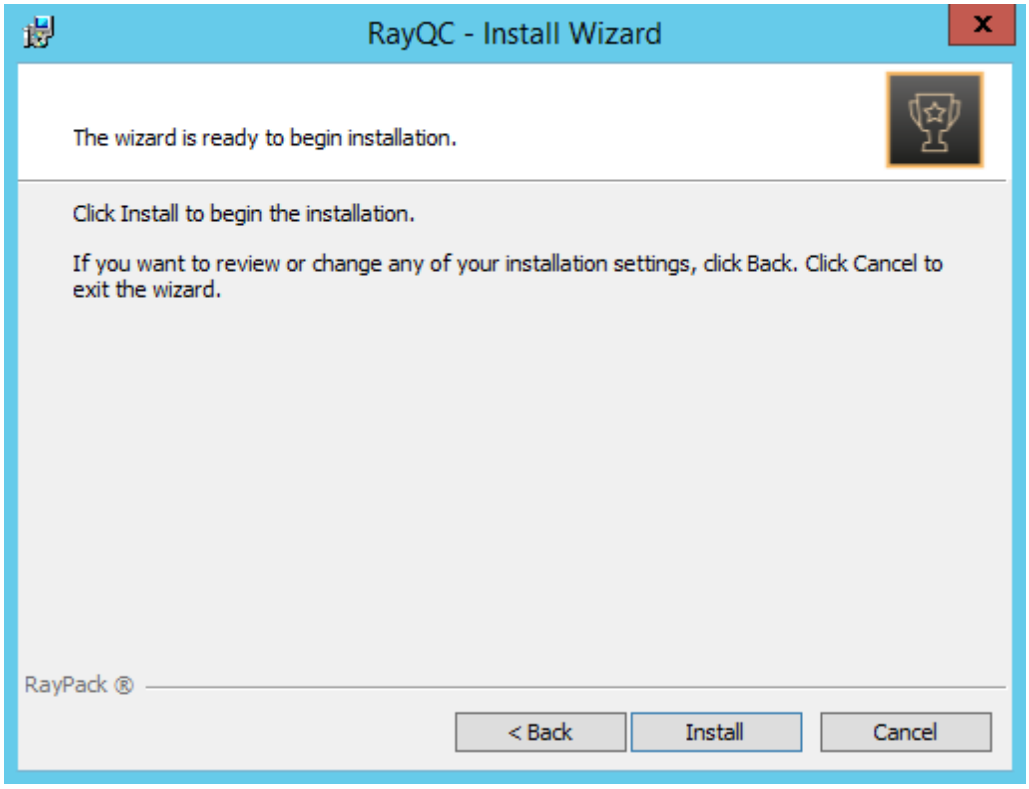
Click the **Next >** button to proceed.

If activation by order number has been selected during the previous step, the **Customer Information** screen is displayed:



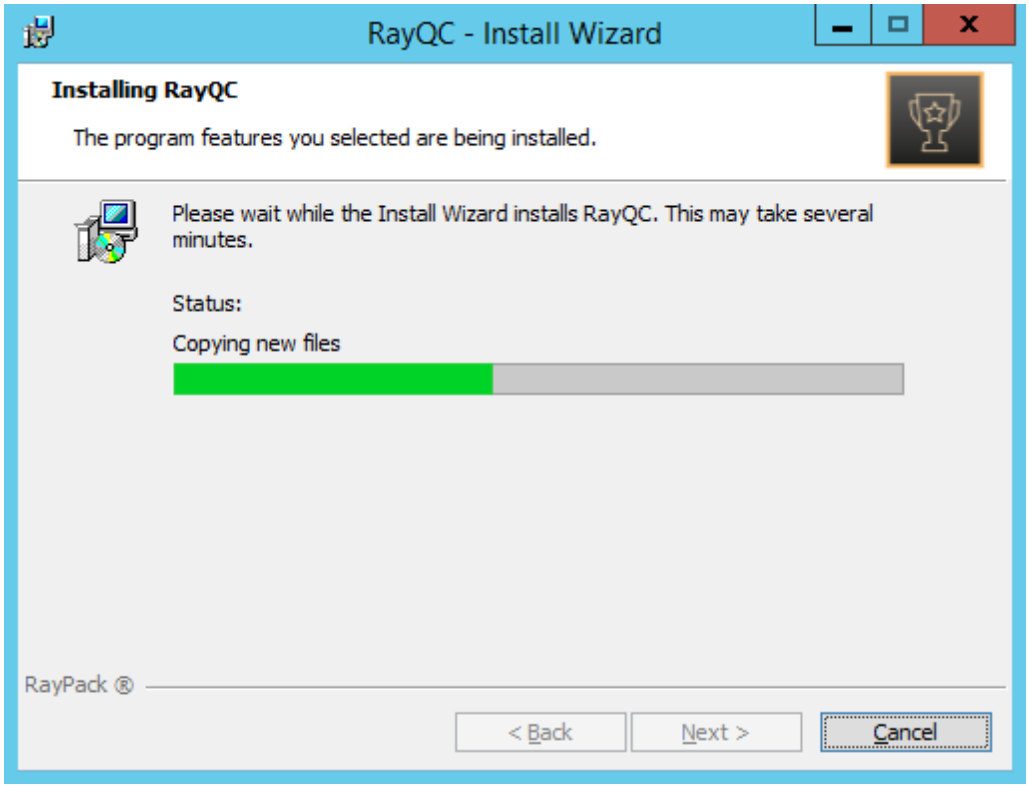
Please enter your individual RayQC Order number and provide user information, such as E-Mail, user name, and company name. The information will be used to verify the order number during the upcoming execution procedure.

Click the **Next >** button to proceed.



All required settings and properties are now defined and RayQC is ready to be installed. Click on the **Install** button to start the process.

A progress indication dialog is displayed as long as the installation steps are executed.



As soon as all required measures are done, the **Install Completed** dialog is presented.

Check the **Launch RayQC** checkbox to launch the application after the setup has been finished. If the **Open**

Release Notes Checkbox is checked, the *Release Notes* will be opened after the setup has been finished.

Click the **Finish** button to exit the setup.

Product Activation

The product can be activated using one of the following methods:

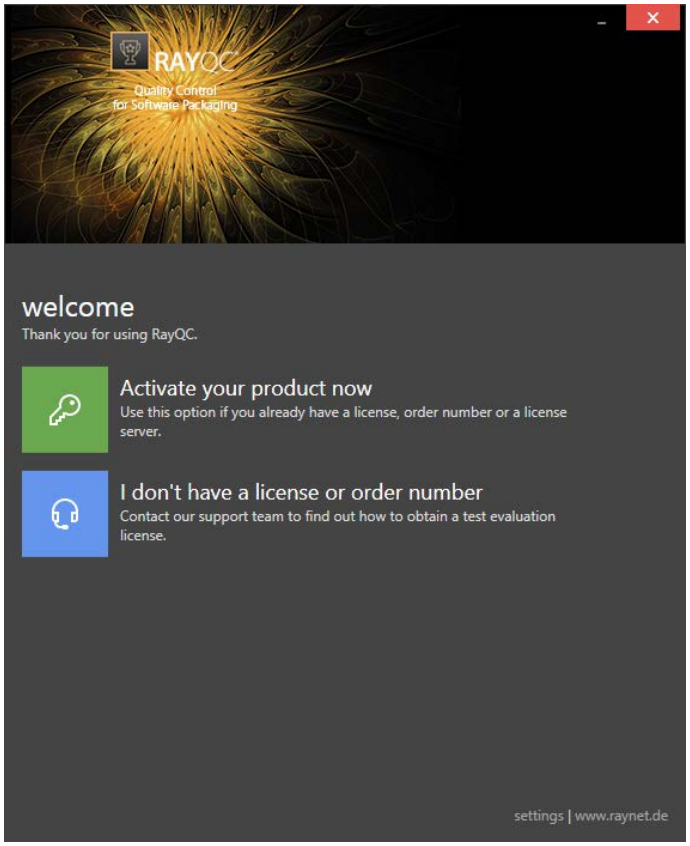
- Directly within the installation (see the *Installation Guide*)
  - By supplying the order number
  - By supplying an already generated license file (`.rsl` format)
- When the product is started for the first time.

If RayQC detects that no valid license is present on start-up, the license activation wizard will be shown after starting the main executable. The tool can be also started manually, by executing `Raynet.LicenseActivation.exe` from the main installation folder.

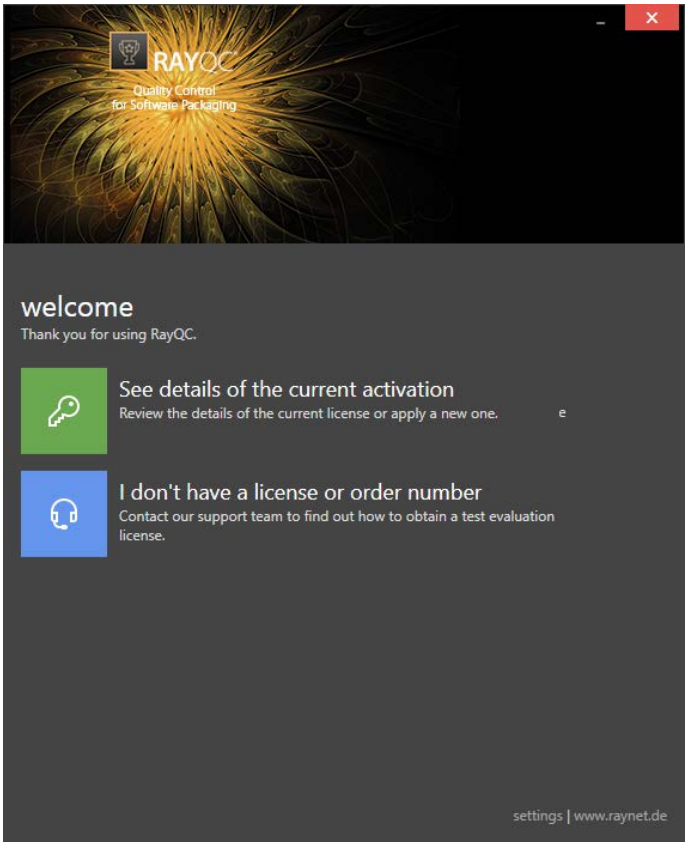
License wizard

This section describes the usage of the licensing wizard.

On the initial start of RayQC, the licensing wizard is shown. If the need to transfer an existing license arises, the license wizard can be started manually. There are a variety of ways in which a license can be activated and below they are described in detail.



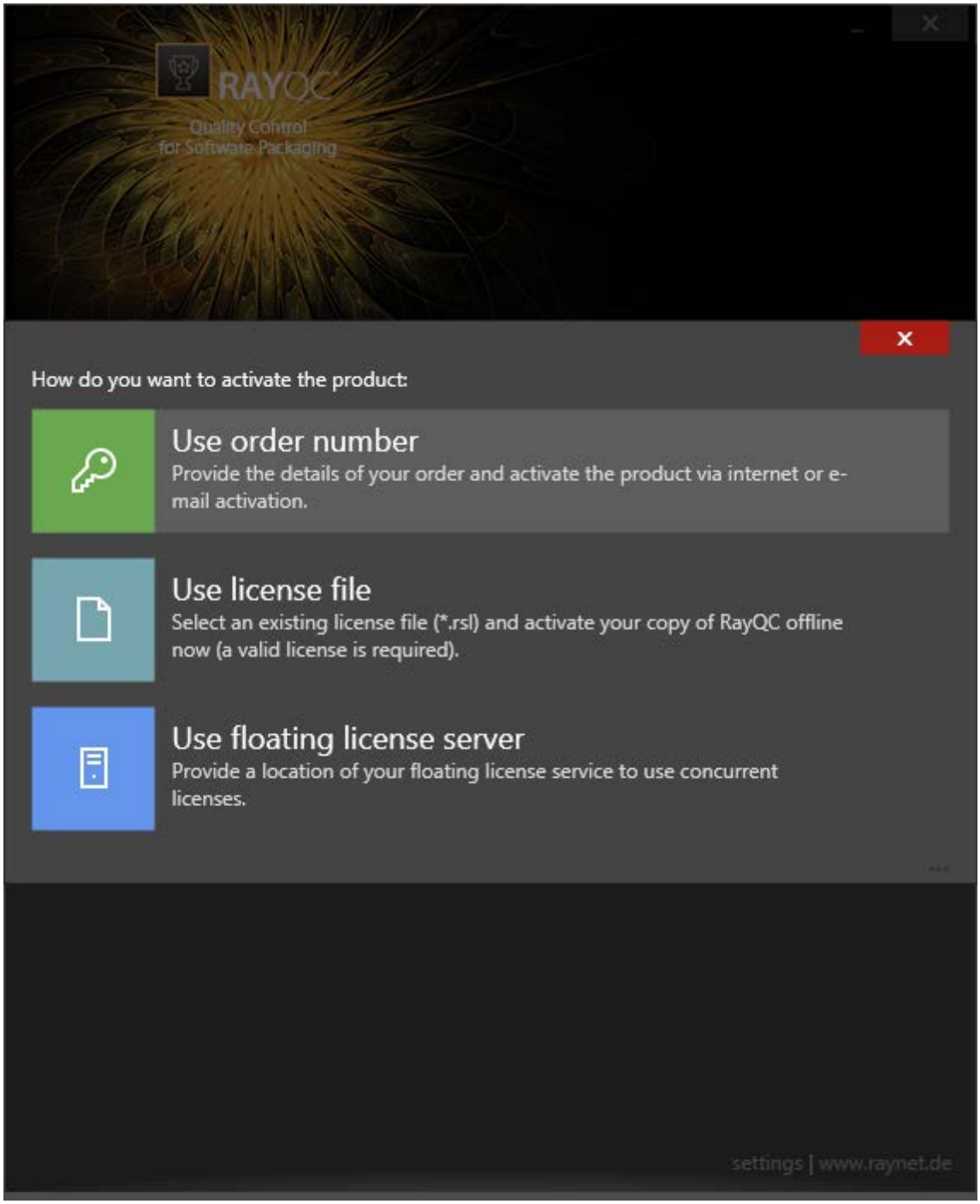
First time activation



The main screen when the product has been already activated

Activate your product now

This option should be used to activate the product using one of the following methods:



- **Use order number**
  Online activation using a valid order number received from Raynet (recommended for most users)

- **Use license file**
  Offline activation using a license file (`.rsl`) received from Raynet

- **Use floating license server**
  Activation using a local floating license server.

See details of the current activation

This options shows the details of the current activation. This option is only visible if the product has already been activated or if a floating license server has been configured

This option also allows to reactivate the product using a different order number or a different floating license server connection details.

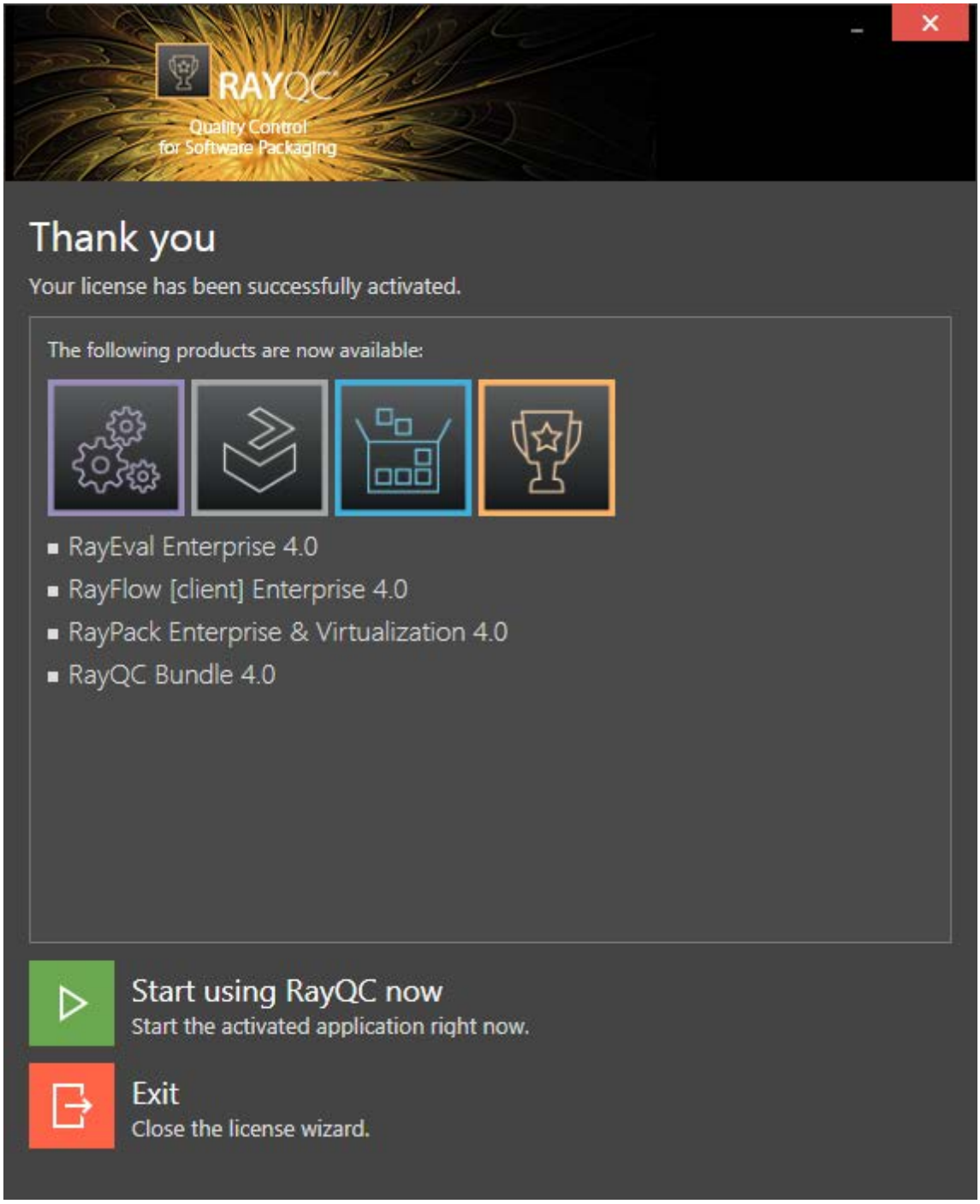
I don't have a license or order number

Choose this option if there is neither a license nor an order number available. For in-depth information please read this section. This option is only visible if the product has not been activated yet.

I want to take my activation back...

Use this option to deactivate a currently licensed version of RayQC. For in-depth information please read this section. This options is only visible if the product has been already activated.

Once the license file has been generated or copied to the correct location the following will be shown...



Note:
Depending on the license, more available products may be shown, as pictured above.

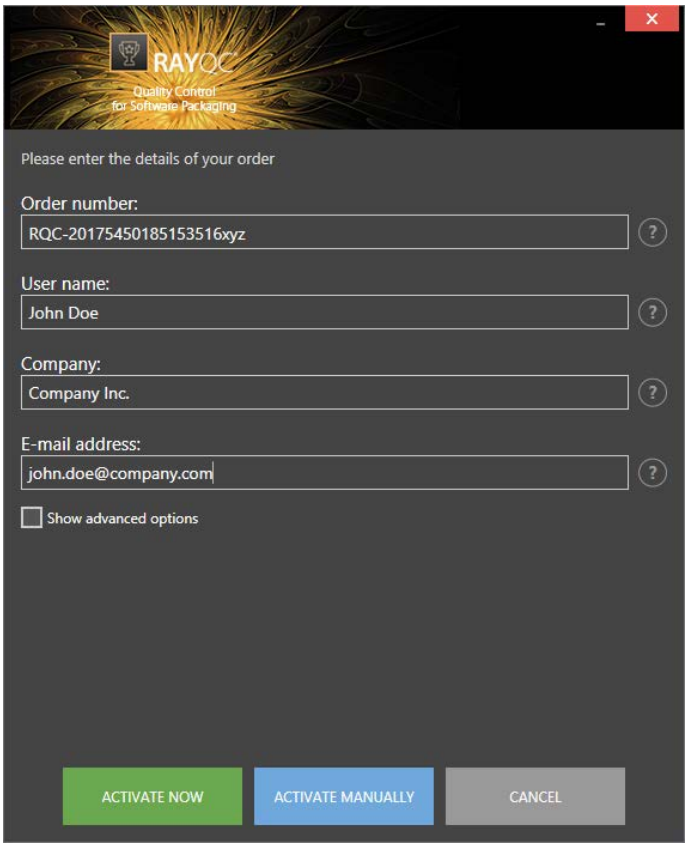
Then the option of starting RayQC or just closing the activation wizard is made available.

Troubleshooting

If any issues arise during the activation process, please contact our help desk to receive assistance in activating RayQC.

Use order number

RayQC can be activated either directly online or via email once the order number has been delivered. The activation process generates a license file (`*.rsl`) that is created (or must be copied) to the installation directory of RayQC (in the same location as the `RayQC.exe`). When performing an online activation, sufficient permissions must be readily available to allow the creation of the license file in the installation directory. The activation **binds** the license to the machine on which it was activated on. This is the only time that an active connection to the internet is required (if activating online).



Choosing the **ACTIVATE NOW** button, connects to the Raynet license server using the information provided and will dynamically generate a license file. Choosing the **ACTIVATE MANUALLY** button will open a dialog as shown here. Choosing the **CANCEL** button will abort the activation process.

Order details

Order number:
This is the unique order number received when RayQC has been purchased. If it is necessary to recover the order number, please contact our sales team.

User name:
This is the name of the user that is activating RayQC. It does not need to be the same name used to order RayQC.

Company:
This is the name of the company for which RayQC will be licensed. This name will appear in the License and Edition view of RayQC.

E-mail address:
This is the email address of the person that performs the activation. We respect the privacy of our customers, this email address will only be used by Raynet and only when there are any problems or important information regarding the license.

Advanced options

On choosing the advanced options check box, extended information and possibilities of the licensing and activation of RayQC are shown.

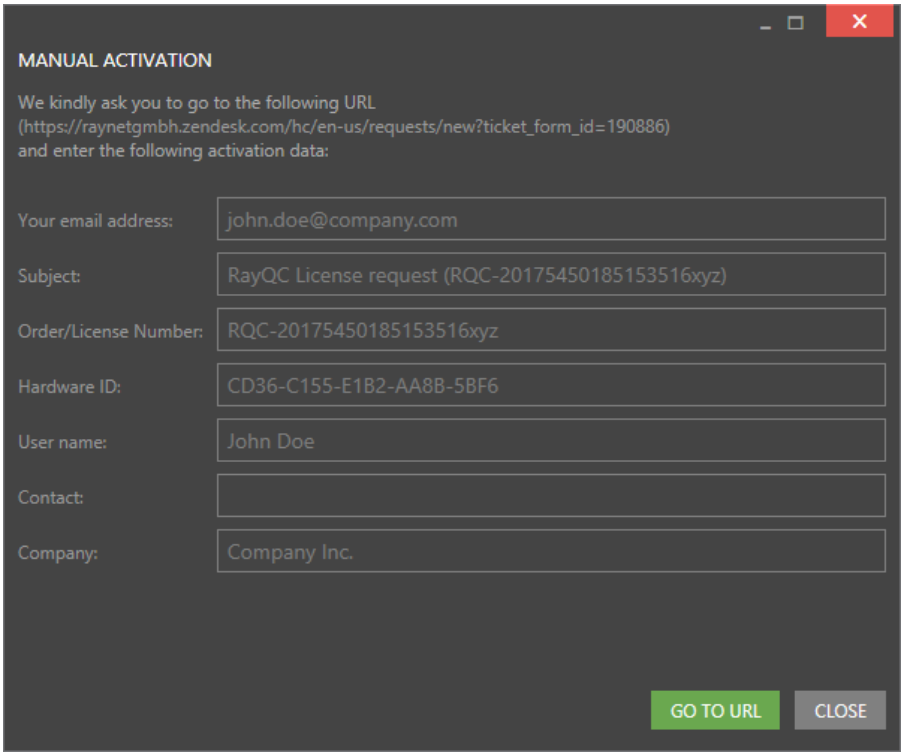
Hardware ID:
This is a ID calculated based on the hardware on which the activation is taking place on. The ID is unique, but cannot be used to personally identify a user. It is used to generate the license for the machine on which the activation process is carried out on.

Transfer the license
If this option is selected, the order number and details may be used to activate RayQC on a second machine, that has differing hardware (which obviously has a different Hardware ID). This assumes that RayQC has been deinstalledfrom the machine on which it was previously activated on. The transfer license functionality is logged on our license servers and is periodically checked to ensure that no abuse is made of this functionality.

If the license transfer is part of a regular maintenance and can therefore be prepared and scheduled, it is highly recommended to use the deactivation function first, to disconnect license and packaging machine. This is the standard way for transferring licenses. The option offered here is intended for unscheduled transfers, required if a machine, for whatever reason, cannot be accessed or used operational any longer.

Manual Activation

On choosing the manual activation, the dialog shown below is displayed.

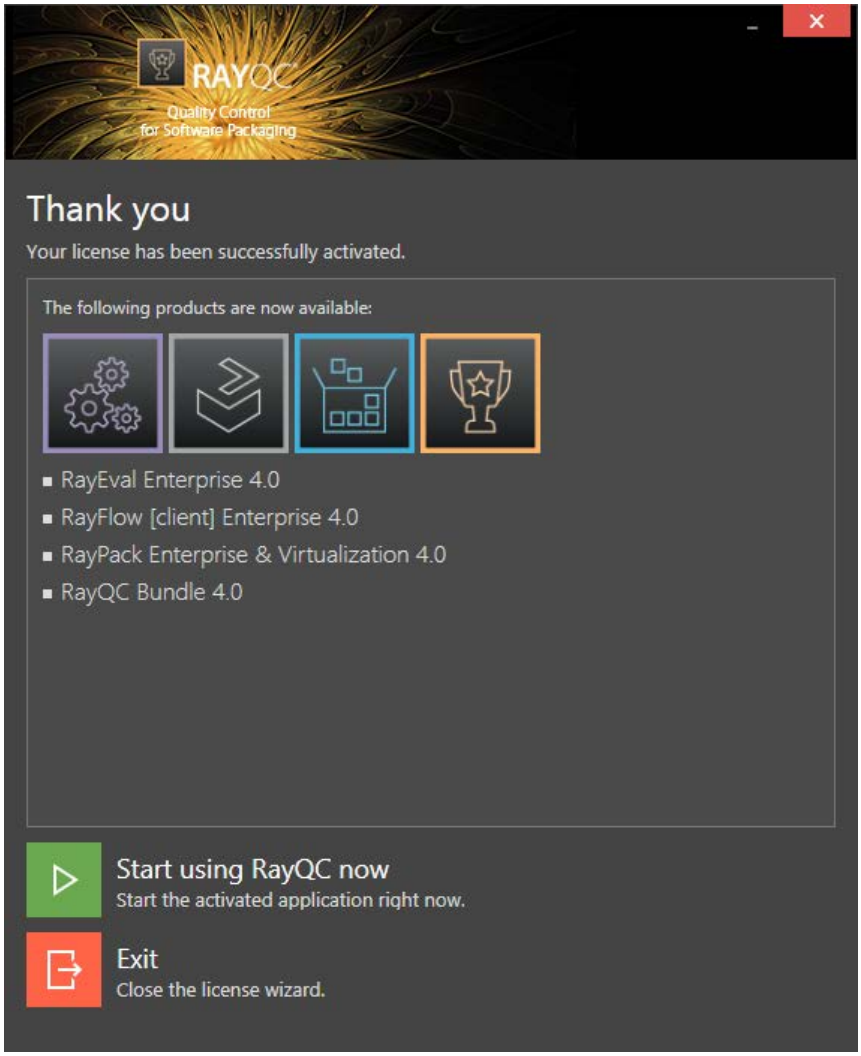


This basically shows the contents of the ticket form that will be opened at Raynet. If there is an internet connection available on the machine, click on the **GO TO URL** button to open the URL shown in the top of the window in the default browser of the system. After a File Order has been opened in the Raynet Support Panel, a license file will be delivered. Information of how to use this file are available here.

If no internet connection is present on the machine on which the activation process is taking place, copy the contents of the dialog onto a machine which has an internet connection and use the URL on that machine. On receiving the ticket, a license file will be generated and sent back. Information on how to use the license file can be found here.

Tip:
Please ensure that when copying the information from the **MANUAL ACTIVATION** dialog everything is added as shown above.

Once the license file has been generated the following will be shown:



Note:
Depending on the license, more available products may be shown. As an example, see the image above.

The option of starting RayQC or just closing the activation wizard are available now.

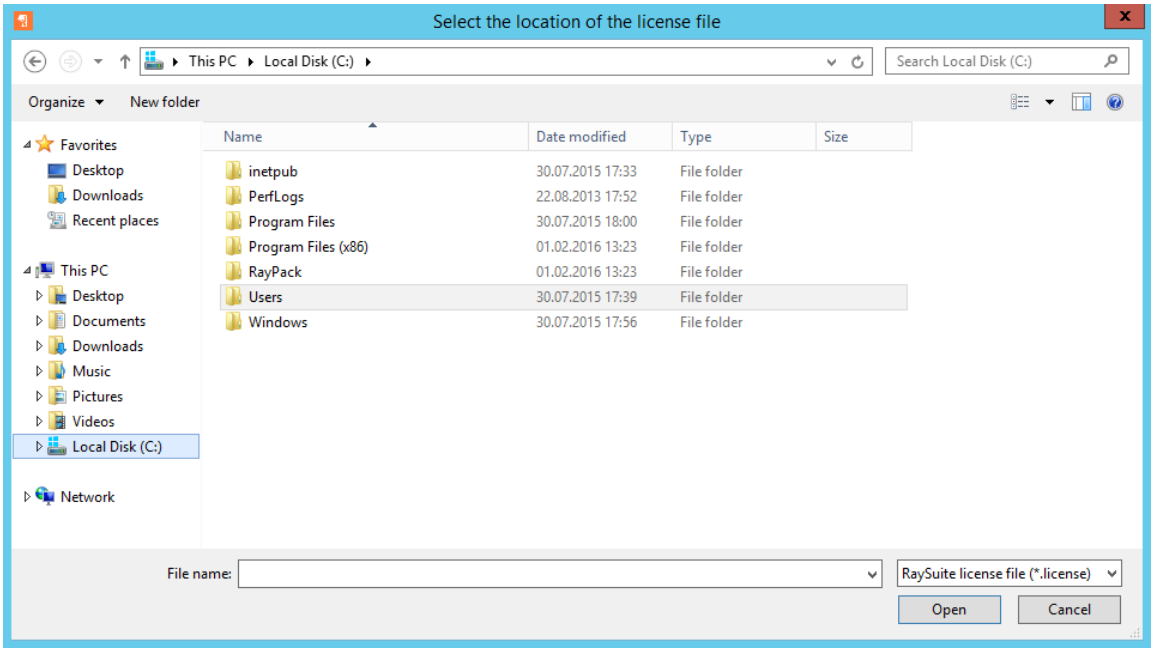
Troubleshooting

If there are any problems during the activation process, please contact our help desk for receiving assistance in activating RayQC.

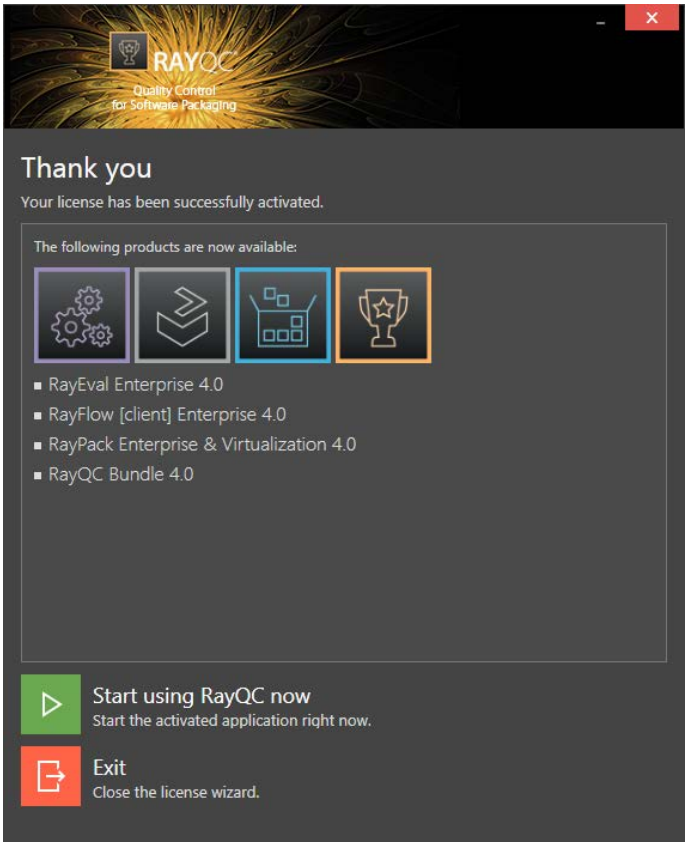
Use license file

If a license is already available, or a license file has been received as a result of activating RayQC via e-mail, then all that is required is to copy the license file into the installation directory of RayQC (the directory in which the `RayPack.exe` resides). Clicking on the **I have a license** button on the **License wizard** dialog opens a dialog box which allows to choose the license file. Once chosen, the file will be copied automatically to the RayQC installation directory. Please ensure that sufficient permissions to allow the creation/copying of a file to the

installation directory of RayQC are available.



Once the license file has been copied to the correct location the following will be shown:



Note:
Depending on the license, more available products may be shown. As an example, see the image above.

The option of starting RayQC or just closing the activation wizard are available now.

Troubleshooting

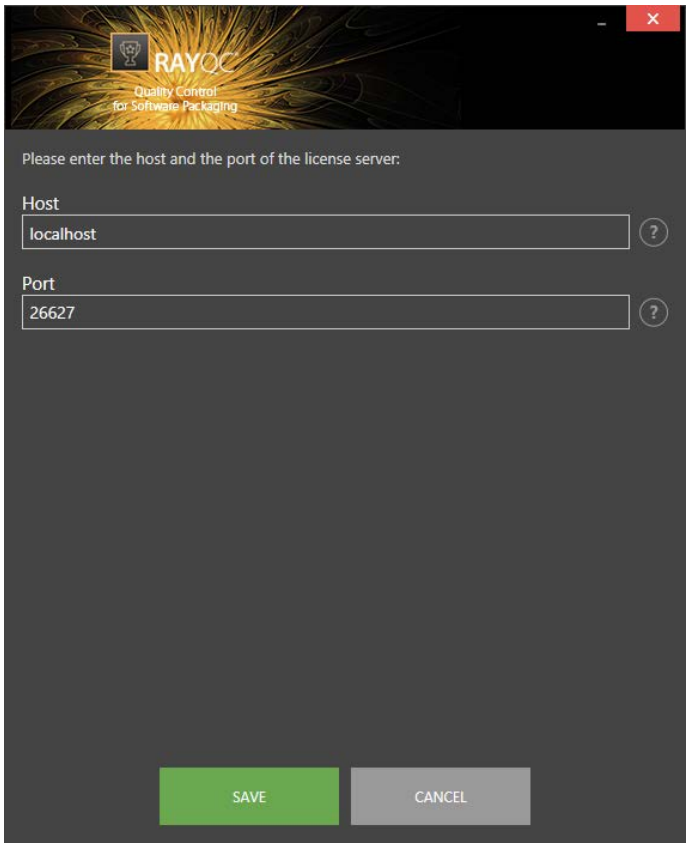
If there are any problems during the activation process, please contact our help desk for receiving assistance in activating RayQC.

Use floating license server

RayQC can be activated using a local floating license server. This requires that the server component is installed (the installation is available separately from the product installer).

Once the server is configured, the following details are required from the server administrator:

- Server name or IP address
- Configured port (by default `26627`)



Enter required values and confirm them by clicking on the **SAVE** button. The server will be contacted once to verify the correctness of the data. If the server is not available at that time, an option will be presented to write the data anyway.

Once the connection details are saved, please restart the product to activate it using the floating license server.

I do not have a license or order number

If neither a license or order number is available, then just simply register with Raynet to download an evaluation license for RayQC. This allows potential customers to test and work with RayQC before purchasing. Choosing **I don't have a license or order number** opens the Raynet website in the default browser, allowing potential customers to download an evaluation copy of RayQC.

I want to take my activation back

Deactivating an existing license for RayQC may be required if the packaging machine used has to be switched. Whenever there is a scheduled migration, e. g. when a virtual machine is transferred in a way that affects the **Hardware ID**, or when a physical machine is no longer used for packaging purposes, deactivating the license is the right thing to do.

To deactivate a licensed RayQC installation

1. Launch RayQC and open the license and edition tab of the **about** area.
2. Click on the **Open the license wizard** button on the lower left hand side of the application window.
3. Use the option **I want to take my activation back...**
4. Enter the **order number** that was originally used to activate RayQC on the current machine. It was part of the resources and information material delivered during product purchase.
5. If required, adjust the user name already entered into the input field **User name**. The users who activate and deactivate an installation do not necessarily have to be the same.
6. Click on **DEACTIVATE NOW**.

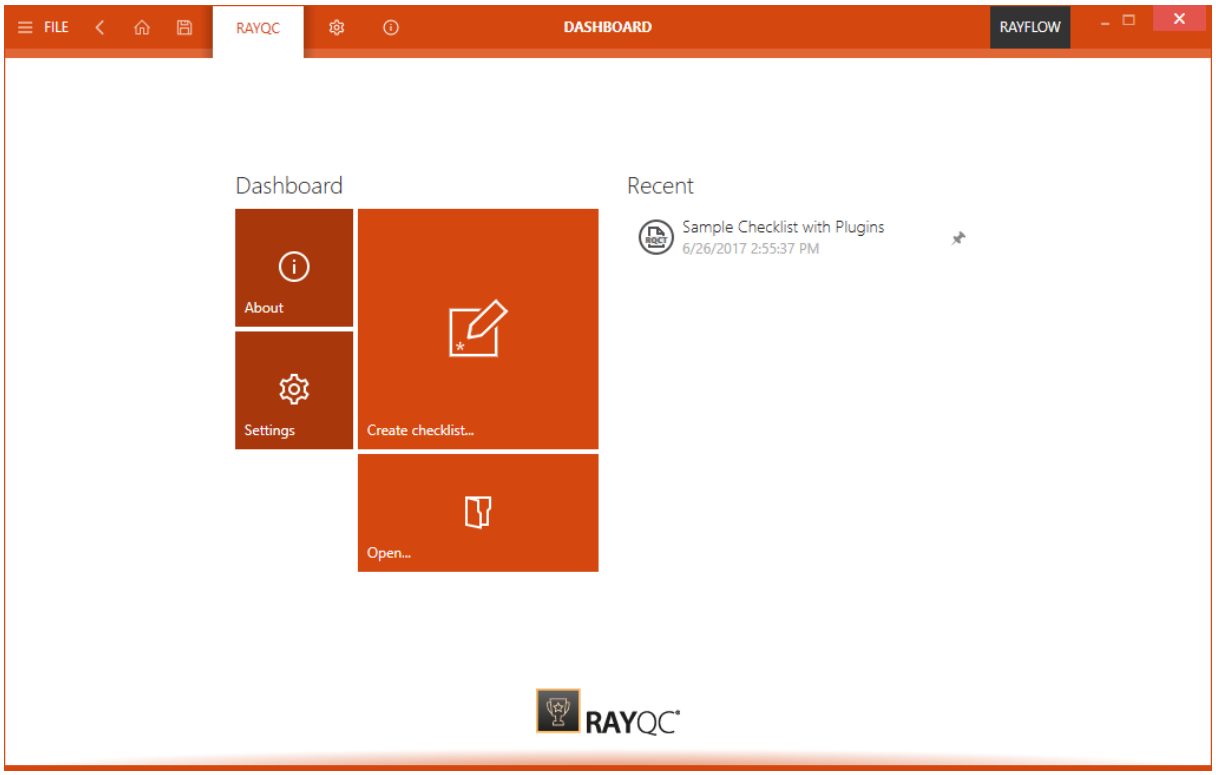
   The license wizard will connect to the Raynet licensing server and send the deactivation information. On success, the number of licenses available for activation, which are bound to the used order number, is incremented by one. With this new free license it is possible to activate any RayQC installation, on the current machine or any other.

Troubleshooting

If any problems during this process occur, please contact our help desk for receiving assistance in deactivating RayQC.

Working with RayQC

Once RayQC is installed on a machine, launching the program executable from the application installation directory (e. g., `C:\Program Files (x86)\RayQC\RayQC.exe`) invokes the RayQC home screen. If you have been working with any other RaySuite application before, the Home screen layout should be quite familiar, since it is a common interface type for all products of this family.



When RayQC is invoked as a tool via RayFlow, it usually does not load the home screen, but directly opens a checklist template or project file.

Once RayQC is up and running, the application screen contains some basic areas, which are always available - even though the actual content of the area varies from view to view:

The Main Toolbar

Throughout RayQC, the Main Toolbar is visible, which, dependent on the contents of the view shown adds or removes menu items dynamically. As a rule of thumb the items shown below are always present on the Main Toolbar.

Click on the items to see more information about the individual topics.

FILE

This opens the **FILE** menu. The **FILE** menu is dynamically created, dependent on what Tool is currently active. Please refer to *FILE menu* section to read more about it.

Back

This opens the previous screen.

Home

Choosing this button will return you to the Home Screen. If any projects and or files are opened, and there is a requirement to save any changes, you will be prompted to save before returning to the Home Screen.

Save

Saves the current file / project. This button is only active if any changes have been made that require saving.

View history

With a left-click on the arrow button, users navigate one step back within the history of recently opened views. Right-clicking the arrow displays the recently visited views, and allows returning to a specific view from that list. This view history is limited to those views without project relation, or with relation to the currently opened project. Thus, returning back to a view is not possible if it was called for a project that is no longer opened.

View title

The view title specifies which content is currently shown as part of the active application context and module.

Window title

The window title displays the current scope of activity. If an editor is active, the file name of the currently opened project is part of the window title as well.

RayFlowStatus

This window tile displays the status of the RayFlow connection. It is either black showing RayFlow or red and showing the currently logged in user. By clicking on this tile a user can either log in or out.

Standard window controls

The standard window controls allow minimizing, maximizing, resizing and closing the application window. The availability of each control follows the Windows schema for standard controls as known from any desktop application.

Application Context

The illustrations above show the main application context status, which is RayQC. Depending on the set of licensed modules and add-ons, further contexts may be added.

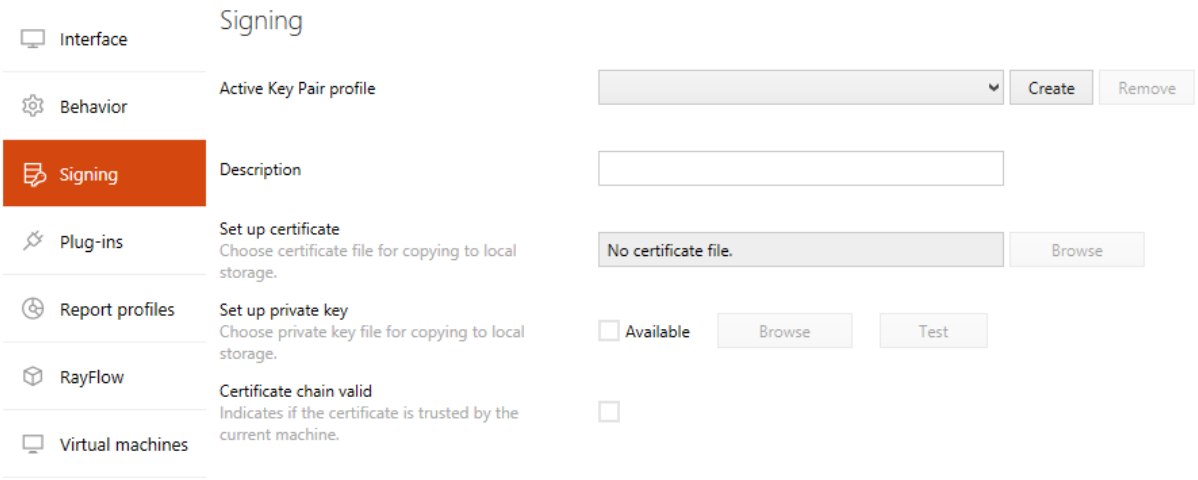
SETTINGS
Opens the **settings** for RayQC.

ABOUT
Contains information about the currently active product instance.

HELP
Opens this help file.

The content area

The content area is the core for actual application activity. According to the active application context, it contains the dashboard (as shown in the screenshot at the beginning of this topic), the checklist viewer or editor, dialogs for settings editing, and so on.



The content area of the settings view signing. The other views of the settings area are available by clicking on their tab labels: behavior, connections, plugins and report profiles

The swipe bars

Whilst the Main Toolbar is designed to provide access to general application functionality, there are additional swipe bars for local activity options at the bottom of some views. This may either be a set of buttons for running checklist evaluations, options for saving or discarding settings, or navigation helpers that allow switching between related views, such as between the checklist viewer and editor.

What's next?

Now that the main areas of the RayQC user interface have been introduced, it is time to do the same with the different types of objects that may be handled within the product. Please go ahead and read the following section Objects for details regarding the differences between Checklists, Templates, and Projects used within RayQC.

Objects

Since RayQC is a checklist based application for quality assurance related test executions, it may actually not be overly surprising to find an object called "RayQC checklist". However, recognizing the differences between checklists and templates, or checklists and projects may be a bit tricky without further explanation. Therefore, here is a brief outline of the objects and their related file name extensions in RayQC.

RayQC Checklist and RayQC Template (*.rqct)

The file types checklist and template exist due to historical reasons. In fact, both names refer to the very same object type RQCT, which is the abbreviation for RayQC Template. The idea is that when a new checklist is created, RayQC uses a default template to generate the basic checklist structure. As long as the checklist has not been executed within RayQC, it is nothing more than a manipulated version based on the original template file. Therefore, when the labels checklist and template are used within the RayQC user interface or documentation, both actually intend to refer to the very same type of file.

The RQCT files used in RayQC 6.1 are ZIP containers that contain the XML checklist file (`checklist.xml`) as well as all other resources required to run the checklist in RayQC: plug-ins, help files, images, etc. are stored within dedicated directories wrapped in the zip container.

Checklist group

A group is a logical bundle of elements within a checklist. The elements of a specific group are handled as a unit, and thus will be displayed in a shared container area when a checklist (template or project) is opened within RayQC. Each group has its own header (including index number, title, and description) and content section (including the groups checklist elements).

Checklist element

An element is a single item within a checklist group. There are four basic types of elements that may be used within checklists: Information, Data Field, Checkpoint, and Multi-Option. The type of element determines the input and result options users face when they actually run a checklist instance (which in fact is called a project, see below). Further details about the specifications of the element types are provided within the Element types chapter later on.

Be aware:
Former releases use different names for the checklist element types. In order to avoid misunderstandings and provide clear names, a change according to this mapping has been done:

- Comment > Information
- User Comment > Data Field
- Checkpoint Entry > Checkpoint
- Multi-Option Entry > Multi-Option

RayQC Project (*.rqcp)

A RayQC project file (abbreviated by RQCP) is a ZIP container, just like RQCTs. In addition to the checklist resources it is extended with the current execution status for this specific checklist instance. Therefore, when a checklist file is opened in RayQC, it is automatically converted into a temporary project file that resides within the session memory. Saving changes made to a checklist automatically preselects the project data type RQCP as its file extension. Only when users manually change the default selection to RayQC Template (`*.rqct`), the original checklist file is physically overwritten.

In order to decide which target format is desired, users have to be sure about what exactly they want to save: The status-free checklist itself, which may later be used as the source for several new projects, or the current state of a specific checklist run, which has to be a project file.

Be aware:
To distinguish evaluation status and element structure information out of a project file, users have to open the project file with a tool like WinZip or 7ZIP. Once opened this way, the checklist status and structure are both directly available as XML files. However, it is not recommended to manipulate the files somewhere outside of the RayQC application scope, since the result will very likely not be in compliance with its schema and value domain restrictions anymore.

plug-ins

Checklists may include different plug-ins, whilst each checklist element (see paragraph above) may be equipped with a maximum of one plug-in. plug-ins enable automated checklist execution, since they provide scripted logic that may be called by the RayQC plug-in interface.

Internal plug-ins

Internal plug-ins have been defined within the RayQC application and are ready for usage within any kind of checklist. Users simply add the plug-in to an element, select the required control options and parameter values, and the scripted logic is executed at checklist project run. Within the current version of RayQC, there are nine Internal plug-ins available for out of the box usage:

- Command plug-in
- File plug-in
- Folder plug-in
- IniFile plug-in
- Local System plug-in
- Logic plug-in
- MSI plug-in
- RayFlow plug-in
- Registry plug-in
- Advanced plug-in

External plug-ins (PowerShell script and Manifest XML)

Whilst Internal plug-ins have been predefined for general usage within RayQC checklists by the Raynet development team, External plug-ins may be created and integrated by RayQC users themselves. They usually provide specific logic for a test criterion required by a single checklist, or a whole checklist group. Depending on the test and assurance needs of each individual customer, these External plug-ins are limited only by the experience the creator has in writing PowerShell scripts, and the given restrictions of the work environment (e. g. access restrictions, best practices, and the like).

Each External plug-in must consist of at least one PowerShell script file, containing the plug-in logic, and an XML manifest file (`Manifest.xml`), declaring the plug-in interface for direct interaction and communication with RayQC. These files have to be stored within the same parent directory that is named according to the plug-in name. The type of supported PowerShell scripts is not determined by RayQC itself, but the PowerShell interpreter on the machine the plug-ins are executed on.

RayQC automatically checks the application installation directory (e. g., `C:\Program Files (x86)\RayQC\`) for the existence of a `\plug-ins\` directory. If it exists, the content is analyzed in order to find plug-ins. These **global external plug-ins** may be used from within any checklist created and executed from the machine that hosts the current RayQC instance.

Another type of external plug-in is the **local external plug-in**. These ones are basically of the same structure (PowerShell script and manifest within the same folder which is named after the plug-in name), but have to reside within the plug-ins directory that is part of the RQCT / RQCP ZIP container. Therefore, local external plug-ins move with their parent checklists and projects, as they are essential parts of them. They are more portable than global external plug-ins, which reside on one specific QC device.

Please refer to the Plug-ins section of this document for details on how to actually use plug-ins within checklists.

Note:
Please note that the possibility of using external PowerShell plug-ins within a checklist is only available with RayQC Enterprise Edition.

Condition

Conditional statements are an important aspect of dynamic checklist evaluation. In RayQC, users are able to define conditions, as combination of several conditional statements, which may decide about the actual evaluation path of a checklist. For example: If the result of Checkpoint A is true and the result of Multi-Option B is false, checklist group X has to be evaluated. If not, checklist group X is invisible and does not affect the result of the checklist.

The logical idea of conditions in RayQC 6.1 is based upon the so called "disjunctive normal form" (DNF), which allow building any kind of condition as a combination of ORs between ANDs. To be more precise: A DNF is a disjunction of conjunctive clauses. Within our checklist editor, clauses are called buckets. Within each bucket, there may be several conditional statements, but all have to evaluate to true in order to let the bucket evaluation result become true. If the condition for an element contains more than one of those buckets, it is sufficient to have one bucket evaluate as true to let the whole condition evaluate as true. So you see, the buckets are made up of ANDs, and are chained by ORs.

Please refer to the Conditions section of this document for details on how to actually use conditions within checklists.

What's next?

Now that the most important objects within the RayQC universe are a bit more familiar, it is time to see how they are used in productive work scenarios. Go ahead and get details about typical workflows in RayQC.

Typical Workflows

Working with RayQC is basically divided into two main areas of activity: Preparing checklists and executing them as projects. Therefore, the main workflows are bound to these core tasks:

- Creating a new checklist
- Editing a checklist
- Running a checklist as project
- Exporting evaluation result reports

In order to accomplish these workflows, some additional tasks have to be prepared in advance:

- Define settings for RayQC usage
- Define external plug-ins to use within checklists

Please click on the sections given above to get details on the specific workflow.

Be aware:
RayQC is available in different product editions. Please note that the license that has been used to activate the current application instance takes direct effect on the set of features which are effectively available. Therefore, some of the mentioned workflows and activities may not be executable with the current RayQC installation. Please verify the actual set of licensed features by reviewing the information provided within the license and edition tab of the about view, or contact your Raynet sales or consulting representative.

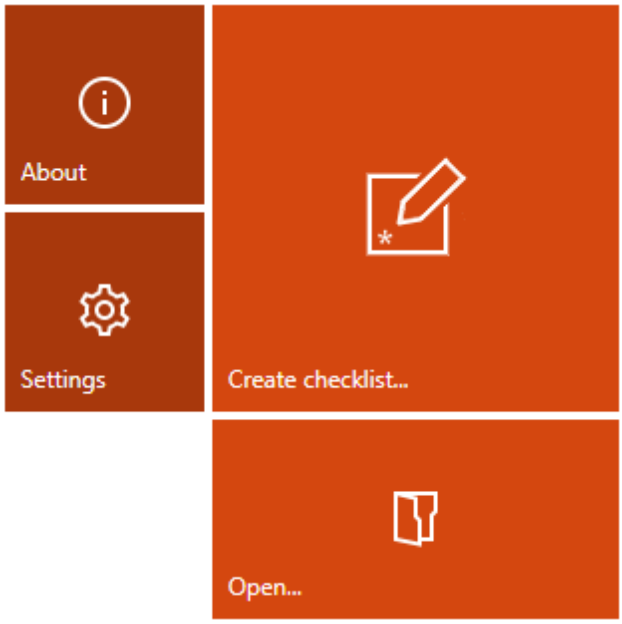
What's next?

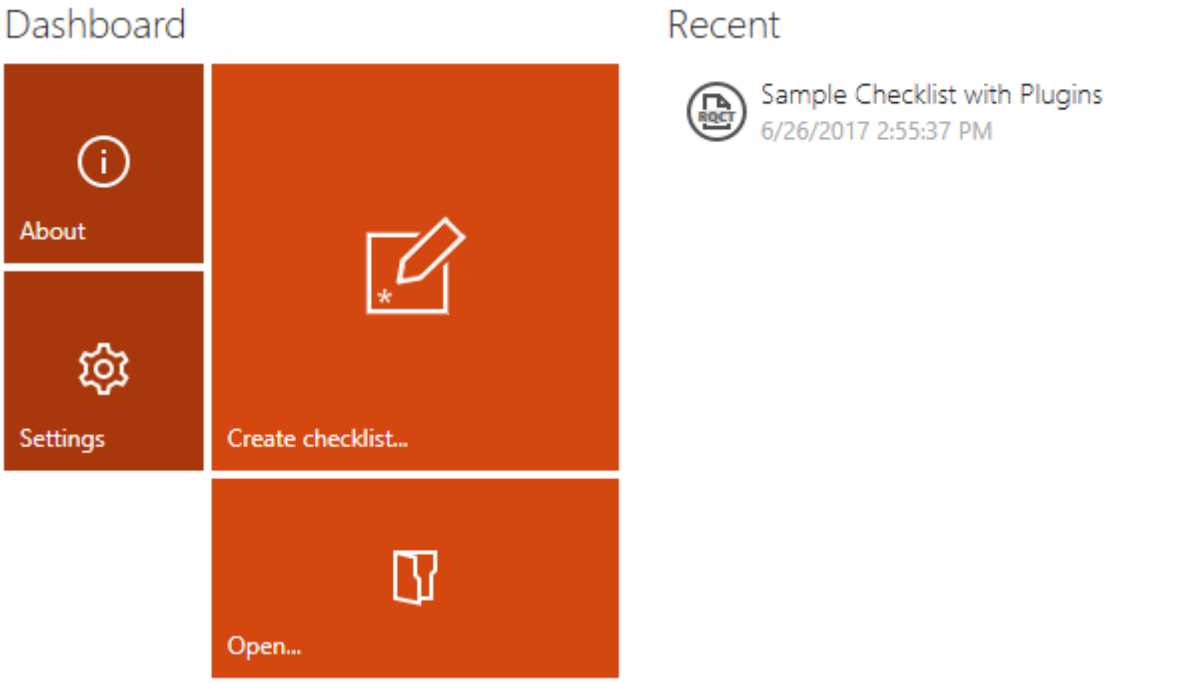
Now that the most common activities and objects these are applied to are known, it is time to take a look at the user interface that is present when projects or checklists are opened in RayQC. As outlined before, the starting point of RayQC daily business is the Home screen with its dashboard and recent list. Advanced RaySuite users may want to skip the in depth description of the Home screen, and directly go ahead to read about the Checklist Viewer (for project evaluation) and the Checklist Editor (for checklist design).

The Home Screen

The Home screen of RayQC contains a **Dashboard** and an optional **Recent** section for quick file access. Whenever the application is launched without a specific file trigger, this view is the starting point for the new RayQC session.

The contents of the Home screen may be configured via the **Settings** area: Users can decide whether or not the **Recent** list is visible at the right-hand side of the **Dashboard**. The screenshots below show the two operational modes: the first one without the **Recent** list, the second one with the visible **Recent** list.





The Dashboard Tiles

Clicking any of the tiles on the **Dashboard** opens a specific RayQC view:

- For first time users it is highly recommended to take a look at the *Get Started Guide*, available from the **About** tile.
- After gaining overall knowledge of RayQC, a walk through the **Settings** section is due. Especially for those RayQC instances, that have to operate connected to a RayFlow server.
- With a properly set up configuration, the first checklist should be created, which requires clicking on the **Create checklist...** tile.
- Once a checklist is created, it may be opened for evaluation execution or structural editing. Either way, the **Open...** tile allows to select a checklist template from the file system.

The Recent List

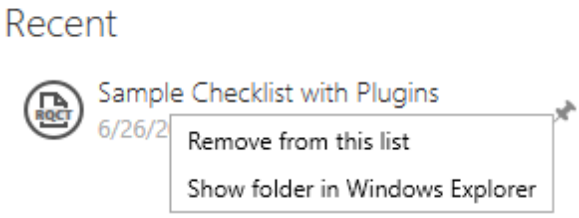
If configured to be visible, the **Recent** list is shown on the right-hand side of the **Dashboard** content area. It lists recently accessed checklists and files of the currently logged in user.

The list contains the last 8 files that have been opened in RayQC by default. The ninth opening of a file removes the oldest item from the recent list in order to add the new one. Clicking on a list item immediately opens the related file in RayQC.

Be aware:
If the physical file that is connected to a recent list item has been removed or renamed, it cannot be found and opened any more. Usually RayQC checks the availability of the recent objects whenever the Home screen is loaded.
However, if an item becomes physically unavailable whilst the Home screen is active, RayQC will display a message when the item is clicked, and ask the user if the file should be removed from the recent list. It is recommended to do so.

The default behavior outlined above may be changed by pinning files to the **Recent** list. At the left hand side of each recent list item, there is a pin icon. It is gray for items that are not pinned to the list, and black for those who are currently pinned for permanent visibility. Clicking the pin icon switches between the pinned and unpinned status of an item.

Note:
Unpinning an item from the recent list may remove it from the visible list scope if it has not been opened in RayQC for a while. The rule is that all pinned files are shown ordered by last opening date, and the free slots of the list are filled with the lately opened files, which are as well ordered by last opening time. Therefore, if a pinned file has not been opened lately, it may very well be crowded out by other files.



The file icons used within the list indicate the specific file type by its color: template files are represented by dark orange icons, projects have a lighter orange and a checkmark within their icon.

Right-clicking an item present within the recent list reveals the context menu. Users have three options to select from:

- **Remove from this list**: Removes the checklist item from the recent list.
- **Show folder in Windows Explorer**: Opens a new explorer window which shows the folder containing the checklist.

The Checklist Viewer

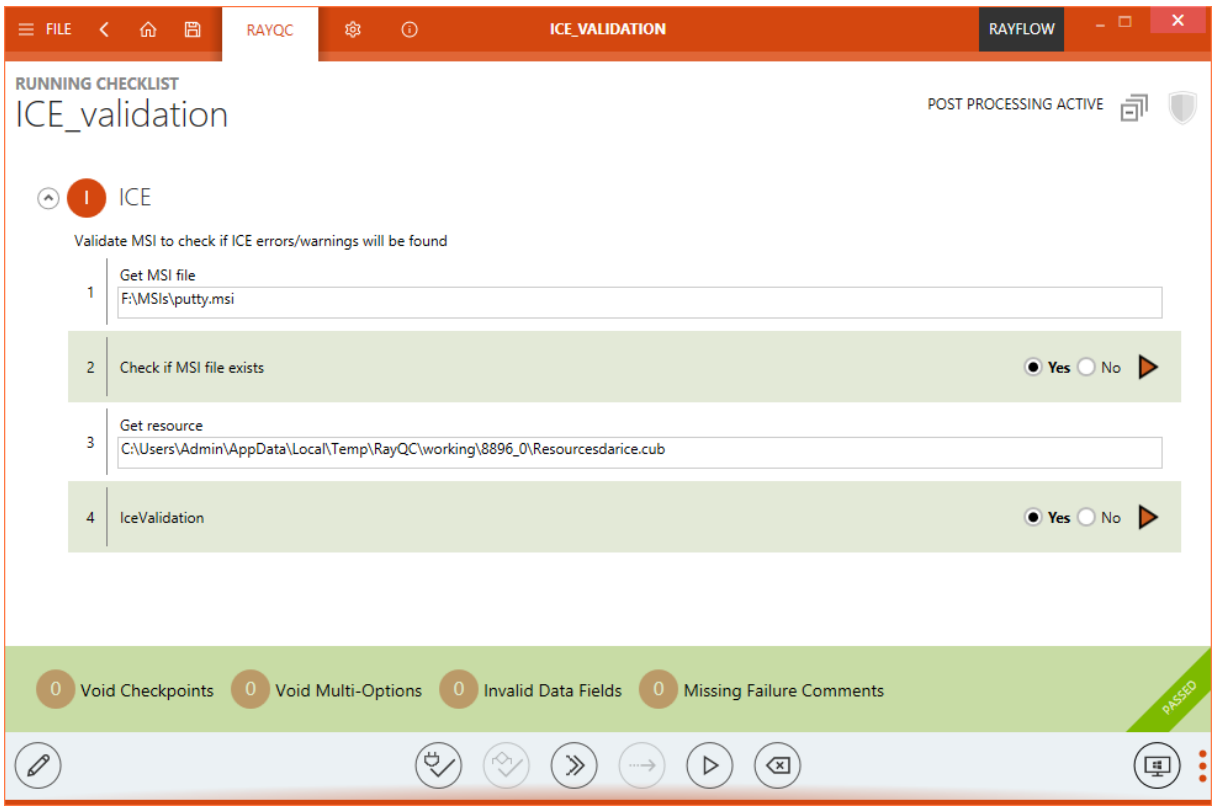
The Checklist viewer is the user interface view that is active when a checklist is opened for review / execution within RayQC. Therefore, to display the Checklist Viewer:

- Use the open checklist tile from the **Dashboard** on the Home screen.
- Click on one of the template items from the **Recent** list on the Home screen.
- Use the Open view from the **FILE** menu and select the template file type.
- Hit **Control** + **O** to browse the Windows system for a project file.

Be aware:
Whenever a file is opened in RayQC, it is checked for structural validity. Files that contain not well formed source structures are rejected, and cannot be displayed within the Checklist Viewer or Editor.

Especially when users try to open files, which have originally been saved in one of the *deprecated RayQC file formats*, they are rejected if they do not match the current template or project file format restrictions.

Either way, the selected file (checklist or project) is opened within the Checklist Viewer. The screenshot below shows this user interface state with one of the RayQC sample checklists opened:



The application window is separated into different areas with specialized functionality and data as requested for the optimal user interaction support. Please refer to the *Working with RayQC* section for details regarding global interface objects and their usual naming in RayQC.

Main Toolbar

The menu bar has been extended with the actual checklist name of the currently viewed project on the left-hand side.

Be aware:
Any checklist opened in the Checklist Viewer is automatically transformed into a RayQC project, residing within the temporary system memory until it is saved permanently on any system location. Leaving the Checklist Viewer without saving the changes made to the actual checklist run / evaluation results, does not take any effect on the underlying checklist. Saving the changes of the current checklist evaluation / run creates a RayQC project file type by default. To manipulate the checklist structure underneath the RayQC project has to be executed within the Checklist Editor view. Please refer to the Checklist Editor section within this document to get more details on how to manipulate checklist structures.

Content area

Checklist area

The actual content area begins with the listing of the checklist items in their group containers. According to the order and nesting designed within the Checklist Editor, all items that do not depend on conditional options are displayed with their type specific input controls. Users may directly enter the results of their checklist execution, call help files for further information on the checklist in general, or a specific checklist element. plug-ins can be executed and comments may be entered. All in all, the checklist area is the place where the actual end-user evaluation works with RayQC.

Checklist elements are equipped with an index value, which is unique within all items of the same parent group container. The index does not only give information about the position of the element within the item sequence of the box, but also about the indentation level of the item. The index is a multi-level indicator value, with a colon separating the different tree levels. For example: An item with the index value 2.3 is the third child of the second checklist item within a group.

The elements within a group are displayed with alternating background colors in order to support easy visual element distinction. Once Checkpoint elements have been evaluated, an additional background color markup is applied to them: If it failed, the background turns slightly orange, if it passed the background turns slightly green.

Please refer to the *Checklist Structures* section for details regarding the different options that may be applied towards checklist design and functionality.

Task bar

The task bar below the checklist elements displays a set of **status indicators**, which support users in their aim to completely evaluate the checklist elements with the least possible effort.
At the left-hand side, there are four **status buttons**, indicating the completeness of each checklist element group:

- Missing Entry Selections
- Missing Multi-Option Selections
- Empty Data Fields
- Missing Failure Comments

A number greater than zero in the middle of the circle button indicates that the checklist evaluation task is not complete yet, since there is at least one more user input or activity required (See the right button presented within the illustration above). Whenever a task group is incomplete, the circle button background is dark orange, whilst buttons for completed task groups are shown with a light orange background color (See the left button presented within the illustration above). The color coding is designed as a quick hint for missing information, whilst the number indicates the amount of open tasks that need completion before the checklist evaluation / run is complete.

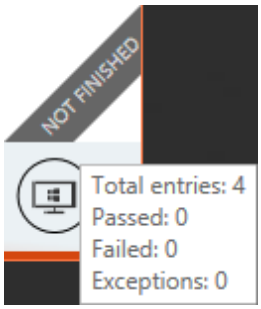
Clicking on one of the task buttons with an orange background color focuses the first incomplete checklist element according to the task group title. For example: The button for missing Multi-Option selections is orange

and displays a value of 3. Therefore, there are 3 checklist elements of type Multi-Option that have not been answered yet. Clicking on the dark orange button scrolls the checklist area to a scope that displays the first open Multi-Option element. As soon as the user makes a selection for that element, the number of missing items displayed in the button is decreased by one. Clicking the button again loads the next Multi-Option checklist element that needs user interaction into the visible scope of the checklist area.

Since checklists may become comprehensive and complex for in-depth quality assurance procedures, the task bar indicators are helpers for those situations where checklists may not be fully evaluated within one working session. They also help to keep track on conditionally displayed checklist elements. Therefore, it is recommended to use the task bar buttons after the initial run through the entire checklist and complete the tasks with their guidance.

Another visual indicator within the task bar is the **result ribbon** at the right-hand side. When a checklist is opened for execution as a project for the first time, the default ribbon state is a gray background and the label **NOT FINISHED**.

Hovering over the ribbon reveals a summary of checklist properties, such as the number of successful and failed checks, as well as the total number of currently available checklist elements. This total is updated according to the actual state of element availability as derived from conditional statement examination.

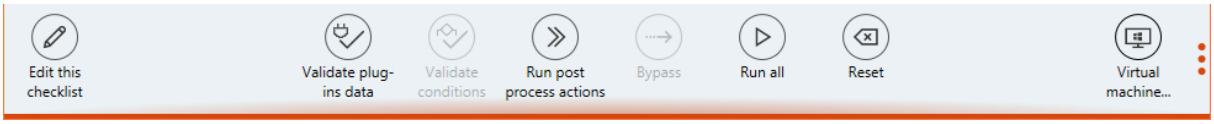


Once all checklist elements are evaluated, the result of the checklist run is available: The checklist test has either been passed (indicated by a green background color and the label **PASSED** for the ribbon), or failed (indicated by an orange background color and the label **FAILED** for the ribbon).

Well, actually there are some more constellations and conditions that decide whether a checklist has been passed or failed, but please read about them within the *Checklist Structures* chapter. For now, it is just required to know that the ribbon at the right-hand side of the task bar actually indicates the result of a checklist project evaluation.

Swipe bar

The swipe bar contains controls to use in combination with the currently opened resource file:



Edit this checklist

Clicking on this button opens the Checklist Editor, with the template of the currently visible checklist already loaded for manipulation. As a handy alternative, use the swift **Shift** + **Tab** shortcut to switch between the Viewer and Editor mode.

Validate plug-ins data

If the currently opened project contains plug-in calls, hitting this button checks whether they are logically correct or not. Possible reasons for conflicts are:

- Wrong input parameter values or formats
- Invalid relations between elements and plug-ins (order of usage)

The result of the plug-in check is a message dialog, stating that all plug-in integrations are flawless, or that issues have occurred. If there are issues, the message dialog may be expanded to display details on the conflicts found within the plug-in definitions. In this case it is recommended to change the mentioned plug-in parameters and recheck the checklist until all issues have been cleared.

Validate conditions

If the currently opened project contains conditions, hitting this button checks whether they are logically correct or not. Possible reasons for conflicts are:

- A defined condition combination will never occur, e. g. because two or more condition terms demand different results from the very same checklist item.
- Condition terms are defined as duplicates, e. g. one term demands result A from checklist item number 1, and the next term of the same conditional construction also demands result A from checklist item number 1.

The result of the condition check is a message dialog, stating that all conditions are flawless, or that issues have occurred. If there are issues, the message dialog may be expanded to display details on the conflicts found within the conditional statements. In this case it is recommended to change the mentioned condition terms and re-check the whole condition set of the checklist until all issues have been cleared.

Run post process actions

This button can be enabled for a checklist via the **Enable post process actions.** checkbox. This checkbox is available under the **Post Processing** tab of the checklist editor. Based on the conditions defined under this tab, a set of predefined actions will be executed, either by clicking upon this button or automatically when a user selects the **Run All** button. Furthermore, post process action **Create and upload report to RayFlow** can also be initiated via the command line switch.

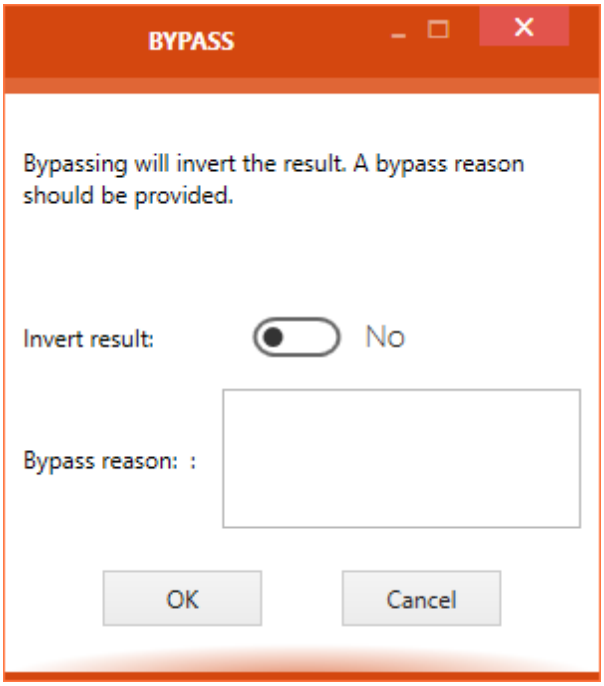
For further information on configuration of post processing actions, please refer to the *Post Processing* section of the chapter *Checklist Structures*.

Bypass

The bypass option is available if the checklist is configured to allow manual result bypassing. For those projects that base on checklists with bypass permission, the bypass button becomes available once the checklist elements have been fully evaluated and a result (**PASSED** or **FAILED**) is displayed in the ribbon at the right-hand side of the task bar (see *above*).

Hitting the Bypass button displays the **BYPASS** dialog. Within this dialog, the user should write a note why the bypass was required. Setting the result bypass state to **Bypass** (by clicking on the radio control item option **Bypass**) reverts the original checklist project result (e. g. from original result **FAILED** to the new, bypassed result **PASSED**).

Clicking the **OK** button within the **BYPASS** dialog saves the new result settings and closes the **BYPASS** dialog.



It is possible to revert the bypass, which restores the original checklist evaluation result state again. To do so, users call the **BYPASS** dialog and move the Invert result toggle slide to **YES**.
However, conditions regarding required circumstances for bypassing and bypass revocation have to be defined by the creator of the original checklist template used within the current project instance.

Run all

If a project contains checklist items with plug-in usage, hitting the **Run All** button automatically executes all plug-ins at once. All plug-ins are run in turn, beginning from the one that has the highest position within the checklist tree. The execution is done sequential, which means that plug-in B will start when plug-in A has finished. Plug-in A and B will not be executed in parallel. Therefore, plug-ins that require input based on earlier plug-in execution results will always rely on current results when the **Run All** function is used.

If the **Run All** button is used after one or more plug-ins have been run (manually or automatically), RayQC displays a dialog, asking the user if former results should be overridden by the new execution, or if the **Run All** execution should be aborted instead.

Be aware:
On Run All, element's visibility will be evaluated after each plug-in execution in order to process the checklist correctly. Only elements that require a manual input, will not be processed and their condition will not be fulfilled by any plug-in.

Reset

Using the **Reset** button clears all checklist element results that are part of the respective checklist in the Checklist Viewer. If a checklist has been opened in a project scope including a former evaluation state, resetting does not reset to that state, but to the default initial state as given from the checklist element definitions.

Be aware:
Reset does not reset the original checklist element settings of a checklist under construction. It simply removes the result information entered within the Checklist Viewer mode. Adjustments made towards the checklist elements within the Checklist Editor are kept unchanged.

Virtual machines

By using the **Virtual machines** switch it is possible to execute parts or whole checklists on a virtual machine.

Show / Hide swipe bar labels

As already described within the *Working with RayQC* section, the swipe bar comes in two display modes: expanded and collapsed. The expanded mode displays a label for each button on the swipe bar, whilst the collapsed mode contains only the buttons without the labels. Hitting the button with the three vertical dots at the right-hand side of the swipe bar switches between the expanded and collapsed swipe task bar display modes.

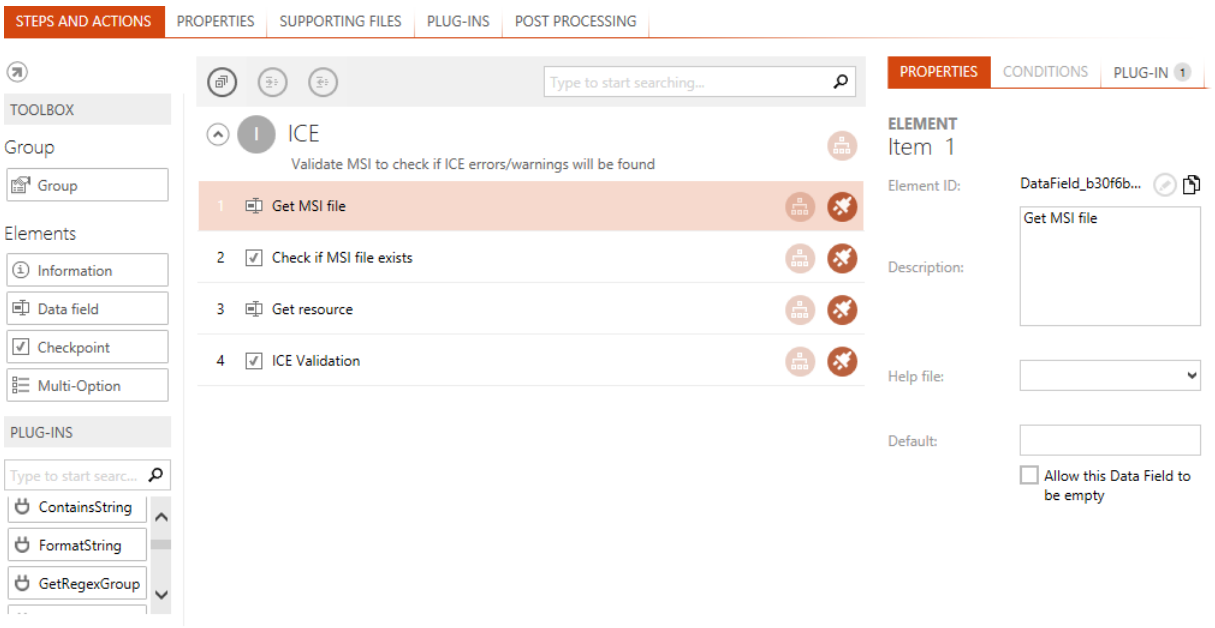
The Checklist Editor

The Checklist Editor is the RayQC interface for the manipulation of checklist structures and settings. To display a checklist template within the Editor environment:

- Use the open checklist tile from the **Dashboard** on the Home screen.
- Click on one of the template items from the **Recent** list on the Home screen.
- Use the Open view from the **FILE** menu and select the template file type.
- Hit **Control** + **Shift** + **O** to browse the Windows system for a template file.

Either way, the selected file (checklist or project) is opened within the **Checklist Viewer**. To switch to the Editor interface, use the **EDIT** button, available from the swipe bar at the bottom of the application window area.

The screenshot below shows one of the RayQC checklist template examples ready for manipulation within the Checklist Editor interface:



The application window is separated into different areas with specialized functionality and data as required for optimal user interaction support. Please refer to the *Working with RayQC* section for details regarding global interface objects and their usual naming in RayQC.

Main Toolbar

The menu bar has been extended with the actual view name of the editor on the left-hand side. It also provides direct access to the **FILE** menu, with options to trigger standard procedures such as saving the current checklist or creating a new one.

Toolbar

Collapse / Uncollapse All

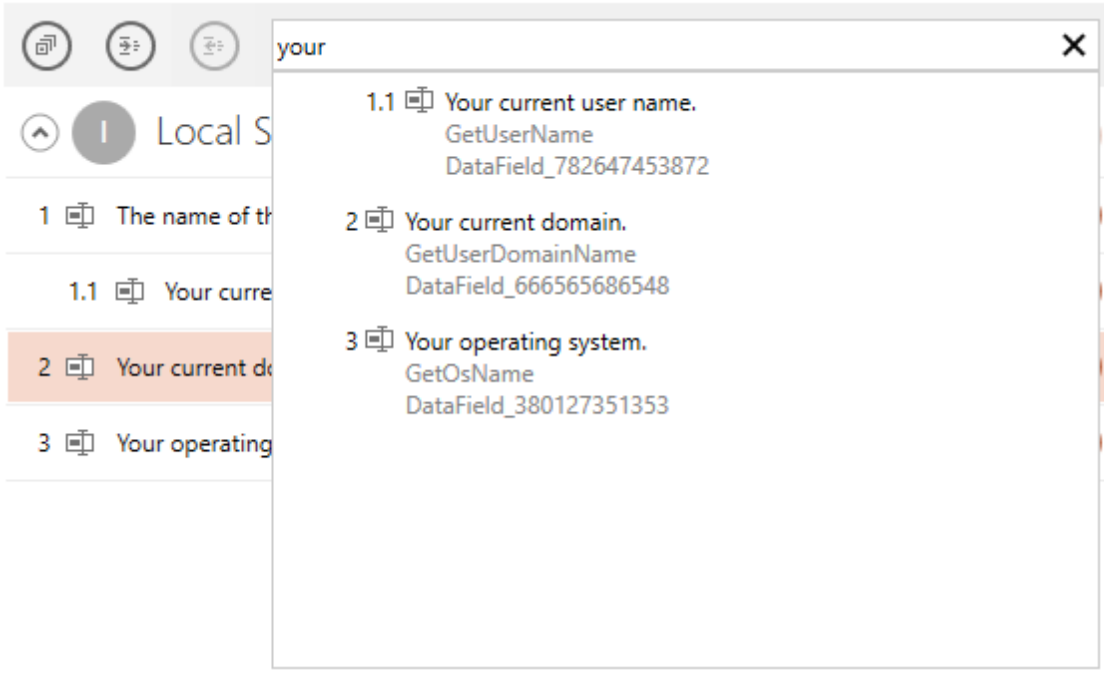
Toggle all checklist groups between collapsed / expanded state.

Increase / Decrease indent

Increases / decreases indentation level of the current selection. the buttons are grayed out if the action is invalid in the context of the currently selected item.

Searchbox

Start typing to perform a global search in the current checklist. When an item in the drop-down is pressed, the checklist editor jumps to that element.



Content area

Checklist title

The title of the currently opened checklist may be edited by clicking the edit button next to it. A direct value editor dialog is displayed, ready to type the new title. As an alternative, the title can also be modified from the Properties tab of the checklist editor interface.

Checklist canvas

All other properties of a checklist are available for manipulation via the tabbed views contained within the checklist canvas. Please refer to the *Checklist Structures* section for details regarding the different options that may be applied towards checklist design and functionality.

Swipe bar

View this checklist

Clicking on this button opens the Checklist Viewer, with the project representation of the currently visible checklist already loaded for testing and evaluation purposes. As a handy alternative, use the swift **Shift** + **Tab** shortcut to switch between the **Viewer** and **Editor** mode.

Show / Hide swipe bar labels

As already described within the *Working with RayQC* section, the swipe bar comes in two display modes:

Expanded and collapsed. The expanded mode displays a label for each button on the swipe bar, whilst the collapsed mode contains only the buttons without the labels. Hitting the button with the three vertical dots at the right-hand side of the swipe bar switches between the expanded and collapsed swipe bar display modes.

Shortcuts

RayQC offers a set of shortcuts for frequently used interface functionality. They are available from both core user interface areas: Checklist Viewer and Checklist Editor.

Key combination	Purpose
Control + N	Create new checklist template
Control + O	Open project
Control + Shift + O	Open template
Control + S	Save current project or template (overwriting existing version)
Control + Shift + S	Save current project as (optional definition of new format, location, and name)
Shift + Tab	Switch between Checklist Viewer and Checklist Editor interface
Control + right arrow key	Only from within the Checklist Editor interface: Increase the indentation of a checklist element
Control + left arrow key	Only from within the Checklist Editor interface: Decrease the indentation of a checklist element

Communication with RayFlow

As outlined before, RayQC is an application that can either operate in a standalone mode, or it can be integrated into the RaySuite. The core features of the RaySuite integration are direct communication channels between the central RayFlow database and RayQC:

Communication from RayFlow to RayQC

- Initiate RayQC checklist evaluation
- Initiate automated RayQC report generation

Communication from RayQC to RayFlow

- Authenticate to RayFlow
- Get RayFlow package/connection information using RayFlow internal plug-ins
- Update RayFlow package objects with values from a checklist project
- Export RayQC report files to RayFlow database objects

Note:
Since this document is a RayQC related user guide, the following sections describe the communication options from the RayQC point of view.

RayFlow is a workflow management system with a universal customization potential, which means that each phase, each object property, each tool configuration, and many other options are tailored towards the requirements of an enterprise packaging environment. Therefore, references from this general document to RayFlow objects, properties and settings will be given according to the standard configuration, whilst most RayFlow instances will have similar, but not equal interface and procedure designs.

Please contact your RaySuite administrator, or refer to the RayFlow or RaySuite documentation for further information regarding required RaySuite / RayFlow system configurations and customizations.

Additionally, please refer to the section about RayFlow settings provided within this document to gain knowledge about required RayFlow server connection credential handling.

From RayFlow to RayQC

The RayQC integration into RayFlow is solved as a tool utilization. Tools may be configured for any RayFlow process phase, and are used to trigger external functionality which will be applied on the package object from RayFlow. This RayQC tool integration is usually something that is defined for the QA activity within the packaging phase, or user acceptance checks within the UAT phase. RayFlow is absolutely free for additional tool integrations, but according to the Best Practice EALM workflow Raynet suggests that those two phases benefit most from a direct RayQC tool integration.

The upcoming procedure descriptions start from a RayFlow package data object. A user has to be logged on to a RayFlow client and access the required workflow phase in order to successfully trigger this kind of RayQC interaction. From the overview of packages that currently reside in the workflow phase, the user has to right-click the package within the RayFlow user interface in order to display the tools available for further processing of the package object.

Initiate RayQC checklist evaluation

From the list of available tools presented in the context menu, users have to pick "Evaluate with RayQC" to actually initialize a RayQC session with fitting RayFlow connection and package object reference parameter values.

Usually the tool trigger opens RayQC with a specific checklist template or project opened in the Checklist Viewer mode, ready for immediate evaluation activity. The checklist is not transferred from RayFlow to RayQC, but is known in RayFlow by its path. This path information is sent to RayQC as part of the command executed when the RayQC tool call is done.

Be aware:
The checklist and all related resources have to be available for the user who launches RayQC as a tool via RayFlow. The required access rights, and most likely network shares as well, have to be prepared in advance and aligned between all applications and user profiles that are active parts of the RaySuite system environment.

If there are checklist elements which have been configured to operate as RayFlow parameter containers, they are already filled with the properties of the related data object when the checklist is opened this way. The tool triggered RayQC session has automatically established a connection to the RayFlow database, in order to retrieve object properties if required. The usual identifier used for this data exchange is the unique package id property from RayFlow. It has to be part of the command executed when the RayQC tool call is done.

These two steps are the core of the evaluation initialization via RayFlow: Opening a RayQC instance with a data link to the RayFlow package object and a predefined checklist that is ready for evaluation.

Initiate automated RayQC report generation

This method takes the procedure described above to another level of convenience, efficiency, and result quality: When an automated report generation is triggered by a RayQC tool integration, the procedure goes beyond opening a RayQC instance with a data link to the RayFlow package object and a predefined checklist: it automatically runs all plug-in integrations which are configured within the checklist template, generates a report file and sends it back to the file repository of the package object within the RayFlow database.

However, there are some requirements regarding design limitations for automated checklists:

- RayQC can only launch plug-ins in this automated scope if they do not require user interaction.

  For example:
  It is not possible to execute the `FileOpenDialog` function from the internal `File` plug-in, because it needs manual response from a user: the selection of a file within the opened dialog window.
  RayQC is capable of recognizing these interactive functions for internal plug-ins. However, if a custom plug-in requires user interaction, RayQC does not have enough information to recognize this for auto-reports; and gives a warning when Run All is executed for a checklist with non-leading interactive plug-ins. Therefore, it is part of the checklist author's responsibility to prepare interaction free checklists for automated execution purposes.

- When the checklist has been processed, RayQC creates a report, no matter if the list is finished or not.
  There may be checklist elements which require manual input (user interaction). These element results will not be provided in the scope of an automated run. Therefore, a checklist that contains such elements can never terminate as **PASSED** or **FAILED**.

- During the data transfer phases between RayFlow and RayQC there has to be a stable connection to the

RayFlow webservice for data exchange. If this connection gets lost, the process cannot be completed automatically and has to be finished manually.

- Automated report generation procedures may only run on evaluation systems that are already prepared for the test cases executed by the specific checklist. Since RayFlow simply launches a local checklist and receives results as a report, there is no possibility to adjust local settings, copy resources, or the like. These tasks have to be prepared in advance.

RayQC creates a local copy of the checklist report that has been generated for auto-upload. Therefore, if the report file somehow gets damaged during the data transfer phase, the information it contained is not lost, but retrievable from the `%appdata%` folder of the current RayQC user on the local evaluation machine.

From RayFlow to RayQC and back

When a RayQC evaluation session has been initiated via RayFlow, it is possible to load RayFlow package object data into dedicated elements of a checklist (see Initiate RayQC checklist evaluation or Element Options > RayFlow parameter for details). This is the ingoing data transmission path. Raynet always strives to establish both, efficient and consistent workflows. Therefore, only receiving data from RayFlow is not enough. RayQC is also able to send data back to RayFlow package objects and update the current property values according to the values present within the checklist. If direct property updates are not desired, it may suffice to automatically add checklist evaluation reports to the media repository of the package object in RayFlow. Both options are available in RayQC as described in this section.

Update RayFlow package objects with values from a checklist project

Since all changes to RayFlow data objects are fully documented, workflow managers can freely make use of this option without running the risk of losing control or auditing acceptability due to untraceable phase transitions.

To actually use the Update RayFlow feature of RayQC, there are just a few preparations that need to be done:

1. The list of required parameter names for property interchange between RayFlow and RayQC needs to be communicated directly between the responsible persons for RayFlow maintenance and RayQC checklist creation.

2. RayQC checklists must be prepared to handle RayFlow data object properties.

   a. Within elements
   To do so, elements have to be extended with functionality provided by the internal RayFlow plug-in to get or set RayFlow parameters. As soon as a function of the RayFlow plug-in is executed, data exchange is triggered. Either receiving data from or sending data / files to the connected RayFlow server.

   b. As part of the Post Processing tasks of the checklist
   To do so, users have to define the RayFlow fields that will receive the update as well as the conditions that need to be fulfilled to trigger the post processing. Post processing itself needs to be activated for the Checklist as well.

3. The RayQC evaluation session has to be started by RayFlow, or a valid set of RayFlow connection credentials has to be provided, along with the reference id of the RayFlow package object that is the original data source from the RayFlow database. Actually, it is also possible to establish connectivity manually, by using the connection settings and local values for the package id, but the easiest and most seamless strategy is to trigger the direct communication from RayFlow.

Export RayQC report files to RayFlow database objects

The direct report export to RayFlow is available for all RayQC checklist evaluation procedures that have either been initiated directly from RayFlow or are operated on a system with an active RayFlow server connection. As soon as a checklist is opened this way in either the Checklist Viewer or the Checklist Editor, the Export to RayFlow option is visible and available from the **FILE** menu. During RayQC sessions without active RayFlow data link, the option is grayed out and cannot be used. To actually send a report from RayQC to RayFlow, users simply open the FILE menu, click on the RayFlow option from the menu on the left side, and select Upload Report.

Exporting report data does not change the properties of the RayFlow package data object, but simply adds a report file to the media repository of the package object.

RayQC offers three report target formats: PDF, HTML and DOCX. There are different methods to set the actively used format for report exports to RayFlow: either the tool integration command line call includes a value for the `-r` parameter regarding the report profile selection, or the default behavior settings in RayQC are used. If a specific format is given via the command line, it always precedes the general settings.

Reports may be exported at any time during the evaluation process, which may lead to files with information about **PASSED**, **FAILED**, or **NOT FINISHED** checklist evaluations within the RayFlow media repository of a package. The mere existence of a report file in RayFlow may therefore not be considered as an unambiguous indicator for a finished or successful evaluation phase.

The RayQC Command Line Interface

RayQC offers a standardized command line interface for parameterized application initialization. The following parameters may be used in conjunction with the RayQC executable:

Standard parameters

Parameter	Description	Example
`-checklist` `-c`	Load checklist from the given file name	`"C:\Program Files (x86)\RayQC\RayQC.exe" -c "C:\Users\Admin\AppData\Local\RayQC\Demo.rqct"`

`-element`
`-e`
	Pass values to checklist elements	Example for single element:
		`"C:\Program Files (x86)\RayQC\RayQC.exe" -c "C:\Users\Admin\Desktop\ICE validation.rqct" -e DataField368751317554=C:\MSI\orca.msi`
		
		Example for multiple elements:
		`"C:\Program Files (x86)\RayQC\RayQC.exe" -c "C:\Users\Admin\Desktop\Example.rqct" -e DataField1=Abc;DataField2=123`

`-help`
`-h`
`-?`
	Shows the command line interface help	`"C:\Program Files (x86)\RayQC\RayQC.exe" -h`

`-q`	Silent mode	The checklist is opened and executed in silent mode. Commonly used with switch `-s`, which defines where to save the results of silent operation. RayQC returns the following exit codes when run with this switch: `0` = checklist complete and passed `1` = checklist failed `2` = checklist incomplete `3` = other errors (read standard output for more information) This parameter is optional. Example: `"C:\Program Files\RayQC\RayQC.exe" C:\checklist.rqcp -q -s C:`
`-s`	Full file path of report (requires silent mode to be enabled)	Defines a full path where to save reports. This switch requires that silent mode is activates (`-q`). This parameter is optional.

RayFlow related parameters

Parameter	Description	Example
`-package` `-p`	The package ID from RayFlow	`"C:\Program Files (x86)\RayQC\RayQC.exe"` `-p "RSRFC123456789"`

`-project`
`-P`
	The project ID from RayFlow	`"C:\Program Files (x86)\RayQC\RayQC.exe"`
		`-P "RSRFP4711"`
		`-u "http://rayflow.Raynet.de"`
		`-l "DBowser@Raynet.de"`
		`-pw "Secret"`

`-url` `-u`	The URL for accessing RayFlow	
`-login` `-l`	The login name for the RayFlow connection	
`-password` `-pw`	The password for the RayFlow login	

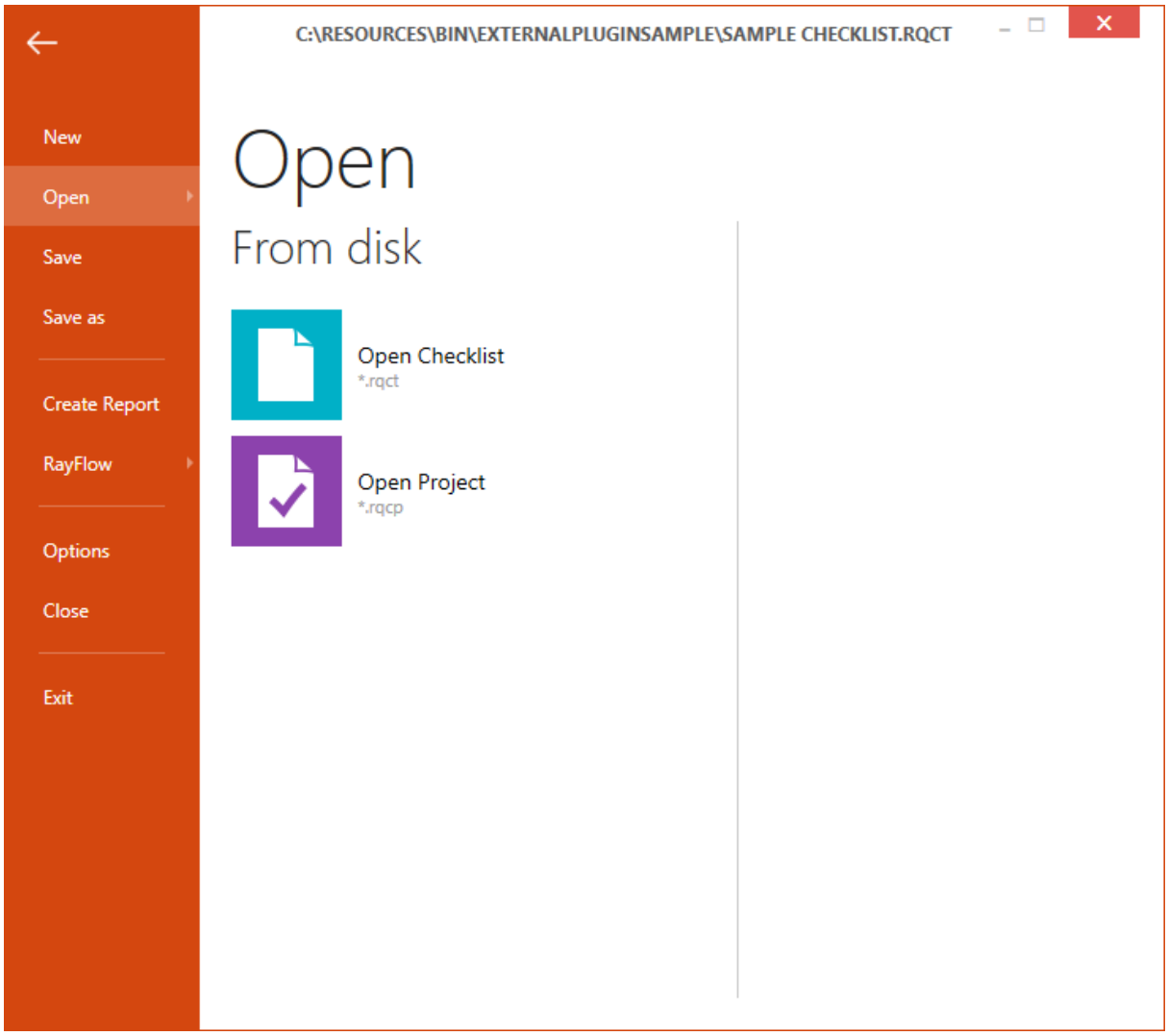
`-auto`
`-a`
	Automatically run checklist and upload report	`"C:\Program Files (x86)\RayQC\RayQC.exe"`
		`-a`
		`-r "Raynet standard PDF"`
		`-c "C:\Users\Admin\AppData\Local\RayQC\Demo.rqct"`

`-report` `-r`	Report profile name	
`-t`	Use currently logged on RayFlow user as transmission user	

Note:
The parameter `-e` (`-extension`) has been deprecated since in RayQC 2.1 and has been replaced by `-r` (`-report`). Additionally, since RayQC 4.0 `-e` parameter has been restored with a different functionality and it can be used now to pass the value to checklist elements.

The FILE menu

Clicking the **FILE** button contained within the Main Toolbar opens the file menu. This menu allows users to quickly access common functions.



New

Opens the dialog that allows creating a new checklist file. If a checklist or project is loaded and there are outstanding changes, the user will be asked to save them before continuing.

Open

Open an existing checklist or project. If a checklist or project is loaded and there are outstanding changes, the user will be asked to save them before continuing.

Save

Saves any outstanding changes in the currently open checklist or project. Please note that this button is only active when pending changes are detected.

Save as

Allows you to save the currently opened checklist or project under another name, location or file type.

Create Report

This option calls the create report dialog for the definition of target file name and type.

RayFlow

This functionality is only available when RayQC is embedded in the RaySuite framework; and therefore connected to a RayFlow server. Users may either upload a *report*, or trigger the update of data fields in the RayFlow project, which are defined as part of the *post processing* routine for a particular checklist. Users need to be logged in to the RayFlow server to have these options enabled.

Options

Opens the **Settings** view.

Close

Closes the current checklist or project. If any changes are pending, the user will be asked to save them before continuing.

Exit

Closes the current project and the whole RayQC application. If any changes are pending, the user will be asked to save them before continuing.

Settings

The Settings area of RayQC is accessible from 3 locations:
- Settings tile on the Dashboard
- Settings tab of the Main Toolbar, which is visible in all views of the application
- FILE menu - Options

It is recommended to check the settings at least once before the productive work with RayQC begins, since the configuration options offered there determine some properties of the user interface that may help you to gain quick orientation.

Available settings options

Within the settings area, there are seven configuration groups:

- **Interface**
  Options for user interface settings regarding the RayQC application
- **Behavior**
  Some automated product behavior can be defined here.
- **Signing**
  Definition of options for checklist signing
- **Plug-ins**
  Adding or removing global PowerShell and DLL plug-ins
- **Report profiles**
  Definition of options for report generation
- **RayFlow**
  Required information to connect RayQC with RayFlow.
  These settings are only required when your instance of RayQC is used in combination with RayFlow, Raynet's workflow management system for packaging processes. If you do not use RayFlow or any of the other solutions from the RaySuite product family yet, please visit the Raynet website, or contact your Raynet sales representative for further information on how the RaySuite can improve your packaging related business processes.
- **Virtual Machines**
  This is where the virtual machines that are to be used with RayQC can be defined.

Please refer to the specific sections of this document to get further details on the different settings configuration groups.

Settings storage locations

When making any changes in the settings area, the changes are saved permanently on a per-user basis after clicking on the **Accept** button in the swipe bar. As long as the changes are not accepted, they will only be valid for this RayQC session and go back to the default settings after restarting RayQC.

The configuration state is saved in the user profile of the currently logged on user: (`%appdata%\Raynet\RayQC\Config\`). RayQC creates several settings configuration files:

- `Config.xml`
  Contains the options defined within the **Behavior** tab of the RayQC settings area.

- `KeyPairs.xml`
  Contains the options defined within the **Signing** tab of the RayQC settings area.

- `Recent.xml`
  The XML file that contains the list of items which are shown within the Recent list on the RayQC Dashboard for a specific Windows User Profile.

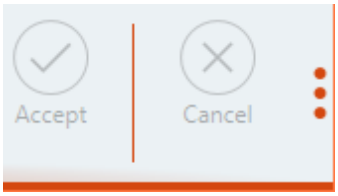
- `Report.xml`
  Contains the options defined within the **Report Profiles** tab of the RayQC settings area.

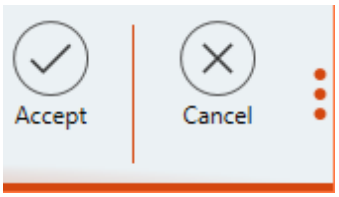
Be aware:
`RayFlow.xml` has been deprecated and is not used any more since RayQC 4.0.

If these files are not available when RayQC is launched, they are automatically restored with their default values as soon as the settings area is loaded within the application interface.

Saving and discarding changes to settings

The swipe bar of the settings area contains two buttons: Accept and Cancel. As long as a user has not made any changes compared to the currently saved settings, both buttons are grayed out and inactive. They become active and click-able with the first change of a single settings option, for example when a toggle slide of the behavior tab is switched.

	Inactive swipe bar state - settings have not been modified compared to the state that is saved within the configuration files. The swipe bar is collapsed. To expand it (and read the button labels), users have to click on the vertical bar of orange dots at the right-hand side.
	Active swipe bar state - settings have been modified compared to the state that is saved within the configuration files. The swipe bar is collapsed. To expand it (and read the button labels), users have to click on the vertical bar of orange dots at the right-hand side.
	Inactive swipe bar state - settings have not been modified compared to the state that is saved within the configuration files. The swipe bar is expanded. To collapse it (and hide the button labels), users have to click on the vertical bar of orange dots at the right-hand side.

	Active swipe bar state - settings have been modified compared to the state that is saved within the configuration files. The swipe bar is expanded. To collapse it (and hide the button labels), users have to click on the vertical bar of orange dots at the right-hand side.

All changes are temporary, and will not be stored within the settings files named above, until they are saved by clicking the accept button. Users may do that at any time and from any view of the settings area: RayQC always saves changes from all settings views at the same time.

This means that a user may switch a behavior setting, move to the connections tab, define RayFlow credentials there, switch to signing and report profiles without performing modifications there, and then decide to either accept or cancel all changes with one click on the respective button. It is not necessary to save changes before a tab is left.

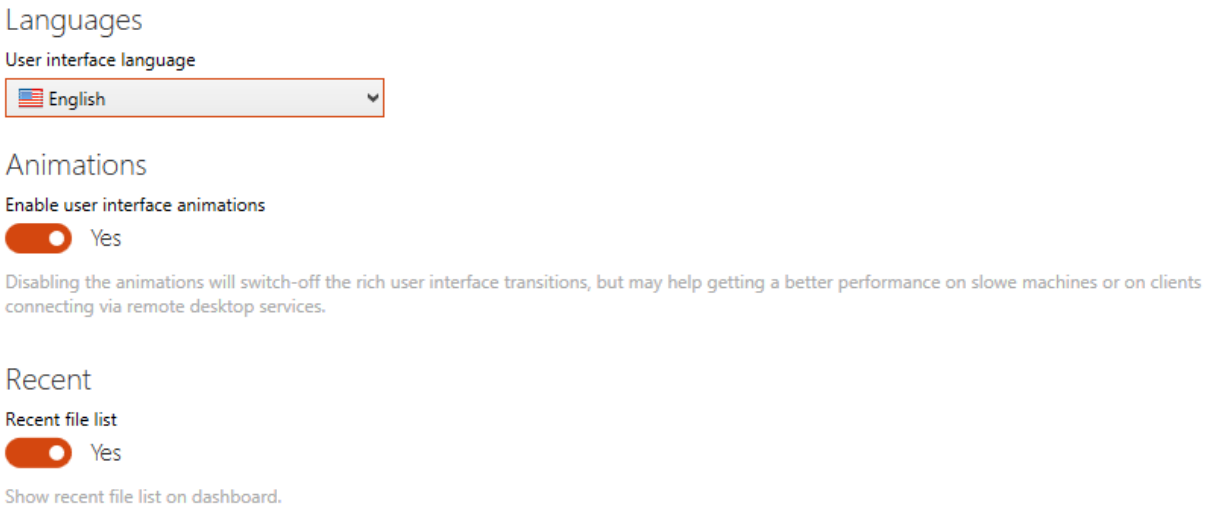
However, unaccepted changes take immediate effect on the currently running RayQC instance, but are lost as soon as the user closes the application. At the next application launch, the permanent settings are displayed and applied to the system operations again.

Interface

This section of the **SETTINGS** contains the options to define the language that is used throughout RayQC and the option to switch the animations on and off.

Languages

The languages option enables users to choose the language that should be used for the RayQC application. the language is chosen by selecting one of the options that can be found in the drop-down menu which is available under the User language interface option.



As shown in the screenshot there are currently two different languages available for the user interface of RayQC: The available languages are English and German. The active language can be chosen by selecting it in the drop-down menu and then saving the changes by clicking on the **Save changes** button which is located in the swipe-bar.

Note:
In order to apply the language settings, it is necessary to restart RayQC.

Animations

The default display mode of RayQC uses animations at several places, especially when users switch from one view to another. Whilst this is pretty nice eye-candy, it might be an effect that annoys power users. It may also be disturbing when RayQC is run via slow remote connections with low bandwidth and delayed reaction time. Disabling animations may save precious time in these environments. Set the slide toggle to **NO** to disable animations, or to **YES** to keep the smooth transitions in place.

Recent file list

This option is set to **YES** by default, which means that the list of recently opened projects and checklists is shown on the RayQC Dashboard. Setting this switch to "NO" hides the recent list. Since the list is recorded on a per user basis and accelerates access to recently used RayQC files, it is recommended to keep this configuration option active.

Note:
Switching the **Recent** file list option does not stop RayQC from recording which files have been accessed, but simply hides the list from the **Dashboard**. Therefore, it is possible to switch the option from inactive to active at any time – the real list of recently accessed files is always available.

Behavior

Within the behavior view of the RayQC settings section, users can manipulate the following configuration options:

Write protect plug-in results

Set this slide toggle to **YES** to prevent manual overriding of element values which were automatically calculated as plug-in results. The default setting is **NO**, which means that users may change the result that has been delivered by plug-ins. There is no strict recommendation to lock plug-in results or not - it is up to the decision of the quality control manager to allow manual adjustments or not.

Sign checklists on saving

The signing options defined within the signing tab o the settings view do not automatically trigger checklist signing..They just prepare RayQC to be able to sign checklists in general. Setting the slide toggle to **YES** enables checklist signing, whilst setting it to **NO** disables it. The combination of general settings and separate option switch makes it fairly easy to keep permanent signing options but at the same time be able to make actual use of them flexibly as required.

The slide toggle is not accessible for modifications as long as there is no keypair profile defined in the signing tab. Please define a key pair profile by using the interface in the signing tab to enable the toggle control.

Reject checklists without signature

One aspect of system security is checklist signing. Since checklists may run scripts and execute applications on the local device, it is a matter of enterprise security to make sure that no malware is brought into the system, and that no unauthorized modifications are executed. By activating this option, any checklist that is not signed will not be opened by RayQC - or executed for that matter. Allowing only signed checklists is the safer mode of operation, since it allows to knowing about the origin of checklists at any time.

Reject untrusted certificates

If an environment is set up to reject unsigned checklists, the idea to make sure signing has been performed by the application of a trusted certificate is just one tiny step away.

Note:
Activating this option does not automatically reject checklists that are not signed at all. It simply rejects those checklists, that are signed, with untrusted certificates. To make sure only checklists signed by trusted certificates are loadable, both options **Reject checklists without signature** and **Reject untrusted certificates** have to be enabled!

Signing

This signing view is only relevant for those RayQC instances, which are intended to create and edit signed checklist templates. Signing settings are not required for mere evaluation of signed checklist.

Managing signing options contains the following procedures:

- Select the active key pair profile
- Create new profiles
- Test key validity
- Edit profiles
- Remove profiles

Read on to get further details on how to accomplish these management tasks.

Key Pair Profile properties

In order to establish a working signing configuration, the following property settings are required:

Description

Short textual information about the profile for later recognition.

Certificate file

The `*.cert` file has to be copied to the %appdata%\Raynet\RayQC`\config\Security` directory. This is done automatically when a `*.cert` file is selected via the dialog triggered by the browse button at the right-hand side of the certificate ield. Also the certificate properties from the file are retrieved and displayed in the “Set up certificate” field.

Private key selection

The `*.pem` file has to be copied to the `%appdata%\Raynet\RayQC\config\Security` directory. This is done automatically when the `*.pem` file that matches the certificate is selected via the dialog triggered by the browse button at the right-hand side of the “Set up private key” status indicator.

Once the key is available in RayQC, its validity can be tested as described below.

Certificate chain valid

This indicator provides information about the trust status of the certificate used within the current keypair profile. If the machine that runs the RayQC instance trusts the certificate, the checkbox is active. If the certificate is untrusted, it is disabled. The validity of the certificate is important for those setting configurations that cause RayQC to reject checklists, that are signed but by untrusted certificates (compare with the section about behavior settings).

Note:
The "Certificate chain valid" state indicator relies on Windows certificate store status information that is already available at application launch. Therefore, it is recommended to establish the trust, if required at all, before RayQC is started and the actual key pair settings are defined.

Tip:
To establish certificate trust, the root certificate of the one used within the current key pair profile has to be installed as trusted certificate. The installation may be done on a per-user or per-machine basis. It is recommended to prepare a machine based installation, to allow all users of a QC device to operate on the same certificate preparation level. The procedures required for certificate installation is Windows based, and therefore not described in detail within this documentation. Please refer to MSDN or other Microsoft resources to get details regarding this procedure.

Define the active Key Pair profile

1. Go to the **settings** area and click on the **signing** tab.
2. Expand the options list provided within the drop down menu named **Active KeyPair Profile**.
3. **Select** the desired **profile**.

Note:
The profile selection is not saved permanently yet - closing RayQC at this point is equal to discarding the changes to the signing settings profile stock. To actually save the profile selection beyond the next application closure, the **Accept** button on the swipe bar has to be used.

To create a new Key Pair Profile

1. Go to the **settings** area and click on the **signing** tab.
2. Click on the button **Create** at the right-hand side of the Active KeyPair Profile selector control.
3. Enter the **name** of the new profile as demanded by the displayed dialog.

   The profile name has to be unique among the key pair profiles stored for the currently logged in Windows User on the machine that is running RayQC.
4. Click **OK** to save the new profile
5. A new option is automatically added to the Active KeyPair Profile selector control.
6. The new profile needs to be **edited** in order to set the parameters required to actually sign checklists with it.

Note:
The new profile is not saved permanently yet - closing RayQC at this point is equal to discarding the changes to the signing settings profile stock. To save the profile for later re-use, the **Accept** button on the swipe bar has to be used.

To test a key

1. Go to the **settings** area and click on the **signing** tab.
2. Expand the options list provided within the drop down menu named **Active KeyPair Profile**.
3. **Select** the desired **profile**.
4. Click on the **Test** button at the right-hand side of the "Set up private key" controls.
   If the button is disabled and cannot be clicked, the private key may not have been uploaded yet. If so, please use the browse button to transfer the key to the RayQC settings storage location.
   The button should automatically become active.
5. A dialog is displayed, requesting the password required to encrypt the private key for signing.
   Please, enter the password and click **OK** to start the key validation.
6. The test result is shown within an information dialog at the end of the test routine.

Note:
There are several reasons why a key test might fail. Please follow the details of the failure message to adjust the right parameters for achieving successful validity.

To edit a Key Pair Profile

1. Go to the **settings** area and click on the **signing** tab.
2. Expand the options list provided within the drop down menu named **Active KeyPair Profile**.
3. **Select** the desired **profile**.
4. The **properties of the profile are loaded** into the input fields below the Active KeyPair Profile selector.
5. Adjust them according to your needs. They are immediately effective and applied to the running RayQC instance.

Note:
The new profile properties are not saved permanently yet - closing RayQC at this point is equal to discarding the changes. To save the profile for later re-use, the **Accept** button on the swipe bar has to be used.

To remove a Key Pair Profile

1. Call the **settings** area and open the **signing** view.
2. Expand the options list provided within the drop down menu named **Active KeyPair Profile**.
3. **Select** the desired **profile**.
4. Click on the **Remove** button at the right-hand side of the Active KeyPair Profile control.
5. The profile is removed from the options list.

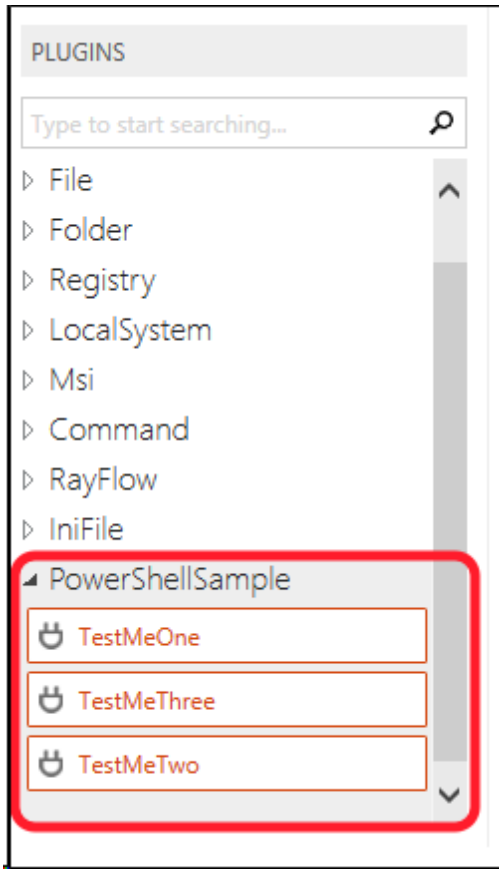
Note:
The profile is not deleted permanently yet - closing RayQC at this point is equal to discarding the changes. To permanently remove the profile, the **Accept** button on the swipe bar has to be used.

Also be aware:
Once a profile is removed, it is not possible to modify checklist templates that have been signed by this certificate. Therefore, please be sure to have a copy of the certificate and key files in case of needing them again at a later time.

Plug-Ins

External Plug-Ins

External plug-ins add new functionality to the checklist processing by adding PowerShell based or DLL based plug-ins to RayQC. After loading a checklist and opening the editor the functions of external plug-ins may be found in list of plug-ins in the toolbox on the left side. Functions based on external plug-ins can be identified by the orange font color.



Be Aware:
To load a plug-in RayQC must be licensed to use it. The Standard Edition of RayQC is not able to process PowerShell based plug-ins. The Enterprise Edition is needed to run these plug-ins.

For DLL based plug-ins, a valid plug-in license is necessary. This can be obtained from Raynet.

Handling External Plug-Ins

RayQC has 2 ways of handling external plug-ins:

1. Local plug-ins: these plug-ins are stored inside the checklist file and will stick with it unless removed from the checklist. A checklist using local plug-ins can be processed on any RayQC instance that loads the checklist file

without any additional actions.

2. Global plug-ins: global plug-ins are stored in the subfolder “plug-ins” in the application directory. Creating a checklist using a global plug-in will not save the plug-in inside the checklist file. Because of this, other instances of RayQC need to have the same set of global plug-ins in order to load and process such checklists.

When to Use Which: Benefits / Drawbacks

Local plug-ins: the plug-ins stick to the checklist and it may be run on any RayQC instance with a proper licensing.

Global plug-ins: using global plug-ins for checklist design unifies the set of used external plug-ins. If changes or fixes are made to a global PowerShell plug-in, each checklist will use the new version automatically. If local plug-ins are used, each checklist must be updated separately. On the other hand all plug-ins must be installed again if the checklist is to be processed on another installation of RayQC that does not have these plug-ins already installed.

The Plug-In Manager Dialog

Both local and global plug-ins utilize the same dialog for plug-in management. The plug-in manager for global plug-ins can be found in the settings screen while the manager for the local plug-ins is located in the checklist editor.

Both dialogs have the same look and functionality. To avoid accidentally editing global plug-ins while editing a checklist, the plug-in manager for global plug-ins is deactivated if a checklist is open.

Be Aware:
When using the Standard Edition of RayQC, the plug-in manager allows to add / remove PowerShell plug-ins, even though the Standard Edition of RayQC cannot load such plug-ins and they won’t be displayed in the toolbox. The same is true for non-licensed DLL based plug-ins.

Add a new PowerShell plug-in

1. Press the button labelled **Add PowerShell plug-in**.

2. In the following dialog select the directory that contains the plug-in files (`manifest.xml` and all script files that are used by the plug-in).

Remove a PowerShell plug-in

1. Select the plug-in to remove from the list.

2. Click **Remove selected**.

OR right click the plug-in to remove and select delete from the context menu.

Be Aware:
Local plug-ins that are used by a checklist that is currently loaded cannot be removed. Remove all usages of the plug-in to allow the deletion of the plug-in.

Add a new PowerShell plug-in

1. Press the button labelled **Add DLL plug-in**.

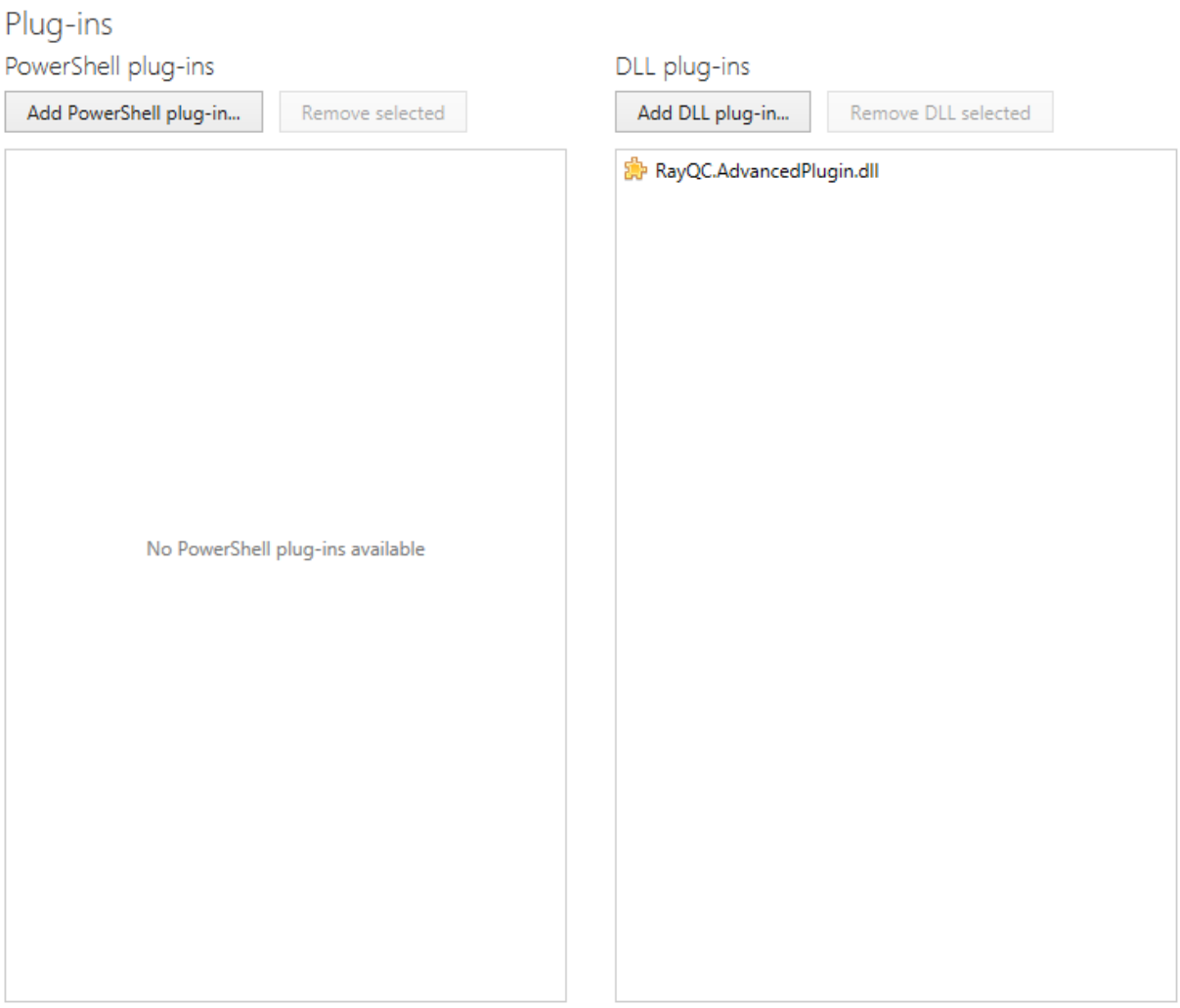
2. In the following file dialog select the plug-in DLL.

Remove a DLL plug-in

1. Select the plug-in to remove from the list.

2. Click **Remove DLL selected**.

OR right click the plug-in to remove and select delete from the context menu.



The Plug-In Manager Dialog

Report profiles

Report profiles are required to configure the format definitions for exports of project evaluations.

Managing report profiles contains the following procedures:

- Select the active connection profile
- Create new profiles
- Edit profiles
- Remove profiles

Read on to get further details on how to accomplish these management tasks.

Report Profile properties

In order to prepare decent report format profiles, the following configuration settings are required:

Profile Description

Short textual information about the profile for later recognition.

Exporting Format

There are 3 predefined export formats users may select from: **PDF**, **DOCX**, and **HTML**. Each of them has certain advantages and disadvantages, therefore it is important to know what purpose a report file has in order to be able to decide about the fitting export format. If, for example, a report goes out to inform a product owner, QA manager or customer about the outcome of specific QA tasks, PDF is most likely the right choice. However, if the reports are intended to be edited or extended by a later process step of the whole workflow RayQC is part of, DOCX is easier to handle. Selecting HTML makes sense when structured information is required, that might even be evaluated and analyzed by automated procedures.

However, the export format of a report profile is not carved in stone at profile creation, but may be adjusted whenever required. It is just a matter of convenience and efficiency to define the optimal choice right from the start.

Page Size

The preferred page size for reports should be strongly connected to the default document guidelines applied to standard documents of an enterprise. RayQC offers users to select from a predefined set of internationally used page dimension standards:

- ISO / DIN A4 (210 mm x 297 mm)
- ISO / DIN A5 (148 mm x 210 mm)
- ISO / DIN B4 (250 mm x 353 mm)
- ISO / DIN B5 (176 mm x 250 mm)
- ANSI / ASME Legal (8.5 in x 14 in)
- ANSI / ASME Letter (8.5 in x 11 in)

Font Size

Instead of requesting detailed font size definitions for each headline type, paragraph, foot note, etc., RayQC simply offers 3 size settings (small, medium, large) to the user and defines dependencies according to the base font size setting on its own.

Logo Path

It is possible to place a product or company logo within the exported report file. The logo file may be stored at any network or local location, as long as it is accessible for RayQC at the time of report generation. It is recommended to use common graphic formats, such as PNG or JPG, since these are displayed without any issues on nearly every operating system. If, for example, the report is generated as HTML file, a copy of the original logo file is added to the report files folder. Once the report is copied to another device, it may not be possible to display exotic graphic types there.

Element Filter

With a click on the "Element Filter" button below the standard property set of a report profile, users open a new dialog which contains options regarding the visibility of specific element types within the generated report files:

Element type: Information
Activate the checkbox to display the content of this element type within generated report files. Since Information elements do not contain test result information, but usually inform the tester about the checklist context, it may not be required to include them in reports. However, it is recommended to keep them within the reports, in order to make sure that the test routines are fully comprehensible for the reader of the report.

Element type: Checkpoint
Use the checkboxes to determine which elements will be included in reports: Passed, failed, or void elements. If the report purpose is to outline issues that were revealed by a checklist run, it may be sufficient to include failed elements. However, a full report of the checklist routines should contain all elements, and therefore activate all presented checkboxes.

Element type: Multi-Option
The checkboxes allow to include all elements that have a selected result (=valid), and / or all those that have not been answered (=void). Once again, the right constellation depends on the report purpose: A full report should contain all elements, whilst a short issue report should be reduced to problematic or missing results.

Element type: Data Field
Activating the checkboxes in this area determines whether or not empty and / or filled elements should be part of report files. If none is selected, this element type is not part of the generated reports.

As soon as all desired changes are executed, the dialog may be closed with a click on the OK button at the bottom. The changes are immediately taken into account for exports executed during the current working session of RayQC.

Signature Fields

With a click on the Signature Fields button below the standard property set of a report profile, users open a new dialog which contains 2 groups of properties for signature options:

- Signatures on the summary page
- Signatures on the protocol pages

It is recommended to enable signature fields on the summary page, whilst having them on each of the potentially numerous details pages might be a bit of an overkill for average acknowledgment needs.

For each group there are these settings to define:

- Show signature fields
  Activate the checkbox to display signature fields. If the checkbox is inactive, the other settings for this group do not take effect on the actual report generation process. However, they may be defined as preparation for later use.

- Number of signature fields
  RayQC allows to define signature sets of 1, 2 or 3 signature demands. The actual requirement depends on the QA process applied to the specific project or enterprise RayQC is embedded with.

- Label of Signee [1-3]
  Make sure to match the number of signature fields to the labels defined for them: If 3 signature fields are displayed but only one has a label definition, the other 3 will most likely not be overly useful, since the role or person whose acknowledgment is demanded is unknown to the recipient of the report.

As soon as all desired changes are executed, the dialog may be closed with a click on the OK button at the bottom. The changes are immediately taken into account for exports executed during the current working session of RayQC.

Page Filter

With a click on the Page Filter button below the standard property set of a reports profile, users open a new dialog which contains three options users may activate independently:

Summary Page

Activate the checkbox to add a cover page to the report. The cover contains charts as visual result summary, as well as the checklist name and result, the date of the report creation, and a copy of the current task box status of the checklist. It is recommended to include the cover, as it provides swift access to the key figures of the checklist evaluation.

Protocol Page(s)

The protocol pages contain the checklist elements, as far as configured within the Element Filter dialog. If no elements have been activated there, an empty protocol page will be the result.

Separating Blank Page

If the blank page is part of the report, it is added directly after the cover page and right before the protocol page(s). It is not possible to add a blank page to reports that do not contain both cover and protocol pages. The purpose of the blank page is to get the report details on the right side for printing it in duplex mode.

As soon as all desired changes are executed, the dialog may be closed with a click on the OK button at the bottom. The changes are immediately taken into account for exports executed during the current working session of RayQC.

Define the active Report Profile

1. Go to the **settings** area and click on the "**report profiles"** tab.
2. Expand the options list provided within the drop down menu named **Active Report Profile**.
3. **Select** the desired **profile**.

Note:
The profile selection is not saved permanently yet - closing RayQC at this point is equal to discarding the changes to the report profile stock. To actually save the profile selection beyond the next application closure, the **Accept** button on the swipe bar has to be used.

To create a new Report Profile

1. Go to the **settings** area and click on the "**report profiles"** tab.
2. Click on the button **Create** at the right-hand side of the Active Report Profile selector control.
3. Enter the **name** of the new profile as demanded by the displayed dialog.

The profile name has to be unique among the report profiles stored for the currently logged in Windows User on the machine that is running RayQC.

4. Click **OK** to save the new profile
5. A new option is automatically added to the Active Report Profile selector control.
6. The new profile needs to be **edited** in order to set the parameters required to actually provide a custom report design.

Note:
The new profile is not saved permanently yet - closing RayQC at this point is equal to discarding the changes to the report profile stock. To save the profile for later re-use, the **Accept** button on the swipe bar has to be used.

To edit a Report Profile

1. Go to the **settings** area and click on the "**report profiles"** tab.
2. Expand the options list provided within the drop down menu named **Active Report Profile**.
3. **Select** the desired **profile**.

4. The **properties of the profile are loaded** into the input fields below the Active Report Profile selector.

5. Adjust them according to your needs. They are immediately effective and applied to the running RayQC instance.

Note:
The new report profile properties are not saved permanently yet - closing RayQC at this point is equal to discarding the changes. To save the profile for later re-use, the **Accept** button on the swipe bar has to be used.

To remove a Report Profile

1. Go to the **settings** area and click on the "**report profiles"** tab.

2. Expand the options list provided within the drop down menu named **Active Report Profile**.

3. **Select** the desired **profile**.

4. Click on the **Remove** button at the right-hand side of the Active Report Profile control.

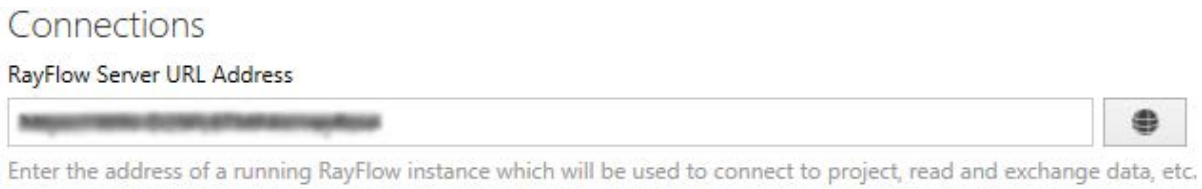
5. The report profile is removed from the options list.

Note:
The profile is not deleted permanently yet - closing RayQC at this point is equal to discarding the changes. To permanently remove the report profile, the **Accept** button on the swipe bar has to be used.

Also be aware:
At least one profile needs to be present within the selector, therefore the last report profile cannot be removed.

RayFlow

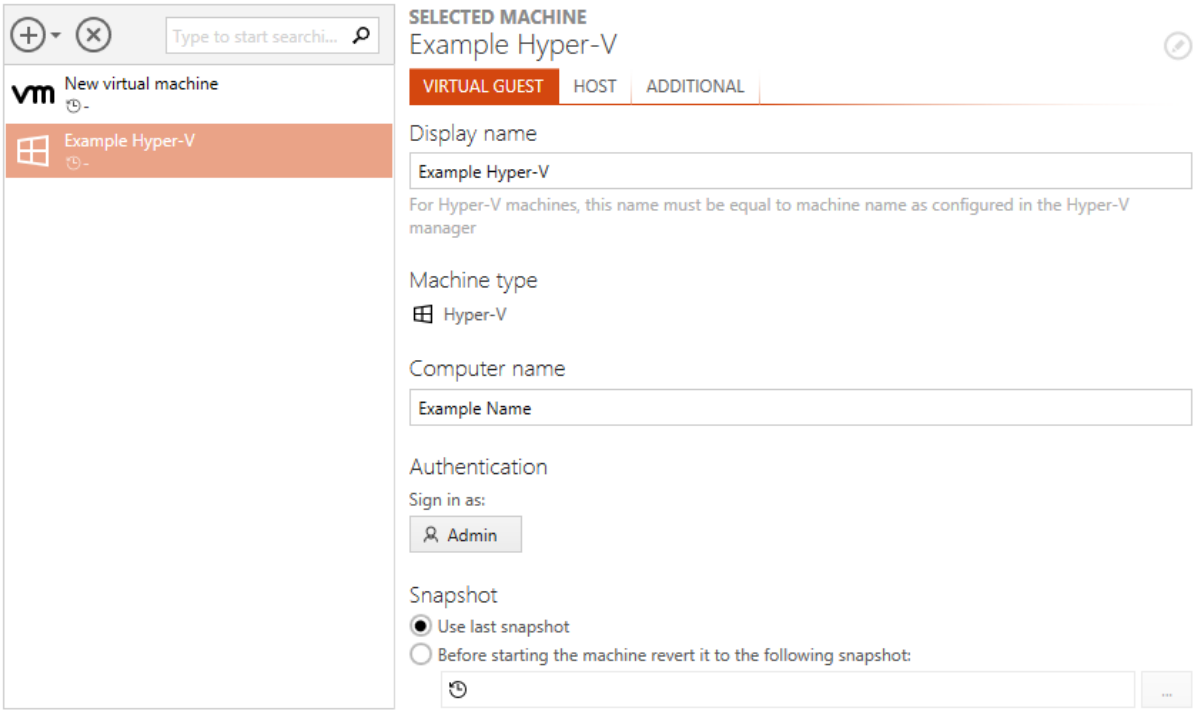
This settings view is only relevant for those RayQC instances, which are about to be connected to a RayFlow server. This is where the settings for the connection with the RayFlow server can be configured.



The URL address of the RayFlow Server is shown here. Click on the button to open the link to the server in a web browser.

Virtual Machines

The settings for virtual machines can be configured in this tab of the **Settings** section.



By default, the list of machines is empty. Each machine that needs to be used, has to be imported first. The view is divided into three sections:

- **Function Buttons (Add, Remove, Search)**
  This panel is used to add a new machine, remove a selection, and search for machines in the list. In order to filter the list, type a few letters into the search field and the list will be filtered automatically. To clear the results either click **X** or clear the content of the search box.

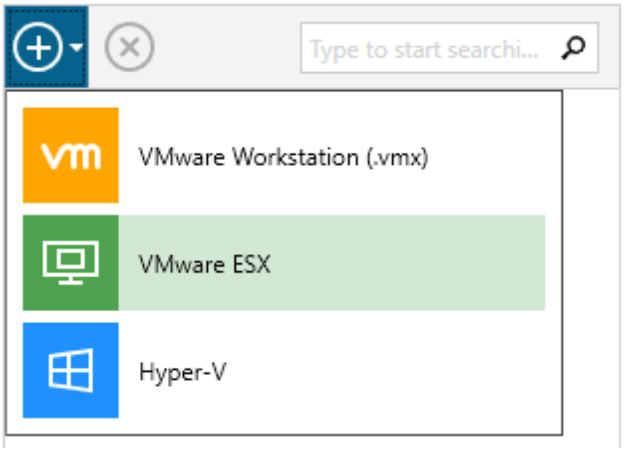
- **List of Machines**
  This is a list of all the machines defined for RayPack Studio products. Each machine is represented by an icon representing its type (**VMware Workstation**, **VMWare ESX**, or **Hyper-V**), machine name, and (if configured) the name of the snapshot that is to be used. Select any machine from the list to see its details shown on the right side.

- **Virtual Machine Editor**
  The right panel contains various controls and inputs about the currently selected machine. Later in this chapter their meaning is discussed together with slight differences between different machine types.

To Import a Virtual Machine...

Warning:
It is not possible to import virtual machines if RayQC is already installed on a virtual machine!

1. Press the **+** button to expand the new machine drop-down menu.



2. Select type of machine to import.
3. The machine will be added to the list. Use the editor to configure the details for the machine.

Be aware:
Due to technical limitations of the APIs of VMware it is not possible to define both Workstation and ESX machines. Once one of these has been imported, the other type will be disabled!

To Delete a Virtual Machine...

1. Select a machine from the list.
2. Press the **X** button in the functional panel.

Editing Virtual Machines

The virtual machine editor is divided into three tabs. The visibility of tabs depends on the type of the currently selected machine.

Virtual Guest Tab

This tab contains common properties which are valid for all types of supported virtual machines.

- **Display Name**
  This is the name under which the machine is shown in the list. For VMware Workstation and ESX the name can be a user-friendly string which helps to determine the purpose of the machine (for example *My Repackaging Machine* etc.). For Hyper-V machines, this value must be equal to the name that is visible in the Hyper-V manager.

- **Machine Type**
  The type of the current selection. This setting is read-only.

- **Path to VMX File** (only Workstation and ESX)
  This is the full path to the `.vmx` container file. For Workstation machines, this must be a full absolute path to a file, for example `C:\Virtual\Machine\Machine.vmx`. For ESX, the name must include a datastore token (a name surrounded by square brackets) followed by the relative path of the `.vmx` file. The path can be viewed

in the properties dialog directly in a vSphere client. A sample value would be `[DATASTORE]Machine\Machine.vmx`.

Be aware:
Failure to provide valid file paths to the VMX makes the machine unable to work with. Machines with invalid paths or unsupported file formats will be marked as invalid and the user will not be able to use them.

- **Computer Name** (Hyper-V only)
  This should be the full DNS computer name of a virtual machine.

- **Authentication**
  This setting configures credentials used to connect to a virtual machine. This field is always required even if auto-logon is enabled on the guest Operating System. In order to configure the credentials, press the button **Configure credentials...** and enter the user name and the password. In order to log in as a domain user, use `DOMAIN\USER` syntax.

Be aware:
Passwords are stored in an encrypted form in a `.txt` file. This however offers by no means a state-of-the-art security. Expect that these values can be decrypted easily and as such never store confidential data on machines with shared access to RayQC configuration files.

- **Snapshot**
  Defines which snapshot is to be used. While it is generally possible to always use the last snapshot (the current one) in a production environment the name of a snapshot should always be specified either by typing the name manually in the textbox below or by pressing the **...** button and using the selector dialog.

Host Tab (Only VMware ESX and Hyper-V)

This tab contains the properties of the hypervisor server. Because VMware Workstation uses solely local files, the tab is not visible when working with Workstation machines. When this tab is shown, all values presented here are mandatory.

- **Host Address + Port**
  Defines the location and the port of the host. Contact the responsible administrator to find out these values. By default port `5985` is used for Hyper-V and `443` for ESX machines, but they can be overridden individually.

- **Authentication**
  This setting configures the credentials used to connect to the host. This field is mandatory even if auto-logon is enabled on the guest Operating System. In order to configure the credentials, press the button **Configure credentials...** and enter the user name and the password. In order to log in as a domain user, use the `DOMAIN\USER` syntax.

Note:
The credentials for the host are usually not the same as the credentials that are used to connect to the VM.

Be aware:
Passwords are stored in an encrypted form in a `.txt` file. This however offers by no means a state-of-the-art security. Expect that these values can be decrypted easily, and as such never store confidential data on machines with shared access to RayQC configuration files.

Additional Tab

- **Comments**
  This is an additional invisible field that can be used for notes and remarks about the machine.

.

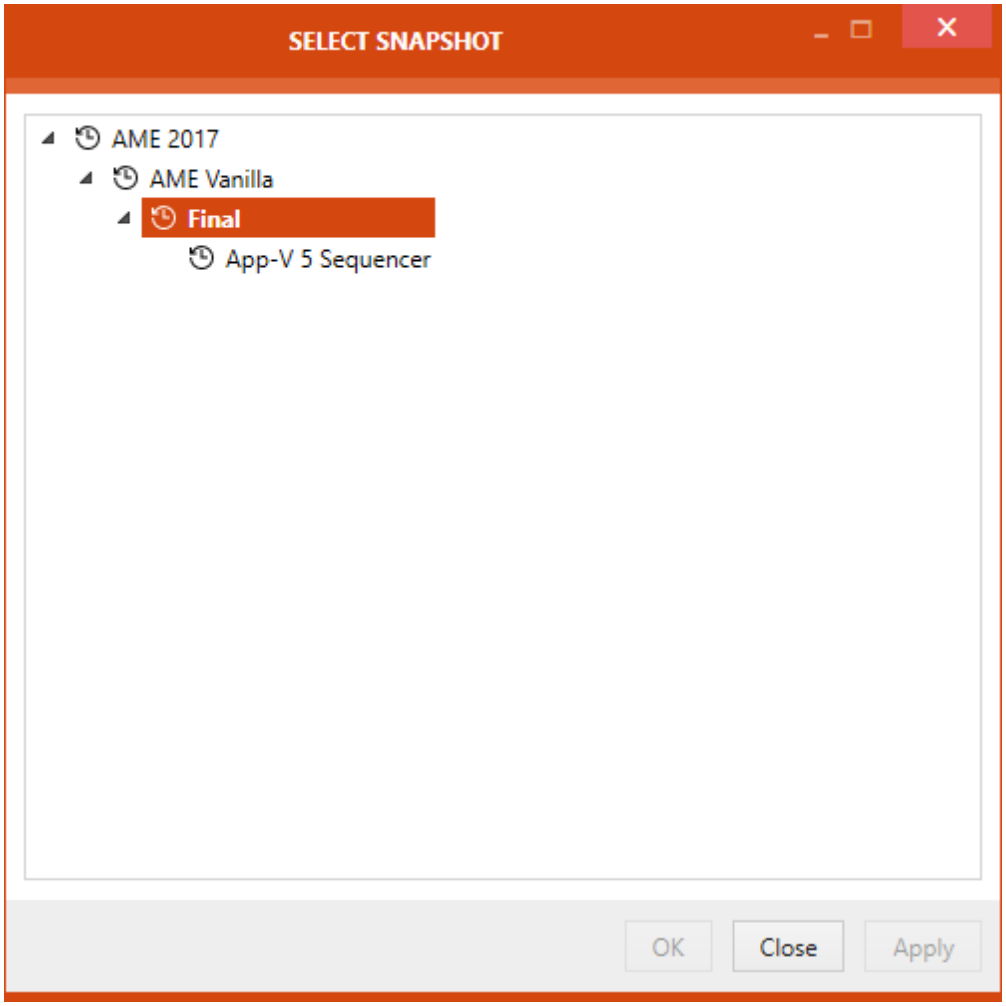
- **Copy files using shared folder**
  When this option is active, shared folders are used to transfer files to the virtual machine. This functionality may be limited in some environments. When unchecking this option a legacy file-based transfer will be used which may be slightly slower than the shared folder approach.

- **Execute tasks on a virtual machine as Administrator**
  By default, the tasks are executed as Administrator to ensure that the setup files changing machine files and registries are allowed to do it. For certain environments and operations this behavior may not be desired. Unchecking this checkbox makes Raynet start the tasks as the invoker.

Be aware:
This option is only recommended for troubleshooting or for tasks which do not require administrative privileges. Certain Raynet functions / plugins may not work when the task is not running as administrator.

Snapshot Selector

To start the **Snapshot Selector** first select the **Before starting the machine revert it to the following snapshot** option that is available beneath the Snapshot heading in the **Virtual Machines** tab in the **Settings**. After the option has been selected, click on the **Browse** [**...**] button to open the **Snapshot Selector**.



In the **Snapshot Selector** the available snapshots for the virtual machine are shown in a tree structure. Either select a snapshot or close the **Snapshot Selector** without changing the current selection. The following options are available in the **Snapshot Selector**:

- **OK**: This option is used to close the **Snapshot Selector** saving the currently selected status.
- **Close**: This option is used to close the **Snapshot Selector** without saving the changes.
- **Apply**: This option is used to save the currently selected status without closing the **Snapshot Selector**.

Preparing Virtual Machines

Depending on the type of the virtual machine, some extra steps may be required for it to work with RayQC. The following guide outlines the required steps.

General (All Types)

- The guest machine must be able to see the host machine by its DNS name. For example, if the host machine name is MWS0189, then the guest must be able to resolve its name to an IP address.
- On the host machine, incoming traffic on a specific port must be enabled. The same remote port must be allowed by the firewall configuration on the guest machine. The port range used by default is `48654-48999`, starting from the lowest available. The port range can be configured by changing configuration files.

- It is highly recommended to disable the User Access Control (UAC) on the guest machine. This is especially important for silent operations, so that the setups can be started without user interaction.
- It is highly recommended to enable auto-logon to the machine for the configured user. This ensures that the machine can be automatically powered on and started without waiting for the user to log-in interactively. The following link describes how to configure it: https://support.microsoft.com/en-us/help/324737/how-to-turn-on-automatic-logon-in-windows.
- Even if auto-logon is enabled on the guest machine, the matching credentials must be provided in the RayQC configuration. These are used to execute the programs and commands on the VM in the right context (so that the user is able to see dialogs etc.), but due to technical limitations on virtualization solutions they cannot be used to bypass the initial login / lock screen.
- Although possible, we do not recommend using machines without snapshots. It is recommended to have at least snapshot on the VM, and to have it selected in the VM configuration.

VMware Workstation

- VMWare Tools must be installed on the guest machine.
- VIX API must be installed on the host machine (this is already done if VMware Workstation is installed on your host machine).
  - There is a known issue in VMware Workstation 14 and later, where the necessary COM interfaces are not registered by default. You can fix this problem by applying the steps, described in the following Knowledge Base Article: https://raynetgmbh.zendesk.com/hc/en-us/articles/360000277786-RSC200351-Executing-Virtual-Machine-Operations-on-VMware-Workstation-14.

VMware vSphere / ESXi

- VMWare Tools must be installed on the guest machine.
- PowerCLI must be installed on the host machine in a version that is compatible with the vSphere / ESXi version. See https://code.vmware.com/web/dp/tool/vmware-powercli/11.0.0 for more information.
- PowerShell 3.0 or higher must be installed on the host.

Hyper-V

Configuring Hyper-V requires a few extra steps. A comprehensive guide can be found under the following Support Knowledge Base Article:
https://raynetgmbh.zendesk.com/hc/en-us/articles/360000308223-RSC200355-How-to-Configure-a-RayPack-Studio-Application-for-Hyper-V

Short checklist of the prerequisites for Hyper-V machines:

- WINRM has to be configured with `TrustedHosts` entries on both guest and host.
- PowerShell 3.0 or higher must be installed on the host.
- RayPack Studio Tools for Hyper-V must be installed on the guest machine and be a part of the base snapshot (so that they start each time when the machine boots after reverting to a selected snapshot).

The following checklist helps to find and fix any possible issues when working with Hyper-V machines:

1. Is PowerShell 3.0 installed (on both the Guest and the Host machine)?
   a. Check `$PSVersionTable.PSVersion` in PowerShell.
2. Is the machine properly configured in the **Settings** > **Virtual Machines** screen (pay attention to hardcoded IP addresses which may be dynamically assigned by DHCP).
3. Is RayPack Studio Tools for Hyper-V installed on the Guest machine? Is the process `vm-proxy.exe` from

RayPack Studio Tools for Hyper-V running?

4. Is WINRM configured?
   a. Check `winrm qc`.
5. Does WINRM have the proper `TrustedHosts` entries on both the VM and the server?
   a. `winrm s winrm/config/client '@{TrustedHosts="RemoteComputer"}'`.
   b. `winrm g winrm/config/client` - shows the current `TrustedHosts` lists.
   c. More information: https://technet.microsoft.com/en-us/library/ff700227.aspx
6. Does WINRM have a connection to the VM and vice-versa?
   a. `- Test-WSMan -ComputerName IP`.
7. Are all necessary ports unblocked on the physical machine?
   a. The default port range is `48654-48999`.

Note:
This version of RayQC does not support connecting to Hyper-V clusters.

Changing TCP / IP Configuration

In some cases it may be required to use custom port ranges, timeouts, etc. for VM related functionality.

The following table summarizes the available options:

Setting name	Default value	Description
`TcpIpDefaultPort`	`48654-48999`	The port range used for TCP / IP communication. Use minus (-) and comma (,) to indicate which ports are valid for incoming communication. Make sure that these ports are not blocked by your firewall. PackBot tries to find first valid free port and listens for it from lower to higher numbers.
`TcpIpMaxRetry`	`3`	The maximum number of retries before asserting that the machine is not available.
`TcpIpDefaultReceiveTime`	`240000`	Reverts to the default value if Windows does not define its own timeouts.
`TcpIpDefaultSendTimeout`	`240000`	Reverts to the default value if Windows does not define its own timeouts.

The options can be configured by editing the configuration file `RayQC.exe.config`. Each option is defined as a pair of key and value in the `<appSettings />`. For example, to change the default port to `50000` and the maximum number of connection retries configure the following:

```
[...]
<appSettings>
    <add key="TcpIpDefaultPort" value="50000" />
    <add key="TcpIpMaxRetry" value="5" />
[...]
```

Note that these options are not present out-of-the-box in the configuration file, in which case the defaults from the above table should be used.

Advanced Configuration Options

Besides the product configuration options that are available from the Settings section of the application UI, there are some additional configuration settings users may adjust to tailor RayQC towards their individual requirements.

Logging RayQC activity

The program data directory (`%AppData%\Roaming\Raynet\RayQC\Logs`) is used by default to store the application activity log files. A separate log file is created for each started RayQC Instance. If the default settings remain unchanged, RayQC adds a new line to this log file for every system INFO level message that is generated during application use.

In order to change the default settings for the log file behavior, users have to manually edit the `log4net.config` file, which resides in the root of the installation folder of RayQC (usually something like `C:\Program Files (x86)\RayQCAdvanced\`). The settings which are most likely to be of interest for adjustments are:

Log file storage location

The log files are by default stored in `"%appdata%\Raynet\RayQC\Logs`. However, it is possible to define any other absolute local paths as well as shared network locations for logging resource storage.

```
<file type="log4net.Util.PatternString"
        value="%env{AppData}\\Raynet\\RayQC\\Logs\\%date{yyyy-MM-dd HH-mm-
ss}.log" />
```

Be aware:
The user that runs RayQC must have write permissions in the log file location in order to initiate and maintain the message flow to the log file. If the user does not have sufficient access rights, there will be no error message, or actual product usage cutback, but simply a loss of system activity documentation. Please refer to the Troubleshooting section for additional instructions in case of missing logs.

Max. log file size

The max. log file size may be defined as "`KB`", "`MB`" or "`GB`". The default setting for newly installed RayQC instances is "`2048KB`"

```
<maximumFileSize value="2048KB" />
```

Log level

The most frequently used log level settings are `DEBUG`, `INFO`, `WARN`, `ERROR`, `FATAL`, `OFF`, whilst `OFF` prevents logging at all, `FATAL` is the most restrictive but still writing setting, and `DEBUG` the most talkative option.
The recommendation is to use the `DEBUG` level for newly setup systems, since a lot of the information logged in

this mode may help to adjust settings regarding access rights, and the like. As soon as the application and system are up and running productively, setting the log level to `WARNING` should be sufficient for permanent maintenance.

```
<level value="DEBUG" />
```

Default logging configuration

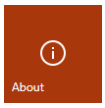
The default configuration file is given below as a review and backup support:

```
<?xml version="1.0" encoding="utf-8" ?>
<log4net>

    <appender name="RollingFileAppender"
type="log4net.Appender.RollingFileAppender">
    <file type="log4net.Util.PatternString"
        value="%env{AppData}\\RayQC\\Logs\\%date{yyyy-MM-dd HH-mm-ss}.log" />

    <rollingStyle value="Once" />
    <maxSizeRollBackups value="2" />
    <maximumFileSize value="10240KB" />
    <staticLogFileName value="false" />
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%date [%thread] %-5level %logger - %message%
newline" />
    </layout>
  </appender>

  <root>
    <level value="DEBUG" />
    <appender-ref ref="RollingFileAppender" />
  </root>

</log4net>
```

Further Information

RayQC uses an external library to provide logging functionality. Please refer to the online-documentation provided for the log4net project (http://logging.apache.org/log4net/) in order to get further details regarding available configuration and usage options. log4net can be adjusted to connect directly with databases or event-loggers. There are numerous options for layout and behavior manipulations. RayQC system administrators with a slight affection for perfection are highly welcome to configure their very own logger version.

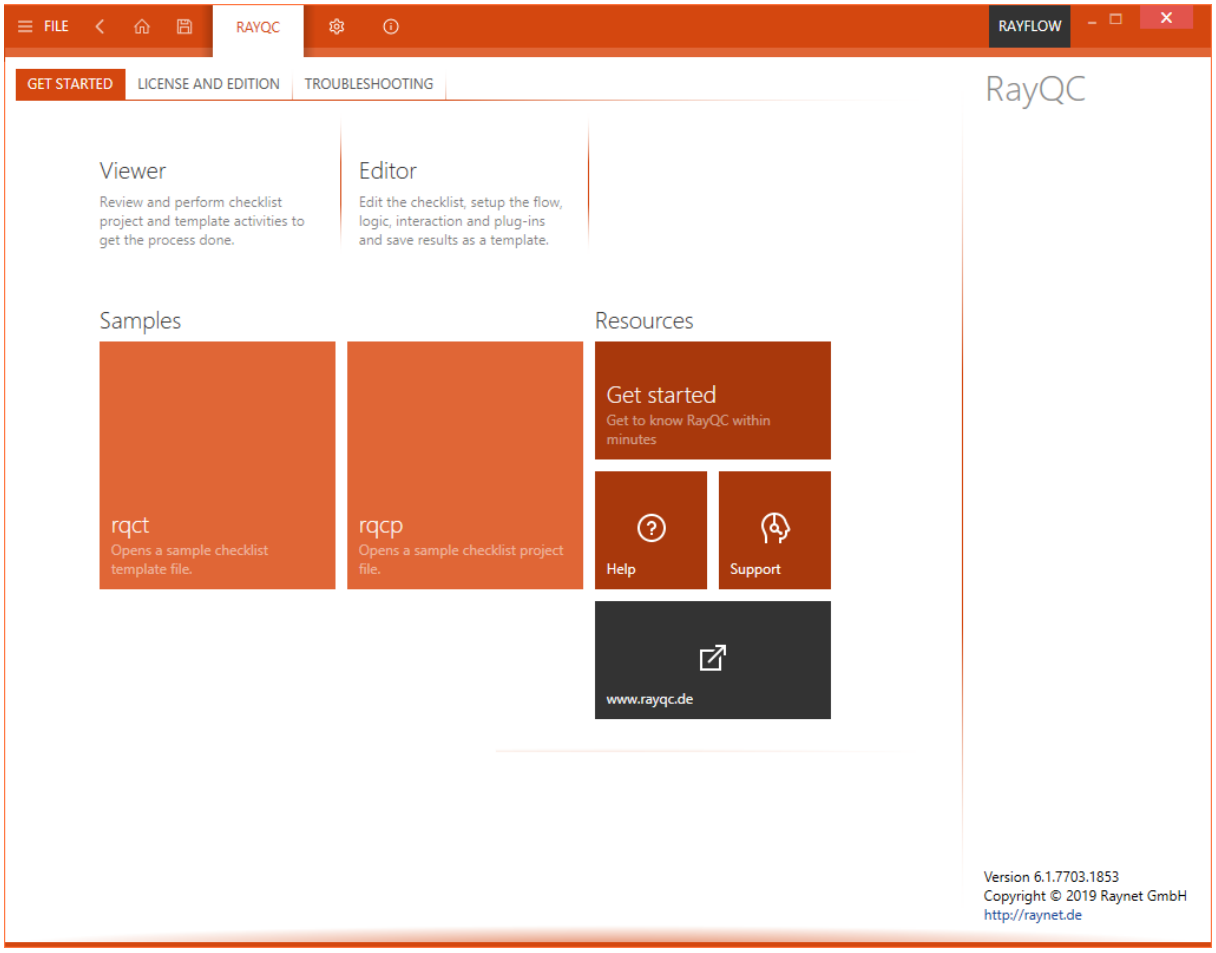
About



Clicking on the **About** tile from the **Home Screen** displays the about area. It contains the **get started** view, providing information about the individual tools of RayQC, Samples and links to various resources. Additional supportive views regarding **license and edition** as well as **troubleshooting** are available by clicking on the other view tabs in the **About** area.

Get Started

Choosing the **get started** button from the Home Screen reveals this view in the about section. It contains information about the individual tools of RayQC, Samples, and links to various resources. Additional supportive views regarding **license and edition**, as well as, **troubleshooting** are available by clicking on the view tabs.



Meet RayQC

These are the two main components of the application UI: **Editor** and **Viewer**.

Samples

RayQC contains samples to show how to use the different interfaces in order to create checklists with plug-ins, conditions, post processing options, and how to evaluate them later on. Click on one of the tiles to open a sample checklist template or project.

Resources

RayQC includes various resources that can be used to make the experience with RayQC more productive and provide help where needed. Please note that some resources (including some items in this help file) are only available online or with an internet connection.

get started

This opens the get started help file, which includes tips & tricks how to configure and use RayQC.

help

Opens this document.

support

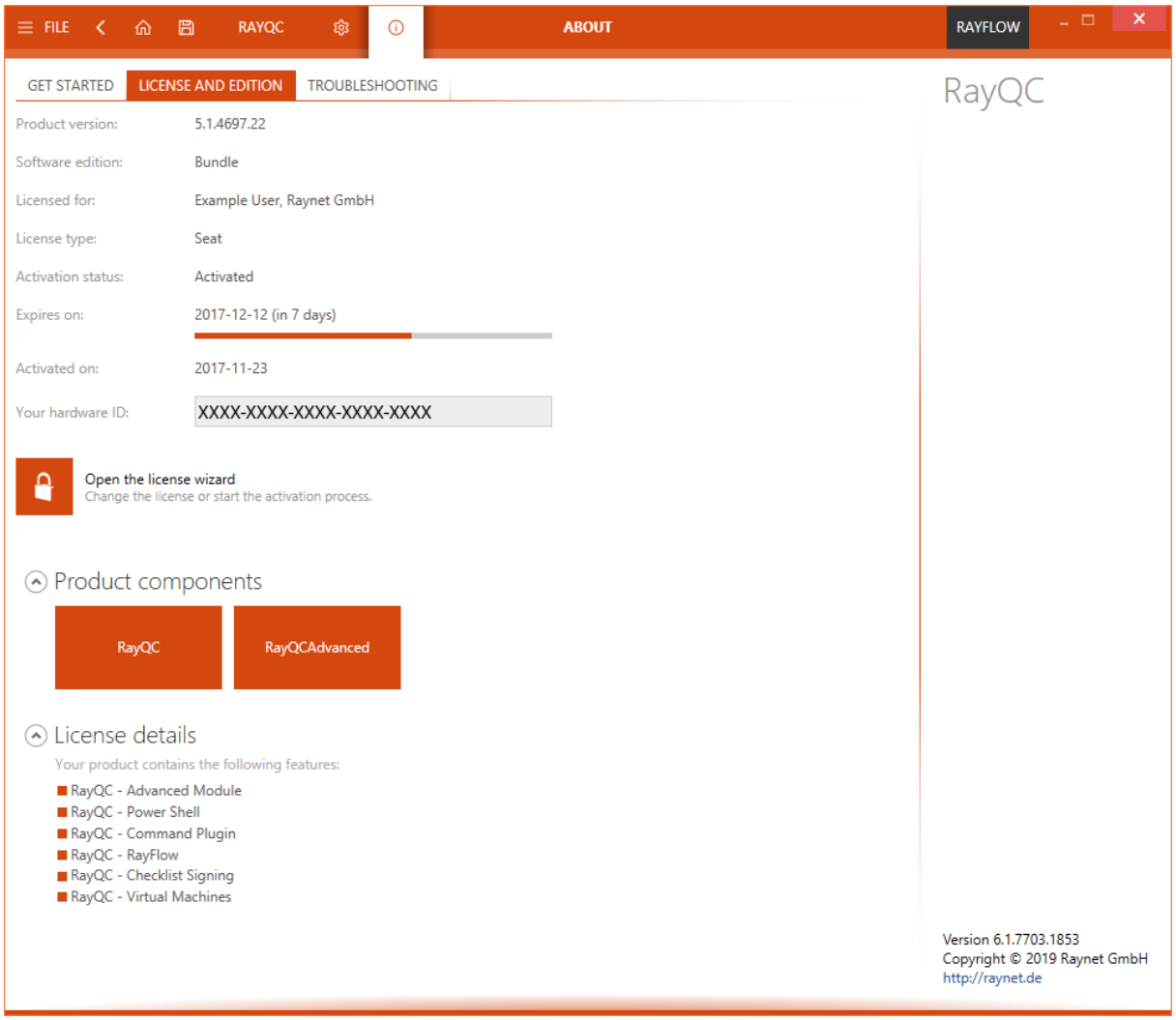
Opens the website for product support contact.

www.rayqc.de

RayQC on the web.

License and Edition

The **About** area also contains the **LICENSE AND EDITION** tab, providing all usage relevant license information about the current product instance.



Changing the license can be achieved using the license wizard by choosing **Open the license wizard**.

Active product edition

RayQC is available in different product editions: Standard, Bundle, and Enterprise. Please note that the license that has been used to activate the current application instance takes direct effect on the set of features which are effectively available. Therefore, some of the options documented within this User Guide may not be executable with the current RayQC installation.

The features which are part of the activated product edition are shown as a list under **Your RayQC features**. If a bundle license has been activated, the following features should be available:

- RayQC - Checklist

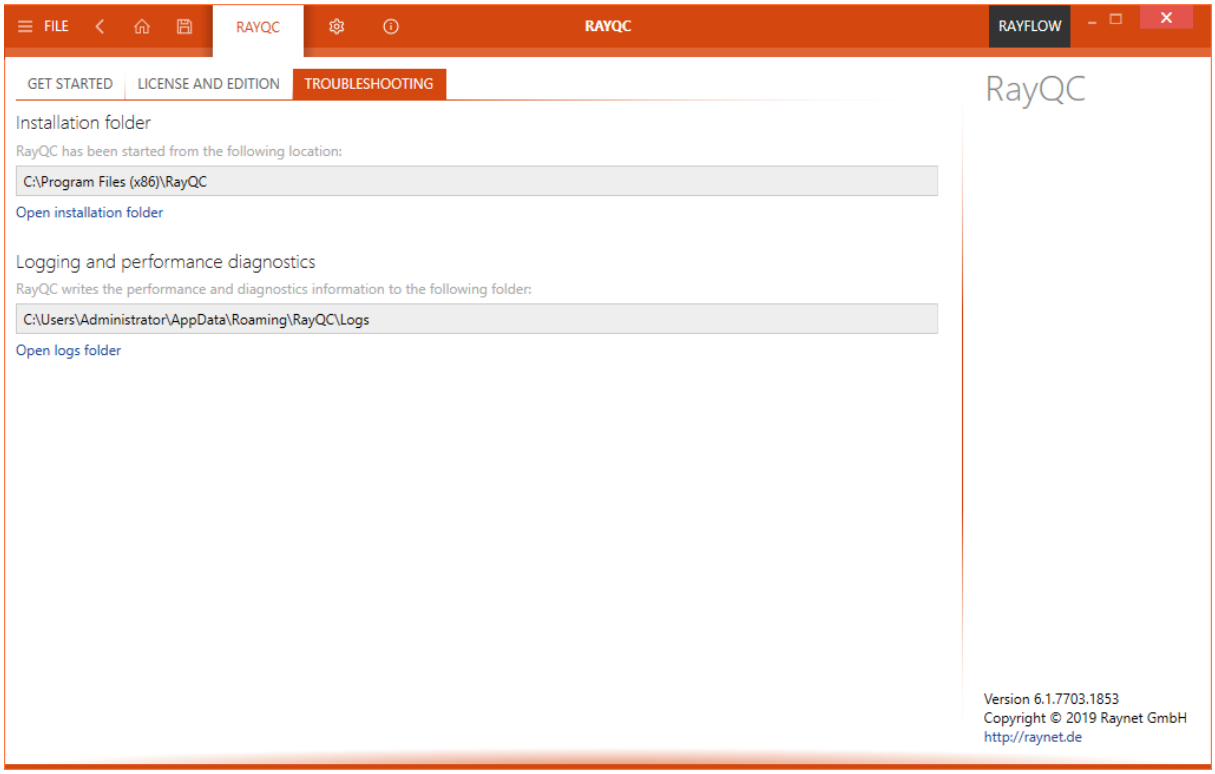
- RayQC - Advanced Module
- RayQC - Power Shell
- RayQC - Command Plugin
- RayQC - RayFlow
- RayQC - Checklist Signing
- RayQC - Virtual Machines

Note:
Please contact our *support team* or your Raynet *sales representative* if you have any questions regarding the licensing status of your RayQC instance.

Troubleshooting

The **About** area contains the **TROUBLESHOOTING** tab, providing handy information about the crucial system paths defined for the current product instance.

Any information displayed within this tab is read-only. It is provided to ensure transparency about the current instance settings, which is vital for proper troubleshooting measures in case of support relevant issues.



Installation folder

This path shows where the currently running instance of RayQC is installed. Click on the **Open installation folder** link to open a windows explorer instance at the displayed location for further instance resource review.

Logging and performance diagnostics

RayQC is by default configured to write log files with information about each product work session. A new log file will be generated for every launch of the current application instance. These log files contain vital information for any troubleshooting or help desk measure, such as environment information about the physical machine RayQC resides on, the steps performed during a session, and exception details thrown by the application in case of operational issues. Please make sure that the target directory displayed here provides sufficient disk space and is accessible for the currently logged in user. Click on the **Open logs folder** link to open a windows explorer instance at the displayed location for further instance resource review.

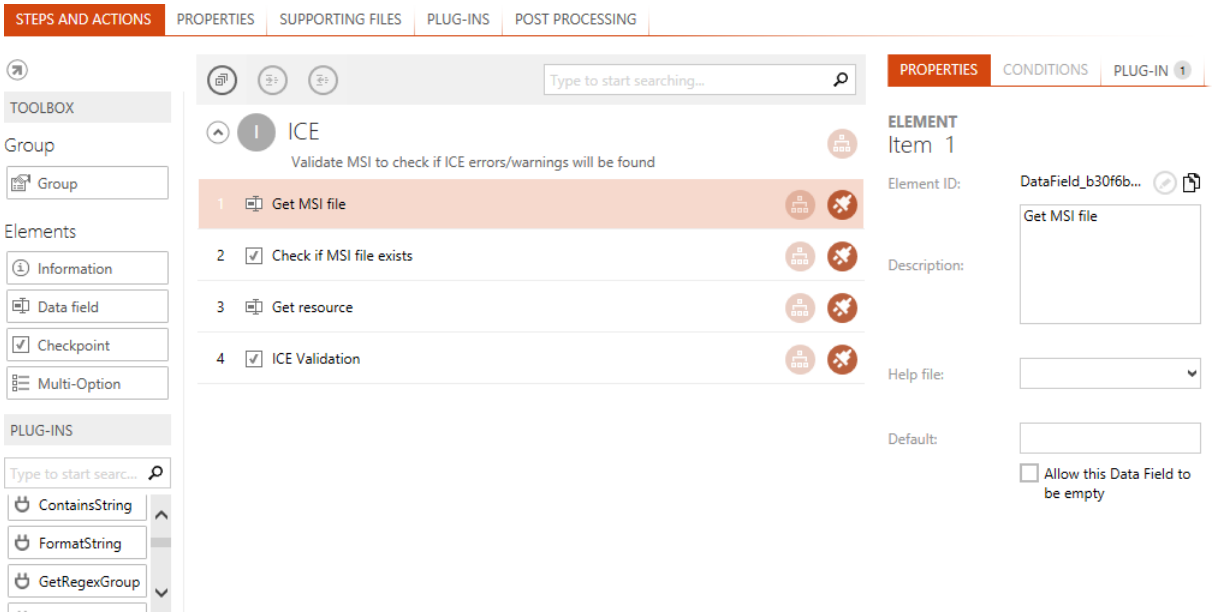
Checklist Structures

As already mentioned before, RayQC checklists are XML based. Therefore, the backbone of checklists may be analyzed either from the underlying XML file angle, or from the superficial Checklist Editor interface scope.

Please refer to the Appendix sections Basic Checklist Structure and Checklist Example to take a look at the structure of the XML base. The topics within this section Checklist Structures are designed to describe things from the UI angle of the Checklist Editor.

Basic Checklist Properties

The screenshot below shows the Checklist Editor interface when a minimal checklist project is opened for manipulation. Actually, it is the default project that is automatically created within the temporary session memory whenever a user creates a new checklist template file. The editor interface is a composition of different areas of activity:



The title of the currently opened checklist may be edited by selecting the PROPERTIES tab.

The listing of currently defined checklist elements and checklist properties is displayed within the **checklist canvas** below the title.

The canvas tab navigation provides access to the editor interfaces for the following property groups:

- Steps and Actions allows to manipulate the overall checklist structure, with groups and the elements nested within them. Defining conditions, using internal or external plug-ins, changing element and group order - all of that is executable here.
- Properties provides access to change basic checklist properties, such as description, bypassing options, and the like.
- Supporting Files is designed as collecting container for all external files used as resource within the

checklist, such as images, help files, and the like.

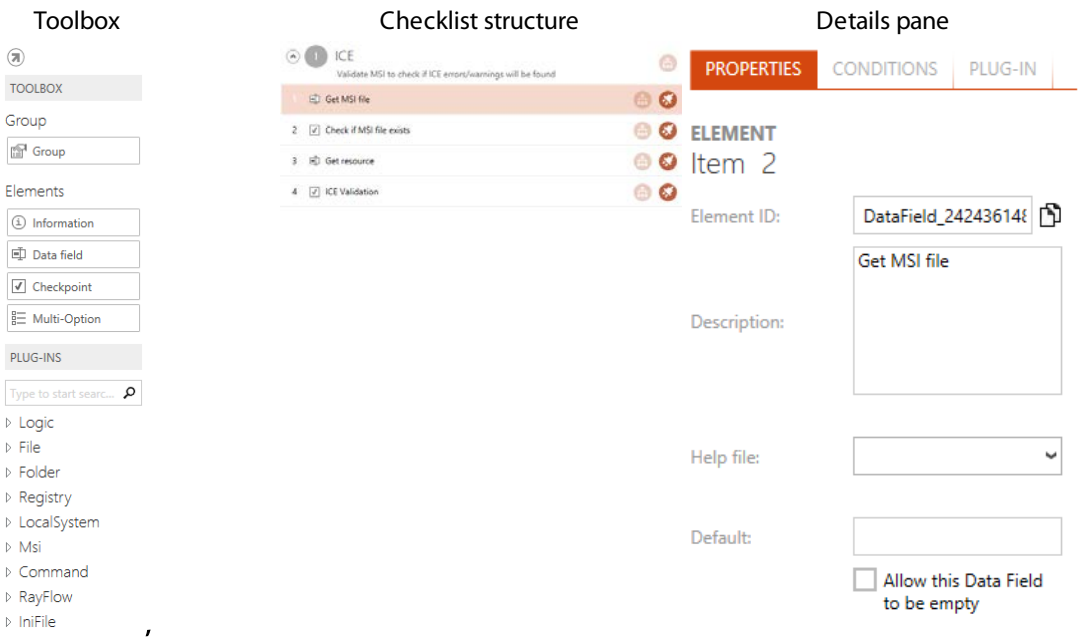
- Plug-Ins allows to add external plug-ins to the current checklist template. Once added, external plug-ins may be used from the plug-in section in the toolbox of the Checklist Editor interface.
- Post Processing enables standard actions RayQC executes whenever a project run based upon the checklist is finished, such as sending updates and reports to RayFlow.

The following sections provide details about each tab and the manipulation options they provide.

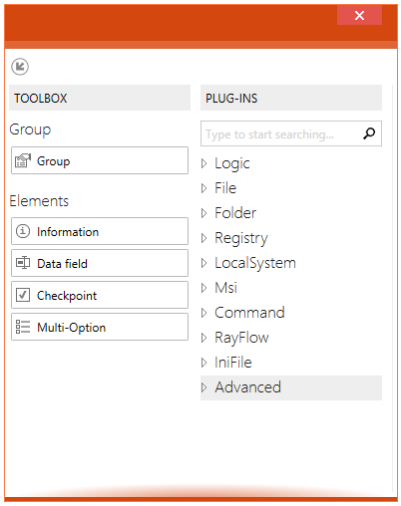
Steps and Actions

View organization

This default checkbox editor view is separated into three columns of activity:



The **toolbox** at the left contains items that may be added to the checklist structure: groups, elements, and plug-ins.



The toolbox allows adding new objects to the current checklist item flow. By simply dragging a group or element to the area with the already defined elements and dropping it at the desired target position, users add a default object that is ready for adjustments. Please refer to the *element* related sections to get more information about the options available for adding, moving and manipulating checklist items. An additional section is available with *group* option details.

Use the arrow icons at the upper left corner of the toolbox to undock or dock the whole column. The undocked toolbox allows users that operate on monitors with small resolutions to organize their Checklist Editor interface according to their individual space requirements. Adjusting the height and the width of the undocked toolbox is possible by using the standard application window resizing functionalities provided by the underlying operating system.

The **checklist structure** is displayed within the center column of this view. It shows all groups and elements of a checklist along with their hierarchy structure.

Selecting an object within the center column loads it's data into the **details pane** at the right-hand side of this editor view. From this pane, users have access to modify group & element properties (such as description, options, etc.), conditions and plug-in usage.

Checklist organization

A checklist consists of any number of *groups*, which can in turn contain any number of *elements*. Whist groups are aligned in a flat sequence of single objects, elements may be adjusted in multi-level tree structures. From an organizational point of view, groups are task bundles, whilst their elements are single task steps users have to perform whilst evaluating the checklist.

Both, groups and elements, may be equipped with conditions, so that they are executed according to the results of prior element tests. Checklists are assumed to be worked in a top-down step by step manner. Therefore, the evaluation of conditions for dynamic content availability always expects prior checklist items to be already evaluated.

The result of all currently visible checklist items is summed up to the general checklist result, which is displayed as a color coded ribbon at the lower right corner of the Checklist Viewer interface. An element result may consist of a boolean Yes or No information, a comment string, or the selection of a single value from an option set. According to the integration of automated plug-ins, item results may be entered manually by the evaluating user, or automatically as a result of a plug-in logic execution.

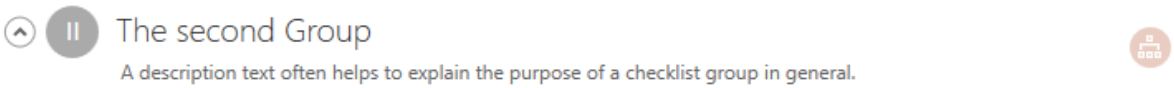
The default checklist is already equipped with a group, which itself has been filled with a Data Field checklist element. Starting from this minimal setup, users are free to add as many groups and elements as required to fulfill the checklist purpose.

Be aware:
In order to keep checklist templates as clear as possible, it is not allowed to add elements to a checklist directly. They always have to be defined within a group container.

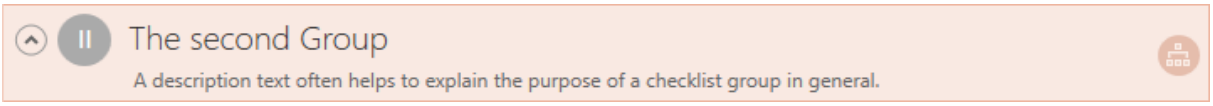
Groups

Group objects are structural elements, designed to support organizational needs and logical restrictions for checklists.
Within the **toolbar**, group items are represented by a group item. Users have to apply drag and drop on it in order to create new groups within checklists.

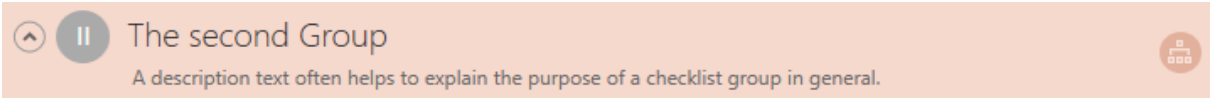
Within the **checklist structure** column, each group is displayed with a headline that contains its roman index number value (I, II, III, IV, etc.) and the group title. Elements that belong to the group are displayed slightly indented, as an element hierarchy tree below the group headline. Once the user clicks on the headline, the background-color switches to orange and the group properties are loaded into the tabbed area of the **details pane** on the right-hand side of the Editor interface.



The headline of a group within the Checklist Editor: Roman index value, title and description



The headline of an unselected group when the mouse pointer hovers above it.



The headline of a selected group with colorized background

Please refer to the *Groups* section for further details regarding the options available within group objects, such as conditions, element positioning and nesting, and the like.

Checklist elements

The checklist elements are single test steps. They have to be bundled within groups in order to provide a decent checklist task organization. Therefore, elements always reside within group containers. They are represented by single boxes with a light orange background color once they are selected by a left-click. Depending on the *element type*, different *options* may be set by the integrated **control** buttons.

Each element has a number, displayed in the upper left corner of the element resemblance within the checklist structure. The number is unique and incrementing within each parent group. If elements are nested, each level has got its own numbering resemblance (e. g., 1, 1.1, 1.2, 1.2.1, 1.2.2, etc.).

An element within the structure of the Checklist Editor: Index value, description, type icon, plugin and condition indicator icons

The unselected element item when the mouse pointer hovers over it.

The selected element item with colorized background.

Groups

Whenever a user selects a group object, its details become visible and editable by the interface that is immediately loaded into the details pane. There are two sub-tabs present in the details pane when groups are selected: **Properties** (including title and description) and **Conditions** (including conditions that determine the visibility of the group contents for later checklist evaluation runs).

Basic Group Properties

Group Title

The group title is the string that is always visible for group recognition. Therefore, the group title should be a short, cut to the core outline of the group purpose.

When a new group is dropped within the checklist structure, a default value ("Group Title") is already present within the title input field, ready for proper adjusting.

The title has to be an alphanumerical string, and is not evaluated regarding string formatting tags.

Group Description

The group description is only visible in the Checklist Viewer and Editor interfaces. The description is designed to provide detailed information and evaluation instructions for the elements bundled inside of the specific group container.

In order to allow editors to create well-structured and easily readable descriptions, it is actually possible to enrich the string with basic *formatting options*, which even allows images to be added to the description area.

Group Conditions

Conditions can be defined for groups and elements. A condition is always based upon the result of one or more element evaluation results. Therefore, the first group of a checklist may never be equipped with conditional dependencies, since no elements have been evaluated before.

However, as soon as the second group is added to a checklist, it is possible to define **Conditions** for the visibility (and along with it the availability for evaluation) of this group. Users may add **Conditions** that depend on the result of Checkpoint and Multi-Option elements.

Conditions can also be added from any existing checklist element result, no matter if it is an automated, plug-in-based element or a manually filled in value as long as that element is located in a group that is located prior to this group.

Be aware:
When **Conditions** are heavily used, it may happen that a user loses track of the actual evaluation path. Since it is possible to add **Conditions** regarding the result of conditional objects, it is actually possible to create conditional statements that may never result in the visibility of a certain group or element. Therefore, it is highly recommended to double-check conditional constructs, and additionally use the **Validate Conditions** button from the **Checklist Viewers Swipe bar**. If the check returns issues, these should be cleared before the checklist is deployed for productive use.

To enable the condition interface, users have to activate the **Conditions** tab within the details pane of a group container. As soon as the tab is active, users are able to drag and drop elements from the checklist structure to establish one or more buckets of conditional statements.

Each conditional statement consists of two parameters: The checklist element it relates to, and the actual value of that element that is expected in order to evaluate this specific conditional statement to true. Use the show button to highlight the element a conditional phrase depends on within the Checklist Structure area. Please refer to the *Conditions* section for further details on the mechanisms behind conditional statements, buckets and their evaluation by RayQC.

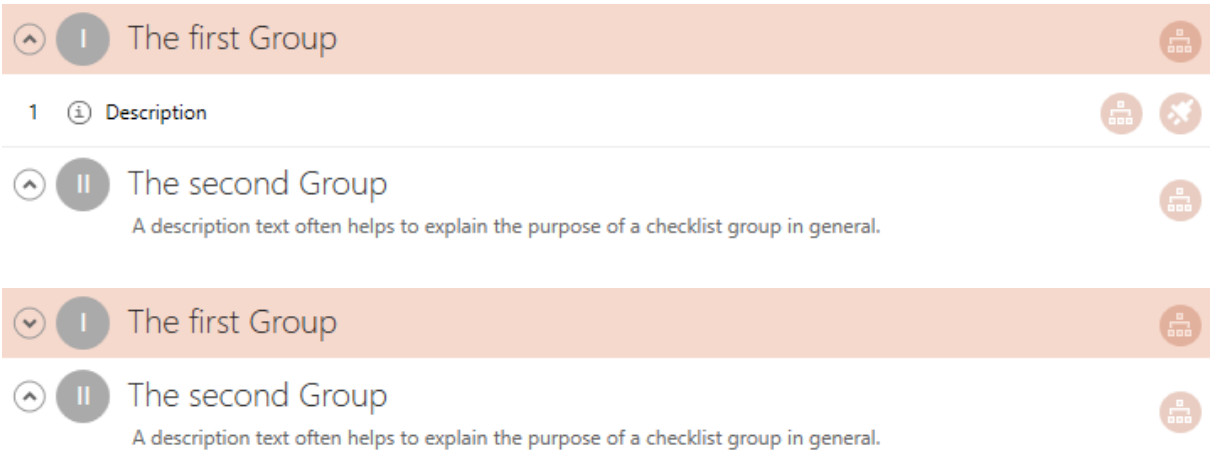
The conditional statement line is removed from its parent bucket as soon as the delete icon at its right-hand side is clicked.

Group Controls

Once a group has been added to a checklist, it offers some controls for manipulation:

Expand & Collapse Group Content

The screenshots below show a default group container in its expanded and collapsed display mode. To switch between these modes, the group object is equipped with an arrow at the left-hand side of the **Group Title**.



If the arrow points up, clicking it collapses the group container.

If the arrow points down, clicking it expands the group container.

Note:
Expanding or collapsing a group does not take effect on the availability of the group elements for future evaluation procedures. Groups may be expanded and collapsed from both, the Checklist Viewer, and Checklist Editor interface. The current display mode is not stored permanently, but is reset to the default as soon as the checklist object is closed. Switching from **Viewer** to **Editor** does not reset the extent of a group container.

Move Group Up or Down

Groups are aligned as a flat sequence of one container following the other. Therefore, they all reside on one horizontal level, but may be reordered towards their vertical sequence. To do so, users apply drag & drop: Clicking on the group header, dragging the group container to the desired new position before or after another group, and dropping it there is done in seconds. RayQC presents an error tooltip if moving a group leads to inconsistencies regarding conditions and / or plug-in execution result usages: Elements and groups may not refer to objects that are positioned later within a checklist. If this rule would be broken by a move action, dropping to an invalid state is prohibited and a message is displayed showing the conflicting element(s).

Remove Group from Checklist

Every user with access to the Checklist Editor view may remove groups from the current base checklist structure. To do so, users either right-click the group header and select **Delete** from the context menu, or hit **Delete** on the keyboard once a group header has been selected by a left-click.

A confirmation dialog is displayed, expecting the user to acknowledge the deletion of the group and all attached elements, conditions and logic. Please note that the group deletion is permanent and irreversible as soon as the changes to the checklist file are saved.

- Hitting the **YES, REMOVE** button within this confirm dialog actually executes the object deletion.
- Hitting the **DO NOT REMOVE** or **CANCEL** button within this confirm dialog aborts the deletion.

Since each checklist project must contain at least one group object, it is not possible to delete the last group from a checklist.

Be aware:
Removing a group automatically removes all items that have been organized within the group from the checklist. Therefore, dependencies users have built with conditions targeting the elements placed within the group become invalid and are automatically removed along with the group object. However, objects that have been referenced from the file system, such as plug-ins, help files, images, and the like will not be affected by removing a group that makes use of them. Please remove these items from your *supporting files* pool to maintain a strictly clean checklist resource library.

Elements

Elements are the actually decisive objects within a RayQC checklist. They are defined within the XML based structure of the template file type (`*.rqct`), and come alive when result values are added to form a checklist project instance (`*.rqcp`)

As outlined before, there are some basic properties which are present for all element types within RayQC, whilst others are rather type-specific. The following summary is describing the general properties, whilst later sections go into individual details.

Elements are displayed as indented boxes within the group elements of checklists. The displayed content depends on the currently active application view (Checklist Viewer or Checklist Editor) and the selection status of the element.

Basic Element Properties

The basic properties are named in the order of their position within the element boxes.

Position index value

The index value is a dot separated list of hierarchical level values, displayed within the upper left corner of the element box representations. All elements have an explicit index value, unique within their parent group. The first digit group gives an outline of the position within the elements on the root level of the groups item tree structure. The second and all subsequent digit groups define the position of the item within the child element list of their parent element. The maximum depth of a checklist element hierarchy is 4, which leads to a position index length limitation of 4 levels (e. g. 1.2.3.4).

Users who evaluate the checklist refer to elements by their position index value, since the internal identifier (element id) is by default not visible for them. However, the index value is present in all checklist views and view modes.

The position index cannot be entered manually, but changed by moving the element, either dragging it up and down within the vertical evaluation sequence of elements, or using the left or right icons (which vertically changes the indent, and therefore the hierarchy level, but not the vertical sequence of elements).

If an element is not available for evaluation in the Checklist Viewer (e. g., due to condition restrictions), the position index value sequence of the parent group is not changed, which means that there may be gaps when the checklist is opened within the Checklist Viewer. (For example, an item number 7.3 may directly be followed by 7.5, due to a conditional availability of item 7.4)
Triggering index value changes by the appearance of dynamic tree branches of the checklist would surely lead to clear element identification issues, which is why living with gaps in the index sequence is the preferred operational method.

Element type

Each element has a specific, fixed type. Once an element has been added to a checklist, its type cannot be changed any more. The element type decides about options and controls available for an element, since each type has a unique purpose by design, which require different editor choices and possibilities.

Within the Checklist Viewer, evaluators recognize the item type by the result input controls. Information elements do not have result input controls, Data Field elements display a text input field, Checkpoint elements a radio button, and Multi-Option elements a selector control. Within the Checklist Editor, authors recognize the type by the icon displayed within elements box representation.

Please refer to the upcoming Element Types section for further details.

Element ID

The element ID is the internal identifier of elements, used for example to share item result values between items and their conditions, plug-ins, and the like. Prior versions of RayQC revealed the ID within the Editor or Viewer interface in order to allow users to manually establish relations between elements. However, thanks to the advanced Editor interface with its drag & drop capabilities, there is no longer need for manual relation definition. Therefore, neither element ID's, nor group ID's are displayable within the application views. If users have to find out about these properties, they have to open the underlying checklist structure XML file with an external editor.

Description

Each element has a description property, designed to contain the task description given for the current checklist item. The Information type for example usually does not consist of much more than the description property.
The text can contain RayQC specific format markup tags to allow editors to add some structure, or even descriptive images, to their task definitions.
It is recommended to outline not only instructions on the actual task, but also note conditional options and dependencies set upon the item within the text.

Note:
If the description alone does not provide enough possibilities to give the full task description, each item may be augmented with an additional help file. BUT: the text of an element is always plain to see, whilst help files have to be brought to view by clicking on the help icon optionally displayed within the elements box representation in the Checklist Viewer mode.

Element Types

RayQC knows four basic types of checklist elements:

Information

- Designed to display info text and help files as additional support for users who evaluate the checklist
- Do not require result feedback during evaluation
- Do not directly affect the overall checklist result
- May not be used as trigger elements for plug-in execution
- May contain conditions, but are not usable as criterion for conditional statements themselves (i. e. they can be displayed dynamically based on the results of other elements, but cannot be decisive for the availability of

other elements or groups)

Data Field

- Designed to gather textual evaluation results
- Require a string as check result
- Do not directly affect the overall checklist result, but usually have to be filled with a result in order to allow a checklist to switch from **NOT FINISHED** to a final **PASSED** or **FAILED** status.
- May be used as both: trigger elements for plug-in execution and target containers for plug-in execution return values
- May contain conditions, but are not usable as criterion for conditional statements themselves (i. e. they can be displayed dynamically based on the results of other elements, but cannot be decisive for the availability of other elements or groups)

Checkpoint

- Designed to gather exact result states
- Demand a boolean Yes / No check result, usually with a predefined setting regarding the expected (correct) evaluation result
- Have direct decisive influence on the overall checklist result (**PASSED** or **FAILED**)
- May be used as both: trigger elements for plug-in execution and target containers for plug-in execution return values
- May contain conditions, and are usable as criterion for conditional statements themselves (i. e. they can be displayed dynamically based on the results of other elements, and can also be decisive for the availability of other elements or groups)

Multi-Option

- Designed to gather exact result values, mainly for checklist branch availability control and the collection of meta-data
- Allow to select the actual result from a set of predefined options (1 of n)
- Do not directly affect the overall checklist result, but have to be filled with a result in order to allow a checklist to switch from **NOT FINISHED** to a final **PASSED** or **FAILED** status.
- May be used as both: trigger elements for plug-in execution and target containers for plug-in execution return values
- May contain conditions, and are usable as criterion for conditional statements themselves (i. e. they can be displayed dynamically based on the results of other elements, and can also be decisive for the availability of other elements or groups)

These RayQC checklist element types are equipped with differing sets of options and methods. Just to name one: Conditions may only be defined based upon the result of Checkpoint and Multi-Option elements. Therefore, any checklist without at least one of those element types is strictly unconditional and has an always identical straight forward evaluation path.

Please read the following sections for in-depth information regarding the element types and their individual specifications.

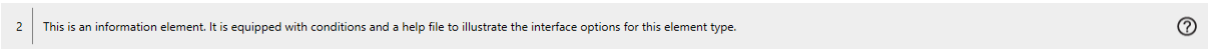
Adding Information items to a checklist group basically leads to the display of an info box. Therefore, Information elements are designed to provide additional hints and details regarding a specific Checkpoint or test group of a checklist.

However, the Information item may be manipulated towards the checklist template requirements by customizing the following properties:

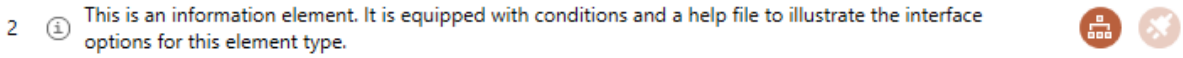
- Formatting the element description text with RayQC **markup** tags
- Adding a **help file** that may be opened within an external viewer
- **Conditions** regarding the actual availability of the item itself (according to the general requirements for condition usage)

Be aware:
Items of this type may not be evaluated within conditional statements themselves, since they do not have an actual result value that conditions might somehow interpret to decide about the right conditional path to follow.
The same reason (missing container to carry result information) prevents them from being triggers for plug-in executions.

Please refer to the linked document sections to read details about the options available for each Information item property.



Information elements as they are displayed within the Checklist Viewer interface. The background color may be grey or white due to the alternating schema of the group elements.



Information elements as they are displayed within the Checklist Editor interface. The background color may be white, light orange or orange due to the selection and hover state of the element.

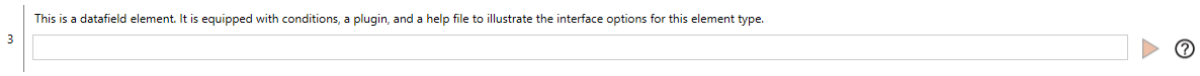
Checklist items of this type demand input as evaluation run result. It does not matter if the text is entered manually by the evaluating user, or automatically by plug-in logic.

However, Data Field items may be manipulated towards the checklist template requirements by customizing the following properties:

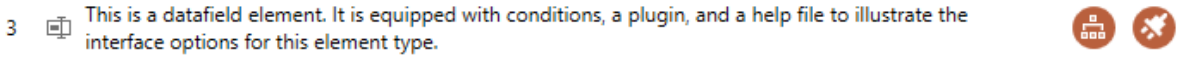
- Formatting the element description text with RayQC **markup** tags
- Adding a **help file** that may be opened within an external viewer
- **Plug-ins** can be added to use external script or executable logic for evaluation purposes.
- **Conditions** regarding the actual availability of the item itself (according to the general requirements for condition usage)
- If the **no restraint** option is active, the Data Field is optional, which allows to finish the evaluation without providing the comment text

Be aware:
Items of this type may not be evaluated within conditional statements themselves, since they do not have a result value type conditions might somehow interpret to decide about the right conditional path to follow.

Please refer to the linked document sections to read details about the options available for each Data Field item property.



Checklist Viewer display of a Data Field element with the text input field expecting the result value. The background color may be grey or white due to the alternating schema of the group elements.



Data Field elements as they are displayed within the Checklist Editor interface. The background color may be white, light orange or orange due to the selection and hover state of the element.

Checkpoint entries usually require a definitive **Yes** or **No** answer from the evaluating user. The result may be selected manually, or injected by a specific plug-in result.

However, a Checkpoint comes along with an expected answer, which is **Yes** by default. When the checklist is evaluated, the expected answer option is shown by marking it bold (as shown within the screenshot below). Even though this default setting may be switched to expect **No** as correct answer, the user is still forced to note a result. This default demand may be switched off by the don't evaluate option (see *below*).

If the expected answer is not the evaluation result, the executing user has to document a failure comment. The intention behind this Checkpoint behavior is to improve the test result quality by adding information about reasons for failures. If a checklist result is sent to the person that has to fix issues that came up during the evaluation process, getting details on circumstances and symptoms of an issue definitely support a quick and correct resolving

If a Checkpoint answer is not matching the expected answer, the result of the whole checklist is set to failed. This default behavior can be overridden by the **Allow Exception** option. If this option is enabled, users are requested to add a reason for the exception which will be displayed along with the standard failure comment.

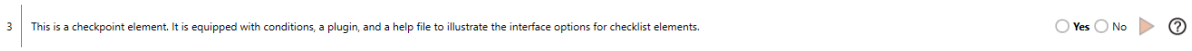
This setting once again supports the idea of proper communication between the team members which are involved in the whole workflow of object creation and testing: Even though a checklist template editor may know which Checkpoints can trigger non-decisive exceptions, the evaluator himself may very well not be able to correctly judge which exceptional failure is acceptable and which one is not. At the same time it is possible that external dependencies, such as test environment settings or unusual test preconditions, take effect on the result, which should not lead to a general failure of the whole checklist.
Therefore, adding not only the option for exceptions, but at the same time a requirement for additional exception descriptions allows to document the check result in a proper manner for both - further test object and workflow improvements.

Use the links provided below to directly jump to detailed sections regarding available property setting options for checkpoint entry elements:

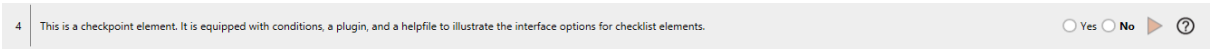
- Formatting the element description text with RayQC **markup** tags
- Adding a **help file** that may be opened within an external viewer

- **Plug-ins** can be added to use external script or executable logic for evaluation purposes.
- **Conditions** regarding the actual availability of the item itself (according to the general requirements for condition usage)
- **Switching the value** means to define No as expected result for this Checkpoint entry (not Yes as expected by default)
- If the **Evaluate this element** option is not checked, there is no expected check result, which allows to finish the evaluation independent from the actual answer
- **Exceptions** enable a successful checklist evaluation result even if this particular Checkpoint entry was marked to have failed

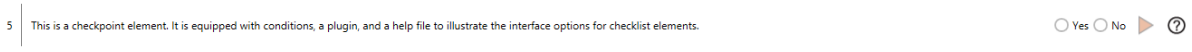
Tip:
The result of Checkpoint entries can be selected as decisive condition for the availability of checklist groups and items. Please refer to the *conditions* section for details!



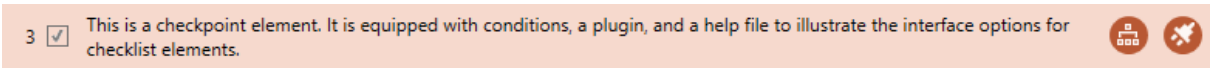
Checklist Viewer display of a Checkpoint element with default setting: Yes is the expected, correct result. The background color may be grey or white due to the alternating schema of the group elements.



Checklist Viewer display of a Checkpoint element with switched value setting: No is the expected, correct result.



Checklist Viewer display of a Checkpoint element with don't evaluate setting: There is no explicit result expectation.



Checkpoint elements as they are displayed within the Checklist Editor interface. The background color may be white, light orange or orange due to the selection and hover state of the element.

Multi-Option elements require a definitive "1 from n" answer selection from the evaluating user, where n is any number greater than 1. The result may be selected manually or injected by a specific plug-in result.

In contrast to *Checkpoint elements* with their **Yes** / **No** labels, there is no validation setting for the Multi-Option item choice option labels. When a checklist template editor adds Multi-Option items, a custom label text for each added result option has to be defined. This label is the only decision criteria evaluators will have to make their choice.

Multi-Option elements have two essential purposes: First of all they are suitable for mere information requests from the evaluating user. Furthermore, they are the perfect companion for the creation of dynamic content switches. Whenever a checklist has to provide several evaluation paths that depend on a specific test result or user choice, providing a set of triggers as Multi-Option choices allows making different branches of the checklist template available or unavailable.
Please refer to the *conditions* section for further details on the possibilities for dynamic content definition.

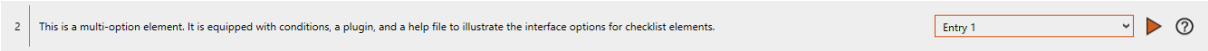
During checklist evaluation procedures, the selected option does not take effect on the **PASSED** or **FAILED** status of the overall checklist, since there is no right or wrong, true or false definition for the options. However, as long as there are Multi-Option items without user selection within a checklist project, the status will always

be **NOT FINISHED**.

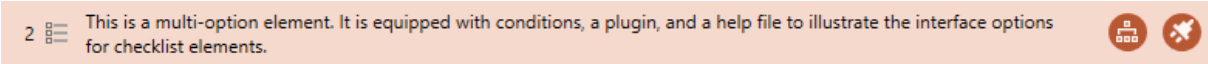
Please use the links provided below to directly jump to the details sections regarding available property setting options for Multi-Option elements:

- Formatting the element description text with RayQC **markup** tags
- Adding a **help file** that may be opened within an external viewer
- **Plug-ins** can be added to use external script or executable logic for evaluation purposes.
- **Conditions** regarding the actual availability of the item itself (according to the general requirements for condition usage)

Tip:
The result of Multi-Option elements can be selected as decisive condition for the availability of checklist groups and items. Please refer to the *conditions* section for details!



Checklist Viewer display of a Multi-Option element with the drop-down selector control expecting the result choice. The background color may be grey or white due to the alternating schema of the group elements.



Multi-Option elements as they are displayed within the Checklist Editor interface. The background color may be white, light orange or orange due to the selection and hover state of the element.

Element Options

The following section provides a general overview regarding the available options for checklist items. Whilst several options may be defined for any type of item, some options are not available for all item types. Please refer to the element types section for a list of available options per type.

Element options are defined within the Checklist Editor and apply to the availability, evaluation and display style evaluators experience during checklist runs within the Checklist Viewer interface.

Each item of a checklist may be linked to a help file, containing additional information about the specific test task, such as constraints, procedure steps, and the like. At present, RayQC allows to add RTF and PDF files as help files.

Be aware:
Whilst RTF files are displayed by an integrated viewer, PDF files may not be accessible on all evaluator systems. An external PDF viewer application is required to open PDF help files, e. g. Adobe Reader.

If an item has a help file link, a question mark icon is displayed at the right-hand side of the item box within the Checklist Viewer interface. As soon as a user clicks the icon, an additional application window is launched, displaying the content of the linked help file. In case, no help file is linked to the element, an error message is shown when this icon is clicked upon.

Note:
Help files have to be added to a checklist before they can actually be used within elements. Please refer to the information provided within the section about the supporting files tab of the Editor interface for instructions how to manage help files.

To activate this option

1. Within the Checklist Editor, scroll to the checklist item that has to be equipped with a help file link, and select it with a left click.
2. Activate the properties tab of the details pane on the right.
3. Select one of the available help files by expanding the help files selector menu and clicking on the desired file name.
4. Save the changes to the checklist template.
5. Switch to the Checklist Viewer to check whether the link to the help file is valid, and clicking the help icon really opens the expected file.

To deactivate this option

1. Within the Checklist Editor, scroll to the checklist item whose help file link has to be removed, and select it with a left click.
2. Activate the properties tab of the details pane on the right.
3. Click on the help files selector to expand the menu, and hit Delete on the keyboard to remove the relation between the help file and the currently active element.
4. Save the changes to the checklist template.

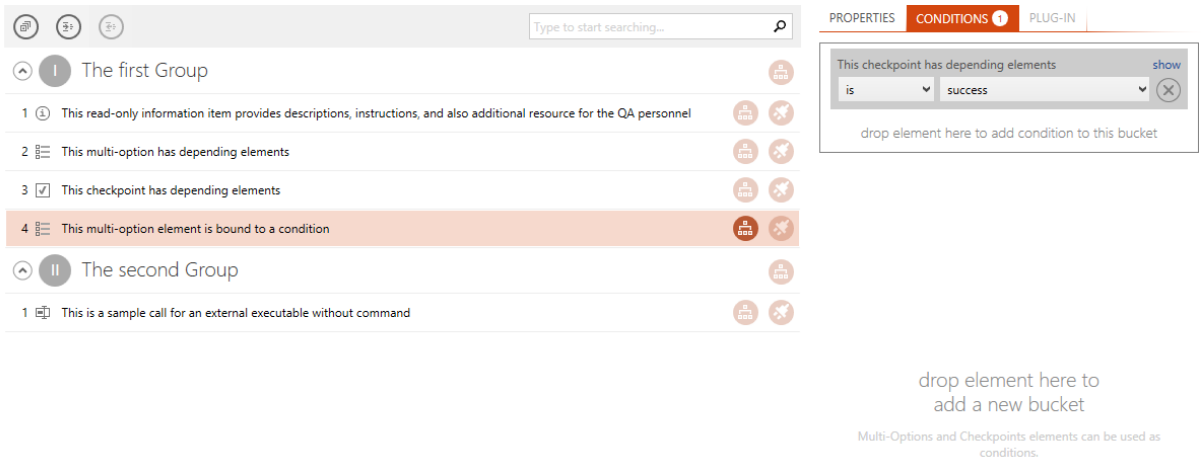
Note:
Removing the relation between a help file and an element does not remove the help file from the pool of supporting files stored within the checklist container. Removing a help file from the container has to be executed by the controls provided within the supporting files tab of the Checklist Editor interface.

Important aspects of dynamic checklist evaluations are conditional statements. In RayQC, users are able to define a combination of several conditional statements, which may decide about the actual evaluation path of a

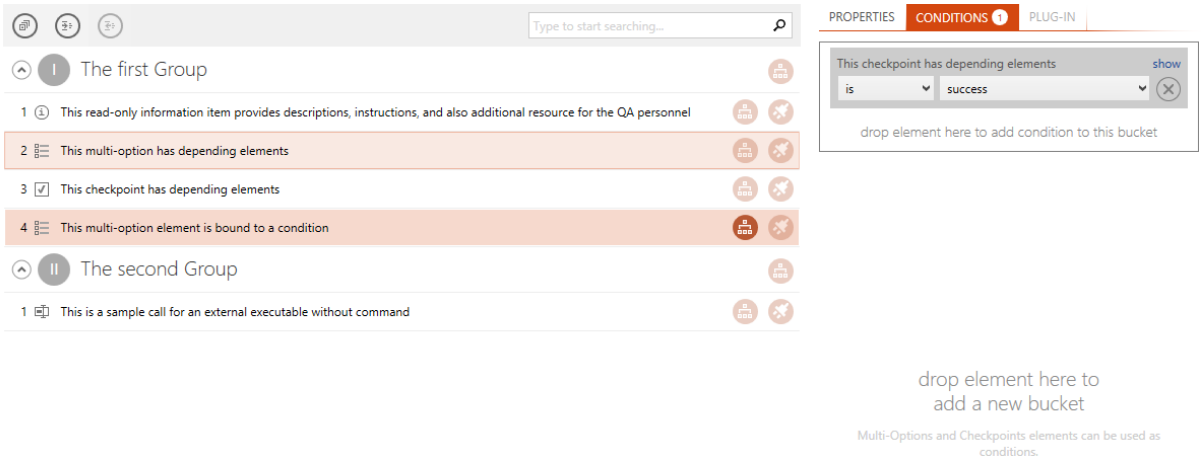
checklist. For example, if the result of Checkpoint A is **TRUE** and the result of Multi-Option B is **FALSE**, checklist group X has to be evaluated. If not, checklist group X is invisible and does not affect the result of the checklist. **Conditions** may be applied to single elements as well as to complete group objects.

The logical idea of **Conditions** in RayQC 6.1 is based upon the so called "disjunctive normal form" (DNF), which allows building any kind of condition as a combination of **OR**s between **AND**s. To be more precise: A DNF is a disjunction of conjunctive clauses. Within our checklist editor, clauses are called buckets. Within each bucket, there may be several conditional statements, but all have to evaluate to **TRUE** in order to let the bucket evaluation result become **TRUE**. If the condition for an element contains more than one of those buckets, it is sufficient to have one bucket evaluate as **TRUE** to let the whole condition evaluate as **TRUE**. So you see, the buckets contain a number of **AND**s, and are chained by **OR**s.

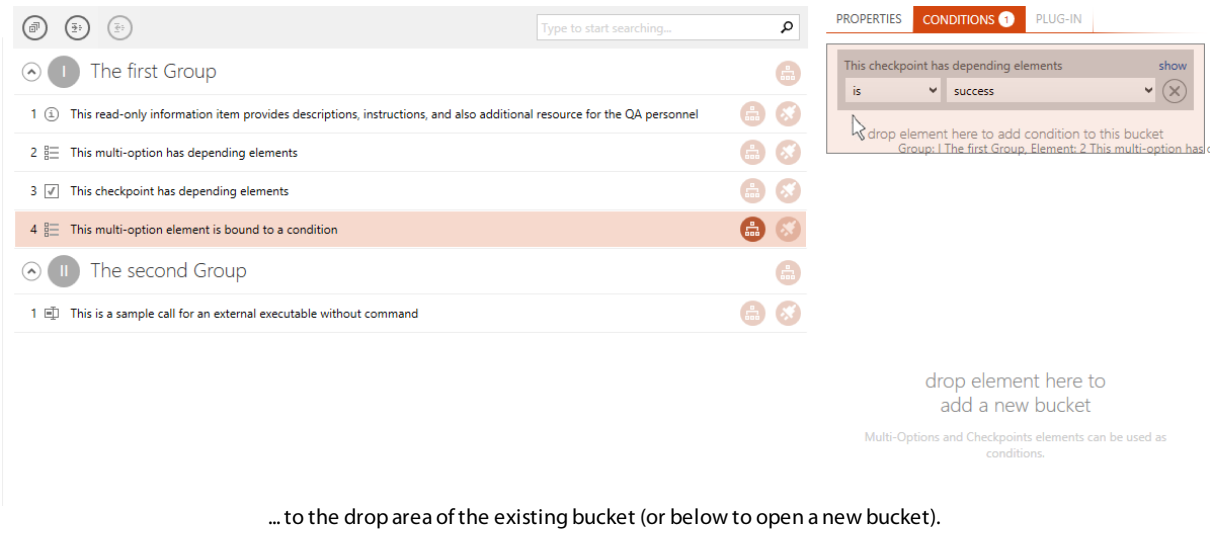
Handling conditions via the RayQC Checklist Editor user interface has been adjusted to follow the latest RaySuite interface guidelines: Adding a condition to an element is achieved by a combination of drag and drop operations, followed by selections from predefined drop-down controls:



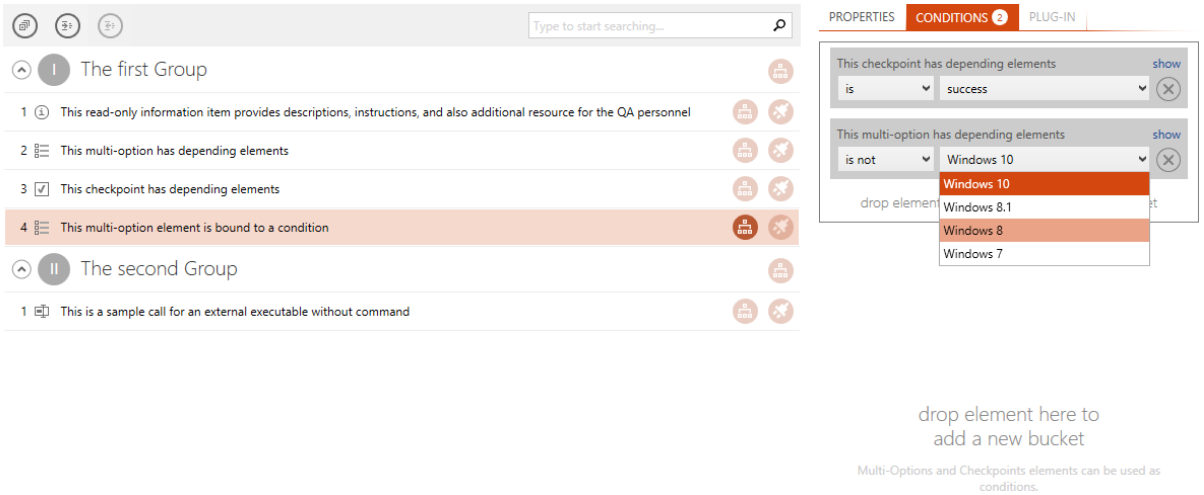
A Multi-Option element with an already existing condition, containing one bucket with one conditional statement that needs to be extended.



To add another clause to the condition, the decisive element has to be dragged...



... to the drop area of the existing bucket (or below to open a new bucket).



At the end the user just needs to select the expected result for a true conditional clause evaluation from the automatically added bucket item.

Conditions specified for a certain item decide, whether or not the item itself is available for user interaction in the Checklist Viewer interface. The results of unavailable items will not be considered when the overall checklist result is calculated.

The number of conditional statements is limited by logical restrictions **Conditions** have to stand up against:

- A **Condition** controls the availability of the checklist item (or group) that contains the conditional statements..
  (It is not possible to define a **Condition** in item A that controls the availability of item B or group C.)

- A conditional statement can only target checklist item results.
  (It is not possible to define the availability of an element dependent on the availability of another element or a group.)

- A conditional statement targets exactly one Checkpoint or Multi-Option element result value.
  (It is not possible to check against the textual result of Data Field elements.)

Hint:
To check against the textual result of Data Field elements, use functions like: `ContainsString`, `CompareValue`, or `NumberInRange`.

- A conditional statement can only refer to results of items that are listed higher within the checklist item sequence (e. g. in a prior group or with a higher position within the same group as the item that includes the conditional statement definition).
  Therefore, the first item within a checklist may not be bound to **Conditions**, since there are no prior item results to validate against.

- A **Condition** is evaluated to be either true or false.
  If it evaluates to true, the item (or group) is automatically available for processing within the Checklist Viewer.

- A **Condition** is by default evaluated to true if all of the conditional statements evaluate true.
  (Item evaluation is by default executed in conjunction mode (i.e. statements are combined by logical AND operators.)
  It is possible to change the default assessment method to disjunction mode, i. e. conditional statements are combined by logical **OR** operators, please see *below*.

- A **Condition** always controls the availability of the owning object along with all direct children of that object (e.g., all elements and sub-elements of an unavailable group are unavailable as well).

Be aware:
When **Conditions** are heavily used, it may happen that a user loses track of the actual evaluation path. Since it is possible to add conditions regarding the result of conditional objects, it is actually possible to create conditional statements that may never result in the availability of a certain group or element. Therefore, it is highly recommended to double-check conditional constructs, and additionally use the **Validate Conditions** button from the **Swipe bar**, which is available in the Checklist Viewer. If the check returns issues, these should be cleared before the checklist is deployed for productive use.

Note:
The items stored within a checklist group are indexed. If an item is not available for user interaction, its index number is missing from the present checklist number sequence. This choice of checklist interface design leads to the fact that users may very well notice that there are items which are not visible or accessible for them. However, providing a stable index reference for each checklist item, independent from the current availability for evaluation, is considered more important from an organizational / communicative point of view, than the absence of any indicators regarding conditional items.

To activate this option

1. Within the **Checklist Editor**, scroll to the checklist item that should only be available under certain circumstances.

2. Select the element with a left-click and select the **Conditions** tab on the **Details** pane.

Be aware:
The **Conditions** tab itself is permanently displayed for all checklist items and groups. However, if there is no checklist element present that might actually be evaluated for conditional statement definitions, users are simply unable to drop items into it.

3. Drag any Checkpoint or Multi-Option element that resides earlier within the checklist structure to the **Conditions** tab, and drop it there. If it is the first drop, a new bucket is created automatically. If one or more buckets for conditional phrases are already present, the new one will either be added to one of the existing buckets, or wrapped into a new one - dependent on the exact position of element dropping. Please observe the hover effect shown by RayQC as soon as an element is dragged into the Details pane area to get info about the target object: new or existing bucket.

   Please keep in mind, that the conditional statements within one bucket are evaluated in an **AND** relation: All statements have to evaluate to true in order to have the bucket evaluated to true. The buckets themselves are related by **OR** operators: If one of the buckets evaluates to true, the whole condition is regarded to be true, which allows the parent element (or group) to be seen (and evaluated) by users within the checklist viewer.

Be aware:
Checkpoint elements may have an active **Expected Value** option, setting the "correct" result to **NO**. In this case the conditional statement result option **TRUE** resembles the checkpoint entry result choice **NO** (and vice versa for result option **FALSE** and result choice **YES**).

4. Save the changes made to the checklist and switch to the Checklist Viewer mode.

5. Use the **Validate Conditions** button from the Swipe bar to make sure no invalid conditional statement combination has been created.

   If there are issues returned from the check, click on the **MORE** button within the info dialog to get details regarding the incorrect statements. Resolve the issues and re-run the check until no further issues are reported.

6. Check whether the item becomes available as the expected combination(s) of evaluation results are selected.

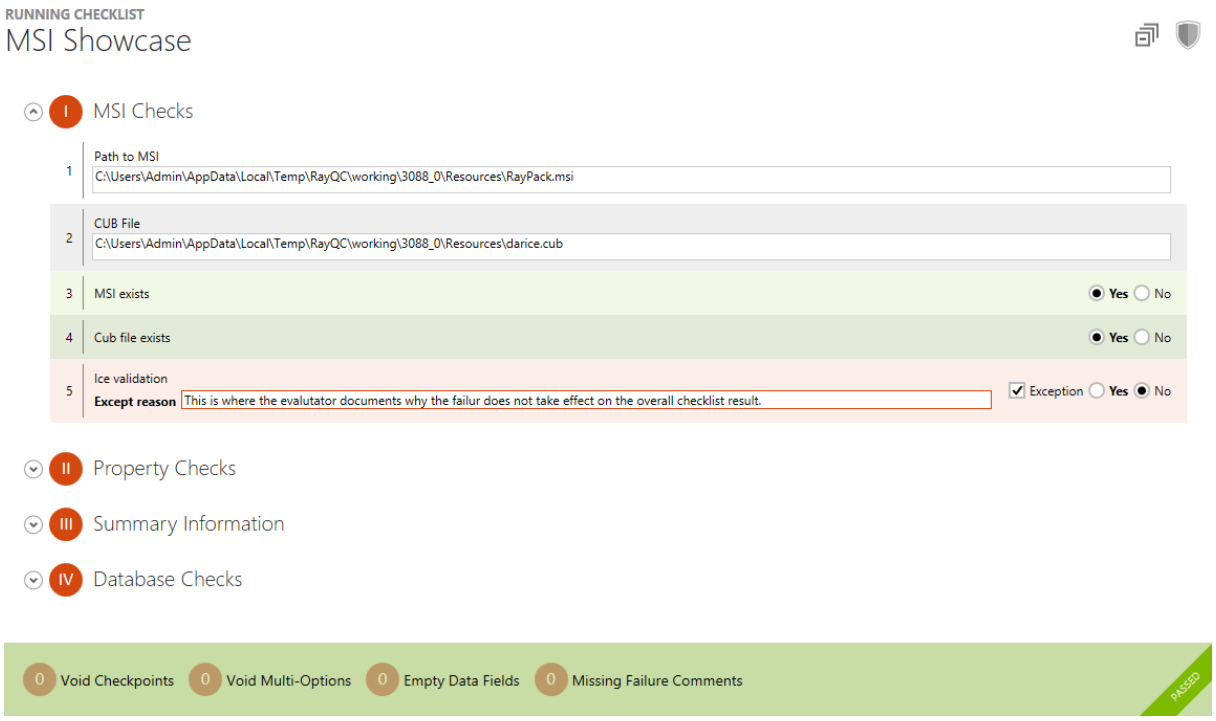
When each of the items present within a checklist is considered to be a rule to check, **Exceptions** are the circumstances under which breaking the rule does not really matter. **Exceptions** can only be added to Checkpoint elements, since they are the most strict checklist elements, and therefore most likely to step into an exceptional result state.

When a checklist editor enables the **Exception** option for a checkpoint, he allows the evaluator to document the actual test result as is, but define it to be uncritical towards the checklist result calculation. For example, when the expected result for a checkpoint is **Yes**, but the evaluator selects **No**, the whole checklist will be marked as failed. Even though a failure reason will be given by the evaluator, this functionality would not suffice for those cases, in which the failure reason is not related to the original test object, and therefore should not affect the overall test result.

Note:
It is not possible to activate the exception option and when the **Evaluate this element** option for an item is not enabled.

To activate this option

1. Within the Checklist Editor, scroll to the Checkpoint element that has to be equipped with an exceptional result state.
2. Open the **Properties** tab within the **Details** pane at the right-hand side.
3. Activate the **Allow exceptions** checkbox.
4. Save the changes made to the checklist and switch to the Checklist Viewer mode.
5. Set all checklist item results to evaluate to true to set the checklist result to **PASSED**.
6. Set the result state of the manipulated Checkpoint element to the result that resembles a failure.
7. The checklist item is marked as failed and the exception checkbox becomes available for activation. A failure reason is expected to be entered. The current checklist result is **NOT FINISHED** (since the failure reason is missing).
8. Activate the **Exception** checkbox. A text input field becomes visible, expecting an **Exception reason** from the evaluator. The failure reason is no longer required.
9. Enter an **Exception reason**. The checklist result switches to **PASSED**.



The screenshot shows a checklist in the state that has been outlined in the exception activation procedure above. Even though element no 5 of the first group is marked with a failed check result, the exception allows a positive overall checklist result as soon as the reason is entered.

The **Expected Value** option is only available for Checkpoint elements. Such elements can have only two result states: **Yes** and **No**, where **Yes** is by default the expected result required to be able to achieve an overall checklist result of **PASSED**.

Sometimes it may be tricky to set the textual description of a Checkpoint to actually match this behavior. To avoid misunderstandings due to complex test descriptions, RayQC has been extended with the **Expected Value**, which defines **No** to be the expected, correct answer.

When an evaluator uses a checklist, the expected result that evaluates to true is marked with a bold font color. So, by default **Yes** is marked bold. If **Expected value** is set to **No** then - **No** is marked bold.

To activate this option

1. Within the Checklist Editor, scroll to the Checkpoint that has to be equipped with a switched value expectation.

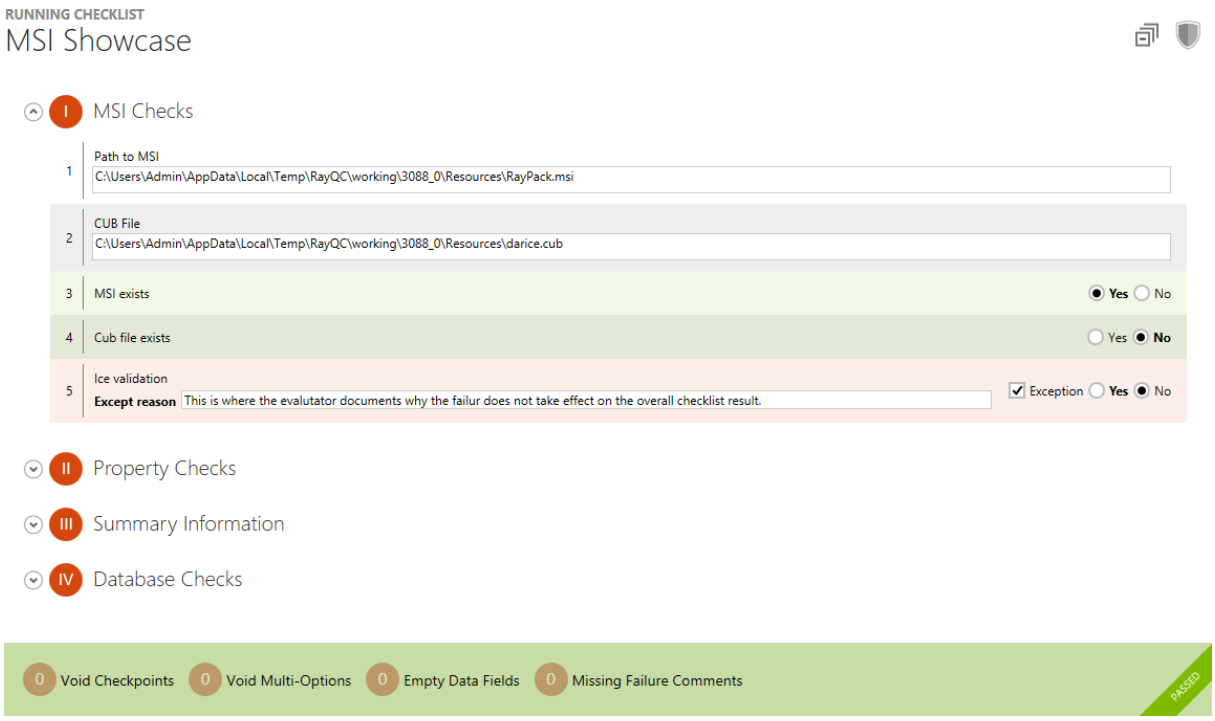
2. Select the Checkpoint element with a left-click, and call its **Properties** tab within the **Details** pane at the right-hand side.

3. Switch the current expected value setting from **Yes** (default) to **No**.

4. Save the changes made to the checklist and switch to the Checklist Viewer mode.

5. Set all checklist item results to evaluate to true to set the checklist result to **PASSED**.

6. Set the manipulated Checkpoint result to the not bolded result option **Yes**. The item box background turns from green to orange, and the overall checklist result switches to **FAILED**.



The screenshot above shows a checklist with two checkpoint entries:

- Item number 3 shows the standard behavior: **Yes** is the expected value which is displayed in a bold font color. If it is selected by the evaluator, the item background switches to green and the item result is considered true for the overall checklist result calculation.

- Item number 4 shows the value switch behavior: **No** is the expected value which is displayed in a bold font color. When **Yes** is selected by the evaluator, the item background switches to pink and the item result is considered false for the overall checklist result calculation.

Note:
Changing the expected value to **No** may affect **Conditions** defined for the result of the Checkpoint element, since it changes the correct value for the item, which is what a condition asks for. Please review affected conditions after changing value switch settings!

RayQC uses the result of Checkpoint and Data Field element types to calculate the result state of the whole checklist. Therefore, if a Data Field is missing, or a Checkpoint result has not been inserted yet, a checklist is not complete, and thus cannot be **PASSED**.

In order to set user input optional for elements without actual influence for the checklist result, for example for items which collect additional meta-data, there are two item properties:

- **Evaluate this element** for Checkpoint elements

  This option is active by default, which leads to the standard behavior of using the Checkpoint result for the overall checklist result calculation.
  Deactivating this setting leads to the removal of any result expectation for the item. Therefore, in the Checklist Viewer interface, there is no bold font style for both result options.

  Additionally, optional Checkpoint results may still be used to define conditions that control the availability of other checklist items. If no result is selected, the conditional statement cannot be evaluated, which means that the conditional element will not be displayed at all. If a user selection is present, the conditional statement is evaluated according to the standard condition execution ruleset.

Note:
It is not possible to activate the exception option when the **Evaluate this element** option for an item is not enabled.

- **Allow this Data Field to be empty** for Data Field elements

  This setting has no visual effect on the items box within the Checklist Viewer interface. However, if it is activated for a Data Field element, the checklist evaluation can be terminated without providing any result content for the element.

Both are handled the very same way: The irrelevant checklist item is displayed within the general checklist flow, and users may very well specify a result. However, if an optional value is not set, the checklist can be finished, and therefore result in **PASSED** or **FAILED**.

To activate this option

1. Within the Checklist Editor, scroll to the item that has to be defined as optional.

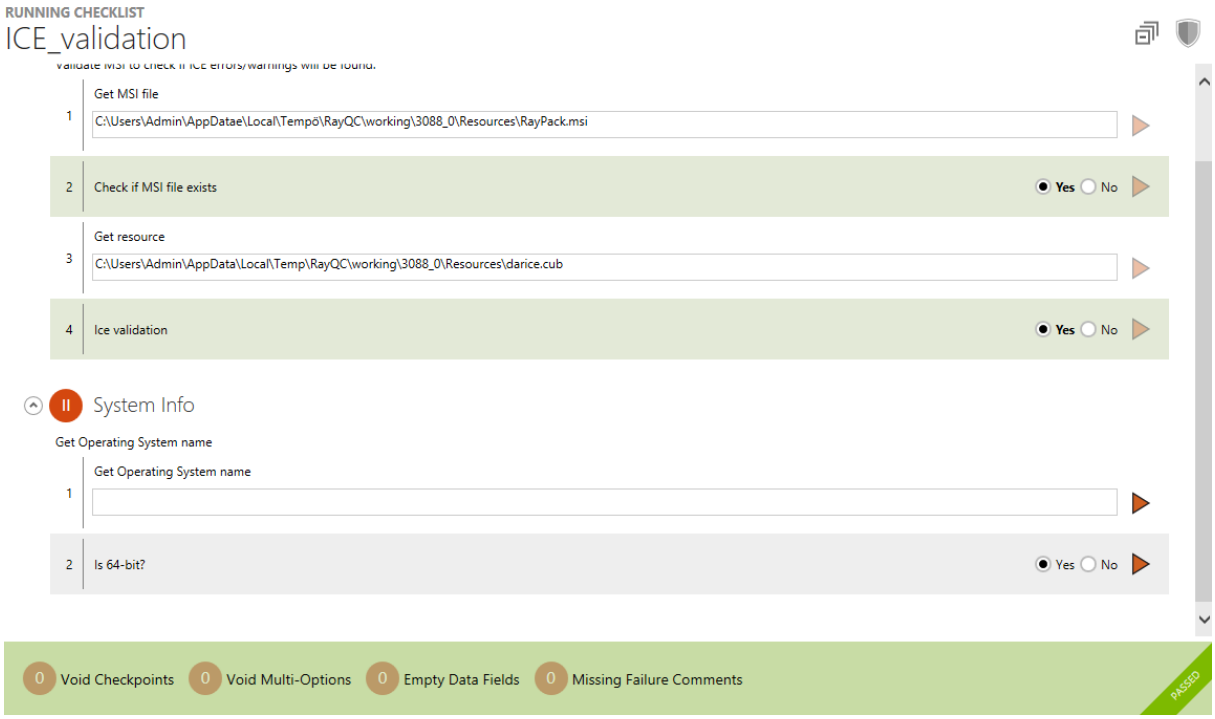
2. Select the element with a left-click, and open its properties tab from the **Details** pane at the right-hand side.

3. Activate the **Allow this Data Field to be empty** checkbox

4. Save the changes made to the checklist and switch to the Checklist Viewer mode.

5. Leave the manipulated item untouched, and set all other checklist item results to evaluate to true. The checklist result is automatically set to **PASSED** even though the optional item has not been answered.

6. Enter a result (text or **Yes** / **No** decision) for the manipulated item. Observe the unchanged checklist result state.



The screenshot above shows a checklist with two optional items:

Number 2 in the second group is an optional Checkpoint element, number 1 is an optional Data Field. Both are not answered and yet the checklist result **PASSED** is available.

Conditions are one appropriate tool to add dynamic aspects to a checklist. The other tool is the open plug-in interface RayQC provides for all element types. A plug-in can read fields that are already filled, process these values and fill other fields with data. The effort involved in filling a checklist can actually be reduced significantly in this manner.

Tip:
This section deals with the plug-in options present for element configuration within the Checklist Editor interface and the effects plug-ins have on checklist projects within the Checklist Viewer.
Please refer to the plug-ins section for further details regarding the available types of internal standard plug-ins, required files and definitions for the integration of external (custom) plug-ins, and tips on how plug-ins may be combined for advanced checklist logic implementations.

To activate this option

1. Within the Checklist Editor, scroll to the element that has to be equipped with a plug-in.

2. Select the element with a left-click, and open its plug-in tab from the Details pane at the right-hand side.

   Please note that Information elements may not be extended by plug-in functions, and therefore do not have a plug-in tab within their Details pane.

3. Browse the plug-in list displayed within the toolbar for the desired function. Use the search field if required. Drag the desired plug-in function to the plug-in tab of the details pane displayed for the currently selected element. (As an alternative, plug-in functions may as well be dropped on an elements that is already present within the Checklist structure area in the center of the Checklist Editor interface. However, dropping a plug-in function there still requires to opening the plug-ins tab of the affected element in order to be able to define the required parameter settings of the plug-in function.)

   Please note that every plug-in function has a defined set of valid parent elements. If a function cannot be used in combination with a specific element type, RayQC prevents users from dropping the plug-in function onto this element type. The internal plug-in documentation contains information about valid function - element type combinations. External plug-ins have to contain this information as part of their manifest file.

   Another important fact about plug-in usage is that each element may only be combined with a maximum of 1 plug-in function. Therefore, if a user tries to drop a plug-in function on an element that already has a plug-in usage, RayQC displays a dialog, asking the user what to do: Replace the existing plug-in function for that element, or abort adding the new plug-in function relation.

   Plug-in usage modifications may take serious effect on the logical flow of checklists. Just imagine that in scenario A a specific element converts a boolean result (e.g. by using the InvertBoolean function of the Logic plug-in), whilst this inversion is missing in scenario B: This small difference may cause conditional dependencies and references established by element result usage for function parameter definitions to be evaluated totally different, which in turn may lead to a dramatically changed evaluation flow. Well, it is highly recommended to double-check the outcome of plug-in replacement in order to prevent undesired side-effects.

4. As soon as a plug-in has been added to an element, RayQC displays input controls for the parameters needed to successfully execute the plug-in function.
   Since each plug-in comes with an individual set of parameters, a full list cannot be provided in this section. It is highly recommended to read the plug-in section for details regarding the internal plug-ins delivered with the RayQC installer resources, and review the manifest file or contact the author of external plug-ins for a more detailed technical documentation of those plug-ins.

5. Now it is time to save the checklist template changes, and test them by walking through the steps of the checklist within the Checklist Viewer.

6. Make sure to set all element results as required to provide all parameter values and element / group availability status settings needed to make the element with the newly created plug-in available and fully prepared.

7. A rightwards pointing arrow icon should be available within the element checkbox displayed within the Checklist Viewer interface. Clicking that icon triggers the plug-in execution.

8. Make sure that the plug-in logic is executed as expected. If it seems due, prepare different parameter sets to determine successful and failing executions are covered with decent checklist reactions (e. g. error messages displayed within comment items, dynamic checklist branch availability, etc.). The result returned by the plug-in function execution is by default stored as the result of the element that hosts the plug-in function.

However, due to type transitions, element options and other affecting settings, the result that is actually displayed within the Checklist Viewer interface may very well differ from the original return value of the plug-in function. This has to be kept in mind whilst testing and debugging the different function scenarios.

Elements of the Data Field type have the ability to read data from the connected RayFlow server via the RayFlow plug-in interface. Each property of a specific RayFlow workflow data object (e. g. package orders) may be requested and used within the data field item text.

Since communication with RayFlow is established via web service and always relates to a specific data object within the RayFlow database, there are some prerequisites that have to be given in order to use the RayFlow parameter option:

- Connection settings need to be given. These may either be defined globally via the RayQC settings view connections, or locally for projects that have been opened via the tool integration of RayFlow, and equipped with parameters containing the connection credentials.
- A RayFlow workflow data object identifier must be given. This unique ID may either have been injected directly from RayFlow as a parameter provided by the RayQC tool integration command, or may be derived from manually or automatically filled in content from another data field item.
- The attribute name of the RayFlow property must be known.

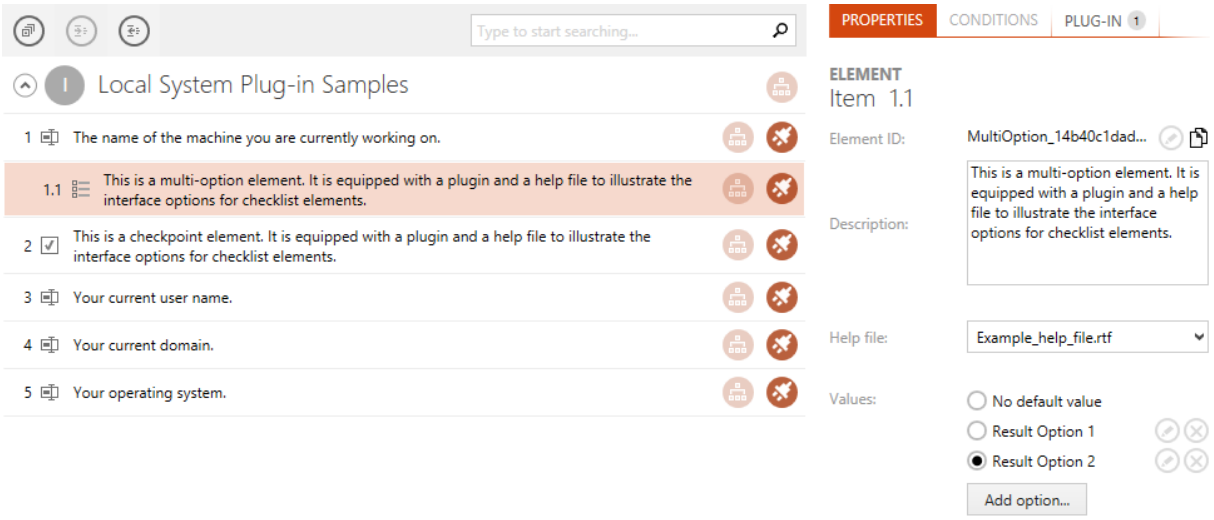
Note:
The set of actually available values for retrieval from RayFlow depends on the individual object structure defined for the connected RayFlow instance. Please contact your RaySuite system administrator or refer to your RaySuite instance documentation for details regarding object definitions and the interface provided for external communication.

Note:
For further information on options provided by the RayFlow plug-in, please refer to the RayFlow plug-in section of this document.

Element Controls

Additionally to the options users may set for the internal properties of checklist items, there are some controls that help to organize the elements in relation to the whole checklist. The Checklist Editor interface provides the following controls to manipulate the item set of a checklist:

- *Add an element*
- *Delete an element*
- *Move an element up or down*
- *Increment or decrement the indentation level of an element*



The screenshot above shows an element box with some active control options. Since the item is a multi-option type element, only the type specific controls are displayed. The first line of the item box contains information and controls about the item position within the checklist group.

Since not only elements, but also groups may be used to structure a checklist, some additional information is provided for global group management as well:

- *Add a group*
- *Delete a group*

Please read the following sections to gain in-depth knowledge on how these controls are used, and when their usage may be handy for the provision of decent checklist designs.

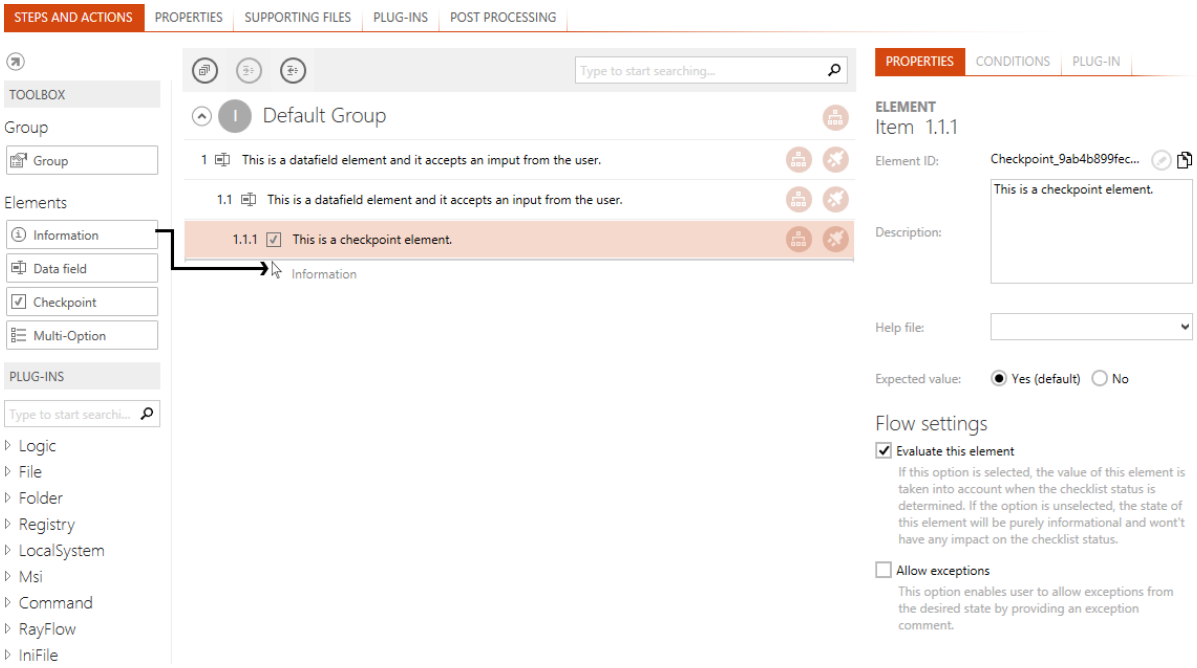
Once a checklist is opened within RayQC and the Checklist Editor mode is active, it is possible to add new elements to any of the existing groups of the checklist.

To add an element

1. Scroll the checklist area to display the desired target position of the new checklist item.
2. If the target group is collapsed, *expand* it to reveal the items already bundled inside of it.
3. Move the mouse pointer to the Toolbar area on the right and click on the item type that should be added.
4. Keep the left mouse key pressed and drag the item to the group container area at the left. An indicator bar is shown, pointing to the expected position of the new item. If no indicator bar is shown, the current mouse pointer position is not a valid target area.
5. As soon as the desired position is marked, drop the item by releasing the left mouse key. The group content is updated with the new item. The index values of all later items within the group are updated to match the new group content collection.

If the new element has been dropped at the first position within a group, it is automatically added on the highest nesting level, indicated by the index value of 1 set for the new item.
If the new element has been dropped below an existing item, it is by default added as the next item in the checklist. For example, if the element above the new one has an index value of 1, the new element becomes

the next element with index value 2.
In case the element is dropped on a child element then the element becomes the child element of the parent element. E.g. If the parent element has an index value of 1 and it has a child element with index value 1.1, then when an element is dropped on 1.1, it will get the next index value that is 1.2.
The maximum depth of a checklist element hierarchy is 4, therefore it is not possible to drop elements to a deeper level than the fourth.



The screenshot above shows the drag path (dark gray line) of a new information item. If the user drops the new item right now, it is added below the item number 1.1.1, and 1.1.2 will be the designated item number.

In order to adjust the horizontal position of an element, the controls *left* and *right* have to be used. Furthermore, a user can change the vertical position of an element by dragging and dropping the element as described previously.

As soon as an element has been added to the group, its *basic properties* and *type* specific *controls* are available for manipulation.

Note:
Please make sure to save the changes made to the checklist structure to keep them permanently. If the changes are not saved, closing the checklist file leads to the loss of all new settings!

Once a checklist is opened within RayQC and the Checklist Editor mode is active, it is possible to remove elements from any of the existing groups of the checklist.
A group has to have at least one element, thus deleting the last element of a group is not possible. If the whole group is obsolete, directly deleting the group object from the checklist is the recommended procedure.

Be aware:
Deleting an element simultaneously deletes all sub-elements that have been nested below the element, as well as all conditions pointing to the element.

To remove an element

1. Scroll the checklist area to display the group that contains the item that is about to be deleted.

2. If the target group is collapsed, expand it to reveal its items.

3. There are 2 options to delete the item:
   - Right-click on the item and select Delete from the context menu
   - Left-click on the item and press the Delete key on your keyboard

4. Within the confirmation dialog, review the list of affected conditions and sub-elements to make sure the item deletion does not cause unexpected side-effects.

5. Click on "YES, REMOVE" to delete the item, or click on "DO NOT REMOVE" or CANCEL to abort the deletion.

   Clicking "Yes, remove" automatically closes the confirmation dialog, removes the selected item from the checklist group and updates both items with conditions pointing to the deleted element as well as the index values of all items within the same group but with a later position.

Note:
Please make sure to save the changes made to the checklist structure to keep them permanently. If the changes are not saved, closing the checklist file leads to the loss of all new settings!

Building a decent checklist structure is not necessarily done with the first attempt. It is far more likely that elements need to be adjusted regarding their position within checklist groups. Therefore, RayQC offers simple drag-and-drop option to move an element (and all its sub-elements) up or down in the checklist.

Be aware:
Moving elements upwards may take effect on the conditions defined for the item. Since conditions always have to be defined for items that are positioned higher than the item they are defined for, moving an item up might cause invalid conditional structures, and is therefore suppressed by the Checklist Editor interface logic.

Whilst checklist groups are aligned in a flat sequence without the possibility to build hierarchical tree structures, elements within a group may very well be arranged in trees to visually support dependencies and relations between the items.

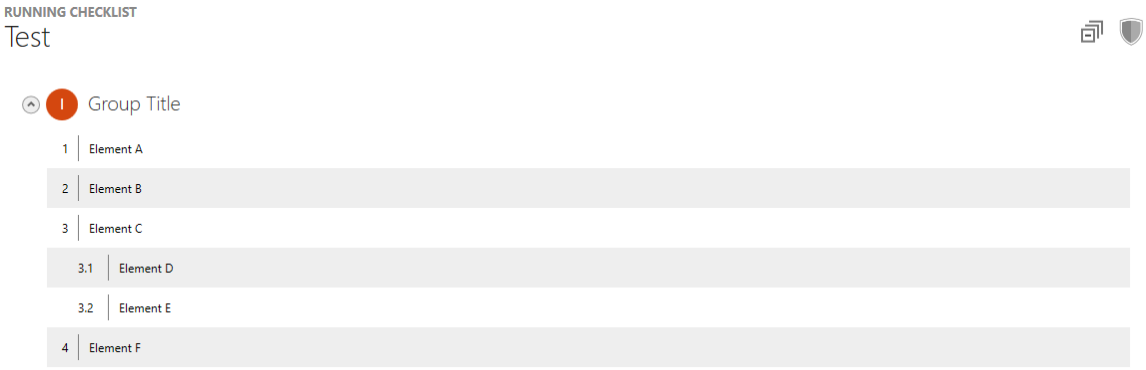
Changing the indent of an element is therefore always an activity that has to be viewed relative towards the element neighbors:
Incrementing the indent of an element turns the element (and all its children) into a subelement of the item at the next higher position on the same tree level.

An example

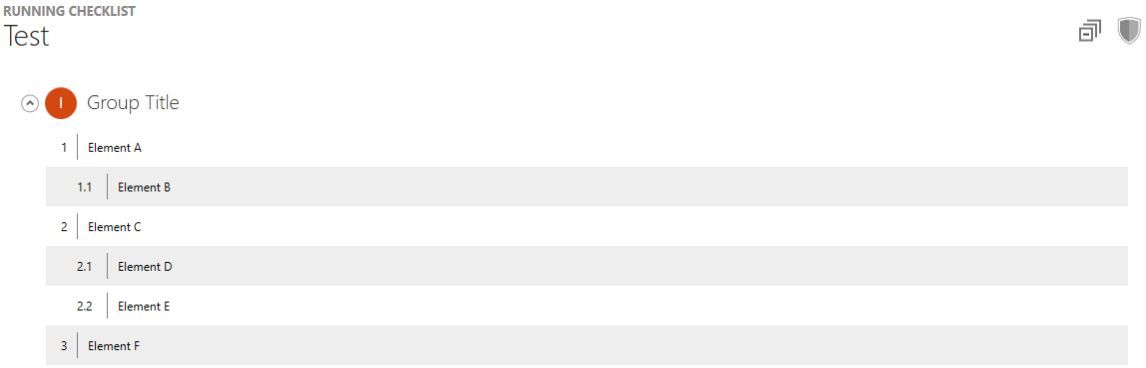
A group contains the items A at index position 1, B at index position 2, and C at index position 3. Incrementing the indent of item B leads to the following new structure:

- Item A stays at position 1.
- Item B is now a subelement of item A, and therefore has the new index value of 1.1.

- Item C is moved one position up, since the index position 2 became vacant when B was reorganized.
  If item C has subelements, their root index value is decremented by one, whilst all other levels stay as they are. (e. g. 3.1 would now be 2.1; 3.4.5 would now be 2.4.5, and so on).



Initial element hierarchy



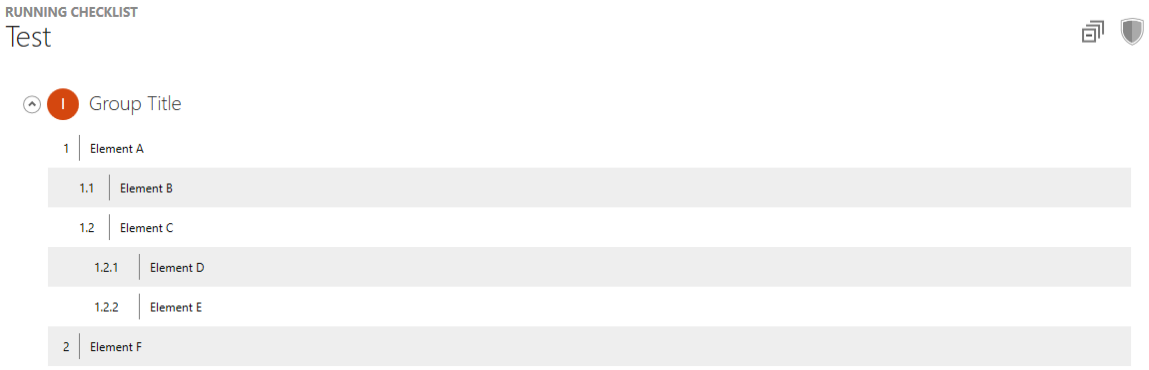
Updated hierarchy after incrementing indent of item B

Decrementing the indent of an element turns the element (and all its successors) into a subelement of the item at the next higher position on the same tree level.

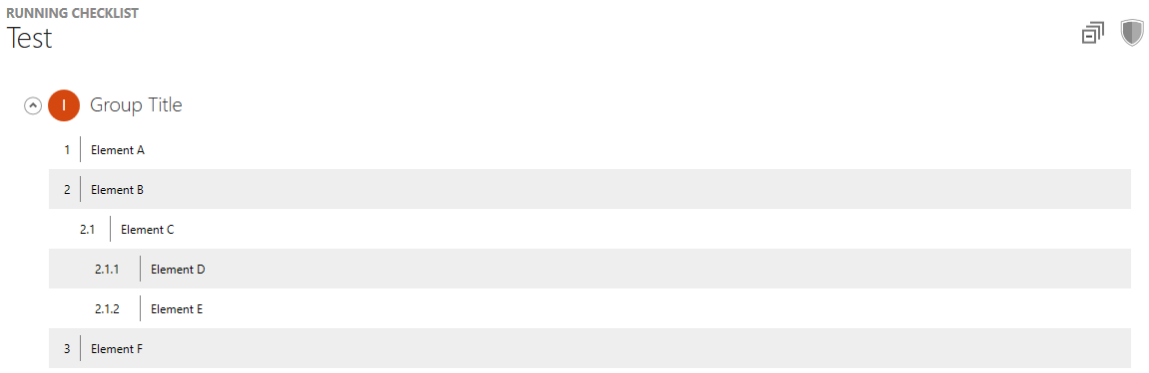
Another example

A group contains the items A at index position 1, B at index position 1.1, and C at index position 1.2. Decrementing the indent of item B leads to the following new structure:

- Item A stays at position 1.
- Item B is now a root element on the same level as A, and therefore has the new index value of 2.
- Item C is no longer a subelement of A, but of B and is moved to index position 2.1.



Initial element hierarchy



Updated hierarchy after decrementing indent of item B

As outlined in the example above, whilst incrementing the indent does not break the current nesting of child elements, decrementing the indent may very well do so. Therefore, it is highly recommended to double check the actual effects. Due to that fact, indent manipulations are always executed level by level. This way it is possible to review the actual results of an indent manipulation step before the next change is executed.

Be aware:
It is not possible to ...

- ... increment the indent of a leaf item.
- ... increment the indent of an item beyond the 4th level.
- ... decrement the indent of a root item.

To increment (decrement) an elements indent

1. Scroll the checklist area to display the group that contains the item that is about to be moved.
2. If the target group is collapsed, *expand* it to reveal its items.
3. Click on the right or left pointing arrow icon at the right-hand side of the box that represents the element that you want to move.
4. The elements of the group are immediately reorganized to match the new hierarchy.

Once a checklist is opened within RayQC and the Checklist Editor mode is active, it is possible to add new groups to the checklist area.

To add a group

1. Scroll the checklist area to display the desired target position of the new group object.

2. Move the mouse pointer to the Toolbar area on the right, and click on the group icon.

3. Keep the left mouse key pressed and drag the new group to the checklist area in the middle. An indicator bar is shown, pointing to the expected position of the new group. If no indicator bar is shown, the current mouse pointer position is not a valid target area.

4. As soon as the desired position is marked, drop the group by releasing the left mouse key. The checklist content is updated with the new group. The index values of all later groups are updated to match the new container collection.

Once a checklist is opened within RayQC and the Checklist Editor mode is active, it is possible to remove groups from the checklist sequence.
A checklist has to have at least one group, thus deleting the last group from a checklist is not possible.

Be aware:
Deleting a group simultaneously deletes all elements that have been nested inside the group, as well as all conditions pointing to these elements.

To remove a group

1. Scroll the checklist area to display the group that contains the item that is about to be deleted.

2. If the target group is collapsed, expand it to reveal its controls.

3. There are 2 options to delete the item:
   - Right-click on the item and select Delete from the context menu
   - Left-click on the item and press the Delete key on your keyboard

4. Within the confirmation dialog, review the list of affected conditions and elements to make sure the group deletion does not cause unexpected side-effects.

5. Click on "YES, REMOVE" to delete the group, or click on "DO NOT REMOVE" or CANCEL to abort the deletion.

   Clicking "YES, REMOVE" automatically closes the confirmation dialog, removes the selected group from the checklist and updates both items with conditions pointing to now deleted group elements as well as the index values of all groups with a later position.

Note:
Please make sure to save the changes made to the checklist structure to keep them permanently. If the changes are not saved, closing the checklist file leads to the loss of all new settings!

Both RayQC checklist interface modes, Checklist Viewer and Checklist Editor, provide a context menu for elements, which is displayed when a user right-clicks an elements display box. Especially when a checklist editor tests and re-adjusts his creation, it saves a lot of annoying scrolling and source revision effort to use these direct links and functions.

The Element Context Menu within the Checklist Viewer

Reset Elements (Group)

This context menu option when clicked upon, resets all elements of the current group to the pre-execution state.. The option is initially disabled, when the checklist has not been executed.

Be aware:
Resetting an item may cause subsequent items to be reset as well, for example those who have conditional statements or result value dependencies related to the currently reset elements. Double-check the checklist status indicators after a reset action to get a hint regarding missing item results.

Run Plug-Ins (Group)

This option when selected by right clicking on a group, runs all the plug-ins which are associated with an element of the current group.

The Element Context Menu within the Checklist Editor

Indent

This context menu option indents the selected element to the right. This option is disabled for Group elements and for the elements which are already indented to the last possible position. For example if in a checklist there are 2 elements, item 1 and item 2, at index position of 1 and 2 respectively. It will only be possible to indent the second element once. Instead of using this option from the context menu, users can also indent an element by selecting it and then using shortcut `Ctrl+Right`.

Unindent

Contrary to the indent option, the Unindent option moves the selected element horizontally to the left by removing the last digit of the index value and increasing the second last by 1. For example: Element 1.1.4 becomes element 1.2 when it is unindented.

It is not possible to unindent a group element and an element which is already at its highest possible horizontal index value. For example an element at position 1 cannot be unindented.

This behavior can also be achieved by selecting an element and using the short cut key: `Ctrl+Left`.

Delete

As the name says, this option is used to delete an element from the checklist. Please note that a checklist must have at least one group and an element within that group. Hence it is not possible to delete the last group and the last element within that group from the checklist. A user can select an element and use the keyboard key `Del` to achieve the same.

Properties

The properties tab groups general checklist settings into one view. Properties defined here take effect on any project file saved from the template.

Checklist Title

The checklist title is saved directly within the checklist template XML structure. It is an arbitrary alphanumerical string, displayed within the application title bar whenever the checklist project is opened.

The default value for newly created checklist templates is simply "Checklist Title".

Tip:
The title cannot only be edited from this view, but from any Checklist Editor tab. The title is displayed above the actual tab content, and may be edited by a click on the edit icon at its right-hand side.

Checklist Description

The checklist description should provide information about the purpose of the checklist. What is actually evaluated by the elements? Is there special information or resource material required to evaluate the elements correctly?
Since the description is directed towards the checklist structure, it is saved within the overall XML structure of the checklist template.

It is possible to format the description text by using special markup tags. Please refer to the topic Formatting Markup Options for further details.

Report File Naming Format

The report file naming format determines the standard file name generated when users export project data from the internal RayQC `.rqcp` file type into `.docx`, `*.pdf`, or `.html`. Besides static strings, users have the option to use checklist specific variables within the naming format string.

The default format definition already contains the title variable. To define a variable within the format string, the variable keyword has to be encapsulated with a leading and trailing hash key (e. g. #title#).

As for now, the following list of variables is available for report file naming format definitions:

Variable	Content	Example
`#title#`	The checklist title	"Checklist title"
`#date#`	The current date (yyyy-mm-dd)	"2015-04-01"
`#status#`	The current project status (passed, failed or not finished)	"passed"
`#bypass#`	The bypass status of the checklist result (regular or bypassed)	"regular"

Reports generated from a project give a one time snapshot of a specific evaluation status for the underlying check routine. Therefore, the given format string is saved within the checklist header section of the underlying XML structure.

Tip:
Use the FILE button from the Main Toolbar above the Editor area, and select Create Report to generate a report file.

Allow Bypass

The bypass option allows evaluators to bypass the final result of a checklist run. In some exceptional cases it might be required to override the original evaluation result manually (e. g. define a failed evaluation run as passed, even though some partial results may differ from the expected results). RayQC offers a bypass option for this purpose within projects if the **Allow bypassing this list** checkbox has been activated for the checklist.

The legitimacy of a bypass usually depends on the overall scope of the checklist evaluation procedure. Therefore

the bypass option setting is not stored within the individual, project-specific part of the checklist, but as an attribute of the checklist header within the XML core structure itself.

Due to the same reasons, the **Bypass message** property is also saved as part of the checklist core structure. This advice is intended to give instructions regarding the circumstances that allow, or even demand, a bypass. The bypass message is a direct element of communication between checklist designer and checklist evaluator.

Please refer to the Checklist Viewer section for details on how to accomplish a result bypass for a specific checklist project evaluation result.

Supporting Files

What are supporting files?

Sometimes it is necessary to enrich checklist templates with more information than can easily be handled by simple textual descriptions for groups and elements. Supporting files are a decent way for checklist editors to add PDF or RTF documents as help content and PNG images as illustrations.
Whilst each checklist element may be equipped with a specific help file, images may be used freely within checklist, group and element descriptions by using the markup options for these properties.

The benefit of organizing supporting files in a separate dialog is reusability: Once a supporting file has been added to the checklist, it may be utilized as often as the checklist editor sees fit. This is especially handy in those cases, where the same supporting file has to be provided for several elements or groups with conditional availability during the actual checklist evaluation run. The file resources are stored once (directly within the checklist container), and referenced as often as required. By providing a freely usable pool of supporting files it is possible to keep the overall file size of RayQC checklist containers at a necessary minimum.

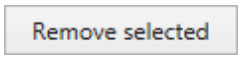
View organization

Since help files and images are stored separately within checklist template containers, they are managed by separate interfaces: the upper control element group within the Supporting Files view provides the controls to add and remove help files, whilst the lower group provides the very same functionality for images.

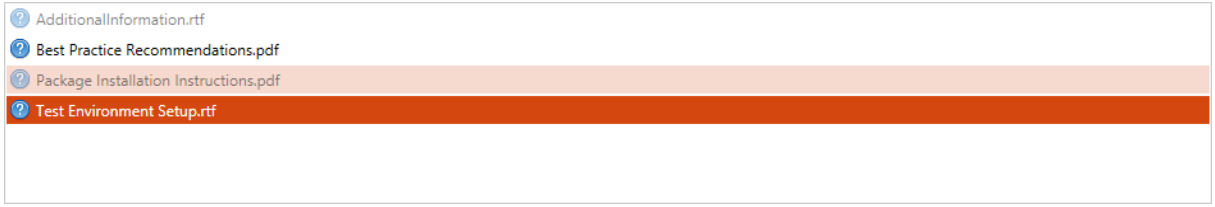
Each set of controls consists of

- A button to browse for a new file / image that has to be added

- A button to remove the currently selected file from the checklist container



- A list of currently available files



The list comes along with a special display style schema for the contained items:

- Grayed out icons symbolize items that have been added to the checklist, but are at present not used in relation to a specific element (for help files) or description field (for images). Additionally, they are marked with “NOT USED” at the right-hand side of the document.
- Fully solid icons symbolize an uploaded and used item.
- A light orange background and black font color are applied when the mouse pointer hovers over an item.
- A dark orange background and white font color are applied when an item is selected by a left-click.

To add supporting files to a checklist

Adding help files

1. Click on the **Add file** button above the list of already uploaded items

2. Within the displayed Open dialog window: **Browse the file system** for the desired file.
   Please note that the default file type filter is set to RTF. If a PDF has to be added, users have to switch the filter to display PDF files on the file system. Other file types are not supported.

3. **Select the file** with a left-click.
   Please be aware that a help file name has to be unique within the supporting files that are available for a checklist. Thus, it is not possible to add a file that has the same name as an already present help file item. The check for duplicate file names is not case sensitive.

4. Click on the **Open** button at the lower right corner of the dialog.

5. The dialog is closed and the file is added to the list of already uploaded help files.
   Since the file is not related to an element yet, it is displayed in the grayed out state.

Adding images

1. Click on the **Add image** button above the list of already uploaded items

2. Within the displayed Open dialog window: **Browse the file system** for the desired file.
   Please note that the default file type filter is set to PNG. Other file types are not supported.

3. **Select the file** with a left-click.
   Please be aware that a help file name has to be unique within the supporting files that are available for a checklist. Thus, it is not possible to add an image that has the same name as an already uploaded item. The check for duplicate image names is not case sensitive.

4. Click on the **Open** button at the lower right corner of the dialog.

5. The dialog is closed and the PNG is added to the list of already uploaded images.
   Since the image is not used within a description text yet, it is displayed in the grayed out state.

To remove supporting files from a checklist

1. **Select** the item within the list of help files or images
2. **Right-click** the item and select **Delete** from the **context menu**
   or
   Left-click the item and click on the **Remove selected** button above the item type list
3. The item is immediately removed from the supporting files pool.

Be aware:
It is not possible to add help files or images until they have been uploaded to the stock of supporting files.

To use supporting files within a checklist structure

Be aware:
It is not possible to add help files or images until they have been uploaded to the stock of supporting files.

Using help files

1. Open a checklist within the Checklist Editor interface.

2. Go to the Steps And Actions tab.

3. Select the element that has to be extended with the help file from the checklist structure.

4. Open the Properties tab of the element within the details pane.

5. Select the desired help file from the Help file drop-down selector.
   The checklist element display mode within the Checklist Viewer is immediately updated with a question mark icon.
   Clicking on the icon opens the related help file.

Using images

1. Open a checklist within the Checklist Editor interface.

2. Go to the Steps And Actions tab.

3. Select the group or element whose description text that has to be extended with the image from the checklist structure.

4. Open the Properties tab of the element within the details pane.

5. Use the markup for image integration to insert the illustration as an inline element to the description text. File names do not have to be entered case sensitive towards the original file name displayed within the supporting files pool.

   For example:
   An image with the name `DemoImage.png` has to be integrated into the description text "`Lorem ipsum dolor sit amet`" like this:
   `Lorem ipsum` **`[image]DemoImage.png[/image]`** `dolor sit amet`

6. The description text displayed within the Checklist Viewer and Editor is immediately updated to display the image at the defined position.

Plug-Ins

Local External Plug-Ins

Local External plug-ins add new functionality to the checklist processing by adding PowerShell based or DLL based plug-ins to RayQC.As compared to the external global plug-in, local external plug-ins are stored inside the checklist file and their functions are only available to the checklist containing it. After loading a checklist and opening the editor the functions of internal plug-ins may be found in the list of plug-ins in the toolbox on the left side.

Be Aware:
To load a plug-in RayQC must be licensed to use it. The Standard Edition of RayQC is not able to process PowerShell based plug-ins. The Enterprise Edition is needed to run these plug-ins.

For DLL based plug-ins, a valid plug-in license is needed. This can be obtained from Raynet.

Note:
For more information on handling external plug-ins and benefits/drawbacks of using external local and global plug-in, refer to the plug-ins section of the Settings chapter.

The plug-in Manager Dialog

Both local and global plug-ins use the same dialog for plug-in management. The plug-in manager for local plug-ins is located under the plug-ins tab of the checklist editor.

Be Aware:
Even when using the Standard Edition of RayQC the plug-in manager allows to add/remove PowerShell plug-ins. But keep in mind that such plug-ins can't be loaded by the Standard Edition and as such they won't be displayed in the toolbox. The same is true for non-licensed DLL based plug-ins..

Add a new PowerShell plug-in

1. Press the button labelled 'Add PowerShell plug-in'.

2. In the following dialog select the directory that contains the plug-in files ('manifest.xml' and all script files that are used by the plug-in).

Remove a PowerShell plug-in

1. Select the plug-in to remove in the list.

2. Click 'Remove selected'.

OR right click the plug-in to remove and select delete from the context menu.

Be Aware:
Local plug-ins that are used by a checklist that is currently loaded cannot be removed. Remove all usages of the plug-in to allow the deletion of the plug-in.

Add a new DLL plug-in

1. Press the button labelled 'Add DLL plug-in'.

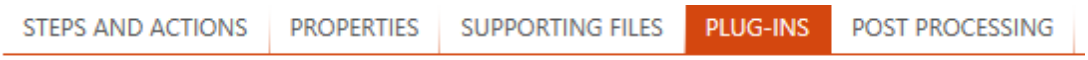
2. In the following file dialog select the plug-in DLL.

Remove a DLL plug-in

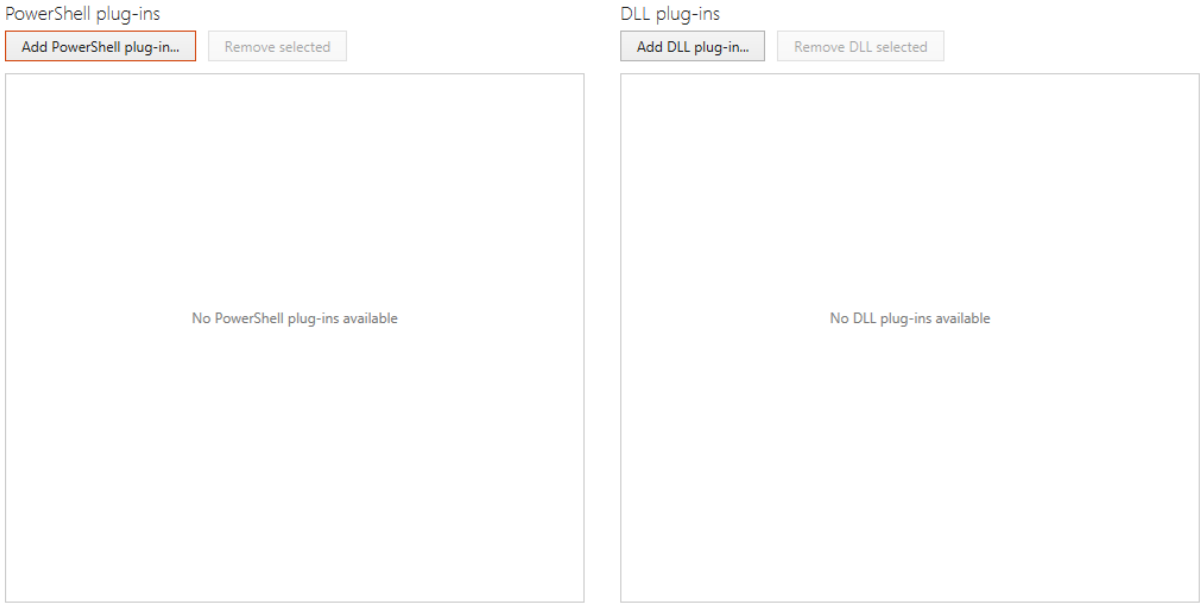
1. Select the plug-in to remove in the list.

2. Click **Remove selected**.

OR right click the plug-in to remove and select delete from the context menu.



Plugins tab within the Checklist Editor



The Plug-In Manager Dialog

Post Processing

What is post processing?

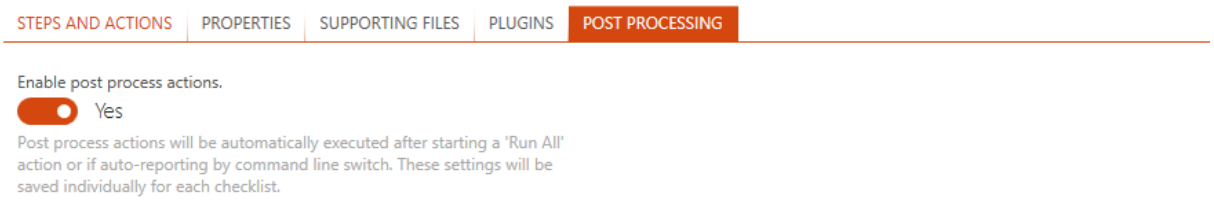
It is quite likely that RayQC is used as a tool that is integrated into RayFlow. This means that evaluations are commonly triggered from a RayFlow server. In order to provide bidirectional communication, there is not only a way to get data from RayFlow into RayQC, but also to return information (report files and updated values for RayFlow data fields) back to the RayFlow server. In order to standardize and automate this conversation as far as possible, users may define certain post processing tasks. If post processing itself is activated for a checklist, RayQC checks for condition fulfillment, and executes the post processing actions if one of the active conditions is meet.

Post processing may either be triggered manually by using the post processing button from the Swipe bar of the Checklist Viewer, or automatically as extension of a **Run All** procedure execution. The latter option is the required one for fully automated checklist evaluations. RayFlow triggers the checklist run including the automation parameter, and RayQC automatically responds with the information defined within the post processing section.
However, it is also handy for users to be able to upload checklist evaluation results to RayFlow themselves, since not all checklists may be fully automated. This is where the button for post processing execution kicks in.

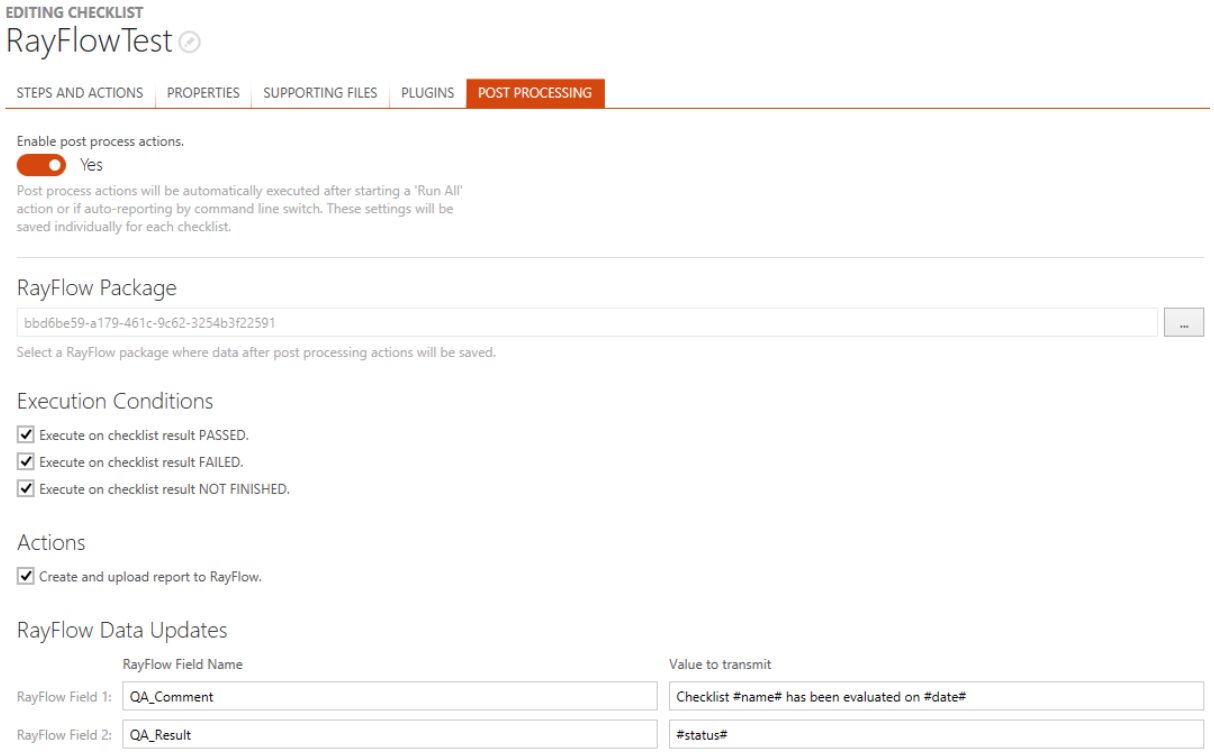
Be aware:
Post processing can only operate successfully, if the current evaluation session has a valid RayFlow connection. If no *parameter injection* is given and the user has not manually authenticated to RayFlow, there is no valid target for the data RayQC has to send. The result is an error message, which will be displayed if post processing fails due to missing connectivity.

To enable or disable Post Processing options for a checklist

The default state of the post processing options, which is given for example when a user calls the **POST PROCESSING** tab for a specific checklist for the first time, is disabled.



To enable post processing, users have to activate the **Enable post process actions**. Further adjustments are required to fine-tune what has to be executed and under which circumstances RayQC executes it. Setting **Enable post process actions** to **No** does not delete the settings defined so far, but simply deactivates the execution of the defined actions and hides the post processing option controls. It does not matter how often a user enables and disables the post processing, the latest settings are preserved and become editable with every post processing enabling.



Post Processing Options

If post processing is enabled, users may decide about the following settings:

RayFlow Package

Stores an ID of a RayFlow package to which data from post processing actions will be saved. After clicking on the

browse button [...] users will be able to select a package using the common RayFlow pop-up window.

Execution Conditions

As outlined above, there are several triggers that may cause RayQC to initiate post processing. The first step of post processing is to check whether the conditions for further activity execution are meet:

- Execute on checklist result **PASSED**
  If a checklist evaluation has led to the overall result **PASSED**, this condition is fulfilled (=evaluates to true).

- Execute on checklist result **NOT PASSED**
  If a checklist evaluation has led to the overall result **FAILED**, this condition is fulfilled (=evaluates to true).

- Execute on checklist result **NOT FINISHED**
  If a checklist evaluation has not led to a **PASSED** or **FAILED** overall status, this condition is fulfilled (=evaluates to true).

If 1 of the enabled conditions is fulfilled, actions and data field updates are actually executed according to the settings described below:

Actions

Activate the **Create and upload report to RayFlow** checkbox to automatically generate a report and export the resulting file to RayFlow. Since the report generation is not triggered manually, RayQC applies the settings defined within the currently active *report profile* for the report generation.

If the checklist evaluation has been triggered by RayFlow, and an explicit report profile has been forced by parameter, RayQC applies the rules defined for that specific report profile.

RayFlow Data Updates

Besides simply sending a report file to the RayFlow database, it is also possible to update dedicated fields of the package order object that is connected to the current checklist run.

Note:
Please remember that RayFlow connections may be defined either by the currently active *RayFlow connection profile*, or by injection via *command line* parameters used to initiate the active RayQC session.

The post processing configuration interface allows defining a set of RayFlow fields that will be updated with specific values. To demand a field update, users have to fill in the field name into the left column (RayFlow Field Name) of the RayFlow Data Updates control elements matrix, and the value that has to be set into the corresponding **Value to transmit** field.

Possible values for data field updates are:

- Predefined RayQC checklist object variables:
  - #title# is replaced by the checklist title
  - #date# is replaced by the current date (`YYYY-MM-DD`)
  - #status# is replaced by the checklist result (**PASSED**, **FAILED**, or **NOT FINISHED**)
  - #bypass# is replaced by the bypass status of the checklist result (**regular** or **bypassed**)

- Free text entered by the checklist designer.

- A mixture of both, free text and predefined variables

Checklists on the File System

Checklist template files (`*.rqct`) are usually stored at a file system location that is within reach of those users who have to work with them later. This means that storing checklists on a shared network location requires that checklist evaluators have access to that share in order to be able to evaluate the checklists as projects later.

As outlined before, there are two types of RayQC related files: RQCT and RQCP.
The first one is a ZIP containter containing:

- File `checklist.xml`: contains the different steps of the checklist
- File `postprocess.xml`: contains the data from the "Post Processing" tab
- Folder Help: contains the help files from the "Supporting Files" tab
- Folder Images: contains the images from the "Supporting Files" tab
- Folder Resources: contains the resources from the "Supporting Files" tab
- Folder plug-ins: contains the plug-ins from the "plug-ins" tab

The second type is also a `.zip` container that contains the same information as the `.rqct` file, along with: File `state.xml`: contains the current evaluation status of the actual project instance for testing

Formatting Markup Options

Some checklist properties may be equipped with formatting markup tags to provide a handy set of options for information structuring.
Tags are evaluated within the following checklist properties:

- Checklist description (in the Properties tab of the checklist)
- Group description
- Element description

This list of tags may be used:

- It is common to mark some text to be [bold]bold[/bold]. Along with the line break tag, this markup allows to define headlines and build logical blocks of information.

- Text with [italic]italic[/italic] style usually indicates quotes or references to external documents.

- Sometimes it might be required to define areas with [underline]underlined[/underline] style.

- Text with a specific [red]font color[/red] may be used to emphasize very important aspects of the description, or mark result specific information.
  RayQC accepts red and green as font color options.

- Text may be equipped with line[br]breaks.
  The break tag is the only one that does not require an opening and closing markup string.

Standard Checklist Procedures

Now that the overall structure of checklists has been outlined, it is time to give some details regarding the usual procedures users have to perform on RayQC checklist files.

Create Checklist Templates

Since direct XML source fumbling is finally a pain of the past with the latest RayQC release, this user manual will not give detailed instructions on checklist creation via the XML editor. The appendix section contains information about the XML schema and some basic usage samples.

To create a new checklist template

1. **Launch RayQC**, or - if it is already up and running - navigate to the **dashboard**.

2. Click the **create checklist** tile

3. Define the **name** and **location** for the new checklist template file (`.rqct`).
   Please keep in mind, that it is very likely that other users will require access to the checklist template in order to evaluate the checklist as project later. Therefore, it is recommended to use a shared location that evaluators have access to.

4. The template is immediately created with the default settings and contents, and opened for manipulation within the **Checklist Editor** interface.

Be aware:
RayQC automatically opens the template, which resides within the session memory. Changes to that file are not permanent until any of the provided Save functions is used. Be aware that saving as project file keeps the current checklist structure and evaluation status information, whilst saving as a template only saves the changes affecting the checklist structure (including plug-ins, supporting files and post processing actions) - evaluation status information is not stored within template files. Please refer to the objects section for further details.

5. Adjust the checklist properties according to your requirements.

6. Add groups as required, and define basic group properties as well as details such as conditions by using the group controls. Once the group container itself is prepared, it is time to add some actual checklist items to it, edit their basic properties, define required options, and use the controls provided per element to adjust the tree structure of group items.

   To add objects to a checklist, drag one of the object icons for groups or checklist items from the toolbox on the left-hand side of the Checklist Editor interface to the checklist area in the middle. Drop the new objects at the desired target position and adjust their default settings towards your individual needs.

7. If help files or other resources, such as images, are used within your checklist definitions, make sure to add them using the file managers that can be found under the Supporting Files tab.

8. If conditions are part of your group and element definition, it is recommended to use the Check Conditions button from the Swipe bar as soon as your checklist structure is complete. This button can be found in the Checklist Viewer. Switch to this view by clicking the **View this checklist** button at the bottom left. If there

are issues regarding the conditional statements that have been defined, solve them and repeat the Check Conditions procedure until all issues are cleared.

9. Save the changes to the checklist file by using the **Save** or **Save as** option from the File menu. The disk symbol in the upper left corner in the title bar can also be used as a shortcut for saving the current checklist.

Be aware:
Simply using Save will automatically detect whether evaluation status data is in the current project and will automatically save the current checklist as a project file (`*.rqcp`) including the state to ensure the saving of the result. If there is no evaluation data the checklist template will be saved as a template (`*.rqct`) file.

Evaluate Checklist Projects

Evaluating a checklist project means to open a checklist template with the Checklist Viewer and execute the test steps one after another. To save the evaluation results beyond the current work session, it is required to save the checklist as a project file (`.rqcp`). This is automatically done if the standard command **Save** is used.

To evaluate a checklist project

1. **Launch RayQC**, or - if it is already up and running - navigate to the **dashboard**.

2. Click the **open checklist** tile to load an existing checklist template.
   As an alternative, use the **open project** tile to go on with the evaluation procedure for an already existing project.

3. Select the required template or project file from the file system, using the system browser dialog displayed after clicking one of the tiles mentioned above.

Note:
If the checklist evaluation is triggered by a RayQC tool integration in RayFlow, the first three steps are obsolete, since they will be triggered automatically by RayFlow.

4. The file is immediately opened for evaluation within the **Checklist Viewer** interface.

5. Walk through the test steps one by one and note the results to make sure the sequence of result provision is kept as demanded by the checklist structure.

6. The type of result information required to complete a test item result depends on the item type:

   a. Information items do not require any user feedback at all. They are designed to provide information, e.g., about the test scenario or environment requirements.
   b. Data Field items expect a textual execution result to be documented. Simply enter the result into the text input field provided below the step description.
   c. Checkpoint items expect a boolean **Yes** or **No** result information. The answer that is defined as the expected, correct test result, is displayed in a bold font color. If neither **Yes** nor **No** is bolded, the result of this test item is not used for the overall calculation of the checklist result.
   d. Multi-option items require selecting one of the offered result options.

7. Double-check the results provided for checkpoint and multi-option items, since they are common references for dynamic checklist content control. The correct checklist procedure may very well be determined by the results defined for these types. Therefore, it is of high importance to give the correct results during the first checklist run, since later corrections may lead to additional result update measures.

8. Take a look at the icons presented within the definition of the single test steps:

   a. If a question mark icon is shown: Click it to read the additional information provided within the linked help file.
   b. If an arrow icon is shown: Click it to run the plug-in that has been attached to the checklist element

9. Make sure that all status indicator boxes from the task bar are displayed with a zero in the upper left corner and a green background color. If any of these boxes looks different, there are still missing test results and the overall checklist result will be **NOT FINISHED**. The checklist evaluation is complete when the status ribbon at the right-hand side of the task bar displays either **PASSED** or **FAILED**.

10. Save the result of your work in order to keep the current checklist result permanently. To do so, select one of the options available from the File menu:

    a. Option Save: Saves the changes to the currently opened RayQC project file
    b. Option Save as: Allows to save the changes in a newly created project file or to overwrite any existing project file.
    c. RayFlow - Upload Report: Creates the report and uploads it to the RayFlow server
    d. RayFlow - Update data fields: Based upon the post-processing configuration, the “Update data fields” button updates the relevant data fields in the RayFlow project

11. Depending on the demanded test reporting and RayFlow connection, it may be required to generate a checklist project report for local storage or transfer to RayFlow.

    a. To generate a report for local storage: Open the File menu and select Create Report. The displayed dialog requires target location, file name and report profile selection. Click Create to generate the report file at the selected destination.

       To create or change a report profile, go to Settings – Report Profiles.
    b. To generate a report for RayFlow storage: Open the file menu and click on RayFlow. In the displayed dialog click on the tile that is labelled with Upload Report. After pressing the button a dialog opens to enter the file name (no file extension needed). Uploading of the report to RayFlow will use the active report profile from the settings screen for the report generation. The file name suggestion for report creation is based on the checklist property 'File name pattern'.

Create Checklist Evaluation Reports

Working with RayQC checklist templates and projects is fine for checklist editors and evaluators, but not a handy solution for result communication with others that do not have access to RayQC.
To provide decent communication material, export functionality is integrated into both core interfaces: Checklist Viewer and Checklist Editor.

Reports are always based on the current project status as it is present within the temporary session storage. When the project file itself is updated, all former reports remain unchanged, and therefore represent an outdated project status.

To create a checklist evaluation report

1. **Launch RayQC**, or - if it is already up and running - navigate to the **dashboard**.

2. Click the **open checklist** tile to create a new project based upon an existing checklist template.
   As an alternative, use the **open project** tile to go on with the evaluation procedure for an already existing project.

3. Select the required project file from the file system, using the system browser dialog displayed after clicking one of the tiles mentioned above.

Note:
If the checklist evaluation is triggered by a RayQC tool integration in RayFlow, the first 3 steps are obsolete, since they will be triggered automatically by RayFlow.

4. The file is immediately opened for evaluation within the **Checklist Viewer** interface.

5. Depending on the demanded test reporting and RayFlow connection, it may be required to generate a checklist project report for local storage or transfer to RayFlow.

   a. To generate a report for local storage: Open the File menu and select Create Report. The displayed dialog requires target location, file name and report profile selection. Click Create to generate the report file at the selected destination. To create or change a own report profile, go to Settings – Report Profiles.

   b. To generate a report for RayFlow storage: Open the file menu and click on RayFlow. In the displayed dialog click on the tile that is labelled with Upload Report. After pressing the button a dialog opens to enter the file name (no file extension needed). Uploading of the report to RayFlow will usse the active report profile from the settings screen for the report generation. The file name suggestion for report creation is based on the checklist property 'File name pattern'.

Edit Checklist Templates

When a checklist template is opened in RayQC, it is opened within the Checklist Viewer, which allows direct evaluation steps to be taken. In order to actually edit the underlying template, the following procedure has to be executed:

To edit a checklist template

1. **Launch RayQC**, or - if it is already up and running - navigate to the **dashboard**.

2. Click the **open checklist** tile to create a new (temporary) project based upon an existing checklist template.

3. Select the required template file from the file system, using the system browser dialog displayed.

4. The file is immediately opened within the **Checklist Viewer** interface.

5. Click on the "Edit this checklist" button on the left side of the swipe bar below the actual checklist content.

6. The checklist template structure is displayed, ready for manipulation:

   a. Add new groups
   b. Edit basic group properties
   c. Remove groups
   d. Add new items to groups
   e. Edit basic item properties
   f. Edit element and group properties by using the property section on the right side (e. g. change conditions, options, etc.) or change their indention.
   g. Remove elements

7. Switch to the Checklist Viewer and review the effects the changes actually have on the behavior and content display of the checklist.

Remember that changes to the element positioning and conditions of elements and groups may lead to unexpected results.

8. When the checklist structure has been adjusted towards the desired state, the changes have to be saved. Saving can either mean to overwrite the existing checklist template, or save as a new one. Both methods start with clicking on the File menu, and selecting Save as. Select "RayQC Templates (`.rqct`)" as file type.

   a. To create a new checklist template file:

      i. Browse to the desired target destination and enter the name of the new file.
      ii. Click Save to create the new checklist template.

   b. To overwrite an existing template file:

      i. Browse to the location of the template file that has to be replaced by the current checklist template structure
      ii. Click on the file name of the template file that has to be replaced
      iii. Click Save.
      iv. Within the appearing confirmation dialog, click Yes to actually overwrite the existing file.

Be aware:
Simply using Save will automatically detect whether evaluation status data is in the current project or not. If evaluation status data is available, RayQC will automatically save the current checklist as a project file ('*.rqcp) including the state to ensure the saving of the data. If there is no evaluation data, the checklist will be saved as a template (*.rqct) file. If you only want to save the checklist without the evaluation status data, use "Save as" – "Save as Checklist" from the file menu.

Delete Checklists

Deleting a checklist is actually not a task that is performed directly from within RayQC. Even though it is possible to remove checklist projects or templates from the Recent list on the dashboard, this does not delete the original files from the system. To delete a checklist, users have to use the standard Windows functionality provided by the system explorer interface. This is an intended limitation to prevent data loss.

Plug-Ins

With the increasing complexity of a checklist, its processing can sometimes become very time-consuming. The RayQC plug-in interface is designed to provide a way to automate and streamline checklist evaluation procedures in order to avoid issues such as missing test standards, handling errors, and time constraints.

Plug-ins are capsuled, reusable logic units, accessible from RayQC checklists by direct function calls including required in- and output parameters. plug-in parameters can be derived from and injected into element results, which even allows to build conditional statements for checklist workflow control depending on plug-in execution results.

Therefore, advanced users of RayQC will in any case have to deal with plug-ins, understand how they are designed and integrated, know how to utilize internal ones, and finally be capable of creating new, external plug-ins for custom testing purposes.

The following sections are separated into logical units to provide insight into the RayQC plug-in technology from different angles:

- Plug-in types
- Using plug-ins in checklists
- Documentation of built-in RayQC plug-ins
- Requirements for custom plug-in preparation
- Additional information is provided within the Appendices of this document
- Troubleshooting advice for plug-in usage

Plug-In Types

Talking about plug-in types in a RayQC manner means to deal with criteria such as plug-in author and plug-in location. According to these decisive properties, there are standard plug-in types:

Internal Plug-Ins

- Directly integrated with the RayQC application.
- Their functionality is static, and can therefore not be changed but only called by users.
- Raynet is responsible for the provision of backwards-compatibility. Announcements regarding changes are given as part of the Release Notes published for each new product release.

External Plug-Ins

- RayQC supports two kinds of external plug-ins: DLL and PowerShell
- Their functionality is provided by RayQC users and can therefore be changed individually according to your needs
- Users are responsible for the provision of backwards-compatibility and internal version management.
- **Local** external plug-ins are stored in the `/plug-ins/` folder within the checklist container file.

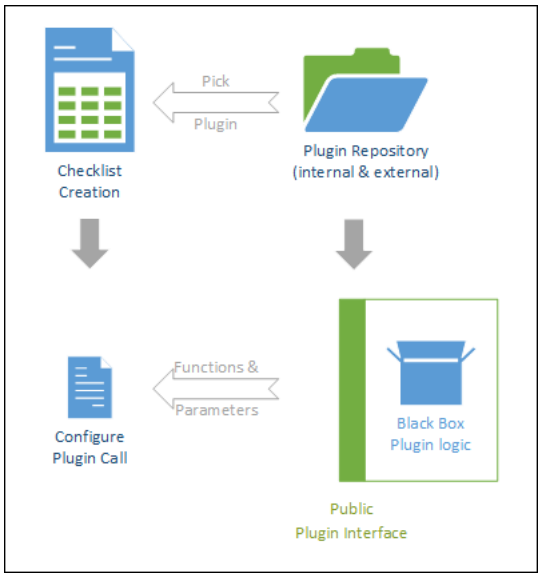
- **Global** external plug-ins are stored in the `/plug-ins/` folder within the RayQC application installation directory (usually something like `C:\Program Files (x86)\RayQC\`). They can be used by any checklist run from the local RayQC instance.

Summary: Plug-In Availability

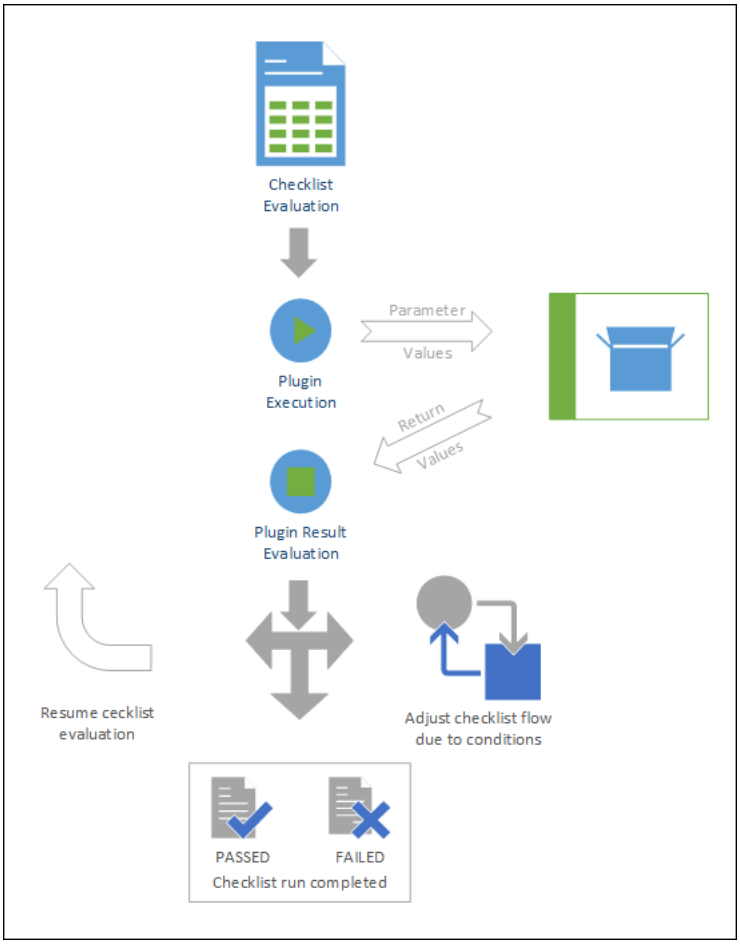
- **Internal** plug-ins are available for all checklists.
- **Local external** plug-ins are available for one specific checklist. In order to make them available for several checklists, their resource files have to be copied directly into the checklist container file.
- **Global external** plug-ins are available for all checklists that are run from the RayQC instance that owns the plug-ins subdirectory which includes the plug-in resources within its application installation directory (e. g. `C:\Program Files (x86)\RayQC\`)

Using Plug-Ins in Checklists

Plug-ins have to be configured during the creation of a checklist and will be executed during the evaluation of a checklist. The following illustrations show on a generalized level, how these procedures are intended to be aligned.



Plug-in configuration during checklist creation



Plug-in usage during checklist evaluation

Tip:
There are sample templates for plug-in usage, delivered along with the RayQC application resources. As soon as RayQC is installed, samples are stored within the RayQC program folder (e. g. `C:\Program Files (x86)\RayQC\Samples\Sample Checklist.rqct`, which is designed to present some functions of the RayQC plug-ins, along with their configuration options).

Configuration during Checklist Creation

Tip:
There are sample templates for plug-in usage, delivered along with the RayQC application resources. As soon as RayQC is installed, samples are stored within the RayQC program folder (e. g., C:\Program Files (x86)\RayQC\Samples\`Sample Checklist.rqct`, which is designed to present some functions of the RayQC plug-ins, along with their configuration options).

The following procedure outlines the standard requirements for plug-in configuration. Additional information regarding the relevant activities for *custom plug-in preparation* is provided in a later section.

1. Plug-ins are triggered from checklist elements, therefore the initial starting point for any plug-in usage is to *create a checklist* that contains at least one element that can incorporate a plug-in: a Checkpoint, Data Field, or Multi-Option element. Please note that it is not possible to add plug-ins to Information elements, as the return value of the plug-in function execution updates the element result content, and Information elements do not have a result value at all.

2. Open the checklist template file for manipulation within the **Checklist Editor** tab **Steps and Actions**.

3. Users will later trigger plug-in execution from the box representation of an element (within the **Checklist Viewer**). Therefore, focus the element that should later contain the plug-in trigger button by clicking on it. Activate the plug-in tab of the element within the **Details** pane on the right-hand side.

4. Browse the plug-in list at the left-hand side of the editor interface. Expand the list of functions per plug-in by clicking on the arrow icon left of the plug-in name. Once the right function is found, drag it to the desired checklist element within the checklist structure column in the middle of the application screen (or directly to the **Details** pane's plug-in tab of the currently selected element), and drop it there.

   If it is not possible to drop the function at the desired target element, there is a mismatch between the supported parent element types of the plug-in function and the type of the target element. Please refer to the *Internal plug-ins* section to research the valid function / element combinations for internal plug-ins. If external plug-ins are used, the definition of allowed target elements per plug-in function can be retrieved from the plug-in manifest file.

   Please note that each checklist element (except for groups and information elements)can be extended with zero or one plug-in functions. It is not possible to add two or more functions to the very same checklist element.

5. Once a plug-in function has been added to a checklist element, the functions details are displayed within the plug-in tab of the details pane for the checklist element. These details do not only contain static information about plug-in name, plug-in version, and function name, but also the controls for individual adjustments of input parameter values, execution options, and result handling settings.

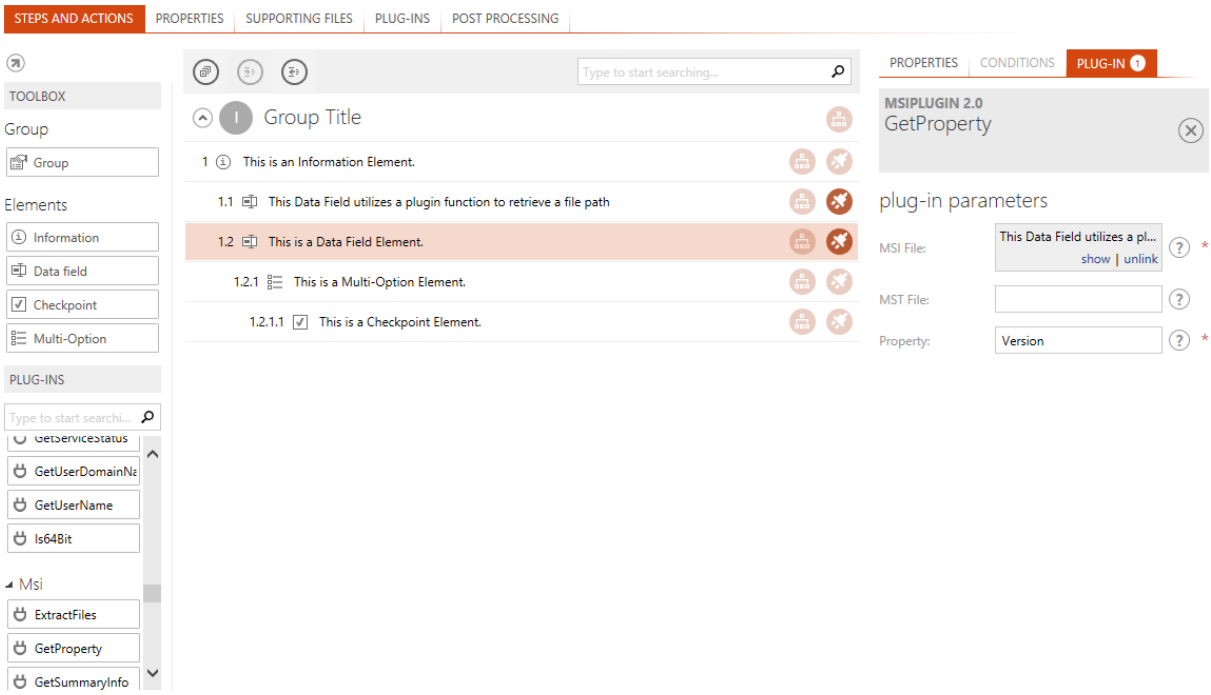
6. Please refer to the plug-in documentation for further details on the list of parameters for a specific function. The *documentation for internal plug-ins* is available within this document. The specifications for custom (external) plug-ins have to be provided by the plug-in author.

The following example uses the internal File plug-in as base for a configuration description. The specific function that is about to be executed is `GetProperty`, which allows retrieving either the version or size property of a specific file. (Once the property is read, another plug-in function may be used to evaluate the result. To compare the property with a specific value, users may add another checklist element and use the Logic plug-in function `CompareValue`.)

7. Input parameters can be defined by manually entering a static value or by reading the current value of another checklist element.

   To use the value of a checklist element as input parameter value, users have to drag the checklist element from the checklist structure area in the middle of the screen and drop it into the desired plug-in function input parameter control within the details pane.

   Please be aware that this kind of relation can only be established if the checklist element that is used as parameter value has already been evaluated when the current plug-in is executed. This means that it is not possible to use an element that has a later position within the checklist element structure than the element that contains the plug-in itself. For the situation displayed within the screenshot below: Element 1.1 may be used as input parameter reference for element 1.2 or any later element, but 1.2.1 may not be used for 1.1 or 1.2.



The selected element 1.2 has been extended with a plug-in function. The input parameter File is defined by the value stored within another checklist element: 1.1.
The functions of the File plug-in on the left hand side are expanded and contain the GetProperty function, which has been added to element 1.2. A look at the details pane on the right reveals the currently established usage connection, as the plug-in and function name of the related logic are displayed here.

8. Let us take a closer look at the input parameters for the `GetProperty` plug-in function:

   File
   The name and path of the file whose property has to be checked. In our example we have an element with the index number 1.1 as reference value for this parameter. Element 1.1 itself is equipped with a plug-in function that allows selecting a file via a system browser dialog: `FileOpenDialog`. Therefore, once the plug-

in of element 1.1 is executed, its value is set to the name and path of the selected file.

Property
The file name and path described above is used to retrieve the file property Version as result of element 1.2. The input parameter property has two options users may select from: **Version** and **Size**.

9. Now that the ingoing data flow is defined, a question might arise: Where does the result of the plug-in execution go to? Well, the current definition of internal plug-ins is restricted to allow exactly one outgoing (result) parameter per function. The return value of the function execution is automatically used as value of the element that hosts the plug-in function itself.

   Therefore, in our example described above, the result of the file property retrieval is written into the Data Field element 1.2, as well as the result of the `FileOpenDialog` function execution is written into the Data Field element 1.1. Within the Checklist viewer, users are able to see the function result as value of the control item that represents the element. In our example, the file path and the file property are entered into text input fields.

   Executing functions hosted by Checkpoint elements leads to the selection of either **Yes** or **No**. (Please keep in mind that **Yes** is usually the same as **evaluates to true**, but **evaluates to true** may as well be equal to **No** if the expected result of the element has been switched. Please refer to the *element type descriptions* for further information!)

   The third option for result parameter value indication within the **Checklist Viewer** is the auto-selection of an option from a Multi-Option drop-down menu.

   Since result values may not be provided in a standardized format, or easily usable for further evaluation, there may very well be need to format transitions and comparisons. Please take a look into the section about the internal **Logic** plug-in and its functions to find out how RayQC can support the individual QA test scenario requirements.

10. As soon as the configuration is done, the plug-in execution may be tested. To do so, users have to switch to the **Checklist Viewer** interface. Please keep in mind to save changes made to a checklist template in the meanwhile in order to prevent data loss.

11. Within the **Checklist Viewer**, evaluate the whole set of checklist items until the ones with the plug-in functions are reached. It is very common that plug-ins require results of prior elements as input parameters. The plug-in execution would fail if there were missing element results that are actually referred to by plug-in parameters. Therefore, it is recommended to run through the entire checklist evaluation course to make sure the parameter call is triggered from a realistic checklist status.

12. From the **Checklist Viewer**, users may check the existence of mandatory input parameter definitions for the defined plug-in function executions by using the Validate plug-ins data button from the Swipe bar. If RayQC encounters any issues, a message is displayed. Clicking on the **MORE** button reveals details on the erroneous parameter set. The info message refers to the element by naming the group index, which is represented by a roman number (e. g., I; II; III), followed by the element index value, which is represented by a period separated list of Arabic numerals (e. g., 1; 1.1; 2.4.3.1). The combination of these two index values unambiguously determines the affected element. To solve the stated parameter validation issues, users have to switch back to the **Checklist Editor** interface and edit the plug-in parameters of the mentioned elements.

13. However, if the plug-in execution does not run as expected, adjust the configuration, use the *troubleshooting* advice provided within this document, contact your local RayQC system administrator, or send a support request to Raynet: support@Raynet.de.

Execution during Checklist Evaluation

Tip:
There are sample templates for plug-in usage, delivered along with the RayQC application resources. As soon as RayQC is installed, samples are stored within the RayQC program folder (e. g. `C:\Program Files (x86)\RayQC\Samples\Sample Checklist.rqct`, which is designed to present some functions of the RayQC plug-ins, along with their configuration options).

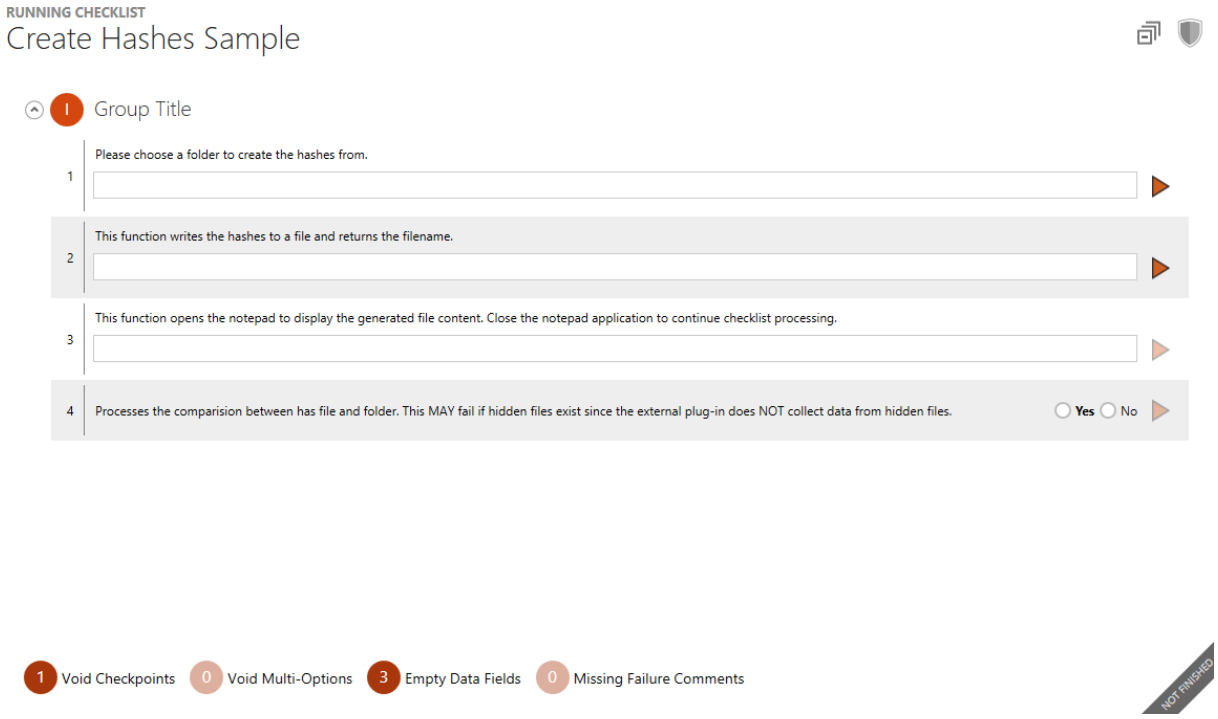
How to trigger plug-in execution

Once a plug-in has been integrated into the checklist template file, it may be run from the **Checklist Viewer** interface.

Be aware:
Even though it is technically not required, it is highly recommended to save the current checklist definition before a test run is executed. Especially the integration and execution of external plug-ins may lead to severe system interferences. Please keep in mind, that RayQC does not prevent critical script logic to be run as part of external plug-ins. It is the plug-in and checklist authors area of responsibility to make sure no harm is caused by their functions on the affected systems.

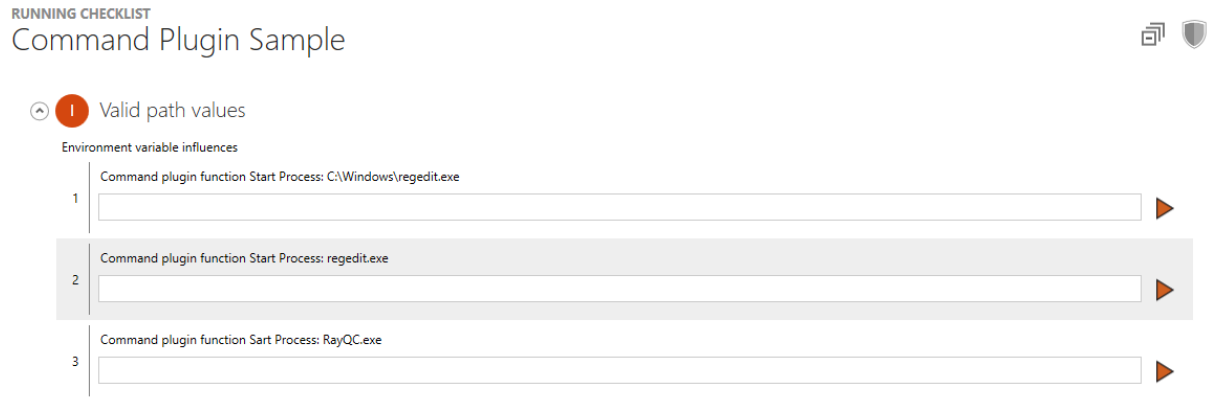
The representative box for a checklist element contains an arrow icon to indicate plug-in existence. As soon as the evaluating user clicks this button, the plug-in logic is executed according to the settings defined within the checklist, the plug-in script logic, and the current state of the target system, or respectively any other object of investigation within plug-in reach.

The screenshot below displays some element boxes taken from the plug-ins checklist template sample. The trigger icons for plug-in execution are shown within checkpoint, user comment, and comment element boxes. However, it is also possible to integrate plug-in execution into the interface of multi-option items.



In order to provide correct and reliable execution results, users should evaluate the whole sequence of checklist items until the one with the plug-in is reached. It is very common that plug-ins require results of prior elements as input parameters. The plug-in execution would fail if there were missing element results which are actually used by the current plug-in configuration. Therefore, it is recommended to run through the whole checklist evaluation course to make sure the parameter call is triggered from a realistic checklist status.

The actual result of plug-in execution depends on the plug-in selected, the function called, the parameters used, and - of course - the status of the target system that is checked or manipulated by the plug-in logic. Let us take a look at the **Command** plug-in example shown below. The Data Field element (1) available from the sample checklist group "I: Valid path values" is about to call the **Registry Editor** (`regedit.exe`) of the local machine. With a click on the plug-in trigger button, the configured command line is fired. The file name and full path to the executable was provided and regedit is started successfully. But how about the other examples? Will they plug-in execution for element 2 launch the **Registry Editor** as well?



Yes, it does launch the **Registry Editor** - but only on those machines, who have `C:\Windows\` added to the PATH environment variable. RayQC reads these variables during executable path resolution.

The same notation would also work for `notepad.exe` or `explorer.exe` as long as the underlying Windows operating system is not highly customized regarding its system settings and default parameters. Addressing applications in non-standard local or network locations requires to give the fully qualified path (e. g., `C:\Program Files (x86)\RayPack\RayPack.exe` to launch RayPack). The executable whose execution is triggered by element no 3 resides within the RayQC program directory (typically something like `C:\Program Files (x86)\RayQC\`), which is by default checked for a matching file as well. All other plug-in parameters that represent paths have to be given fully qualified, the **Command** plug-in is the only exception on this rule.

Run single plug-in vs. run all plug-ins

As outlined before, it is possible to trigger a single plug-in execution with a click on the rightwards pointing arrow icon which is available directly from the elements box representation. However, there may be automated checklists, which fully consist of stacked plug-in-based checks. If these checklists are opened for evaluation, it is faster and more convenient to use the **Run All** button from the **Swipe Bar** below the checklist area.

The sequence of plug-in execution is initially determined at the beginning. When the first currently available checklist item with an integrated plug-in is reached and executed, the results are written to the checklist project residing within the session memory. At this point, the group and item sequence is recalculated in order to be able to consider result-driven changes to the availability of upcoming checklist objects (please refer to the conditions section for details).

This procedure of

a) plug-in run
b) sequence recalculation according to plug-in run results
c) proceeding to the next plug-in according to the newly calculated sequence

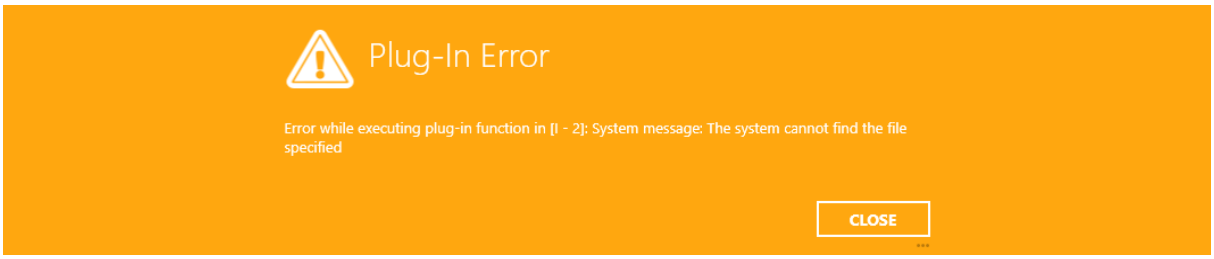
is repeated until all checks have been run according to the requirements defined by the checklist template and the actual execution results.

Be aware:
If a plug-in does not run fully automated, but requires user interaction (e. g. when a browser dialog invocation is triggered), the next plug-in execution will not be launched until the execution of the last one has finished (e. g. a file has been selected from a system browser dialog, or a registry value has been read). However, it does not matter if the plug-in execution was successful or not, as soon as the triggered function has terminated, RayQC proceeds to the next checklist plug-in call.

Typical plug-in execution issues

In our example above, the plug-in execution ran flawlessly. Unfortunately, there may be less lucky circumstances that raise issues on plug-in execution.

For example, when a plug-in logic requires additional resources that are not available, because executing user does not have sufficient access rights to the affected system parts, a path information is invalid, or the like. If an external plug-in script is triggered from a checklist, but this plug-in is not present in a known and reachable location, a Windows Script Host Error message is shown, outlining the reasons for the invalidity of the plug-in script invocation. There are some *troubleshooting* hints given later in this document, which may be of help in such a situation. However, the following screenshot displays a typical RayQC message, caused by an invalid command name argument (e. g., according to our sample checklist: `regedit.dll` instead of `regedit.exe` in element no 2.)

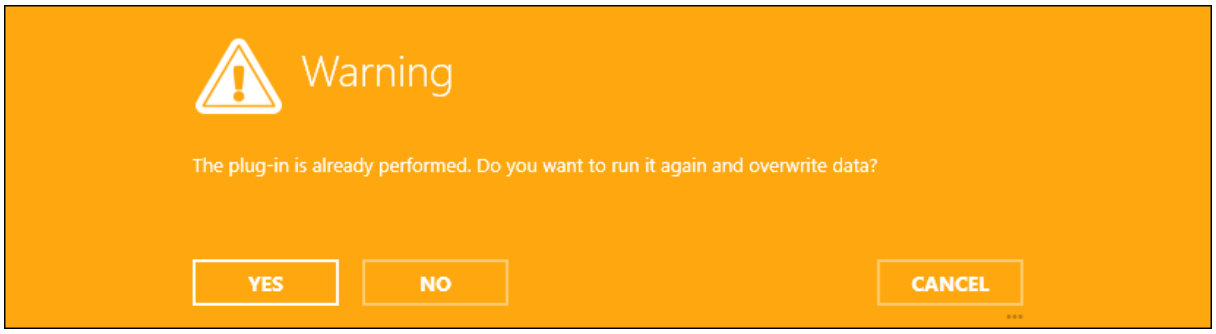


If a plug-in has not been configured correctly, there may be issues with parameter settings that are incorrect towards format restrictions, or with missing parameter information that is mandatory for plug-in execution. In this cases, RayQC usually displays an error message, describing the parameter that actually raised the issue, along with some helping notes on how to solve it.

Note:
If the plug-in execution does not run as expected, adjust the configuration, use the *troubleshooting* advice provided within this document, contact your local RayQC system administrator, or send a support request to Raynet: https://raynetgmbh.zendesk.com.
Please keep in mind that our support team cannot assist on the elimination of issues caused by external plug-in logic that you implemented on your own. If there are further requirements for Raynet specialist back up regarding external plug-ins, there are additional trainings or consultant services Raynet is always pleased to offer.

Multiple plug-in execution

Whenever a checklist evaluation has been started by giving element results and / or running plug-ins, RayQC stores the current item availability and result state within the session memory. Due to *conditional* statements, changing a single item result may significantly change the availability of checklist items and groups. Combining the idea of plug-in execution results which may be injected as results into element results, and conditions that depend on element results, it becomes clear that rerunning a plug-in may lead to differing results (especially during the checklist preparation phase when the configuration settings or system preconditions usually change between the plug-in executions).



In order to make sure that users are aware of rerunning a plug-in, RayQC displays an info dialog, asking for a confirmation of the follow-up triggering. Please make sure that overwriting existing data is really the intended action. To be absolutely sure about the correct checklist flow results, it is recommended not to rerun a single plug-in, but to *reset* the whole checklist and start again from the beginning. Especially results generated from highly complex conditional statements may turn out to be confusing when only parts of a checklist are rerun, or retriggered in case of plug-ins.

Internal Plug-Ins

When RayQC is installed on a system, a number of built-in (internal) plug-ins become available. Their functionality is static, and can therefore not be *changed*, but only *called* by elements.

The following sections describe the purpose and public interface of the internal plug-ins. In order to get details on how to integrate them into checklists, or on how to handle plug-in execution during checklist evaluation, please refer to the specific sections within this document.

Read on for technical information regarding

- Command plug-in
- File plug-in
- Folder plug-in
- Ini File plug-in
- Local System plug-in
- Logic plug-in
- MSI plug-in
- RayFlow plug-in
- Registry plug-in
- Advanced plug-in

Be aware:
If not explicitly described differently, the following general rules apply to plug-in handling:

- All input parameters for internal plug-ins are mandatory.
- Whenever a plug-in execution fails due to missing or invalid parameter definitions, the value of the plug-ins parent element is set to an informative error message.
- The same error message communication is applied whenever a timeout exceeds or another issue prevented the plug-in from standard termination.
- The maximum number of characters for a string is 1,073,741,823.
- The range for an integer value is from –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
- All string comparisons are case sensitive. Some functions may alter this by using an additional boolean parameter.

Note:
Further information regarding external (custom) plug-ins will be given in the External Plug-Ins section of this document.

Command Plug-In

The Command plug-in offers simple access to the Windows command prompt, allowing users to either start a process or run a VB script.

Function summary

- RunVBS
- StartProcess

Function details

RunVBS

Launches a visual basic script and returns the last line of output.
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
Script	formatted string	The name and path of the VB Script that has to be run. Example: `C:\Program Files (x86)\RayPack\RayPackHelper.vbs`
Arguments	string optional default value: none	The optional set of arguments used to transfer additional parameter information for the script execution itself. Example: `"MSI C:\RayPack\Projects\FileZilla_3_1_1_2\_MSI\FileZilla.msi"`
Timeout	integer optional default value: -1	Time in milliseconds the script is allowed to run. Options: • -1 indicates infinite timeout. • 0 leads to immediate timeout at script execution. (This setting may be used for test purposes) • All other positive integer values are evaluated as allowed script run time in milliseconds.
ResultSource	formatted string	The definition of the scripts communication pipe that has to be used as source for the return value. Options: • `"Standard output"` • `"Standard error"` • `"Exit Code"`

StartProcess

Starts a process.
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
Command	formatted string	The name and path of the command that has to be executed. In case the file is located in a folder that is part of the environment variable PATH, the path of the file can be omitted. Example: `ipconfig`
Arguments	string optional	The optional set of arguments used to specify details of the command parameter.

	default value: none	Example: `/all`
Timeout (ms)	integer optional default value: -1	Time in milliseconds the script is allowed to run. Options: • -1 indicates infinite timeout. • 0 leads to immediate timeout at script execution. (This setting may be used for test purposes) • All other positive integer values are evaluated as allowed script run time in milliseconds.
ResultSource	formatted string	The definition of the scripts communication pipe that has to be used as source for the return value. Options: • `"Standard output"` • `"Standard error"` • `"Exit Code"`

File Plug-In

The File plug-in deals with functionality required to retrieve information about and to manipulate files.

Function Summary

- CompareFiles
- CopyFile
- DeleteFile
- FileExists
- FileOpenDialog
- GetACL
- GetCertificateData
- GetHash
- GetProperty
- GetTextFileContent
- Unzip
- VerifyZip

Function Details

CompareFiles

Binary content comparison of files.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
FileA	formatted string	The name and path of the first file for the comparison. Example: `C:\Temp\Sample.vbs`
FileB	formatted string	The name and path of the second file for the comparison. Example: `C:\Temp\Sample2.vbs`

CopyFile

Copies a file. If the file targeted by source is not found, the returned result is **No**, otherwise **Yes**.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
Source	formatted string	The name and path of the source file. Example: `C:\Temp\Sample.vbs`
Target	formatted string	The target destination (full path with file name). Example: `C:\Windows\Sample2.vbs`

DeleteFile

Deletes a file of the given name.
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
Filename	formatted string	The name and path to the file that is to be deleted. Example: `C:\Windows\SampleFile.msi`
Delete after reboot	boolean string optional default value: `false`	If set, the file will be deleted after reboot. • `false`: The file will be deleted directly. • `true`: The file will be deleted after a reboot.

Be aware:
The **Delete after Reboot** option will only be working if RayQC has been started using administrative rights!

FileExists

Checks if a file exists. If the file is not found, the returned result is **No**, otherwise **Yes**.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
File	formatted string	The name and path of the file whose existence has to be verified. Example: `C:\Temp\Sample.vbs`

FileOpenDialog

Displays a new instance of an open file system dialog.
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
FileType	formatted string	• 1..n filter definitions, separated by a pipe (`|`) • Each filter definition consists of "`description|pattern`" Example: To filter for text files: `Text files (*.txt)|*.txt|All files (*.*)|*.*`

		To deactivate the filter: `All files (*.*)|*.*`
FullFilename	boolean string optional default value: true	Options: • `true`: Result contains full file path • `false`: Result contains file name (without path)

GetACL

Get a file's access control list.
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
File	formatted string	The name and path to the file whose ACL info has to be retrieved. Example: `C:\Windows\SampleFile.ini`

GetCertificateData

Reads the subject string from the certificate of a file.
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
File	formatted string	The name and path of the file whose certificate info has to be retrieved. Example: `C:\Windows\SampleFile.ini`

GetHash

Returns the hash value for a file.
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
File	formatted string	The name and path of the file whose hash value has to be

		calculated. Example: `C:\Windows\SampleFile.ini`
Algorithm	formatted string optional default value: SHA-1	List of available calculation algorithms to select from. Options: • `MD5` • `SHA-1` • `SHA256` • `SHA384` • `SHA512` • `RIPEMD160`

GetProperty

Get the value for the chosen file property.
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
File	formatted string	The name and path of the file whose property value has to be retrieved. Example: `C:\Windows\SampleFile.ini`
Property	formatted string	List of available properties to select from. Options: • `Version`: the file version • `Size`: the file size in bytes

GetTextFileContent

Gets the content of a text file including white characters.
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
Text file path	string	The name and path of the file whose property value has to be retrieved. Example: `C:\Windows\example.txt`

Unzip

Extracts files from a ZIP archive.
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
File	formatted string	The name and path of the `.zip` file.

		Example: `C:\Windows\SampleContainer.zip`
Cleanup	boolean string optional default value: `true`	The path to the folder that contains the unzipped files is determined by the SHA-1 hash value of the `.zip` container. Options: • **enabled**: If the calculated unzip target folder already exists, it will be reused. • **disabled**: The calculated unzip target folder will be cleared if it already exists, and populated with the current `.zip` content.

VerifyZip

Validates the integrity of a ZIP file. If needed, this function processes a full archive extraction for checking.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
File	formatted string	The name and path of the `.zip` file. Example: `C:\Windows\SampleContainer.zip`
Extract Check	boolean string optional default value: `false`	Determines whether or not RayQC extracts the files if the requested `.zip`. Options: • **enabled**: Files are extracted to check the archive structure. • **disabled**: Files are not extracted.

Folder Plug-In

The Folder plug-in deals with functionality required to retrieve information about folders and to manipulate them.

Function Summary

- CompareFolders
- CompareFolderWithHash
- CopyFolder
- CreateFolder

- FolderExists
- FolderOpenDialog
- GetACL

Function Details

CompareFolders

Binary content comparison of files within 2 folders.
Usable in combination with elements of type Checkpoint.

Input parameters

Name	Type	Description & Examples
FolderA	formatted string	The path to the first folder for the comparison. Example: `C:\Temp\Sample\`
FolderB	formatted string	The path to the second folder for the comparison. Example: `C:\Temp\Sample2\`
Include Subdirectories	boolean string optional default value: true	Options: • enabled: Recursive comparison of sub-directories is applied • disabled: Comparison is not executed on files within sub-directories

CompareFolderWithHash

Compares the hash values of the files stored within a folder with a given table of hash | path values.
Usable in combination with elements of type Checkpoint.

Input parameters

Name	Type	Description & Examples
Folder	formatted string	The path to the folder for the comparison of file hashes. Example: `C:\Temp\Sample\`
HashFile	formatted string	The path to the file that contains the hash values for comparison. Example:

		`C:\Temp\SampleHashValues.txt`
Hash Algorithm	formatted string optional default value: SHA-1	List of available calculation algorithms to select from. Options: • `MD5` • `SHA-1` • `SHA256` • `SHA384` • `SHA512` • `RIPEMD160`
Include Subdirectories	boolean string optional default value: true	Options: • enabled: Recursive comparison of sub-directories is applied • disabled: Comparison is not executed on files within sub-directories

CopyFolder

Copies a folder. Returns **Yes** if copying was successful.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
Source	formatted string	The path to the source folder. Example: `C:\Temp\Sample\`
Target	formatted string	The target destination. Example: `C:\Windows\Sample2\`
Include Subdirectories	boolean string optional default value: true	Options: • enabled: Recursive copy of sub-directories is executed • disabled: Copy is not executed sub-directories and files within sub-directories

CreateFolder

Creates a folder. Returns **Yes** if the creation was successful.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	Type	Description & Examples
Folder Name	formatted string	The path of the folder that has to be created. Example: `C:\Temp\Sample\NewFolder\`

FolderExists

Checks if a folder exists. If the directory is not found, the returned result is **No**, otherwise **Yes**.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	Type	Description & Examples
Folder	formatted string	The path of the folder whose existence has to be verified. Example: `C:\Temp\Sample\ExistingFolder\`

FolderOpenDialog

Shows an open folder system dialog.
Usable in combination with elements of type Data Field.

This function does not accept parameters.

GetACL

Get a folder's access control list.
Usable in combination with elements of type Data Field.

Input parameters		
Name	Type	Description & Examples

Folder	formatted string	The path of the folder whose ACL info has to be retrieved. Example: `C:\Temp\Sample\Folder\`

IniFile Plug-In

The IniFile plug-in deals with functionality required to read or write INI file contents.

Function Summary

- GetSectionKeys
- GetSections
- KeyExists
- Read
- SectionExists
- Write

Function Details

GetSectionKeys

Returns a pipe-separated section names list.
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
File Name	formatted string	The path to the INI file. Example: `C:\Temp\Sample.ini`
Section	string	Name of the section which has to be searched for the key. Example: `[URL]`
CaseSensitive	boolean optional default value: true	Specify if the input parameter is case sensitive or not Example: • `enabled`: input parameter is case sensitive • `disabled`: input parameter is not case sensitive

GetSections

Returns the pipe-separated key names list of a specified section.
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
File Name	formatted string	The path to the INI file. Example: `C:\Temp\Sample.ini`
CaseSensitive	boolean optional default value: true	Specify if the input parameter is case sensitive or not Example: • `enabled`: input parameter is case sensitive • `disabled`: input parameter is not case sensitive

KeyExists

Checks whether or not a specific INI file key exists.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
File Name	formatted string	The path to the INI file. Example: `C:\Temp\Sample.ini`
Section	string	Name of the section which has to be searched for the key. Example: `[URL]`
Key	string	Name of the key that has to be looked for. Example: `Protocol`
CaseSensitive	boolean optional default value: true	Specify if the input parameter is case sensitive or not Example: • `enabled`: input parameter is case sensitive • `disabled`: input parameter is not case sensitive

Read

Reads an INI file value.
Usable in combination with elements of types Data Field and Multi-Option.

Input parameters		
Name	**Type**	**Description & Examples**
File	formatted string	The path to the INI file. Example: `C:\Temp\Sample.ini`
Section	string	Name of the section which has to be searched for the key. Example: `[URL]`
Key	string	Name of the key that has to be read. Example: `Protocol`
CaseSensitive	boolean optional default value: true	Specify if the input parameter is case sensitive or not Example: • `enabled`: input parameter is case sensitive • `disabled`: input parameter is not case sensitive

SectionExists

Checks for an existing INI file section.
Usable in combination with elements of types Data Field and Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
File Name	formatted string	The path to the INI file. Example: `C:\Temp\Sample.ini`
Section	string	Name of the section which has to be searched for the key. Example: `[URL]`

CaseSensitive	boolean optional default value: true	Specify if the input parameter is case sensitive or not Example: • `enabled`: input parameter is case sensitive • `disabled`: input parameter is not case sensitive

Write

Writes an INI file value and returns **Yes** on success.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
File	formatted string	The path to the INI file. Example: `C:\Temp\Sample.ini`
Section	string	Name of the section that contains the key. If the section is not present in the INI file, it will be added. Example: `[URL]`
Key	string	Name of the key whose value is about to be set. If it is not present in the target section, it will be added. Example: `Protocol`
Value	string	Value that has to be set for the specified key. The value must not be empty, and is not checked for suitable handling of special characters. If the key already exists within the target INI file section, the existing content will be overwritten. Example: `HTTPS`
CaseSensitive	boolean optional default value: true	Specify if the input parameter is case sensitive or not Example: • `enabled`: input parameter is case sensitive • `disabled`: input parameter is not case sensitive

Local System Plug-In

The Local System plug-in deals with functionality required to read gather information about the Local System.

Function Summary

- GetEnvVariable
- GetMachineName
- GetOsName
- GetProcessorCount
- GetResources
- GetServiceStatus
- GetUserDomainName
- GetUserName
- Is64Bit

Function Details

GetEnvVariable

Data Field

Returns the current value of the referenced environment variable.
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
Variable	string	Name of the environment variable whose value has to be retrieved. Example: `TMP`

GetMachineName

Data Field Multi-Option

Returns the name of the machine RayQC is running on.
Usable in combination with elements of types Data Field and Multi-Option.

This function does not accept parameters.

GetOsName

Returns the name of the operating system.

Data Field | Multi-Option

Usable in combination with elements of types Data Field and Multi-Option.

Input parameters		
Name	**Type**	**Description & Examples**
Translate	boolean optional default value: true	Switch for return value type definition. Example: • `enabled`: the translated OS name is returned • `disabled`: the raw OS version string is returned

GetProcessorCount

Return the processor count for the current machine.

Data Field | Multi-Option

Usable in combination with elements of types Data Field and Multi-Option.

This function does not accept parameters.

GetResources

Returns a path to the current working directory where supporting files [Resources] can be found.

Data Field

Usable in combination with elements of types Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
File	formatted string optional default value: none	If a file name is given, RayQC will extend the return value with this file name to provide the full path to the resource file including the actual file name for direct file access. Example: Using `Sample.txt` as value for the File parameter `C:\Users\Admin\AppData\Local\Temp\RayQC\working\2040_0\Resources\` as path retrieved by the function leads to `C:\Users\Admin\AppData\Local\Temp\RayQC\working\2040_0\Resources\Sample.txt` as total return value of the plug-in function execution

GetServiceStatus

Returns the status of the given service.
Usable in combination with elements of type Data Field and Multi-Option.

Input parameters		
Name	**Type**	**Description & Examples**
ServiceName	string	Name of the service whose status has to be retrieved. Example: `Remote Registry`

GetUserDomainName

Returns the user domain name for the user running RayQC.
Usable in combination with elements of types Data Field and Multi-Option.

This function does not accept parameters.

GetUserName

Returns the user name for the user running RayQC.
Usable in combination with elements of types Data Field and Multi-Option.

This function does not accept parameters.

Is64Bit

Returns **Yes** if the current operating system is a 64bit system.
Usable in combination with elements of type Data Field.

This function does not accept parameters.

Logic Plug-In

The Logic plug-in deals with functionality required to convert, compare and analyze values.

Function Summary

- BooleanToString
- CompareValue
- ContainsString
- GetTrimmedString
- GetTrimmedStringStatic
- InvertBoolean

- FormatString
- GetRegexGroup
- GetStringLength
- GetSubString
- MatchRegex
- NumberInRange
- StringToString
- TrimWhitespace

Function Details

BooleanToString

Data Field | Multi-Option

Converts a boolean value or checkpoint result to a defined string. This function can for example be used to map a checkpoint state to a multi-option item. Usable in combination with elements of type Data Field and Multi-Option.

Input parameters

Name	Type	Description & Examples
Value	boolean default: `true`	Boolean value, which is in most cases defined as reference to a checkpoint result. Options: • `enabled`: the text entered into the **True Text** parameter is returned. • `disabled`: the text entered into the **False Text** parameter is returned.
TrueText	string	The text that is returned if value is `true`. Example: `64bit Architecture`
FalseText	string	The text that is returned if value is `false`. Example: `32bit Architecture`

CompareValue

Compares two values, which may be required to compare the result of two checklist elements, or a static string with an element result.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
Actual	string	The first string for the comparison. Example: `Raynet GmbH`
Expected	string	The second string for the comparison. Example: `raynet gmbh`
Case sensitive	boolean optional default: `true`	Defines if the comparison has to be executed case sensitive or not. Example: `enabled =` The 2 strings used as examples above will be regarded to be different. `disabled =` The 2 strings used as examples above will be regarded to match.
Custom Failure Comment	string optional	The default failure comment of the Checkpoint element can be exchanged by this string if required. Example: This environment requires the manufacturer property and the author attribute of the summary information stream to match case sensitive.

ContainsString

Searches for a specific string in another string.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
Text	string	The haystack to search within. Example: RayQC is a rule-based tool used to create and execute test criteria in one or more checklists.
Search	string	The needle to search for.

		Example: `Checklist`
CaseSensitive	boolean optional default: `true`	Defines if the needle has to be found exactly in the case usage as defined within the parameter search. Example: `enabled =` The needle defined above will not be found within the example haystack. `disabled =` The needle defined above will be found within the example haystack..
Custom Failure Comment	string optional	The default failure comment of the Checkpoint element can be exchanged by this string if required. Example: Case sensitive substring search did not lead to a match.

FormatString

Data Field | Multi-Option

Formats a string by replacing up to five defined placeholders with arguments. A placeholder has the format '#x#' with 'x' as number of the argument.
Usable in combination with elements of type Data Field and Multi-Option.

Input parameters		
Name	**Type**	**Description & Examples**
Text	string	The string that contains the placeholders that have to be replaced. Example: The value of the `#1#` property is defined as `#2#`.
Replacement #<number>#	string	The value of the `#1#` property is defined as `#2#` The value inserted into the original text at the position of the placeholder `#number#`. Example: `manufacturer`

GetRegexGroup

Data Field | Multi-Option

Checks whether a text matches the pattern defined by a singular regular expression group.
Usable in combination with elements of type Data Field and Multi-Option.

Input parameters		
Name	**Type**	**Description & Examples**

Text	string	The string that is scanned for regular expression matching. Example: `uninst_myApp.lnk`

Regular Expression	string	The regular expression pattern the string is evaluated by. The sample pattern given below matches the sample text above.
		
		Example:
		`First test value is "AAA" and the second test value is "111".`
		
		`"(.*?)"`
		
		`Gets the string between first quote.`

Tip:
Please take a look at the `Regular Expressions` section for further information and assistance regarding regex usage.

GetStringLength

Returns the length of a string. If no string is given, 0 is returned

Data Field | Multi-Option

Usable in combination with elements of types Data Field and Multi-Option.

Input parameters		
Name	**Type**	**Description & Examples**
Text	string optional default: none	String of which the characters are counted. Example: The following sentence has a string length of 161: RayQC increases the quality of your software deployments through multi-level and standardized test plans by relieving the manual testing tasks of your workforce.

GetSubString

Returns a substring from a given text.

Data Field | Multi-Option

Usable in combination with elements of type Data Field and Multi-Option.

Input parameters		
Name	**Type**	**Description & Examples**
Text	string	The string that is used to extract the substring from. The samples used here lead to the extraction of the substring `experience`.

		Example: `Improve end-user experience and productivity!`
First	integer minimum value: 1	The first letter, number or special character that is returned as part of the substring. The counted value starts at 1 for the first character of the **Text** parameter. Using negative numbers will count from the end (`true` for both numbers) Example: `17`
Last	integer	The last letter, number or special character that is returned as part of the sub- string. The counted value starts at 1 for the first sign of the **Text** parameter. **Be aware:** The value given for this parameter may not be lower than the value entered for the parameter First. Example: `26`

GetTrimmedString

Returns a sub-string as result of trimming a given string.

Data Field | Multi-Option

Usable in combination with elements of types Data Field and Multi-Option.

Input parameters		
Name	**Type**	**Description & Examples**
Text	string	The string that is trimmed. The sample used here lead to the extraction of the trimmed sub-string `00FF00`. Example: `<#!"§00FF00+#>`
TrimStart	string optional default: none	The set of characters that is trimmed from the start of the original string. The order of the characters is not important. Example: `<§"!#`
TrimEnd	string optional default: none	The combination of signs that is trimmed from the end of the original string. The order of the characters is not important. Example: `#+>`

GetTrimmedStringStatic

Returns a sub-string as result of trimming a given string.

Usable in combination with elements of types Data Field and Multi-Option.

Input parameters		
Name	**Type**	**Description & Examples**
Text	string	The string that is trimmed. The sample used here lead to the extraction of the trimmed substring `00FF00+#`. Example: `<#!"§00FF00>`
TrimStart	string optional default: none	The set of characters that is trimmed from the start of the original string. **Be aware:** The order of characters is important. The given string **must** exactly **match** the start of the Text parameter, otherwise the trimming will not be executed. Example: `<#!"§`
TrimEnd	string optional default: none	The combination of signs that is trimmed from the end of the original string. **Be aware:** The order of characters is important. The given string **must** exactly **match** the end of the Text parameter, otherwise the trimming will not be executed. Example: `#+>`

InvertBoolean

Inverts a boolean (**Yes** / **No**) value.

Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
Value	boolean	The original boolean value that has to be inverted.

		Be aware: Inverting a boolean value does not invert the value expectation of the element itself! If **Yes** is the expected result of the Checkpoint element that has been extended by this plug-in function, and the Value parameter is set to **No**, the element will evaluate to `false` / **No**. Please make sure to double-check expected element evaluation results whenever *expected value* and / or invert boolean are applied to any Checkpoint element! Options: • `enabled`: is inverted to **No** • `disabled`: is inverted to **Yes**

MatchRegex

Checks if the text contains a string that matches a specific regular expression. If `FailureComment` is not set, it is autmatically generated by the plug-in.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
Text	string optional	The string that is scanned for regular expression matching. Example: `uninst_myApp.lnk`
Regular Expression	string	The regular expression pattern the string is evaluated by. The sample pattern given below matches the sample text above. Example: `^unins(|t|tall)\d*\.(cif|cfg|dat|dll|ini|exe|xml|lnk)$`
FailureComment	string optional default: generated by RayQC	The failure comment that is presented if the regular expression does not match the **Text** parameter.

Tip:
Please take a look at the *Regular Expressions* section for further information and assistance regarding regex usage.

NumberInRange

Checks the input parameter for being an integer in the given range.
Usable in combination with elements of type Checkpoint.

Input parameters

Name	Type	Description & Examples
Text	string	The numeric string that is evaluated regarding its integer value. Example: • `15` • `F`
HexValue	boolean optional default: `false`	If a hexadecimal number is given as input parameter value, this option has to be set to `true` in order to achieve a valid check result. Options: • `enabled` = checkbox is activated: input is evaluated as hexadecimal value. • `disabled` = checkbox is not activated: input is evaluated as integer value.
Minimum	integer optional default: none	The minimum value of the checked range (inclusive). Example: • `14` leads to evaluation to `true` for the sample **Text** parameter defined above. • `15` leads to evaluation to `true` for the sample **Text** parameter defined above. • `16` leads to evaluation to `false` for the sample **Text** parameter defined above.
Maximum	integer optional default: none	The maximum value of the checked range (inclusive). Example: • `14` leads to evaluation to `false` for the sample Text parameter defined above. • `15` leads to evaluation to `true` for the sample Text parameter defined above. • `16` leads to evaluation to `true` for the sample Text parameter defined above.

StringToString

Converts string to string based on two synchronized lists. Lists are regarded to be synchronized if they have the same number of elements. This function can for example be used to map plug-in function output to multi-option values.
Usable in combination with elements of type Data Field and Multi-Option.

Input parameters

Name	Type	Description & Examples
Value to convert	formatted string	The string that will be converted Example: `There is a bird in the tree.` Following the example lists below, this string will be converted to "`There is a cat in the house.`"
From	formatted string	The list of strings that is being searched for. List items have to be separated by the pipe sign (|). Example: `Bird|Mouse|Dog|Tree`
To	string	The list of strings that will replace the strings defined in parameter **From**. Example: • `Cat|Home|Fish|House`

TrimWhitespace

Returns the given text with trimmed whitespaces.

Data Field | Multi-Option

Usable in combination with elements of types Data Field and Multi-Option.

Input parameters		
Name	**Type**	**Description & Examples**
Text	string	The string that has to be trimmed. The sample string has both: 1 leading and 1 trailing whitespace. Example: `" This is a sample text! "`
TrimStart	boolean optional default: `false`	Decides whether or not leading whitespaces have to be trimmed. Options: • `enabled` = checkbox is active: leading whitespaces are trimmed. • `disabled` = checkbox is not active: leading whitespaces are not trimmed.
TrimEnd	boolean optional default: `false`	Decides whether or not trailing whitespaces have to be trimmed. Options: • `enabled` = checkbox is active: trailing whitespaces are trimmed. • `disabled` = checkbox is not active: trailing whitespaces are not trimmed.

Regular Expressions

Regular expressions (abbreviated to RexEx or RegExp) are sequences of characters forming a search pattern that can be used to identify a textual material of a given pattern. The syntax of regular expression is standardized and used along many different products and operating systems, particularly for find and replace functionalities.

RayQC uses regular expressions to provide a flexible way of defining compact text matching conditions, which can used to determine whether or not a specific test result is tolerable.

The following cheat sheet presents some commonly used regular expression patterns and example patterns with explanation.

Phrase	Description	Examples
Letters or digits	Literal meaning, case sensitive	`Abc` matching `Abc` but not `abc`
Pipe (`|`)	Alternative	`A|b` matching `A`, `b`, `Ab` but not `BC`
Asterisk (`*`)	Zero or more instances of the previous set	`A*b` matching `Aaaaab`, `Ab`, `bcd` but not `Ac` `A|b*` matching `a`, `b`, `bb`, `bbb` but not `CDE`

Plus (`+`)	One or more instances of the previous set	`A+b` matching `Ab`, `AAAb` but not `b`
		`(a|b)+cd` matching `acd`, `bcd`, `aaacd`, `abcd`, `bacd` but not `cd`
		`Folder[0-9]+` matching `Folder1`, `Folder2`, `Folder21` but not `Folder` or `FolderA`

Question mark (`?`)	Zero or one instance of the previous set	`Ab?(c|d)` matching `Ac`, `Ade`, `Abcd` but not `Abb`
Round brackets `(` and `)`	Grouping of sets	`(abc|def)ghi` matching `abcghi`, `defghi`, `abcdefghi` but not `abcdef` `(a|b|c)?def` matching `adef`, `cdefghi`, `def` but not `abc`
Caret (`^`)	On the beginning of the string marks that no characters are allowed before	`^abc` matching abc, abcdef but not defabc `^(folder1|folder2|folder3)\\test` matching `folder1\test`, `folder2\test\test2` but not `C:\folder1\test`
Dollar (`$`)	At the end of the string marks that no characters are allowed after	`:\\test\\folder$` matching `C:\test\folder`, `D:\test\folder` but not `C:\test\folder1`

Square brackets `[` and `]`	Allowed set of characters	`[a-z]` matching `abc`, `def` but not `123`
		`^[A-D]:\\Test\\` matching `C:\Test\` and `D:\Test\` but not `E:\Test\`
		`folder[a-zA-Z0-9]` matching `folderA`, `folder1` but not `folder\test`

Caret in square brackets `[^]`	In square brackets – negation of character set	`C:\\folder[^\\]*\\test` will match `C:\folder\test` but not `C:\folder\subfolder\test`
Dot (`.`)	Any character	`Folder.\\test` will match `FolderA\`, `Folder4\`, `Folder$\` but not `Folder\`

Backslash (`\`)	Escape character	`C:\\test\\test2` matches `C:\test\test2` but not `C:\\test\\test2`
		`Test\*` matches `Test*`
		`Test\(a\)` matches `Test(a)` but not `Testa`

`(?i)`	Used at the beginning on the expression – do not match case	`(?i)abc` will match `abc`, `Abc`, `ABC123` but not `123`

Examples

```
^(?i)(%windir%)\\Installer$
```

Matches	Does not match
%windir%\Installer %windir%\INSTALLER	%windir%\Installer\123-123.msi

```
^(?i)%ProgramFiles\(x86\)%\\Common Files\\(InstallShield|Wise Installation)$
```

Matches	Does not match
%ProgramFiles(x86)%\Common Files \InstallShield %ProgramFiles(x86)%\COMMON FILES \Wise Installation	%ProgramFiles(x86)%\Common Files\Microsoft %ProgramFiles%\Common Files\InstallShield %ProgramFiles(x86)%\Common Files\ %ProgramFiles(x86)%\Common Files\ Wise Installation\Subfolder

```
^unins(|t|tall).*\.(cif|cfg|dat|dll|ini|exe|xml|lnk)$
```

Matches	Does not match
uninst.exe uninst_myApp.lnk uninstallapp.cfg uninst.ini	MyProgram_uninstall.cfg uninstall.txt unins.txt

```
^_isreg32\.dll$
```

Matches	Does not match
_isreg32.dll	_ISREG32.dll isreg32.dll _isreg.dll _isreg32.dll.backup

```
^(?i)(HKEY_LOCAL_MACHINE|HKEY_CURRENT_USER)\\(Software\\Wow6432Node|Software)
\\InstallShield
```

Matches	Does not match
HKEY_LOCAL_MACHINE\Software\InstallShield HKEY_LOCAL_MACHINE\Software\Wow6432Node \InstallShield HKEY_CURRENT_USER\SOFTWARE\Wow6432Node \InstallShield HKEY_CURRENT_USER\Software\INSTALLSHIELD	HKEY_CLASSES_ROOT\Software\InstallShield HKEY_CURRENT_USER\Software \Progrems\InstallShield

```
^%USERPROFILE%\\(.+\\)?Temp$
```

Matches	Does not match

`%USERPROFILE%\Temp`
`%USERPROFILE%\test\Temp`
`%USERPROFILE%\test\test2\Temp`
	`%USERPROFILE%\Temp2`
	`"%USERPROFILE%\Temp"`
	`%userprofile%\Temp`

MSI Plug-In

The MSI plug-in deals with functionality required to read information from MSI packages, allows validating MSI packages against ICE rule sets, and is able to extract the files contained within MSI packages.

Function Summary

- ExtractFiles
- GetProperty
- GetSummaryInfo
- IceValidation
- Query

Function Details

ExtractFiles

Extracts the file resources from an MSI to a temporary folder and returns the folder name.
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
MSIFile	formatted string	The name and path of the MSI file. Example: `C:\Temp\Sample.msi`
ReUse Data	boolean string optional default value: true	The path to the folder that contains the extracted files is determined by the SHA-1 hash value of the MSI file. Options: • `enabled`: If the temporary extraction folder already exists, it will be reused. • `disabled`: The temporary extraction folder will be cleared if it already exists, and populated with the current set of extracted files.
MST File	formatted string	• The name and path of an optional MST file that may

	optional default value: none	extend the MSI package. • Multiple transform files can be applied using a comma separator. Example: `C:\Temp\German.mst`

GetProperty

Data Field | Multi-Option

Looks for the value of a property in an MSI package. This means that according to the given property name, the content of the value column is queried from the Property table of the installer database.
Usable in combination with elements of type Data Field and Multi-Option.

Input parameters		
Name	**Type**	**Description & Examples**
MSIFile	formatted string	The name and path of the MSI file. Example: `C:\Temp\RayEval.msi`
MSTFile	formatted string optional default value: none	• The name and path of an optional MST file that may extend the MSI package. • Multiple transform files can be applied using a comma separator. Example: `C:\Temp\German.mst`
Property	string	Name of the property whose value has to be retrieved. Example: `Manufacturer`

GetSummaryInfo

Data Field | Multi-Option

Requests an attribute from the summary information stream of an MSI package.
Usable in combination with elements of type Data Field and Multi-Option.

Input parameters		
Name	**Type**	**Description & Examples**
MSIFile	formatted string	The name and path of the MSI file. Example:

		`C:\Temp\RayEval.msi`
MSTFile	formatted string optional default value: none	• The name and path of an optional MST file that may extend the MSI package. • Multiple transform files can be applied using a comma separator. Example: `C:\Temp\German.mst`
Attribute	string	Name of the attribute whose value has to be retrieved. This parameter has to be selected from the default pool of attributes: Options: • `Author` • `Comments` • `CreatingApp` • `Keywords` • `LastSavedBy` • `RevisionNumber` • `Subject` • `Template` • `Title` • `CharacterCount` • `CodePage` • `CreateTime` • `LastPrintTime` • `LastSaveTime` • `PageCount` • `Security` • `WordCount`
Date/Time Format	formatted string optional default value: dd.MM.yyyy HH:mm:ss	If the requested attribute is a date or date-time combination, the selected format schema will be applied. Options: • `d.MM.yyyy HH:mm:ss` • `dd.MM.yyyy` • `MM/dd/yyyy HH:mm:ss` • `MM/dd/yyyy`

IceValidation

Validates an MSI package by applying checks as defined by the rules of the parametrized CUB file.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
MSI File	formatted string	The name and path of the MSI file.

		Example: `C:\Temp\RayEval.msi`
MST File	formatted string optional default value: none	• The name and path to an optional MST file that may extend the MSI package. • Multiple transform files can be applied using a comma separator. Example: `C:\Temp\German.mst`
Cub Filename	formatted string	The name and path of the CUB file that contains the rules to validate the MSI against. Example: `C:\Temp\Darice.cub`
Suppressed Checks	formatted string optional default value: none	Comma separated list of ICE rule names, which are present in the CUB file, but should not be applied during validation. Example: `ICE01,ICE33,ICE64`
MaxWarnings	integer optional default value: -1	Maximum number of warnings that may occur during the ICE validation without causing the plug-in result to turn false. Options: • `-1`: The number of warnings that arise during the ICE validation does not have any effect on the checkpoint result. • `0`: Any CUB rule evaluation that reveals a warning causes the plug-in result to turn false. • `Any other integer > 0`: As long as the number of warnings that arise during ICE validation is below the entered value, the checkpoint result evaluates to true.
MaxErrors	integer optional default value: 0	Maximum number of errors that may occur during the ICE validation without causing the plug-in result to turn false. Options: • `-1`: The number of errors that arise during the ICE validation does not have any effect on the checkpoint result. • `0`: Any CUB rule evaluation that reveals an error causes the plug-in result to turn false. • `Any other integer > 0`: As long as the number of errors that arise during ICE validation is below the entered value, the checkpoint result evaluates to true.

Query

Data Field | Multi-Option

Executes a reading query on the installer database. The default query schema is defined as:

```
SELECT item FROM table WHERE key = value
```

Please note that the supported set of SQL operators and language constructs is limited for installer database queries, which is why the query function is limited to a straight forward select statement.

The result set of the query execution may contain 0,1, or several data objects.

Usable in combination with elements of type Data Field and Multi-Option.

Input parameters

Name	Type	Description & Examples
MSIFile	formatted string	The name and path of the MSI file. Example: `C:\Temp\RayEval.msi`
MSTFile	formatted string optional default value: none	• The name and path of an optional MST file that may extend the MSI package. • Multiple transform files can be applied using a comma separator. Example: `C:\Temp\German.mst`
Select	string	SQL expression that determines the data column that has to be retrieved from the installer database. Example: `Action`
From	string	SQL expression determining the table that contains the information that has to be read. Example: `CustomAction`
Where	string optional default value: none	SQL expression that determines a data column that is used as base for the equative result set filtering. If the Equals parameter is not empty, Where must be provided as well. Example: `Type`
Equals	string optional default value: none	Value used to filter the query result set. It has to be present within the column specified by the Where parameter. If the Where parameter is not empty, Equals must be provided as well.

Example:
`19`

RayFlow Plug-In

The RayFlow plug-in deals with functionality required to establish communication between the RaySuite components RayQC and RayFlow.

Function Summary

- AddComment
- GetComments
- GetConnectionInfo
- GetField
- GetFile
- GetPackage
- SetField
- UploadFile

Function Details

AddComment

Adds a comment to a RayFlow package (task). Returns **Yes** if the comment has been added successfully.

Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
RayFlow Package ID	string	The ID of a RayFlow package (task). Example: `123e4567-e89b-12d3-a456-426655440000`
Comment	string	A text that will be written as a comment. Example: `This is a comment from RayQC.`

GetComments

Data Field

Gets comments from a RayFlow package (task).
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
RayFlow Package ID	string	The ID of a RayFlow package (task). Example: `123e4567-e89b-12d3-a456-426655440000`

GetConnectionInfo

Data Field

Requests one of the parameter values from the current RayFlow connection settings.
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
Connection Information	formatted string	The specific property of the connection credentials that has to be retrieved. Options: • `User` • `URL` • `Package ID` • `Project ID`

GetField

Data Field Multi-Option

Request a value from the RayFlow data object referenced by the current connection and therefore requires an active connection to the RayFlow database server.
Usable in combination with elements of type Data Field and Multi-Option.

Input parameters		
Name	**Type**	**Description & Examples**
RayFlowField	string	The name of the RayFlow data field that has to be retrieved. Example: `Manufacturer`

RayFlowPackageID	string	The ID of the RayFlow package, usually delivered with the `GetPackage` plug-in. Example: `c32715f5-186d-49c7-b998-30f5d50v95fb`

GetFile

Downloads the specified file from RayFlow server. Returns the full path to the downloaded file.
Usable in combination with elements of type Data Field.

Input parameters		
Name	**Type**	**Description & Examples**
RayFlow Package ID	string	The name of the RayFlow data field that has to be retrieved. Example: `Manufacturer`
Download file	boolean default: `true`	Indicates whether a file should be downloaded or only the existence of this file should be verified. • **Enabled** = the file will be downloaded. • **Disabled** = the file will verified on RayFlow server. If the file will be found the file name is returned, otherwise `false` is displayed.
Output directory	string optional	The full path to the output directory, where file will be downloaded. If the path is not set, the file will be saved in the `%TEMP%` folder. Example: `C:\Downloads`
File name	string	The exact name of the file OR the regular expression that should be used to find requested file. Example: `FileZilla_1.0.0.mst`
Match name as regular expression	boolean default: `false`	Indicates whether the **File name** parameter uses a regular expression. • **Enabled** = regular expression specified in **File name** parameter will be used to fetch the correct file to download. • **Disabled** = regular expression will not be used when fetching the file name.
RayFlow data field ID/name	string optional	The name of ID of a data field that stores a name of a file to download.

		Example: `MsiTransform`
Use field ID instead of field name	boolean default: `true`	Indicates whether the **RayFlow data field ID/name** parameter should use the field ID instead of field name. • **Enabled** = use data field ID • **Disabled** = use data field name

GetPackage

Request a package ID from RayFlow and therefore requires an active connection to the RayFlow database server.
Usable in combination with elements of type Data Field.

This function does not accept parameters.

SetField

Updates a value of the RayFlow data object referenced by the current connection, and therefore requires an active connection to the RayFlow database server.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
RayFlowField	string	The name of the RayFlow data field that has to be updated. Example: `Manufacturer`
Value	string	The value that will be written into the field defined above. Example: `Raynet GmbH`
RayFlowPackageID	string	The ID of the RayFlow package, usually delivered with the `GetPackage` plug-in. Example: `c32715f5-186d-49c7-b998-30f5d50v95fb`

UploadFile

Uploads a file to RayFlow. This affects the RayFlow data object referenced by the current connection, and therefore requires an active connection to the RayFlow database server.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
File	formatted string	The path to the file that has to be uploaded. Example: `C:\Temp\Sample.pdf`
RayFlowPackageID	string	The ID of the RayFlow package, usually delivered with the `GetPackage` plug-in. Example: `c32715f5-186d-49c7-b998-30f5d50v95fb`

Registry Plug-In

The Registry plug-in deals with functionality required to retrieve information about registry hives, keys, and values.

Function Summary

- EntryExists
- GetType
- GetValue
- KeyExists
- ListOfSubkeys

Function Details

EntryExists

Checks if a registry entry exists. If the entry is not found, the returned result is **No**, otherwise **Yes**.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
64 Bit Architecture	boolean default: true	The architecture specific area definition of the registry search. Options: • `enabled =` the entry is searched within the 64bit related areas of the registry.

		• `disabled` = the entry is searched within the 32bit related areas of the registry.
Hive	formatted string	The registry hive in which RayQC searches for the key and entry defined later. Options: • `HKEY_CLASSES_ROOT` • `HKEY_CURRENT_USER` • `HKEY_LOCAL_MACHINE` • `HKEY_USERS` • `HKEY_CURRENT_CONFIG`
Key	string	The name of the key in which RayQC searches for the entry defined later. Example: `Software`
Entry	string	The name of the entry RayQC searches for. (Often referred to as "value", even though it seems a bit confusing that a value may have a value itself, which is why the registry plug-in searches for entries, and not values.) Example: `RayPack`

GetType

Gets the type of a registry key.
Usable in combination with elements of type Data Field and Multi-Option.

Input parameters		
Name	**Type**	**Description & Examples**
64 Bit Architecture	boolean default: true	The architecture specific area definition of the registry search. Options: • `enabled`: the entry is searched within the 64bit related areas of the registry. • `disabled`: the entry is searched within the 32bit related areas of the registry.
Hive	formatted string	The registry hive in which RayQC searches for the key and entry defined later. Options: • `HKEY_CLASSES_ROOT` • `HKEY_CURRENT_USER`

		• HKEY_LOCAL_MACHINE • HKEY_USERS • HKEY_CURRENT_CONFIG
Key	string	The name of the key in which RayQC searches for the entry defined later. Example: Software
Entry	string	The name of the entry RayQC searches for. (Often referred to as "value", even though it seems a bit confusing that a value may have a value itself, which is why the registry plug-in searches for entries, and not values.) Example: RayPack

GetValue

Gets a registry value.
Usable in combination with elements of type Data Field and Multi-Option.

Data Field | Multi-Option

Input parameters		
Name	**Type**	**Description & Examples**
64 Bit Architecture	boolean default: true	The architecture specific area definition of the registry search. Options: • enabled: the entry is searched within the 64bit related areas of the registry. • disabled: the entry is searched within the 32bit related areas of the registry.
Hive	formatted string	The registry hive in which RayQC searches for the key and entry defined later. Options: • HKEY_CLASSES_ROOT • HKEY_CURRENT_USER • HKEY_LOCAL_MACHINE • HKEY_USERS • HKEY_CURRENT_CONFIG
Key	string	The name of the key in which RayQC searches for the entry defined later. Example: Software

Entry	string	The name of the entry whose value has to be returned. This is an optional parameter and if empty then it requests the default value. (Often referred to as "value", even though it seems a bit confusing that a value may have a value itself, which is why the registry plug-in searches for entries, and not values.) Example: `RayPack`

KeyExists

Checks if a registry key exists. If the key is not found, the returned result is **No**, otherwise **Yes**.
Usable in combination with elements of type Checkpoint.

Input parameters		
Name	**Type**	**Description & Examples**
64 Bit Architecture	boolean default: true	The architecture specific area definition of the registry search. Options: • `enabled`: the entry is searched within the 64bit related areas of the registry. • `disabled`: the entry is searched within the 32bit related areas of the registry.
Hive	formatted string	The registry hive in which RayQC searches for the key defined later. Options: • `HKEY_CLASSES_ROOT` • `HKEY_CURRENT_USER` • `HKEY_LOCAL_MACHINE` • `HKEY_USERS` • `HKEY_CURRENT_CONFIG`
Key	string	The name of the key whose existence has to be checked. Example: `Software`

ListOfSubkeys

Gets a list of subkeys for a given registry key.
Usable in combination with elements of type Data Field and Multi-Option.

Input parameters		
Name	**Type**	**Description & Examples**
64 Bit Architecture	boolean default: true	The architecture specific area definition of the registry search. Options: • `enabled`: the entry is searched within the 64bit related areas of the registry. • `disabled`: the entry is searched within the 32bit related areas of the registry.
Hive	formatted string	The registry hive in which RayQC searches for the key and entry defined later. Options: • `HKEY_CLASSES_ROOT` • `HKEY_CURRENT_USER` • `HKEY_LOCAL_MACHINE` • `HKEY_USERS` • `HKEY_CURRENT_CONFIG`
Key	string	The name of the key in which RayQC searches for the entry defined later. Example: `Software`
Entry	string	The name of the entry RayQC searches for. (Often referred to as "value", even though it seems a bit confusing that a value may have a value itself, which is why the registry plug-in searches for entries, and not values.) Example: `RayPack`

Advanced Plug-In

Allows a user to select target package for collision test. A wizard called **Select Collision Testing Targets** is launched when a checkpoint containing this plug-in is executed. This wizard lists the packages that are stored in the RayQC *Advanced* catalog.

Function Summary

- GetCollisionTargets
- GetRulesets
- StartTestScenario

Function Details

GetCollisionTargets

Allows a user to select target package for collision test. A wizard called **Select Collision Testing Targets** is launched, when a checkpoint containing this plug-in is executed. This wizard catalogs the packages stored in RayQC *Advanced* database.
This plug-in is only usable with an element of type Data Field.

GetRulesets

Launches the **Select Rulesets** wizard in which a ruleset can be selected to execute different test types including Collision, Virtualization, and System Readiness.
This plug-in is only usable with an element of type Data Field.

StartTestScenario

Starts the test scenario based upon the input parameters. After executing the test, this plug-in returns the test result along with a path to the report file.
This plug-in is only usable with an element of type Checkpoint.

Input parameters

Name	Type	Description & Examples
MSI path	string	The name and path of the source package for testing Example: `C:\Users\Administrator\Desktop\RayTest.msi`
Ruleset ID	formatted string optional: true	Identifier of the test ruleset to be executed upon the source package. Use angle brackets for a description (they are ignored by the parser, only the number is interpreted). Example: 1 <Collision>
MST path	string optional: true	The name and path of the MST file. Example: `C:\Users\Administrator\Desktop\Raytest.mst`
Collision target	formatted string optional: true	Path to the target package for collision test. Use angle brackets for a description (they are ignored by the parser, only the number is interpreted). Example: `db:8 <Skype 10.0>`

External Plug-Ins

RayQC is a tool for both: static and dynamic test procedure bundling, based on manually or automatically executed test steps which are organized in easily maintainable checklists. Some basic logic for system checks and manipulations come along with the RayQC application itself, as so called internal plug-ins. Each of these plug-ins has a clear public interface of functions, along with input and output parameters. With a decent amount of creativity, checklist authors can actually define collections of test steps, which perform a broad variety of automated tests by simply configuring and combining these internal plug-ins. A good share of common test logic requirements can be covered without any additional customization.

However, Raynet knows how flexible evaluators have to react and adjust in order to be able to meet ever changing demands on quality, performance and efficiency. Therefore, RayQC contains an interface for the integration of custom plug-in logic that extends the services provided by internal plug-ins as far as the creativity and PowerShell programming skills of checklist creators may carry them.

The communication between external plug-ins and RayQC has to be standardized, since all RayQC instances of a specific RayQC product version should be able to work with any external plug-in. This is not only a core demand of enterprise wide teamwork, but also helps to maintain compatibility and migration paths over several product versions with a manageable amount of workload.

External plug-in logic has to be written in PowerShell, and introduced to the RayQC plug-in interface by an XML based manifest file. Please refer to the upcoming sections for details on these specific files and their requirements.

Be aware:
In order to be able to use external plug-ins with RayQC, it has to be ensured that the PowerShell version supported by the device that hosts the application matches the PowerShell version of the actual plug-in script. It is highly recommended to synchronize the PowerShell version among all devices that are assigned for QA execution to prevent compatibility issues in the first place. RayQC currently supports PowerShell version 3.0 and higher.

Structure of a Plug-In

Each external plug-in consists of a logic part, which has to be written in Powershell, and a declarative part, provided as XML manifest file. These file types always have to be considered as a dynamic duo that should never fall apart. Therefore, it is usually required to update both files when the plug-in function is changed, e. g. by adding a new function or modifying function signatures.

There are some properties that have to be defined clean and clear for each plug-in, in order to allow RayQC to recognize the plug-in resource bundle and integrate it into the checklist workflows:

Plug-In Name

Each external plug-in must have a name that is unique within the current plug-in context. The plug-in name is also the name of the directory which contains the plug-in script logic and manifest files. RayQC searches for these resource structures to build the set of available plug-ins for a specific checklist when it is opened for evaluation within the Checklist Viewer or for structural manipulations within the Checklist Editor.

Plug-In Context

The plug-in context is the environment within reach of the RayQC application instance used to build or evaluate plug-in augmented checklists. Usually it is limited to the local plug-ins sub-directory for a checklist template, along with the global plug-ins sub-directory within the RayQC application instance root folder (usually

something like `C:\Program Files (x86)\RayQC\`).
Since RayQC is able to work with shared checklist resources from spread network locations, this context scope is quite dynamical and can be changed with every checklist transfer operation.

Plug-In Version

Another important plug-in property is the version, which has to be incremented whenever the plug-in sources are updated. It is documented within the manifest file, and used by RayQC to determine if a copy of the parsed plug-in logic that resides within the session memory is still up to date compared to the plug-in version stored on the physical file system.

Even though it is not demanded by technical restrictions, it is highly recommended to work with versions with different level indicators regarding their update status. Simply increasing version numbers as 1, 2, 3, 4 may surely do. However, providing additional information regarding the grade of manipulation may help to keep plug-in resources easily maintainable. The suggested plug-in version structure is `[Major Upgrade].[Minor Update].[Small Revision]`.
Especially in extended QA teams it is very handy to provide this kind of meta information for shared resources if they are not explicitly managed by software versioning and revision control systems.

Plug-In Signature

This signature is built by the combination of plug-in name and version. Regarded as a clear identifier for a specific plug-in state, the context wide Best Practice constraint for unique plug-in signatures may save from severe maintenance troubles that might occur if plug-ins are not well aligned.

Just imagine the confusion caused by different version states and identical copies of the same plug-in stored in varying local and global `\plug-ins\` directories: What happens to a checklist that is copied from one repository branch to another? Will the local version of the plug-in provided within the new context situation match the version that has been used in the former context? Or does the user have to copy the plug-in resources along? What if a plug-in with the same name but different version already exists in the new context?

Be aware:
It is highly recommended to define an enterprise wide guideline for the provision of well aligned plug-in management. It should contain clear rules regarding versioning, naming, copying, and adjusting plug-ins. All users with access to the set of checklists and plug-ins should be aware of these guidelines and follow their terms and conditions.

The Manifest

From a checklist creator's point of view, a plug-in is determined by its name, available functions and their parameters. Especially for internal plug-ins, this set of information is all a checklist author may access. (Well, actually he has additional information available from this document, but who provides a decent user guide for the set of custom plug-ins established within an enterprise QA department?)

The manifest file of a plug-in has to define this set of plug-in properties in a human as well as system readable format. This is why Raynet decided to use XML as manifest file markup language, since it is highly structured and maintains intuitive accessibility for checklist authors.

There are several angles from which manifest files may be explored: from the naked XML structure with its tags and attributes, or from the logical and functional requirements regarding information needed for the interface definition. The following section describes the informative scope, whilst the Appendix section provides access from the other side.

Plug-In Interface Definition Requirements

Each manifest file represents a specific group of functions and arguments that are somehow related to each other. From this point of view a plug-in is nothing more than a container for standardized program logic.

Each plug-in needs a certain set of information to be clearly identifiable. These information includes Name, Version, Filename, and PowerShell version which supports the related PowerShell script. These first set of information is needed to be plainly visible from the manifest.

```
  <Name>PowerShellSample</Name>
  <Description>This a pretty straight external plug-in with two sample function for
demonstration.</Description>
  <Version>1.2.3.4</Version>
  <Filename>PowerShellSample.ps1</Filename>
  <PowerShellVersion>3</PowerShellVersion>
```

For this sample manifest file the plug-in name is `PowerShellSample`, it has `1.2.3.4` as its version. Furthermore this manifest file links to the `PowerShellSample.ps1` script which is supported by PowerShell version `3.0` and higher.

Alright, the foundation is there - RayQC can recognize the plug-in. But how to communicate what the plug-in is capable of? Well, what is usually required from scripted logic is a classical sequence of using some pieces of information, processing them somehow, and giving feedback about the result of the processing. So, each of these sequences is a capsuled and clearly defined functional group. We interpret plug-ins as containers for functionality, so a plug-in with only one function may exist, but to be true is a poor container. Let's be generous and assume that there has to be at least one function, but authors are free to add as many as pleases them.

Combining these properties of plug-in functions and its parameters, the translation to XML is quite obvious: Each

function description is embraced by a parent tag, whilst these have to be parts of the plug-in themselves.

```
<Name />
  <Description />
  <Version />
  <Filename />
  <PowerShellVersion />
  <Functions>
    <FunctionParameters>
      <FunctionName />
      <Description />
      <Parameters />
    </FunctionParameters>
  </Functions>
</plug-inData>
```

There seem to be some important facts missing in the sample XML given above. The functions and their parameters are not named, so how could RayQC know which sub-tree to enter when a checklist author selects a plug-in function from the Checklist Editor interface? And up to now there is no reference to the actual source code with the function logic (the part that actually uses the input to somehow generate output).

Well, each function and its parameter needs an identifier (a unique one per function, that is easy to find when a user reads the manifest - it seems to be a good idea to set it as an attribute for the function tag).

```
  <Name>PowerShellSample</Name>
  <Description>This a pretty straight external plug-in with two sample function for
demonstration.</Description>
  <Version>1.2.3.4</Version>
  <Filename>PowerShellSample.ps1</Filename>
  <PowerShellVersion>3</PowerShellVersion>
  <Functions>
    <FunctionParameters>
      <FunctionName>TestMeOne</FunctionName>
      <Description>PowerShell Function One</Description>
      <Parameters>
        <BooleanParameter>
          <Name>param1</Name>
          <Description>Expects a boolean input (true/false).</Description>
          <IsOptional>false</IsOptional>
          <Default>true</Default>
        </BooleanParameter>
        <StringParameter>
          <Name>param2</Name>
          <Description>Expects a string that matches the regular expression
'{0}'.</Description>
          <IsOptional>false</IsOptional>
        </StringParameter>
      </Parameters>
    </FunctionParameters>
  </Functions>
</plug-inData>
```

That looks a lot better, especially the `name` and `description` attribute we added for each function and its parameters is quite handy to have.Additionally we have also defined the parameter type, which in turn defined the type of input value that is expected by the script.

Additional XML Definitions

Well... we are almost done. There is a tiny task of defining the function logic which has not been performed yet.

```
<?xml version="1.0" encoding="utf-8"?>
<plug-inData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Name>PowerShellSample</Name>
  <Description>This a pretty straight external plug-in with two sample function for
demonstration.</Description>
  <Version>1.2.3.4</Version>
  <Filename>PowerShellSample.ps1</Filename>
  <PowerShellVersion>3</PowerShellVersion>
  <Functions>
    <FunctionParameters>
      <FunctionName>TestMeOne</FunctionName>
      <Description>PowerShell Function One</Description>
      <Parameters>
        <BooleanParameter>
          <Name>param1</Name>
          <Description>Expects a boolean input (true/false).</Description>
          <IsOptional>false</IsOptional>
          <Default>true</Default>
        </BooleanParameter>
        <StringParameter>
          <Name>param2</Name>
          <Description>Expects a string that matches the regular expression
'{0}'.</Description>
          <IsOptional>false</IsOptional>
        </StringParameter>
      </Parameters>
    </FunctionParameters>
  </Functions>
</plug-inData>
```

More information regarding the utilization of these attributes will be provided later.
So let's save the work done up to now as `Manifest.xml` and then proceed to the next section: The plug-in Logic Script.

Tip:
Preparing the manifest files is as easy as pie with a decent XML editor at one's side. The editor should provide code highlighting and optional support for XML schema definition assistance. There are many free XML editors available online, and most software development suites carry more or less sophisticated XML support along. Feel free to pick an editor of choice.

Note:
Please refer to the Appendix of this document for details regarding formal restrictions for the plug-in manifest file.

The Plug-In Logic Script

The logic performed when a plug-in function is triggered from a checklist needs to come in a strictly tailored dress of standard definitions. Each script must carry definitions and
subroutines to establish a baseline of communication capabilities between the RayQC plug-in interface and the

Checklist Viewer logic resolver.

The Plug-In Execution Procedure in a Nutshell

RayQC supports 2 kinds of plug-ins: Internal and External. The external plug-in type can be further divided into categories: local and global plug-ins. Although the scope of both local and external plug-ins is different,they do share the same plug-in execution logic. Each plug-in consists of a plug-in script and a `manifest.xml` file. The manifest file acts as an interface between the RayQC plug-in execution logic and the plug-in script. When a checklist containing the plug-in is loaded, the checklist is validated against the `checklistschema.xsd` file and the manifest file is validated against the `manifestschema.xsd` file. During execution, depending upon the plug-in logic, input parameters are passed to the plug-in script through the function which is defined in the manifest.xml file. If the function name and its arguments in the manifest file don't match the ones in the script file, an error is thrown. After plug-in execution, the script returns the result value which in turn is mapped to the checklist element, called the plug-in, through the manifest file. For more information on the structure of checklists and its manifest files, refer to the Appendices chapter of this guide.

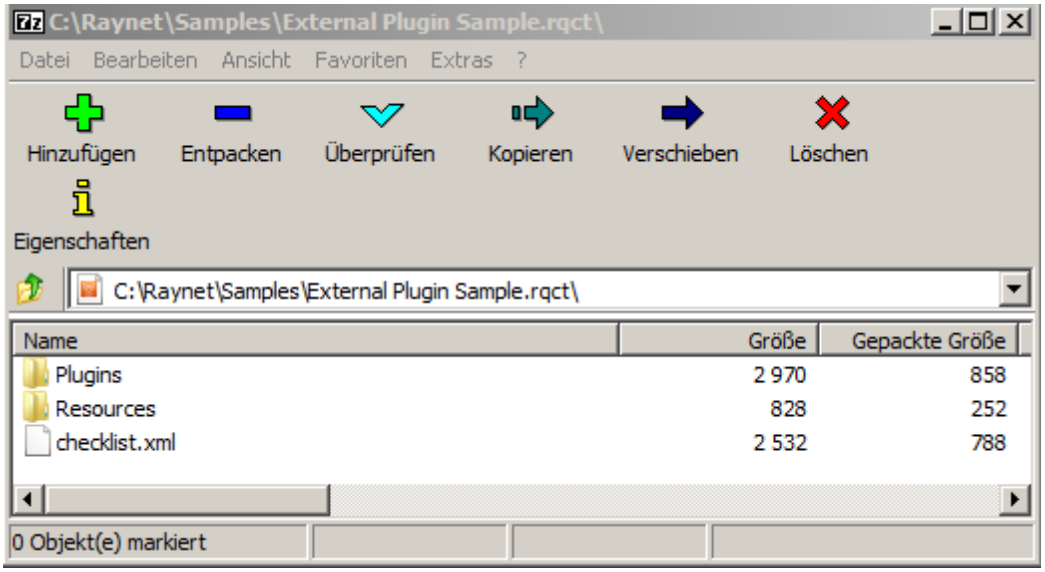
Note:
The following section does not provide a full plug-in script sample, but describes the sections that need to be present. Please refer to the External plug-ins sample available from the application root directory (typically something like `C:\Program Files (x86)\RayQC\Samples\`), and the Appendix of this document for further code details.

Local and Global Plug-Ins

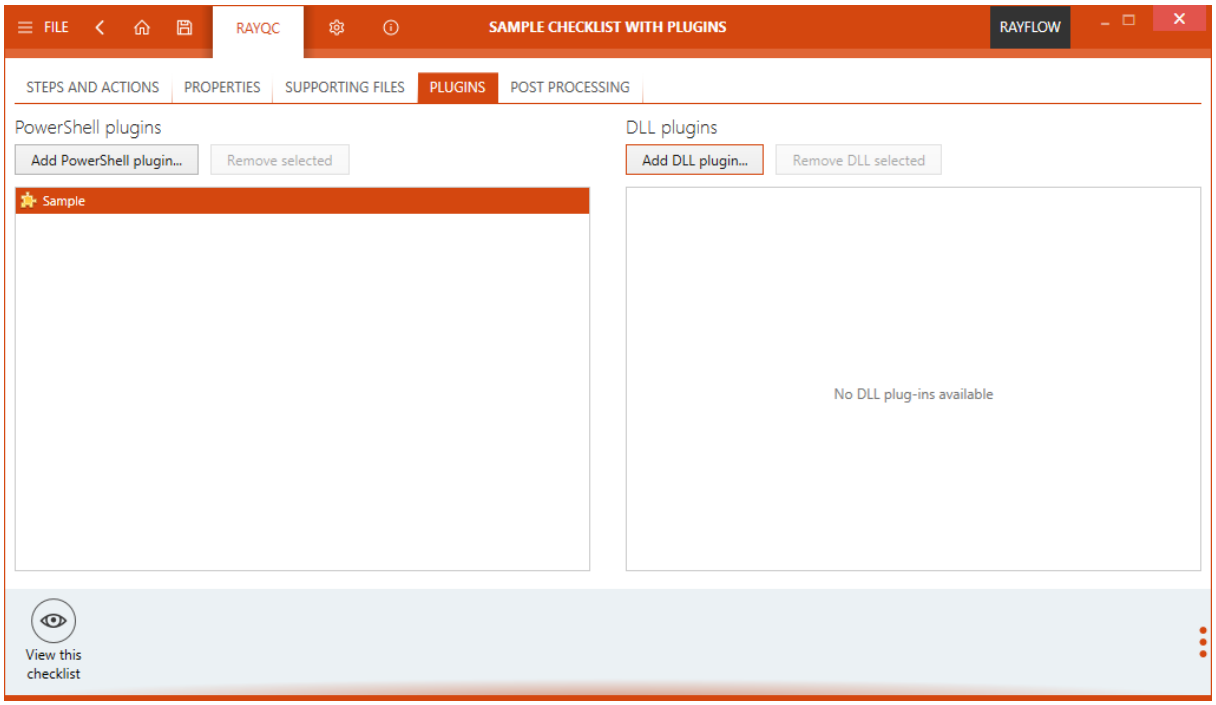
As already outlined before, there are 2 different ways to manage the aspects of storage organization for external plug-ins. Both locations have their specific pro's and con's, which demands a moment of analysis before the location for a newly created plug-in is defined.

Local external plug-ins

Local external plug-ins are stored in the `/plug-ins/` sub-directory within the checklist container file. A user can directly manage these plug-ins via the plug-in manager in the checklist editor.
The screenshot below displays an example of a checklist container file with plug-ins sub-directory, which in turn holds the local plug-in for the respective checklist.



The screenshot below displays the plug-in inside the plug-in manager of the Checklist Editor for the checklist `External plug-in Sample.rqct`.

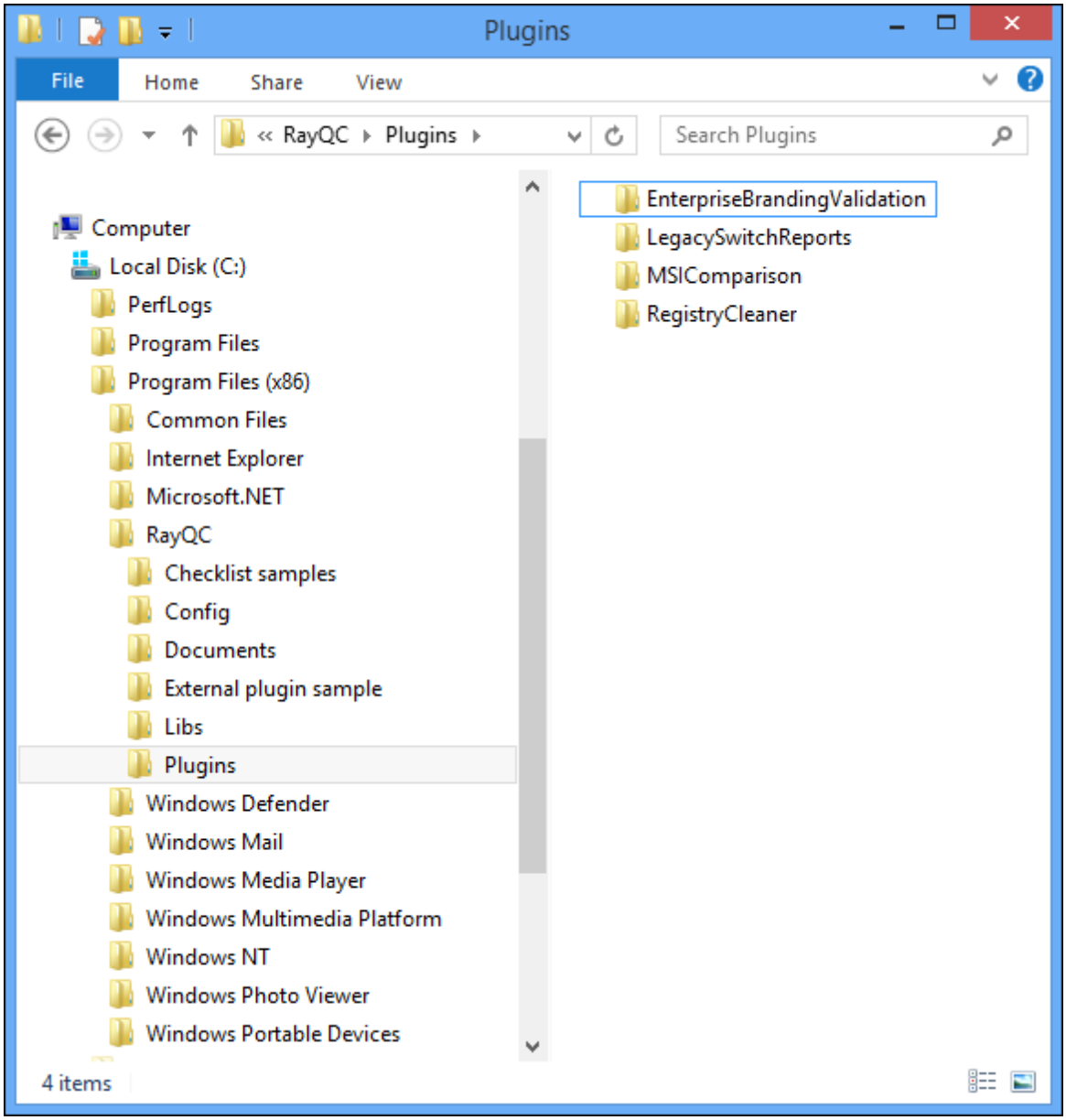


Global External Plug-Ins

RayQC supports two types of global external plug-ins:PowerShell and DLL.These plug-ins are stored in the `\plug-ins\` sub-directory within the RayQC application installation directory (usually something like `C:\Program Files (x86)\RayQC\`). They can be used by any checklist run from the local RayQC instance. If plug-in functionality has to be provided for a large percentage of checklists that have to be performed on a regular basis by several evaluators, it is recommended to store them within the RayQC application installation directory. All checklists that are created or evaluated by this specific RayQC installation may use the external plug-ins stored within the global plug-ins directory.

Be aware:
If there are several versions of the same plug-in available within the local external plug-in folder of a specific checklist, and the global external plug-in of a RayQC instance, there will be conflicts regarding the collection of the plug-in selector options within the Checklist Editor. Additional issues may occur when checklist projects are evaluated based on ambiguous plug-in versions. Since the management of external plug-in resources is part of the RayQC administrator responsibilities, it is highly recommended to define a global guideline for plug-in storage strategies and rules.

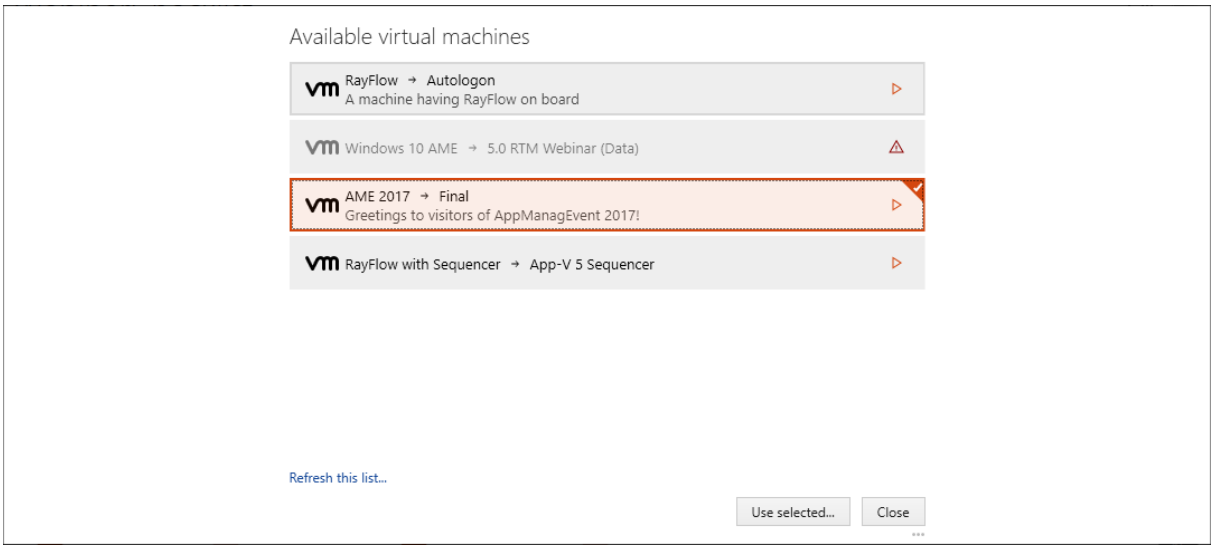
The screenshot below shows a standard explorer window displaying the global plug-in directory of a RayQC installation with a set of globally available custom plug-ins. Each plug-in resides within a sub-folder, which is named according to the actual plug-in name:


Plugins
File
Home
Share
View
« RayQC ▸ Plugins ▸
Search Plugins
Computer
Local Disk (C:)
PerfLogs
Program Files
Program Files (x86)
Common Files
Internet Explorer
Microsoft.NET
RayQC
Checklist samples
Config
Documents
External plugin sample
Libs
Plugins
Windows Defender
Windows Mail
Windows Media Player
Windows Multimedia Platform
Windows NT
Windows Photo Viewer
Windows Portable Devices
EnterpriseBrandingValidation
LegacySwitchReports
MSIComparison
RegistryCleaner
4 items

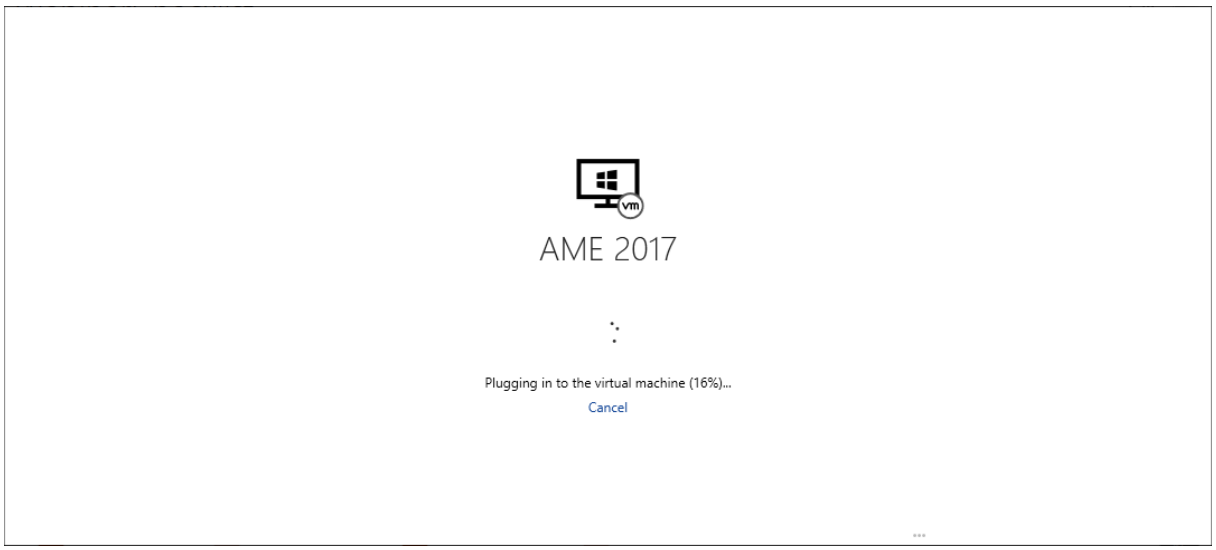
Using Virtual Machines

A checklist, a single group or a single element can be remotely executed on a virtual machines by using the **Virtual machine...** button located in the Swipe-Bar that is available

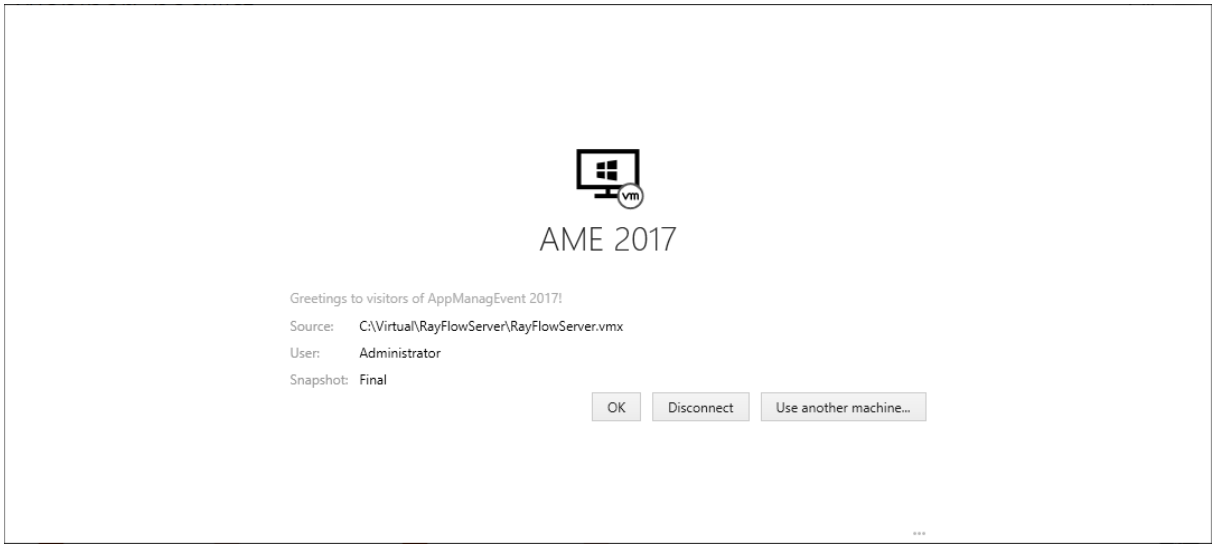
Using this button will open the **Available virtual machine** screen.



In this screen, the virtual machines that shall be used to execute the checklist can be selected. Machines that have been configured but are currently not available are grayed out and marked by a warning symbol. The selected machines are highlighted as shown in the screenshot above. Click on the **Use selected...** button to use the selected machines or click on the **Close** button to return to the checklist on the local machine.

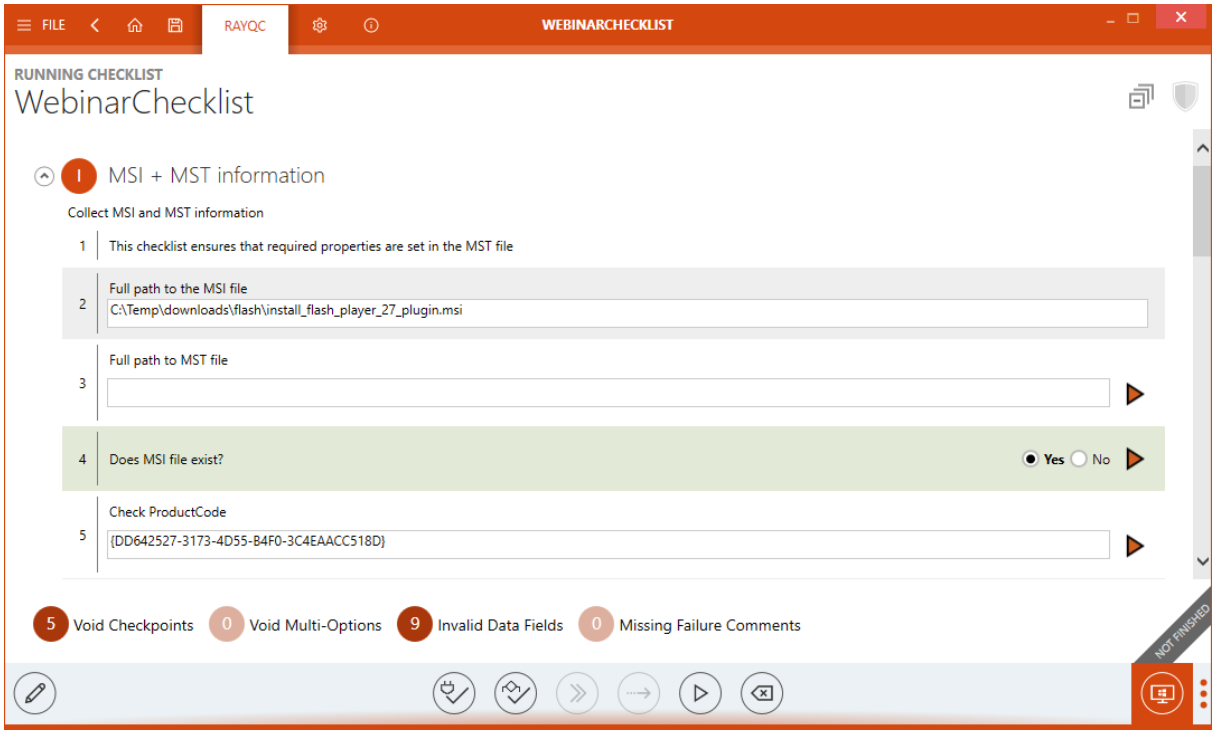


RayQC will now connect to the selected virtual machine. While plugging in to the machine, the action can be aborted by clicking on the **Cancel** link. After the connection has been successfully established, the **Overview** screen for the virtual machine is shown.



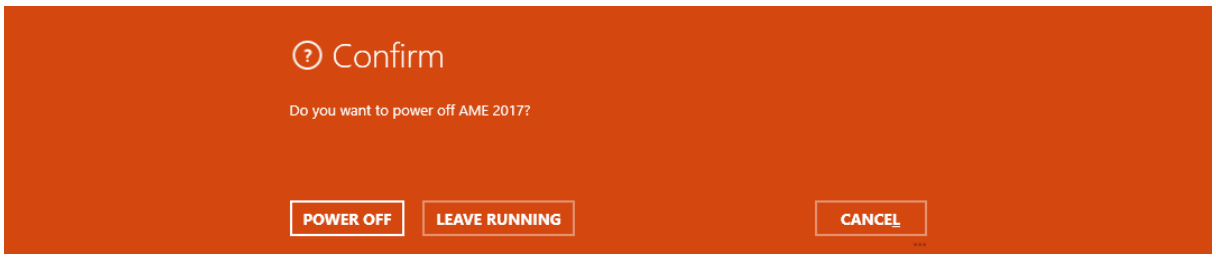
This screen can be used to either change the virtual machine, continue with the checklist, or disconnect. Furthermore, some information about the virtual machine, like path, user, and the selected snapshot are shown in this screen.

Use the **OK** button to continue, the **Disconnect** button to return to the local machine, or the **Use another machine...** button to select another virtual machine from the list of available virtual machines.



After continuing to the checklist it can be used as if working on a local machine. Only the highlighted **Virtual machine...** button shows, that the checklist is currently being used on a virtual machine and not on the local machine.

Clicking on the **Virtual machine...** button will open the **Overview** screen once more, which can be used to disconnect or change the virtual machine. When leaving the currently selected virtual machine, a confirmation screen will be shown.



In the **Confirm** prompt, there are three options available:

- **POWER OFF**: Can be used to power off the virtual machine and return to the checklist on the local machine or select another virtual machine.
- **LEAVE RUNNING**: Can be used to return to the checklist on the local machine or select another virtual machine, but the current virtual machine will not be turned off but stay active.
- **CANCEL**: Is used to abort the action and return to the currently used virtual machine.

Working with RayFlow

Introduction

RayFlow is a workflow process management tool with the ability to support diverse workflow processes. The possibility to be customized to fit the user's needs and requirements makes it one of the most efficient and user friendly workflow management tools. This guide shows how to configure and manage RayFlow, so that IT departments can stay ahead, save time, increase productivity, and decrease IT costs.

RayFlow is based on the client-server architecture in which all the information, data, and configuration is stored on the RayFlow server. Users work on this server remotely through the RayFlow web and Windows-based clients.

Where to find the latest information about RayFlow

For further information on RayFlow, including its features, functionality and latest updates visit www.raynet.de.

Enabling RayFlow features in RayQC and RayQC Advanced

RayFlow connection are stored in profile configuration. The minimal configuration requires entering the URL address containing a valid and running instance of RayFlow server.
See RayFlow configuration for RayQC.

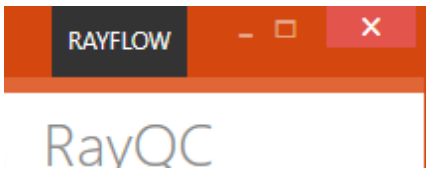
Signing-in to RayFlow

Saving and opening files from RayFlow requires that the current user is signed-in to the RayFlow instance specified in the configuration screen for RayQC and RayQC Advanced.

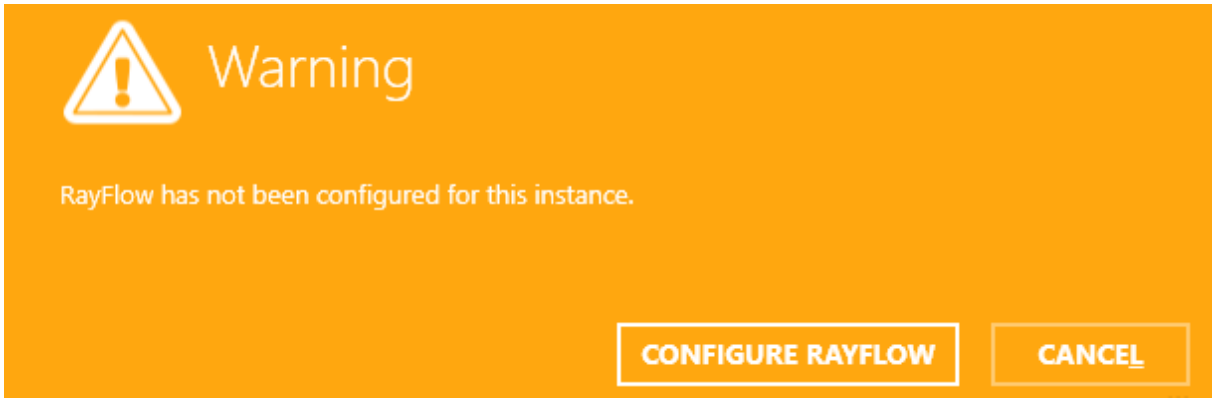
Sign-in procedure makes sure that:

- User has permissions to a specified RayFlow project
- User has permissions to see/edit required RayFlow tasks

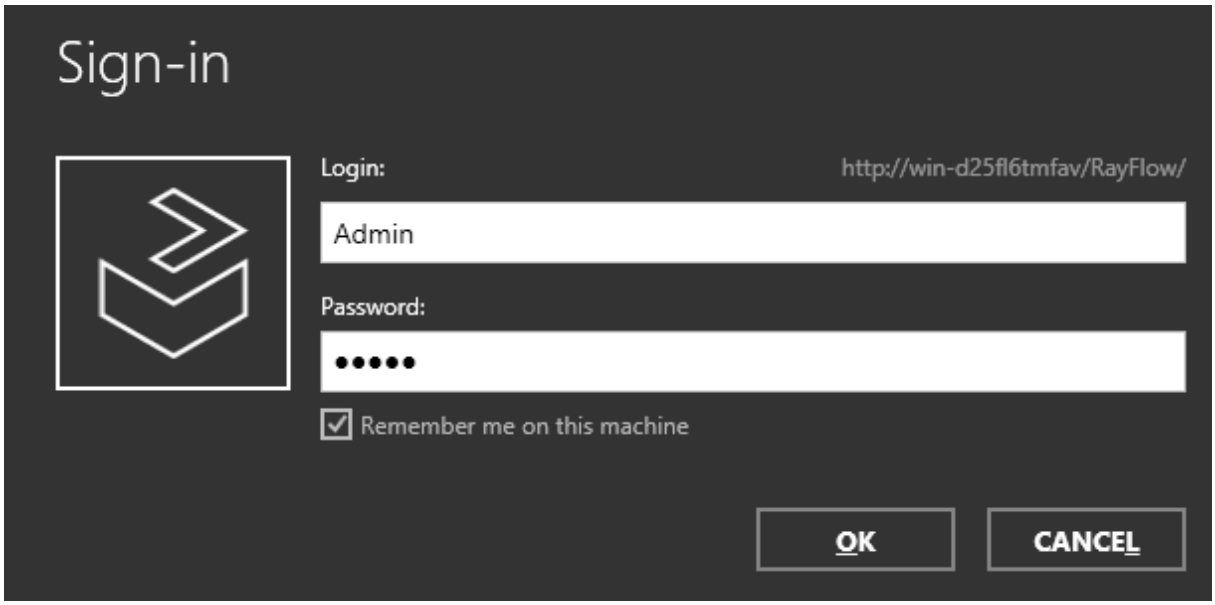
In order to sign-in, press the RayFlow button, located in the top right corner of the screen.



If the server is not valid or not specified, the following warning will be shown:



Otherwise, the sign-in overlay will be displayed over the current window:



The overlay contains the following information:

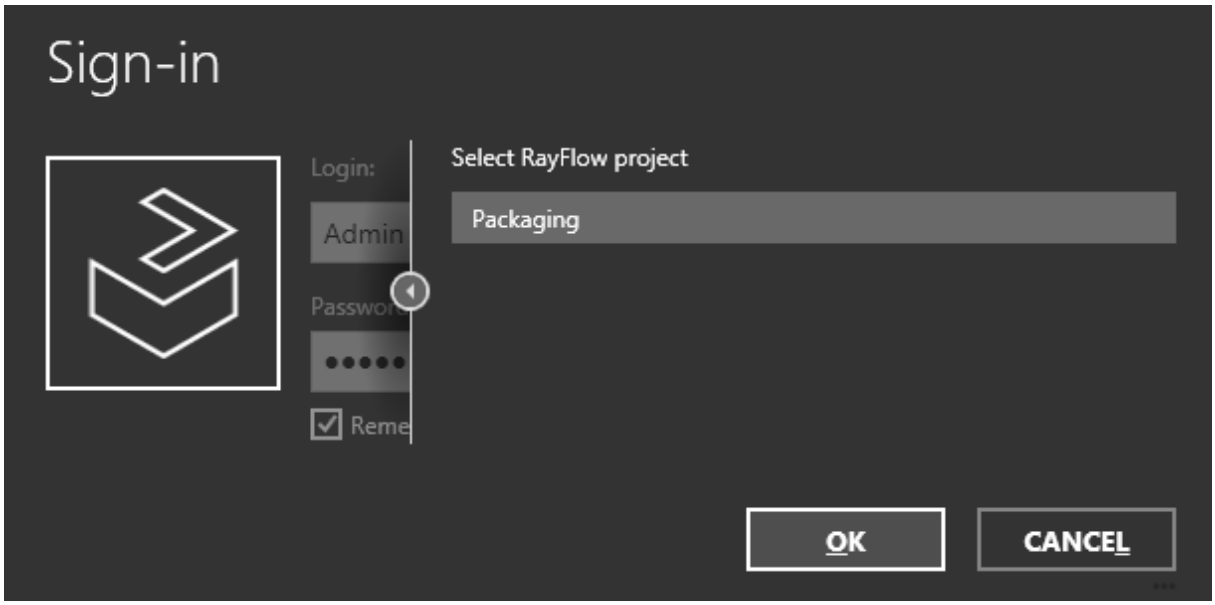
- The URL address of the current RayFlow instance, as configured in the profile settings
- The text fields for user name and user password
- A checkbox to remember the credentials on the current machine

Note:
RayFlow credentials have to be delivered by the local RayFlow administrator.

Login and password are required to sign-in to RayFlow.

Once the credentials are verified, a selection of projects available in the current RayFlow instance will be shown.

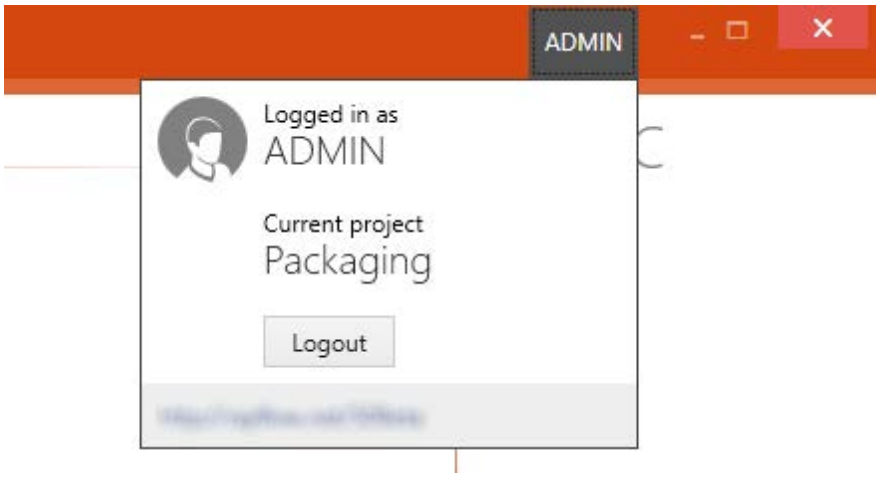
Note:
Only projects to which the current user has view permissions are displayed.



Once the project is selected the sign-in procedure is complete, and certain RayQC functions (like opening and saving files to RayFlow are available).

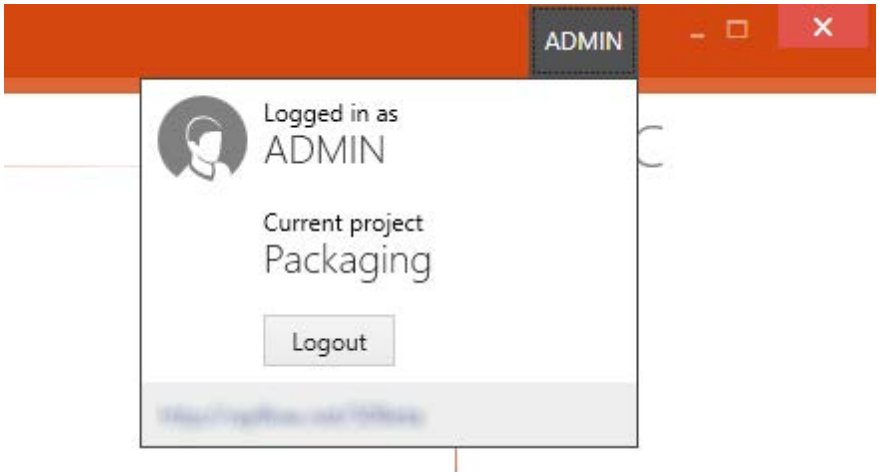
Once authenticated, the RayFlow button in the top right corner of the screen changes its color and displays the user name of currently authenticated user. After clicking on it additional details are shown, including:

- Current project
- An URL address of the current instance
- A button to logout



Signing out from RayFlow

In order to sign-out from RayFlow, or sign-in as another user, press the RayFlow button in the right top corner:



And then click the **Logout** button to sign-out from RayFlow.

Troubleshooting

The following section gives a list of typical issues users may have to face whilst working with RayQC. Since some of them may easily be solved, it is highly recommended to review this section before the product support is contacted. Please refer to the RaySuite Knowledge Base as well, since it contains information about current Known Issues and additional hints on handy solutions and procedures regarding working with RayQC.

License Activation Tool is Shown at RayQC Launch

The License Activation Tool is invoked whenever RayQC is launched without a valid license in place. Therefore, if users cannot run RayQC but always get the License Activation Tool instead, there must be an issue with the current licensing status of the RayQC application. In order to check the license status, please follow the steps outlined below.

Make sure to have your original RayQC order number at hand. It has been delivered along with the RayQC resources handed over to you for productive usage or validation purposes. Since the license checkup may include steps that require product activation, it is required to have the order number. Additionally, a machine with internet connection (for online activation or support driven activation via mail) has to be within reach as well if the machine RayQC is running on does not have access to the internet itself.

1. Open the RayQC application installation directory (e. g. `C:\Program Files (x86)\RayQC\`) and check the existance of a `*.license` file.
   a. If a `.license` file is in place, move it from this directory to another one (or simply rename its file extension from "license" to "license_") and try to launch RayQC again.

Note:
Moving or renaming the old `.license` file allows you to restore it later, or to send it to the support team in case of more complex issues. Please do not delete the old `.license` file before the RayQC application runs as expected.

2. Once the file is no longer in place, or if there has not been a `.license` file at all, the expected result is the launch of the License Activation Tool.

3. Click on the "I have my order number" button and enter your data as demanded. For details on how to accomplish that, please refer to the *Get Started Guide* that has been provided along with your RayQC installer resources.

4. Activate the product either by using the online activation via web-service, or by sending the required information to our Support Panel. If your activation data is complete and valid, the result depends on the activation method used: Either the web-service initiates the automated creation of a new .license file within the application installation directory, or the support team will reply and send a `.license` file you have to copy to the application installation directory manually.

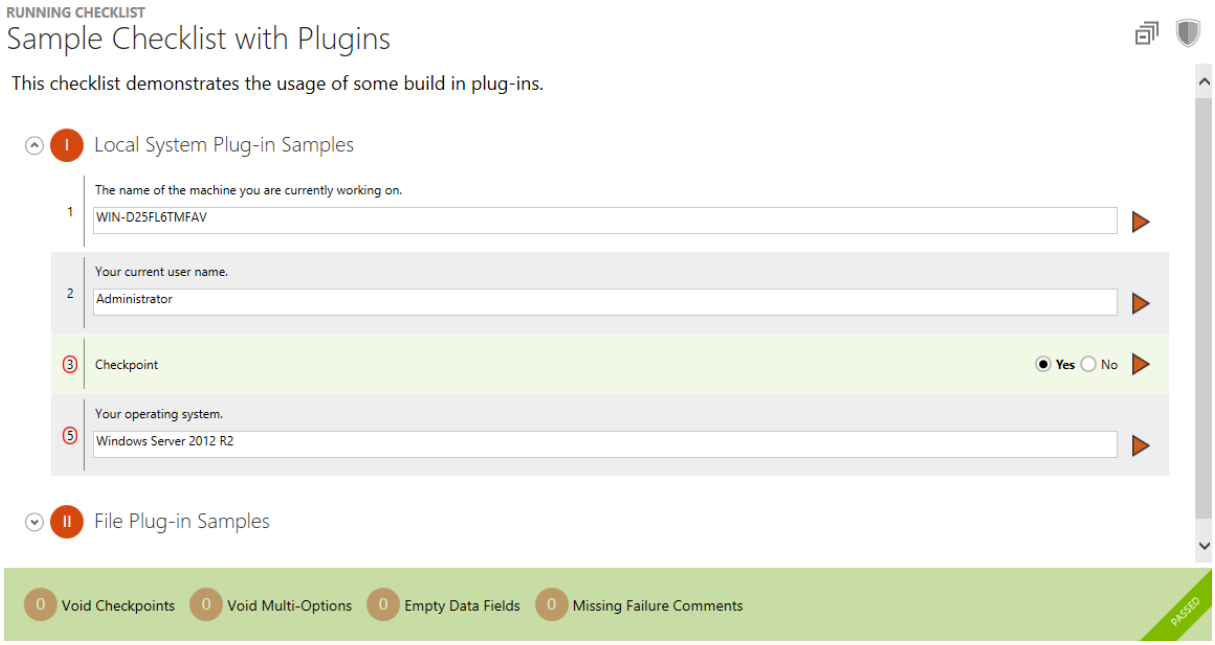
5. Either way, if the activation routine was executed successfully, you should have a new `.license` file within the application installation directory, and RayQC should launch as expected when the product's executable file is run.

If the issue is not solved by the steps named above, it is recommended to contact the Raynet support team via our Support Panel.

Missing Item Numbers in a Checklist Group

Usually there is a straight forward numbering given for the checklist items bundled within each checklist group container. Each newly added item increments the index value of the indentation level it has been added to.

However, there may be circumstances that lead to checklists that seem to have broken numberings when they are viewed or evaluated via the **Checklist Viewer** interface. If evaluators complain about this fact, checklist authors do not have to consider this a bug, but simply open the project with the **Checklist Editor** interface. If the missing index values belong to items that have an active condition option, their index value is missing in the **Checklist Viewer** by design.



Imagine the influence conditional items had on checklist item numbers if the index values were changed whenever the availability of an item is changed: The ID of all later items within the same group would change as well. This behavior would make it quite hard to refer to checklist items by their group and index value, which is why RayQC simply numbers all items independent from their current state of availability.

Missing Plug-Ins

Plug-ins are an essential part of the dynamic checklist features in RayQC. They may either be missing during the checklist creation / edition within the Checklist Editor, or be out of reach for execution during the evaluation via the Checklist Viewer.

Missing Plug-Ins During Checklist Preparation

Search for the plug-in resources at one of the type specific storage locations

When checklist editors enable the plug-in option for an element, there are three possible sources for the offered list of plug-ins. If a plug-in is expected to be available within the plug-in selector control for a specific checklist element, but is not, the following steps lead to enlightening information that should help to solve the issue.

- **Internal plug-ins**
  - Available for all checklists made with RayQC as they are delivered along with the RayQC resources and installed as mandatory feature by default.
  - Open the RayQC program directory (usually `C:\Program Files (x86)\RayQC\`). Check if a `\Libs\` directory is present and contains a file called `RayQC.plug-ins.dll`.
  - Please check whether the file and folder names are correct towards case sensitive lookups.

- **Global external plug-ins**
  - Available for all checklists evaluated with the local instance of RayQC, which include the plug-in prepared by any RayQC user.
  - Start RayQC and then go to the **Settings** ->**plug-ins** tab. Check if the desired plug-in is listed in the plug-in manager.
  - Open the RayQC program directory (usually `C:\Program Files (x86)\RayQC\`). Check if a `\plug-ins\` directory is present and a (matching) set of plug-in logic `([plug-inName].ps1)` and interface definition (`Manifest.xml`) is present in a sub-directory.

- **Local external plug-ins**
  - It is stored within the checklist file /checklist project file
  - Open the checklist file / project in the Checklist Editor. Click on the plug-ins tab and make sure that the plug-in is shown in the manager
  - As RayQC checklist file is basically a ZIP container that contains the actual `checklist.xml` file, supporting files and a plug-in folder which contains the local plug-in. Extract the checklist file and go to the plug-ins folder. Verify the functions and their arguments matches in the script and its `manifest.xml` file.

If any of the conditions named above is not given, the plug-in resources are missing and have to be copied to or created at the specified location.

Check the access rights on plug-in resource files

If the plug-in is available at one of the locations listed above, check whether the user that currently runs RayQC has access to the resource files. To do so, use the file / folder permission management of Windows.
Since a plug-in needs to be executed, the minimal required access level is execute.

If this condition is not meet, adjust the permissions as required and retry to use the plug-in within RayQC.

If all of the conditions named above are reviewed and regarded to be fulfilled, please contact your RaySuite / RayQC system administrator, or contact the Raynet support team via our Support Panel for further advice.

Check the compatibility and correctness of external plug-in resource files

Please verify the following dependencies:

- Each external (i.e. not built-in) plug-in needs to consist of a `.ps1` script and `.xml` manifest file. These two files must be stored in the same directory and manifest file should always be called `manifest.xml` file.
- The function names in the manifest file must match the respective function names in the PowerShell script. Same is true for the function arguments.
- `manifest.xml` file must always adhere to the structural criteria laid down by the `manifest.xsd` file

Missing Plug-Ins during checklist evaluation

Actually, all scenarios given above may very well cause a plug-in to be unavailable for execution during the evaluation phase. However, there are some more which are likely and therefore worth to take a look at:

Checklists launched for evaluation via RayFlow

If the checklist execution has been triggered by a RayFlow tool integration command, make sure it contains all required parameters in a suiting condition.
Due to the flexible configuration and parameter injection concept of the RaySuite products, there may be issues with the provided RayQC call regarding:

- the tool integration in RayFlow itself
- the RayFlow workflow data object property values (e. g. unescaped special characters in parameter values, wrong path values, etc.)
- unavailable resources, such as network shares, which may be referenced within the plug-in definition files, the RayFlow tool integration, and the like

Checklists that have been moved from one device (physical or virtual) to another

If a checklist template is moved from one location to another, it is mandatory to copy all local external and global external plug-in resources along to be available on the new checklist environment. (The same requirement for resource copies is given when a checklist contains relative path definitions to help files, images, and the like) As an alternative, the checklist resources may as well be stored on a shared location. Please make sure either the local or network path is correct and the required resources are there, accessible, and have not been damaged during the file transfer.

Logging RayQC Activity Fails

RayQC usually logs system activity in a file called `RayQC.log`, which is stored within the local program data directory (e. g., `C:\ProgramData\`). The log file location can directly be accessed via the Open logs folder option, which is available under the Troubleshooting tab of About page.

There may be reasons that prevent the log from being written. Please follow the steps described below in order to eliminate the most likely reasons for logging issues:

The Target Log File Location is Not Accessible for the Current User

To check whether the currently logged in windows user profile has sufficient access to the log file location:

1. Navigate to the RayQC application root directory, and open the log4net config file from the `\Config\`

subfolder (e. g., `C:\Program Files (x86)\RayQC\Config\log4netconfig.xml`)

- If it does not exist, create it, and add a copy of the default log4net configuration file to it (see Advanced Configuration Options).

2. Search for the `file` tag and read file name given as `value` attribute.

   - If it is a file name, such as the default value `RayQC.log`, this file is created directly within the application root directory.
   - If it is a relative path definition, the file is created relative to the application root directory.
   - If it is an absolute path, the file is created exactly at the specified location.
   - If it is empty - note the path to a location that is accessible for the currently logged in windows user profile and save the changed config file state

3. Browse to the specified location:

   - If the directories named within the path do not exist physically: create them.

   - If the directories named within the path actually do exist, check their security settings (right-click > properties > security > [current user] and make sure that the currently logged in user has writing access to the log file folder. (Ask your RaySuite system administrator for support if security settings are restricted as well.)

4. Close and re-open RayQC. Load any checklist template or project.

5. A log file should have been created within the target log file location by now.

The Target Log File Location Directory Does Not Exist

1. Please follow the instructions given above to make sure local and static locations are given correctly within the config file. Other possible locations for the log file creation may reside as a shared network location.

2. Try to copy the path into a windows explorer address bar to open the specified target address.

   - If the location is not available, there may be issues such as type errors, changed network shares or temporary network provision issues. Please adjust the path to a local destination as long as the network share is not available.
   - If the location is available, make sure that there are sufficient writing access permissions for the required windows user profile given.
   - If the location is accessible, make sure that there is enough free disk space for log file maintenance.

3. Try to create an arbitrary file at the specified target location (e. g., `MyTextFile.txt`).

   - If an error message is displayed, fix the mentioned issues and re-try.

4. Close and re-open RayQC. Load any checklist template or project.

5. A log file should have been created within the target log file location by now.

The Log File is There, But Nothing is Written Into It

1. Navigate to the RayQC application root directory, and open the log4net config file from the `\Config\` subfolder (e. g., `C:\Program Files (x86)\RayQC\Config\log4netconfig.xml`)

2. Search for the `maximumFileSize` tag and check the given value. It has to be a full number followed by either "KB", "MB" or "GB" (without blank space in between).

3. Set the value to "2048KB"

4. Additionally, search for the `level` tag and check the given value. If it is set to OFF, logging is actually deactivated. Set the value to DEBUG.

5. Close and reopen RayQC. Load any checklist template or project.

6. The log file should have been extended with new messages by now.

The Log File is Overloaded with Information

1. Navigate to the RayQC application root directory, and open the log4net config file from the `\Config\` subfolder (e. g., `C:\Program Files (x86)\RayQC\Config\log4netconfig.xml`)

2. Search for the `level` tag and check the given value. Set it to ERROR to reduce the number of lines that are written to the log file.

3. Additionally, review the layout type definition(s). Edit the pattern to reduce the information written for each line of the log file. Please refer to the log4net online documentation for further details on how to accomplish this task.

4. Delete the existing log file, close and re-open RayQC. Load any checklist template or project.

5. The log file content should have been adjusted to match the newly defined settings by now.

Changes to the Log Configuration File Cannot Be Saved.

On systems where UAC is activated, there may be issues with writing to files which reside within the Windows Program Files directory. A simple workaround for these issues is to copy the file to a location where writing is allowed, execute and save the changes there, and finally copy the manipulated file back to the original location. In most cases this should suffice. If this solution does not work, contacting the system administrator is due.

If all of the typical issues named above are reviewed and eliminated as possible issue reasons, please contact your RaySuite / RayQC system administrator, or contact the Raynet support team via our Support Panel for further advice.

Connections to Virtual Machines

Troubleshooting Problems with Hyper-V Connectivity

The following checklist helps to find and fix any possible issues when working with Hyper-V machines:

1. Is PowerShell 3.0 installed (both on Guest and Host Machine)?
   a. Check `$PSVersionTable.PSVersion` in PowerShell
2. Is the machine properly configured in the **Settings** > **Virtual Machines** screen (pay attention to hardcoded IP addresses which may be dynamically assigned by DHCP)
3. Is RayPack Studio Tools for Hyper-V installed on the Guest machine? Is the process `vm-proxy.exe` from RayPack Studio Tools for Hyper-V running?
4. Is WINRM configured?
   a. Check `winrm qc`
5. Does WINRM have proper `TrustedHosts` entries on both VM and server?
   a. `winrm s winrm/config/client '@{TrustedHosts="RemoteComputer"}'`
   b. `winrm g winrm/config/client` - shows the current `TrustedHosts` lists
   c. More information: https://technet.microsoft.com/en-us/library/ff700227.aspx
6. Does WINRM have a connection to the VM and vice-versa?
   `a.- Test-WSMan -ComputerName IP`
7. Are all necessary ports unblocked on the physical machine?
   a. The default port range is `48654-48999`.

Changing TCP / IP Configuration

In some cases it may be required to use custom port ranges, timeouts, etc. for PackBot related functionality.

The following table summarizes the available options:

Setting name	Default value	Description
`TcpIpDefaultPort`	`48654-48999`	Port range used for TCP/IP communication. Use minus (-) and comma (,) to indicate which ports are valid for incoming communication. Make sure that these ports are not blocked by your firewall. PackBot tries to find first valid free port and listen on it from lower to higher numbers.
`TcpIpMaxRetry`	`3`	Maximum number of retries before asserting the machine is not available..
`TcpIpDefaultReceiveTime`	`240000`	Reverts to default value if Windows does not define its own timeouts.
`TcpIpDefaultSendTimeout`	`240000`	Reverts to default value if Windows does not define its own timeouts.

Additional information

Visit www.RayQC.de for further information regarding the product and current community incentives. It's also recommended taking a look at additional resources available at the Knowledge Base for Raynet products: https://Raynet.de/Support/.

Raynet is looking forward to receiving your feedback from your RayQC experience. Please contact your Raynet service partner or use our Support Panel to add your ideas or requirements to the RayQC development roadmap!

Help & Support

Request RayQC support

Our Raynet support team gladly assists you on any questions or issues you encounter regarding RayQC. Feel free to sign in and open incidents via our Raynet support panel.

Join the RaySuite community

The RaySuite community resides within our Knowledge Base: https://raynet.de/Support/. Once you have signed up for access to the Raynet support panel, you automatically have access to the Knowledge Base, too.
You will surely come to a point where you would love to suggest a new feature for the future development of RayQC. Maybe you need to find some tips & tricks to hit your target right. The RaySuite community is your place for discussing such topics, for sharing and expanding your own experience.

Step in contact with Raynet's testers, evaluators, and consultants in order to learn how to polish your quality assurance activities to a level of highest quality standards. Since Raynet has years and years of experience, we know what to do, and how to do it. Don't row your boat alone when you have the chance to join our RaySuite community for free.

Contact your Raynet sales representative

Our sales team is the right contact for any license or edition question you might encounter. You would like to benefit from a professional RayQC training? Ask for dates and locations to find the fitting training occasion. You are highly welcome to step in contact via sales@raynet.de.

Appendices

The following sections are designed to give detailed information about the general checklist and external plug-in structure required by RayQC. Please refer to the sample directories within your RayQC application installation directory (e. g., `C:\Program Files (x86)\RayQC\`) for further sample source code and checklist files.

Basic Checklist Structure

Every checklist a user wants to open and process with RayQC has to be valid against the RayQC XML Checklist Schema Definition. The definition file `ChecklistSchema.xsd` is available from the root of the application installation directory (e. g., `C:\Program Files (x86)\RayQC\`). Open this file in a suiting editor of your choice to take a look at the requirements for a minimal RayQC checklist:

- XML version must be `1.0` or higher
- Encoding must be `utf-8`
- Root node must be `<checklist>`
- `<checklist>` child `<checklistHeader>` must be present exactly once
- `<checklistHeader>` child `<title>` must be present exactly once but may be empty
- `<checklistHeader>` child `<description>` may be present exactly once but may be empty
- `<checklistHeader>` child `<reportFilename>` may be present exactly once but may be empty
- `<checklistHeader>` child `<bypassMessage>` may be present exactly once but may be empty
- `<checklist>` child `<checklistContent>` must be present exactly once
- `<checklistContent>` child `<group>` must be present at least once
- `<group>` attribute `[id]` must be present exactly once per `<group>` and unique throughout the checklist structure
- `<group>` child `<groupHeader>` must be present exactly once
- `<groupHeader>` child `<title>` must be present exactly once but may be empty
- `<groupHeader>` child `<description>` must be present exactly once but may be empty
- `<group>` child `<groupContent>` must be present exactly once
- At least one `<groupContent>` Can have unbounded number of following child elements with unique global id
  - `<checkpoint>` with attribute {`id`]present and unique throughout the checklist structure
  - `<information>` with attribute `[id]` present and unique throughout the checklist structure
  - `<dataField>` with attribute `[id]` present and unique throughout the checklist structure
  - `<multiOption>` with attribute `[id]` present and unique throughout the checklist structure

According to this description, the following checklist source code is valid:

```
<?xml version="1.0" encoding="utf-8"?>
<checklist lastChange="2015-04-14T11:05:21.6638863+02:00" allowBypass="false">
  <checklistHeader>
    <title />
    <description />
    <reportFilename>RayQC Report - #title#</reportFilename>
    <bypassMessage>Bypassing will invert the result. A bypass reason should be
provided.</bypassMessage>
  </checklistHeader>
  <checklistContent>
```

```
    <group id="Group_381632152800">
      <groupHeader>
        <title />
        <description />
      </groupHeader>
      <groupContent>
        <information id="Information_381632152800">Description</information>
      </groupContent>
    </group>
  </checklistContent>
</checklist>
```

However, a checklist file without any content may be valid, but is definitively useless at the same time. In order to understand the required and optional elements and nestings of RayQC checklists, it is highly recommended to review the sample checklist files that are available with the RayQC application installation directory (e. g., `C:\Program Files (x86)\RayQC\Samples\`).

Checklist Example

The following XML structure is created as default for new checklists and is applied when users hit the Create new checklist button from the Dashboard:

```
<?xml version="1.0" encoding="utf-8"?>
<checklist lastChange="2015-04-14T11:13:39.6809931+02:00" allowBypass="false">
  <checklistHeader>
    <title>Checklist Title</title>
    <description />
    <reportFilename>RayQC Report - #title#</reportFilename>
    <bypassMessage>Bypassing will invert the result. A bypass reason should be
provided.</bypassMessage>
  </checklistHeader>
  <checklistContent>
    <group id="Group_663731202081">
      <groupHeader>
        <title>Group Title</title>
        <description />
      </groupHeader>
      <groupContent>
        <information id="Information_663731202081">Description</information>
      </groupContent>
    </group>
  </checklistContent>
</checklist>
```

Please take a look the checklist sample directory from within your application installation directory (e. g., C:\Program Files (x86)\RayQC\Samples\). This directory contains a set of checklists, ready to review and use as samples to start from.

Tip:
Another approach to deeper understanding for the XML structures is to simply create a checklist template in RayQC and open the checklist.xml file from within the template container *.rqct with an external XML editor such as Foxe. Observing the effect of minimal changes in the Checklist Editor interface is as well possible by comparing two checklist source versions, e. g. by using a tool like WinMerge.

Basic External Plug-In Structure

As already described within the plug-ins section of this document, an external plug-in consists of two basic elements: the **script file** and the descriptive **manifest file**.

The script file

Source code that is about to be used as plug-in script for RayQC must validate against a set of minimal requirements in order to be integrated correctly:

- External plug-ins have to be valid PowerShell `.ps1` files. (To launch other files, like executables, the internal command plug-in can be used.)
- Parameter definitions within script and manifest file must match

The manifest file

Every plug-in script users want to use in combination with RayQC checklists has to come along with a manifest XML file that is valid against the RayQC plug-in Schema Definition. The definition file `ManifestSchema.xsd` is available from the root of the application installation directory (e. g., `C:\Program Files (x86)\RayQC\`). Open this file in a suiting editor of your choice to take a look at the requirements for a minimal RayQC plug-in manifest file:

- XML version must be `1.0` or higher
- Encoding must be `utf-8`
- Root node must be `<pluginData>`
- `<pluginData>` must have non-empty attribute `name`
- `<pluginData>` attribute `description` must be present exactly once but may be empty
- `<pluginData>` must have non-empty attribute `version`
- `<pluginData>` must have non-empty attribute `FileName`
- `<pluginData>` may have one non-empty attribute `Encrypted`
- `<pluginData>` must have non-empty attribute `PowerShellVersion`
- `<pluginData>` child `<Function>` must be present exactly once
- `<Function>` must contain one or more child `<FunctionParameters>`
- `<FunctionParameters>` must have non-empty attribute `FunctionName`
- `<FunctionParameters>` attribute `<description>` must be present exactly once but may be empty
- `<FunctionParameters>` child `<Parameters>` must be present exactly once
- At least one `<Parameters>` child out of the following options must be present
  - `<BooleanParameter>` : Expects a boolean input (true/false)
    - `<Name>` : Name of the parameter
    - `<Description>` : User defined description of the parameter
    - `<IsOptional>` : When set to true, providing input for this parameter is optional
    - `<Default>` : Default parameter value
  - `<StringParameter>` : Expects a string as input parameter

- `<Name>` : Name of the parameter
- `<Description>` :User defined description of the parameter
- `<IsOptional>` : When set to to true, providing input for this parameter is optional
- `<Default>` : Default parameter value
- `<RegexString>`: Regular expression
- `<ValidationMessage>` : Validation message that is shown if the string does not satisfy the regular expression

o `<NumberParameter>` : Expects an integer value between Min and Max

- `<Name>` : Name of the parameter
- `<Description>` : User defined description of the parameter
- `<IsOptional>` : When set to true, providing input for this parameter is optional
- `<Default>` : Default integer parameter value
- `<Min>` : Minimum input integer value
- `<Max>` : Maximum input integer value
- `<Increment>` : Increment value

o `<MultiStringParameter>` : Provides user option to select from drop-down a string that is provided as parameter

- `<Name>` : Name of the parameter
- `<Description>` : User defined description of the parameter
- `<IsOptional>` : When set to true, providing input for this parameter is optional
- `<Default>` : Default string from the string list
- `<StringList>` : List of strings defined. e.g. `<string1/> <string2/> <string3/>`

According to this description, the following manifest file source code is valid:

```
<?xml version="1.0" encoding="utf-8"?>
<pluginData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Name />
  <Description />
  <Version />
  <Filename />
  <PowerShellVersion />
  <Functions>
    <FunctionParameters>
      <FunctionName />
      <Description />
      <Parameters />
    </FunctionParameters>
  </Functions>
</plug-inData>
```

However, a manifest file without any productive content may be valid, but is definitively useless at the same time. In order to understand the required and optional elements and nestings of RayQC plug-in manifest files, it is highly recommended to review the sample plug-ins and their manifest files, which are available with the RayQC application installation directory (e. g., `C:\Program Files (x86)\RayQC\Samples\`).

External Plug-In Example

The following script is derived from the `PowerShellSample.ps1`, which is part of the `ExampleplugInSample.rqct` checklist file. This file is located in the RayQC application installation directory (e.g. `C:\Program Files ()x86)\RayQC\Samples`)

```
function TestMeOne($param1, $param2)
{
    "Hello from TestMe ONE with $param1, $param2"
}

function TestMeTwo($param3, $param4)
{
    "Hello from TestMe TWO with $param3, $param4"
}

function TestMeThree($param5)
{
    "Hello from TestMe THREE with $param5"
}
```

Raynet GmbH
Technologiepark 20
33100 Paderborn, Germany
T +49 5251 54009-0
F +49 5251 54009-29
info@raynet.de
support@raynet.de

www.raynet.de