RCE User Guide

Build 9.1.1.0202001220840_RC

Table of Contents

List of Figures

List of Tables

Chapter 1. Preface

This chapter gives an introduction to RCE.

1.1. Abstract

RCE (Remote Component Environment) is an open source software that helps engineers, scientists and others to create, manage and execute complex calculation and simulation workflows. A workflow in RCE consists of components with predefined inputs and outputs connected to each other. A component can be a simulation tool, a tool for data access, or a user-defined script. Connections define which data flows from one component to another. There are predefined components with common functionalities, like an optimizer or a cluster component. Additionally, users can integrate their own tools. RCE instances can be connected with each other. Components can be executed locally or on remote instances of RCE (if the component is configured to allow this). Using these building blocks, use cases for complex distributed applications can be solved with RCE.

1.2. Intended Audience

The intended audience of this document consists of engineers, scientists, and everybody else interested in developing automated workflows with RCE.

1.3. License Information

RCE is published under the Eclipse Public Licence (EPL) 1.0. It is based on Eclipse RCP 4.8.0 (Photon), which is also published under the Eclipse Public Licence (EPL) 1.0. RCE also makes use of various libraries which may not be covered by the EPL; for detailed information, see the file "THIRD_PARTY" in the root folder of an RCE installation. (To review this file without installing RCE, open the RCE release .zip file.)

For downloads and further information, please visit https://rcenvironment.de/.

1.4. Compatible Operating Systems

RCE releases are provided for Windows and Linux. It is regularly tested on

- Windows 7
- Windows 10
- Windows Server 2016
- CentOS 7
- Debian 9
- SUSE Linux Enterprise Desktop ("SLED") 12 SP2

- Ubuntu 18.04 LTS

1.4.1. Support of 32 Bit Operating Systems

Starting with release 8.0.0, RCE is only shipped for 64 bit systems. If you still require 32 bit packages, you can continue to use previous RCE releases, but there will be no standard feature or bugfix updates for them.

1.4.2. Known Issues

1.4.2.1. KDE on Red Hat Enterprise Linux 7

On Red Hat Enterprise Linux 7 with KDE 4, RCE (like any other Eclipse-based application) can cause a segmentation fault at startup. If you encounter such an issue, you can try choosing a different GTK2 theme:

1. Open the **System Settings** application (systemsettings).
2. Go to **Application Appearance**
3. Open **GTK** page
4. Switch the GTK2 theme to "Raleigh" or "Adwaita" and click on **Apply**

1.4.2.2. KDE with Oxygen

On Unix Systems using KDE as desktop environment and Oxygen as theme it can happen that RCE crashes when certain GUI elements are shown. It is a known issue in the theme Oxygen and happens on other Eclipse-based applications as well. If you encounter such an issue, please choose a different theme like "Raleigh" or "Adwaita".

1.4.2.3. Jython scripts are executed sequentially

The Script component can use Jython for the evaluation of scripts and the pre- and postprocessing of integrated tools always uses Jython. Due to a known bug in the Jython implementation it is not possible to execute several Jython instances in parallel. Therefore, the execution will be done sequentially. If several Script components should be executed in parallel, Python should be used instead.

1.4.2.4. 32-bit Java is not supported

Running RCE with a 32-bit Java Runtime Environment doesn't work. On some operating systems an error dialog will be displayed in this case, on some other systems nothing will happen at all. Therefore, always make sure a 64-bit Java Runtime Environment is used to run RCE.

Chapter 2. Setup

This section describes the installation and configuration of RCE.

2.1. Installation on Linux

2.1.1. Prerequisites

To run RCE on a system, the only prerequisite is an installed Java Runtime Environment (JRE), version 8u161 or above. If you don't already have one on your machine, use your system's package manager to install it; the most common choice is the OpenJDK JRE.

> **Note**
>
> Some pre-installed components of RCE have additional dependencies. Please refer to Section 2.3 (Workflow Components) for more details.

2.1.2. Installation

There are two installation options on Linux: installing RCE from .deb/.rpm packages, or extracting it from a zip file (which was traditionally used by earlier versions of RCE). Whenever possible, using the packages is recommended, as RCE is automatically installed into the proper system locations, and can be cleanly managed using your distribution's package manager. If you are using a Debian-based distribution, a package repository is provided as well, which makes it even easier to install and upgrade RCE. The following table compares these options:

Table 2.1. Linux installation options

Installation type	Multi-user operation supported	Daemon operation (system service) supported	Installing more than one version at the same time	File system location	Updating to a new version	Verifying digital signatures	Registers start menu entry and icon
Using the package repository *(Debian/ Ubuntu/ Mint only)*	**yes**	**yes**	no	`/usr/share/rce`	Using the distribution's update manager (automatic or manual)	**automatic**	**yes**
Manual installation of the.deb/.rpm package	**yes**	**yes**	no	`/usr/share/rce`	Manually download and install a newer package	manual (gpg/ shasum)	**yes**
Unpacking the .zip file	no	no	**yes**	(anywhere)	Use "Help > Check for Updates" in RCE **-or-** delete the old	manual (gpg/ shasum)	no

Installation type	Multi-user operation supported	Daemon operation (system service) supported	Installing more than one version at the same time	File system location	Updating to a new version	Verifying digital signatures	Registers start menu entry and icon
					installation directory and manually download and unpack a newer zip file		

2.1.2.1. Debian/Ubuntu/Mint - Installing from the Package Repository (recommended)

To register the RCE .deb package repository in your system, you have to add the repository location, and add the RCE signing key. Then, you can install RCE like any other software package. There are two different ways to do this, both leading to the the same result.

- (Option 1) If you prefer to use graphical tools, start the program "Synaptic"; if it is not available, install if from your Software Center - or similar - first. Please note that this option might not be available on all systems. If this is the case for you, you can still install the package using the manual process listed as Option 3.
  - Open "Settings > Repositories".
  - Click on "Authentication keys", and then on "Download a key...". Copy the text

    ```
    258BC129EDA2389D3ECD2DE6BA880CB39DC1CE34
    ```

    into the dialog that appears and click "Ok". The list of keys should now contain an entry "RCE 6.x-9.x Automatic Signing Key <rcenvironment-builds@lists.sourceforge.net>".
  - Click on a section named "Additional Repositories", "Other Sources" or similar. Select the "Add..." or "Add new..." option. Copy the text

    ```
    deb https://software.dlr.de/updates/rce/9.x/products/standard/releases/latest/deb/ /
    ```

    into the dialog that appears and click "Ok".
  - Close the settings dialog and click "Reload" to update the list of available software. After this, you should find an entry "rce" in this list (for example, using the Search button), which you can install like any other software. You will also receive automatic notifications when a newer version of RCE 9.x is available.
- (Option 2) If you prefer to execute all steps manually using the command line, enter the following commands as "root" (or with the "sudo" prefix). Obviously, this is the most complicated way.
  - `apt-key adv --recv-key 258BC129EDA2389D3ECD2DE6BA880CB39DC1CE34`

    Note

    If you see a message containing the text `"RCE 6.x-9.x Automatic Signing Key <rcenvironment-builds@lists.sourceforge.net>"`, the key import was successful. Sometimes, this step fails with the message "`key [...] not found on keyserver; gpg: no valid OpenPGP data found.`" If this happens, just repeat the command a few times until it works. This issue may also be caused by an outdated gpg version (gpg version 2.0.9 on SUSE Linux Enterprise Desktop 11 is known to be problematic, while gpg version 2.0.22 works).

    The RCE signing key is published on the keyserver `pool.sks-keyservers.net`, which is the default keyserver for most distributions of gpg. If your installation does not default to that keyserver, you may

try to explicitly specify the keyserver at `hkps://hkps.pool.sks-keyservers.net` or at `hkp://pool.sks-keyservers.net`. Please consult the documentation of your installation of gpg to determine how to specify the keyserver.

If you cannot access keyservers at all for some reason, you can also download the key manually from

```
https://github.com/rcenvironment/rce-signing/blob/master/rce_9.x_signing_key.asc
```

After the download has completed, you can import the key by executing

```
apt-key adv --import rce_9.x_signing_key.asc
```

from the command line.

- ```
  echo "deb https://software.dlr.de/updates/rce/9.x/products/standard/releases/latest/deb/ /"
    >/etc/apt/sources.list.d/dlr_rce_9_releases.list
  ```

  Note

  Although it may be split across lines here, this command must be executed as a single line.

- ```
  apt-get update
  ```

- ```
  apt-get install rce
  ```

Note

For previous versions of RCE there existed a script that imported the signing key from github and added the software repository to the system repositories automatically. Due to recent changes to our workflow for producing RCE releases, we are currently improving and adapting these scripts. Once they are available, you can find them at

```
https://github.com/rcenvironment/rce-signing
```

Executing the script

```
register_rce_9.x_deb_repository.sh
```

will import the signing key onto your system and subsequently register the repository with your package manager.

Once you have installed RCE using either of these approaches, any RCE 9.x upgrade will automatically show up via the update mechanism of your operating system. Depending on the upgrade settings of your system, they may be installed automatically, or be presented to you for selection.

Note

Although technically possible, RCE 9.x will not auto-upgrade to 10.x (or higher) to maintain compatibility within networks of RCE 9.x instances. You will need to manually install the 10.x repository location in order to upgrade.

2.1.2.2. Debian/Ubuntu/Mint - Installation of the .deb Package (alternative)

To install the .deb file manually, download the latest version from

```
https://software.dlr.de/updates/rce/9.x/products/standard/releases/latest/deb/
```

You can either install it using the graphical package management tools of your distribution (double-clicking the .deb should start them), or by running `sudo dpkg -i <filename>` from a terminal. To upgrade an existing installation, simply install the newer package. The package manager will detect the upgrade and handle it properly.

2.1.2.3. CentOS/Red Hat - Installation of the .rpm Package (recommended)

To install the .rpm file, download the latest version from

```
https://software.dlr.de/updates/rce/9.x/products/standard/releases/latest/rpm/
```
```

You can either install it with the graphical package management tools of your distribution (double-clicking the .rpm should start them), or by running the appropriate installation command as the "root" user. On CentOS or Red Hat, the command is `yum install <filename>`; on SUSE, use `zypper install <filename>`. To upgrade an existing installation, simply install the newer package. The package manager will detect the upgrade and handle it properly.

2.1.2.4. All Distributions - Running from the .zip File (alternative)

If none of the previous installation options fits your needs, you can also extract RCE from a zip file downloaded from

```
https://software.dlr.de/updates/rce/9.x/products/standard/releases/latest/zip/
```

- If you prefer graphical tools, double-click the .zip file to open it with your distribution's archive manager. Extract it to a location of your choice, and open that location in your file-system explorer. Typically, double-clicking the "rce" executable will work out of the box and start RCE. If this does not happen, right-click the executable, open its "properties" section (or similar), and look for an option to mark it as executable. Confirm the dialog, then double-click it again.

- If you prefer using the command line, use the `unzip` command to extract the zip file to a location of your choice. In the location where you unpacked the files to, you can usually simply enter

  ```
  ./rce
  ```

  to start RCE. In some cases, you may first need to make it executable using the command

  ```
  chmod +x rce
  ```

  Note

  The path to your installation location must not contain any colons to avoid Java Virtual Machine errors when starting RCE.

2.1.3. Digital Signatures and Download Verification

Starting with RCE 6.x, we provide digital signatures for our releases. These can be used to verify the integrity of the downloaded files, as *any* unsigned software can be tampered with while downloading it over an unsecured (HTTP) connection. This is especially important when installing RCE from a user account with root privileges, for example as a system daemon

For each type of release files (.zip, .deb, and .rpm files), a `SHA256SUM.asc` file is provided next to the actual files. This file contains checksums for each release file, and these checksums are signed with our release key. It is named "RCE 6.x-9.x Automatic Signing Key", has the id 9DC1CE34, and is valid until Nov 22, 2020. Its full fingerprint is `258B C129 EDA2 389D 3ECD 2DE6 BA88 0CB3 9DC1 CE34`.

On Linux, the required command-line tools to verify these signatures (gpg and shasum/sha1sum/sha256sum) are usually already installed as part of your distribution. If they are not, use your system's package manager to install them. To perform the actual verification:

- Execute

  ```
  gpg --recv-key 258BC129EDA2389D3ECD2DE6BA880CB39DC1CE34
  ```

  to import the signing key. This only needs to be done once per key, e.g. once for all RCE 9.x releases.

  Note

  If you see a message containing the text `"RCE 6.x-9.x Automatic Signing Key <rcenvironment-builds@lists.sourceforge.net>"`, the key import was successful.

Sometimes, this step fails with the message "`key [...] not found on keyserver; gpg: no valid OpenPGP data found.`" If this happens, just repeat the command a few times until it works. This issue may also be caused by an outdated gpg version (gpg version 2.0.9 on SUSE Linux Enterprise Desktop 11 is known to be problematic, while gpg version 2.0.22 works).

The RCE signing key is published on the keyserver `pool.sks-keyservers.net`, which is the default keyserver for most distributions of gpg. If your installation does not default to that keyserver, you may try to explicitly specify the keyserver at `hkps://hkps.pool.sks-keyservers.net` or at `hkp://pool.sks-keyservers.net`. Please consult the documentation of your installation of gpg to determine how to specify the keyserver.

If you cannot access keyservers at all for some reason, you can also download the key manually from

```
https://github.com/rcenvironment/rce-signing/blob/master/rce_9.x_signing_key.asc
```

After the download has completed, you can import the key by executing

```
apt-key adv --import rce_9.x_signing_key.asc
```

from the command line.

- Download the `SHA256SUMS.asc` file from the same location as the installation package and place it in the same folder as the downloaded .zip-file. Run

  ```
  gpg --verify --yes SHA256SUMS.asc
  ```

  in the location where you saved it to; this verifies the digital signature. Inspect the output to see if it is correct; you should find the text "Good signature from "RCE 6.x-9.x Automatic Signing Key <rcenvironment-builds@lists.sourceforge.net>" (or a similar translation). Once the command has terminated, you should find a file `SHA256SUMS` in the current folder.

  Note

  When following these steps, it is normal to receive a warning about the fact that the owner of this key cannot be verified. If you have received this user guide from a trustworthy source (e.g. an official RCE project site secured with HTTPS), you can assume that the key is correct, as the command used to import the key has already verified the key's integrity. Alternatively, you can fetch the key's fingerprint from a trustworthy source (e.g. from a secure intranet page, or the official @rcenvironment Twitter feed accessed via HTTPS) and compare it with the one shown in the command's output. If they match, you can trust that you are using the authentic key.

- Run

  ```
  shasum -c SHA256SUMS
  ```

  in the same folder. This verifies that the actual download matches what has been digitally signed for the release. You should see the installation package's name, followed by "OK".

  Note

  On some distributions, you need to use `sha1sum` or `sha256sum` instead of `shasum`.

2.1.4. Starting RCE as a GUI Client

Once your RCE instance has started, you can open the configuration file with the menu option "Help > Open Configuration File". Edit the file, save it, and then restart RCE using the "File > Restart" menu option to apply the changes. There are configuration templates and other information available via the "Help > Open Configuration Information" option. The available configuration settings are described in the configuration chapter.

Note

On Ubuntu, the Ubuntu overlay scrollbars can sometimes lead to problems with the RCE GUI. To avoid these problems, you can start RCE from a terminal with `env LIBOVERLAY_SCROLLBAR=0 ./rce` to disable the overlay scrollbars for RCE. Alternatively, if you want to disable the overlay scrollbars

permanently for all programs, execute `echo "export LIBOVERLAY_SCROLLBAR=0" > /etc/X11/Xsession.d/80overlayscrollbars` as a superuser and then restart your computer.

2.1.5. Starting a Non-GUI ("Headless") Instance

RCE can also be run from the command line without a graphical user interface (which is called "headless" mode), which uses less system resources and is therefore recommended when the GUI is not needed.

To run a headless RCE instance, open a terminal and run the command

```
rce --headless -console
```

While RCE is running, you can enter various console commands described in Section 3.3, “Commands”; note that you need to prefix all RCE commands with "rce" here. To perform a clean shutdown, for example, type `rce stop` and press enter.

2.1.6. Installation as a Daemon on a Linux Server

For ad-hoc or temporary RCE network setups, running a headless RCE from the command line is perfectly fine. For more permanent installations, however, we recommend installing RCE as a system daemon instead. This has the advantage that RCE automatically shuts down when the server is shut down, and automatically restarts when the server does.

2.1.6.1. Installation and Daemon Management

Executing the following steps will install RCE as daemon. An RCE daemon will start automatically on system boot and stop before system shutdown.

The recommended (and supported) way to install a Linux daemon is to install RCE from a .deb or .rpm package, or (equivalently) installing it from the APT repository. While installing a daemon from the .zip file distribution can be made to work, it creates unnecessary complications regarding installation paths and file permissions. As registering a daemon requires root privileges anyway, there should be no reason to use the .zip file; if you have a compelling use case for this, please contact the RCE team at `rce@dlr.de`.

Once RCE is properly installed, registering it as a daemon is very simple. A command named `rce-daemon` is provided to control the whole daemon life cycle. The only manual step you need to perform before using that command is creating a user account that the daemon will run under. For now, this account has to be named `rce-daemon`; use your distribution's console commands or GUI tools to add it. (Please note that this account needs to have a home directory to hold the daemon's profile directory.) Future RCE versions will make this account name customizable.

Once this user account is created, you can use the following commands to manage the daemon installation:

- `rce-daemon install` - installs and starts the daemon instance
- `rce-daemon uninstall` - stops and uninstalls the daemon instance; the profile directory remains unchanged
- `rce-daemon start` / `rce-daemon stop` / `rce-daemon restart` - standard daemon controls
- `rce-daemon locate` - prints the location of the daemon's RCE instance's profile directory and relevant files
- `rce-daemon status` - displays if the RCE daemon is currently running or not

2.1.6.2. Daemon Configuration

After installation, the daemon instance will be started automatically. This will create a default configuration file if it does not exist yet.

To configure the daemon instance, use `rce-daemon locate` to find its configuration file, edit and save it, and then use `rce-daemon restart` to apply the new configuration.

Note

The need to restart the daemon is temporary; future versions of RCE will apply configuration changes as soon as the configuration file is changed.

2.2. Configuration

This section describes the configuration of RCE. Configuration is done within one single configuration file. It is located in the profile directory. From the graphical user interface, you can easily access it from the tool bar or the *Help* menu. Note: To apply changes you need to restart RCE. The format of the configuration file is JSON. See http://www.json.org/ for the format definition. Also refer to the example configuration files in the installation data directory.

2.2.1. Configuration Locations and Files

Starting with RCE 6.0.0, all user data is strictly separated from the RCE installation itself. Each set of user data is contained in a so-called "profile". Each profile defines what is called an RCE "instance". Each profile (and therefore, each instance) belongs to exactly one user, and each user can have multiple profiles. The default profile is located within the user's "home" directory ("/home/<user id>/" on Linux), in the ".rce/default" sub-folder.

Note

Note that ".rce" is a hidden directory; you may need to set operating-specific options to see hidden files and directories.

All manual configuration takes place in the profile's central configuration file, "configuration.json". As of RCE 9.0.0, most configuration settings only take effect on startup, so you need to restart RCE after editing it. (This will be changed in a future release.) This applies to all types of installations.

2.2.2. Configuration Parameters

Configuration parameters are grouped within the configuration file. Below are lists of the configuration parameters. There is one list per JSON configuration group. You can find example configurations in "configuration_reference_sample" in the installation data directory or by opening the configuration information in RCE.

Table 2.2. "general"

Configuration key	***Comment***	***Default value***
instanceName	The name of the instance that will be shown to all users in the RCE network. The following placeholders can be used within the instance name:	"<unnamed instance>"

Configuration key	*Comment*	*Default value*
	• ${hostName} is resolved to the local system's host name. • ${systemUser} is resolved to the Java "user.name" property. • ${profileName} is resolved to the last part of the current profile's file system path. • ${version} is resolved to the build id. Example: "Default instance started by \"${systemUser}\" on ${hostName}".	
isWorkflowHost	If set to *true*, the local instance can be used as a *workflow host* by other RCE instances. I.e., the workflow controller can be set to this instance and the workflow data is stored there as well.	false
isRelay	If set to *true*, the local node will merge all connected nodes into a single network, and forward messages between them. This behaviour is transitive; if a relay node connects to another relay node, both networks will effectively merge into one. If set to *false* (the default value), the local node can connect to multiple networks at once without causing them to merge.	false
tempDirectory	Can be used to override the default path where RCE stores temporary files. Useful if there is little space in the default temp file location. Must be an absolute path to an existing directory, and the path must not contain spaces (to prevent problems with tools accessing such directories). The placeholder *${systemUser}* can be used for path construction, e.g. "/tmp/custom-temp-directory/${systemUser}"	An "rce-temp" subdirectory within the user or system temp directory.
enableDeprecatedInputTab	If set to *true* the tab 'Inputs' is enabled again in the properties view of running workflows. It is disabled by default due to robustness and memory issues. It is recommended to use the 'Workflow Data Browser' to see inputs received and outputs sent.	false

Table 2.3. "backgroundMonitoring"

Configuration key	*Comment*	*Default value*
enabledIds	Comma-separated list of identifiers referring to certain kind of monitoring data that should be logged continuously in the background. Currently, only 'basic_system_data' is supported.	
intervalSeconds	Logging interval	10

Note

IMPORTANT: When setting up a network of RCE instances, keep in mind that the RCE network traffic is currently *not encrypted*. This means that it is *not secure* to expose RCE server ports to untrusted networks like

the internet. When setting up RCE connections between different locations, make sure that they either connect across a secure network (e.g. your institution's internal network), or that the connection is secured by other means, like SSH tunneling or a VPN. Alternatively, you can set up SSH connections in RCE instead of the standard RCE connections.

Table 2.4. "network"

Configuration key	***Comment***	***Default value***
requestTimeoutMsec	The timeout (in milliseconds) for network requests that are made by the local node. If this time expires before a response is received, the request fails.	40000
forwardingTimeout Msec	The timeout (in milliseconds) for network requests that are forwarded by the local node on behalf of another node. If this time expires before a response is received, an error response is sent back to the node that made the request.	35000
connections	A map of all connections that the local instance tries to establish on startup. This allows the local instance to act as a client. For each connection a unique identifier (id) must be given.	{} (an empty map in JSON format)
connections/[id]/host	IP address of the host to connect to. Host names and IPv4 addresses are permitted.	-
connections/[id]/port	Port number of the remote RCE instance.	-
connections/[id]/ connectOnStartup	If set to *true*, the connection is immediately established on startup.	true
connections/[id]/ autoRetryInitialDelay	The initial delay, in seconds, to wait after a failed or broken connection before a reconnect attempt is made. This configuration must be present to enable the auto-reconnect feature.	-
connections/[id]/ autoRetryDelayMultiplier	A decimal-point value >= 1 that the delay time is multiplied with after each consecutive connection failure. This provides an "exponential backoff" feature that reduces the frequency of connection attempts over time. This configuration must be present to enable the auto-reconnect feature.	-
connections/[id]/ autoRetryMaximumDelay	Defines an upper limit for the delay time, even when applying the multiplier would create a higher value. This can be used to maintain a minimum frequency for retrying the connection. This configuration must be present to enable the auto-reconnect feature.	-
serverPorts	A map of all server ports that the local instance registers for other instances to connect to. This allows the local instance to act as a server. For each server port a unique identifier (id) must be given.	{} (an empty map in JSON format)
serverPorts/[id]/ip	IP address to which the local instance should be bound.	-
serverPorts/[id]/port	Port number to which other instances connect to.	-
ipFilter	Allows to limit the incoming connections to a set of IP addresses.	-
ipFilter/enabled	If set to *true*, the ip filter active.	false
ipFilter/allowedIPs	List of IP addresses, which are allowed to connect to the instance.	[] (an empty list in JSON format)

Table 2.5. "componentSettings"

Configuration key	*Comment*	*Default value*
de.rcenvironment. cluster	Configuration of the cluster workflow component.	-
de.rcenvironment. cluster/ maxChannels	Maximum number of channels, which are allowed to be opened in parallel to the cluster server.	8

Table 2.6. "thirdPartyIntegration"

Configuration key	*Comment*	*Default value*
tiglViewer	Configuration of the external TiGL Viewer application integration. This needs to be configured to enable RCE´s TiGL Viewer view and thus, the TiGL Viewer workflow component. Note:TiGL Viewer must be downloaded and installed separately.	-
tiglViewer/binaryPath	The path to the TiGL Viewer executable file. Must be an absolute path.	-
tiglViewer/ startupTimeoutSeconds	The timeout in seconds, to wait for the external TiGL viewer application to start and determine its process id.	10
tiglViewer/embedWindow	If set to *false*, the external TiGL Viewer application Window will not be embeded into RCE´s TiGL Viewer view.	true

Table 2.7. "sshServer"

Configuration key	*Comment*	*Default value*
enabled	If set to *true* the local instance acts as an SSH server.	false
ip *(deprecated alias: "host")*	The host's ip address to bind to. If you want to make the SSH server accessible from remote, you should set this to the IP of the machine's external network interface. Alternatively, you can set this to "0.0.0.0" to listen on all available IPv4 addresses, if this is appropriate in your network setup.	127.0.0.1
port	The port number to which SSH clients can connect to.	-
idleTimeoutSeconds	The time to keep an idle SSH connection alive, in seconds. For typical SSH usage, the default value is usually sufficient. Higher values are, for example, needed when invoking long-running tools using the SSH Remote Access feature.	10
accounts	A map of accounts. For each account a unique identifier (account name) must be given.	{} (an empty map in JSON format)
[account name]/ passwordHash	The hashed password for the account, if password authentication is used. If the SSH account is configured using the configuration UI, the hash is automatically computed and stored here.	-
[account name]/password *(deprecated)*	The password for the account. SSH passwords can also be configured as plain text, which is however not recommended. To prevent misuse of the configured	-

Configuration key	*Comment*	*Default value*
	login data, any configuration file with SSH accounts *must* be secured against unauthorized reading (e.g. by setting restrictive filesystem permissions). A more secure alternative is to just store the password hash.	
[account name]/publicKey	The public key for the account, if keyfile authentication is used. Only RSA keys in the OpenSSH format are supported. The public key has to be entered here in the OpenSSH format (a string starting with "ssh-rsa", like it is used for example in authorized_keys files). Only applicable on RCE version 7.1 or newer.	-
[account name]/role	The role of the account. See next table for a list of the possible roles.	-
[account name]/enabled	If set to *true*, the account is enabled.	true

Table 2.8. Possible roles for SSH accounts

Role name	*Allowed commands*
remote_access_user (Standard role for using SSH remote access tools and workflows)	ra|sysmon
remote access (backwards compatibility alias for remote_access_user)	ra|sysmon
remote_access_admin	ra|ra-admin|sysmon|components
workflow_observer	components|net info|sysmon|wf list|wf details
workflow_admin	components|net info|sysmon|wf
local_admin	cn|components|mail|net|restart|shutdown|stop|stats|tasks|auth
instance_management_admin	im|net info|auth
instance_management_delegate_user	cn|components|net|restart|shutdown|stop|stats|tasks|wf|ra-admin|auth
developer	<all>

Note

The command *wf open* is only accessible to the role developer, as it influences the GUI of the server-instance.

Table 2.9. "sshRemoteAccess"

Configuration key	*Comment*	*Default value*
sshConnections	A map of SSH connections.This allows the local instance to act as a SSH remote access client. For each connection a unique identifier (id) must be given.	{} (an empty map in JSON format)
sshConnections/[id]/displayName	The name for the connection that will be shown in the network view.	-
sshConnections/[id]/host	The remote RCE instance to connect to. Host names and IPv4 addresses are permitted.	-

Configuration key	*Comment*	*Default value*
sshConnections/[id]/port	Port number of the remote RCE instance.	-
sshConnections/[id]/ loginName	The login name for authentication.	-
sshConnections/[id]/ keyfileLocation	Path to the private key file, if keyfile authentication is used. Only RSA keys in the OpenSSH format are supported.	-
sshConnections/[id]/ noPassphrase	This option should only be set if a private key that requires no passphrase is used for authentication. If set to *true*, RCE does not ask for a passphrase before connecting.	false
sshConnections/[id]/ connectOnStartup	If set to *true*, the connection is immediately established on startup. (Only possible when the password is stored in the secure store, which currently only works on Windows machines.)	false
sshConnections/[id]/ autoRetry	If set to *true*, RCE will try to automatically reconnect the connection (every 10 seconds) if it can not be established or is lost of a network error. (Only possible when the password is stored in the secure store, which currently only works on Windows machines.)	false

Table 2.10. "smtpServer"

Configuration key	*Comment*	*Default value*
host	The IP address or hostname of the SMTP server, which should be used for mail delivery.	-
port	Port number of the SMTP server.	-
encryption	Can either be "explicit" or "implicit". Select "implicit" if you want to connect to the SMTP server using SSL/ TLS. Select "explicit" if you want to connect to the SMTP server using STARTTLS. Unencrypted connections are not permitted.	-
username	The login name for authentication.	-
password	The obfuscated password for authentication. Plaintext password cannot be used here. To create the obfuscated password from the plaintext password, you need to use the Configuration UI described in Section 2.2.3, “Configuration UI”	-
sender	Email address, which should be displayed as the sender in the sent email.	-

Note

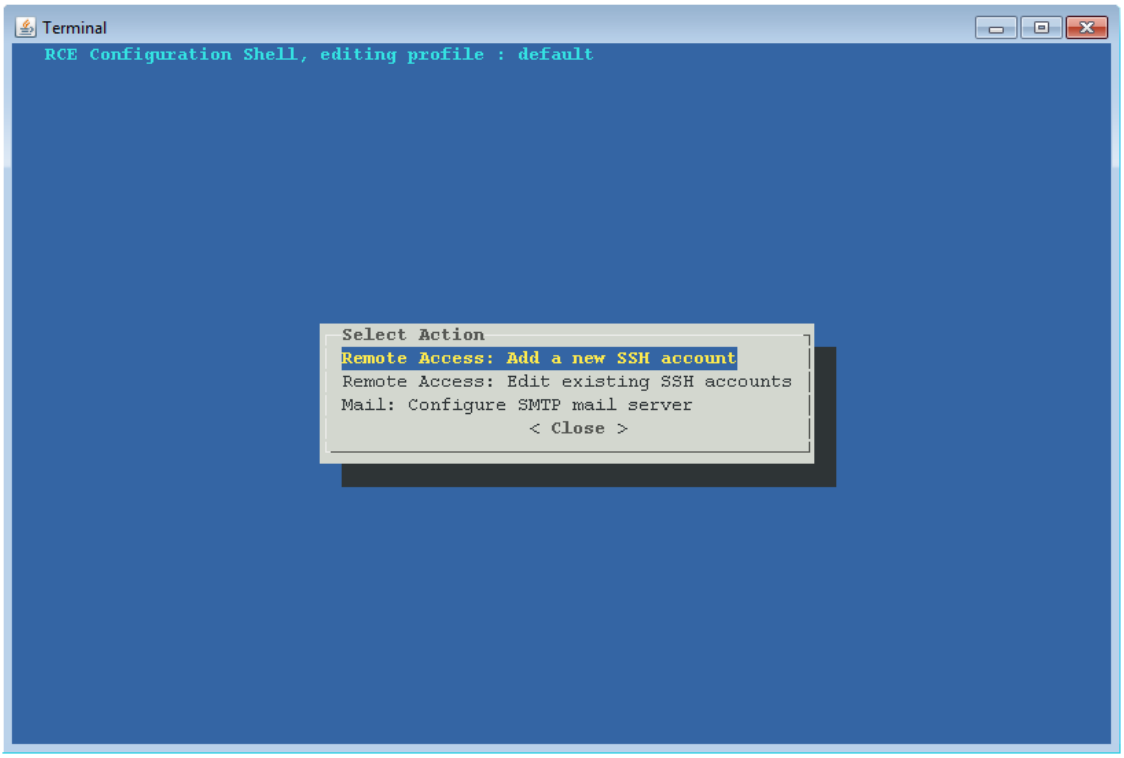
The used SMTP server needs to be configured using the Configuration UI described in Section 2.2.3.2, “Mail: SMTP server configuration”, since the password needs to be obfuscated.

2.2.3. Configuration UI

If you want to configure SSH accounts with passphrases or you want to configure e-mail support for the instance, you need to use the Configuration UI. You can access the interactive tool by executing

RCE from the command line with the option "rce --configure" or by using the "Launch Configuration UI" script in the "extras" folder of your RCE installation directory.

Figure 2.1. Configuration tool for SSH account and SMTP server configuration



2.2.3.1. Remote Access: SSH account configuration

If the RCE instance shall act as a SSH server, you can configure SSH accounts using the Configuration UI, which encrypts the SSH passwords before storing them in the configuration file.

> **Note**
>
> All SSH accounts configured with this tool initially have the role "remote_access_user", which allows to execute commands needed for remote access on tools and workflows. If you want to change the role of an SSH account, you can do this by editing the configuration file manually (see Table 2.8, “Possible roles for SSH accounts”).

2.2.3.2. Mail: SMTP server configuration

If you use the tool output verification (cf. Section 3.2.8, “Manual Tool Result Verification”) and want RCE to send the verification key via email, you need to configure an SMTP server. RCE does not send e-mails directly to the recipient, but instead sends the e-mails to an SMTP server, which delivers them to the recipient. You need to use the Configuration UI to configure such an SMTP server, since the password used for authentication needs to be obfuscated before it is stored in the configuration file. The SMTP server parameters that need to be configured are described in more detail in Table 2.10, “"smtpServer"”

> **Note**
>
> Due to a known bug on Windows system with a German keyboard layout, the Configuration UI inserts the characters "q@" into a text field if you want to insert the @ sign. You can manually remove the additional character "q".

Chapter 3. Usage

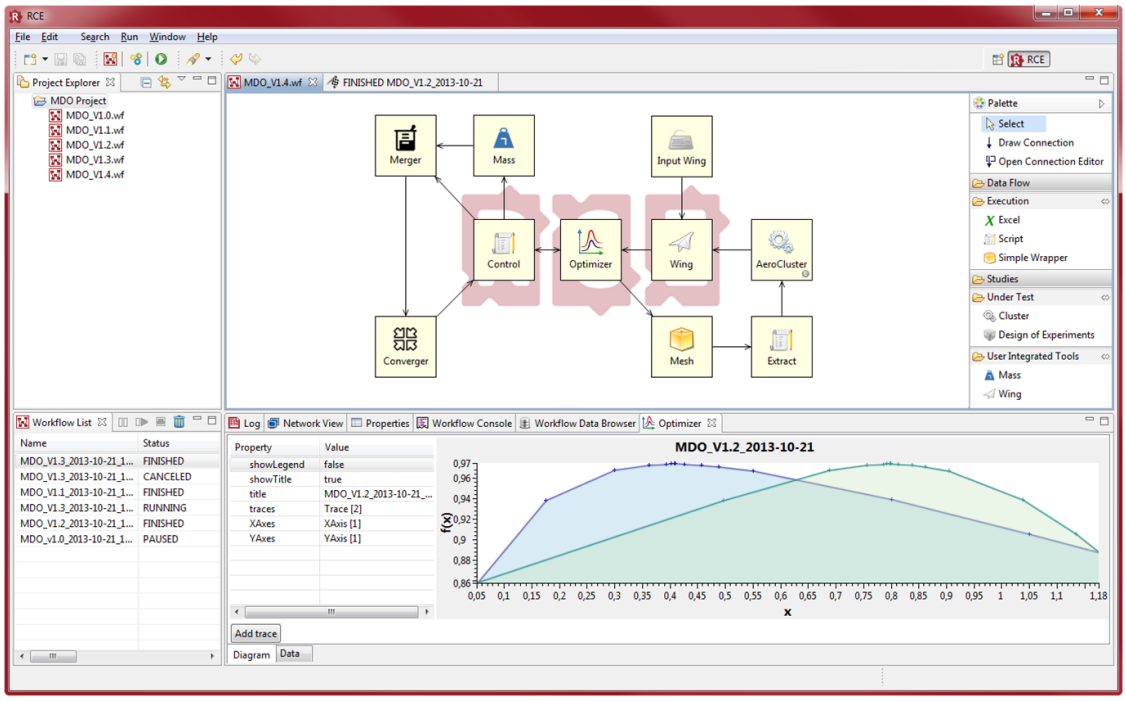
This chapter describes the main usage concepts.

3.1. Graphical User Interface

This section introduces the Graphical User Interface (GUI).

The GUI of RCE is composed of different views and editors (besides standard GUI elements such as the menu bar, status bar, etc.). Views can be (re-)arranged by the user. They can even be closed and opened again. Some views are closed by default, but can be opened as desired. To open a view go to: Window → Show view.

Figure 3.1. Workbench with different views and the workflow editor opened



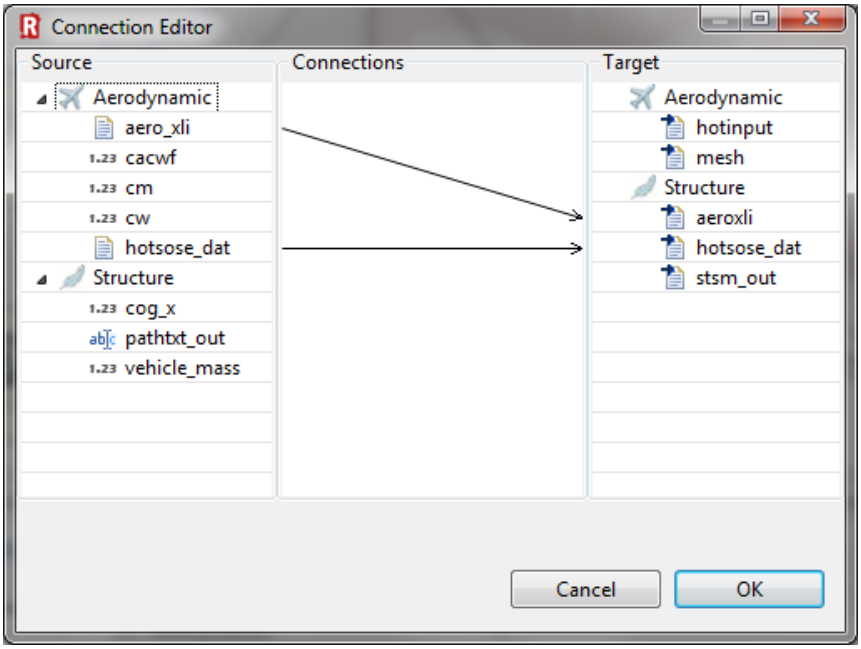
Left hand side:

- *Project Explorer:* View to manage projects. All relevant data including workflow files needs to be organized in projects.
- *Workflow List:* Lists all active workflows and allows to manage them (stop, pause, resume, dispose).

Right hand side and center:

- *Workflow Editor:* Core view of RCE used to build and configure workflows.
- *Palette:* Lists all available workflow components. If RCE runs in a distributed environment this includes local as well as remote workflow components. At the top, it also provides actions for connecting workflow components. We show the connection editor in the following Figure. Additionally, connections of the workflow are shown in the *Properties* view at the bottom, if the background of the workflow editor is selected.

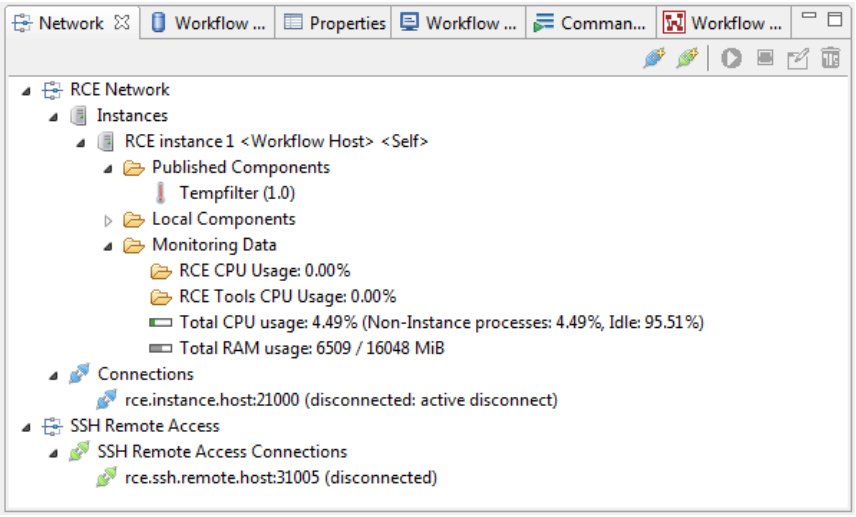
Figure 3.2. Connection Editor



Bottom:

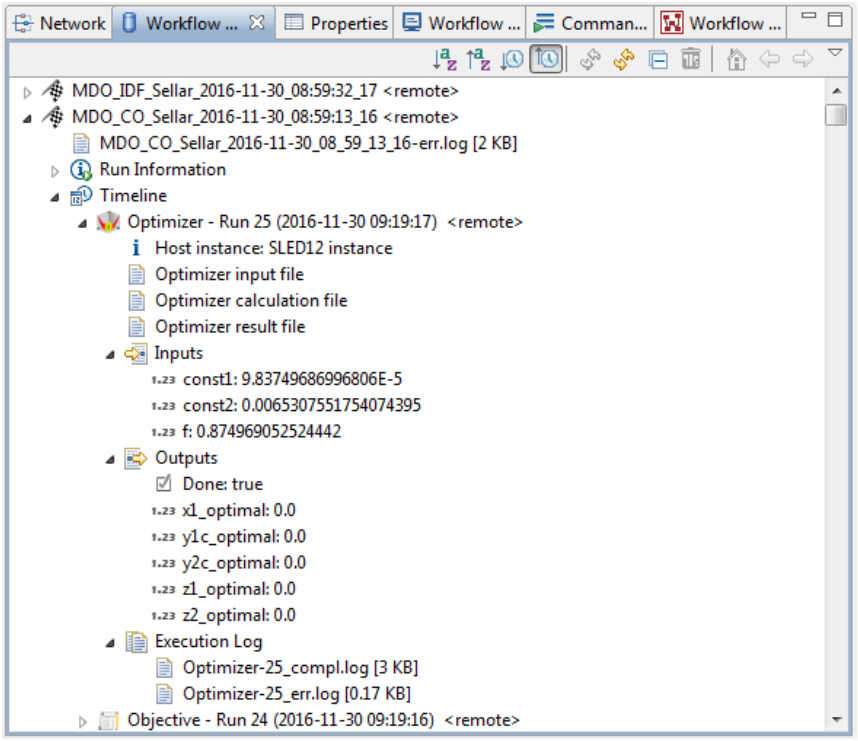
- *Log:* Shows all log output of RCE, e.g. error messages during workflow execution.
- *Network View:* Shows all RCE instances of the distributed RCE network and their published workflow components. It also shows the outgoing connections of the own RCE instance and allows to manage them (start, stop, etc.). Furthermore, you are able to see monitoring data like CPU or RAM usage for each instance.

Figure 3.3. Network View



- *Workflow Data Browser:* Shows workflow related result data.

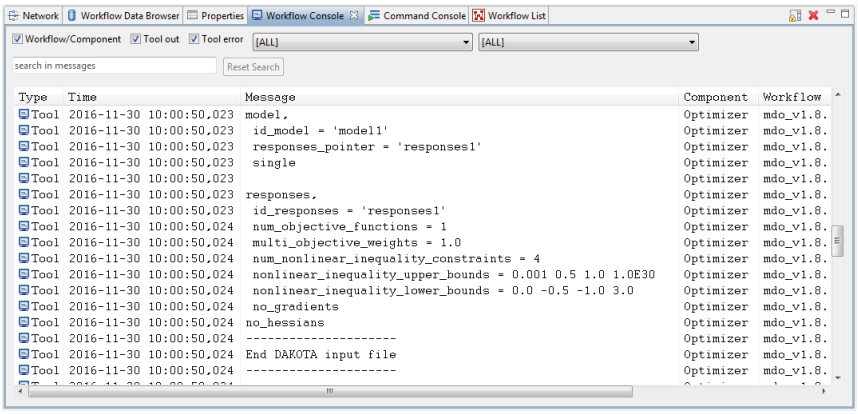
Figure 3.4. Workflow Data Browser



- *Properties:* Allows configuration of workflow components (e.g. Inputs/Outputs) if they are selected in the workflow editor. View adapts to selected workflow component.

- *Workflow Console:* Shows all native console line output of integrated tools during workflow execution. Provides full text search.

Figure 3.5. Workflow Console



3.2. Workflows

This section describes the basics of workflows in RCE.

3.2.1. Rationale

RCE is designed to execute automated, distributed workflows. Workflows consist of so called workflow components which can be coupled with each other. Loops are supported, even multi-nested ones.

3.2.2. Getting Started

To get started with workflows in RCE it is recommended to both read the following sections about workflows and walk through the example workflows provided in RCE. The sections here refer to the workflow examples where it is useful and vice versa.

Workflows in RCE are encapsulated in so called projects. To create the workflow examples project go to: File → New → Workflow Examples Project. A dialog appears. Leave the default project name or give it a name of your choice and confirm by clicking `Finish`. In the *Project Explorer* on the left-hand side, the newly created project is shown. The example workflows are grouped in sub folders. It is recommended to walk through the workflows following the prefix starting with `01_01_Hello_World.wf`.

3.2.3. Workflow Components

Workflow components are either tools that are integrated by users or are provided by RCE supplying multi-purpose functionality. The following list shows workflow components provided by RCE grouped by purpose (workflow components that are deprecated (i.e., they are removed soon) or that are not recommended to use anymore are left out):

- *Data:* Database
- *Data Flow:* Input Provider, Output Writer, Joiner, Switch
- *Evaluation:* Optimizer, Design of Experiments, Parametric Study, Converger, Evaluation Memory
- *Execution:* Script, Cluster, Excel
- *XML:* XML Loader, XML Merger, XML Values
- *CPACS:* TiGL Viewer, VAMPzero Initializer

> **Note**
>
> The Optimizer component uses the Dakota toolkit [https://dakota.sandia.gov/] in order to perform the actual optimization. This toolkit is included in the RCE distribution, i.e., it is installed together with RCE. On some systems, however, notably Ubuntu 18.04, this toolkit cannot be executed, as the required library *libgfortran3* is not installed by default. If the toolkit cannot be executed, the Optimizer component will issue the error `Could not start optimizer. Maybe binaries are missing or not compatible with system.; cause: Optimizer exited with a non zero exit code. Optimizer exit code = 127 (E#1543567120128)` or similar in the workflow console and the data management.
>
> On Ubuntu 18.04, this library can be installed by installing the package *libgfortran3*. For other systems, please refer to the documentation or the administrator of your system in order to satisfy the missing dependency of the Dakota toolkit.

The example workflows in subfolder `02_Component Groups` introduce some of the workflow components provided. Additionally, there is a documentation for each workflow component available in RCE. To access it, you can either rightclick on a component in a workflow and select `Open Help` or press `F1`. A help view opens on the right-hand side. Moreover there is an entry `Help Contents` in the `Help` menu where you can navigate to the component help you require.

The XML and CPACS components are able to read or extract data from an XML file via dynamic in- or outputs. The XPathChooser is a feature that provides help selecting the item, which shall be read or

extracted. Add an in- or output and press the `XPath choosing...` button to open a window where you can select the XML file which contains the item that shall be selected. After choosing the file, the XPathChooser opens containing a tree, symbolizing the XML file. By selecting an element, the text below is updated and displays the current path. The last two columns are used to choose attributes. The attribute name can be selected via the column `Attributes`. In the column `Values` the proper value can be selected. Use a double-click on an element to expand or fold the tree. The chosen XPath will be written in the text field of the window in which the XPathChooser has been opened originally. Using this text field, new paths can be created. Add a slash and the name of the node that shall be created to the existing path. The new path will be added during the workflow run.

New XPaths can only be generated within the inputs tab. Using the outputs tab will cause an error.

3.2.4. Coupling Workflow Components

A workflow component can send data to other workflow components. Therefore, a so called connection needs to be created between the sending workflow component and the receiving one. For that purpose, workflow components can have so called inputs and outputs. A connection is always created between an output and an input. You can think of a connection as a directed data channel. Data is sent as atomic packages which are not related to each other (there is no data streaming between workflow components). Supported data types are:

Primitive data types:

- *Short Text:*A short text (up to 140 characters)
- *Integer:*Integer number
- *Float:*Floating point number
- *Boolean:*Boolean value (true or false)

Referenced data types (The actual data is stored in RCE's data management and only a reference is transfered):

- *File:*File
- *Directory:*Directory

Other data types

- *Small Table:*Table restricted to values of type Short Text, Integer, Float, Boolean (primitive data types) (up to 100.000 cells)
- *Vector:*one-dimensional "Small Table" (one column) restricted to values of type Float
- *Matrix:*Small Table restricted to values of type Float

Not all of the workflow components support all of the data types listed. A connection can be created between an output and an input if:

- The data type of the output is the same as or convertible to the data type of the input.
- The input is not already connected to another output.

Note that data from an output can be sent to multiple inputs, but an input can just receive data from a single output.

The following table shows which data types are convertible to which other types:

Table 3.1. Data Type Conversion Table

To / From	Boolean	Integer	Float	Vector	Matrix	Small Table	Short Text	File	Directory
Boolean		x	x	x	x	x			
Integer			x	x	x	x			
Float				x	x	x			
Vector					x	x			
Matrix						x			
Small Table									
Short Text						x			
File									
Directory									

3.2.5. Execution Scheduling of Workflow Components

The execution of workflows is data-driven. As soon as all of the desired input data is available, a workflow component will be executed. Which input data is desired is defined by the component developer (for RCE's default workflow components), the tool integrator, and/or the workflow creator. The workflow component developer and tool integrator decide which options are allowed for a particular workflow component. The workflow creator can choose between those options at workflow design time. The following options exist:

Input handling:

- *Constant:* The value won't be consumed during execution and will be reused in the next iteration (if there is any loop in the workflow). The workflow will fail if there is more than one value received, except for nested loops: All inputs of type *Constant* are resetted within nested loops, after the nested loop has been finished.
- *Single (Consumed):* The input value will be consumed during execution and won't be reused in the next iteration (if there is any loop in the workflow). Queuing of input values is not allowed. If another value is received before the current one was consumed, the workflow will fail. This can guard against workflow design errors. E.g., an optimizer must not receive more than one value at one single input within one iteration.
- *Queue (Consumed):* The input value will be consumed during execution and won't be reused in the next iteration (if there is any loop in the workflow). Queuing of input values is allowed.

Execution constraint:

- *Required:* The input value is required for execution. Thus, the input must be connected to an output.
- *Required if connected:* The input value is not required for execution (e.g., if a default value will be used as fall back within the component). Thus, the input doesn't need to be connected to an output. But if it is connected to an output, it will be handled as an input of type *Required*.
- *Not required:* The input value is not required for execution. Thus, the input doesn't need to be connected to an output. If it is connected to an output, the input value will be passed to the component if there is a value available at the time of execution. Values at inputs of type *Not required* cannot trigger component execution except if it is the only input defined for a component. Note: With this option, non-deterministic workflows can be easily created. Use this option carefully. If in doubt, leave it out.

Note: With RCE 6.0.0 the scheduling options changed. Below is the migration path:

- *Initial* was migrated to *Constant* and *Required*.
- *Required* was migrated to *Single (Consumed)* and *Required*.
- *Optional* was migrated to *Single (Consumed)* and *Required if connected*.

If you encounter any problems with workflows created before RCE 6.0.0, it is likely, that it affects the migration to *Single (Consumed)* instead of to *Queue (Consumed)*. We decided to migrate conservatively to not hide any existing workflow design errors. So, if queuing of input values is allowed for an input, just change the input handling option to *Queue (Consumed)* after the workflow was updated. Another issue can affect the migration of *Optional*. If it affects an input of the script component, check the option, which let the script component execute on each new value at any of its inputs. Also check *Not required* as an alternative execution constraint option.

3.2.6. (Nested) Loops

Workflow components can be coupled to loops. A loop must always contain a so-called driver workflow component. Driver workflow components (group "Evaluation") are: Optimizer, Design of Experiments, Parametric Study, Converger (see the example workflow "02_02_Evaluation_Drivers"). The responsibilities of a driver workflow component in a loop are:

- Send values to the loop and receive the result values.
- Finish the loop based on some certain criteria.

If a loop contains another loop, we speak of the latter as a nested loop. A nested loop can contain again another loop and so on. To create workflows with nested loops (see example workflows in "03_Workflow_Logic"), some certain concepts behind nested loops must be understood:

- Loop level: If a loop contains another loop, that loop is considered as a nested loop with a lower loop level. From the perspective of the nested loop, the other loop is considered as a loop with an upper loop level.
- If a driver workflow component is part of a nested loop, you need to check the checkbox in the "Nested Loop" configuration tab
- Data exchange between loops of different loop levels is only allowed via a driver workflow component. Thereby, only particular inputs and outputs of driver workflow components are allowed to be connected to inputs and outputs of the next upper loop level and particular ones to inputs and outputs of the same loop level. For example, if a 'same loop level' output is connected to a loop with an upper loop level, the workflow won't succeed or might even get stuck. Below you find tables of inputs and outputs for each driver workflow component and whether they must be connected to the same loop level or to the next upper loop level.

> **Note**
> In the inputs and outputs tables of driver workflow components (in 'Inputs/Outputs' properties tab), the loop level requirement is present in a particular column for each input and output.

Table 3.2. Inputs of Optimizer

Input	Loop Level
* - lower bounds - start value	To next upper loop level
* - upper bounds - start value	To next upper loop level

Input	Loop Level
* - start value	To next upper loop level
* (Objective functions)	To same loop level
* (Constraints)	To same loop level
d*.d* (Gradients)	To same loop level

Table 3.3. Outputs of Optimizer

Output	Loop Level
*_optimal	To next upper loop level
Done	To next upper loop level
* (Design variables)	To same loop level
Gradient request	To same loop level
Iteration	To same loop level

Table 3.4. Inputs of Design of Experiments

Input	Loop Level
*_start	To next upper loop level
*	To same loop level

Table 3.5. Outputs of Design of Experiments

Output	Loop Level
Done	To same loop level
*	To same loop level

Table 3.6. Inputs of Parametric Study

Input	Loop Level
*_start	To next upper loop level
*	To same loop level

Table 3.7. Outputs of Parametric Study

Output	Loop Level
Done	To same loop level
*	To same loop level

Table 3.8. Inputs of Converger

Input	Loop Level
*_start	To next upper loop level

Input	Loop Level
*	To same loop level

Table 3.9. Outputs of Converger

Output	Loop Level
Converged	To next upper loop level
Converged absolute	To next upper loop level
Converged relative	To next upper loop level
*_converged	To next upper loop level
Done	To same loop level
*	To same loop level

3.2.7. Fault-tolerant Loops

Workflow components of a loop can fail. There are two kind of failures:

- A workflow component fails gracefully, i.e. it couldn't compute any results for the inputs received but works normally. In this case, it sends a value of type "not-a-value" with the specified cause to its outputs which finally are received by the driver workflow components as results.
- A workflow component fails, i.e. it crashes for an unexpected reason. In this case, the workflow engine sends values of type "not-a-value" with the specified cause as results to the driver workflow component.

In the "Fault Tolerance" configuration tab of workflow driver components, it can be configured how to handle failures in loops, for both kind of failures separately.

3.2.8. Manual Tool Result Verification

After the execution of an integrated tool, the results are sent via outputs to the next workflow component (e.g. to the next integrated tool). By default, this is done in an automated manner without any user interaction. If the data should be verified by a person responsible for the tool before they are sent further, manual verification of tool results must be enabled in the tool integration wizard in the 'Verification' tab of the 'Inputs and Outputs' page.

In case manual verification of tool results is enabled, the results are hold after each tool execution and the corresponding workflow component remains in state "Waiting for approval". Then, there are two options:

- Approve tool results: The tool results are sent via the outputs to the next workflow component and the workflow continues normally.
- Reject tool results: The tool results are not sent via the outputs to the next workflow component and the workflow is cancelled.

To apply one of the options, a so called verfication key is required. The verification key is generated by RCE after each tool execution and is written to a file on the file system of the machine which executed the tool. (The location is specified in the 'Verification' tab of the 'Inputs and Outputs' page in the tool integration wizard.) Optionally, the verification key can also be sent via e-mail if e-mail support is

configured for the RCE instance where the tool is integrated. (E-mail support can only be configured using the Configuration UI as described in Section 2.2.3.2, “Mail: SMTP server configuration”) E-mail delivery can be enabled and the recipients can be defined in the 'Verification' tab of the 'Inputs and Outputs' page in the tool integration wizard.

Once the verification key is known (either from the file or an e-mail), perform follwing steps to approve or reject the tools results:

- Start an RCE instance with a graphical user interface. (Your tool must be available, i.e. it must appear in the palette of the workflow editor.)
- In the menu bar at the top, go to Run -> Verify tool results...
- A dialog appears that guides you through the verification process.

3.3. Commands

This section introduces the list of commands available for the command line and the interactive shell.

3.3.1. Command Line Parameters

General syntax

```
>  rce --[RCE arguments] -[RCP arguments] -[VM arguments]
```

Table 3.10. Command line arguments for RCE

Argument	Type	Description
profile "<profile id or path>"	RCE	Sets a custom profile folder to use. If only an id (any valid directory name) is given, the profile directory "<user home>/.rce/*id*" is used. Alternatively, a full filesystem path can be specified.
profile	RCE	If the profile argument is specified without a profile id or path, RCE launches the Profile Selection UI, which allows to select a profile folder for the startup as described in Section 3.3.2, “Profile Selection UI”.
batch "<command string>"	RCE	Behaves like the "exec" command, but also implies the "--headless" option and always shuts down RCE after execution.
headless	RCE	Starts RCE in a headless modus without GUI. It will remain in the OSGi console and waits for user input.
exec "<command string>"	RCE	Executes one or more shell commands defined by <command string>. For the list of available commands, refer to the command shell documentation. This argument is usually used together with --headless to run RCE in batch mode. Multiple commands can be chained within <command string> by separating them with " ; " (note the spaces); each command is completed before the next is started. You can use the "`stop`" command at the end of the command sequence to shut down RCE after the other commands have been executed. However, any error

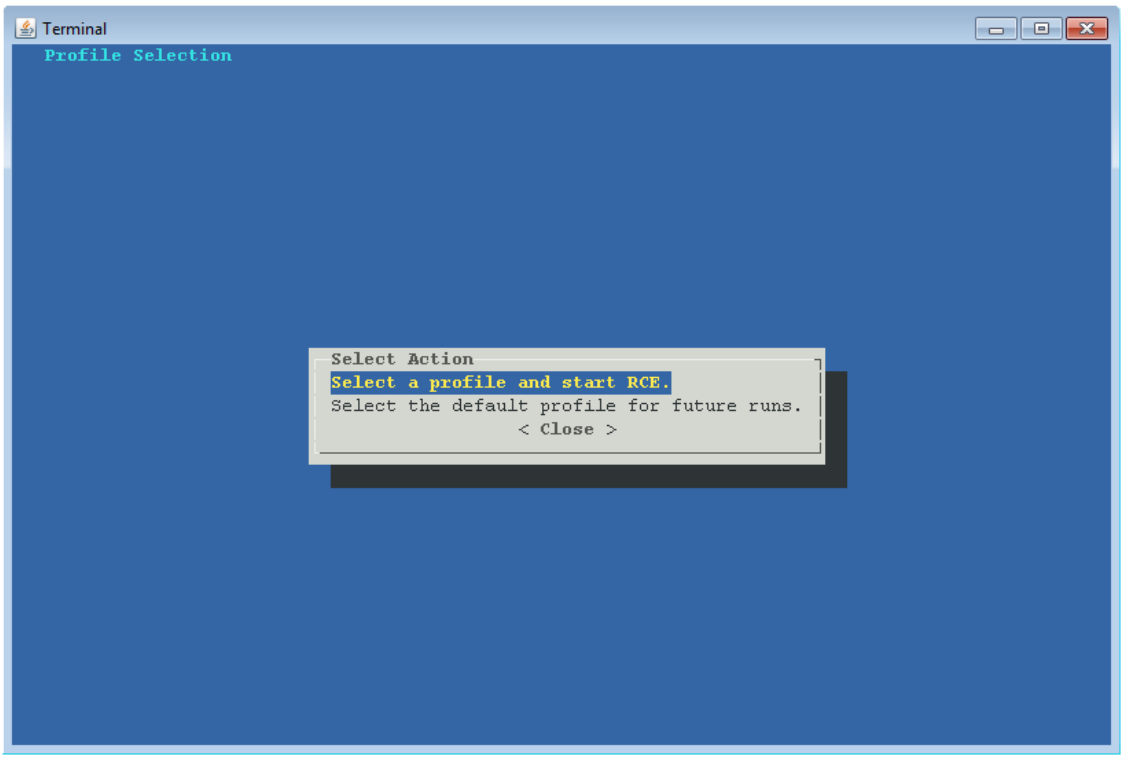
Argument	Type	Description
		during execution of these commands will cancel the sequence, and prevent the "`stop`" command from being executed. To ensure shut down at the end of the command sequence, use the `--batch` option instead of "`--exec`". As an example, `rce --headless --exec "wf run example.wf ; stop"` will execute the "example.wf" workflow in headless mode and then shut down RCE. However, if the workflow fails to start, RCE will keep running, as the "`stop`" command is never executed. To attempt execution of the workflow file, but then always shut down regardless of the outcome, use `rce --batch "wf run example.wf"` instead.
configure	RCE	Starts the RCE Configuration UI (Section 2.2.3, “Configuration UI”) which can be used to configure SSH accounts with passphrases or to configure e-mail support for the RCE instance.
data @noDefault	RCP	Set the default workspace location to empty
consoleLog	RCP	Logs everything for log files on the console as well.
console	RCP	Runs RCE with an additional OSGi console window, which allows you to execute RCE shell commands. See the Command Shell documentation for more information.
Deprecated: console <port>	RCP	Specify the port that will be used to listen for telnet connections. (*NOTE:* this access is insecure; configure SSH access instead)
clean	RCP	Cleans before startup
vmargs	VM	Standard JVM arguments
Dde.rcenvironment.rce.configuration.dir=<insert-config-path>	VM	Sets the configuration directory
Drce.network.overrideNodeId =<some-id>	VM	Sets the local node id, overriding any stored value. This is mostly used for automated testing. Example: "-Drce.network.overrideNodeId=a96db8fa762d59f2d2782f3e5e9662d4"
Dcommunication.uploadBlockSize=<block size in bytes>	VM	Sets the block size to use when uploading data to a remote node. This is useful for very slow connections (less than about 10 kb/s) to avoid timeouts. The default value is 262144 (256 kb). Example: "-Dcommunication.uploadBlockSize=131072" - sets the upload block size to 128kb (half the normal size)

3.3.2. Profile Selection UI

During startup of the instance, the Profile Selection UI allows to select a profile folder which should be used for the current run of RCE. Furthermore it allows to specify a default profile for future runs.

You can access the Profile Selection UI by executing RCE from the command line with the option "rce --profile".

Figure 3.6. Profile Selection UI



If the first option "Select a profile and start RCE" is chosen, a list of available profiles is presented. On selection of one of these profiles, RCE is started using this profile.

If the second option "Select the default profile for future runs" is chosen, a list of available profiles is presented. On selection of one of these profiles, RCE will not be started using this profile, but instead the selected profile will be marked as the default profile for future runs. This selection can be temporarily overwritten again by using the '-profile "<profile id or path>"' option. The default profile setting will be stored for the current user and the current installation location of RCE. Different users on the same machine can therefore configure different default profiles. Furthermore, different installations of RCE can have different default profiles configured.

Note

The Profile Selection UI will only display profiles if they have been started once with RCE 7.0 or newer.

3.3.3. Command Shell

RCE provides an integrated shell (sometimes referred to as "console") for executing commands. It can be accessed in three different ways:

- Start RCE with the "-console" command-line option, or add "-console" to the rce.ini file before starting; this will open an OSGi console window. Due to the nature of an OSGi console, all RCE commands must be prefixed with "rce". For example, type "rce help" to show the available commands.
- *Deprecated:* Start RCE with the "-console <port>" command-line option; this will accept telnet OSGi console sessions on that port. As with the "-console" option, RCE commands must be prefixed with "rce" (for example, type "rce help").

Note that this option is *insecure*, as there is no authentication nor encryption, so it should only be used in fully trusted networks. Whenever possible, use the SSH console (see below) instead .

- Configure SSH access. To do so, refer to Section Configuration Parameters. After RCE has started, you can access the shell on the configured port with a standard SSH client. On Windows systems, the "putty" software works well as a client.

  As this option creates a pure RCE shell (as opposed to the OSGi consoles created above), you can enter RCE commands without a prefix - for example, just type "help" to list the available commands. Note that to avoid confusion, adding a "rce" prefix still works, but it is not necessary.

The following table lists some shell commands; more documentation coming soon.

Table 3.11. Shell Commands

Argument	Description
help	Lists all available commands.
auth	Short form of "auth list".
auth create <group id>	Creates a new authorization group whith the given <group id> (an identifier consisting of 2-32 letters, numbers, underscores ("_") and/or brackets).
auth delete <group id>	Deletes the local authorization group with the given <group id>.
auth export <group id>	Exports the group with the given group id as an <invitation string> that can be imported by another node, allowing that other node to join this group.
auth import <invitation string>	Imports a group from an <invitation string> that was previously exported on another node.
auth list	Lists the authorization groups that the local node belongs to.
cn	Short form of "cn list".
cn add <target> ["<description>"]	Adds a new network connection. (Example: cn add activemq-tcp:rceserver.example.com:20001 "Our RCE Server")
cn list	Lists all network connections, including ids and connection states.
cn start <id>	Starts/Connects a READY or DISCONNECTED connection (use "cn list" to get the id).
cn stop <id>	Stops/Disconnects an ESTABLISHED connection (use "cn list" to get the id).
components	Short form of "components list".
components list [--local] [--as-table]	Lists components published by reachable RCE nodes. The "--local" option only lists components provided by the local node. The "--as-table" option formats the output as a table that is especially suited for automated parsing.
components list-auth	Shows a list of all defined authorization settings. Note that these settings are independent of whether a matching component exists, which means that settings are kept when a component is removed and later added again.
components set-auth <component id> <groups>	Assigns a list of authorization groups to a component id. Note that authorization settings always apply to all components with using this id, regardless of the component's version. The <component id> needs to be defined as listed by the "components list" command, e.g. "rce/Parametric Study", "common/MyIntegratedTool", or "cpacs/MyCpacsTool". This id must be enclosed in double quotes if it contains spaces.

Argument	Description
	The <groups> to assign need to be provided as comma-separated list of user-defined authorization groups. This replaces any previously assigned groups. Note that the specified groups must have been created or imported beforehand; see the "auth create" and "auth import" commands for details. Instead of a list of groups, the special value "public" can be used to grant access to any user within the visible network, while "local" revokes any previously granted access by remote users.
mail <recipient> <subject> <body>	Sends an email to the specified recipient.
net	Short form of "net info".
net filter	Shows the status of the IP whitelist filter.
net filter reload	Reloads the IP whitelist configuration.
net info	Lists all reachable RCE nodes.
ra-admin list-wfs	Lists the ids of all published workflows.
ra-admin publish-wf [-g <group name>] [-k] [-t] [-p <JSON placeholder file>] <workflow file> <id>	Publishes a workflow file for remote execution via "ra run-wf" using <id>. -g name of the group in which the workflow will be shown in the Palette on the client instance -k (keep execution data): if set, the workflow execution data will not be deleted after the workflow is run -t (temporary/transient): if set, the workflow is automatically unpublished when the RCE instance is shut down -p: adds a placeholder file for the given workflow; see the "wf run" command's documentation for details. This operation verifies that the workflow contains the required standard elements before publishing. Note that a snapshot of the workflow file (and optionally, the given placeholder file) is taken before publishing; subsequent changes of the workflow file do NOT affect the published workflow.
ra-admin unpublish-wf <id>	Unpublishes (removes) the workflow file with the given <id> from remote execution.
restart	Restarts RCE.
shutdown	Shuts down the local RCE instance.
ssh	Short form of "ssh list".
ssh add <displayName> <host> <port> <username> <keyfileLocation>	Adds a new ssh connection.
ssh list	Lists all ssh connections, including ids and connection states.
ssh start <id>	Starts/connects the ssh connection with the given <id> (use " ssh list" to get the id).
ssh stop <id>	Stops/disconnects the ssh connection with the given <id> (use " ssh list" to get the id).
stop	Shuts down the local RCE instance (alias of "shutdown").
sysmon api <operation>	Fetches system monitoring data from all reachable nodes in the network, and prints it in a parser-friendly format.

Argument	Description
	Available operations: avgcpu+ram <time span> <time limit> - fetches the average CPU load over the given time span and the current free RAM. Operation parameters: time span - the maximum time span (in seconds) to aggregate load data over time limit - the maximum time (in milliseconds) to wait for each node's load data response.
sysmon local/-l	Prints system monitoring data for the local instance.
sysmon remote/-r	Fetches system monitoring data from all reachable nodes in the network, and prints it in a human-readable format.
version	Shows version information.
wf	Short form of "wf list"
wf list	Lists all current workflows, their states and execution ids.
wf cancel <workflow execution id>	Cancels a running or paused workflow.
wf delete <workflow execution id>	Deletes a finished, cancelled or failed workflow from the data management and disposes it.
wf details <workflow execution id>	Shows details about one workflow.
wf open <workflow execution id>	Opens a runtime viewer of a workflow. Requires GUI. When using SSH, this command is only available to users with the role *developer*.
wf pause <workflow execution id>	Pause a running workflow.
wf resume <workflow execution id>	Resume a paused workflow.
wf run [--delete <onfinished|always|never>] [--compact-output] [-p <placeholder value file>] <workflow file>	Executes the given workflow file and waits until it has completed. Workflow file paths containing spaces must be enclosed in double quotes ("..."). The "--delete" option defines the deletion behavior after workflow completion. Deleting a workflow deletes all of its files in the data management and releases certain resources that may or may not be used after it has finished, for example data to be visualized in component's runtime views. The default of this setting is "onfinished": The workflow is deleted if it terminates in state "Finished" (which means normal completion without errors), otherwise it is left unchanged for inspection. The "--dispose" option defines the deletion behavior from the workflow list. Disposing a workflow does not delete its data from the data management. The default of this setting is "onfinished". The "--compact-output" option reduces this command's output as much as possible, which is intended to simplify scripted calls of this command. The first line printed will either be the workflow's assigned id if the start was successful, or a text starting with `"Error "` if the workflow could not be started. If (and only if) the start was successful, a second line will be printed once the workflow has terminated. The pattern of this second line is "`<workflow id>: <final state>`". The "-p" option can be used to define a placeholder value file (see below).
wf verify [--delete <onfinished|always|never>] [--pr	Runs several workflows and creates a summary of which ones failed and succeeded.

Argument	Description
<parallel runs>] [--sr <serial runs>] [-p <placeholder value file>] --basedir <directory> <workflow file> [<workflow file> ...]	The "--pr" option defines how often the workflow is started in parallel. The "--sr" options defines how often the workflow is started in serial. E.g. "--pr 5 --sr 3" starts the workflow three times with five in parallel. If "*" is used with the "--basedir" option or multiple workflow filenames are passed, "--pr" and "--sr" are applied for each of the workflows. For the "--delete", "dispose" and "-p" options refer to "wf run" above. The "--basedir <directory>" parameter specifies the directory containing the workflow files. File paths containing spaces must be enclosed in double quotes ("..."). The second parameter defines the workflow's filenames. Using "*" as workflow file runs all non-backup workflows in the basedir. Workflow file paths containing spaces must be enclosed in double quotes ("...").

3.3.3.1. Configuration Placeholder Value Files

Some workflow components use placeholders for configuration values. The values for the placeholders are defined at workflow start. When executing workflows from the command line (e.g. in headless or batch mode), the placeholder's values must be defined in a file, which will be passed to the command with the -p option. Placeholder value files have following format:

```
{

<component id>/<component version> : {

<configuration placeholder id> : <configuration value>

}

<component id>/<component version>/<component instance name> : {

<configuration placeholder id> : <configuration value>

}

}
```

> **Note**
>
> Every id and every value must be in enclosed in double quotes ("...").

The `component id` is the id string of a component (e.g. de.rcenvironment.script), the `component version` is the version of the component that is used in the workflow (e.g. 3.4).

There are two ways of defining values for configuration placeholders: per component type and per component instance. When defined per component type, the id and version must be specified (e.g. "de.rcenvironment.script/3.4"). When defined per component instance the component id, component version, and the name of the component in the workflow must be specified (e.g. "de.rcenvironment.script/3.4/MyScript"). In both cases, the `configuration placeholder id`, which is the name of the configuration placeholder, and the actual `configuration value` must be specified.

Component instance values override component type values.

Note

It is possible to mix component type and component instance values.

Below is an example placeholder value file, which defines one placeholder value (component type) for the script component, one placeholer value (component type) for the input provider component and a placeholder value (component instance) for a specified input provider component of the workflow:

```
{
"de.rcenvironment.script/3.4": {
"pythonExecutionPath": "C:/Python/python.exe"
},
"de.rcenvironment.inputprovider/3.2": {
"inputFile": "C:/input/globalInputFile.txt"
},
"de.rcenvironment.inputprovider/3.2/Provider 1" : {
"inputFile": "C:/input/Provider1.txt"
}
}
```

The following table lists components and their configuration placeholders.

Table 3.12. Components and their configuration placeholders

Component	Component id and version	Configuration placeholders
Cluster	de.rcenvironment.cluster/3.2	authuser - user name authphrase - password (base64 encoded)
Input Provider	de.rcenvironment.inputprovider/3.2	<output name> - value of output
Output Writer	de.rcenvironment.outputwriter/2.0	targetRootFolder - path to target root folder
Script	de.rcenvironment.script/3.4	pythonExecutionPath - path to the Python executable (only required if Python is set as script language)

3.4. Integration of External Tools

3.4.1. Basic Concepts

The Tool Integration concept of RCE is used to integrate external tools for calculations, simulations and so on into RCE and use them in a workflow as a component. The tools must fulfill these requirements:

- The external tool must be callable via command line
- It must have a non-interactive mode which is called via command line
- Input for the tool must be provided through command line arguments or files

If these requirements are fulfilled, a configuration file can be created that is used for the integration.

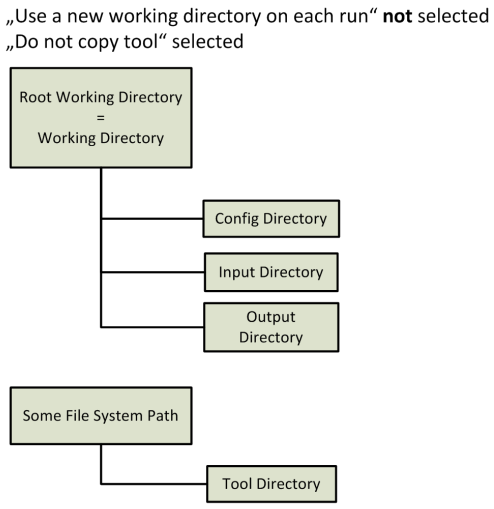
If you use RCE with a graphical user interface this can be done with the help of an wizard which guides you through the settings. This wizard can be found in the menu *Tool Integration -> Integrate Tool...*. Required fields are marked with an asterisk (*). When the wizard is finished and everything is correct, the integrated tool will automatically show up in the Workflow Editor palette.

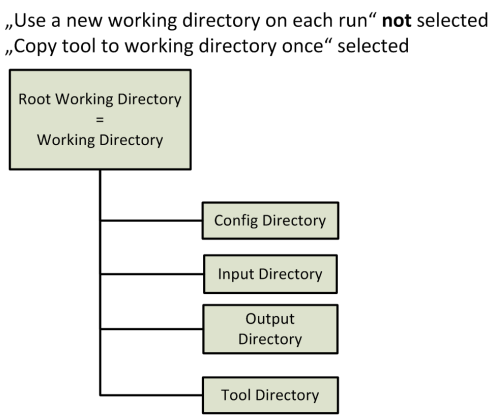
Note

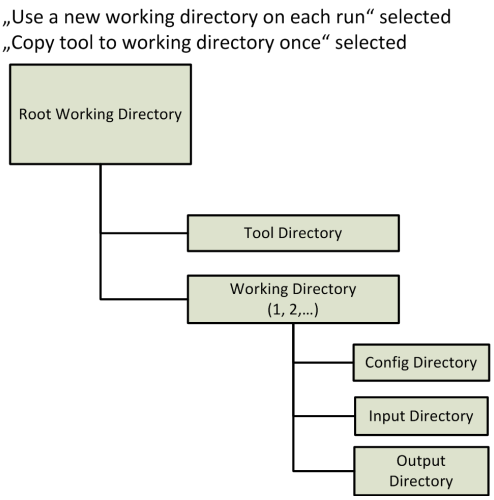
The wizard has a dynamic help, which is shown by clicking on the question mark on the bottom left or by pressing F1. It will guide you through every page of the wizard.

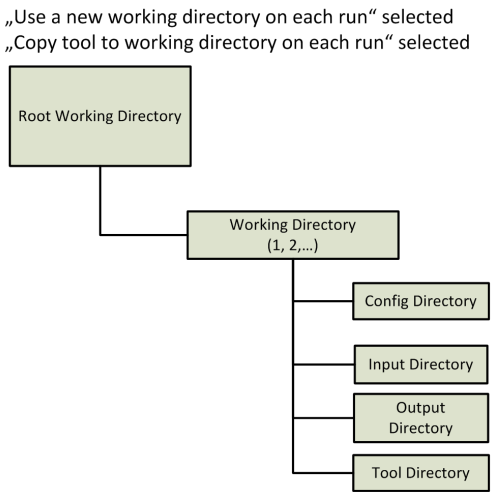
3.4.2. Directory Structure for Integrated Tools

When executing an integrated tool, a certain directory structure is created in the chosen working directory. This structure depends on the options you have chosen in the integration wizard. The two options that matter are "Use a new working directory each run" and "Tool copying behavior".









Root Working Directory: This is the directory you choose in the "Tool Integration Wizard" as "Working Directory" on the "Launch Settings" page.

Config Directory: In this directory, the configuration file that may be created by the tool integration will be created by default. The configuration files can be created from the properties that are defined for the tool on the "Tool Properties" page.

Input Directory: All inputs of type "File" and "Directory" will be copied here. They will have a subdirectory that is named exactly as the input's name (e.g. the input "x" of type "File" will be put into "Input Directory/x/filename").

Output Directory: All outputs of type "File" and "Directory" can be written into this directory. After that, you can use the placeholder for this directory to assign these outputs to RCE outputs in the post execution script. To write the output directory into an output "x" of type "Directory" the following line in the post execution script would be required: ${out:x} = "${dir:output}"

Tool Directory: This is the directory, where the actual tool is located. If the tool should not be copied, it will be exactly the same directory that you choose, otherwise it will be the same as the chosen directory but copied to the working directory.

Working Directory: A working directory is always the location, where all the other directories will be created. If the option "Use a new working directory on each run" is disabled, this will always be the same as the "Root Working Directory". Otherwise, a new directory is created each run (the name will be the run number) and is the working directory for the run.

3.4.3. Copying of Integrated Tools

When a component is created in the integration wizard, a configuration file is created.

All configuration files from the tool integration are stored in the directory

```
<profile folder>/integration/tools/
```

In this directory, there is a separation between different kinds of integration realized through one subdirectory for each. The `common` folder is always existent.

In these subdirectory, the integrated tools are stored, again separated through a subdirectory for each. The name of the directory equals the name of integration of the tool.

If an integrated tool should be copied to another RCE instance or another machine, the directory of the tool must be copied, containing a `configuration.json` and some optional files. It must be put in the equivalent integration type directory of the target RCE instance. After that, RCE automatically reads the new folder and if everything is valid, the tool will be integrated right away.

Note

If you want to delete a tool folder that contains some documentation, this can cause an error. If you have this problem, delete the documentation folder at first (it must be empty), afterwards you can delete the tool folder.

3.4.3.1. Tool Execution Return Codes

The tools are executed by using a command line call on the operating system via the "Execution Script". When the tool finished executing (with or without error), its exit code is handed back to the execution script and can be analyzed in this script. If in the script nothing else is done, the exit code is handed back to RCE. When there is an exit code that is not "0", RCE assumes that the tool crashed and thus lets the component crash without executing the "Post Script". Using the option "Exit codes other

than 0 is not an error" can prevent the component to crash immediately. With this option enabled, the post script wil be executed in any way and the exit code from the tool execution can be read by using the placeholder from "Additional Properties". In this case, the post script can run any post processing and either not fail the component, so the workflow runs as normal, or let the compoennt crash after some debugging information was written using the Script API "RCE.fail("reason")".

3.4.4. Integration of CPACS Tools

3.4.4.1. Additional concepts of CPACS Tool Integration

Extending the common Tool Integration concept, the CPACS Tool Integration has some additional features.

- Parameter Input Mapping (optional): Substitutes single values in the incoming CPACS content, based on an XPath configured at workflow design time as a dynamic input of the component
- Input Mapping: Generates the tool input XML file as a subset of the incoming CPACS file XML structure, specified by a mapping file
- Tool Specific Input Mapping (optional): Adds tool specific data to the tool input file, based on a mapping file and a data XML file
- Output Mapping: Merges the content of the tool output XML file into the origin incoming CPACS file, based on a mapping file
- Parameter Output Mapping (optional): Generates output values as single values of the CPACS result file, based on an XPath configured at workflow design time as a dynamic output of the component
- Execution option to only run on changed input: If enabled, the integrated tool will only run on changed input. Therefore the content of the generated tool input file is compared to the last runs content. Additionally the data of the static input channels are compared to the previous ones.

All the features listed above can be configured in the tool integration wizard on the dedicated *CPACS Tool Properties* page.

The mappings can be specified by XML or XSLT as shown in the following examples. RCE differentiates between these methods in accordance to the corresponding file extension (.xml or .xsl).

Example for an input or tool specific XML mapping :

```
<?xml version="1.0" encoding="UTF-8"?>
<map:mappings xmlns:map="http://www.rcenvironment.de/2015/mapping" xmlns:xsl="http://www.w3.org/1999/
XSL/Transform">

    <map:mapping>
        <map:source>/cpacs/vehicles/aircraft/model/reference/area</map:source>
        <map:target>/toolInput/data/var1</map:target>
    </map:mapping>

</map:mappings>
```

And input or tool specific XSLT mapping:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="cpacs_schema.xsd">
 <xsl:output method="xml" media-type="text/xml" />
 <xsl:template match="/">
  <toolInput>
   <data>
    <var1>
```

```
    <xsl:value-of select="/cpacs/vehicles/aircraft/model/reference/area" />
   </var1>
  </data>
 </toolInput>
</xsl:template>
</xsl:stylesheet>
```

Example of an output XML mapping:

```
<?xml version="1.0" encoding="UTF-8"?>
<map:mappings xmlns:map="http://www.rcenvironment.de/2015/mapping" xmlns:xsl="http://www.w3.org/1999/
XSL/Transform">

    <map:mapping mode="delete">
        <map:source>/toolOutput/data/result1</map:source>
        <map:target>/cpacs/vehicles/aircraft/model/reference/area</map:target>
    </map:mapping>

</map:mappings>
```

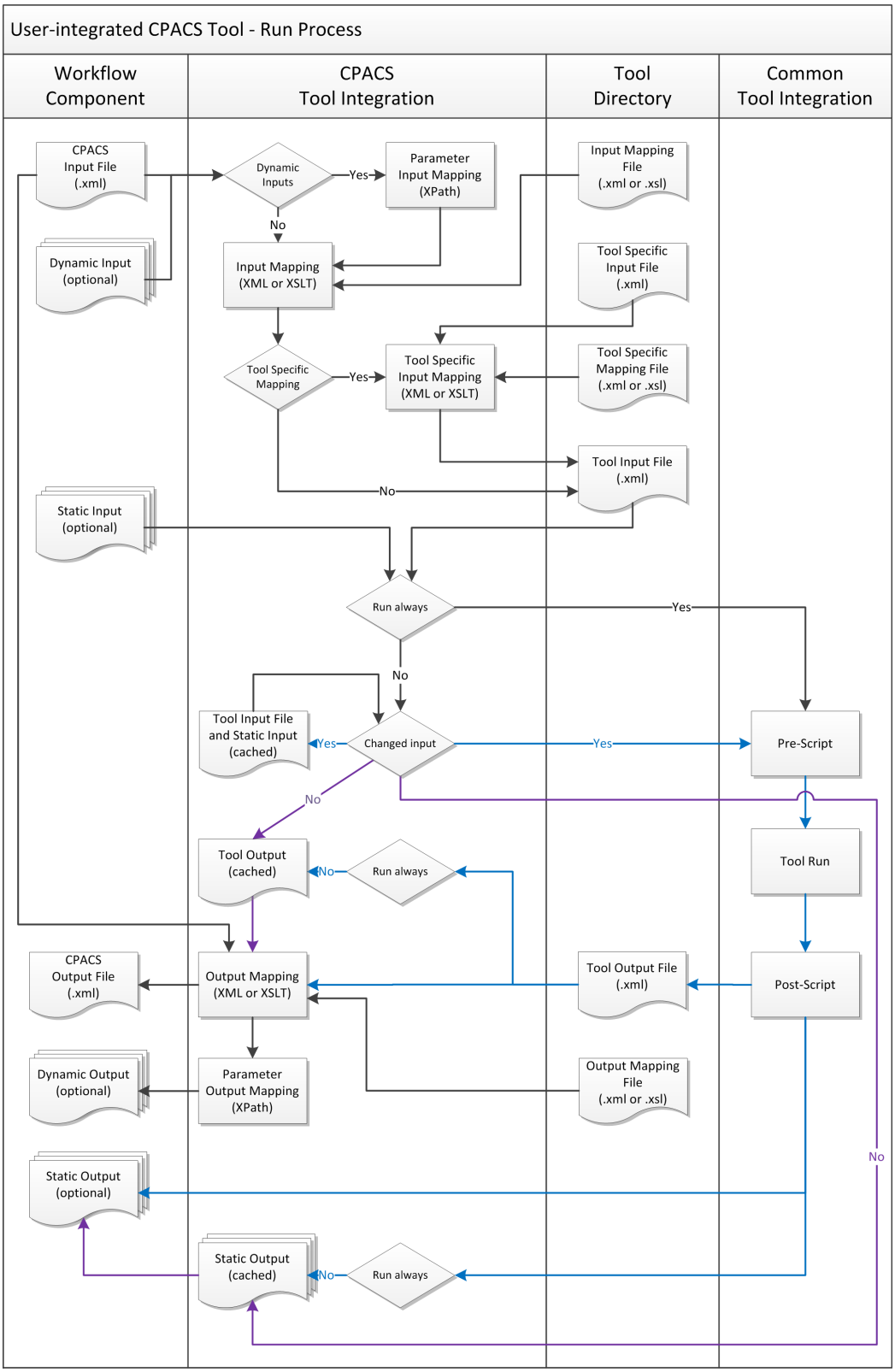
And output XSLT mapping:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" exclude-result-prefixes="xsi">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <!--Define Variable for toolOutput.xml-->
  <xsl:variable name="toolOutputFile" select="'./ToolOutput/toolOutput.xml'"/>
  <!--Copy complete Source (e.g. CPACSInitial.xml) to Result (e.g. CPACSResult.xml)-->
  <xsl:template match="@* | node()">
    <xsl:copy>
      <xsl:apply-templates select="@* | node()"/>
    </xsl:copy>
  </xsl:template>
  <!--Modify a value of an existing node-->
    <xsl:template match="/cpacs/vehicles/aircraft/model/reference/area">
    <area>
    <xsl:value-of select="document($toolOutputFile)/toolOutput/data/result1"/>
    </area>
    </xsl:template>
</xsl:stylesheet>
```

Please ensure to use the proper namespace for map (xmlns:map="http://www.rcenvironment.de/2015/mapping") in XML mapping files.

The figure below illustrates how the additional features are used in the run process of an user-integrated CPACS tool.

Figure 3.7. Run process of an user-integrated CPACS Tool



3.4.4.2. Integrate a CPACS Tool into a Client Instance

1. Start RCE as Client
2. Open the *Tool Integration Wizard* by clicking the *Integrate Tool...* in the *File* menu.

> **Note**
>
> You will always find further help by clicking the *?* on the bottom left corner on each page of the wizard or by pressing *F1*.

3. Choose the option *Create a new tool configuration from a template*.

> **Note**
>
> The CPACS templates delivered with RCE are designed to match the conventions of the old CPACS tool wrapper (respectively ModelCenter tool wrapper). Most of the properties are preconfigured and do not need to be changed.

4. Select one of the *CPACS* templates.
   Click *Next*.

5. Fill in the *Tool Description* page.
   Click *Next*.

6. On the *Inputs and Outputs* page you will find preconfigured static in- and outputs, that will match the old tool wrapper conventions. If your tool needs additional in- or outputs, feel free to configure.
   Click *Next*.

7. Skip the page *Tool Properties* by clicking *Next* since it is not relevant for tools that match the conventions of the old CPACS tool wrapper.

8. Add a launch setting for the tool by clicking the *Add* button on the *Launch Settings* page. Configure the path of the CPACS tool and fill in a version, click *OK*. If you would like to allow users of your tool to choose that the temp directory won't be deleted at all after workflow execution, check the property *Never delete working directory(ies)*. Not to delete the working directory can be very useful for users for debugging purposes, at least if they have access to the server's file system. But this option can result in disc space issues as the amount required grows continuously with each workflow execution. It is recommended to check that option during integrating the tool and uncheck it before publishing the tool.
   Click *Next*.

9. The *CPACS Tool Properties* are preconfigured to match the folder structure defined for the old CPACS tool wrapper. In most cases you do not have to change this configuration. If you are using XSLT mapping, please select the corresponding mapping files. If your tool does not work with static tool specific input, please deselect this property.
   Click *Next*.

10. In the *Execution command(s)* tab on the *Execution* page, you need to define your execution command itself as well as optional pre and post commands. Commands will be processed sequentially line by line. An example for a typical Windows command including pre and post commands will look like the following:

```
rem pre-command
pre.bat

rem tool-execution
YourTool.exe ToolInput/toolInput.xml ToolOutput/toolOutput.xml

rem post-command
post.bat
```

11. Click *Save and activate* and your tool will appear immediately in the palette and is be ready to use.

12. If not already done, do not forget to publish your tool (cf. SectionSection 3.5, "Tool publishing and authorization") after testing it locally. To check if your tool is successfully published to the RCE network open the tab *Network View* at the bottom and checkout *Published Components* after expanding the entry of your RCE instance.

3.4.4.3. Integrate a CPACS Tool into a Server Instance in Headless Mode

The way to integrate a CPACS tool on a server running RCE in headless mode is as follows: Perform the steps to integrate a CPACS tool on a client instance and make sure that the path of the CPACS tool configured on the *Launch Settings* page (step 8) matches the absolute tool path on your server system. Afterwards, you will find the configuration files inside your rce profile folder at the following location:

```
/integration/tools/cpacs/[YourToolName]
```

Copy the folder `[YourToolName]` to the same location inside the profile folder running with your headless server instance. Use the "auth" commands (cf. SectionSection 3.5, “Tool publishing and authorization”) to publish your tool. If the server instance is already running, your tool will be available immediately after publishing.

3.5. Tool publishing and authorization

RCE components and integrated tools can be published to make them usable by other connected ("remote") RCE instances. The publishing options for each component/tool can be defined in the "Component Publishing" view. In this view, each component can be assigned to one of three basic publication levels:

- Local (the default option): Components with the "local" setting can only be used on the local instance; they are not visible to other instances.
- Custom: This setting allows to make the component/tool available only to specific groups of users. To use this setting, one or more authorization groups have to be created first, which is explained in the next section. Each component/tool can then be assigned to one or multiple groups. Users on remote instances can see and use components if they are members of at least one of these groups.
- Public: Components with the "public" setting can be used by all connected RCE instances. This is equivalent to the tool and component publishing in earlier versions of RCE. Tools in the "public" group are also available over SSH connections.

> **Note**
>
> If the "Component Publishing" view is not visible, you can open it from the "Window > Show View" menu. If it is not listed there, choose "Other" and select them from the "RCE" category.

3.5.1. Managing authorization groups

Authorization groups can be created and managed in the "Authorization Groups" dialog, which can be opened from the "Component Publishing" view. To create a new group, click the "Create Group"-button and enter a name for the group. To provide access to this group to other users, select the group in the list and click "Export Group Key". Copy the provided key from the dialog that appears, and pass it on to the users that you would like to invite to this group.

> **Note**
>
> IMPORTANT: This exported group key is similar to a password. When passing it to other users, make sure to use a communication medium that unauthorized users cannot easily intercept. For example, passing the key via an encrypted chat system provided by your employer, or a Team Site that is only accessible to project members, is usually secure enough. On the other hand, sharing it by email outside of your organization is usually unsafe, and we recommend using more secure alternatives.

When the other user receives this key, they can import it into their RCE instance by using the "Import Group Key" button in their "Authorization Groups" dialog. After importing a key on an RCE instance,

all tools published for that group on connected RCE instances are visible and can be used like a "public" component.

3.5.2. Publishing tools on the command console

Creating custom tool groups and publishing tools is also possible using the "auth" commands on the command line. A short reference:

- `auth create <name>` - creates an authorization group
- `auth list` - lists available access groups
- `auth delete <name/id>` - deletes an authorization group; if the name is ambiguous (e.g. there are two groups named "groupName"), you need to add the randomly generated id behind it, separated with a colon (e.g. `groupName:2716ab2d25`)
- `auth export <name/id>` - exports a group key in a form that can be imported by another instance via GUI or command line
- `auth import <exported key>` - imports a group key exported via GUI (as described above) or via the `auth export` command. The group name is embedded in the exported key, and is set automatically.
- `components set-auth <component id> <permissions>` - sets the permissions for a component. Possible values for "permissions" are either "`local`", "`public`", or a comma-separated list of authorization groups/ids.
- `components list-auth` - shows a list of all defined authorization settings. These settings are independent of whether a matching component exists, which means that settings are kept when a component is removed and later added again.

The component ids used in this commands can be derived as follows:

- `rce/<component name>` for standard RCE components, e.g. "`rce/Parametric Study`"
- `common/<tool name>` for integrated tools of type "common" e.g. "`common/ExampleTool`"
- `cpacs/<tool name>` for integrated tools of type "CPACS" e.g. "`cpacs/CPACSExampleTool`"

3.6. Remote Tool and Workflow Access

RCE provides an interface that allows external applications to access and run single tools or complete workflows within RCE instances. This allows existing applications to make use of RCE's features (like network distribution, data management, logging, ...) when integrating the application itself is not possible or desirable. For example, applications that are based on frequent user interaction are not a good fit for being run as part of an automated workflow, but may still want to use some of RCE's features.

This section describes how to publish tools or workflows such that they can be used via Remote Access. It will guide you through the creation of a simple example, which you can expand to build your own solutions.

3.6.1. Basic Concepts

The basic idea of the Remote Access interface is that the external application opens an SSH connection to RCE, and initiates the tool or workflow execution via text commands. Input and output data, as well as log files, are transfered using SCP. For using the integrated tools and workflows remotely, a standalone remote access tool is currently under development and will be released soon. The integrated tools can also be used in a remote RCE instance via SSH connections.

3.6.2. Tool vs. Workflow Execution

In RCE 5.0.0, the "remote access" feature was able to invoke a single integrated tool. Starting with RCE 5.1.0, arbitrary user-defined workflows can be executed. The following sections describe each approach separately.

3.6.3. Setting up the Single Tool Execution Example

These steps will guide you through the configuration of an integrated tool. Using this example, you will be able to easily integrate your own text-based tools into RCE and invoke them using the Remote Access interface.

- Setting RCE up as a "Remote Access Server".
  - Download and unpack/install an RCE distribution.
  - Start RCE and select the "Help > Configuration Information" menu option. From the list of configuration examples, double-click "Remote Access Server" to open it. Mark all text (Ctrl-A) and copy it to the clipboard (Ctrl-C).
  - Select the "Help > Open Configuration File" menu option, select all text (Ctrl-A) and paste the copied text from the example configuration.
  - Save the file, and select "File > Restart" to apply the new configuration. The Remote Access SSH interface is now running, with a pre-configured SSH account of "ra_demo", password "ra_demo".

    Note

    IMPORTANT: "ra_demo" is just an example account for testing within a secure network. Create a new account with a better password (for example using the configuration UI described in the "Configuration" section) before using the Remote Access interface in production. Publishing tools using this default account is a potential security risk.

- Define an example tool using the following steps.
  - Select the "File > Integrate Tool" menu option.
  - Follow the tool integration wizard to integrate your tool (more information can be found using the help function in RCE and in the chapter "Integration of External Tools" of this user guide.
  - Publish your tool in the "public" group (cf. Section 3.5, “Tool publishing and authorization”) to make the tool available via Remote Access.
- You now have an example tool that can be accessed with the "Remote Access" feature. How to setup a client to access this tool see the "Configuring an RCE instance as an SSH client" section
- To get an impression of how this feature interacts with existing RCE features, you can examine several areas within the RCE instance.

- Open the "Workflow List" view in RCE and watch it while the "run-tool" script is executing. After a short preparation time where the input data is uploaded, you will see the automatically generated workflow containing the tool being executed. It will disappear automatically if it finishes successfully; if it fails, it will remain in the list for review.

- Open the "Workflow Console" view; if the tool produced any output, it should be visible there.

- Open the "Workflow Data Browser" and refresh it; there should be a "Remote_Tool_Access-..." workflowe entry matching the remote tool run. When you expand this entry, you should see the uploaded content of the input folder, the generated output folder, any generated text output (in the "Execution Log" folder), and the exit code of the tool process (also in the "Execution Log" folder). It will disappear automatically if the tool finishes successfully

3.6.4. Setting up the Workflow Execution Example/ Template

These steps will guide you through the creation of a remote-executable workflow, and will show you how to invoke it using the provided example scripts.

- Download and unpack/install an RCE distribution.

- Run RCE and select the "Help > Open Configuration Information" menu option. From the list of configuration examples, double-click "Remote Access Server" to open it. Mark all text (Ctrl-A) and copy it to the clipboard (Ctrl-C).

- Select the "Help > Open Configuration File" menu option, select all text (Ctrl-A) and paste the copied text from the example configuration.

- Save the file, and select "File > Restart" to apply the new configuration. The Remote Access SSH interface is now running, with a pre-configured SSH account of "ra_demo", password "ra_demo".

  Note

  IMPORTANT: "ra_demo" is just an example account for testing within a secure network. Create a new account with a better password (for example using the configuration UI described in the "Configuration" section) before using the Remote Access interface in production. Publishing tools using this default account is a potential security risk.

- As a first example we are going to execute the umodified "Remote_Workflow_Access_Template" workflow file in the Workflow Examples Project. If you haven't created this project already, right-click in the Project Explorer on the left side, and choose "New > Workflow Examples Project", and choose a name for it. The template file is contained within the project folder. To get an impression of the basic setup, open the template workflow file. You will see an SCP Input Loader on the left side with two outputs. On the right side, there is an SCP Output Collector with one input (these two are helper components that are only used for remote access workflows). These are the points where the Remote Workflow Access feature sends the provided inputs into your workflow, and collects the final outputs.

- As a security measure, you need to explicitly publish your workflow to allow remote access to it. This is done via a console command at this time; future RCE versions will most likely add a option to do this from the GUI. To issue this command, open the "Command Console" view (if it is not already visible) by selecting "Windows > Show View > Other" from the menu, and then double-clicking "Command Console" in the "RCE" section.

- Right-click your workflow file in the "Project Explorer" and select the "Copy full path" entry in the popup menu to copy the full path to the workflow file to the clipboard.

> **Note**
>
> This step demonstrates how the get the path of a workflow file in the current workspace, but you can use workflow files that are located anywhere on your system.

- To make the workflow available for remote execution, enter the command `ra-admin publish-wf "<workflow file>" <id>` in the command window. Press Ctrl-V in place of *<workflow file>* to insert the path to your workflow file there. For *<id>*, choose a string (without whitespace) that callers can use to execute the workflow. Press "enter" to execute the command. The workflow file will be inspected, and you will either see a message describing what is missing, or a message that the workflow was successfully published. Fix any errors until the workflow is published.

  > **Note**
  >
  > Starting with RCE 6.2.0, published workflows are persistent by default, so they will still be available after the local RCE instance is restarted. Use the `ra-admin unpublish-wf <id>` command to remove a published workflow from remote access.
  >
  > To publish a workflow for the current RCE instance's life-time only, use the `-t` option: `ra-admin publish-wf -t "<workflow file>" <id>`.

- If some of the workflow's components use placeholders for configuration values, you can use the `-p` option to specify a placeholder values file. The structure of placeholder value files is explained in Section 3.3.3.1, “Configuration Placeholder Value Files”. Placeholder files can used with both persistent and non-persistent workflows (see above).

  Example: `ra-admin publish-wf -p myPlaceholderValues.json myWorkflowFile.wf myPublishId`

- You now have a workflow file that can be executed using the "Remote Access" feature.

- To get an impression of how this feature interacts with existing RCE features, you can examine several areas within the RCE instance.

  - Open the "Workflow List" view in RCE and watch it while the "run-wf" script is executing. After a short preparation time where the input data is uploaded, you will see the workflow being executed. It will disappear automatically if it finishes successfully; if it fails, it will remain in the list for review. You can also double-click on a running or workflow to monitor its execution.

  - Open the "Workflow Console" view; if the tool produced any output, it should be visible there.

  - Open the "Workflow Data Browser" and refresh it; there should be an entry for the Remote Access workflow. When you expand this entry, you should see the uploaded content of the input folder, the generated output folder, any generated text output (in the "Execution Log" folder), and the exit code of the tool process (also in the "Execution Log" folder).

3.6.5. Building Your Own Remote Access Workflow

After running the example/template workflow as described in the previous section, you can proceed to building your own actual workflow.

As described above, open the "Remote_Workflow_Access_Template" workflow file. You will see an SCP Input Loader on the left side with two outputs. On the right side, there is an SCP Output Collector with one input (these two are helper components that are only used for remote access workflows). These are the points where the Remote Workflow Access feature sends the provided inputs into your workflow, and collects the final outputs. You can change the data types or add/delete inputs/outputs in the properties view of the input loader/output collector. The Script component in the middle is just a placeholder - unless you need a Script component anyway, you can just delete it.

There are two basic approaches to building your workflow:

- Either you start with the template, and build your workflow between the two standard components. This is straight-forward, but means that you cannot test run the workflow from the RCE GUI (as the Input Loader will fail), but have to use the Remote Access feature to test it.

- The other approach is to build your workflow normally, where you add an SCP Input Loader and SCP Output Collector with the outputs and inputs you need. You can then test (and if needed, modify) your workflow from the GUI until it behaves as it should. Then, mark all components *except* the Input Provider and Output writer in your workflow, and select "Copy" from the right-click menu. Switch to the template file, click an empty area, and select "Paste" from the right-click menu. Then, connect the two template components (Input Provider and Output Writer) as in your original workflow.

  Note

  (Advanced Usage) You can also build your workflow in the template file, add your own Input Provider and Output Writer, and use the new "Enable/Disable Component" feature to toggle between them for testing and Remote Access usage. As this requires some helper components to work, this is not recommended for your first example, but may be a useful trick to keep in mind.

After you have finished building your workflow, the process of publishing and executing it is the same as described above for the unmodified template file. Please note that publishing your workflow for remote execution automatically creates an (invisible) copy of it. Modifications you make to your workflow file are not published right away. Once you have made the changes you want to publish, run the same "ra-admin publish-wf" command again to update the published version.

Note

Tip: To repeat a previous command, presse the "up arrow" key in the Command Console window.

3.7. Connecting RCE instances via the RCE network or via SSH connections

RCE provides two possibilities to connect your RCE instance to other RCE instances and to use the user-integrated tools and components published on those instances: The RCE network connections and SSH connections. RCE connections are meant to be used only in a trusted network (e.g. your institution's internal network). The RCE network traffic is currently *not encrypted*. This means that it is *not secure* to expose RCE server ports to untrusted networks like the internet. In the case that it is not possible or not secure to use RCE connections, SSH connections provide a more secure alternative. The following table compares the two connection types:

Table 3.13. Connection types

Connection type	Access to all published tools	Supporting all RCE network functions (e.g. remote monitoring)	Encrypted traffic	Using login name and password or RSA key	Symmetrical
RCE connections	**yes**	**yes**	**no**	**no**	**yes**
SSH connections	**yes**	**no**	**yes**	**yes**	**no**

3.7.1. RCE Network Connections

RCE connections are meant to be used only in a trusted network (e.g. your institution's internal network). To build up a network of RCE instances, at least one of the instances has to be configured as a server (see the "Configuration" section or the sample configuration file "Relay server" for details).

On the client side, RCE network connections can be added in the "network" view by clicking "Add network connection" and entering the hostname and port of an RCE server instance. The connections are shown in the "RCE Network"->"Connections" subtree. They can also be edited, connected and disconnected in the network view. However, the changes made here are not saved in the configuration yet, i.e. they will be lost when RCE is closed or restarted. To permanently add connections, you can edit the configuration file (see the "Configuration" section for details).

In the "RCE Network" -> "Instances" subtree all RCE instances in the network are listed. When expanding the entry for an instance, you can see monitoring data like CPU or RAM usage for this instance, and the published components and tools of this instance (if it has any).

The published components and tools of the other instances in your network are also shown in the palette of the Workflow Editor. From there, you can use them in your workflows just like your local components and tools. When you start a workflow, in the "Execute Workflow" wizard there is an overview which component will be run on which RCE instance. If a component is available on several instances, you can choose here on which instance it should be run. In the same wizard, you can also choose another instance as the "Controller Target Instance", which means that the workflow execution will be controlled by this instance (see the section "Configuration Parameters" for more information). This can be useful when you start a long-running workflow where all components are run on remote instances and you do not want to keep your local computer connected all the time.

3.7.2. SSH Remote Access Connections

SSH connections provide a more secure alternative to the standard RCE connections and can be used to access tools remotely. The published tools are shown in the palette of the client's Workflow Editor (this may take a few seconds after connecting, as the tool list is fetched from the remote hosts every few seconds). From there, you can use them in your workflows just like your local components and tools. Differently from tools accessed by RCE network connections, in this case the component is shown to be executed on your local instance in the Workflow Execution wizard.

Also workflows that were published on the remote instance (for information about the publishing see section "Remote Tool and Workflow Access") are shown as components in the palette of the client's Workflow Editor in the group "SSH Remote Access Workflows" (if the client runs RCE 7.1 or newer). These remote workflows can be added to workflows and executed like local components/tools.

3.7.2.1. Configuring an RCE instance as an SSH server

The RCE instance that publishes the tool, or another instance connected to it by RCE network connections, has to be configured as an RCE remote access server (see the "Configuration" section or the sample configuration file "Remote access server" for details).

Note

When configuring an SSH account using a key file, note that for using key files, both server and client have to run RCE 7.1 or newer and that only RSA-Keys in the OpenSSH format are supported by RCE. You can generate such keys on Linux e.g. using the "ssh-keygen" tool (which creates the correct format by default). On Windows, you

can use the software "puttygen" to generate a key (choose the type "SSH-2 RSA") and convert it to the OpenSSH format ("Conversions"->"Export OpenSSH key" in puttygen).

3.7.2.2. Configuring an RCE instance as an SSH client

On the client side, SSH connections can be added in the "network" view by clicking "Add SSH Remote Access Connection". In the following dialog, enter the hostname and port of an RCE instance that provides an SSH server as well as the user name and the authentication data of an SSH account configured on this instance. Depending on the SSH account, you have to authenticate using a passphrase or by an RSA private key file. If your private key is protected by a passphrase, select the authentication type "Keyfile with passphrase protection", else select "Keyfile without passphrase protection".

The connections are shown in the "SSH Remote Access"->"SSH Remote Access Connections" subtree. They can also be edited, connected and disconnected in the "network" view. It is possible to store passphrases using the Eclipse Secure Storage Mechanism (this is only available on Windows). However, the changes made here are not saved in the configuration yet, i.e. they will be lost when RCE is closed or restarted. To permanently add SSH connections, you can edit the configuration file (see the "Configuration" section for details). However, it is not possible yet to store the passphrases for such connections permanently.

3.7.3. Example Structure of an RCE network with several project partners

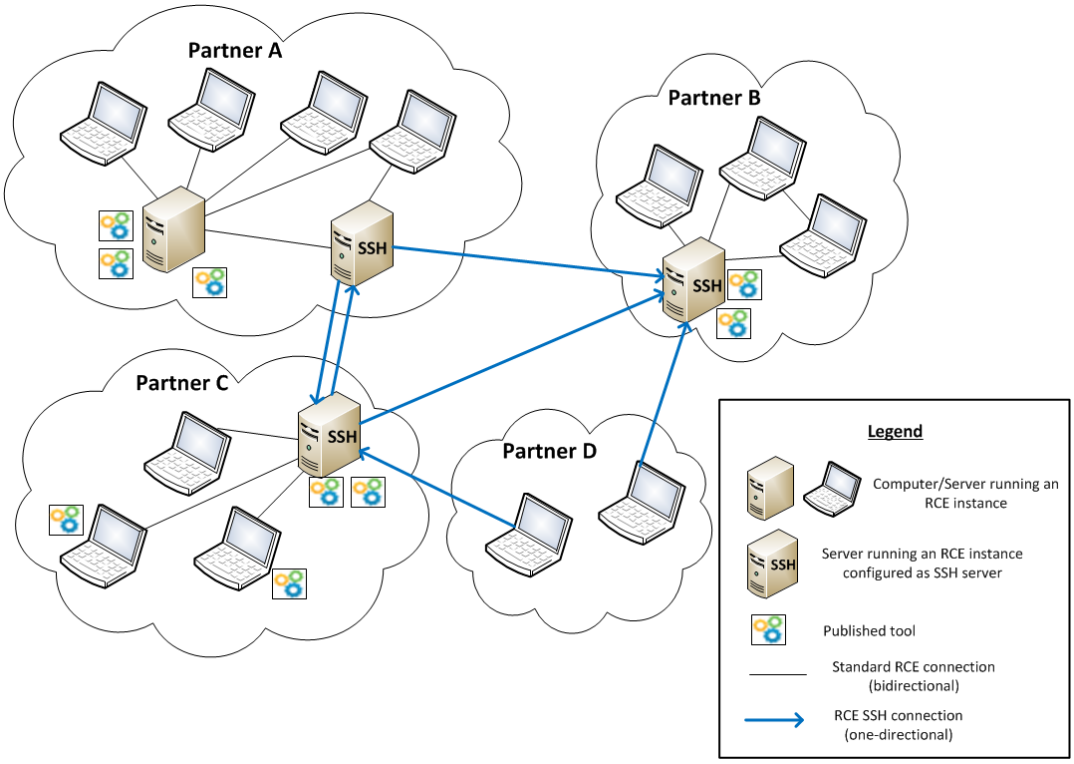
This section describes how an RCE network with several project partners could be structured. RCE provides two different types of connections between RCE instances which can both be used to access published tools:

Standard RCE connections are meant to be used in a trusted (internal) network, as the data exchanged via those connections is not encrypted. These connections are bidirectional and allow to share more data than just the information about published tools (e.g. they can be used to access data about executed workflows on another instance).

RCE SSH connections use the SSH protocol to provide an encrypted connection. They are meant to securely connect RCE instances from different institutions over the internet. These connections are one-directional and allow the client to run published tools and workflows on the server (of course, both instances can be configured as client AND server at the same time if necessary so they can use each other’s tools).

In practice, most partners will probably use both types of connections: RCE connections to connect different RCE instances within their institution and SSH connections to connect to the other partners. The following figure gives an example of how such a project network could be structured:

Figure 3.8. Example RCE network



The four project partners in the example all have an internal network of RCE instances which are connected by standard RCE connections. SSH connections are used to connect between the different partners. Therefore, each partner who wants to publish tools to other partners has one RCE instance which is configured as an SSH server. Only this one instance per institution has to be reachable via SSH over the internet, all other instances in the institution's internal network can be connected to it by standard RCE connections and still publish tools to the other partners/access tools published by other partners.

In the example, the RCE instances in Partner A's network can access the tools published in Partner C's network and vice versa. One of the instances from Partner D can access Partner C's tools; and all the partners can access the tools in Partner B's network.

Each institution in the example has a different internal setup, all of which are possible:

Partner A has a dedicated RCE server where the published tools are located, which is connected to the SSH server by an RCE connection. All other RCE users in the internal network are connected to this server.

Partner B has put all the tools directly on the SSH server.

In Partner C's network, some tools are located on the SSH server, but some tools are also published by users directly on their own machines. As long as they are connected to the SSH server, also those tools can be used by the other partners.

Partner D has not published any tools and has no SSH server, instead the users' computers connect directly to the other partner's SSH servers.

Appendix A. Script API Reference

This section contains a reference for the API that is accessible via the script component.

Method	Description
`def RCE.close_all_outputs ( )`	Closes all outputs that are known in RCE.
`def RCE.close_output ( name )`	Closes the RCE output with the given name
`def RCE.fail ( reason )`	Fails the RCE component with the given reason
`def RCE.get_execution_count ( )`	Returns the current execution count of the RCE component
`def RCE.get_input_names_with_datum ( )`	Returns all input names that have got a data value from RCE.
`def RCE.get_output_names ( )`	Returns the read names of all outputs from RCE
`def RCE.get_state_dict ( )`	Returns the current state dictionary
`def RCE.getallinputs ( )`	Gets a dictionary with all inputs from RCE
`def RCE.read_input ( name )`	Gets the value for the given input name or an error, if the input is not there (e.g. not required and it got no value)
`def RCE.read_input ( name , defaultvalue )`	Gets the value for the given input name or returns the default value if there is no input connected and the input not required
`def RCE.read_state_variable ( name )`	Reads the given state variables value, if it exists, else None is returned
`def RCE.read_state_variable ( name, defaultvalue )`	Reads the given state variables value, if it exists, else a the default value is returned and stored in the dictionary
`def RCE.write_not_a_value_output ( name )`	Sets the given output to "not a value" data type
`def RCE.write_output ( name, value )`	Sets the given value to the output "name" which will be read from RCE
`def RCE.write_state_variable ( name, value )`	Writes a variable name in the dictionary for the components state