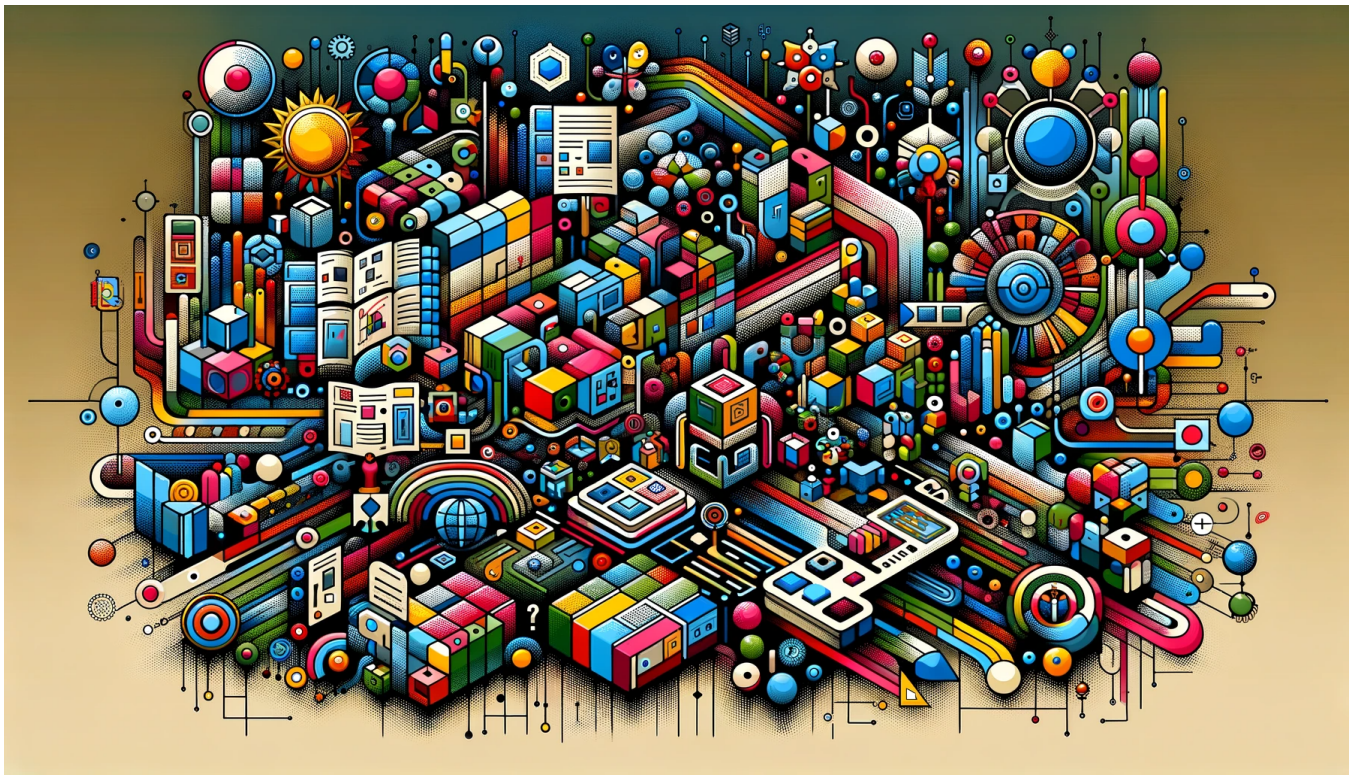


IRTlib Documentation: Software for the administration and delivery of computer-based assessments

Ulf Kroehne

Laste Change: 30th December, 2023

Suggested citation:

Kroehne, U. (2023). IRTlib Documentation: Software for the administration and delivery of computer-based assessments [IRTlib Dokumentation: Software für die Verwaltung und Auslieferung computergestützter Assessments]. DIPF, Frankfurt am Main, Germany. https://doi.org/10.5281/zenodo.10441352

Table of contents

1 IRTlib Software

IRTlib is a software for the delivery of computer-based tests. The software consists of two components:

- *IRTLib Editor*: A software for *test authors*, which is used to configure *Studies*.
- *IRTlib Player*: A software for *data collections*, with which *target persons* work on tasks that are configured in the form of a *Study*.

Instructions for installing and setting up both programmes for initial use can be found under Download & Installation.

Before using the *IRTlib Software* to configure and create deliveries, the assessment content (tasks, instructions, intermediate screens, etc.) must be created in the form of individual *Tasks* using the *CBA ItemBuilder*.

- The *CBA ItemBuilder* can be downloaded here: www.itembuilder.de/software
- An interactive documentation of the *CBA ItemBuilder* is available here: cba.itembuilder.de

Suggested Citation:

> Kroehne, U. (2023). Open Computer-based Assessment with the CBA ItemBuilder. DIPF, Frankfurt am Main, Germany. https://doi.org/10.5281/zenodo.10359757

The development of the *CBA ItemBuilder* and the *IRTlib Software* is coordinated by the *Centre for Technology-Based Assessment (TBA)* at the *DIPF | Leibniz Institute for Research and Information in Education*.

2 Download & Installation

The *IRTlib* software is provided for offline use (currently for Windows operating systems) and for online use (in the form of *Docker containers*).

2.1 Offline (Windows)

The *IRTlib* software (*IRTlib Editor* and *IRTlib Player*) for offline use can be obtained and downloaded from the [Releases] section of the repository https://github.com/DIPFtba/IRTlibDeploymentSoftware. Two ZIP archives are available for download in the Releases section.

`TestApp.Editor.Desktop.exe`: Is in `TestApp.Editor.Desktop.zip` and must be started to use the *IRTlib Editor*.

`TestApp.Player.Desktop.exe`: Is included in `TestApp.Player.Desktop.zip` and must be started to use the *IRTlib Player*.

> **Note: Versions from the current development as *Preview*.**
>
> Note that the latest build can be found in the Preview section of the *Release* section of the repository. Preview versions are the latest version of the software, after the last published version of the *IRTlib Software*. To obtain reproducible results, published versions should always be used.

> **Note: Warning message at programme start**
>
> The automatically created preview versions of the *IRTlib Editor* and *IRTlib Player* are not signed. A warning message from the operating system must be accepted before the programmes can be executed. Depending on the configuration of the operating system, the executable files of the *IRTlib software* can also be classified as unknown and additionally warned against their use.

2.1.1 Study Preparation with Offline Editor

The *IRTlib Editor* for offline use is provided as a ZIP archive (e.g. `TestApp.Editor.Desktop.zip`), which must be unpacked. After unpacking the editor, the application `TestApp.Editor.Desktop.exe` can be started on a Windows device.

The sections Preparation > Overview, Preparation > Studies and Preparation > Survey parts document how to prepare and configure data surveys with the help of *CBA ItemBuilder* items.

2.1.2 Study Execution with Offline Player

The *IRTlib Player* is also available as a Windows application for offline use and is provided as a ZIP archive (e.g. `TestApp.Player.Desktop.zip`). After unpacking the *IRTlib Player*, a published study configuration is required that is to be used for data collection.

After adding the contents of a published study provided as study configuration, the executable file `TestApp.Player.Desktop.exe` can be started (either with or without start parameters).

- **Kiosk Mode**: The *IRTlib Player* can be used directly for data collection via the executable file `TestApp.Player.Desktop.exe` on the computer on which it is running locally. The *Study* can be configured so that it is displayed in a *Kiosk Mode* on one screen and can only be terminated via the *Task Manager* or the *Test Manager Menu* (see *Full Screen Mode* in the section Configuration for display).
- **Local Server**: The *IRTlib Player* can also be run as a local server. After starting the programme `TestApp.Player.Server.exe`, a configured *Study* can also be delivered via *Webbrowser* or other browsers with *Kiosk Mode* (e.g. the Safe Exam Browser). With this configuration, data can be collected, for example, in schools without an internet connection but with a notebook acting as a *bring-in* server.

The sections Data collection > Overview, Data collection > Publish & export and Data collection > Integration & delivery document how data collection can be carried out using the *IRTlib Player* in the various constellations.

2.2 Online (Docker)

The *IRTlib* software (*IRTlib Editor* and *IRTlib Player*) for online use can be obtained as a *Docker* container. An example can be found at https://github.com/DIPFtba/IRTlibDeploymentSoftware.

To use the Docker container, it is recommended to *clone* the repository on the target device and execute the command `./start.sh` in the `docker` folder (requires installed `docker` and `docker compose`) to start the software.

If nothing is changed in the `docker-compose.yml` file, the editor is accessible via port *8002* and the player software via port *8001*.

The section Data collection > Integration & delivery contains further information on using the *Docker* containers.

Part I

Vorbereitung / Preparation

3 Vorbereitung: Übersicht / Preparation: Overview

The preparation of a computer-based assessment based on *CBA ItemBuilder* content begins with the use of the *IRTlib Editor* to create a study configuration. This usually involves the following steps:

> Optional: Using a *Runtime* for *CBA ItemBuilder* before version 9.9?
>
> - **Requirement**: Check the availability of the Runtime. The *IRTlib Editor* can be used to prepare assessments with content created using the CBA ItemBuilder. To use CBA ItemBuilder *Tasks* stored in *project files*, a *runtime* (i.e. the files `main.*.js` and `main.*.css`) is required in the version that corresponds exactly to the version of the CBA ItemBuilder used to create the items (e.g. `9.9.0`). Before using the *IRTLib-Editor*, make sure that the required *Runtime* is included or import the runtime files (see section Settings for details).
>
> Note: When using CBA ItemBuilder items from version 9.9, this step is generally not necessary.

- **Creating a new Study**: The *IRTlib Editor* is used to configure so-called *Studies*. The versions of studies can be tracked in the editor, studies can be published there (i.e. *sealed* for data collections). To start creating content with the *IRTlib Editor*, a study must first be created (see section Studies for, details).

> Note: Creating a *Study* is always necessary.
>
> Note that at least one *Study* must be defined in the *IRTlib Editor* before a study configuration can be used for data collection with an *IRTlib Player*.

- **Define Basic Configurations for Study (Info)**: Basic configurations related to the content of a prepared study include the study name and description, login mode, display configuration, test administrator menu, and how to proceed after completing all content defined in a study (see Studies section for more details).
- **Creating a new Survey Part**: Each *Study* consists of one or more *Survey Parts*. *Survey Parts* are considered to be building blocks of assessments that are administered together, such as items from a particular domain. *Survey Parts* of type *CBA ItemBuilder* can be used to administer *CBA ItemBuilder* tasks in a linear sequence or with *Blockly*-based routing.

> Note: Creating a *Survey* Part is always necessary.
>
> Note that each *Study* requires at least one *Survey Part* defined in the *IRTlib Editor* before a *Study* configuration can be used for data collection with an *IRTlib Player*.

- **Configure basic settings for survey part (Info)**: A *Survey Part* of type *CBA ItemBuilder* is based on a set of *CBA ItemBuilder-Tasks*. Each *CBA ItemBuilder-project file* requires at least one *task*, but projects with multiple *tasks* are also supported. If *CBA ItemBuilder* content with a common time limit is to be administered across *tasks*, *assessment sections* allow the assignment of tasks to a structure that distinguishes assessment content that is administered *before* a time-limited section (e.g. instructions, in the *task* section). e.g. instructions, in the section *preparation-parts*), content that is administered *after* a time-limited section (e.g. acknowledgements, in the section *post-parts*) and tasks with limited time in between (*items*, see section [preparation-parts.qmd]).

- **Add Items**: To finalise the definition of a *Survey Part*, the *CBA ItemBuilder project files* must be imported into the *Items* section. By default, it is assumed that the order of the *CBA ItemBuilder-Tasks* is linear. However, if *Routing* is enabled for a study section, the *Blockly*-based sequence definition can be used to implement different test designs (e.g. multiple booklets, multi-stage tests, etc.).

3.1 Embedded Programme Help

For the use of the *IRTlib Editor*, a programme help is integrated directly into the application, which can be displayed via the small ? symbol in the top right-hand corner.

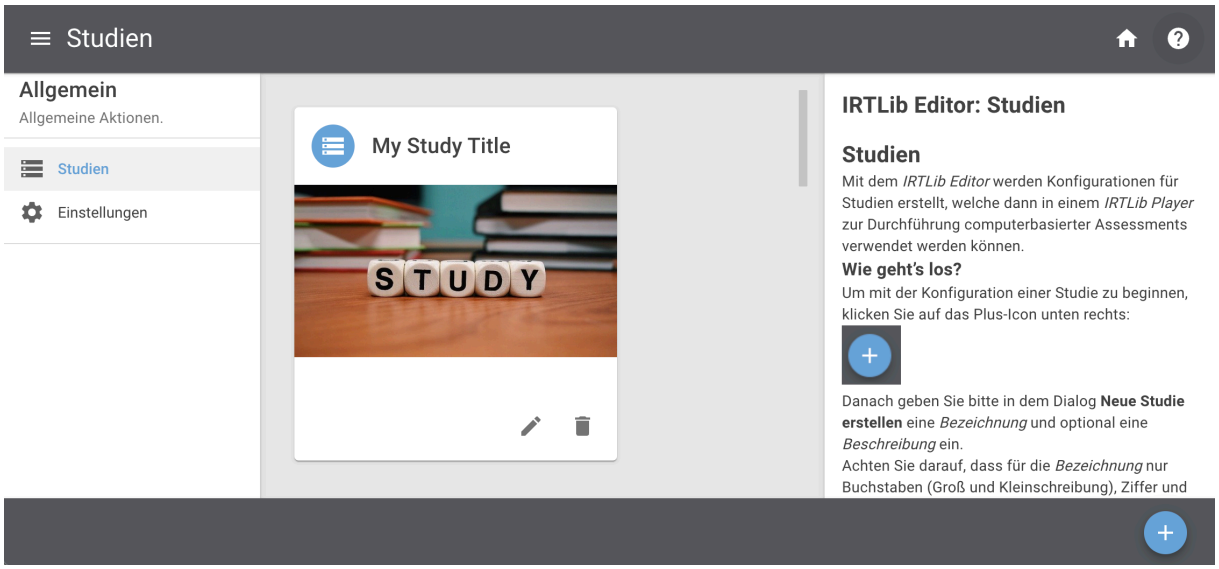


Figure 3.1: Example of the *IRTlib Editor* with embedded help

> **Embedded Programm Help**
>
> The contents of these help pages from the *IRTlib Editor* are integrated into this *IRTlib Documentation* and are always displayed in this frame with the title *Embedded Programme Help*.

3.2 Preparation of CBA ItemBuilder Content

The *IRTlib Software* is required to display the assessment content created with the *CBA ItemBuilder* and to use *CBA ItemBuilder* content for data collections. The *project files* (ZIP archives) that can be created with the *CBA ItemBuilder* must be available for this.

3.2.1 Entry Point (Task)

Each *CBA ItemBuilder project file* must define at least one *task*. Only *tasks* can be used in the *IRTlib software*. It is easy to check that a *task* is fully defined in the *Preview* of the *CBA ItemBuilder*:

> **The *IRTlib software* requires defined *tasks*.**
>
> Assessment content can be assembled from individual *Tasks* using the *IRTlib Software*. The sequence of *Tasks* can be defined statically as a *linear sequence* or as a programmed sequence in *Blockly*. No individual pages within *CBA ItemBuilder-Tasks* can be controlled from the *IRTlib Soft-*

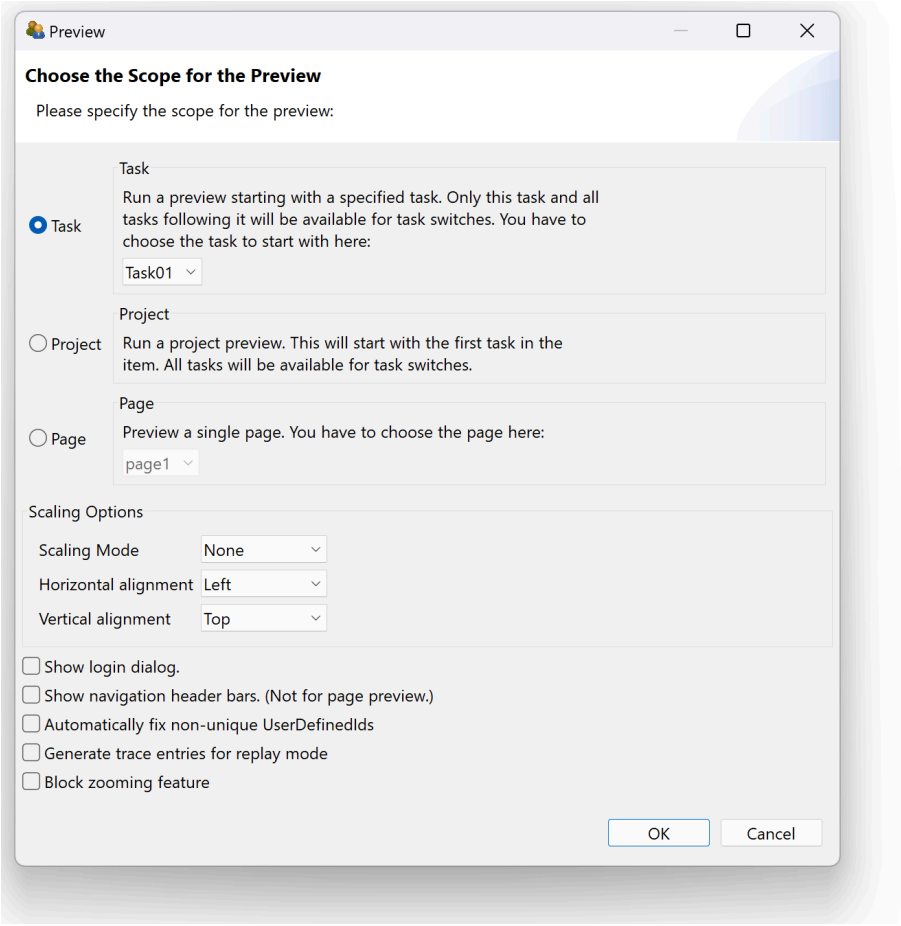


Figure 3.2: Example from *CBA ItemBuilder* to request a *Preview* for a *Task*

> *ware.*

CBA ItemBuilder-Project files which can only be displayed via the *Project* or *Page* option in the *Preview* cannot be used in the *IRTlib Software*.

3.2.2 Display Bhaviour (Scaling Options)

The *Preview* of the *CBA ItemBuilder* can also be used to check whether the assessment contents are displayed in the desired scaling, which can be set under *Scaling Options*, according to the requirements.

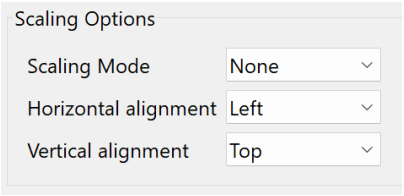


Figure 3.3: Settings for *Scaling Options* from *CBA ItemBuilder-Preview*

Settings similar to *Preview* can be defined in the *IRTlib Editor* for the display settings of a *Study* (see section Studies).

3.2.3 Definition of the Scoring (Results Data)

The *IRTlib software* is designed to collect data with the help of *CBA ItemBuilder-Task*. What result variables are saved from the processing of a *task* can be defined by *item authors* in the *scoring* definition of

a *task*.

> The *scoring* must already be defined in the *CBA ItemBuilder*.
>
> The result variables defined as *Classes* are saved from the processing of *Tasks*, the values of which can either be individual *Hits* or the transfer of information using the so-called *ResultText* operator in the *CBA ItemBuilder*.

Using the built-in *Scoring Debug Window*, the *scoring* of individual *CBA ItemBuider tasks* should already be checked in the *CBA ItemBuilder* before the assessment contents are combined into *studies* with one or more *survey parts* using the *IRTlib software*. Further information on suggested checks is formulated in the section Deliveries Testing and Release.

> Log data is collected automatically.
>
> Without further configuration, log data is automatically recorded in the assessment content created with the *CBA ItemBuilder* and collected via the *IRTlib software*.

3.2.4 Integrated Multimedia Content (Resources)

The assessment content created with the *CBA ItemBuilder* can contain multimedia content (images, videos, audio files). Images and videos are displayed in a size that is used in the corresponding component of the *CBA ItemBuilder* in the *Page Editor*. Images, videos and audio files are saved as *resources* in the *project files* as soon as they have been inserted via the *Resource Browser*. Unused *resources* remain in the *project files*.

> File size of *CBA ItemBuilder project files* should be as small as possible
>
> The file size of *CBA ItemBuilder project files* is particularly relevant for use in online deliveries and should be kept as small as possible.

Before using *CBA ItemBuilder-project files*, it is recommended to consider the following points:

- Images and videos only in the required size: Images and videos can be reduced to the size (width and height) in which they are actually used in *CBA ItemBuilder-Projects* without any loss of quality.
- Compress images if possible: Without changing the image size, images can often be further reduced in file size.
- Compress videos if possible: Without changing the video, videos can often be further reduced in file size.
- Audio not in the highest quality: If not necessary, audio files should be reduced in quality so that they still sound acceptable but are optimised in terms of transmission times.
- Remove unused resources: The *CBA ItemBuilder* provides a button in the *Resource Browser* to automatically remove unused *resources*. This function should be used at the end so that the *project files* do not contain any unnecessary *resources*.

Careful handling of *resources* and optimisation of the file size of *CBA ItemBuilder project files* can be decisive for a smooth and trouble-free assessment, which can be delivered online with the *IRTlib software* and used, for example, via mobile devices with limited internet bandwidth.

4 Vorbereitung Studien / Preparation Studies

Configurations that are created with the *IRTlib editor* are summarized in so-called *studies*. A *study* is intended to summarize the assessment content that is administered in a survey or session.

4.1 Study administration

After starting the *IRTlib Editor*, the *Studies* view is displayed. In this view, the first step to prepare a new configuration is to **add a new study**:

https://youtu.be/7VKf6U3oeM4

The created *studies* appear as cards in the *Studies* view. Note that the order in which the studies are displayed in the *Study view* does not matter.

Detailed instructions on how to create a *study* can be found here in the embedded help:

Embedded program help

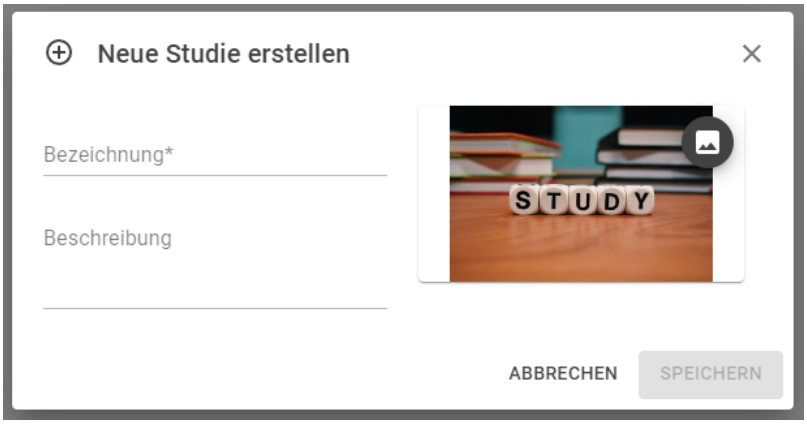
4.1.1 Creating Studies

The *IRTLib Editor* is used to create configurations for studies, which can then be used in an *IRTLib Player* to carry out computer-based assessments.

4.1.1.1 How do I get started?

To start configuring a study, click on the plus icon at the bottom right:

Then enter a *name* and optionally a *description* in the **Create new study** dialogue.
Make sure that only letters (upper and lower case), numbers and a _ are allowed for the *name*.

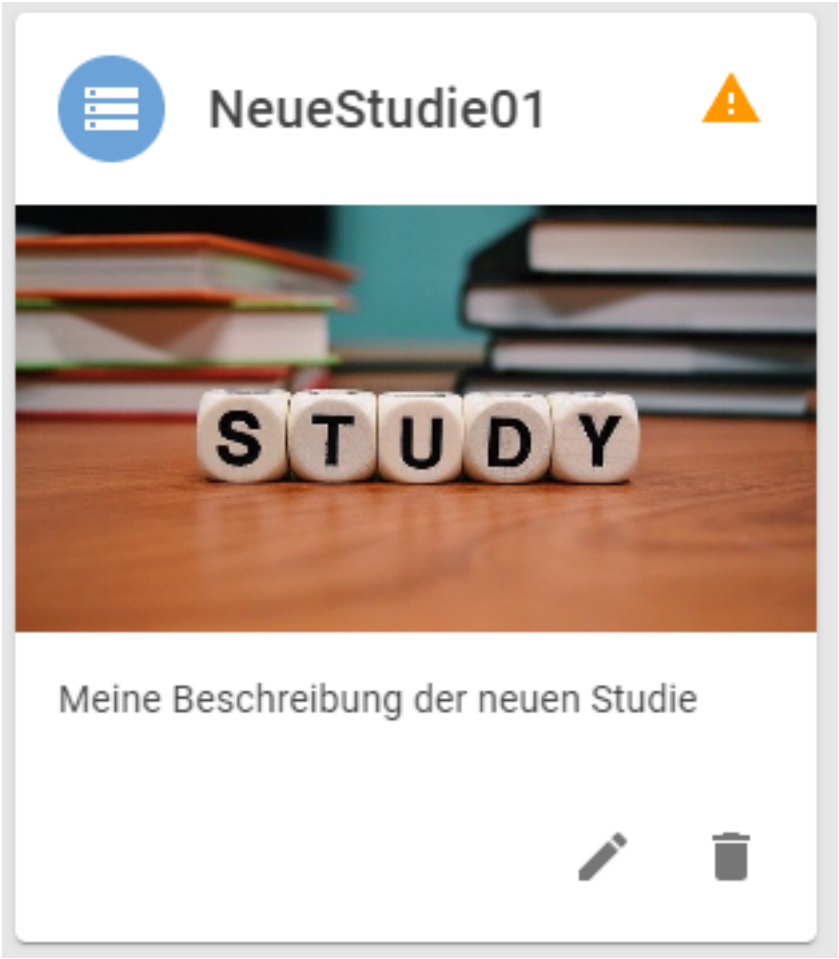


Then click on *Save*.

If required, you can also assign an image to a study using the following icon. This image is used in the *IRTLib Editor* for this study:

4.1.1.2 What's next?

Created studies are displayed as tiles in the study overview:

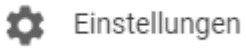


To continue with the creation and configuration of a study, click on the small edit icon:

4.1.2 Further functions and notes

- **Delete Study**: You can also delete studies using the recycle bin icon. The deletion of studies cannot be undone:

- **Change Language**: The menu item *Settings* takes you to the item *General settings*, where you can change the language of the *IRTLib Editor*.

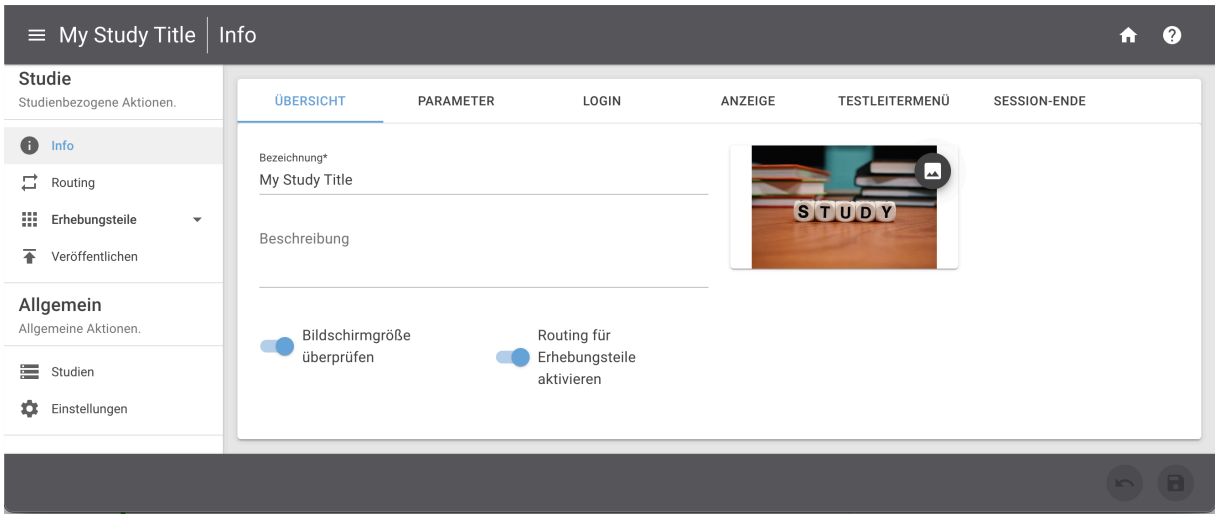


This item also gives you access to the *CBA ItemBuilder Runtimes* available in the *IRTLib Editor* (support for the use of *CBA ItemBuilder* content created with different versions of the programme).

4.2 Basic configurations

The configurations of a particular study, including versioning and publishing, are managed within studies (i.e. after opening a study for editing by clicking on the edit icon at the bottom right of the card).

Created studies that are displayed in the *IRTlib Editor* in the *Studies* view can be opened for editing.



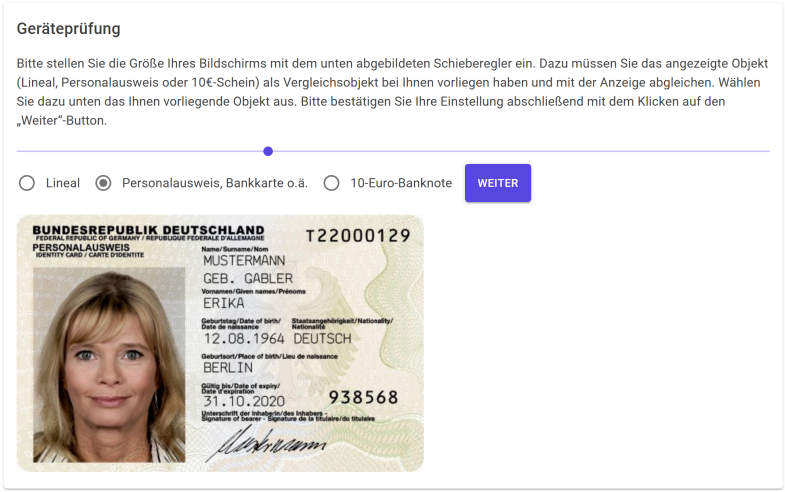
Detailed information on the basic configuration of a study can be found here in the embedded help:

Embedded program help

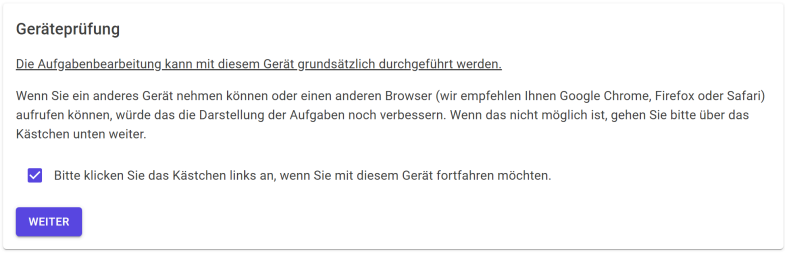
4.2.1 Settings for the Study

- **Name**: How should the *study* be named? Make sure that only letters (upper and lower case), numbers and a _ are allowed for the name.
- **Description**: This optional field is provided so that you can enter a detailed description of the *study*. Special characters and umlauts etc. can also be entered here.
- **Activate Routing for Survey Parts**: *Studies* consist of one or more *survey parts*. The *survey parts* are administered as a linear sequence by default. If the option *Enable Routing for Survey Parts* is selected, the order of the *Survey Parts* can be defined with *Blockly*-based routing. This enables dynamic sequences of *Survey Parts*, whereby call parameters of the study can also be used, for example, to assign different sequences.
- **Check screen size**: In surveys where the screen size is not known, this option can be used to compare the size of objects (EC card, banknote, ID card) with representations on the screen.

  The device check is carried out with the following dialogue:

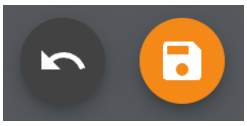


If the option *Force Suitable Screen Size* (in the *Display* section) is not activated, test processing can still be started. If the resolution is too small, the following dialogue is displayed:



Note: This option is currently not configurable.

If changed settings are to be retained, the changes must be saved using the disc symbol. Otherwise, the discard icon can be used:



4.3 Access to studies (login)

The *IRTlib software* supports various ways in which people (test participants, respondents, …) can authenticate themselves for an assessment. The configurations include two aspects:

- *Login mode*: Is access required (login, login+password, passphrases/token) or not? And if credentials are required, what are *valid values*?
- *Login source*: How is the login information retrieved (direct input on the platform, CBA ItemBuilder item, ….) or passed (login parameter or call parameter)?

Detailed information on the configuration of the login of a *study* can be found here in the embedded help:

Embedded program help

4.3.1 Configuration of the login

In the *Login* section, you can configure how test participants who start an assessment (either by calling up a link in a browser that refers to the online *IRTlib Player* or by starting the offline *IRTlib Player*) are to be identified or authenticated.

- **Login Mode**: The *IRTlib Software* supports various *Login Modes*.
  - *Random identifier*: When a session is started for the first time, an identifier is generated in this *Login Mode*. This random but unique character string (a so-called *UUID*, i.e. a *Universally Unique Identifier*) is used as a personal identifier in all data (i.e. result data) and all other stored data (e.g., log data/trace data, snapshot data, etc.).
  - *Username*: If test participants are expected to identify themselves by a unique string (e.g. a number or text used as an access identifier), a *Study* can be configured with the *Login Mode username*. Access to the assessment is then only possible if the character string entered as *username* is valid. The underlying idea is that the study configuration is loaded with a list of valid usernames and that a test participant must enter a valid username before he or she can start the assessment. Only authenticated test takers can access the assessment content defined as *Study* and answer the tasks or questions.
  - *User Name and Password*: If not only valid usernames but also a password is to be used in a *Study* to authenticate test takers, the *Login Mode Username and Password* allows a username and password to be entered. Analogue to *username*, both pieces of information must then be stored in the study configuration.
  - *Access Token*: If the valid user names are not to be saved in the study configuration, the option *Access Token* can be used. Each token that corresponds to a defined schema is then accepted and used as an identifier for the test participants.
- **Storage for Session Data**: In the case of online deliveries, an assessment can be continued after an interruption. This functionality is also required, for example, if the page is reloaded in the browser (e.g. by forcing a *Reload*/F5, or by closing and reopening the URL). To ensure that sessions originating from the same person (i.e. from the same browser) can also be continued, the software can be configured so that the identifier is saved in the client.
- **Valid Values**: The *IRTlib Software* provides the following credential validation mechanisms for the *Login Modes Username*, *Username and Password* and *Access Token*:
  - *List*: A list of valid authorisations (username or username and password, depending on the configuration of the *login mode*) can be defined as part of the study configuration. The information can either be edited in the *IRTlib Editor* or imported from a CSV file. Defined values can also be exported as a CSV file.
  - *Code for checking*: A *Blockly*-function can be specified, which returns *True* if the transferred login data is valid (otherwise *False*).
- **Group login**: Depending on the *Login mode*, the user name or access token is used as the person identifier. If the *Group login* option is activated, these transferred login data are used for authentication to identify the test participant as a member of a group (i.e. only test participants who know the user name can authenticate themselves as part of the group). An additional random identifier is then generated within the group to distinguish different people from a group.
- **Login source**: The *IRTlib Software* supports several possible options for how login credentials can be provided.

- *Platform*: A login dialogue is displayed by the *IRTlib Player* (i.e. the platform). The heading for entering the access data, the labelling of the input for user name and password, the labelling of the Next button, a welcome text and an instruction text as well as an error text for failed login attempts can be configured.
- *Parameters*: Valid login data for test participants can also be provided via the *command line* (i.e. parameters when calling up the offline version of the *IRTlib Player*) or via URL parameters (i.e. parameters when calling up the *study* via a link to an online version of the *IRTlib Player*). In this case, no login dialogue or login item is displayed.
- *Item*: As an alternative to an *IRTlib Player* dialogue, a *CBA ItemBuilder* task can also be configured, which serves as a login input mask. Within the item, a so-called *ExternalPageFrame* is used to send a specific JavaScript command to the *IRTlib-Player* to validate an input (an example can be found here).

  The login item must be available as a *CBA ItemBuilder* project file for the configured runtime environment (Runtime) and added to the study configuration. The integrated import dialogue can be used to add a login item to the study configuration. More information on importing *CBA ItemBuilder* projects can be found in the help for the *Items* section of a *Survey part*.

- **Additional parameters**: In addition to the *authentication* of test participants, the login information can also be stored in the *IRTlib software* as an additional parameter, which can then be used in the flow control, for example.
  - Parameters for file names: The `RawDataPath` (i.e. the relative path under which the offline *IRTlib player* saves the results data) and the `MonitoringFile` (i.e. the name of the file in which the offline *IRTlib player* writes information for study monitoring) can be configured as part of the login data.
  - *Blockly* variables: Additional parameters can also be stored as so-called *preload* variables with the login information.

Table 4.1: Summary of Options that can be Combined as *Configuration of the Login*

Login Mode	Storage for Session Data	Group Login	Valid Values	Login Source	Additional Parameters
Random indictor	yes	no	no	none	no
username	yes	yes	list or code	platform, item + parameter	values or parameter
username and password	yes	yes	list or code	platform, item + parameters	values or parameters
access token	yes	yes	scheme or code	platform, item + parameters	parameters

4.4 Display of assessment content

Studies can define how the *CBA ItemBuilder* content is to be displayed. The settings in the *Display* section can relate to the scaling and alignment of the content as well as the behavior of the *IRTlib Player* application.

Detailed information on configuring the *Display* of a *Study* can be found here in the embedded help:

Embedded program help

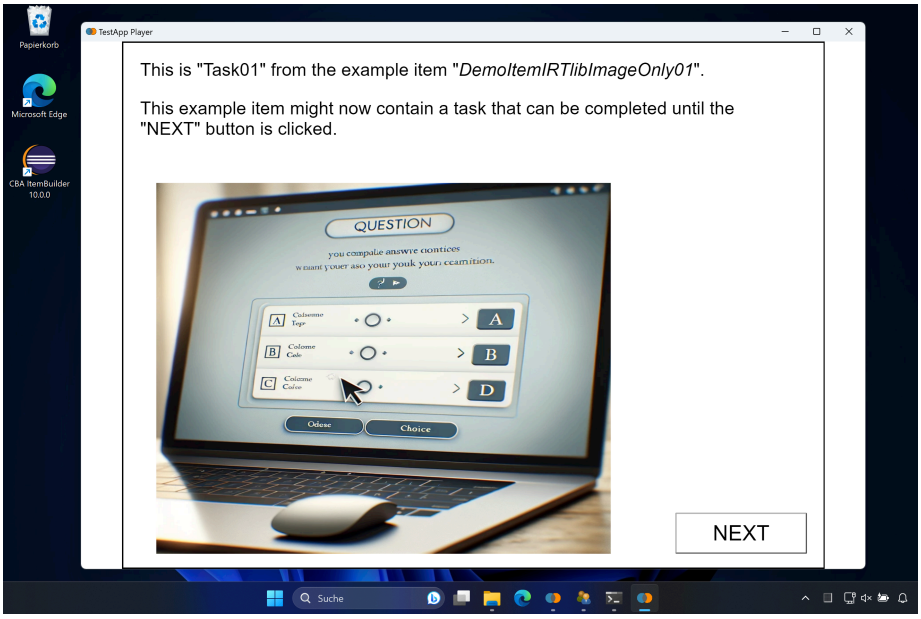
4.4.1 Display Settings

Selected options are available for configuring the display, which relate to the presentation of the assessment content and the use of *CBA ItemBuilder* content, which is created with a defined aspect ratio (width and height).

4.4.1.1 Window Mode

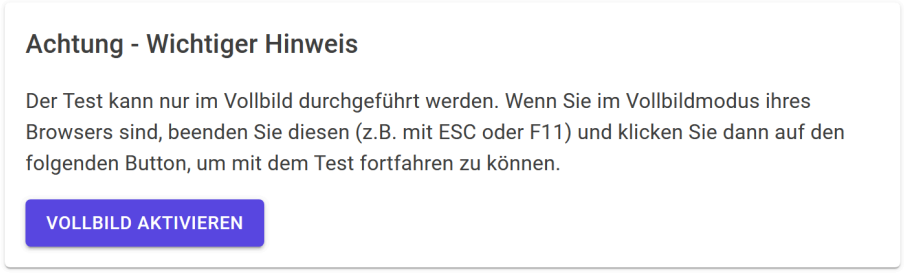
In the **Window Mode** selection, it can be configured whether an additional window is displayed in the *IRTlib Player*. The configuration is implemented differently depending on the environment:

- *Window*: In the configuration *Window*, a regular programm window is used in the offline *IRTlib Player*, in the online *IRTlib Player* the assessment content is displayed in the normal browser area, and the address bar and navigation buttons of the browser are visible in this mode.

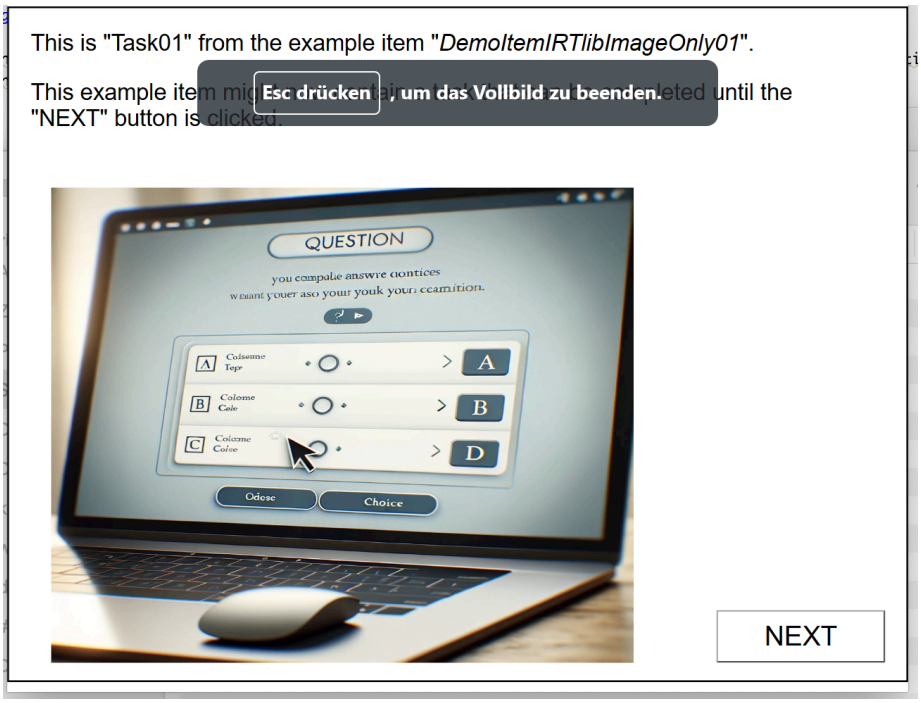


- *Full screen*: The offline *IRTlib Player* starts directly in full screen mode if this option is configured. This is also associated with a *Kiosk Mode*, i.e. access to other programms and (accidental) termination of the programm is only possible via the *Task Manager*. If a test manager, for example, is to be able to end the test, a *Test Administrator Menu* must be configured.

  The online *IRTlib Player* can also display assessment content in full-screen mode if this option is selected. If full screen mode is exited in the browser, the assessment content is then hidden. As it is not possible to automatically switch to full-screen mode in a browser, the target person first sees the following message from the platform:



By clicking on the button *Activate Full Screen* the full screen mode is started and the assessment content is then displayed without window frames and navigation areas of the browser. For a short time, the browser then typically displays a message that full screen mode can be cancelled again with Esc.



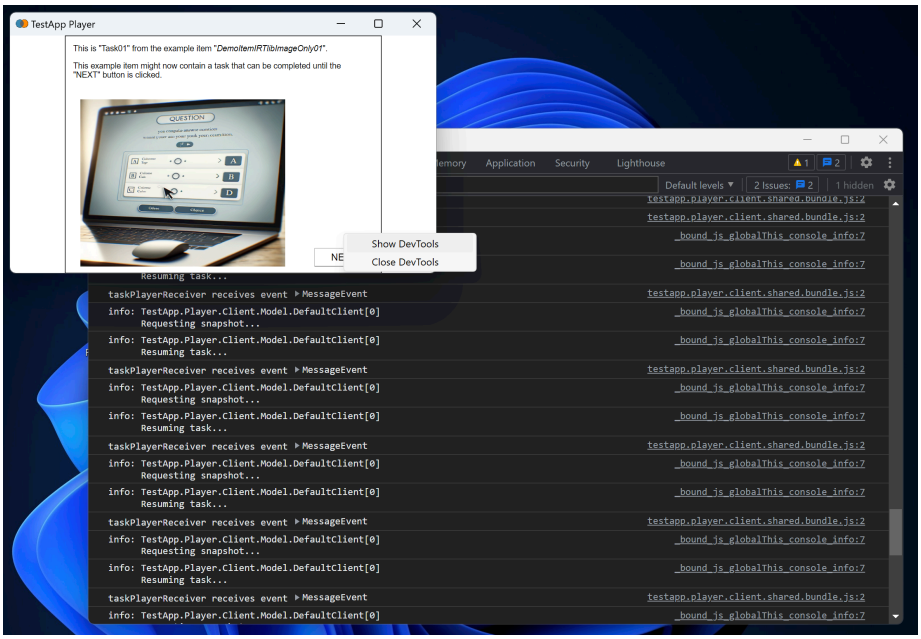
Note that this function is only available in browsers that support full screen mode (especially on older mobile devices, full screen mode is not fully supported; see for details e.g. on caniuse.com).

- *Full Screen, if Supported*: In this mode, the assessment in the online *IRTlib Player* is only displayed in full screen mode if the browser supports full screen mode. However, the content of the computer-based assessment is displayed in windowed mode when a study is delivered online and a browser that does not support full-screen mode is used. For the *IRlLib Player* offline, this configuration is identical to *full screen*.

> **Achtung - Wichtiger Hinweis**
>
> Der Test kann nur im Vollbild durchgeführt werden. Leider kann die Aufgabenbearbeitung auf diesem Gerät nicht durchgeführt werden, da der Browser kein Vollbildmodus unterstützt. Bitte prüfen Sie, ob Sie ein anderes Gerät (Computer oder Laptop) verwenden können!

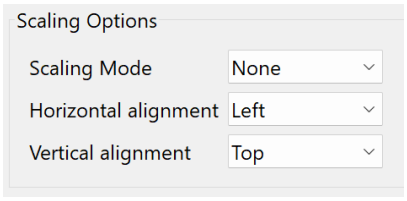
- *Debug*: This mode allows access to the browser's developer tools during test execution, which are intended for debugging by software developers.

  If the offline *IRTlib Player* is started with a study that has the *Debug* entry configured as *Fixed mode*, the so-called developer tools (*DevTools*) can be called up via the right mouse button during the task presentation.



4.4.2 Scaling and Alignment

Assessment content created with the *CBA ItemBuilder* is created for a specific size in pixels (width times height). The content can then be scaled for display on devices with different screen sizes and screen resolutions. In the *CBA ItemBuilder*, the option under *Scaling Options* is therefore available in the *Preview*:

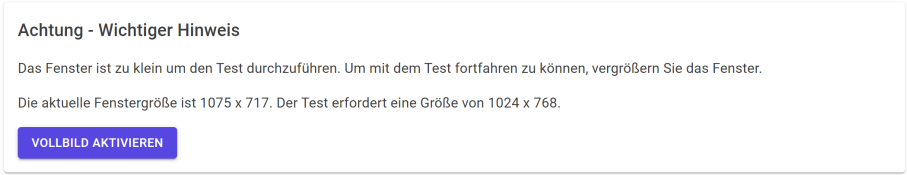


Analogue settings can be made in the *IRTlib Editor*.

- **Scaling**: Setting how content should be adjusted if the available space and size of the items differ (*Scaling Mode*).
  - *None*: The content is displayed without adaptation to the available window or screen size (corresponds to `None`).
  - *Upscale*: Content is enlarged so that the available space is utilised (corresponds to `Up`).
  - *Downscale*: Content is scaled down so that it fits on the screen/in the window (corresponds to `Down`).
  - *Window size*: Contents are enlarged and reduced (corresponds to `Both`).
- **Horizontal Alignment**: The options *centred* / *left* / *right* are used to align item content horizontally if the width of the window or screen is greater than the width of the content.
- **Vertical Alignment**: The options *centred* / *top* / *bottom* are used to align item content vertically if the height of the window or screen is greater than the height of the content.

4.4.2.1 Further Settings

- **Force suitable screen size**: If *Scale down* or *Window size* is not selected for *Scaling*, this option can be used to force that you can only start task editing if the available size of the window or screen is larger than the required width/height of the items. Otherwise, the following message is displayed:



Note: The display settings refer to all *survey parts* within a *study*. If several studies are configured in an *IRTlib player*, the settings must match each other, i.e. it is not possible to administer a study in *window mode: window* or in *window mode: full screen* at the same time with one instance of an *IRTlib player*.
If changed settings are to be retained, the changes must be saved using the disc symbol. Otherwise, the discard icon can be used:

4.5 Menu for test administrators

If the execution of assessments is accompanied by test administrators or interviewers, functions can be defined password-protected for test administrators.

Warning

Even if you do not need the functionality of a test administrator menu to carry out your data collection, you should still define a test administrator menu if you plan to collect data offline with the *IRTlib Player*. This is the only way to ensure that you can exit the application without the Task Manager (and without possible data loss) in the event of unforeseen events.

Detailed information on the configuration of the *Test Manager menu* can be found here in the embedded help:

Embedded program help

4.5.1 Concept of a Test Administrator Menu (Menu for Test Administrators)

The *Test Administrator Menu* is configured in two steps. Firstly, a key combination must be defined with which the test administrator menu can be requested. If this key combination is pressed during test taking, a window for entering the password appears. Test administrators enter the password known (only) to them and thus gain access to selected functions. For this purpose, one or more roles can be defined in the *IRTlib Editor* in a second step.

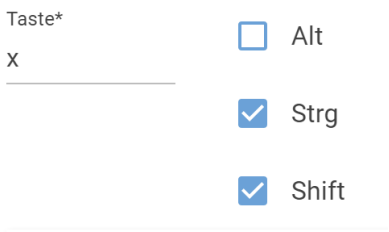
4.5.1.1 Access For Test Administrators

Firstly, a key combination must be defined.

- **Key**: The configuration of the key combination for the test manager menu first requires the definition of a key. To define a key, click in the field and press the key that is to be used for the test manager menu.
- **Modifiers** (Alt, Ctrl and Shift): For a key, you can also specify whether one or more modifiers must be pressed to open the test conductor menu.

Example:

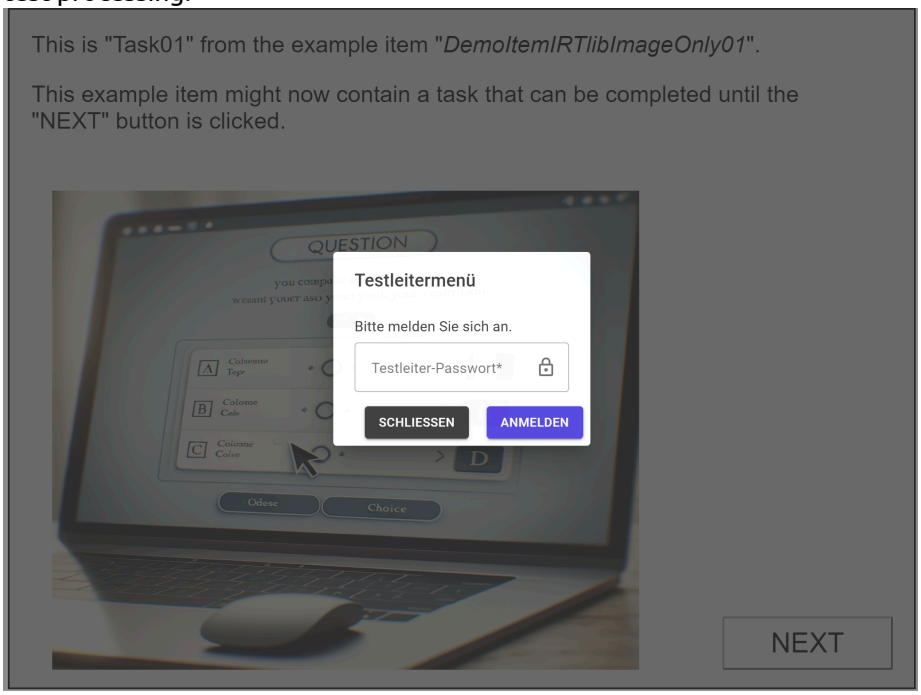
- The following configuration defines the key combination `Ctrl` + `Shift`+ `X`:



The defined key combination only opens the option to enter a password for test administrators during test processing in the *IRTlib Player*. To use the function, a password is required, which is defined together with a role in the second step.

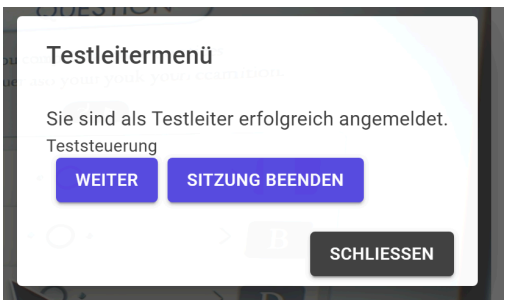
4.5.1.2 Roles

After calling up the defined key combination, the prompt to enter a password is displayed during test processing:

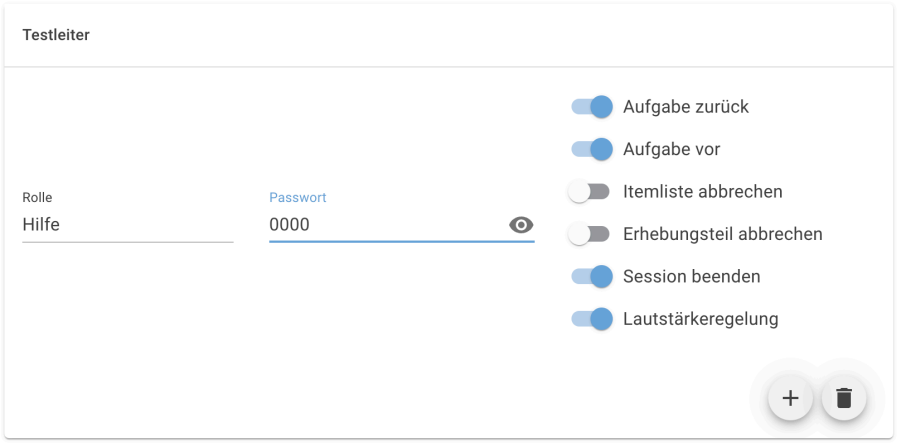


Which functions are actually accessible is controlled by which password is entered. Only if a valid password is known, functions of the test line can be called.
Example:

- In the following configuration, test administrators can use this password to jump to the next task (*Next*) or end the application (*End session*):



  To define a role, first click on the + symbol at the bottom right. The name of a role and a password can then be defined:



The name of the role is for documentation purposes only. The assignment of a unique password and the selection of one or more of the following functions are decisive for the functionality:

- **Task back**: Enables navigation to the previous task.
- **Task forward**: Enables navigation to the next task.
- **Cancel Item List**: Allows you to cancel the processing of the current item list. This option is particularly useful if the *Routing* option is activated in a *Survey section* and the definition of *CBA ItemBuilder* tasks is implemented using item lists.
- **Cancel Survey Part**: Enables the cancellation of the current survey part.
- **Cancel session**: Enables the current session to be ended.
- **Volume control**: Enables the volume to be changed.

The audio file that is played to control the audio output after the volume has been changed can be inserted in the *Audio for sound test* section and stored in the study configuration.
If changed settings are to be retained, the changes must be saved using the disc symbol. Otherwise, the discard icon can be used:

4.6 Completion of surveys

For the integration of assessments into external processes, it is possible to configure how to proceed after processing the assessment content in a *session*, i.e. what will happen at the *end of the session*.

Embedded program help

4.6.1 Session and End of Session

A *Session* refers to the execution of a survey with *one* person at *a* specific time. The content displayed in a session corresponds to a configured *Study* as it can be created in the *IRTlib Editor*. After all parts of the survey defined in a *Study* have been carried out, the *End of Session* is reached.

4.6.1.1 Configuration of the Session End

What happens after a *Session End*, i.e. how the *IRTlib Player* behaves at the end of a session, can be defined with the following options:

- **Start new Session**: A new session is started. This behaviour is not useful if the login data is transferred (either as *Startup parameter* or as *URL parameter*).
- **Display End Text**: If this option is selected, the platform displays the configured text. The text can be configured as a *Message on End Page*.
- **Display End Item**: Analogue to a *Login Item*, a *CBA ItemBuilder* item can also be defined to be displayed at the end of a session.

  The *End-Item* can finally trigger the termination of the offline *IRTlib Player*. An example of an *End Item* with the necessary JavaScript call can be found here.

- **Redirect to Exit URL (Redirect to Exit-Url)**: For online deliveries with the *IRTlib Player* it is possible to redirect to a URL. The *Redirect URL* can then be configured.

4.6.1.2 Further Options

Session ID can be Reused: If this option is activated, multiple data captures can be administered with one session ID.
If changed settings are to be retained, the changes must be saved using the disc symbol. Otherwise, the discard icon can be used:

5 Vorbereitung: Erhebungsteile / Preparation: Study Parts

Assessments that are administered with the *IRTlib software* consist of so-called *survey parts*.After configuring a study, at least one *survey part* must be created.

5.1 Survey part administration

After creating a *study*, the next step in preparing a test evaluation is to **add a new survey part** in the *Survey parts* view:

https://youtu.be/YFgu8uz8nkc

The created *survey parts* appear as cards in the *Survey parts* view. If studies consist of several survey parts, the order of the survey parts can be adjusted in the *Survey parts* / *Overview* view for *linear processes*. If *survey parts* are to be controlled depending on variables (e.g. passed *preload* variables or other *blockly* variables), routing between survey parts can be configured as an alternative.

Detailed instructions for creating *survey parts* can be found here in the embedded help:

Embedded program help

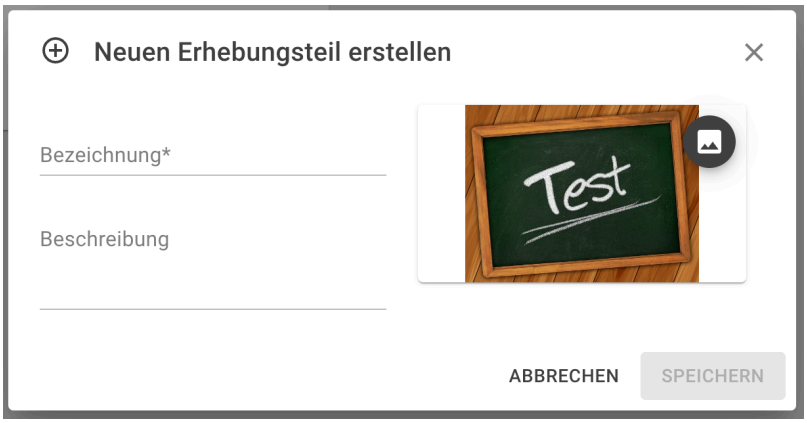
5.1.1 Create Survey Part

The *IRTLib Editor* is used to create configurations for *Studies*, which can then be used in an *IRTLib Player* to carry out computer-based assessments. *Studies* consist of one or more *Survey Parts*.

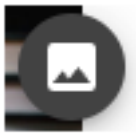
5.1.1.1 How does it work?

Once a *Study* has been created, a *Survey Part* can now be added via the plus icon at the bottom right:

Then enter a *Name* and optionally a *Description* in the **Create new Survey Part** dialogue.
Make sure that only letters (upper and lower case), numbers and a _ are allowed for the *name*.
Then click on *Save*.

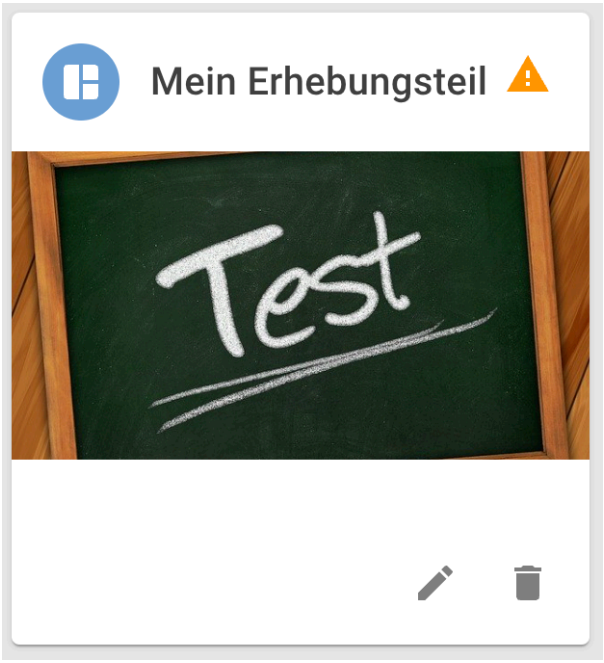


If required, you can also assign an image to a *survey part* using the following icon. This image is used in the *IRTLib Editor* for this *Survey Part*:



5.1.1.2 Edit Survey Part

Created *survey parts* are displayed as tiles in the survey part overview:



- To continue with the configuration of a *survey part*, click on the small edit icon:

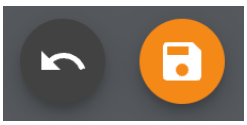
- **Delete survey section**: You can also use the recycle bin icon to delete *survey parts*. The deletion of *survey parts* cannot be undone:

5.1.1.3 Sort survey parts

If the option *Enable Routing for Survey Parts* is not selected in the *Info* view (section *Overview*) in the configuration of a *Study*, then *Survey Parts* are administered in the order in which they are displayed in the survey part administration.

- **Move Survey Parts**: To change the order of *Survey Parts* using drag-and-drop, the *Change Order* mode must first be activated using the following toggle icon:

The tiles can then be put in the desired order. The *Change Order* mode is ended when the disc icon is clicked or the changes are discarded:



The order of *study parts* can be changed in the *study parts* view:

https://youtu.be/Ag0IcETZTdM

Before adding or selecting CBA ItemBuilder projects, as described in the section Assessment contents (items), selected items can be configured in the *Info* view.

A detailed description can be found here in the embedded help:

Embedded program help

5.1.2 Basic Configuration for Survey Parts

5.1.2.1 Name and Description

- **Designation**: The internal name of the survey part, which is displayed in the *IRTlib Editor* for editing and defining the process. Designations must not contain any special characters, spaces or umlauts and must not begin with a number.
- **Description**:Optional, additional description of a survey part.

5.1.2.2 Routing Within Survey Parts

- **Enable Routing**: The configured assessment contents in the *Items* section can be administered as a linear sequence, i.e. in the configured order.If a different sequence is to be used, the *Enable Routing* option can be selected here. The sequence can then be specified as a visual program in the *Routing* section.

5.1.2.3 Further Settings

- **Use snapshot**: So that CBA ItemBuilder *tasks* can be visited multiple times, their content is saved in so-called *snapshots* when the item is exited. *Snapshots* can also be used to display the contents of a *CBA ItemBuilder task* again at a later time. This option should only be deactivated if there is an important reason and the consequences (i.e. the unsaved *snapshot* data) have been carefully considered.

Adding and managing CBA ItemBuilder projects within the *IRTlib Editor* is done in the Items section.

Note on time limit

For the administration of time-limited survey parts, a time limit can be defined under processing-time. If the option *Limit processing time* is activated, one or more *tasks* can be defined, which are displayed in the event of a *timeout*. In addition, content can be defined in the pre-item(s) and post-item(s) section, which is administered before or after the time-limited part.

5.2 Insert assessment content (items)

The contents that are to be used in a survey section of type *CBA ItemBuilder* are transferred to the configuration via the *IRTlib Editor*, i.e. the configuration created with the *IRTlib Editor* also contains the *CBA ItemBuilder Project Files*. The *Items* view is available for adding or updating *CBA ItemBuilder* projects.

A detailed description can be found here in the embedded help:

Embedded program help

5.2.1 Configure items

5.2.1.1 Basic functions

Importing CBA ItemBuilder project files: The *IRTlib Editor* maintains a *list of known items* to which *CBA ItemBuilder* project files that are not yet known can be added. To add a project file, first open the *List of known items* with the + symbol and then select the *Import* button.

Update already imported CBA ItemBuilder project files: If a *CBA ItemBuilder* project file is already included in the *List of Known Items*, the project files can be updated. They are then not added to the *List of known items*, but the existing *CBA ItemBuilder* project file is stored in a newer version. To update an item, it must first be selected in the list of items in a *survey section*. This activates the update symbol. In the *Update item* dialogue that then opens, an updated version of a *CBA ItemBuilder* project file can be added using the *Import* button.

- **Preview of CBA ItemBuilder project files**: Items added in a *Survey Part* section can be viewed directly in the *IRTlib Editor* in a built-in preview function. To view an item, it must first be selected in the list of items in a *survey section*. The *Preview* can then be called up using the eye symbol:

- **Exporting CBA ItemBuilder project files**: *CBA ItemBuilder* project files that have been imported into the *IRTlib Editor* can be exported for further editing with the *CBA ItemBuilder*. To export a selected item from the list of items in a *Survey Part*, the download icon can be called up:

- **Deletion of CBA ItemBuilder project files**: The items inserted in *Survey Parts* can be deleted from a *Survey Part*. The delete symbol removes the item from a *Survey Part*, but it remains in the *list of known items*:

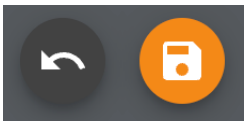
Note: It is not yet possible to delete *CBA ItemBuilder* project files from the *List of known items*. This functionality is not necessary because *CBA ItemBuilder* project files are only transferred from the *IRTlib Editor* to the configuration of a *Study* if *Tasks* from a *CBA ItemBuilder* project file are used in a *Survey Part*.

5.2.1.2 Sorting items (linear process)

- **Sorting CBA ItemBuilder project files**: If the *Enable Routing* option is not selected for a *Survey Part*, then the order can be adjusted in the list of items using the following button:

The items are then administered exactly as they appear for a *Survey Part* in this list.

Note: Changes to the *Items* view must be saved using the disc symbol or discarded using the undo symbol:



5.3 Processing time

If the administration of a linear sequence of *CBA ItemBuilder tasks* is to be administered with a limited processing time, this can be implemented by defining a maximum processing time (in seconds). If, for example, a test content is to be administered for a maximum of *28* minutes, a time of *1680* seconds is defined as the *processing time*. The message that is to be displayed when the processing time expires can be defined as one (or more) *CBA ItemBuilder tasks*.

A detailed description can be found here in the embedded help:

Embedded program help

5.3.1 Define Time Limit

Survey Parts without *Routing* can easily contain a time-limited section. To do this, the option *Restrict Item Time* is activated in the *Time Limit* view and a time limit in seconds (>0) is entered.
Four groups of *CBA ItemBuilder Tasks* are distinguished for a time limit, which are defined in different places in the *IRTlib Editor*. The items for which the time limit is to apply are defined in the *Items* view (analogue to non-time-limited *Survey Parts*):

- **Items**: Items that are displayed until the time limit has been reached.

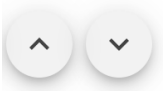
In the *Time Limit* view, the following can also be defined:

- **Timeout Items**: Items that are only displayed if the time-limited items have not been completed within the limited processing time.

Finally, the following tasks can be defined as individual views of the configuration of *survey items*:

- **Prologue Items**: Items that are displayed before the time-limited section.
- **Epilog items**: Items that are displayed after the time-limited section.

The icons for the following operations are available in all the above dialogues:

- Add: 
- Refresh: 
- Preview: 
- Download/Export: 
- Delete: 
- Sort: 

Note: More complex designs with several timers can be implemented with the *IRTlib Editor* if the option *Enable Routing* is activated in the overview view for a *Survey Part*.
Note: Changes to the *Time Limit* view must be saved using the disc symbol or discarded using the undo symbol:

Opening/closing credits items

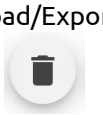
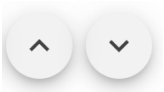
A central concept for the implementation of time limits in the *IRTlib software* is the separation of time-limited items and additional assessment content that is administered *before* or *after* the time-limited part.

- Items administered *after* a potentially time-limited section of an assessment are referred to as *post-items*.

Embedded program help

5.3.2 Items After a Time Limit

Survey Parts allow the definition of items in different sections. Items in this section *Epilogue Item(s)* are displayed after the items defined in the *Items* section of a *Survey Part*. The separation into *Epilogue Item(s)* and *Items* is particularly useful if a time limit is activated under *Time Restriction*. The following options are available for configuring items in the *Epilogue Item(s)* section:

- Add: 
- Refresh: 
- Preview: 
- Download/Export: 
- Delete: 
- Sort: 

Note: Changes to the *Epilogue Item(s)* view must be saved using the disc symbol or discarded using the undo symbol:

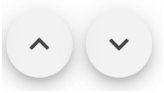
- Items that are administered *before* a potentially time-limited section of a survey part are called *prefix items*.

Embedded program help

5.3.3 Items Before a Time Limit

The *Survey Parts* allow the definition of items in different sections. Items in this section *Prologue Item(s)* are displayed before the items defined in the *Items* section of a *Survey Part*. The separation into *Prologue Item(s)* and *Items* is particularly useful if a time limit is activated under *Time Restriction*.
The following options are available for configuring items in the *Prologue Item(s)* section:

- Add: 
- Refresh: 

- Preview: 
- Download/Export: 
- Delete: 
- Sort: 

Note: Changes to the *Prologue Item(s)* view must be saved using the disc symbol or discarded using the undo symbol:

5.4 Variables

Under Development

This function is currently under development.

Embedded Program Help

(This functionality is still under *development*).

5.5 Codebook

Under Development

This function is currently under development.

Embedded Program Help

(This functionality is still under *development*).

5.6 ItemPool

Under Development

This function is currently under development.

Embedded Program Help

(This functionality is still under *development*).

5.7 Routing within survey parts

If *CBA ItemBuilder* tasks are not to be administered in a linear sequence that is fixed in advance and identical for all test subjects, then the *Routing* function of the *IRTlib software* can be used.

A detailed description of *Routing within survey parts* can be found here in the embedded help:

Embedded program help

5.7.1 Summary of Routing within Survey Parts

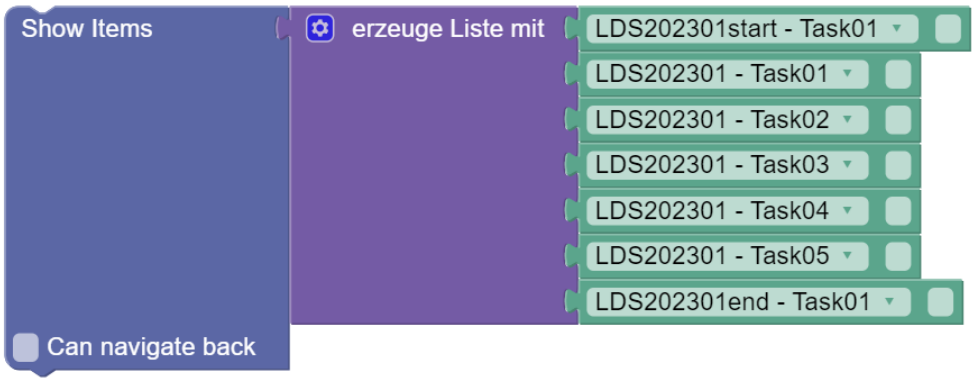
The sequence of *CBA ItemBuilder* tasks can be defined here using *Blockly* (i.e. a form of visual programming). *Blockly*-based sequencing is available if the option *Enable Routing* is selected for a *Survey Part*. The option can be found in the *Info* section of a *Survey Part*. If it is activated, the *Survey Part* contains the entry *Routing*.

5.7.1.1 Examples

The basic idea of using *Blockly* for the definition of processes in *computer-based assessments* will first be illustrated with a few examples.

- **Example for linear sequence**

  Based on the *CBA ItemBuilder Tasks* added to a survey part in the *Items* view, a linear sequence of *Tasks* corresponds to the following *Blockly* definition:

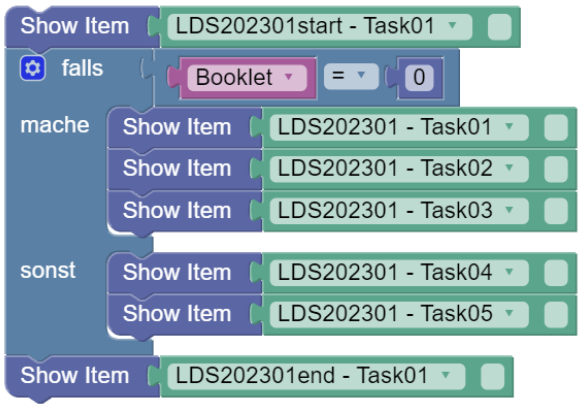


  A list of *CBA ItemBuilder Tasks* is passed to the *Blockly* element *Show Items*, which is created with the operator *create list with*. The list is processed in the order shown, whereby each *CBA ItemBuilder Tasks* is displayed until the `NEXT_TASK`- *Command* is executed.

  An equivalent formulation of a linear sequence can also be made with several *Show Items* blocks if no back navigation is necessary:

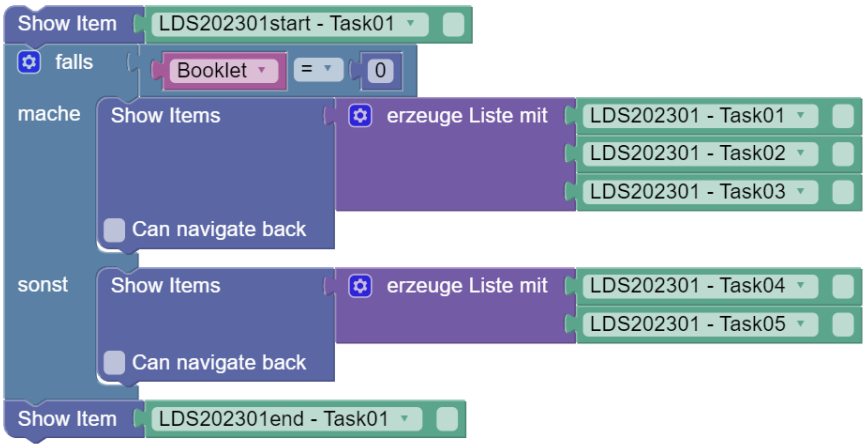
- **Example for simple test booklets**

  With the help of an variable (here: *booklet*) and a simple *if/make*-condition, it is now possible to define a sequence that administers different items depending on the value of the variable:



  The items for start and end are always administered, tasks 1-3 only if the variable *Booklet* has the value *0*, tasks 4 and 5 if the variable *Booklet* has a value other than *0*.

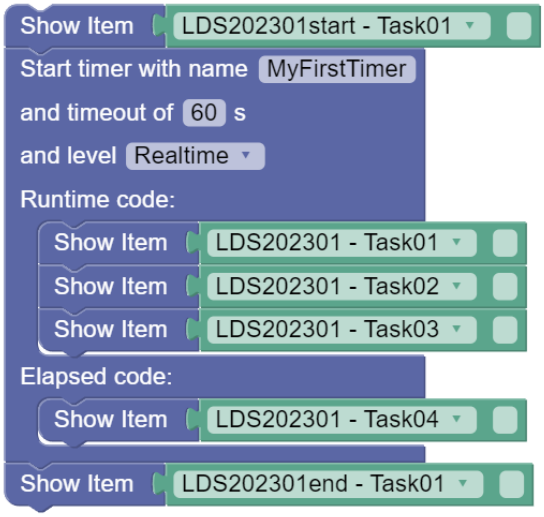
  Alternatively, the identical sequence can also be created using the *Blockly* operator for displaying item lists:



  Both variants are completely equivalent in terms of functionality, but the second approach with lists allows the use of the back navigation option within the booklet-specific tasks.

Example for process with time limit

To implement time-limited sections within a survey section using the *Blockly* configuration, the following *Blockly* component can be used:



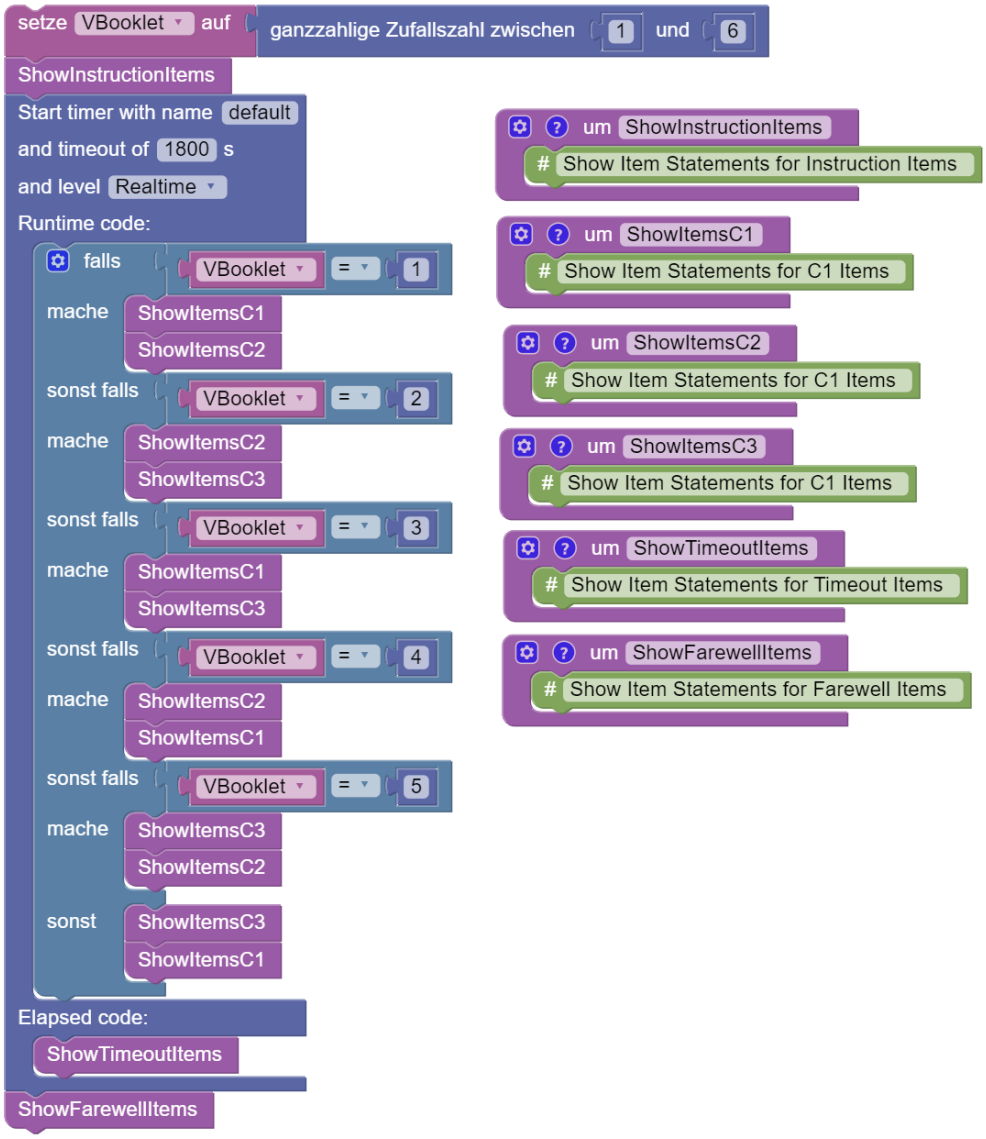
Each sequence begins with a start task that is not time-limited and ends with an end task that is also not time-limited. In between, there is a time limit for a section called *MyFirstTimer*, which has a time limit of *60* seconds.

Tasks 1, 2 and 3 are displayed in the *Runtime code* section with a time limit. If a timeout occurs, i.e. the three tasks are not processed within the *60* seconds, task 4 is displayed (also without a time limit).

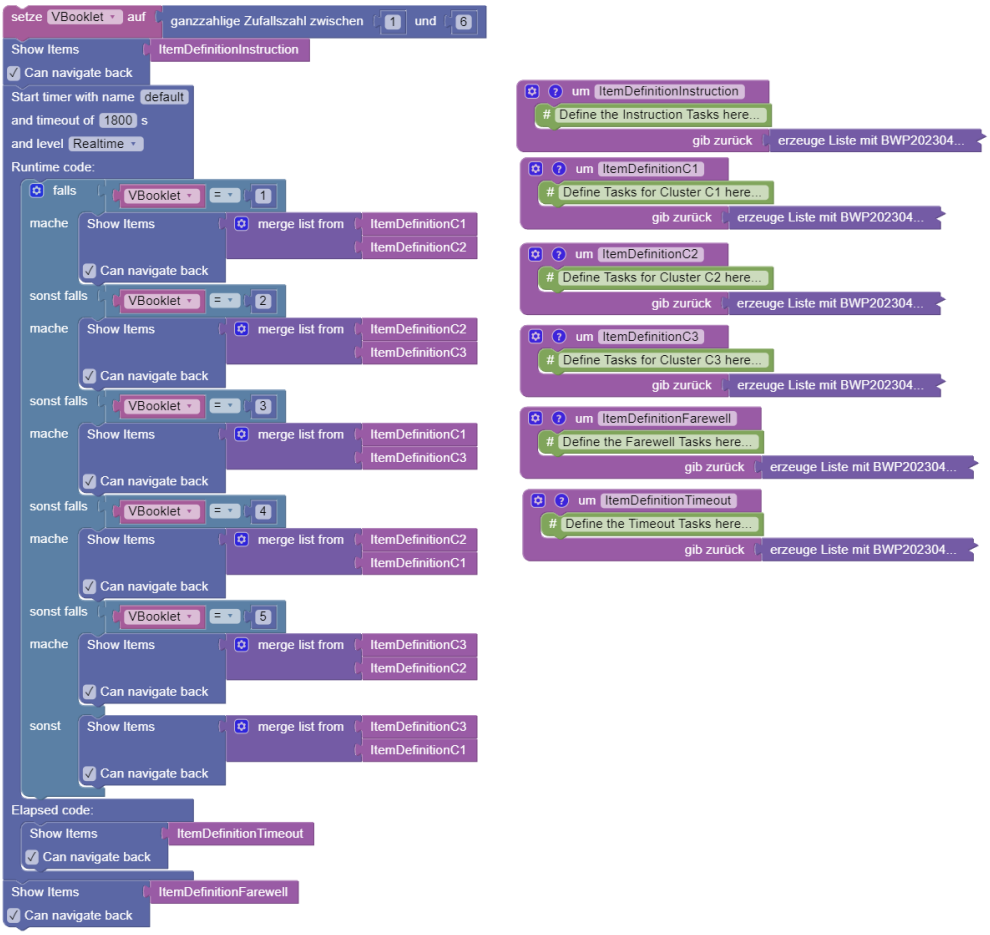
Example of simple booklet design with time limit

For many items, the definition of *booklet designs*, i.e. task sequences with balanced positions, can be simplified using functions or lists.

If no back navigation is necessary, functions can be used for the definition of clusters:



With back navigation, the functions can return lists of tasks:

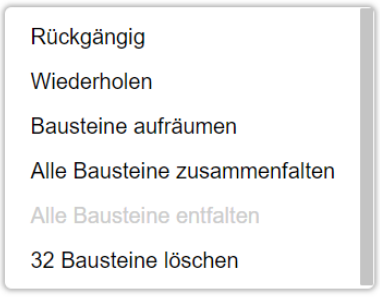


For more information see here.

5.7.1.2 Notes on using the Blockly editor

Processes are defined in the visual *Blockly* editor. Execution begins with the element that is aligned furthest up. If necessary, the workspace can be automatically aligned using the tidy-up function. To add *Blockly* operators, they can be dragged and dropped from the palette.

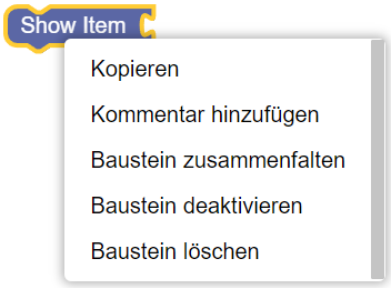
- **Delete**: Operators can be dragged to the recycle bin to delete them. Selected *Blockly* elements can also be deleted using the *Delete(delete)* button. Alternatively, selected *Blockly* elements can also be deleted via the context menu.
- **Redo/Undo**: Individual actions can be undone within the *Blockly* editor. The key combination 'Ctrl + Z' can be used for this. Pressing 'Ctrl + Y' repeats an action. By clicking in an empty section of the *Blockly* editor, you can access a context menu, which also contains the options for *Undo* and *Redo*:



- **Save**: Customisations in the *Blockly* editor must be saved. The floppy disc symbol is available for this purpose at the bottom right:

If you want to discard the change (as a whole), you can use the discard icon at the bottom right.

- **Zoom**: The view in the workspace can be enlarged with the icons + and reduced with −.
- **Context menu**: Further options are available via the right mouse button (context menu) in the *Blocky* editor. To call up these functions, a secondary click (right mouse button) must be performed on a *Blockly* element:
  - Copy duplicates the selected *Blockly* element, including all connected elements.
  - Commenting on blocks is possible.
  - Blocks can be deactivated/activated.
  - Some block types allow you to change the display form external/internal.
  - Blocks that contain further blocks can be folded/unfolded.
  - Blocks can also be deleted via the context menu.



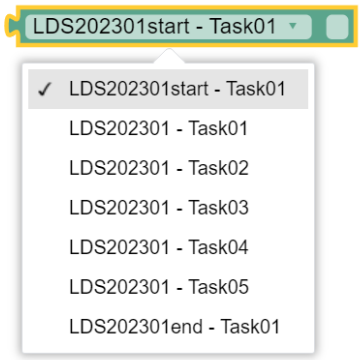
Some *Blockly* elements also provide a *Help* entry in the context menu, which refers to generally accessible *Blockly* documents (https://github.com/google/blockly/wiki/).

5.7.2 Use of Blockly for flow control

The basic functions for using the *Blockly* environment to control assessments can be found in the *Session* section.

5.7.2.1 Show individual items

CBA ItemBuilder tasks that have been imported in the *Items* view for a survey section can be accessed in the flow control, as shown in the examples above, using the following *Blockly* element for *Tasks*:

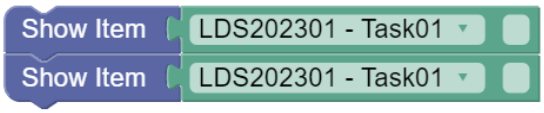


The element, which can be found in the *Session* section of the *Blockly* editor palette, can be configured using the selection list. Each *Blockly* element for tasks can refer to exactly one specific task, i.e. a flow definition usually consists of several such elements.
Blockly elements for tasks cannot be inserted directly into the flow, but are used together with a *Show Item* element:

The example for simple test booklets illustrates that sequences in the *blockly* definition are often defined by a sequence of several *show item* operators. *Show Item* operators can be inserted into conditions and loops, both within the main flow and within functions.

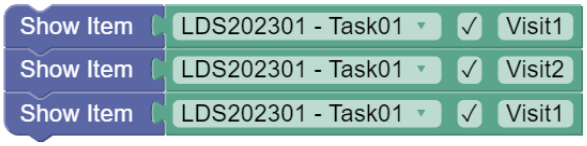
5.7.2.2 Use of scopes (scopes)

With the help of *Blocky*-based flow control, it is also possible to administer *CBA ItemBuilder* tasks multiple times within a flow:



When an item is called up again, the status from the last visit is restored, i.e. processing is continued. If items are to be resubmitted several times, i.e. unedited, automatic restoration may not be desired. The checkbox for specifying a *scope* (scope) can be optionally activated for this purpose:

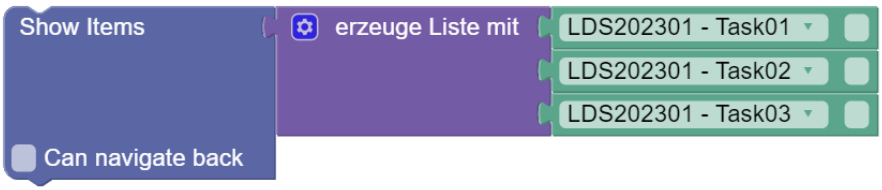
If nothing else is specified, the item is administered in the "default" scope. Alternatively, a text can be defined, as shown in the following example:



On the first visit, the task is displayed in the *Scope* "Visit1". This is followed by a new, independent display of the task in a different *scope* ("Visit2"). In the third call, the task is displayed again with the data that was already collected during the first visit (i.e. the *Scope* "Visit1" is used again).

5.7.2.3 Display multiple items (item lists)

As shown in the example for linear sequence, linear tests can also be displayed using lists of tasks. Lists can be used with the *Blockly* operator *Show Items*:

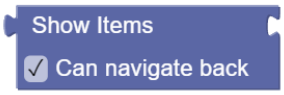


- **Back navigation**: The *Show Items* element for lists can be configured via the *Can navigate back* property. If this property is selected, *CBA ItemBuilder-Tasks* can use the *Command* `BACK_TASK` to request navigation to the previous *CBA ItemBuilder Tasks*.
- **Cancelling lists**: The use of lists also allows lists to be cancelled. Lists can be cancelled in two ways:
  - The *Command* `CANCEL_TASK`, which can be used within CBA ItemBuilder Tasks, is called.
  - In the test administrator menu, which has been configured for the study and, if necessary, customised using the *Blockly* operator Edit test administrator menu, the function *Cancel item list* is called.

  This cancels the administration of an item list and the *Blockly* process is continued after the *Show Items* block.

5.7.2.4 Display of items with storage of the results

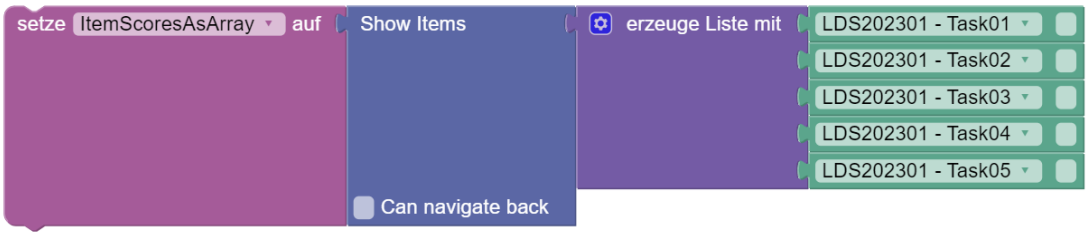
The operators *Show Item* (for individual items) and *Show Items* (for item lists) are also available as operators for value assignments:



These can be used to assign item processing results to variables (string or array) and then evaluate them for process control.

- Single task:

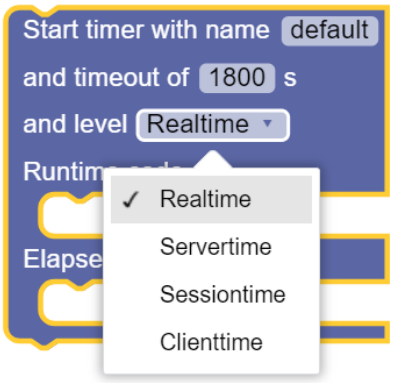
- List of tasks:



5.7.2.5 Definition of time limits

As already illustrated in the example process with time restriction, the *Blockly* block *Start time with name* can be used to implement the time-restricted administration of items.

The *Blockly* element *Start timer with name* allows the definition of time limits. Each time limit can have its own name. The time must also be specified in seconds. This can be used to define the type of time to be used:



- Realtime: The timer runs in real time. It is not affected by server downtimes or a session restart.
- Servertime: The timer runs in server time. Is not affected by a session restart, but does not take server downtime into account.
- Sessiontime: The timer runs within a session. Is interrupted in the event of an interruption due to server downtime or a session restart.
- Clienttime: The timer only runs in client time and is also interrupted when the session is paused.

Finally, two positions can be filled with further *blockly* operators (such as one or more *Show Item* blocks for displaying individual items or one or more *Show Items* blocks for displaying lists):

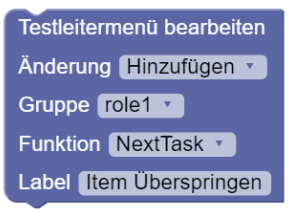
- Runtime code: These blocks are filled until the defined time has elapsed.
- Elapsed code: These blocks are only filled in if the *Runtime code* was not completed within the time.

5.7.2.6 Blockly operators for the test administrator menu

In the study definition, test administrator menu functions can be created for one or more roles. Roles combine different functions that can be differentiated using the password to be entered by the test administrator.
Customise standard functions: The following standard functions can be defined for a study in the *Info / Test leader menu* section:

- *Navigation*: Task forward / Task back
- Lists*: Cancel item list
- Exit*: End survey part and end session
- Volume control*: Adjust the audio volume during the assessment

During the processing of a survey section, the following *blockly* operator can be used to customise the test administrator menu in the flow control for specific contexts:

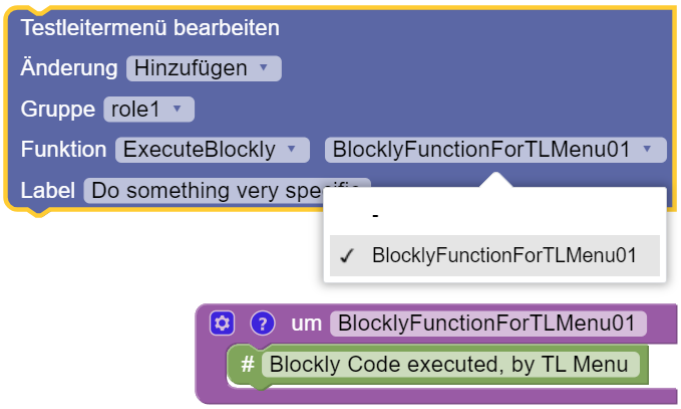


The test administrator menu can be changed for each of the standard functions (in the *Function* section) for a role (in the *Group* section) as well as the button label (in the *Label* section):

- *Add*: Function is added to the test leader menu
- *Remove*: Function is removed from the test conductor menu
- Deactivate*: Function is deactivated in the test conductor menu
- Activate*: Function is activated in the test conductor menu

Calling this *Blockly* operator in the test sequence defines the behaviour of the test administrator menu in the rest of the test sequence. In contrast to *Remove*, *deactivated* functions remain visible in the Test Manager menu, but cannot be executed (until they are *activated* again).

Using Blockly functions in the Test Manager menu: The *Blockly* operator for editing the test leader menu also contains the option to execute *Blockly* code (*ExecuteBockly*) in the *Function* section:

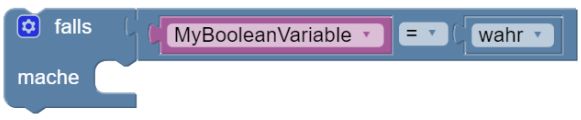


If *ExecuteBlockly* is selected, a function defined within the *Blockly* editor can be selected in the *Blockly* element *Edit test administrator menu*. The *Blockly* operators defined in this function are then executed when a test leader selects the corresponding button in the test leader menu at runtime.

5.7.3 Advanced Blockly usage

5.7.3.1 Flow control with conditions

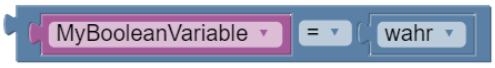
The *Logic* section contains the *Blockly* operator *if/make*, which can be used to implement conditions in the flow. Conditions are logical expressions, e.g. checking whether a preload variable has a certain value:



The blocky operators defined within the condition block (i.e. next to *make*) are only executed if the condition (*if*) is fulfilled. The example checks whether a Boolean variable has the value `true`. The condition is defined as a separate block that is connected to the *blockly* operator *if/make*. Here are the two components separately:

- Condition:

- Logical expression:



5.7.3.2 Use of logical expressions

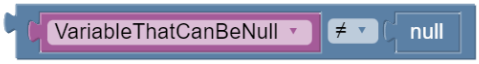
Logical expressions in conditions are based either on value comparisons or returns from functions. Value comparisons can be realised with the following *blockly* element:

The two slots can be filled with values. A corresponding *Blockly* element is provided in the *Logic* section for Boolean values (true/false):

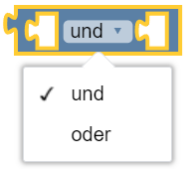
Conditions are also possible with variables of a different data type:

For numerical values, there is a corresponding *Blockly* element in the *Math* section, which contains operators for numbers and simple mathematical operations:

With its help and a numeric variable, the following condition can be formulated:
For technical reasons, it may also be necessary to check whether a variable has no value at all. This can be implemented by using the *blockly* component `null`:

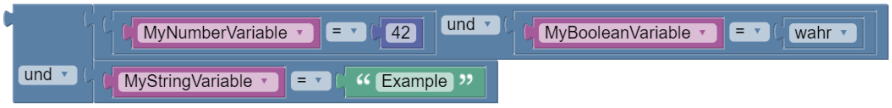


Combination of logical expressions: Individual conditions or logical expressions can be combined with the following *Blockly* element from the *Logic* section:



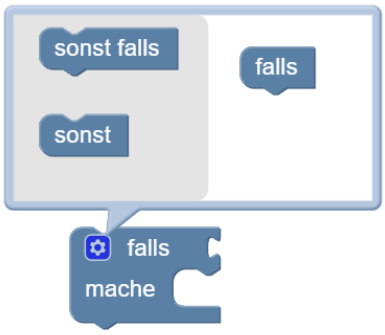
An *and* and an *or* linking of the statements is available for selection. The *and* link is true if both expressions are true, the *or* link is true if at least one of the two expressions (or both expressions) is true.

Several *logical expressions* can be nested inside each other:



Note: For a clearer display, the external display is selected for the external *and* link.

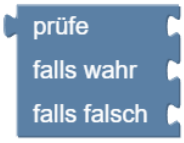
Multiple conditions (if / else): By clicking on the small cogwheel symbol of a condition block (*if/make*), it can be configured:



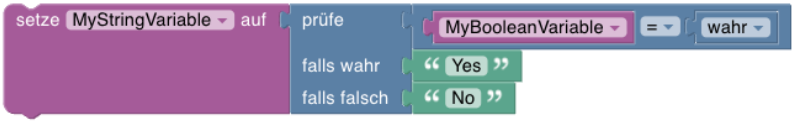
By adding an *unless* section, a further condition can be added. The condition defined in an *if* section is checked if the previous conditions (*if*) are not fulfilled. If a condition is fulfilled, the defined *blockly* operators are executed.

By adding an *if* section, blocks can be added that are executed if none of the conditions are met.

Check operator: For value assignments depending on a condition, the *blockly* editor provides a special operator *check-if-true-if-false*:



The operator combines a value assignment with a logical expression:



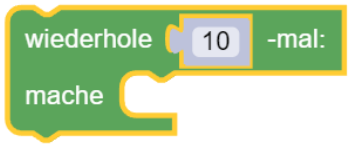
In this example, the string variable `MyStringVariable` is assigned the value `Yes` if the boolean variable `MyBooleanVariable` has the value `true`. If `MyBooleanVariable` has the value `false`, `MyStringVariable` is assigned the value `No`.

Negation: The following *Blockly* operator is available to reverse a logical expression (negation):

5.7.3.3 Sequence control with loops

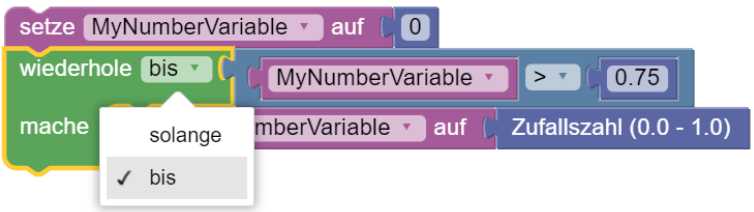
The multiple execution of *blockly* operators (and the actions that can be displayed with them) is possible with loops. The *Loops* section of the *Palette* contains the *Blockly* elements required for this.

Repeat n times: The following *Blockly* operator can be used to repeat the execution of blocks n times:

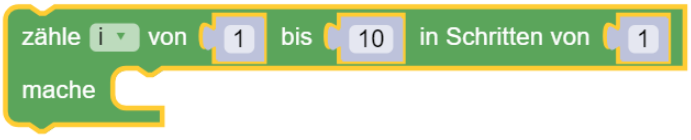


Repeat as long as: Loops can also be repeated *until* a condition is true (or *as long as* a condition is true):

Example:

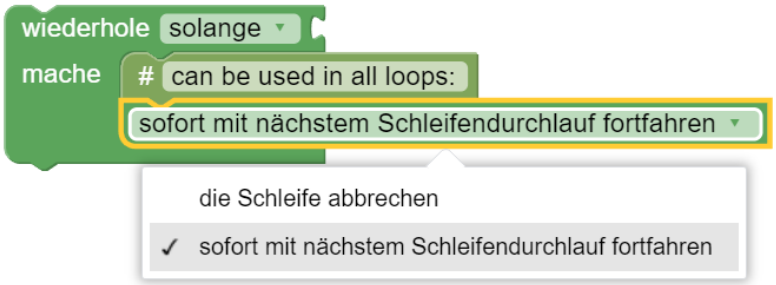


Count from/to: Loop with auxiliary variables:



For each value from list: Loop over all values in a list:

Cancel loops prematurely: The following *blockly* element can be used to cancel a loop (prematurely) or to start the next loop pass prematurely:



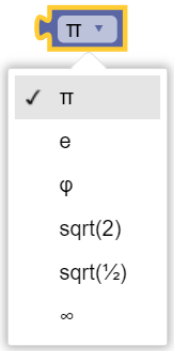
5.7.3.4 Operators for numbers and simple mathematical functions

The *Math* section of the *Palette* contains *Blockly* elements for using numbers and simple mathematical functions.

Expressions

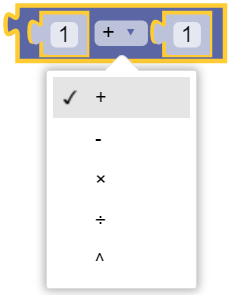
- Numbers: Integers / decimal numbers

- Symbols: Special symbols or constants:

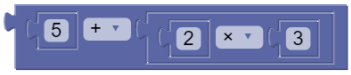


Basic functions

- Addition, subtraction, multiplication, division and power function of two arguments:



Nesting is possible, e.g:



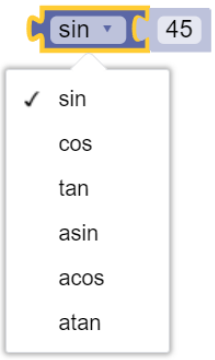
- Division with remainder:

- Whether a number is even can be checked with this *blockly* element:

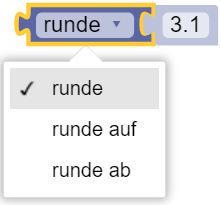
- With the following *blockly* element, a number can be limited to a section:

Built-in functions

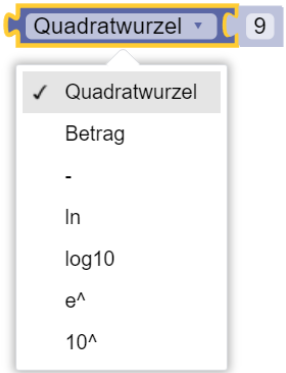
- Trigonometric functions:



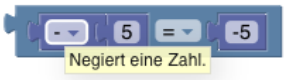
- Rounding of values:



- Further functions:



Note: − allows the negation of numerical values, as can be seen in the following example:



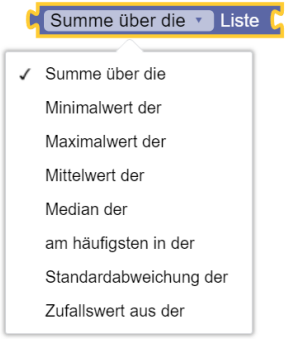
The example shows the *tooltip* for the help available in *Blockly* and an example where the number 5 is converted into the number −5 using the − operator. The outer condition (negating 5 results in −5) is therefore `true`.

Generation of random numbers: Two *blocky* elements are available for generating random numbers:

- Integers (in value range):

- Random number between 0 and 1:

Numeric functions for lists: Predefined functions for lists include:



Notes:

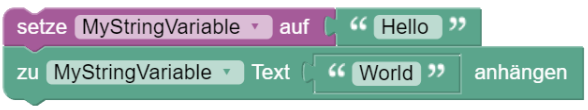
- If required, further functions can be implemented with loops for lists.
- When using the functions, please note that the list function can only be used for lists with numerical data types!

5.7.3.5 Operators for text and simple string operations

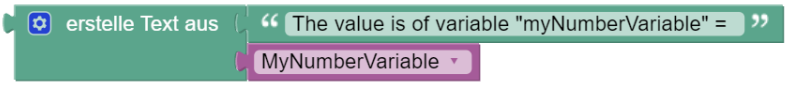
The *Text* section of the *Palette* contains *Blockly* elements for using strings.
Expressions: The following operator is available for creating text:

Chains: Various operators can be used to join text and assign it to variables:

- Append a text to a variable:



- Concatenate texts (and variable values) and pass them on to other *blockly* operators:



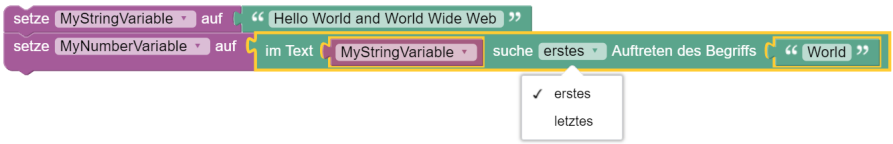
- Assign a variable to merged texts:

Text length: The length of a character string can be determined with the following *blockly* operator:

Check for empty string: Empty string variables can be recognised by the fact that the number of characters is 0.

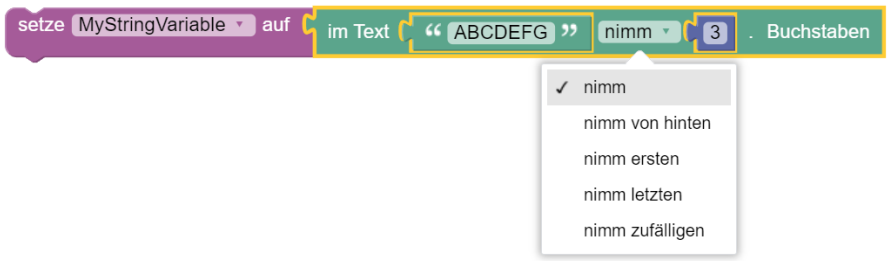
Alternatively, the following *blockly* operator can be used:

Find position in string: An operator that searches *in text* (passed by variable or as an expression) for the *first* or *last occurrence of a term* can be used as follows:



The position of the *term* within the character string (i.e. *in the text*) is returned.
Form sub-strings: The following operator takes the first letters from the transferred string *in text*. The number of letters is also passed.

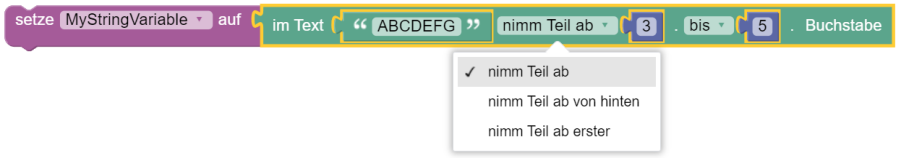
- Example (here, if the option *take first* is selected, the variable `MyStringVariable` is assigned the text `ABC`, i.e. the first three letters of the character string `ABCDEFG`):



letter function	parameter N	meaning
take	Yes	The first N letters are returned
take from last	Yes	The last N letters are returned
take first	No	The first letter is returned (corresponds to take with N=1)
take last	No	The last letter is returned (corresponds to take from behind with N=1)
Take random	No	A random letter is returned

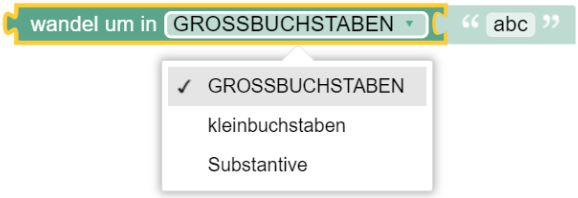
Letters from a character string can also be extracted using the following operator and assigned to a variable, for example:

- Example (here, for example, characters 3 to 5 can be taken from a character string):



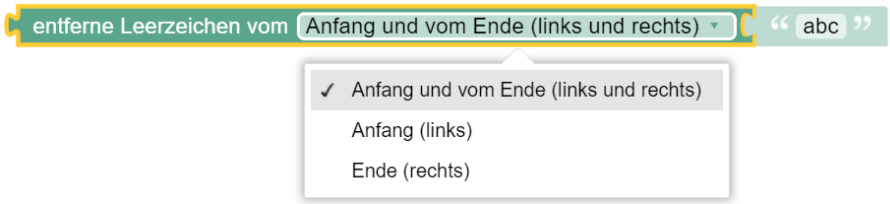
Change texts: Existing texts (either as expressions or from variables of *datatype string*) can be modified by applying operators.

- The following operator can be used to convert text to uppercase or lowercase:



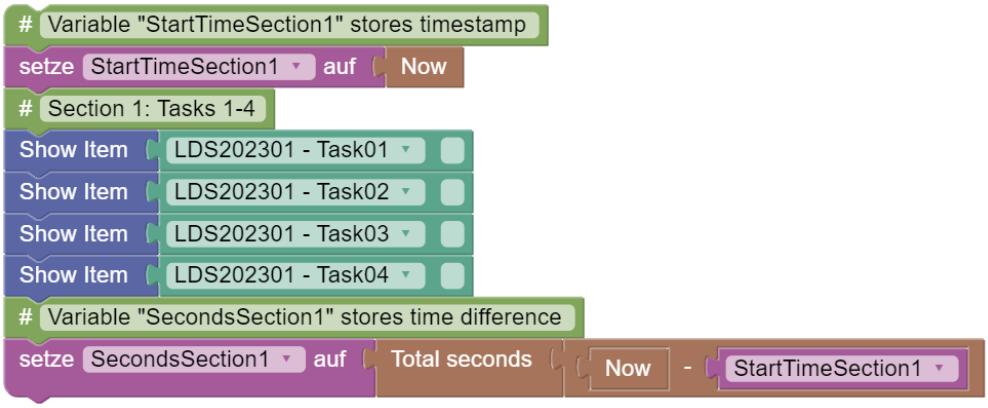
The `nouns` option converts the passed string into a sequence of words with a capital initial letter (except for strings that are written entirely in capital letters).

- Leading, trailing or leading and trailing spaces can be removed using the following operator:

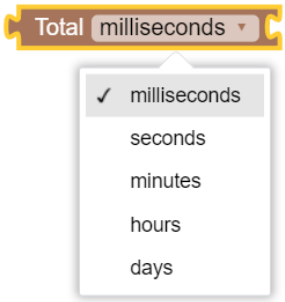


5.7.3.6 Operators for times and simple time operations

The *Date & Time* section of the *Palette* contains *Blockly* elements for using times within flow definitions.
Fixing points in time: Variables of the *datatype DateTime* can be assigned timestamps.
Determine time differences: Complete example: The following *Blockly* code measures the time for processing tasks 1 to 4. To do this, the start time is first recorded, and after the tasks have been processed, the time difference is determined and converted into seconds:



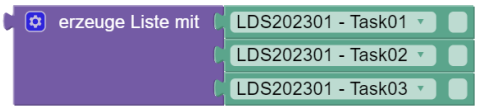
Conversion of time measures



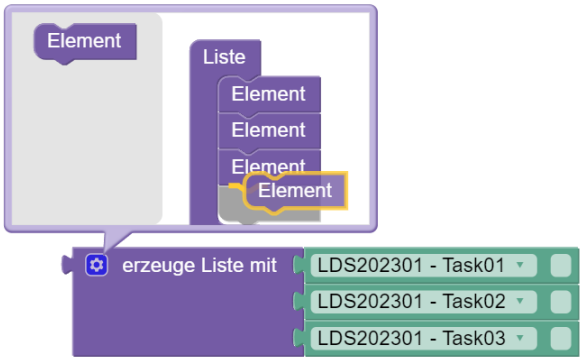
5.7.3.7 Operators for lists

The *Lists* section of the *Palette* contains *Blockly* elements for creating and using lists.
Create list: Various options are available for creating lists.

- Lists can be created from existing elements:

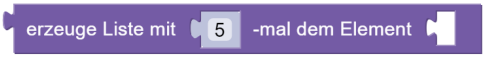


The number of elements of the *create list with* operator can be configured using drag-and-drop after clicking on the *cogwheel* symbol:

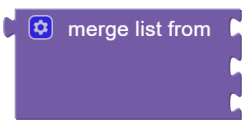


When creating lists, please note that the *Blockly* editor does not check the data type. Lists with values of different data types can be created (incorrectly), but do not lead to a functioning test sequence.

- Lists can be created by repeating an element:



Combining lists: Existing lists can be merged with the following operator:

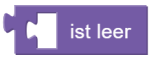


Sublists: A sublist can be selected from lists using the following operator:

Further operator options for *to*: *to from last* and *to last*.

List properties: The following operators are available to query properties of a list:

- The following operator returns *true* if the linked list is empty:

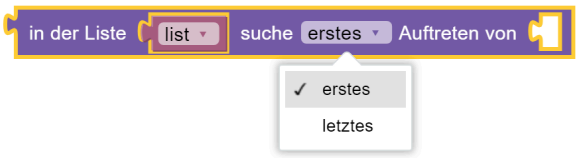


- The following operator returns the length of the list:

- The following operator returns the distinct elements of a list

Search and replace: The following operators are available for searching and replacing elements in lists:

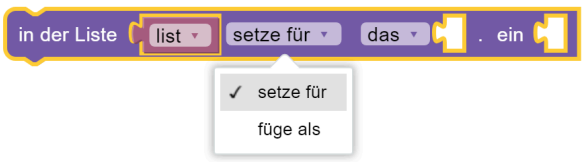
- The following operator finds elements in lists:



- The following operator returns / removes or replaces in a list and returns the element:

Further options of the operator for *that*: *from behind that* / *first* / *last* and *random*.

- The following operator replaces and inserts in a list:



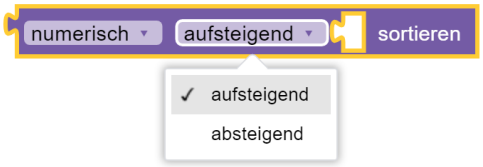
Further options of the operator for *that*: *from behind that* / *first* / *last* and *random*.

Conversion of lists and text: List and text can be converted using separators.

- The following operator creates a text from a list or a list from a text:

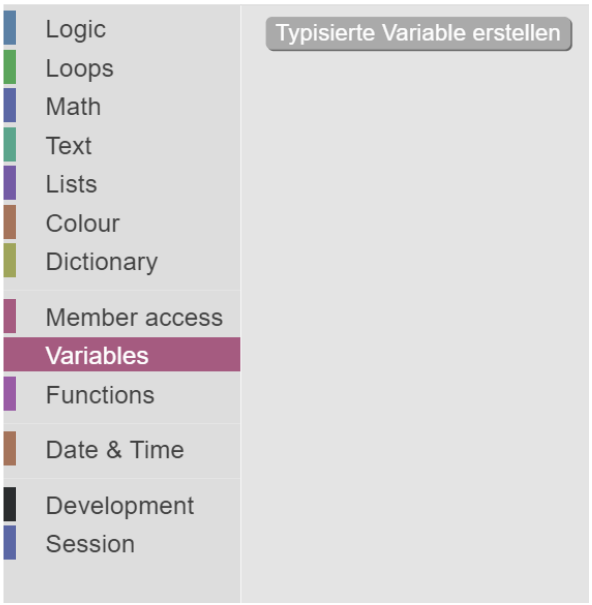
Sort lists: Elements in lists can also be sorted.

- The following operator returns the distinct elements of a list:

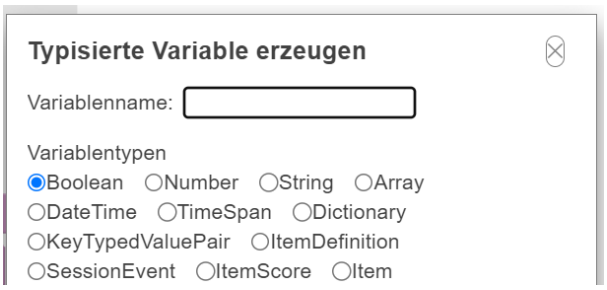


5.7.3.8 Blockly variables

The *Variables* section of the *Palette* contains *Blockly* elements for creating and using variables.
Create variable: To create a *Blockly* variable, the *Palette* contains the *Create typed variable*:



- *Blockly* variables always have a *variable name* and *data type*:



Simple data types and value assignments: The following basal data types are supported:

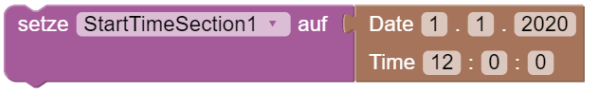
- *Boolean*: Logical truth values and logical expressions (`true` or `false`)

- *Number*: Data type for numerical values (with and without decimal place)

- *String*: Text values or character strings

The following data types are provided for times:

- *DateTime*: Date and time



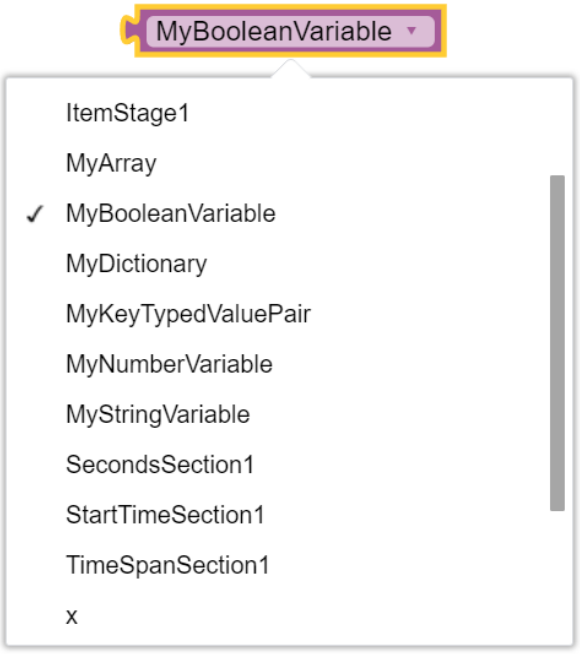
- *TimeSpan*: Time span

Data types for multiple values: In addition to the basal data types, data types for multiple values are also supported:

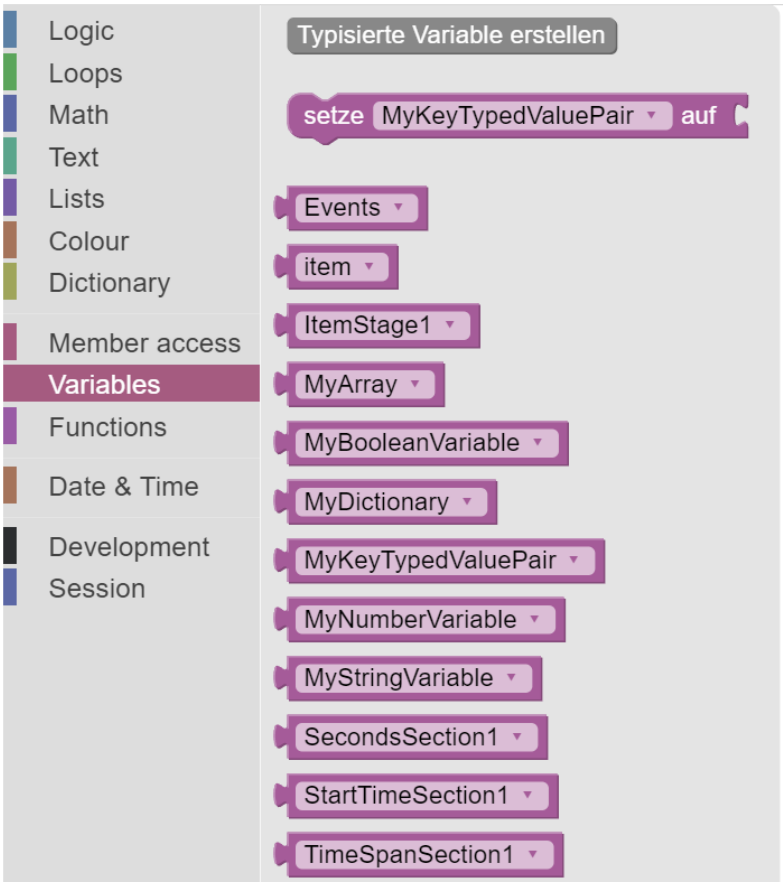
- *Array*: Data type for lists

- *Dictionary*: (documentation missing)
- *KeyTypedValuePairs*: (documentation missing)

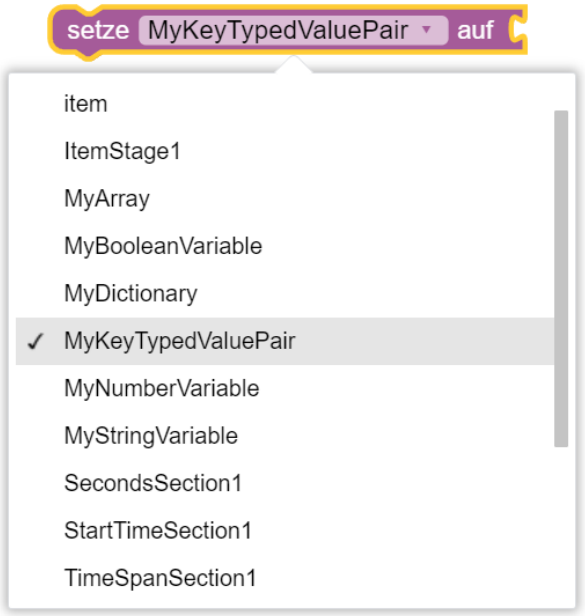
Use variable values: To use variable values, *blockly* elements with *inputs* can hold the following components:



- The variable to be used can be selected. For defined variables, there is also a *Blockly* element in the *Variables* section of the *Palette*:



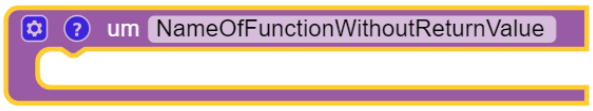
- The palette also contains a *blockly* element of the type *set … on*. This can also be used to select which value of the variable it sets:



5.7.3.9 Blockly functions

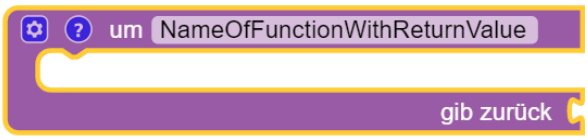
The *Functions* section of the *Palette* contains *Blockly* elements for using functions within flow definitions. Functions combine *blocky* code so that it can only be defined once but used multiple times.
Defining functions: Two different forms of functions can be defined.

- Functions without a return value:



  To be called, functions without a return value can simply be connected to previous and subsequent *blockly* elements in the sequence (i.e. they have an up and down connection):

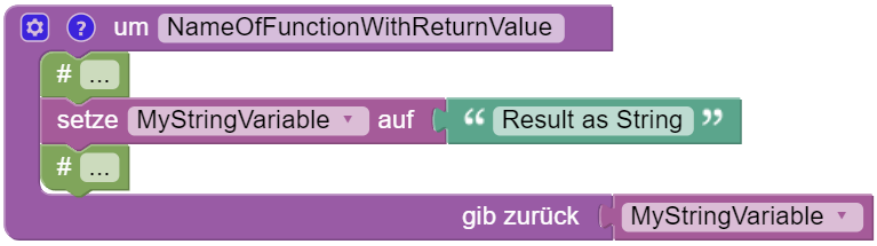
- Functions with return value:



  Functions with a return value can be called in an assignment block (i.e. they have a connection to the left):

The type to which an assignment makes sense depends on the type of the return value.

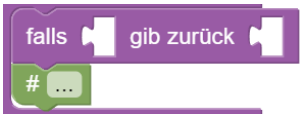
Defining return values of functions: Functions are defined by special *blockly* elements that can be inserted anywhere in the code editor.
Return values can be defined for functions with a return value. The return value can be added directly to the function definition next to *gib zurück*:



In addition, the following two *blockly* elements are available, which can only be used within a function definition (with return value):

- The operator *return* allows a value to be returned. After this, no further *blockly* elements can be placed in the flow within the function (i.e. the *return* operator has no downward connection):

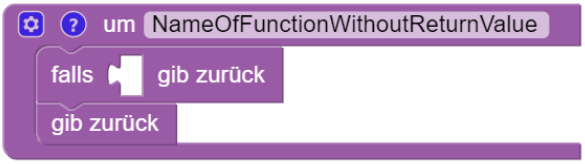
- The *if return* operator only returns a value if a condition is fulfilled. If the condition is fulfilled, the processing of the sequence in the function ends; if the condition is not fulfilled, the processing is continued (i.e. the *if return* operator has a downward connection):



- The *if-return* operator is therefore identical to the following combination of operators:

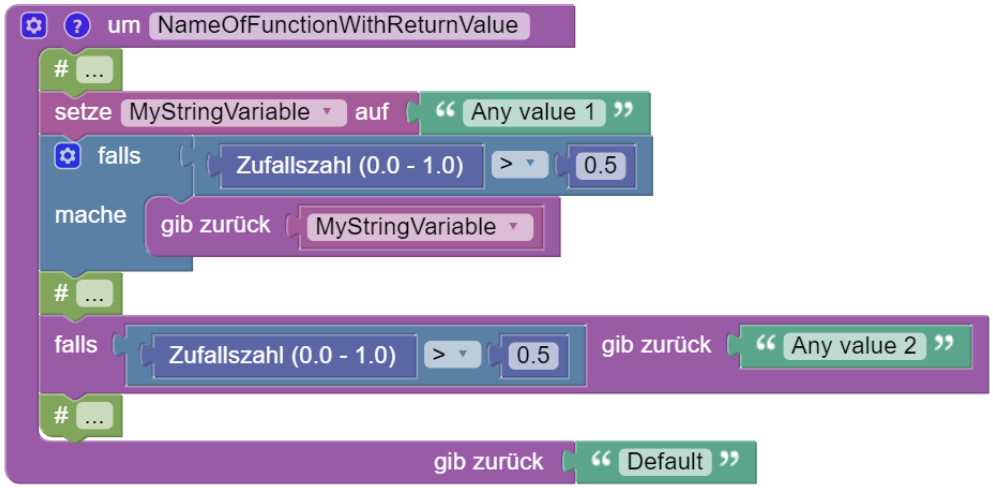
- Both operators (*if return* and *return*) cannot be used outside of functions:

- The two operators (*if return* and *return*) can be used within functions without a return value to terminate the execution of functions (but not to return values):



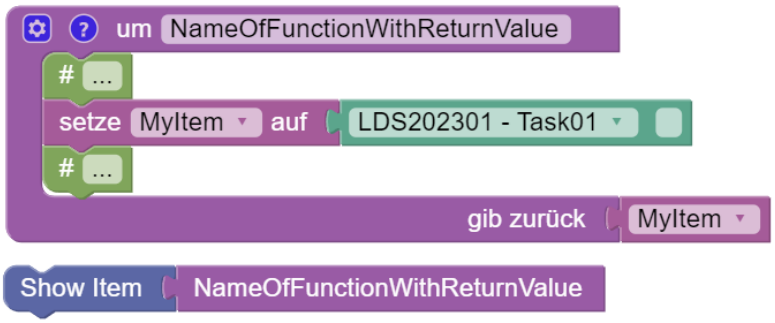
Example:

- The following function returns the value of the variable `MyStringVariable` (*Any value 1*) in 50% of the cases (i.e. if a first drawn random variable is greater than 0.5). In the other 50% of cases, another random variable is drawn, and if this is greater than 0.5, then the text *Any value 2* is returned. If this is not the case either, the text *Default* is returned:

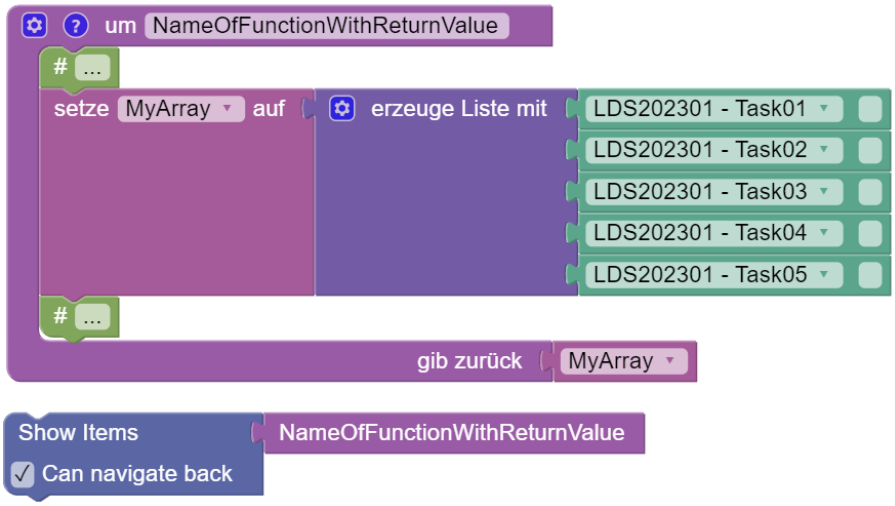


Return values are typed. The flow control also supports functions that …

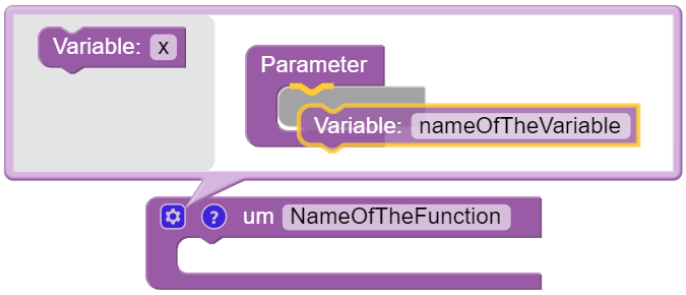
- … return individual *tasks*:



- … Return lists of *tasks*:

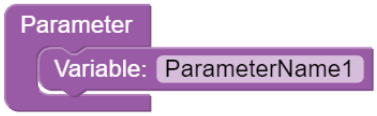


Defining call parameters of functions: Functions can also use parameters that are to be passed when the function is called (*call parameters*). Call parameters can be defined by clicking on the small cogwheel symbol of a function block:



The function is then called by passing it in accordance with the parameter definition:

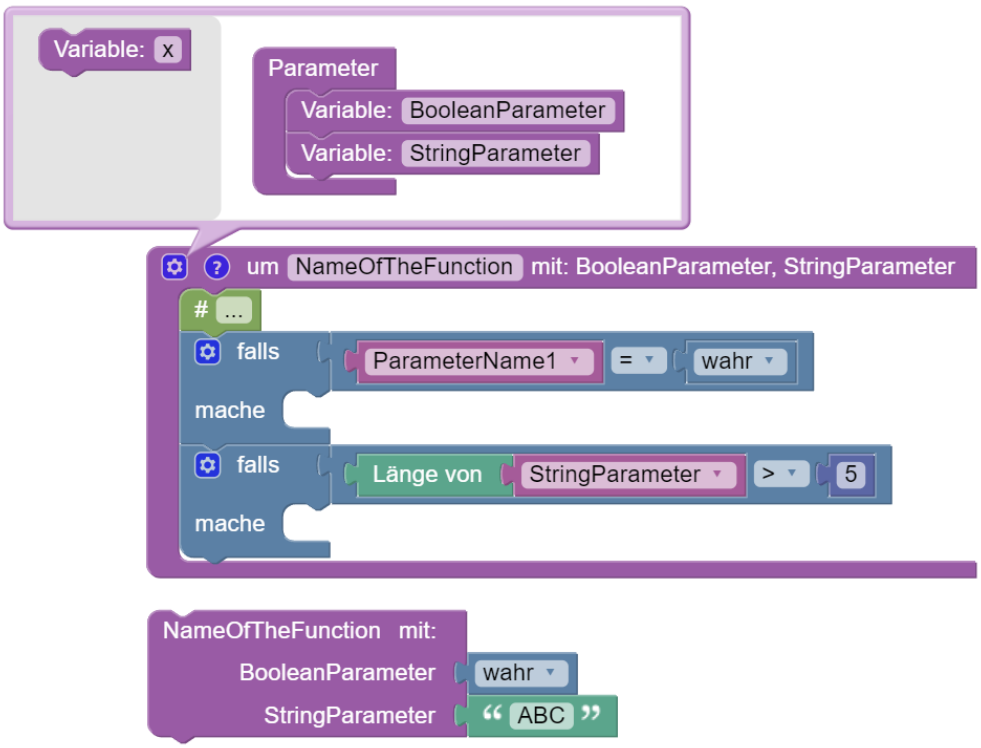
- Definition of a parameter:



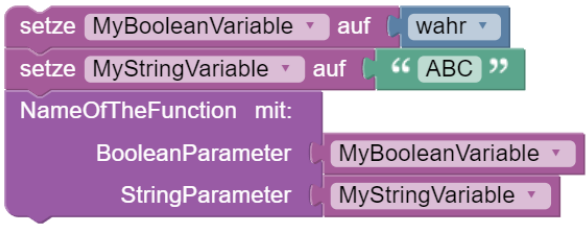
- Call the function with value:

Example:

- The following example shows a function with two parameters, their use within the function using the example of conditions and the call of the function with fixed values:



- Alternatively, the function can of course also be called with variables:



5.7.3.10 Use of item results in the flow control

(documentation follows)

5.7.3.11 Blockly operators to encode missing values

(documentation follows)

5.7.3.12 Blockly operators for writing data

(documentation follows)
Log data: The following operator can be used to store information directly in the log data:

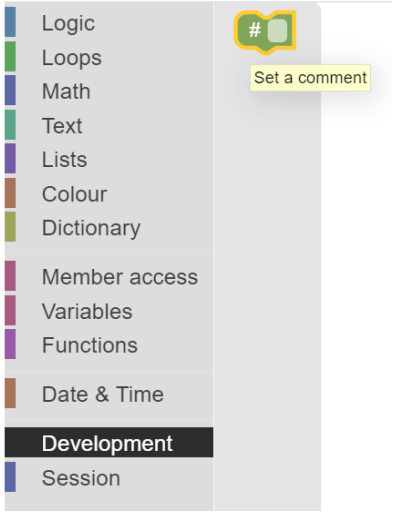
Result data: (documentation follows)
Monitoring data: (documentation follows)

5.7.4 Commenting on Blockly code

The *IRTLib Editor* supports two different options for commenting *blockly* code.

5.7.4.1 Comments as Blockly elements

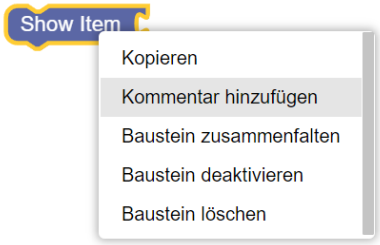
Comments that are to be permanently visible in the process can be added via the plaette in the *Development* section:



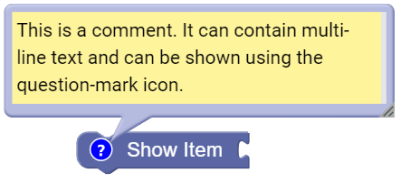
These comments can be moved like blocky operators and show one-line comment text.

5.7.4.2 Detailed comments on Blockly elements

For more detailed comments, each block can be added with a comment (and deleted if available) via the context menu:



These comments can comprise several lines and are displayed when the small ?-icon of a block is clicked.

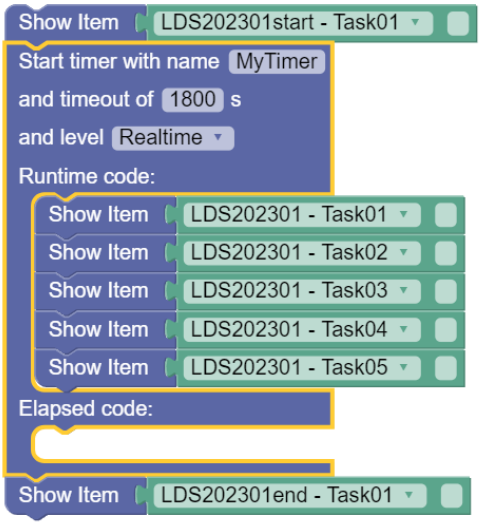


5.7.5 Presentation of Blockly code

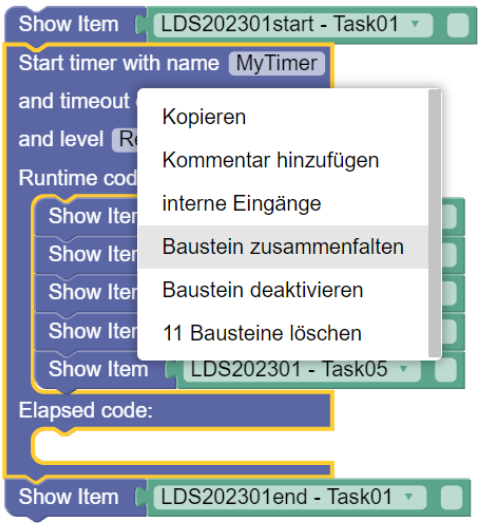
5.7.5.1 Unfolding / folding

Large and complex processes can sometimes become confusing in the *Blockly* editor. In order to hide *blockly* elements that are not required for viewing without changing the function of the flow definition, blocks can be *folded* together:
This is illustrated in the following example:

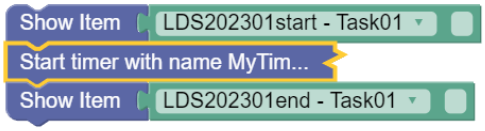
- Unfolded (i.e. complete) representation of the selected block:



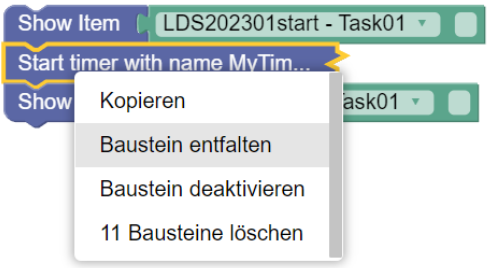
- Option to *fold* the block in the context menu:



- *Collapsed* representation of the block within the flow definition:



- Option to *unfold* the block in the context menu:

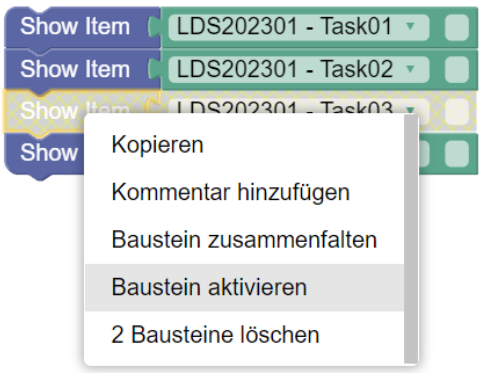


The *folding* / *unfolding* of *blockly* elements does not change the function of a flow definition and is only used for a clearer arrangement of complex flow definitions.

5.7.5.2 Deactivate / activate

Note: This function is still under development.
The *Blockly* editor offers the option to only deactivate *Blockly* elements instead of deleting them. Deactivated *Blockly* elements remain in the flow definition but are not executed.
In the following example, the block for displaying task 3 is deactivated, i.e. only task 1, 2 and 4 are displayed:

Activating or deactivating *Blockly* elements is done via the context menu:



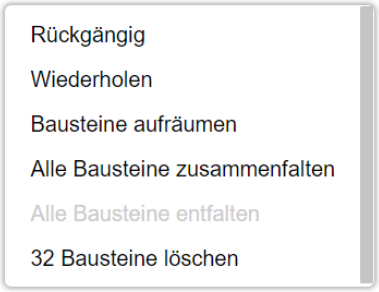
Internal / External: Some *blockly* elements with *inputs* (i.e. places where you can connect further blocks) allow you to switch between two display forms.

- Internal: The *inputs* are arranged within the blocks.

- Externally: The *inputs* are arranged on the side of the blocks.

  Both display formats are equivalent in terms of functionality.

Clean up: The context menu of the *Blockly* editor, which can be opened by clicking in an empty section, contains the *Clean up blocks* function:



By calling *Clean up blocks*, all *Blockly* elements in the *Blockly* editor are aligned vertically one below the other.

5.8 Routing between survey parts

If several *survey parts* are defined for a *study*, the sequence of survey parts can be defined in which respondents or test persons are presented with the contents of the *survey parts*.

In addition to simple linear sequences, sequences of several survey parts can also be configured with *blockly*-based routing.

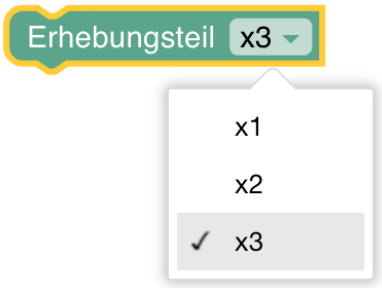
A detailed description of *routing between survey parts* can be found here in the embedded help:

Embedded program help

5.8.1 Summary of Routing between Survey Parts

The order of *Survey Parts* can be defined using *Blockly* (analogue to the definition of the order of *Items* within *Survey Parts*). This option is available if the option *Enable Routing for Survey Parts* is selected in the basic configuration for a study (in the *Overview* view).
For the general principles of using *Blockly* in the *IRTlib Editor*, see the help on *Routing within Survey Parts*.
Functions that are only available in *Routing between Survey Parts* are:

- Display survey part



  This *Blockly*-operator replaces *Show Item* within *Survey Parts*.

- Successful login

  This *Blockly*-operator has the value *true* if valid login information was specified before the maximum number of attempts (here: infinite, i.e. an unlimited number of times).

Note: Changes to the *Routing* view between *Survey Parts* must be saved using the disc symbol or discarded using the undo symbol:

Part II

Datenerhebung / Data Collection

6 Datenerhebung: Übersicht / Data Collection: Overview

6.1 Overview: Steps for using an IRTlib Player for Data Collections

Once a *Study* has been created and configured using the *IRTlib Editor*, a finalised *Version* of this configuration must be created. Versions *seal* and finalise all configurations and have a unique version number (referred to as a *Revision*). The use of *Revisions* makes the administration of data collections with the *IRTlib Editor* and *IRTlib Player* reproducible, as the revision number of a configuration is also saved in the data sets.

- **Check Configurations**:Before finalising and *sealing* a version, it is suggested to check all settings again. The *IRTlib Editor* provides an additional *Validation* feature for this purpose.
- **Create Sealed Version**: If no further changes are required, the version can be *sealed*. This is done by selecting the changes that have not yet been saved and clicking on the lock symbol in the *IRTlib Editor* in the *Publish* view, in which the *Study revisions* are displayed.
- **Export version**: Versions of studies that are available in the *IRTlib Editor* can be exported. It is necessary to export the configuration before it can be used with the *IRTlib Player*. When exporting, the complete study configuration including the imported *CBA ItemBuilder* content is downloaded as a ZIP archive.
- **Import Study into IRTlib Player**: Exported versions of studies from the *IRTlib Editor* can be imported into an *IRTlib Player* for use. There is an automatic mode for individual *Studies*. If several *Studies* are to be used simultaneously in one *IRTlib Player*, this can be configured manually.
- **Testing the Study**: Before the actual data collection can begin, each configuration should first be tested with synthetic test cases (i.e. systematically).

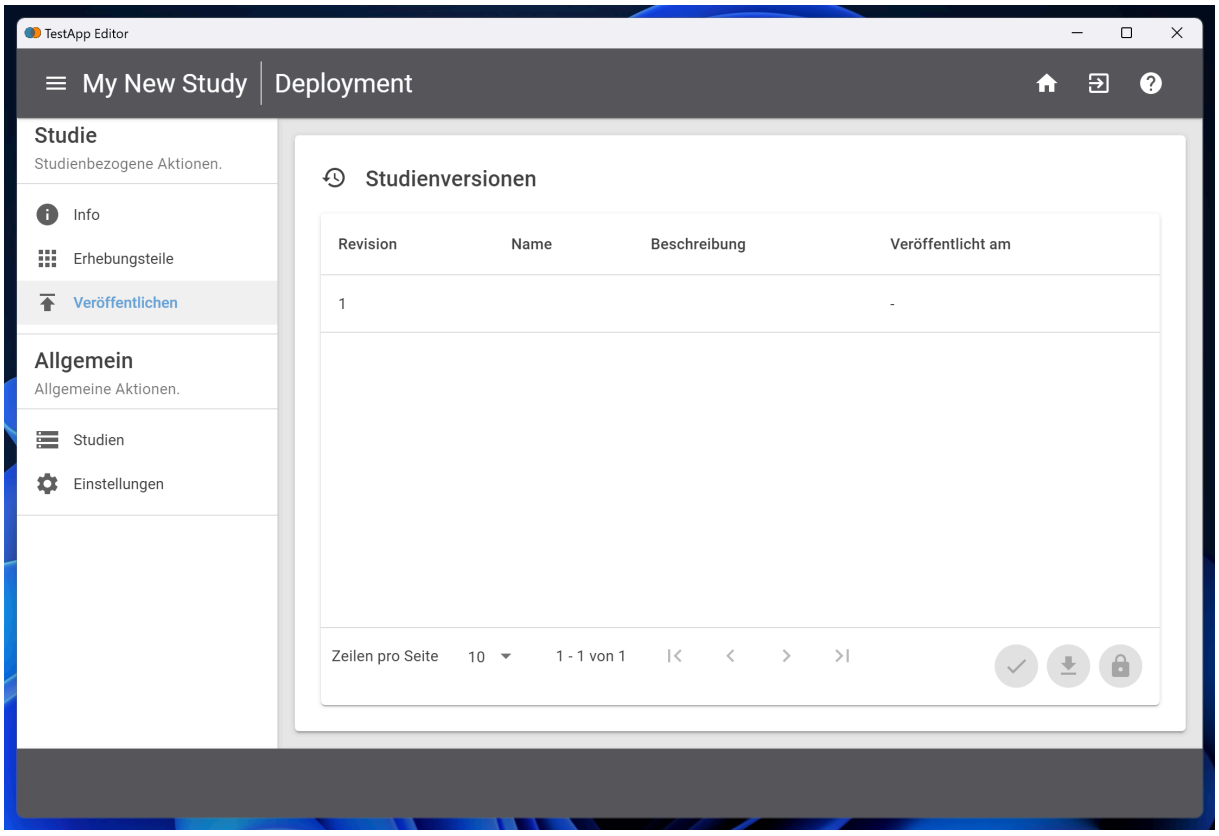
If problems are still detected in an exported study, it is possible to return to the preparation (see Preparation of studies and Preparation of survey parts), modify the study definition and the configuration of the test parts, create another sealed version and continue with the modified configuration.

7 Datenerhebung: Veröffentlichen & Exportieren / Data Collection: Publish & Export

The configuration of *Studies* and the *Survey Part(s)* contained therein is carried out in the *IRTlib Editor*. Changes are always saved within the *IRTlib Editor* when the disc symbol at the bottom right is clicked. During the preparation of a study, the changes are saved if they are to be applied. But once the preparation is completed, changes should no longer be possible or at least be tracked so that the version in the *IRTlib Editor* corresponds to the version in the *IRTlib Player*.

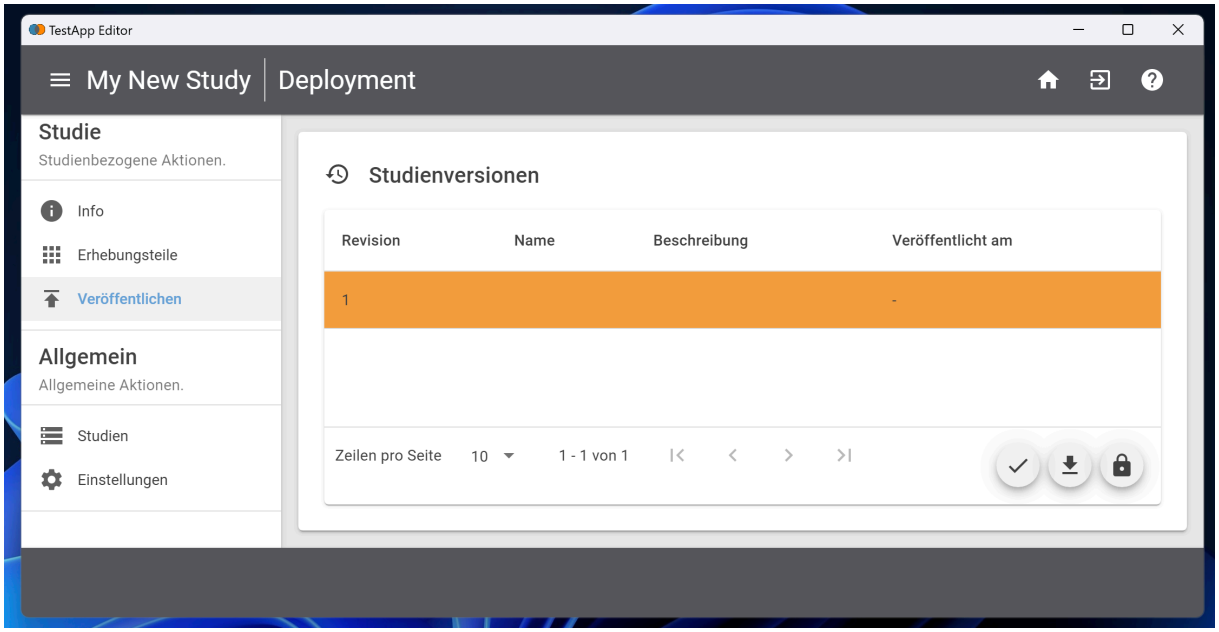
To support this process of using *IRTlib Editor* and *IRTlib Player*, the following concept has been implemented. To ensure that the configuration of a *Study* used for data collection can be clearly identified at all times, the configuration must be sealed before it is transferred to an *IRTlib Player*.

This is done in the *Publish* view of a study, in which the *Study versions* are listed. For a new *study*, this view initially looks like this:



In this state, you can make changes to the settings of the *Study* and all contained *Survey Parts*.

The buttons for *Validate*, *Download* and *Publish* studies are greyed out because no revision is selected. The buttons can be activated by clicking on the line with the unpublished *revision 1*:



Before you continue, please check that you have thought of everything. Use the following *checklist* to do this.

7.1 Checklist before publishing

- Is the *login* configured?

  To ensure that the correct *study* can be started after starting the *IRTlib Player*, a *login mode* suitable for the planned use must be configured. The *login mode* can be defined in the Study configuration in the *Login* section.

- Is a *test leader menu* configured?

  If kiosk mode is activated in the offline *IRTlib Player*, it may be difficult or impossible to exit the application without a configured *test conductor menu*. Key combinations and passwords with roles are defined in the Study configuration in the *Test leader menu* section.

- Are the *items* inserted?

  The assessment content is configured in one or more *survey-parts*. Most *CBA ItemBuilder-tasks* will be located in the *Items* section of a *survey part*.

- Are the runtime environments (*Runtimes*) available?

  Runtimes* are configured in the Settings.

Once you have checked this checklist, you can continue as described in the next section.

7.2 Publish & Export

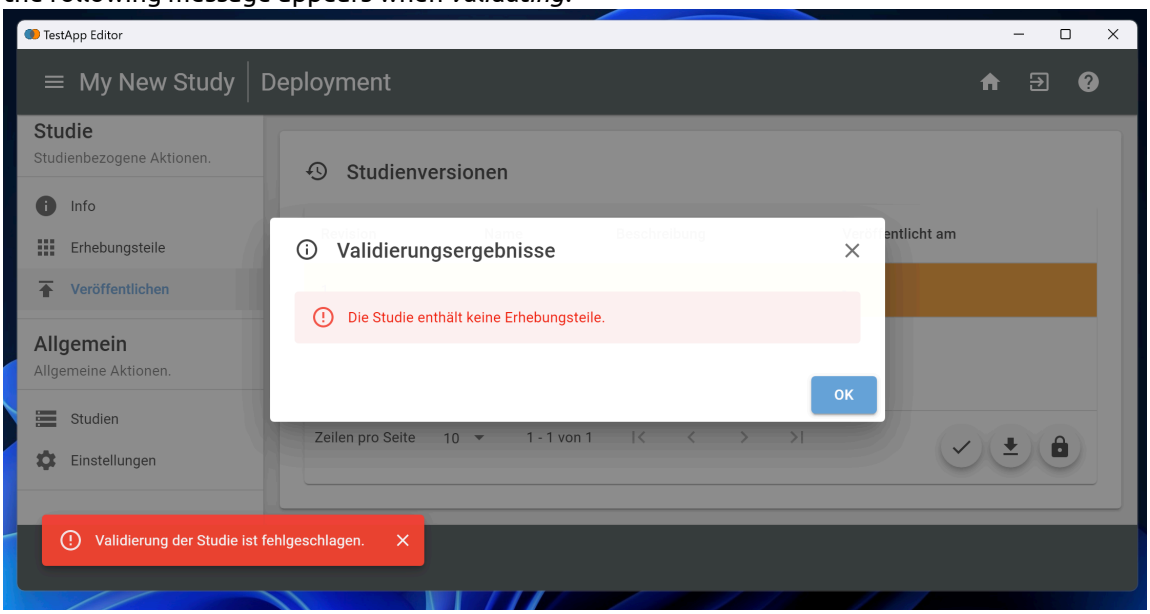
The process to *validate*, *publish* and *download* study configurations is described in the embedded help:

Embedded programme help

7.2.1 Publish

Before a *Version* of a *Study* is published, the *Validate* button can be used to check whether the *Study* has been configured correctly.

For example, a *Study* must always contain at least one *Survey Part* definition If this is not the case, the following message appears when *validating*:



Note: If no errors are found during *Validation*, no further message is displayed and the study can be published.
If there are no more errors and the *Study* is to be prepared for export, a *Version* can be created. The following button is used for this purpose:

The following dialogue then appears:

Data can be collected with an *IRTlib Player* if a published *study* has been downloaded from the *IRTlib Editor* as a ZIP archive.

8 Datenerhebung: In IRTlib Player Importieren / Data Collection: Import into IRTlib Player

8.1 Import Configuration

The following describes how to use a study configuration created with an *IRTlib Editor* that is available as a ZIP archive.

> **Published version required**
>
> A published version of a *Study* is required for data collection with an *IRTlib Player*.

If a *sealed* study configuration has been exported from the *IRTlib Editor*, it can be integrated into an *IRTlib Player*.

Two options are currently supported:

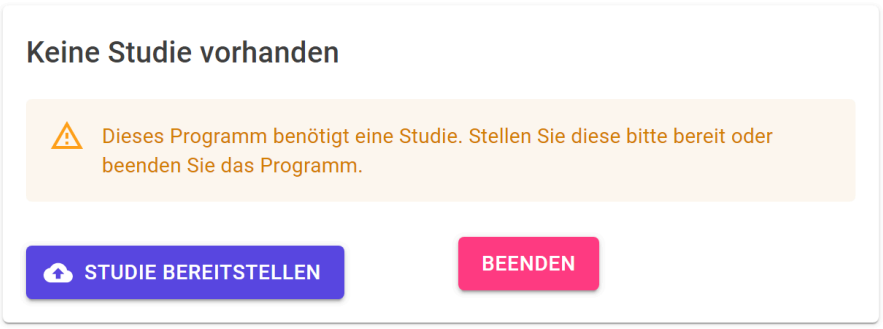
- Automatic import from ZIP archive
- Manual import from ZIP archive

The *automatic import* is only possible for the first *Study* in an *IRTlib Player*. If several *Studies* are to be used in parallel in an *IRTlib Player*, a *manual import* must be configured.

8.1.1 Automatic Import

For an *automatic import* of a *Study* available as a ZIP archive in an offline *IRTlib Player*, the player can first be started via the executable file `TestApp.Player.Desktop.exe`.

If this *IRTlib Player* has not yet been configured with a *Study* (i.e. the player has been downloaded directly from the Github repository as described under Download, for example), the following dialogue appears:

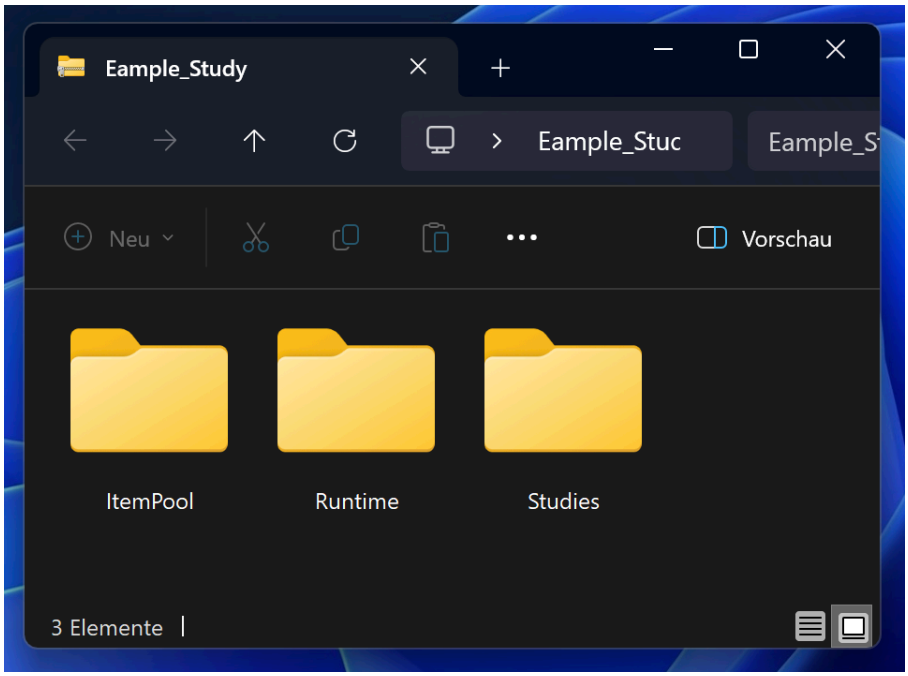


The ZIP archive can be opened directly by clicking the *Deploy Study* button. It is then automatically inserted into the *IRTlib Player* and can be used in the way configured in the *Login* section of the Study.

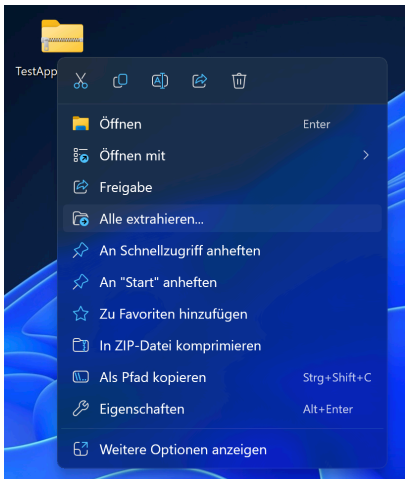
8.1.2 Manual Import

If no automatic import is possible or desired, the contents of the ZIP archive of a *Study* can also be integrated manually into the directory of the *IRTlib Player* provided for this purpose.

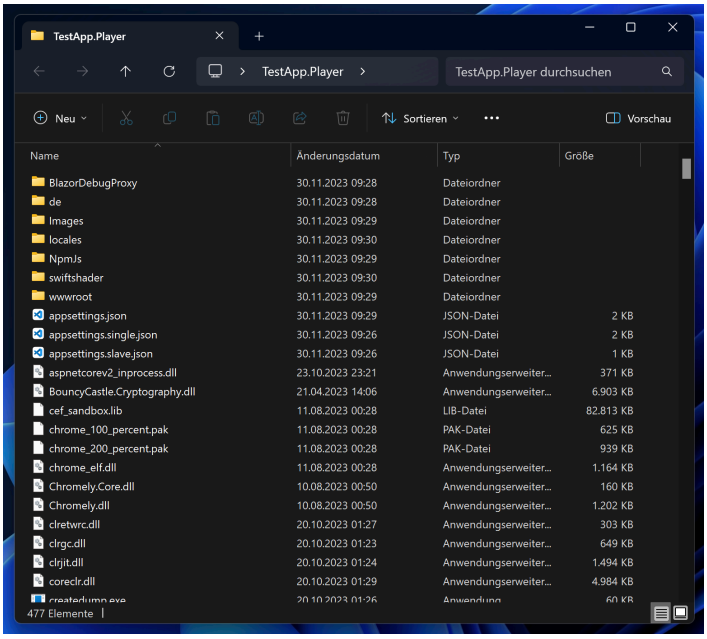
Each ZIP archive with a *study configuration* exported from the *IRTlib Editor* contains three directories. The ZIP archives `{StudyName.zip}` can be opened with the *Windows Explorer*, for example:



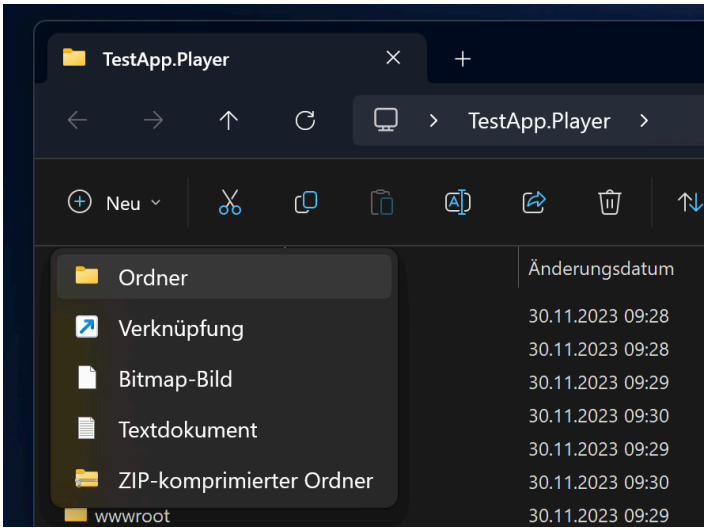
To integrate the *Study* into an *IRTlib Player*, the contents of these three directories can now be integrated into the programme directory of an offline *IRTlib Player*, for example.

Step	Description
1.	Unzip the player (`TestApp.Player.zip`). This can be done using *Windows Explorer*, for example: 
2.	Navigate to the unzipped directory:

Step	Description

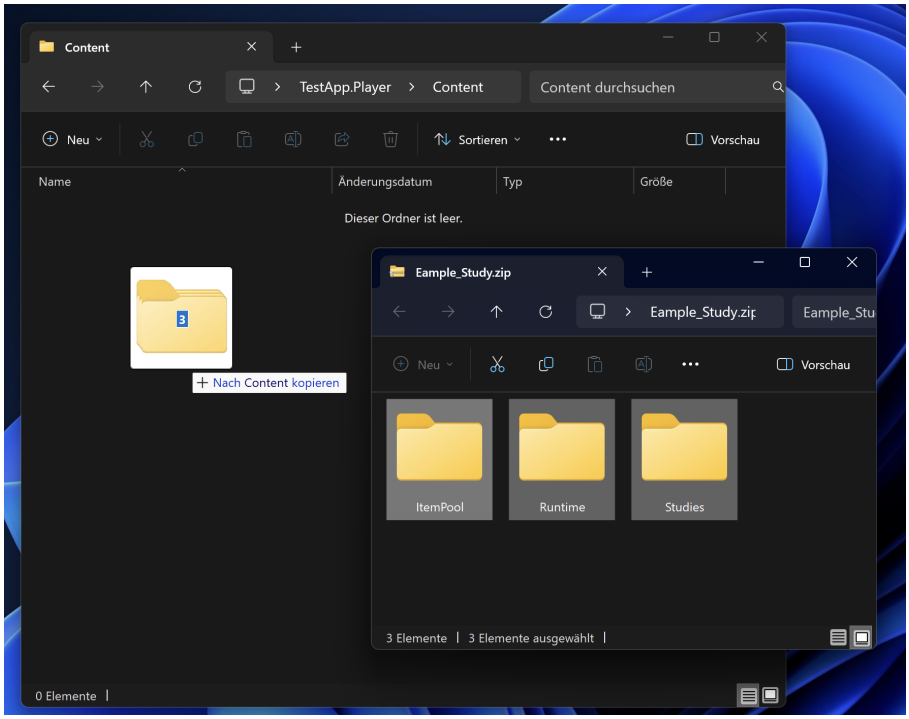


3. Create a new folder `Content` within the player folder (i.e. `TestApp.Player/Content/`). If a *Study* is already configured, then the `Content` folder already exists.

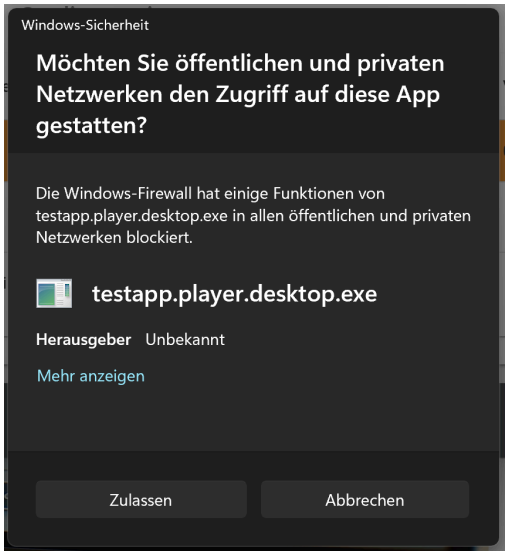


4. Copy the three folders `ItemPool`, `Runtime` and `Studies` from the downloaded study `{StudyName.zip}` into the `Content` folder of the player.

Step	Description



5. Start the file `TestApp.Player.Desktop.exe`
6. If necessary, accept the following warning:



The import of *studies* into an online *IRTlib Player* is analogous to the procedure described here. For this, access to the *volume* `/app/Content` defined in the `docker-compse.yml` file is required for preparation.

8.2 Configure deliveries

The study configurations created with an *IRTlib Editor* can be used with different variants of the *IRTlib Player*.

Three versions are currently available:

- Desktop version (Windows)
- Local server version (Windows)

- Online version (Docker)

8.2.1 Desktop version (Windows)

Basic configurations of the *IRTlib Player* (display in windowed mode vs. full screen mode) are part of the study configuration. Only studies with the same display settings can be used simultaneously in an offline *IRTlib Player*. Multiple copies of an offline *IRTlib Player* on one computer are possible.

To start the (offline) *IRTlib Player* on a computer, the executable file `TestApp.Player.Desktop.exe` must be started.

Data Storage: The data collected during a data collection with the offline *IRTlib Player* is stored locally in a directory. The directory and the file name for the raw data storage are configured in the *Study* and can be customised using startup parameters. The user name or the UUID created is used as the file name of the *raw data archive*. If the user name is used more than once in an (offline) *IRTlib Player*, i.e. if a *raw data archive* with this file name already exists when the *IRTlib Player* is closed, it is not overwritten but a suffix is added (e.g. `PersonIdentifier_1.zip`).

Startup Parameter: The integration of the offline *IRTlib Player* into programmed processes is possible. Login data (user name, user name + password, token) that are configured in a *Study* can be transferred as so-called *startup parameters*. These parameters are then appended to the call of `TestApp.Player.Desktop.exe`.

Example:

```
TestApp.Player.Desktop.exe /RawDataFolder="..\\myDataFolder"
```

Valid startup parameters are:

- `/AutoLoginCreateWithTest="{StudyName}"`: Requests the administration of the study named `{StudyName}`.
- `/AutoLoginUserName="{PersonIdentifier}"`: Passes the login information `{PersonIdentifier}` as the user name.
- `/MyBlocklyVariable="123"`: Passes the value `123` for the *Blockly* variable `MyBlocklyVariable`.
- `/MonitoringFile="..\\last-run.json"`: Path and file name of the so-called monitoring file.
- `/RawDataFolder="..\\myDataFolder"`: Path to the directory in which the *raw data archives* are saved.

Several startup parameters in succession are possible (separated by spaces).

Checking Prerequisites: The *IRTlib Player* should be able to run on Windows computers (currently supported from *Windows 10*) without further installation or runtime requirements. However, special configurations, virus scanners etc. may prevent it from running. A tool for checking prerequisites can be created, for example, based on this example (IRTlibReadiness).

> **!** Important note
>
> The provided Windows version of the *IRTlib Player* enables a simple *kiosk* mode, which is only intended for computers with only one screen (e.g. notebooks). For an exam-safe kiosk solution, the offline version of the *IRTlib Player* can be combined as a *local server* with additional software (such as the Safe Exam Browser).

8.2.2 Local server (Windows)

The ZIP archives of the offline *IRTlib Player* available via the Github repository in the *Releases* section also contain a *local server version* parallel to the application with integrated browser (`TestApp.Player.Desktop.exe`), which can be started via the executable file `TestApp.Player.Server.exe`.

> **!** Important note
>
> The version `TestApp.Player.Server.exe` is intended for offline operation in bring-in networks, e.g., when WLAN routers and server notebooks are brought into schools. This version is not intended for online use (for which the Docker version is provided).

8.2.3 Online version (Docker)

Integration: To carry out data collection, the *Docker* image of the *IRTlib Player* should only be accessible via https-secured connections. This can be realised, for example, using an additional *nginx* configured as a *reverse proxy*.

Access to Editor: Unauthorised persons who are not involved in study preparation must not be able to access the *IRTlib Editor*. The Github repository provides *IRTlib Editor* and *IRTlib Player*. For operational data collection, it is not necessary to run the *IRTlib Editor* online, as the study preparation can also be created with the offline version of the *IRTlib Editor*. If the *IRTlib Editor* is hosted online, it must be protected against unauthorised access.

Access to directories: The item contents that are configured for an assessment are stored in the *volumes* `/app/Content` (*IRTlib Player*) and `/app/data` (*IRTlib Editor*) defined in the `docker-compse.yml` file. To ensure the protection of instruments, unauthorised persons must not be able to access these *volumes*.

Data storage: The data collected during a data collection with the online *IRTlib Player* is stored in the *volume* `app/result`. They can be retrieved from there as directories (one directory per *session*) or as *raw data archives* via an *API* (if an *API* key is defined).

> **⚠** Note
>
> When using *Docker* containers, assessment content and data can be accessed online. Assessment content is only protected via the *login mode* defined in the study configuration. Personal data and assessment content may also be accessible if an *API* key is defined.

Configuration of the player: The file `appsettings.json`, which is contained in the `TestApp.Player` directory, is central to the technical configuration of the player. Three different *API keys* (i.e. access keys) can be stored in this file by storing them in the following `JSON` structure before the *Docker* container is started:

```
"API": {
   "ExternalExportKey": "",
   "DevelopmentKey": "",
   "LoginManagementKeys": []
 }
```

The *API keys* have the following functions:

- `ExternalExportKey`: This *key* is used to gain access to the data collected with the *IRTlib Player*. The data can be accessed via the R package LogFSM, for example, as described in the section *Data retrieval*.

Routes for direct access

The list of the processed session, i.e. the *session identifiers*, can be retrieved as `JSON` with an *API key* for `ExternalExportKey` via the following call:
`https://{U}/{S}/api/session/?apiKey={K}`

- `{U}` is the URL of the *IRTlib Player*
- `{S}` is the identifier of the *study*
- `{K}` is the `ExternalExportKey` as defined in the `appsettings.json`

With a known *Session-Identifier* the raw data can then be retrieved via the following call with an *API-Key* for `ExternalExportKey`:
`https://{U}/{S}/api/session/{ID}/result?apiKey={K}`

- `{ID}` is the *Session-Identifier* (e.g. the user name, depending on the configuration of the login)

- `DevelopmentKey`: This *API-Key* is intended for customising study configurations in a running player.

Under Development

This function is currently under development.

- `LoginManagementKeys`: This list of *API-Keys* is intended for customising login data (accounts) in a running player.

Under Development

This function is currently under development.

Monitoring: (A method for monitoring *Docker* containers is under development).

8.3 Testing and Releasing Deliveries

With the integration of a *study configuration* into an *IRTlib Player*, the preparation is not yet complete. Before data collection can be started with the *IRTlib software*, the following tests should be conducted:

(Tests within the CBA ItemBuilder-Preview): Before configuring a *study* and a *survey* section with *CBA ItemBuilder-tasks*, it should already have been tested in terms of presentation, functionality and scoring in the *Preview* of the *CBA ItemBuilder*.

Functional tests: Especially if *CBA ItemBuilder-tasks* interact with the delivery platform (such as *login* items), functional tests (i.e. tests of specific functions) should be carried out in the delivery software. This also applies to navigation between items and, of course, the flow control configured in the delivery environment.

Cross-browser testing: If studies are not conducted with the offline *IRTlib Player* (which comes with its own browser) and especially if newly or specifically programmed *JavaScript/HTML5* content is used within the *CBA ItemBuilder-Tasks* via so-called *ExternalPageFrames (i.e. `iframes`), testing should be carried out in different browsers.

Performance tests: If large media files (videos, audio files) are included in the *CBA ItemBuilder-Tasks*, it may be advisable to test the feasibility of the assessment even under unfavourable network conditions (e.g. low bandwidth, long latencies, connection drops, etc.).

Load tests: If a large number of test participants are to be tested in parallel (online), it may be advisable to coordinate the load behaviour of the delivery (and, for example, the resources available for the *Docker* container) in advance.

Data storage check: In any case, the fit of the scoring definition within the *CBA ItemBuilder-Tasks* and the configuration in the *IRTlib Editor* should be checked and a data storage check made. This means that synthetic click patterns (i.e. responses) are entered *before* the start of the survey and compared with the responses stored in the data set. In order to be able to easily recognise input errors during the subsequent check, it has proven useful to record screen videos in parallel for the data storage check.

Smoke test: The final form of testing is a run-through test in the fully configured setting from *Study* in an *IRTlib Player*. The survey should be displayed correctly and a readable *raw data archive* should be created.

8.3.1 Suggested Test Plans

> **Note**
>
> The provision of the free research software *IRTlib Editor* and *IRTlib Player* is without guarantee and no liability can be accepted for missing data, data loss or compromised data etc.

Generally valid recommendations for (absolutely) necessary tests are difficult to formulate, the following table is therefore to be understood as a non-binding recommendation, which must be weighed up in each specific case.

Test	Recommendation	Condition
CBA ItemBuilder-Preview	Always	(Errors in presentation, behaviour and scoring that can already be found during item creation should be systematically tested and excluded before creating a study configuration).
Functional tests	If required	Only if *IRTlib Player* and *CBA ItemBuilder-Tasks* have to interact and in relation to functionality defined in the *IRTlib Editor* (e.g. response-dependent branching).
Cross-browser testing	If required	Only if *IRTlib Player* is used online and the browsers do not correspond to the browsers already used for the *CBA ItemBuilder-Preview*.
Performance tests	If required	If large multimedia parts are included or if a poor network connection is to be expected.
Load tests	If required	Only if online *IRTlib Player* is to be administered with many parallel tests.
Data storage check	Always	Check all data (incl. log data if this is required for evaluation).
Smoke test	Always	Smoke test for every version, especially to rule out accidental configuration errors at the *last minute*.

8.3.2 Carry out Data Collections

Once the *Study* has been configured and tested in an *IRTlib Player*, data can be collected. The longer the field time, the more important it is to regularly back up the collected data or to regularly remove the collected raw data archives from the survey hardware.

9 Datenerhebung: Datenaufbereitung / Data Collection: Data Post-Processing

9.1 Data Preparation

Data is saved by the *IRTlib Player* in *raw data archives* per *session* (i.e. per test run with a *Study*). The *raw data archives* are ZIP archives whose file names correspond to the *user name* or the *Universally Unique Identifier* (UUID). Deviations from this scheme are possible if a *raw data archive* with this file name already existed at the time of saving. In this case, the data is not overwritten by the *IRTlib Player*, but a suffix `_1`, `_2`, … is appended until the file name can be used.

- **Offline IRTlib Player**: If not configured otherwise, the results data are saved in the directory `Temp/{Study-Name}/Results`. The *raw data archives* are created when a *session* is ended, i.e. the last defined *CBA ItemBuilder-Task* is exited with `NEXT_TASK`. It is no longer possible to *continue* the *session* that has been started, as may be necessary for instance in the event of a computer crash, after the *raw data archives* have been created.

  The same applies if the offline version of the *IRTlib Player* is used as a local server. The *raw data archives* are saved in the `Temp/{Study name}/Results` directory after test processing.

  The collection of data from the offline *IRTlib Players* corresponds to the collection of the *raw data archives* that are collected on the various devices.

> **Note**
>
> As the offline *IRTlib Players* are not connected to each other, identical login data can be created in parallel in different *IRTlib Players*, depending on the *login mode*. After data collection, the *raw data archives* must therefore be merged with care and, if necessary, separated by subfolders.

- **Online IRTlib Player**: Unless configured otherwise, the online player collects the data in the *volume* that is configured for the results data (see `/app/results` in `docker-compse.yml` file). Each *session* is stored there in a separate subdirectory and can be downloaded by administrators who have access to the *volume* (!).

If an *API*-key is defined for data access, the *download* of the result data can also be carried out via the R package LogFSM.

9.1.1 Data retrieval with `LogFSM`

To do this, the R package can first be installed (once) with the following call:

```
source("http://logfsm.com/latest")
```

The *raw data archives* can then be downloaded using the following R script:

```
library(LogFSM)

if (!dir.exists(paste0(getwd(),"/in/")))
  dir.create(paste0(getwd(),"/in/"))

if (!dir.exists(paste0(getwd(),"/out/")))
  dir.create(paste0(getwd(),"/out/"))

SECRET_KEY <- "(your secret key)"
API_URL <- "(your API-URL)"

LogFSM::TransformToUniversalLogFormat(inputfolders = paste0(getwd(),"/in/"),
                                      inputformat = "irtlibv01a",
                                      zcsvoutput = paste0(getwd(),"/out/data_csv.zip"),
                                      stataoutput = paste0(getwd(),"/out/data_dta.zip"),
                                      spssoutput = paste0(getwd(),"/out/data_sav.zip"),
                                      key = SECRET_KEY,
                                      web = API_URL,
                                      outputtimestampformatstring="dd.MM.yyyy HH:mm:ss.fff")

results <- read.csv(unz(paste0(getwd(),"/out/data_csv.zip"), "Results.csv"),
                    sep=";", encoding = "UTF-8")
```

Data retrieval and conversion of the data with LogFSM

By calling the function `TransformToUniversalLogFormat` from the package `LogFSM`, the data is downloaded and stored in the specified directory `infolders` if an API key (`key`) and an API url (`web`) are passed.

> **Note on `SECRET_KEY` and `API_URL`**
>
> The value for `SECRET_KEY` must correspond to the entry that was defined as `ExternalExportKey` in the `appsettings.json` when configuring the *Docker* image, see section Online-Version (Docker).
> The value for the `API_URL` is formed according to the following scheme: `https://{U}/{S}/api/session/`
>
> - `{U}` is the URL of the *IRTlib Player*
> - `{S}` is the identifier of the *study*

The function `TransformToUniversalLogFormat` from the package `LogFSM` (or analogue to the *command line tool* described below) can also be used to read out already existing local raw data archives.

9.1.2 Data Retrieval via the Command Line

The application `TransformToUniversalLogFormat` used for data retrieval and data conversion via `LogFSM` is available as a console application from the *Releases* section of https://github.com/kroehne/LogFSM/.

Data retrieval and data transformation can also be performed without R.

> **In development**
>
> A certified version of `TransformToUniversalLogFormat` for Apple is currently under development.

9.1.3 Result Data

If the data was retrieved via `LogFSM` from an online *IRTlib Player* or collected offline, it is stored in a directory at the end. Per *session* (i.e. per person or person x time) as a *raw data* archive.

The function `TransformToUniversalLogFormat` in `LogFSM` or via the command line can also be used to read the *raw data archives* from a directory and extract the result data:

```
library(LogFSM)

if (!dir.exists(paste0(getwd(),"/out/")))
  dir.create(paste0(getwd(),"/out/"))

LogFSM::TransformToUniversalLogFormat(inputfolders = paste0(getwd(),"/in/"),
                                      inputformat = "irtlibv01a",
                                      zcsvoutput = paste0(getwd(),"/out/data_csv.zip"),
                                      stataoutput = paste0(getwd(),"/out/data_dta.zip"),
                                      spssoutput = paste0(getwd(),"/out/data_sav.zip"),
                                      outputtimestampformatstring="dd.MM.yyyy HH:mm:ss.fff")

results <- read.csv(unz(paste0(getwd(),"/out/data_csv.zip"), "Results.csv"),
                    sep=";", encoding = "UTF-8")
```

9.1.4 Log Data

Converting the data with `TransformToUniversalLogFormat` in `LogFSM` or via the command line converts the collected log data, which is provided by the *CBA ItemBuilder-Tasks*, into the following formats:

- *Flat and Sparse Log-Data Table*: A large table (as CSV, Stata, SPSS) with one row per event. As the event-specific attributes (i.e. the various additional information available from an event) are distributed across many columns, which are only filled for each *event type*, this table is *flat*, but can also be very *holey*.
- *Universal log format*: Alternatively, the ZIP archives created by `LogFSM` or the command line tool `TransformToUniversalLogFormat` also contain individual data record tables for each *event type*. The event-specific attributes in these tables are less *holey* (i.e. they only contain missing values for optional attributes) and can be combined into a *Flat and Sparse Log-Data Table* if required.
- *XES (eXtensible Event Stream)*: The log data can also be converted to the standardised `XML` format (https://xes-standard.org/).

> **Note on timestamps**
>
> The timestamps collected with the *IRTlib software* are in *UTC* format (*Coordinated Universal Time*).

9.1.5 Files in the raw data archives

The *raw data archives* contain the following files:

- `Trace.json`: Log data (*Traces*) as supplied by the *CBA ItemBuilder-Runtime*, together with the context from the *IRTlib Player*.

The file contains the following structure, separated by commas. The file is not a valid JSON until the last comma is removed and a `[` before and a `]` after the content is inserted.

The entry `Trace` contains the log data (*Traces*) in packets (as supplied by the *CBA ItemBuilder-Runtime*) quoted (i.e. `"` is displayed as `\u0022`). The `TraceId` is a counter which counts the transmitted packets. Timestamp is the timestamp of the transmission. SessionId is the user name or the UUID (*PersonIdentifier*). The `Context` provides a reference to the assessment content (*Element*) via the name of the *CBA ItemBuilder* project, *Task* and *Scope*. The information on the *IRTlib Player* used is stored under `Assemblies` and `StudyRevision` refers to the *Revision* of a (published) *Study*.

```
{
    "Trace": "(TRACE-JSON)",
    "TraceId": 1,
    "Timestamp": "2023-12-04T20:53:06.297Z",
    "SessionId": "(SESSION-ID OR USERNAME)",
    "Context": {
        "Item": "(PROJECT NAME)",
        "Task": "(TASK NAME)",
        "Scope": "(SCOPE)",
        "Preview": ""
    },
    "Assemblies": [
        {
            "Name": "TestApp.Player.Desktop",
            "Version": "(APPLICATION VERSION)",
            "GitHash": "(APPLICATION BUILD HASH)"
        }
    ],
    "StudyRevision": "(STUDY REVISION)"
},
```

- `Snapshot.json`: Snapshot data as supplied by the *CBA ItemBuilder-Runtime*, together with the context from the *IRTlib Player*.

  The file contains the following structure, separated by commas. The file is not a valid JSON until the last comma is removed and a `[` before and a `]` after the content is inserted.

  The `Snapshot` entry contains the snapshot information (as supplied by the *CBA ItemBuilder-Runtime*) quoted (i.e. `"` is displayed as `\u0022`). The `ContextFlag` indicates how the *CBA ItemBuilder-Task* was exited (NextTask, PreviousTask or Cancel). `Timestamp` is the timestamp of the transmission. `SessionId` is the user name or the UUID (*PersonIdentifier*). The `Context` provides a reference to the assessment content (*Element*) via the name of the *CBA ItemBuilder* project, *Task* and *Scope*. The information on the *IRTlib Player* used is stored under `Assemblies` and `StudyRevision` refers to the *Revision* of a (published) *Study*.

```
{
    "Snapshot": "(SNAPSHOT-JSON)",
    "ContextFlag": "NextTask",
    "ContextScope": 0,
    "Timestamp": "2023-12-04T20:53:06.497Z",
    "SessionId": "(SESSION-ID OR USERNAME)",
    "Context": {
        "Item": "(PROJECT NAME)",
        "Task": "(TASK NAME)",
```

```
        "Scope": "(SCOPE)",
        "Preview": ""
    },
    "Assemblies": null,
    "StudyRevision": null
},
```

- `ItemScore.json`: Scoring information (as supplied by *CBA ItemBuilder-Runtime*).

  The file contains the following structure, separated by a comma. The file is not a valid JSON until the last comma is removed and a `[` before and a `]` after the content is inserted.

  The `ItemScore` entry contains the *ItemScore* (as supplied by the *CBA ItemBuilder-Runtime*) quoted (i.e. " is displayed as \u0022).The `ContextFlag` specifies how the *CBA ItemBuilder-Task* was exited (NextTask, PreviousTask or Cancel). `Timestamp` is the timestamp of the transmission. `SessionId` is the user name or the UUID (*PersonIdentifier*). The `Context` provides a reference to the assessment content (*Element*) via the name of the *CBA ItemBuilder* project, *Task* and *Scope*. The information on the *IRTlib Player* used is stored under `Assemblies` and `StudyRevision` refers to the *Revision* of a (published) *Study*.

```
{
    "ItemScore": "(SCORING-JSON)",
    "ContextFlag": "NextTask",
    "ContextScope": 0,
    "Timestamp": "2023-12-04T20:53:06.474Z",
    "SessionId": "(SESSION-ID OR USERNAME)",
    "Context": {
        "Item": "(PROJECT NAME)",
        "Task": "(TASK NAME)",
        "Scope": "(SCOPE)",
        "Preview": ""
    },
    "Assemblies": [
        {
            "Name": "TestApp.Player.Desktop",
            "Version": "(APPLICATION VERSION)",
            "GitHash": "(APPLICATION BUILD HASH)"
        }
    ],
    "StudyRevision": "(STUDY REVISION)"
},
```

- `Session.json`: The file contains data of the *IRTlib Player*, which describe the execution of the *Session*.
- `Log.json`: Log events of the *IRTlib Player* (contains log information for processing the *Blockly* routing).
- `browser.log`: Console output collected during the processing of tasks in the browser (unstructured text, for developers).
- `server.log`: Log output from the server of the *IRTlib Player* (unstructured text, for developers)
- `Keyboard.json`: Keyboard input and timestamps.
- `Monitoring.json`: Copy of the monitoring file that was created.

Part III

Allgemein / General

10 Einstellungen / Settings

The *IRTlib Editor* has a small number of settings.The language can be set to German or English.

10.1 Overview

The *IRTlib software* is currently still under development. Information about the current version (and for *Preview* versions about the build hash) can be found in the section *About the Program*.

Embedded Help

10.1.1 Settings

In this section, settings can be made that affect working with the *IRTlib Editor* and all studies.

10.1.1.1 Runtime Management

To configure studies that use CBA ItemBuilder content with the IRTLib Editor, the appropriate runtime environment (*Runtime*) is required for each version. Current tested versions of the CBA ItemBuilder runtime are already stored in the Editor, but runtimes for other versions of the CBA ItemBuilder or updated or corrected runtimes can also be imported into the Editor in this section. Runtimes that are available in the Editor are automatically integrated as part of the study configuration when studies are published and are thus available to the *IRTLib Player*.

10.1.1.2 General Settings

Change the language for the editor in this section. The setting selected here has no influence on the language of the assessment content in the configured studies.

10.1.2 About the Programme

Under the **Version info** button, you will find a summary of the latest changes and information on the current programme version.

10.2 Runtimes

The *IRTlib Software* can be used with CBA ItemBuilder tasks of different CBA ItemBuilder versions. The required **Runtime** (i.e., the connection between the CBA ItemBuilder tasks and the *IRTlib Software*) is part of the study configuration so that the *IRTlib Player* knows for sure how to use CBA ItemBuilder tasks of a particular version.

Embedded Help

10.2.1 Runtimes

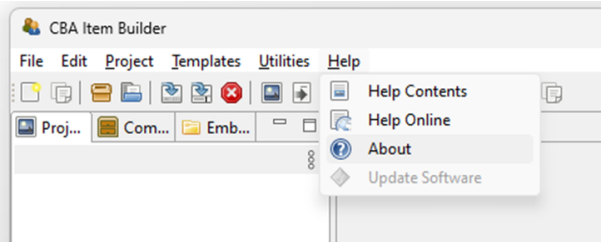
To configure *Studies* that use *CBA ItemBuilder* content with the *IRTLib Editor*, the appropriate runtime environment (*Runtime*) is required for each version. Current tested versions of the *CBA Item-*

Builder runtime are already stored in the *IRTLib Editor*, but *runtimes* for other versions of the *CBA ItemBuilder* or updated or corrected *runtimes* can also be imported into the *IRTLib Editor* in this section.

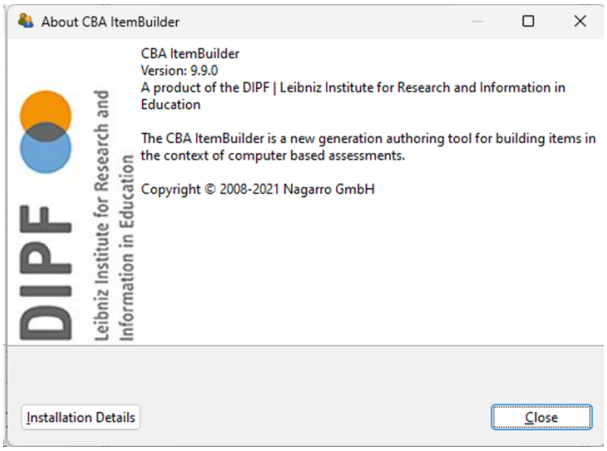
10.2.1.1 Check CBA ItemBuilder Version

It is important to know which version of the CBA ItemBuilder was used to create the items (i.e. the CBA ItemBuilder project files). If in doubt, this information can be found in the *About dialogue* of the CBA ItemBuilder, for example:

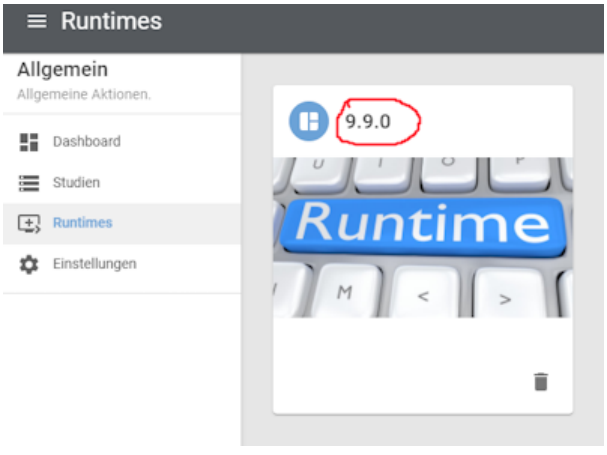
- Step 1: Open the “About” dialogue via the “Help” menu



- Step 2: Search for the version number in the dialogue (here `9.9.0`)



The version number must be listed as one of the cards in the *Settings* of the *IRTlib Editor* in the *Runtimes* section:



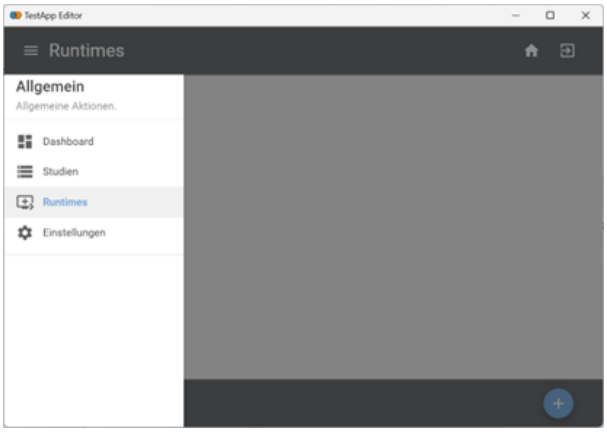
10.2.1.2 Import runtime files

If the corresponding runtime is not already included in the Editor, a new/additional *runtime* can be imported. Study configurations that are created/edited with the *IRTlib Editor* can contain several *Runtimes* for different versions.

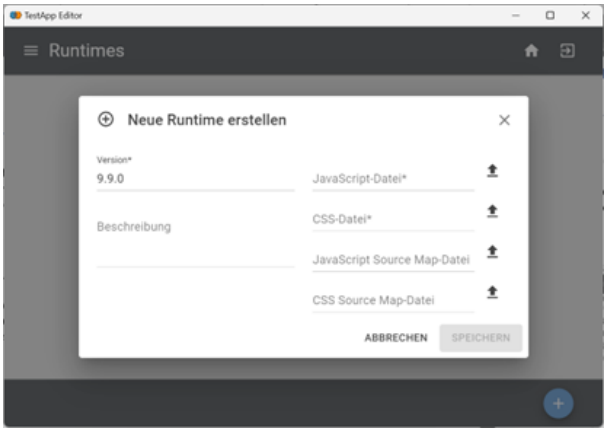
- Step 1: To integrate a runtime, a JavaScript and a CSS file are required. These files can be downloaded here:

https://cba.itembuilder.de/appendix-tables.html#previous-versions

- Step 2: Unzip the downloaded *Runtime* to be used.
- Step 3: Navigate to the *Runtimes* section:



- Step 4: Press the “+” button (bottom right)
- Step 5: Enter the version number with three digits (e.g. 9.9.0):

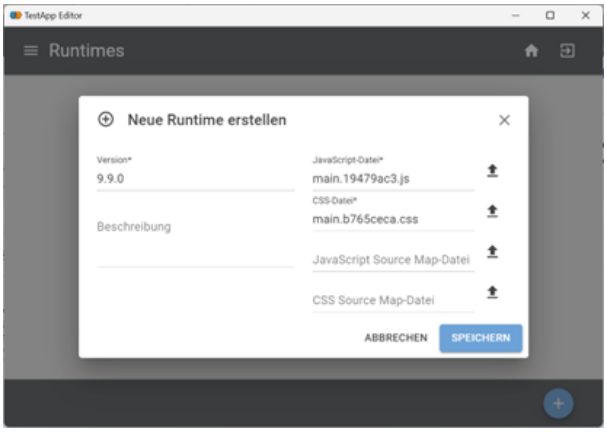


- Step 6: Select the file `main.*.js` from the ZIP archive containing the runtime environment. Note that the * corresponds to the hash of the file (i.e. the full file name looks like `main.19479ac3.js`)

- Step 7: Select the file `main.*.css` from the ZIP archive containing the runtime. Note that the `*` corresponds to the hash of the file (i.e. the full file name looks like `main.b765ceca.css`)

  Note: The *Description* field and the two additional *Map files* (for JavaScript Source and for CSS Source) are optional.

- Step 8: Press the *Save* button to finalise the import of the *Runtime*:



After the import, the supported CBA ItemBuilder versions are listed in the *Runtime* section. To delete a *Runtime* for a specific version, click on the *Trash* icon at the bottom right of the "Map" and confirm with *Delete*.

11 Github Repositorien / Github Repositories

11.1 IRTLib Software

The *IRTlib Software* is free *research software* in the sense of *Open Science*. It can be used for non-commercial applications.

Suggested Citation:

> Kroehne, U. (2023). IRTlib Documentation: Software for the administration and delivery of computer-based assessments [IRTlib Dokumentation: Software für die Verwaltung und Auslieferung computergestützter Assessments]. DIPF, Frankfurt am Main, Germany. https://doi.org/10.5281/zenodo.10441352

> **Note**
>
> Translation: If you want to help us translate this software, you can find more information here.

11.1.1 Download

- Current versions of the IRTlib software (Windows and Docker): GitHub
- Documentation: GitHub

11.2 CBA ItemBuilder

The *IRTlib software* allows the administration of assessment content created with the CBA ItemBuilder.

11.2.1 Download

- Current versions of the CBA ItemBuilder (Windows): https://www.itembuilder.de/software

11.2.2 Source Code

Source code and material for the CBA ItemBuilder are divided into several repositories:

- CBA ItemBuilder (desktop application): GitHub (In preparation / still private)
- Runtime environment: GitHub (In preparation / still private)
- Execution environment for developers: GitHub (In preparation / still private)
- Technical documentation: GitHub (In preparation / still private)
- Technical example items: GitHub(In preparation / still private)

11.2.3 Documentation

Online documentation

- HTML (interactive): https://cba.itembuilder.de

- PDF (static): Open-Assessments-with-CBA-ItemBuilder.pdf
- Sources GitHub (In preparation / still private)

Suggested Citation:

> Kroehne, U. (2023). Open Computer-based Assessment with the CBA ItemBuilder. DIPF, Frankfurt am Main, Germany. https://doi.org/10.5281/zenodo.10359757

12 Über / About

12.1 Acknowledgements

Contributors to this manual were:

- Maximilian Sattler
- Carla Burkart

> ! In development
>
> - Revision, language correction and translation of the manual are currently in progress…

12.2 Development

The development of the *IRTlib Software* takes place at Software-Driven.