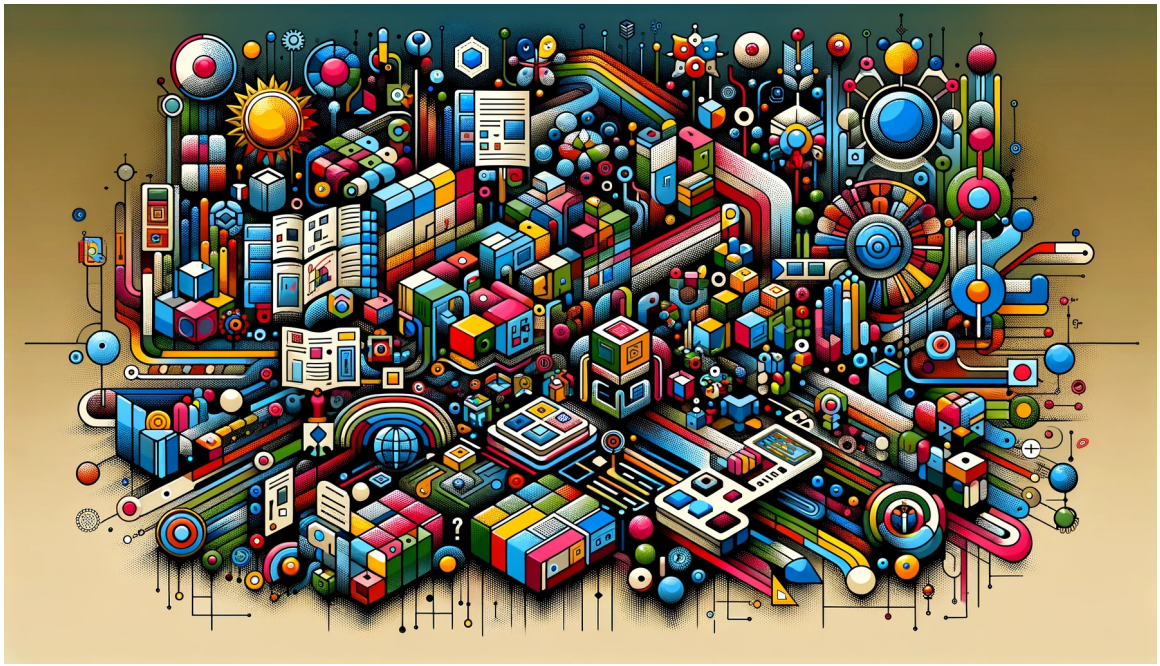


IRTlib Documentation

Software for the administration of computer-based assessments

Ulf Kroehne

Laste Change: 4th December, 2023

Table of contents

1 Einführung / Introduction

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

- *IRTLib Editor*: A software for test authors, which is used to configure *studies*.
- *IRTlib Player*: A *data collection* software program used by test-takers to complete tasks configured for a *study*.

Instructions for installing and setting up both programs for first use can be found at Download.

Before using the *IRTlib software* to configure and create deliveries, the assessment content can be created in the form of individual tasks using the CBA ItemBuilder.

The *CBA ItemBuilder* can be downloaded here: www.itembuilder.de/software

The interactive documentation of the *CBA ItemBuilder* is available here: cba.itembuilder.de

The development of the *CBA ItemBuilder* and the *IRTlib software* is coordinated by the Center 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 (Windows) and for online use (Docker Container).

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.

The two archives `TestApp.Editor.Desktop.zip` and `TestApp.Player.Desktop.zip` are available for download in the [Releases] section (https://github.com/DIPFtba/IRTlibDeploymentSoftware/releases).

> **Note**
>
> Note that the latest build can be found in the Preview section of the *Release* section of the repository.

> **Warning**
>
> The automatically generated preview versions of the *IRTlib Editor* and *IRTlib Player* are not signed. A warning message from the operating system must be accepted before the programs can be executed.

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 run 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 program `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 that acts 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 check out the repository on the target device (requires git) and run 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*.

See the section Data collection > Integration & delivery for more information on using the *Docker* containers.

Part I

Vorbereitung / Preparation

3 Vorbereitung Übersicht / Preparation Overview

Preparing a computer-based assessment based on CBA ItemBuilder items begins with using the *IRTlib Editor* to create a study configuration. This typically involves the following steps:

> Optional
>
> - **Pre-requisite**: Check the availability of the Runtime. The *IRTlib Editor* can be used to prepare assessments using CBA ItemBuilder *Tasks*. Using CBA ItemBuilder *Tasks* are stored in *Project Files* require a *Runtime* (i.e., the files "main..js" and "main..css") in the version corresponding exactly to the version of the CBA ItemBuilder that was used to generate the items (e.g., 9.9.0). Before using the IRTLib editor, ensure the required *Runtime* is included or import the runtime files (see here for details).
>
> Note: This step is usually not necessary for the use of CBA ItemBuilder items from version 9.9.

- **Create a new "Study"**: The *IRTlib Editor* is used to configure so-called *Studies*. The version of studies can be tracked in the editor, and studies can be published (i.e., sealed to be used for data collection). To start creating content using the *IRTlib Editor*, a study must be created first (see section Settings for details).

> Note
>
> Note that at least one study must be defined in the *IRTlib Editor*, before a study configuration can be used to collect data with an *IRTlib Player*.

- **Specify "Basic Configurations" for the Study (Info)**: Basic configurations that apply to the content of a prepared study include the study label and description, the login mode, the display configuration, the menu for test administrators and the specification how to continue after completing all content defined in a study (see section Studies for details).
- **Create a new "Survey-Part"**: Each study is composed of one or multiple survey parts (see here for details). Parts are considered segments of an assessment that are

administered together, like 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 (see Routing Across Survey Parts for details).

> **Note**
>
> Note that each study neads at least one test-part defined in the *IRTlib Editor*, before a study configuration can be used to collect data with an *IRTlib Player*.

- **Configure "Basic Settings" for Survey Part (Info)**: A study part of type "CBA ItemBuilder" is a set of CBA ItemBuilder projects that belong together. Each CBA ItemBuilder project requires at least one Task, but projects with multiple tasks are also supported. If items are administered with a common time limit, test parts allow for assigning tasks to a structure differentiating assessment content presented before a time-restricted section ("instructional items"), after a time-restricted section ("epilogue items"), and items in between with restricted time ("core items", see here).
- **Define "Core Items" (Items)**: To finalize the definition of a study-part, the CBA ItemBuilder tasks must be imported in the section "Items". By default, the sequence of CBA ItemBuilder tasks is assumed to be linear. However, when routing is activated for a study part, Blockly-based routing (see Routing Within Survey Parts for details) can be used to implement various test designs (e.g., multiple booklets, multi-stage testing, etc.).

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

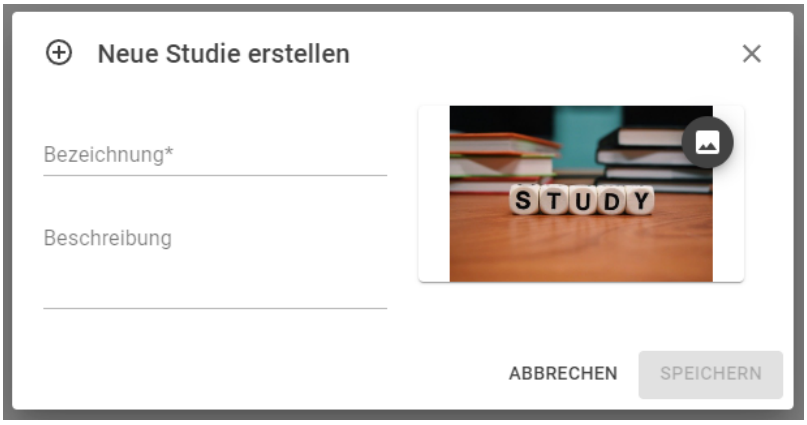
4.2 Studies

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.

4.2.1 How do I get started?

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

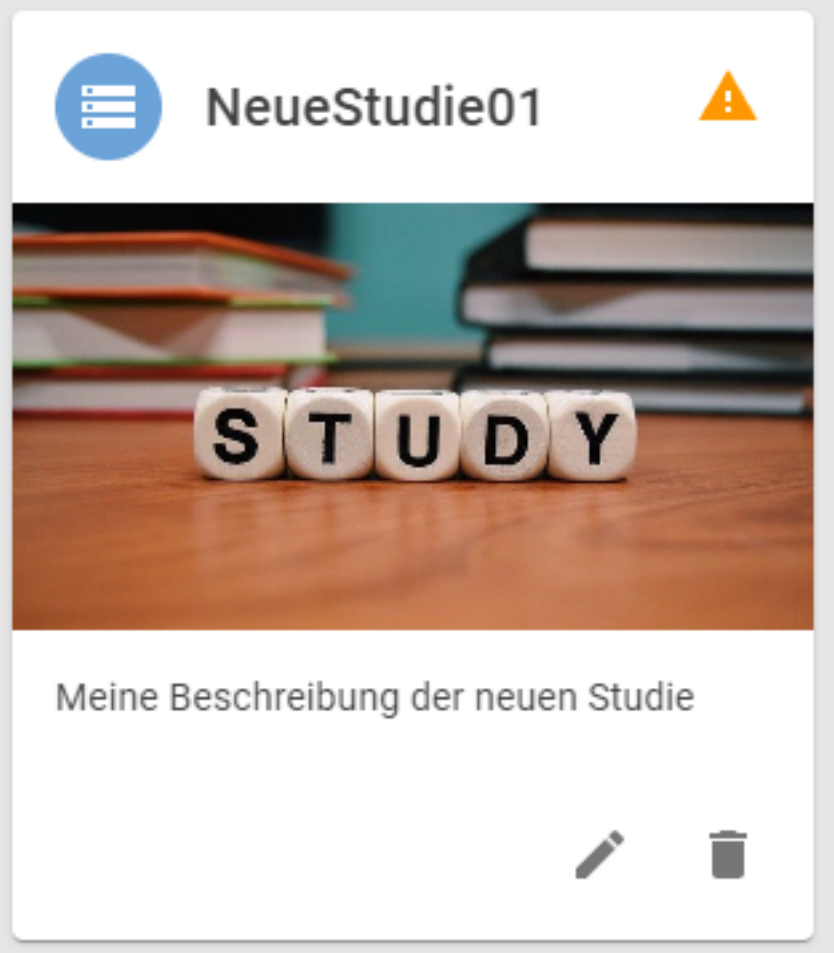
Then enter a *name* and optionally a *description* in the **Create new study** dialog.
Make sure that only letters (upper and lower case), numbers and a _ are allowed for the *name*.



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:

4.2.2 What's next?

Created studies are displayed as tiles in the study overview:

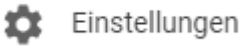


To continue with the creation and configuration of a study, click on the small edit icon:

4.3 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*.

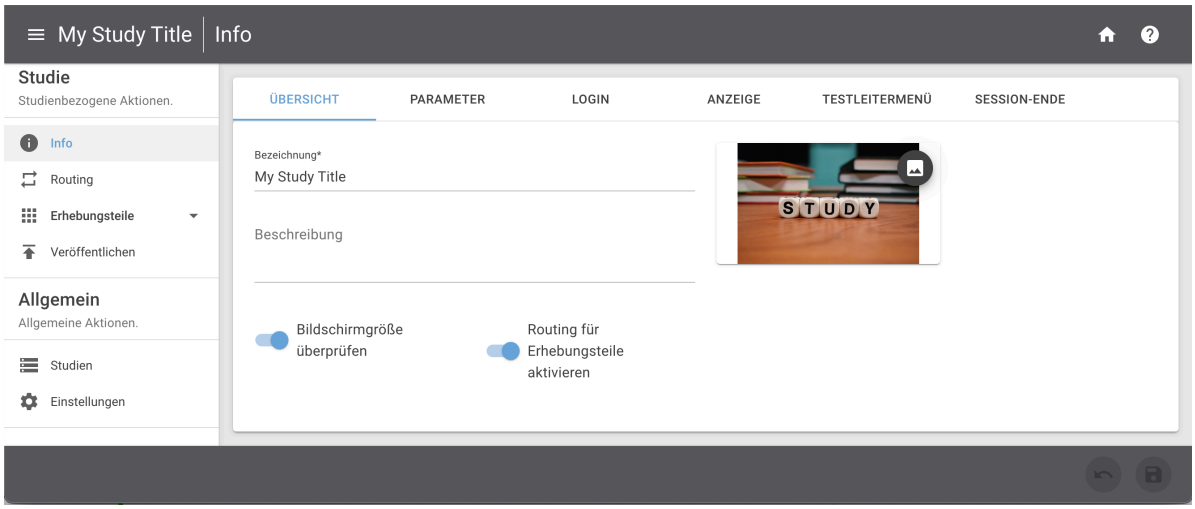


This item also gives you access to the *CBA ItemBuilder Runtimes* available in the *IRTLib Editor* (support for the use of *CBA ItemBuilder* content created with different versions of the program).

4.4 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

4.5 Study Configurations

- **Name**: What should the *study* be called? 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**: (description follows)

4.6 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

4.6.1 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 different *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 are to be used in a *study* to authenticate test takers, the *login mode username and password* allows a username and password to be entered. Analogous 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 modu username*, *username + password* and *access token*:
  - *List*: A list of valid credentials (username or username and password, depending on the *login mode* configuration) 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*, user name or access token serve as 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 various possible options for how login credentials can be provided.
  - *Platform*: A login dialog is displayed by the *IRTlib player* (i.e. the platform). The heading for entering the access data, the labeling of the input for user name and password, the labeling of the Next button, your 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 the offline version of the *IRTlib Player*) or via URL parameters (i.e. parameters when calling the *study* via a link to an online version of the *IRTlib Player*). In this case, no login dialog or login item is displayed.
  - *Item*: As an alternative to an *IRlLib Player* dialog, 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 *IRlLib-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. To add a login item to the study configuration, the integrated import dialog can be used.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

4.7 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.

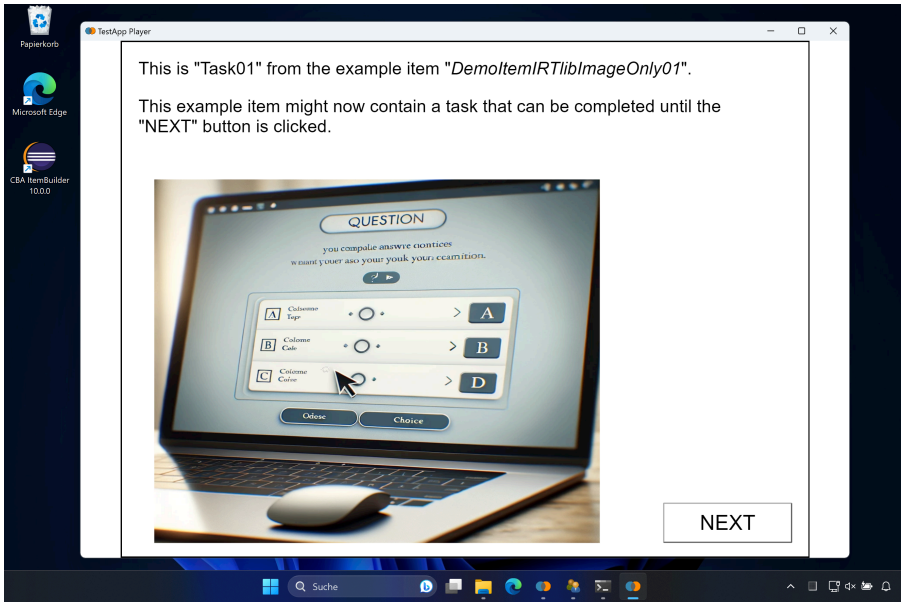
Detailed information on configuring the *Display* of a *Study* can be found here in the embedded help:

Embedded program help

4.7.1 Display settings

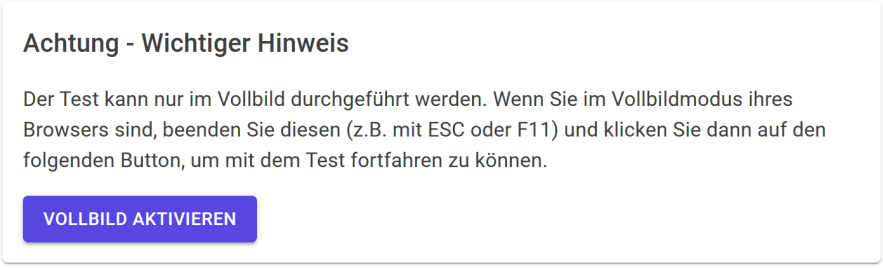
Selected options are available for configuring the display, which relate to the presentation of the assessment content and the use of *CBA ItemBuilder* content, which is created with a defined aspect ratio (width and height).#### Window mode
In the **Window mode** selection, you can configure whether an additional window is displayed in the *IRTlib Player*. The configuration is implemented differently depending on the environment:

- *Window*:In this *window mode*, a regular program 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.

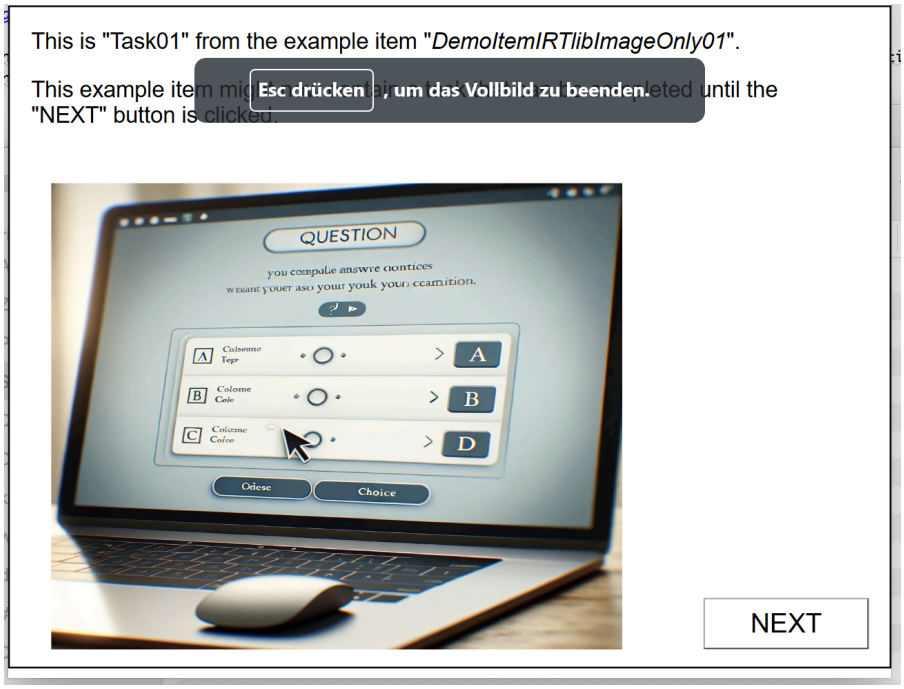


- *Full screen*:The offline *IRTlib Player* starts directly in full screen mode if this option is configured.This is also associated with a *Kiosk mode*, i.e. access to other programs and the (accidental) termination of the program is only possible via the *Task Manager*.If a test administrator, for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

  The online *IRlLib 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 the full screen mode can be ended 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).

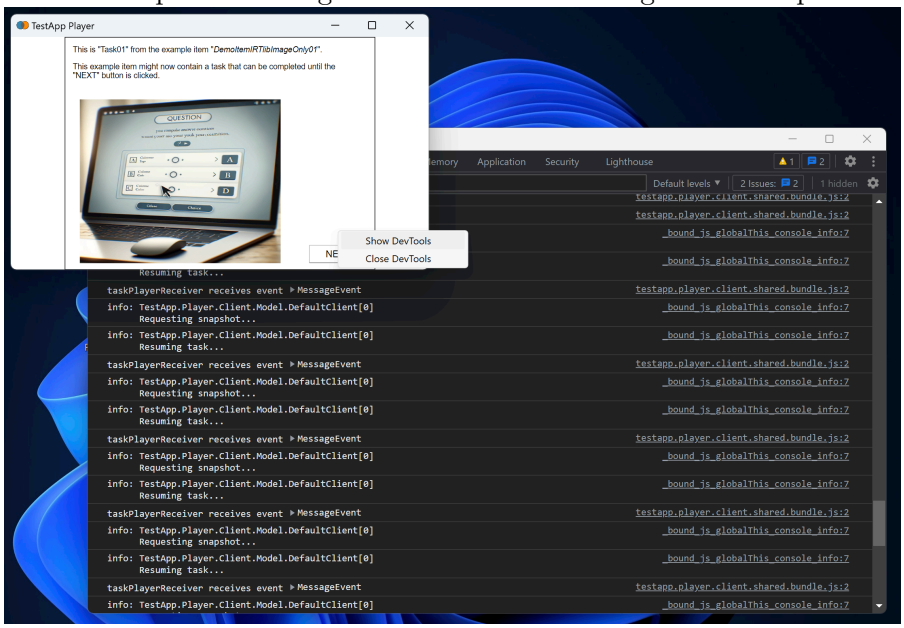
- *Full screen, if supported*:In this mode, the assessment in the online *IRTlib Player* is only displayed in full-screen mode if the browser supports full-screen mode. However, the content of the computer-based assessment is displayed in windowed mode when a study is delivered online and a browser that 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!

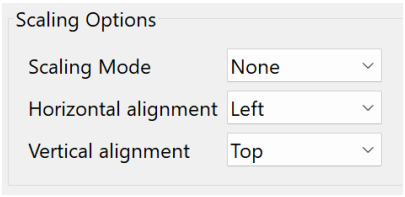
- *Debug*:This mode allows access to the browser's developer tools during test execution, which are intended for debugging by software developers.

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.>



4.7.2 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 options under *Scaling Options* are therefore available in the *Preview*:



Analog 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*).

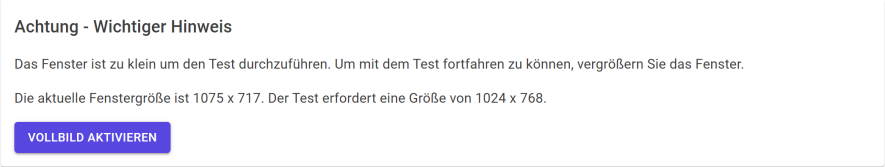
- None*:The content is displayed without adaptation to the available window or screen size (corresponds to `None`).
- Scale up*:Content is enlarged so that the available space is utilized (corresponds to `Up`).

- downscale*: Content is reduced in size so that it fits on the screen/in the window (corresponds to `Down`).
- *Window size*:Content is enlarged and reduced (corresponds to `Both`).
- **Horizontal alignment**:The options *centered* / *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 *centered* / *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.

4.7.2.1 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 simultaneously administrate a study in *window mode: window* or in *window mode: full screen* with one instance of an *IRTlib player*.
If changed settings are to be retained, the changes must be saved using the floppy disk symbol. Otherwise, the discard icon can be used:

4.8 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

4.8.1 Concept of a test administrator menu (menu for test administrators)

The test administrator menu is configured in two steps. First, a key combination must be defined with which the test manager menu can be called up. If this key combination is pressed during test processing, 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 must be defined in the *IRTlib Editor* in a second step.

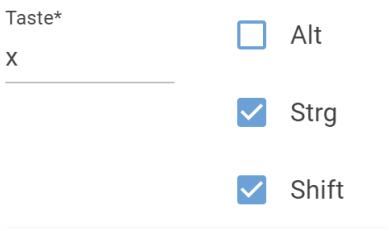
4.8.1.1 Access for test management

First, 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:

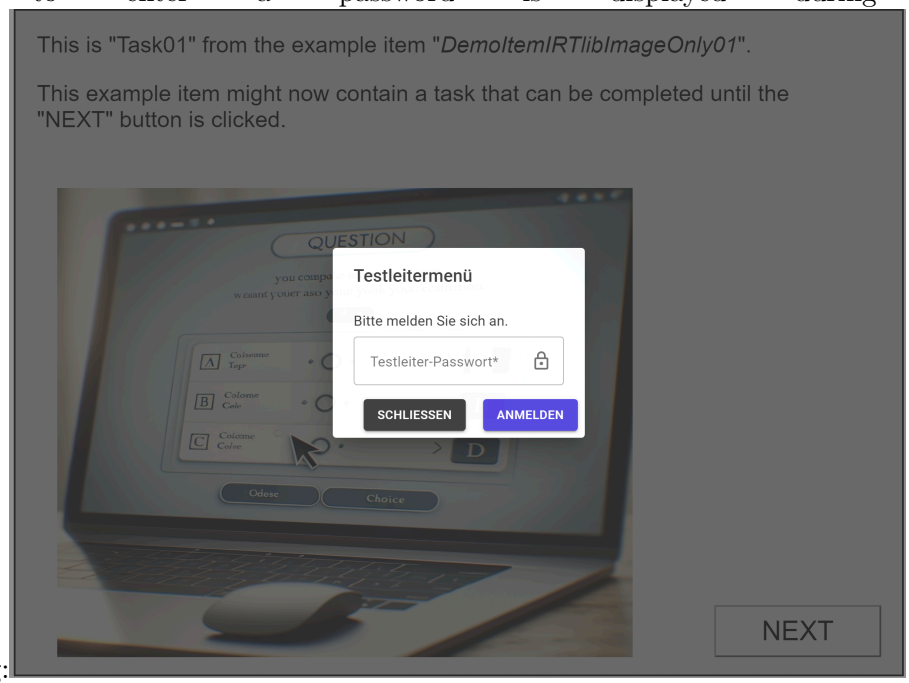
- The following configuration defines the key combination `Ctrl` + `Shift` + `X`:



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.

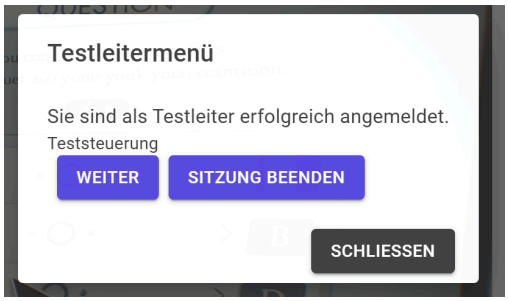
4.8.1.2 Roles

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

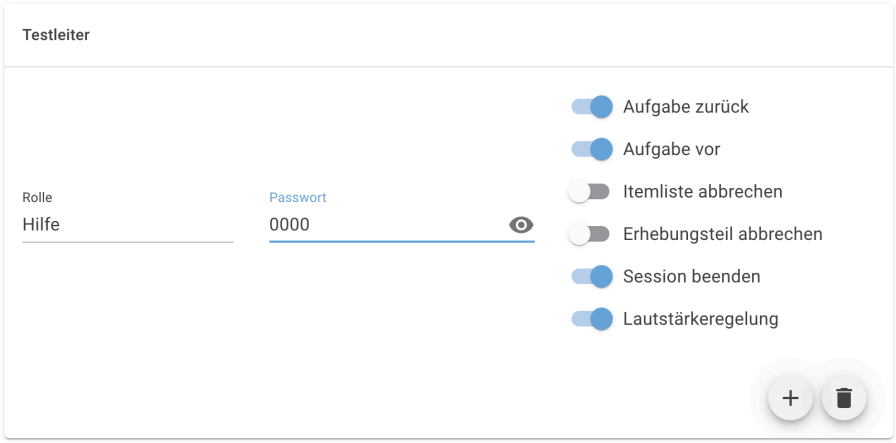


Which functions are actually accessible is controlled by which password is entered. Test line functions can only be accessed if a valid password is known.
Example:

- 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**:Enables the processing of the current item list to be canceled. 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**: Allows you to cancel the current survey part. **Cancel session**: Enables the current session to be ended.
- **Volume control**: Allows you to change the volume.

The audio file, which 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 floppy disk symbol. Otherwise, the discard icon can be used:

4.9 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

4.9.1 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.

4.9.1.1 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 behavior does not make sense if the login data is passed (either as *startup parameter* or as *URL parameter*).
- Show **end text**: If this option is selected, the platform displays the configured text. The text can be configured as *Message on end page*.
- **Display end item**: Analogous 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 (Redirct 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.

4.9.1.2 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 diskette icon. 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

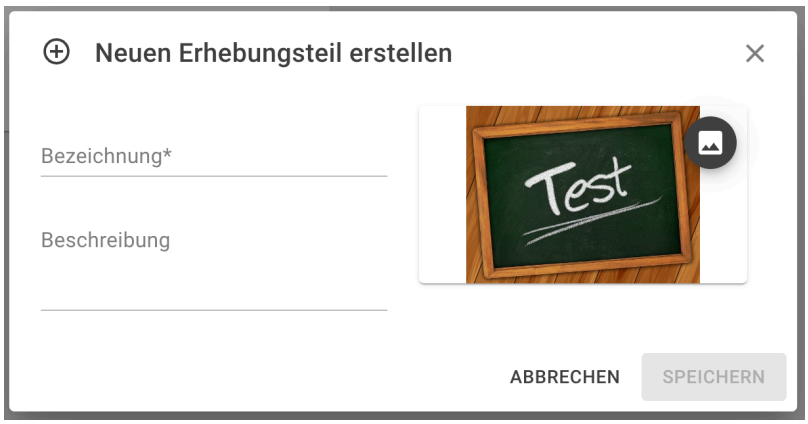
5.1.1 Create survey part

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 *assessment parts*.

5.1.1.1 How does it work?

Once a *study* has been created, a *survey section* can now be added via the plus icon at the bottom right:

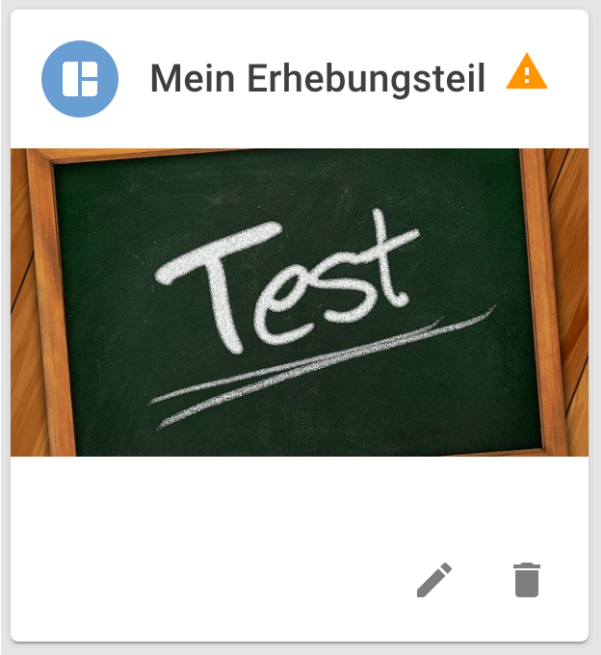
Then enter a *name* and optionally a *description* in the **Create new survey part** dialog. Make sure that only letters (upper and lower case), numbers and a _ are allowed for the *Name*.
Then click on *Save*.



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 floppy disk 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

5.2 Study-Part Configuration

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.

5.3 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

5.3.1 Configure items

5.3.1.1 Basic functions

Import *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* dialog 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* 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 part*.The *Preview* can then be called up via 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 of 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*:

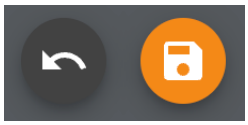
 > Note: It is not yet possible to delete *CBA ItemBuilder* project files from the *List of known items*. This functionality is not necessary because *CBA ItemBuilder* project files are only included in the configuration of a *study* by the *IRTlib Editor* if *tasks* from a *CBA ItemBuilder* project file are used in a *survey* section.

5.3.1.2 Sorting items (linear process)

- Sorting *CBA ItemBuilder* project files**:If the option *Enable routing* 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 diskette icon or discarded using the undo icon:



5.4 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

5.4.1 Define time limit

Survey parts without *routing* can easily contain a time-limited section. To do this, the option *Limit processing time* is activated in the *Processing time* 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 (analogous to non-time-limited *survey parts*):

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

In the *Processing time* 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*:

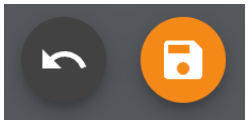
- **Lead items**:Items that are displayed before the time-limited section. **Follow-up items**: Items that are displayed after the time-limited section.

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

- Add: 
- Refresh: 
- Preview: 

- Download/Export:
- Delete:
- Sort:

Note: More complex designs with possibly 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 *Editing time* view must be saved using the diskette 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.

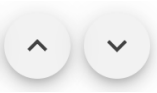
- Items administered *after* a potentially time-limited section of an assessment are referred to as *post-items*.

Embedded program help

5.4.2 Items according to a time limit

The *Survey section* allows the definition of items in different sections.Items in this section *epilog-item(s)* are displayed after the items defined in the *items* section of a *survey part*. The separation into *epilog-item(s)* and *items* is particularly useful if a time limit is activated under *Editing time*.
The following options are available for configuring items in the *Follow-up item(s)* section:

- Add:

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

Note: Changes to the *epilog-item(s)* view must be saved using the floppy disk symbol or discarded using the undo symbol:

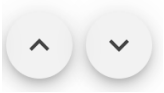
- Items that are administered *before* a potentially time-limited section of a survey part are called *prefix items*.

Embedded program help

5.4.3 Items before a time limit

The *Survey parts* allow the definition of items in different sections. Items in this section *Prefix item(s)* are displayed before the items defined in the *Items* section of a *Survey part*. The separation into *Foreword item(s)* and *Items* is particularly useful if a time limit is activated under *Editing time*.
The following options are available for configuring items in the *Foreword item(s)* section:

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

- Sort: 

Note: Changes to the *Screenshot-item(s)* view must be saved using the diskette icon or discarded using the undo icon:

5.5 Variables

Under Development

This function is currently under development.

Embedded Program Help

(This functionality is still under *development*).

5.6 Codebook

Under Development

This function is currently under development.

Embedded Program Help

(This functionality is still under *development*).

5.7 ItemPool

Under Development

This function is currently under development.

Embedded Program Help

(This functionality is still under *development*).

5.8 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

5.8.1 Summary of routing within survey parts

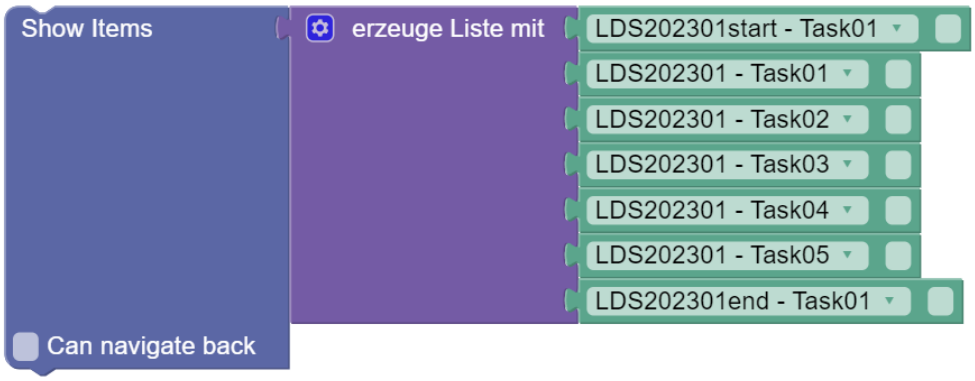
The sequence of *CBA ItemBuilder* tasks can be defined here using *Blockly* (i.e. a form of visual programming). *Blockly*-based sequencing is available if the option *Enable routing* is selected for a survey part. The option can be found in the *Info* section of a survey part. If it is activated, the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

5.8.1.1 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:

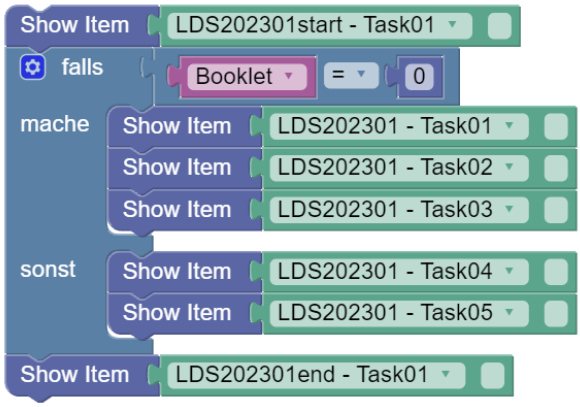


A list of *CBA ItemBuilder Tasks* is passed to the *Blockly* element *Show Items*, which is created with the operator *create list with*. The list is processed in the order shown, whereby each *CBA ItemBuilder Tasks* is displayed until the `NEXT_TASK`- *Command* is executed.

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

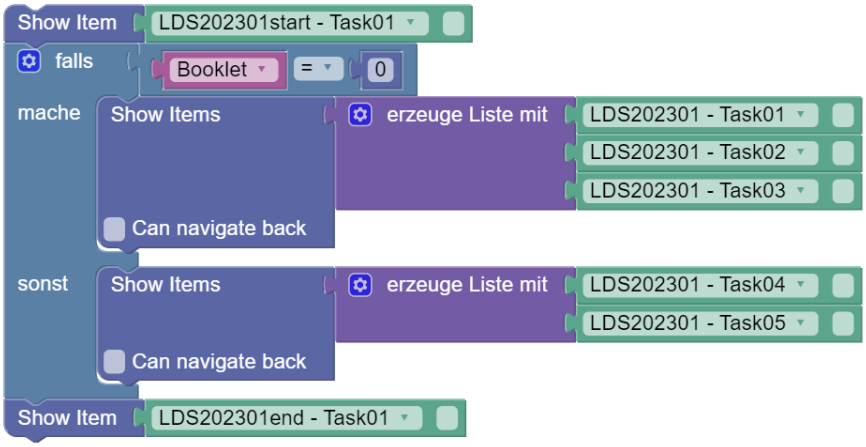
- **Example for simple test booklets**

  With the help of an variable (here: *booklet*) and a simple *if/make*-condition, you can now define a process 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*.

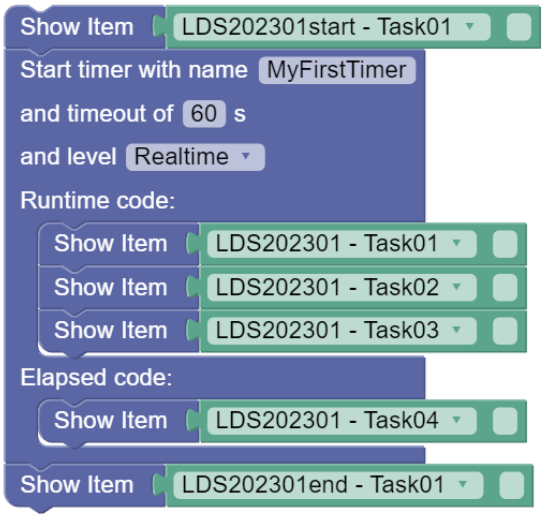
  Alternatively, the identical sequence can also be created using the *Blockly* operator for displaying item lists:



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**

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



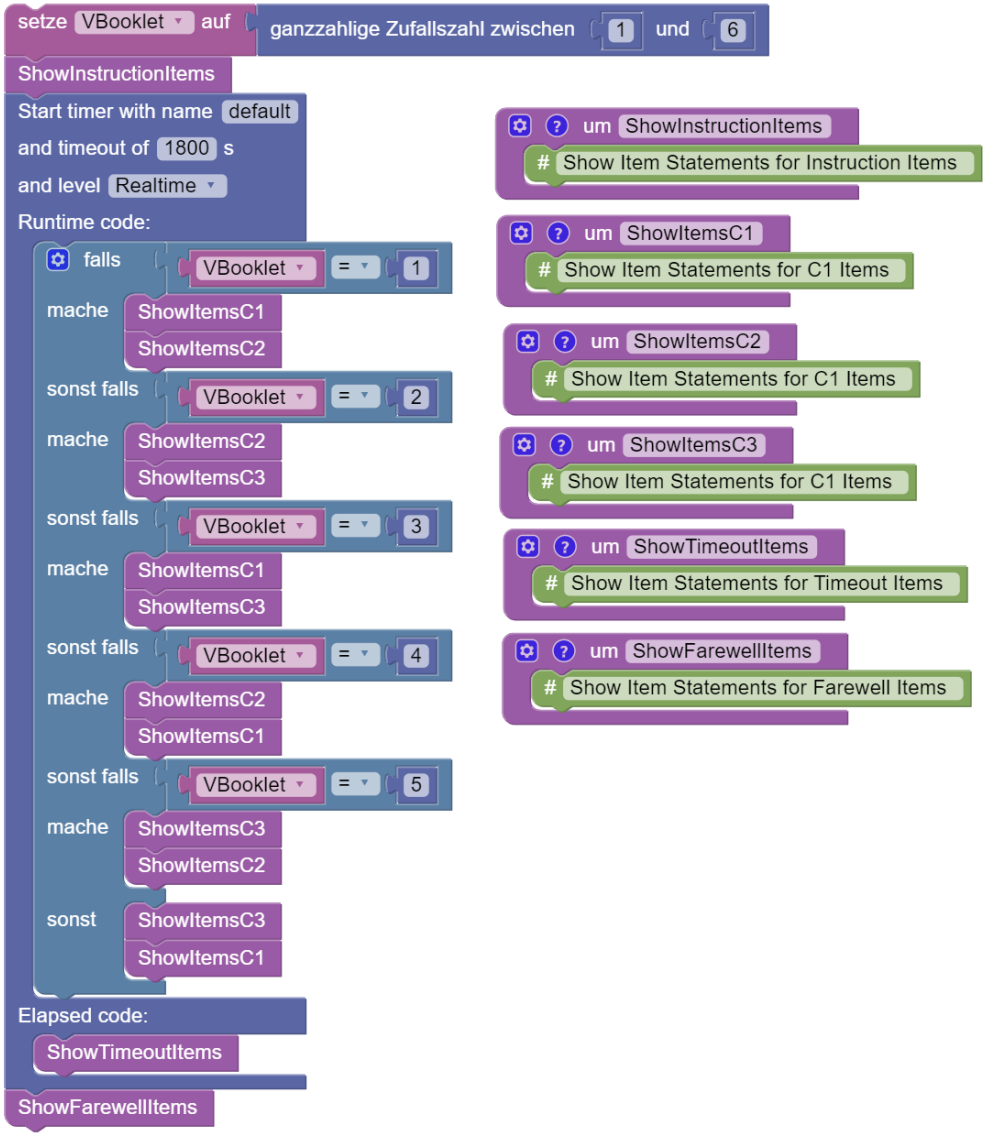
  Each sequence begins with a start task that is not time-limited and ends with an end task that is also not time-limited. In between, there is a time limit for a section called *MyFirstTimer*, which has a time limit of *60* seconds.

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).

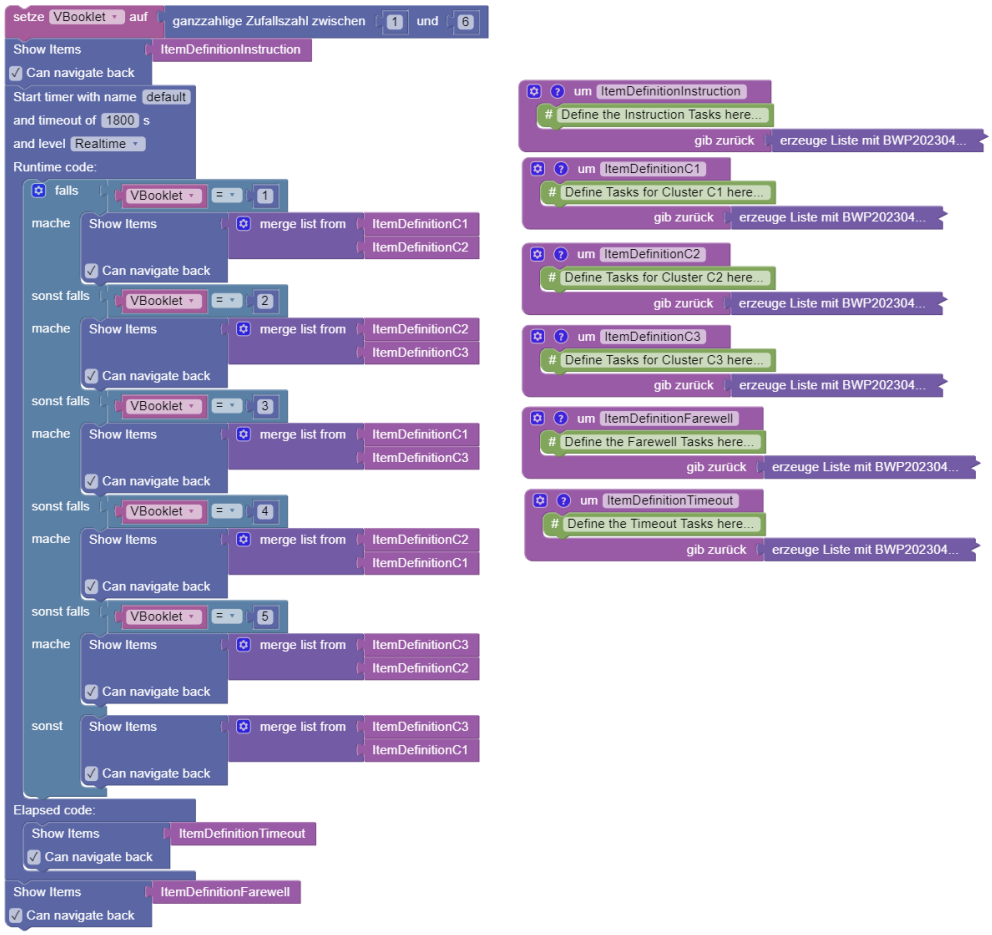
Example for simple booklet design with time limit

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.

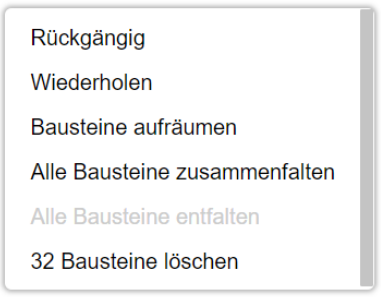
5.8.1.2 Notes on using the *Blockly* editor

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*:



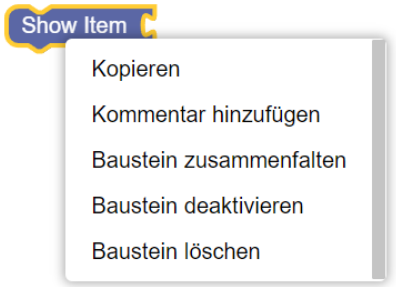
- **Save**: Adjustments in the *Blockly* editor must be saved. The floppy disk symbol is available for this purpose at the bottom right:

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.

- The deletion of blocks is also possible via the context menu.



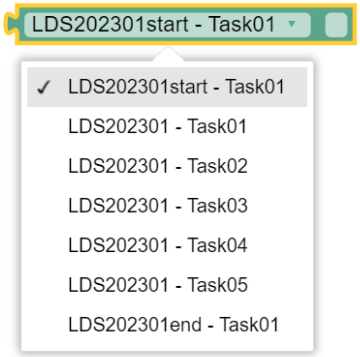
Some *Blockly* elements also provide a *Help* entry in the context menu, which refers to the generally accessible *Blockly* documents (https://github.com/google/blockly/wiki/).

5.8.2 Use of *Blockly* for flow control

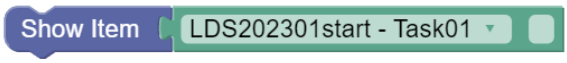
The basic functions for using the *Blockly* environment to control assessments can be found in the *Session* section.

5.8.2.1 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*:



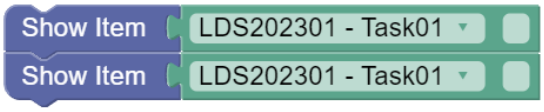
The element, which can be found in the *Session* section of the *Blockly* editor palette, can be configured using the selection list. Each *Blockly* element for tasks can refer to exactly one specific task, i.e. a flow definition usually consists of several such elements.
Blockly elements for tasks cannot 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.

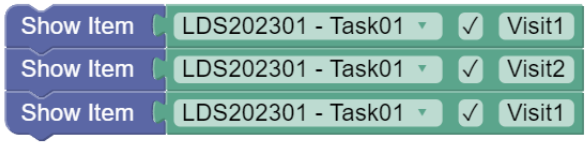
5.8.2.2 Use of scopes (scopes)

With the help of *Blocky*-based flow control, it is also possible to administer *CBA Item-Builder* 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:

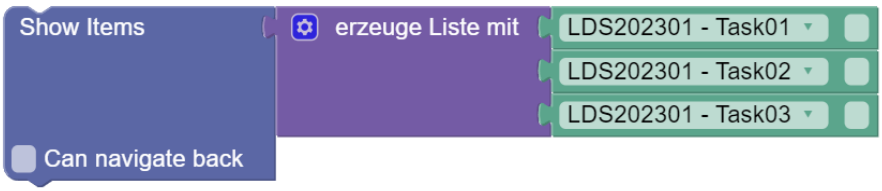
If nothing else is specified, the item is administered in the "default" scope. Alternatively, a text can be defined, as shown in the following example:



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 not displayed with the data that was already collected during the first visit (i.e. the *Scope* "Visit1" is used again).

5.8.2.3 Display multiple items (item lists)

As can be seen 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 request navigation to the previous *CBA ItemBuilder Tasks* with the *Command* `BACK_TASK`.

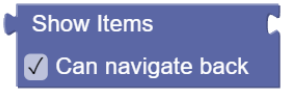
Canceling lists: The use of lists also allows lists to be canceled. Lists can be canceled in two ways:

```
- The *Command* `CANCEL_TASK`, which can be used within CBA ItemBuilder Tasks, is called.
- The function *Cancel item list* is called up in the test administrator menu, which has be
```

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

5.8.2.4 Display of items with storage of the results

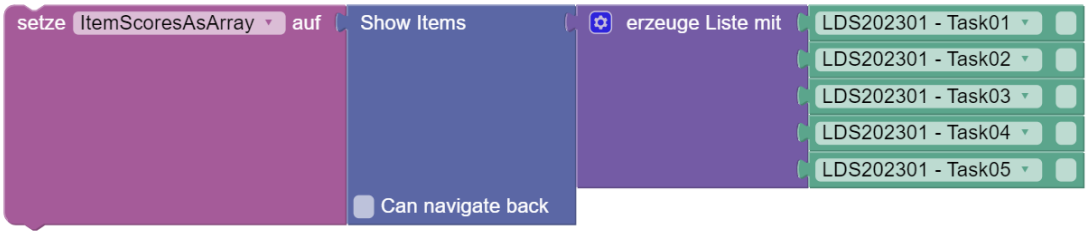
The operators *Show Item* (for individual items) and *Show Items* (for item lists) are also available as operators for value assignments:



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

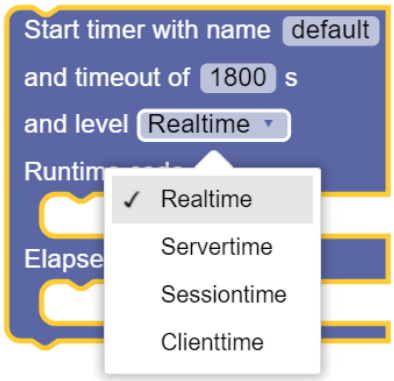
- Single task:

- List of tasks:



5.8.2.5 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. In addition, the time must be specified in seconds. This can be used to define the type of time to be used:



- Realtime:
- Servertime:
- Sessiontime:
- Clienttime:

Finally, two places 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):

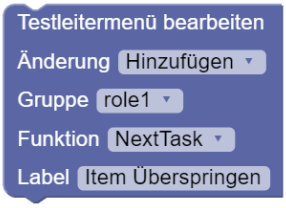
- Runtime code: These blocks are filled in until the defined time has elapsed.
- Elapsed code: These blocks are only filled in if the *Runtime code* has not been completed within the time.

5.8.2.6 *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.
Customize 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 part, the following *Blockly* operator can be used in the flow control to customize the test administrator menu for specific contexts:



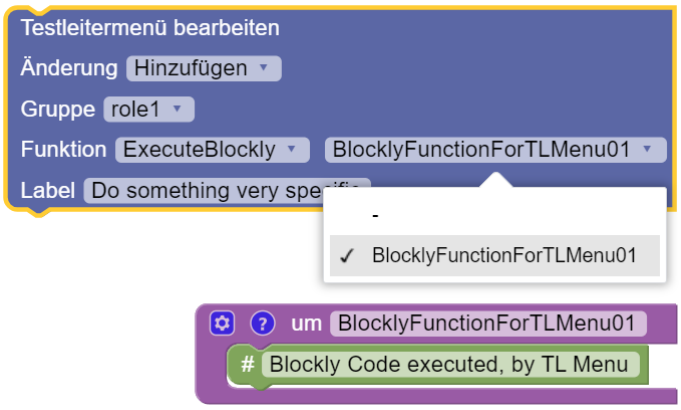
The test administrator menu can be changed for each of the standard functions (in the *Function* section) for a role (in the *Group* section) as well as the button label (in the *Label* section):

- *Add*: Function is added in 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 behavior 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:

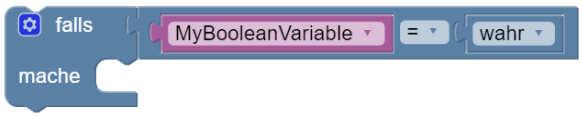


If *ExecuteBlockly* is selected, then a function defined within the *Blockly* editor can be selected in the *Blockly* element *Edit test administrator menu*. The *Blockly* operators defined in this function are then executed when a test leader selects the corresponding button in the test leader menu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

5.8.3 Advanced *Blockly* usage

5.8.3.1 Flow control with conditions

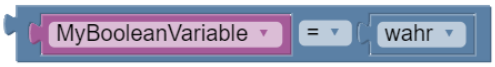
The *Logic* section contains the *Blockly* operator *if/make*, which can be used to implement conditions in the flow. Conditions are logical expressions, e.g. checking whether a preload variable has a certain value:



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, which is connected to the *blockly* operator *if/make*. Here are the two components separately:

- Condition:

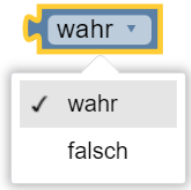
- Logical expression:



5.8.3.2 Use of logical expressions

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

The two slots can be filled with values. For Boolean values (true/false), a corresponding Blockly element is available in the *Logic* section:

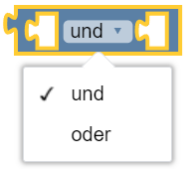


Conditions are also possible with variables of a different data type:

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

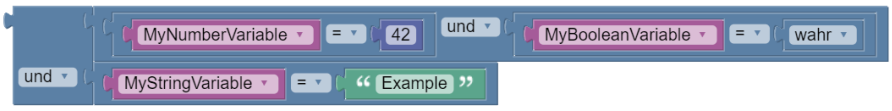
With its help and a numeric variable, the following condition can be formulated:
For technical reasons, it may also be necessary to check whether a variable has no value at all. This can be implemented by using the blockly component `null`:

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



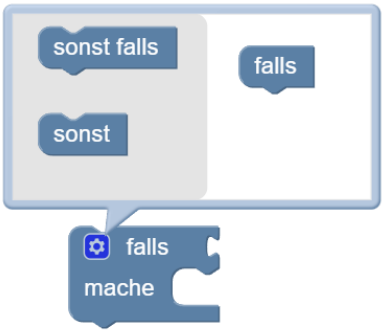
An *and* and an *or* linking of the statements is available for selection. The free *inputs*

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



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

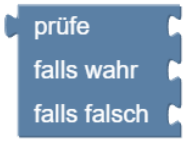
Multiple conditions (*if* / *else*): By clicking on the small cogwheel symbol of a condition block (*if/make*), it can be configured:



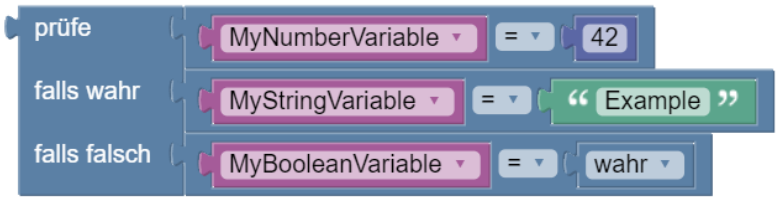
By adding an *unless* section, a further condition can be added. The condition defined in an *if* section is checked if the previous conditions (*if*) are not fulfilled. If a condition is fulfilled, the defined blockly operators are executed.

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

Special case: *check* operator for three conditions: For three conditions, the *blockly* editor provides a special *check-if-true-if-false* operator:



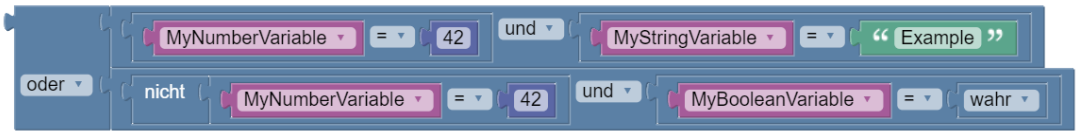
The operator combines two logical expressions, e.g:



The construct is a short form for the following check, as shown in the following table:

MyNumberVariable	MyStringVariable	MyBooleanVariable	Result
= 42	= Example	(any)	true
= 42	≠ Example	(any)	false
≠ 42	(any)	true	true
≠ 42	(any)	false	false

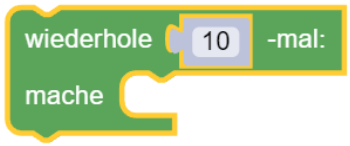
Without the operator for three conditions, the same check could be implemented with the following combination:



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

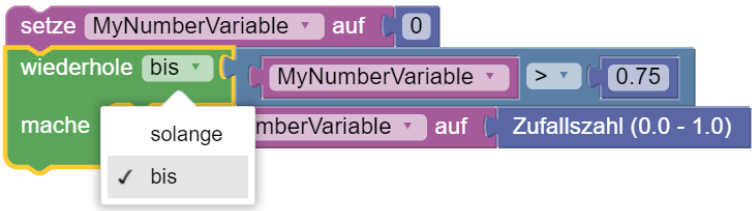
5.8.3.3 Sequence control with loops

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:

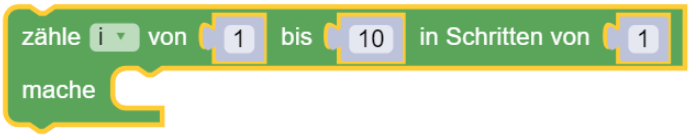


Repeat as long as: Loops can also be repeated *until* a condition is true (or *as long as* a condition is true):

Example:

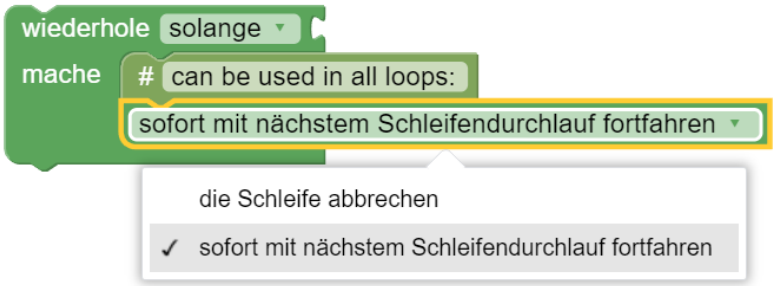


Count from/to: Loop with auxiliary variables:



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

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



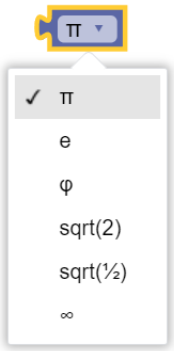
5.8.3.4 Operators for numbers and simple mathematical functions

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

Expressions

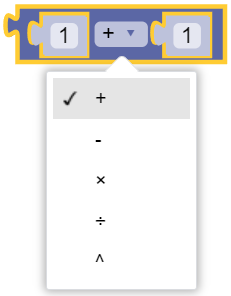
- Numbers: Integers / decimal numbers

- Symbols: Special symbols or constants:

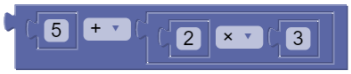


Basic functions

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



  Nesting is possible, e.g:



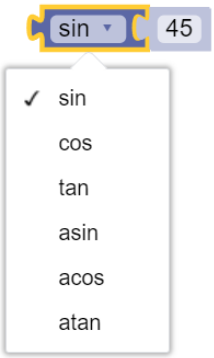
- Division with remainder:

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

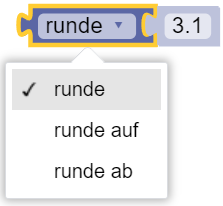
- With the following *blockly* element, a number can be limited to a section:

Built-in functions

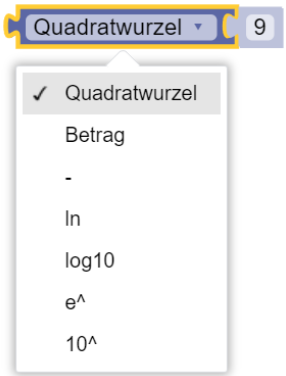
- Trigonometric functions:



- Rounding of values:



- Further functions:

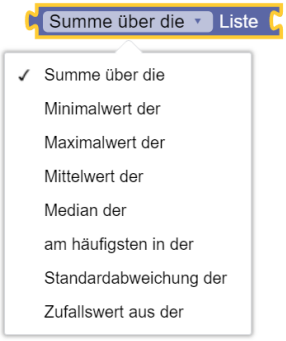


Generation of random numbers: Two *blocky* elements are available for generating random numbers:

- 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, make sure that the selected function can be used for the data types of the list.

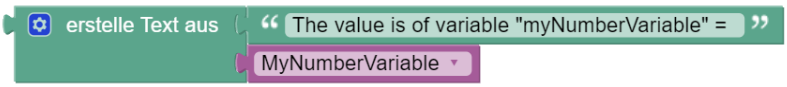
5.8.3.5 Operators for text and simple string operations

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:



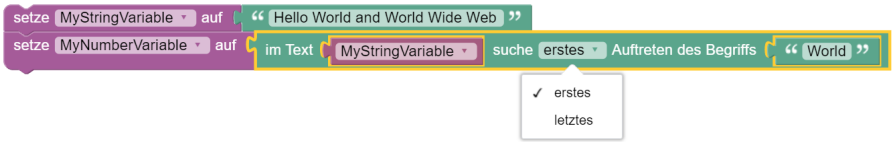
- Assign a variable to merged texts:

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

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

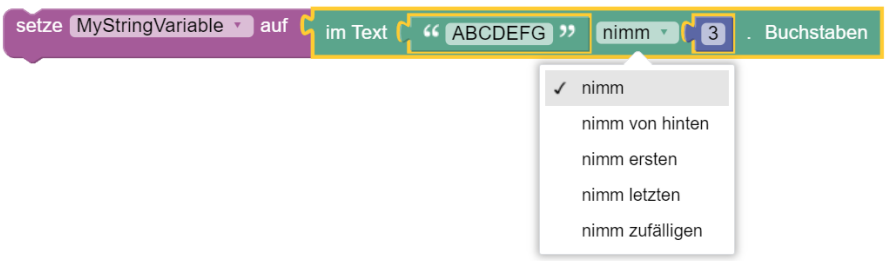
Alternatively, the following *blockly* operator can be used:

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:



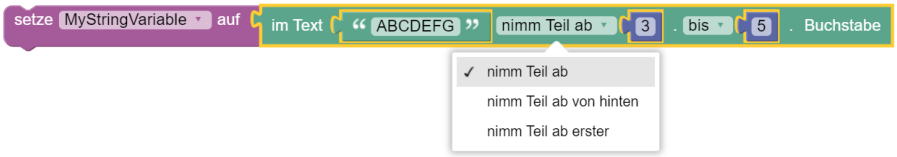
The position of the *term* within the character string (i.e. *in the text*) is returned.
Form sub-strings: The following operator takes the first letters from the passed string *in text*. The number of letters is also passed.

- 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`):



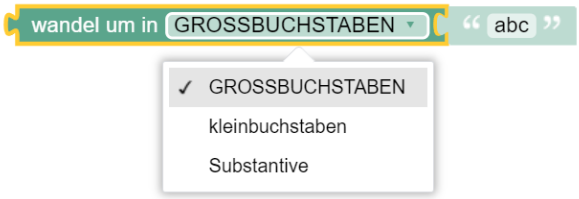
Letters from a character string can also be extracted using the following operator and assigned to a variable, for example:

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

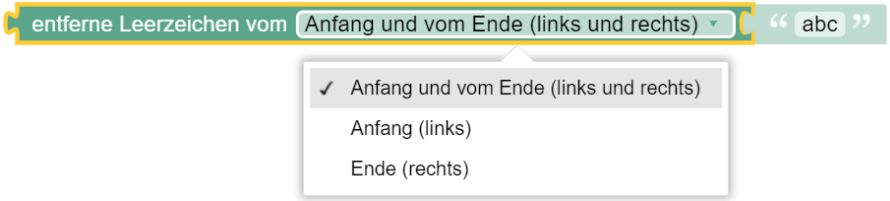


Change texts: Existing texts (either as expressions or from variables of *datatype string*) can be modified by applying operators.

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

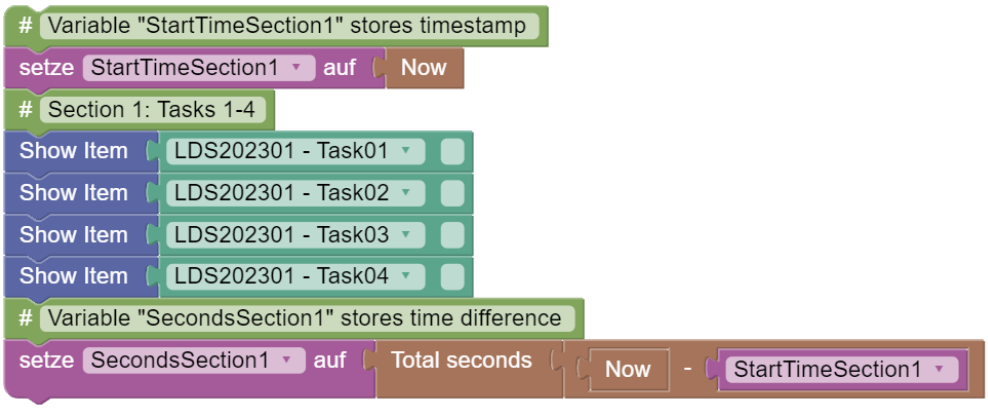


- Leading, trailing or leading and trailing spaces can be removed using the following operator:

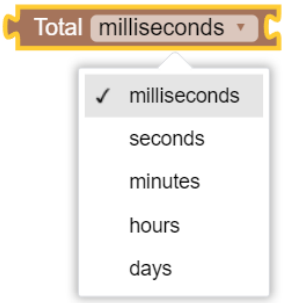


5.8.3.6 Operators for times and simple time operations

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:



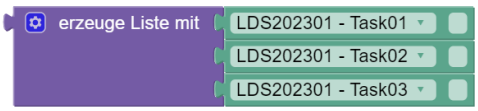
Conversion of time measures



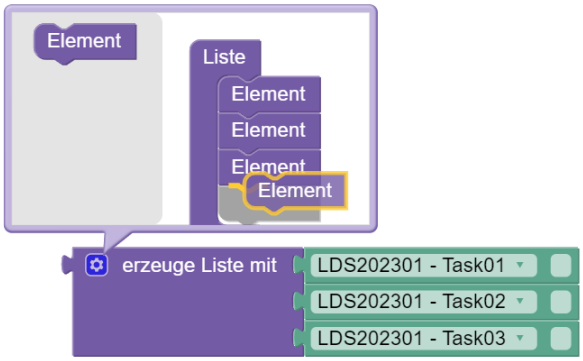
5.8.3.7 Operators for lists

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 the following elements:

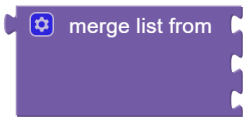


  The number of elements of the *create list with* operator can be configured using drag-and-drop after clicking on the cogwheel icon:



- List can be created by repeating an element:

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

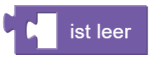


Sublists: A sublist can be selected from lists using the following operator:

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:

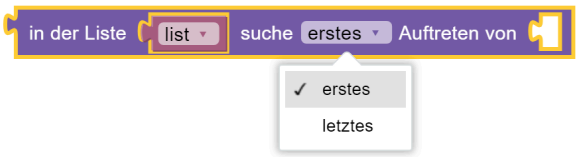


- The following operator returns the length of the list:

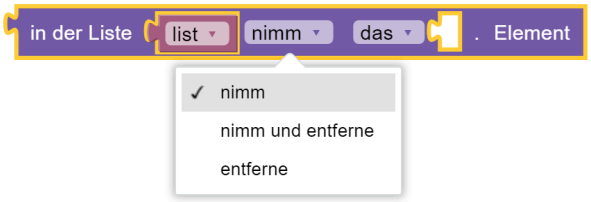
- The following operator returns the distinct elements of a list

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

- The following operator finds elements in lists:

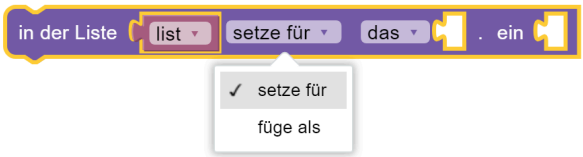


- The following operator returns / removes or replaces in a list and returns the ELement:



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

- The following operator replaces under inserts in a list:



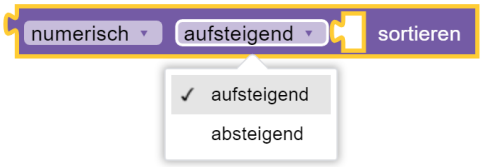
  Further options of the operator for *that*: *from behind that* / *first* / *last* and *random*.

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:

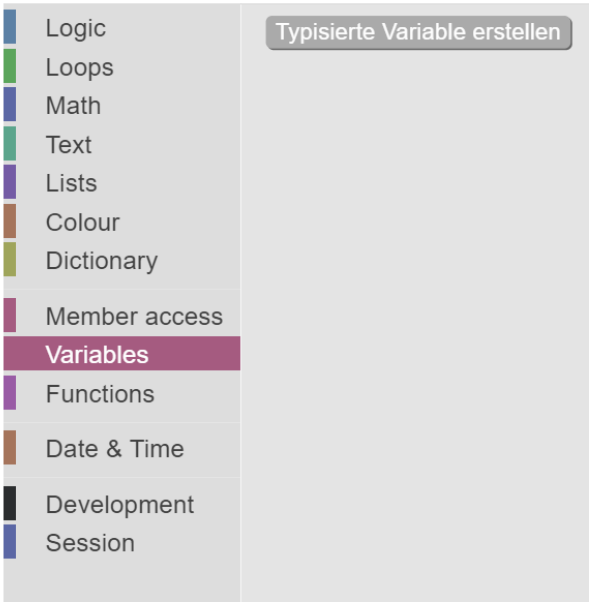
Sort lists: Elements in lists can also be sorted.

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

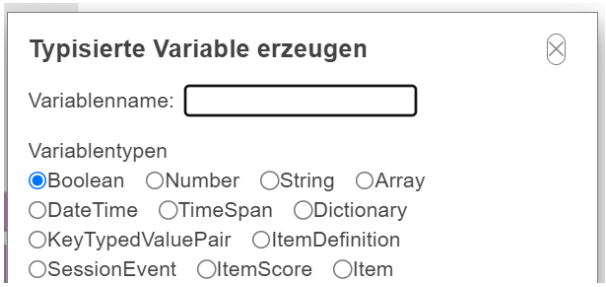


5.8.3.8 *Blockly* variables

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:

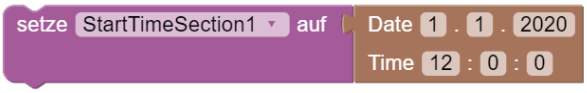
- *Boolean*: Logical truth values and logical expressions (`true` or `false`)

- *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



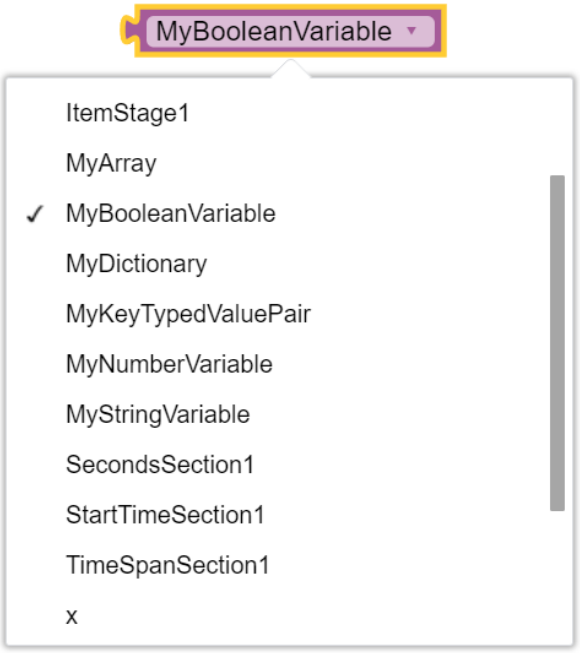
- *TimeSpan*: time span

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

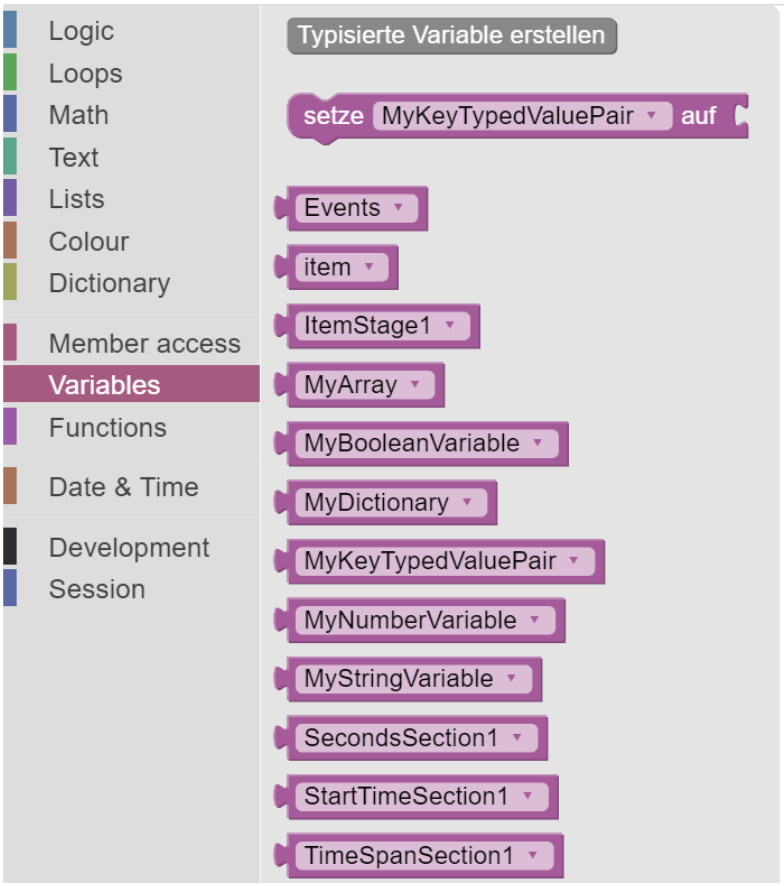
- *Array*: Data type for lists

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

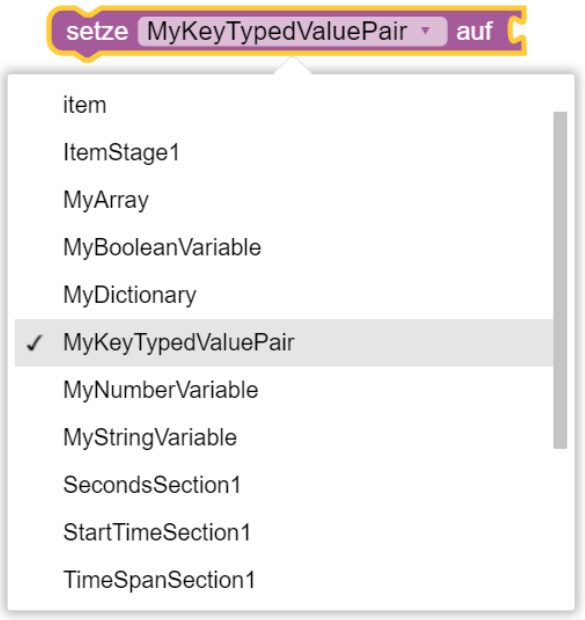
Use variable values: To use variable values, *Blockly* elements with *inputs* can take the following component:



- You can select which variable is used. For defned variables, there is also a *Blockly* element in the *Variables* section of the *Palette*:



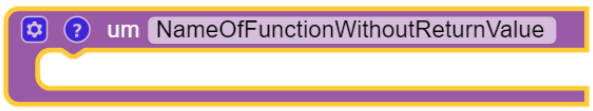
- The palette also contains a *blockly* element of the type *set … on*. This can also be used to select the value of which variable it sets:



5.8.3.9 *Blockly* functions

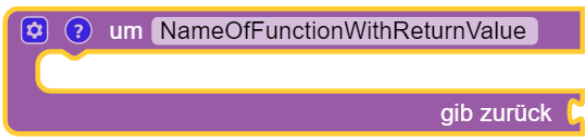
The *Functions* section of the *Palette* contains *Blockly* elements for using functions within flow definitions. Functions combine *blocky* code so that it can only be defined once but used multiple times.
Defining functions: Two different forms of functions can be defined.

- Functions without a return value:



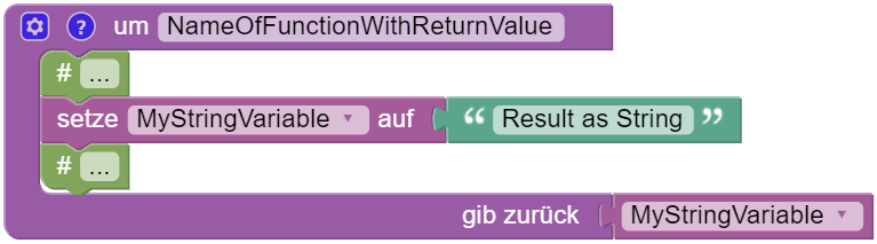
  To be called, functions without a return value can simply be connected to previous and subsequent *blockly* elements in the sequence (i.e. they have an up and down connection):

- Functions with return value:



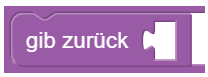
Functions with a return value can be called in an assignment block (i.e. they have a connection to the left):

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*:



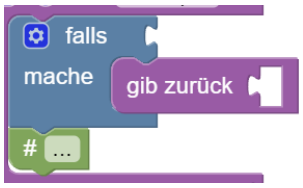
In addition, the following two *blockly* elements are available, which can only be used within a function definition (with return value):

- 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 *gib zurück* operator has no downward connection):



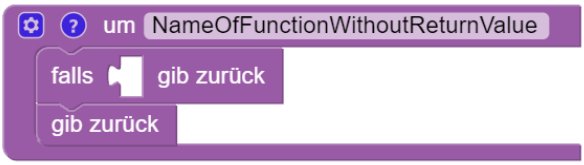
- The *if return* operator only returns a value if a condition is met. If the condition is fulfilled, the processing of the sequence in the function ends; if the condition is not fulfilled, processing continues (i.e. the *if return* operator has a downward connection):

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



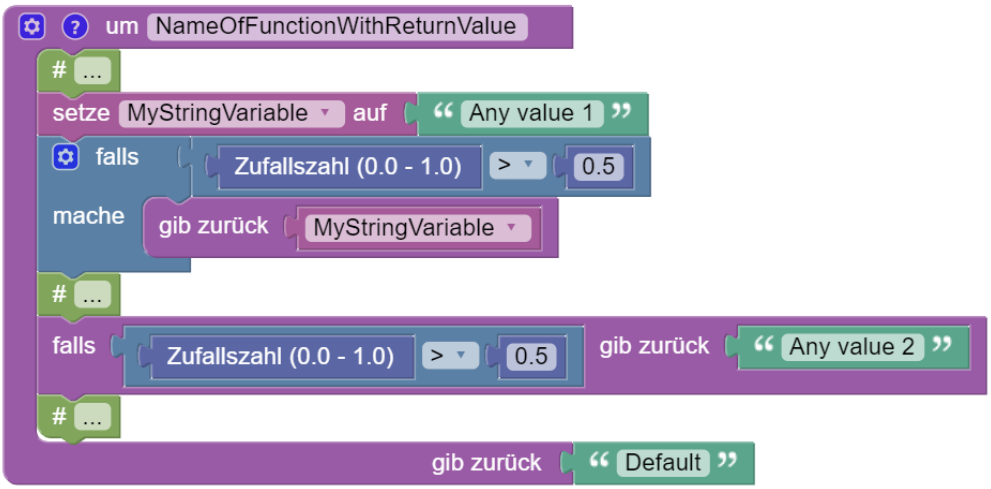
- Both operators (*if returns* and *return*) cannot be used outside of functions:

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



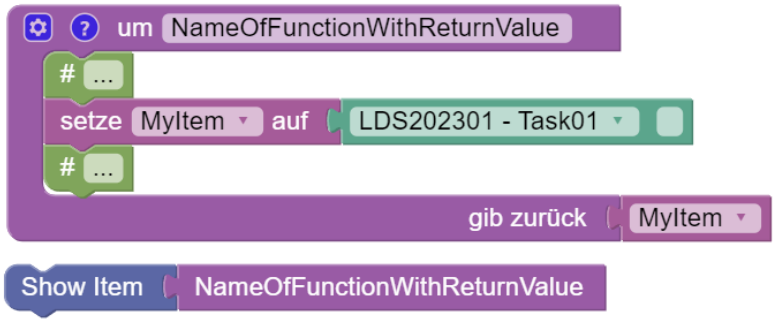
Example:

- 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:

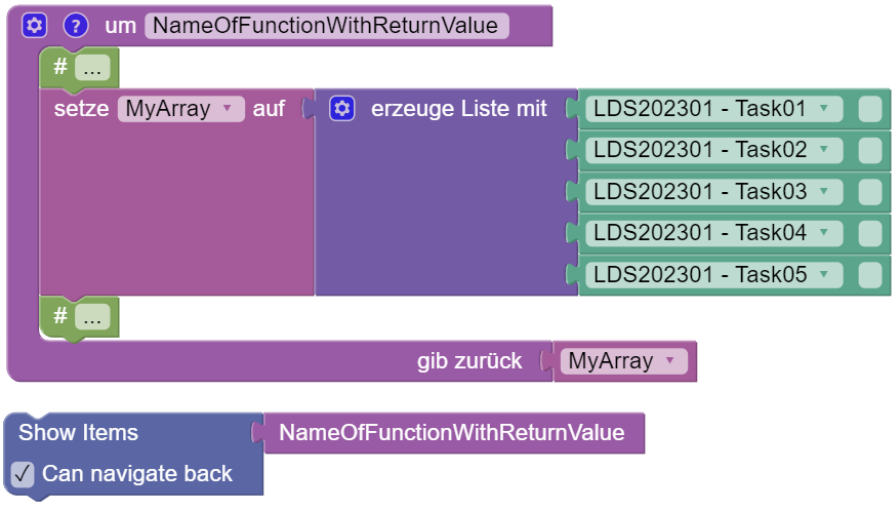


Return values are typed. The flow control also supports functions that …

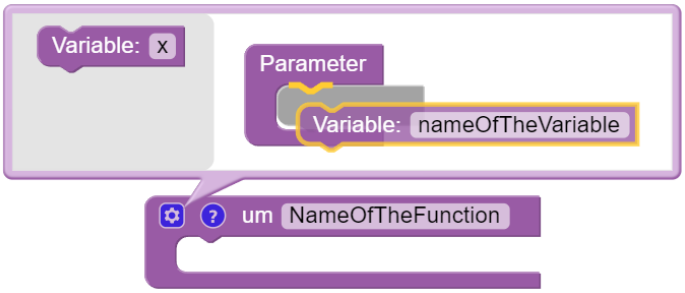
- … return individual *tasks*:



- … Return lists of *tasks*:

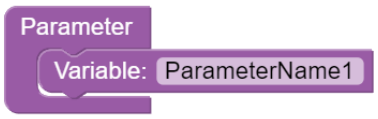


Defining call parameters of functions: Functions can also use parameters that are to be passed when the function is called (*call parameters*). Call parameters can be defined by clicking on the small cogwheel symbol of a function block:



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

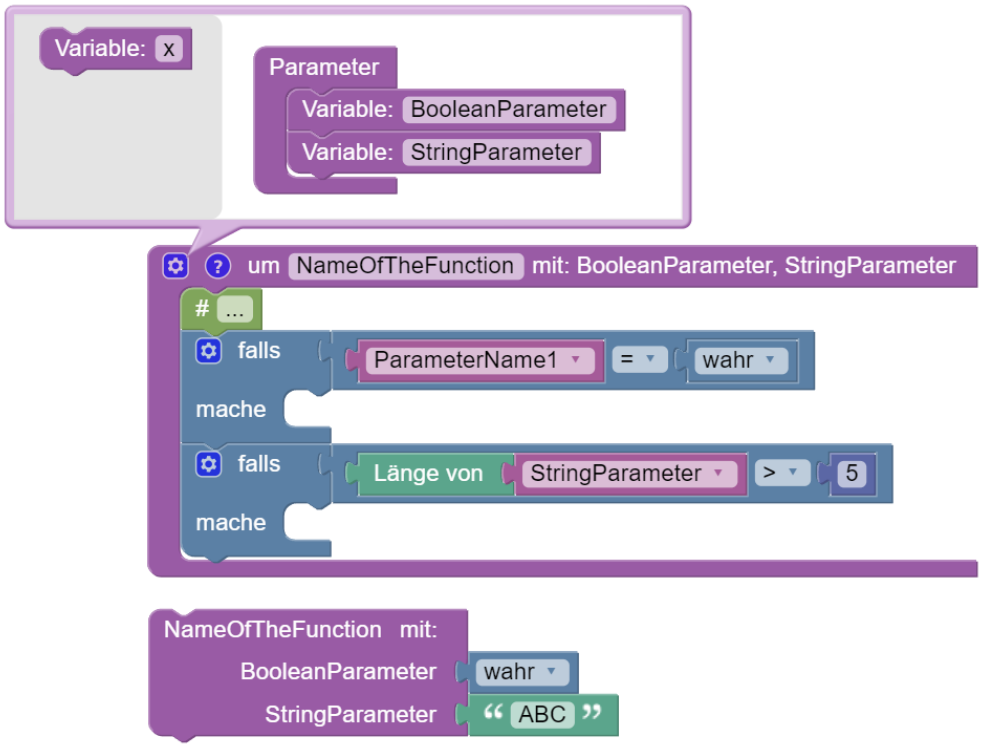
- Definition of a parameter



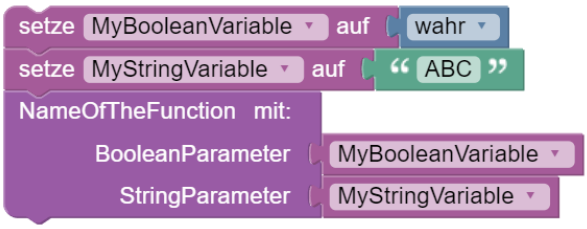
- Call the function with value:

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:



5.8.3.10 Use of item results in the flow control

(documentation)

5.8.3.11 *Blockly* operators for encoding missing values

(documentation follows)

5.8.3.12 *Blockly* operators for writing data

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

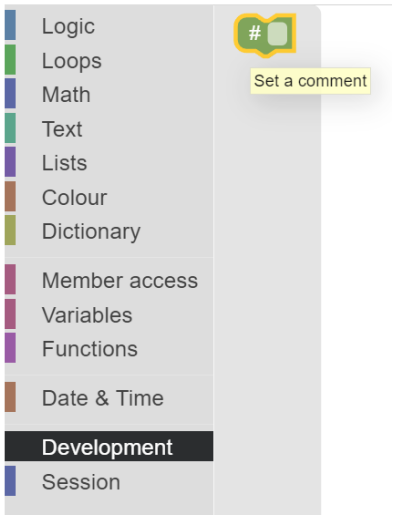
Result data: (documentation follows)
Monitoring data: (documentation follows)

5.8.4 Commenting on *Blockly* code

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

5.8.4.1 Comments as *Blockly* elements

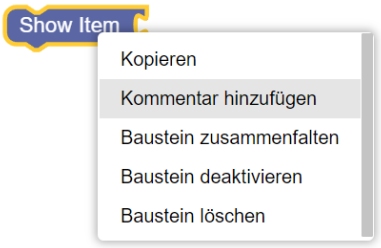
Comments that are to be permanently visible in the flow can be added via the plaette in the *Development* section:



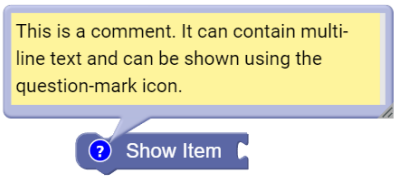
These comments can be moved like blocky operators and show one-line comment text.

5.8.4.2 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.

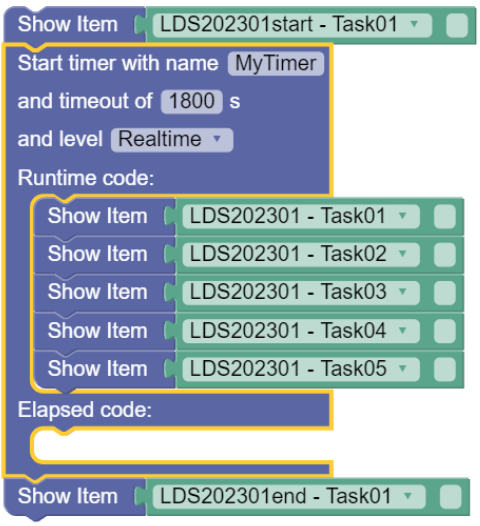


5.8.5 Presentation of *Blockly* code

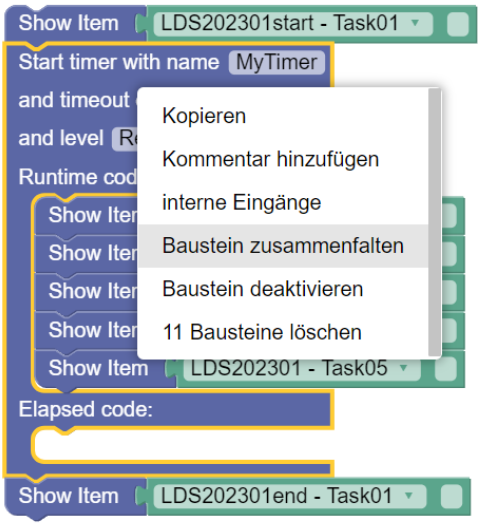
5.8.5.1 Unfolding / folding

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:

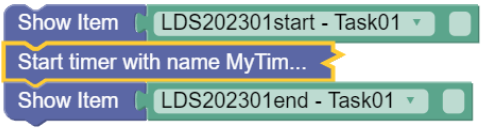
- Unfolded (i.e. complete) display of the selected block:



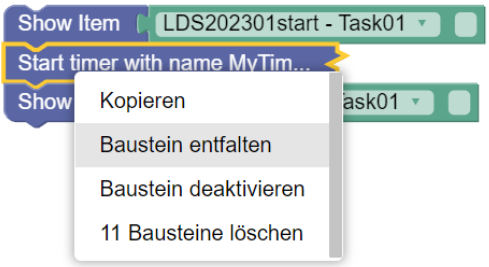
- Option to *fold* the block in the context menu:



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



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

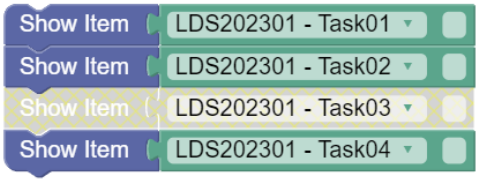


The *folding* / *unfolding* of *blockly* elements does not change the function of a flow definition and is only used for a clearer arrangement of complex flow definitions.

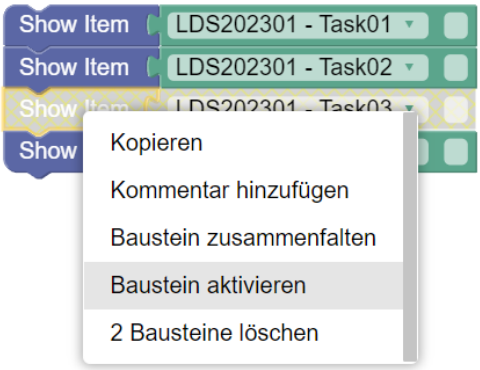
5.8.5.2 Deactivating / activating

The *Blockly* editor offers the option of only deactivating *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:



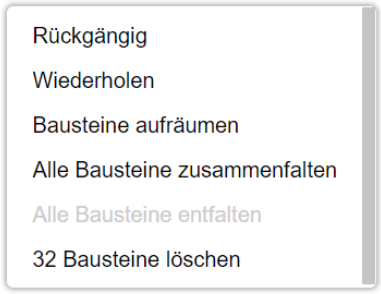
Internal / External: Some *blockly* elements with *inputs* (i.e. places where you can connect further blocks) allow you to switch between two display forms.

- 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.

5.9 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.

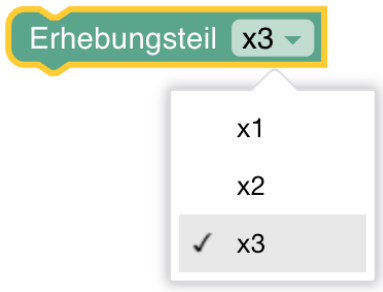
A detailed description of *routing between survey parts* can be found here in the embedded help:

Embedded program help

5.9.1 Summary of routing between survey parts

The order of *survey parts* can be defined using *Blockly* (analogous to defining 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

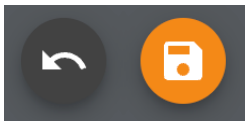


This *Blockly* operator replaces the *Show Item* within survey parts.

- Successful login

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

Note: Changes to the *Routing* view between *Collection parts* must be saved using the diskette symbol or discarded using the undo symbol:



Part II

Allgemein / General

6 Einstellungen / Settings

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

6.1 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

6.1.1 Settings

In this area, settings can be made that affect work with the Editor and all studies.

6.1.1.1 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 area.
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*.

6.1.1.2 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.

6.1.2 About the program

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

6.2 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

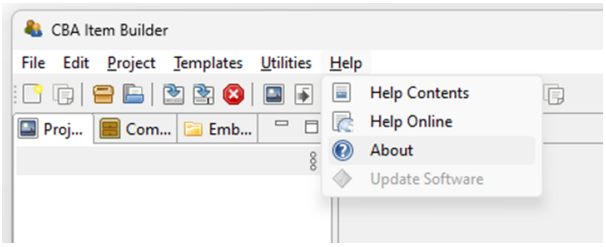
6.2.1 Runtimes

To use the IRTLib Editor for configuring studies that use CBA ItemBuilder content, the appropriate runtime environment (*Runtime*) is required for that 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.

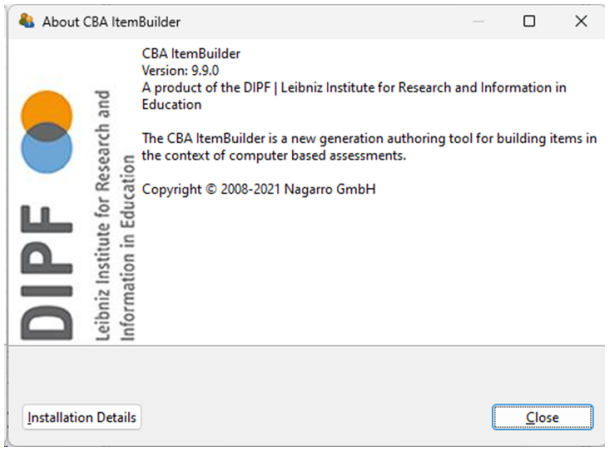
6.2.1.1 Check CBA ItemBuilder Version

You need to know which version of the CBA ItemBuilder was used to prepare the items (i.e., the CBA ItemBuilder project files). If you are in doubt, this information can be found, for instance, in the *About Dialog* of the CBA ItemBuilder:

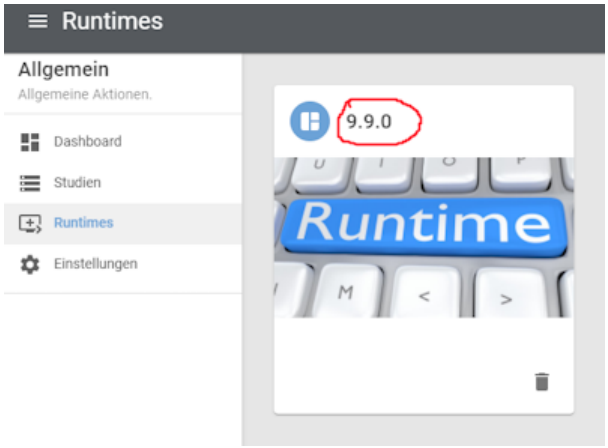
- Step 1: Open the "About"-dialog using the "Help"-menu



- Step 2: Find the version number in the dialog (here `9.9.0`)



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



6.2.1.2 Import Runtime Files

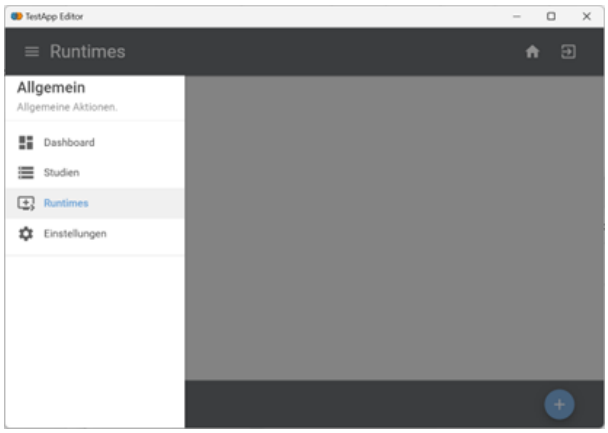
If the appropriate runtime is not already included in the editor, a new/additional *Runtime* can be imported. Study configurations created/edited with the *IRTlib Editor* can contain multiple *Runtimes* for different versions.

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

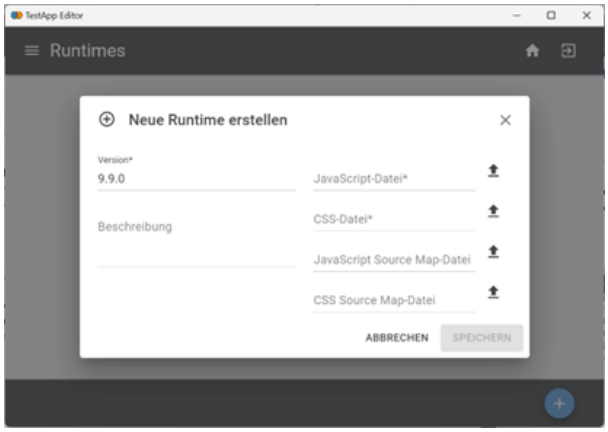
  https://cba.itembuilder.de/appendix-tables.html#previous-versions

- Step 2: Unzip the downloaded *Runtime* that should be used.

- Step 3: Navigate to the section *Runtimes*:



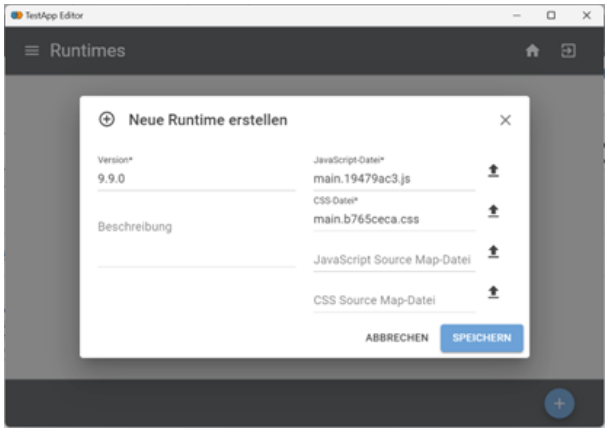
- Step 4: Push the button "+" (bottom right)
- Step 5: Enter the version number with three places (e.g., 9.9.0):



- Step 6: Select the file `main.*.js` from the ZIP archive that contains the runtime. Note that the * equals the hash of the file (i.e., the complete file name looks like `main.19479ac3.js`)
- Step 7: Select the file `main.*.css` from the ZIP archive that contains the runtime. Note that the * equals the hash of the file (i.e., the complete file name looks like `main.b765ceca.css`)

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

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



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

7 Github

7.1 IRTLib Software

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

> **Note**
>
> Translation: If you would like to help us with the translation of this software, you can find more information here.

7.1.1 Download

- Current versions of the IRTlib Software (Windows und Docker): github
- Documentation: github

7.2 CBA ItemBuilder

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

7.2.1 Download

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

7.2.2 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 / Rumtime: 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)

7.2.3 Documentation

Online-Dokumentation

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

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