

Ulf Kroehne

Open Computer-based Assessment with the CBA ItemBuilder

Open Computer-based Assessment with the **CBA ItemBuilder**



December 17, 2024

Suggested Citation:

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

Contents

| | |
|---|--------------|
| List of Figures | vii |
| List of Tables | xxi |
| Preface | xxiii |
| 1 Introduction | 1 |
| 1.1 Installation & Requirements | 3 |
| 1.2 Contact & Support | 5 |
| 1.3 Quick Start: Get the Right Version | 5 |
| 1.4 Quick Start: Preview Items | 7 |
| 1.5 Quick Start: Explore Scoring | 13 |
| 1.6 Quick Start: Explore Log Events | 19 |
| 1.7 Quick Start: Revise Item Content | 23 |
| 1.8 CBA ItemBuilder for Software-Developers | 29 |
| 2 Principles of Computer-Based Assessment | 33 |
| 2.1 Advantages & Benefits of CBA | 34 |
| 2.2 Standardized Response Formats | 38 |
| 2.3 Innovative Item Types / Technology-Enhanced Items (TEI) | 48 |
| 2.4 Item Presentation and Navigation | 49 |
| 2.5 Scoring and Calibration | 59 |
| 2.6 Automated Item Generation | 67 |
| 2.7 (Automated) Test Assembly | 68 |
| 2.8 Log and Process Data | 76 |
| 2.9 Feedback | 86 |
| | iii |

| | | |
|----------|---|------------|
| 2.10 | Item and Test Security | 90 |
| 2.11 | Design Principles of the CBA ItemBuilder | 91 |
| 3 | Designing Items Using Static Content | 95 |
| 3.1 | Overview of the User Interface | 96 |
| 3.2 | CBA ItemBuilder <i>Projects Files</i> | 106 |
| 3.3 | Quick Start: Create Single Page Items | 113 |
| 3.4 | Pages and Page Types | 133 |
| 3.5 | Basic Containers (<i>Frame, Panel and PageArea</i>) | 138 |
| 3.6 | <i>Tasks</i> as Entry Points | 151 |
| 3.7 | Layout Pages using Components | 159 |
| 3.8 | Components to Display Text | 172 |
| 3.9 | Components to Collect Responses | 180 |
| 3.10 | Images and Multimedia Components | 203 |
| 3.11 | Links between Pages | 214 |
| 3.12 | Runtime Commands | 225 |
| 3.13 | Components for Special Page Types | 230 |
| 3.14 | Embedding HTML Content | 236 |
| 3.15 | Pages as Dialogs (<i>Popups</i>) | 240 |
| 4 | Enriching Items using Dynamic Content | 245 |
| 4.1 | Syntax Overview | 247 |
| 4.2 | Variables and Value Maps | 252 |
| 4.3 | Conditional Links | 266 |
| 4.4 | Finite-State Machine(s) | 281 |
| 4.5 | Task Initialization Syntax | 327 |
| 4.6 | Interactive Content in <i>ExternalPageFrames</i> | 328 |
| 5 | Scoring of Tasks | 339 |
| 5.1 | Terminology, Concepts and User Interface | 340 |
| 5.2 | Scoring using FSM Variables | 346 |
| 5.3 | Definition of Explicit Scoring Rules | 348 |

| | | |
|----------|---|------------|
| 5.4 | Automatically Generated Variables | 361 |
| 5.5 | Checklist and Complete Workflow | 363 |
| 6 | Recipes and Examples | 365 |
| 6.1 | Regular Expressions | 366 |
| 6.2 | Ressources Files | 373 |
| 6.3 | Global Properties | 377 |
| 6.4 | FSM and Conditional Link Syntax Examples | 382 |
| 6.5 | Calculators Examples | 393 |
| 6.6 | ExternalPageFrame Examples | 395 |
| 6.7 | Adaptive Testing with the CBA ItemBuilder | 405 |
| 6.8 | (More) Efficient use CBA ItemBuilder | 408 |
| 6.9 | Creating Assessments in Multiple Languages | 418 |
| 7 | Test Assembly and Deployment | 419 |
| 7.1 | Quick-Start: Assessments using R (and Shiny) | 419 |
| 7.2 | (Technical) Terminology and Concepts | 421 |
| 7.3 | Using CBA ItemBuilder Items with R (Shiny Package) | 429 |
| 7.4 | Using CBA ItemBuilder Items with TAO (using Portable Custom Interactions) | 436 |
| 7.5 | Using CBA ItemBuilder Items with the IRTlib-Software | 440 |
| 7.6 | Using CBA ItemBuilder Items in SCORM Packages (with xAPI) | 445 |
| 7.7 | Using CBA ItemBuilder Items in Custom Web Applications (Taskplayer API) | 451 |
| 8 | Assessment Cycle and Workflows | 455 |
| 8.1 | Planning of CBA Projects | 456 |
| 8.2 | Distributing Content to Project Files and Tasks | 461 |
| 8.3 | IT-Management of CBA Projects | 465 |
| 8.4 | Testing CBA Projects | 476 |
| 8.5 | Running Assessments | 480 |
| 8.6 | Data Processing after Assessments | 481 |
| 8.7 | Documentation and Archiving of Computer-Based Assessments | 484 |

| | |
|--|------------|
| Closing Chapter | 493 |
| Appendix | 497 |
| A Glossary of Terms | 497 |
| B Useful Tables | 507 |
| B.1 Main Menus | 507 |
| B.2 Operators | 512 |
| B.3 Regular Expression Symbols | 529 |
| B.4 Technical Configuration | 530 |
| B.5 CBA ItemBuilder Versions | 532 |
| B.6 Component Register | 533 |
| B.7 Documenetation Log Events | 537 |
| Bibliography | 551 |
| Index | 565 |

List of Figures

| | | |
|------|---|----|
| 1.1 | Item illustrating <i>Installation Wizard</i> (html ib). | 3 |
| 1.2 | CBA ItemBuilder <i>Main Window</i> . | 6 |
| 1.3 | CBA ItemBuilder <i>About-Dialog</i> . | 7 |
| 1.4 | Example item to test <i>Preview</i> (html ib). | 8 |
| 1.5 | CBA ItemBuilder <i>File-Menu</i> . | 10 |
| 1.6 | CBA ItemBuilder <i>Project-Menu</i> . | 11 |
| 1.7 | CBA ItemBuilder <i>Preview-Dialog</i> . | 11 |
| 1.8 | CBA ItemBuilder <i>Preferences-Dialog</i> . | 12 |
| 1.9 | Warning about autostart of audio and video components. | 13 |
| 1.10 | CBA ItemBuilder <i>Scoring Debug Window</i> . | 16 |
| 1.11 | <i>Scoring-example</i> (score-variables, html ib). | 18 |
| 1.12 | <i>Scoring-example</i> extended (with result-variables, html ib). | 19 |
| 1.13 | CBA ItemBuilder <i>Tracing Debug Window</i> . | 22 |
| 1.14 | <i>Button-example</i> for single-choice response format (html ib). | 23 |
| 1.15 | CBA ItemBuilder <i>Edit Text Occurrence Dialog</i> . | 27 |
| 2.1 | Item illustrating <i>QTI</i> interactions for simple items (html ib). | 39 |
| 2.2 | Example illustrating different operationalizations of <i>Response Time</i> (html ib). | 45 |
| 2.3 | Item illustrating <i>Average Answering Time</i> for item batteries (html ib). | 46 |
| 2.4 | Item illustrating different item formats described in (Scalise and Gifford, 2006, html ib). | 49 |
| 2.5 | Example for full page navigation. (html ib). | 52 |
| 2.6 | Example for forced choice navigation. (html ib). | 53 |
| 2.7 | Example for multiple forced choice questions per page. (html ib). | 53 |

| | | |
|------|--|-----|
| 2.8 | Examples for <i>Blocked Item Response</i> with different response formats (html ib). | 54 |
| 2.9 | Example for a C-Test / Text Completion Test (html ib). | 55 |
| 2.10 | Example for a <i>Digits Symbol Substitution Test</i> (html ib). | 55 |
| 2.11 | Example for question and stimulus navigation (html ib). | 56 |
| 2.12 | Examples for free <i>Between-Unit</i> navigation (html ib). | 57 |
| 2.13 | Examples for restricted <i>Between-Unit</i> navigation (html ib). | 58 |
| 2.14 | <i>Fixed Form Testing</i> with linear sequence of <i>Tasks</i> | 69 |
| 2.15 | <i>Fixed Form Testing</i> with multiple <i>Blocks</i> or <i>Parts</i> | 70 |
| 2.16 | Conditional <i>Test Part</i> using between-part <i>Routing</i> | 71 |
| 2.17 | Example for a simple <i>Booklet Design</i> using within-part <i>Routing</i> | 71 |
| 2.18 | Basic Principle of <i>Multi-Stage Testing</i> | 72 |
| 2.19 | Simplified Illustration of <i>Computerized Adaptive Testing</i> | 73 |
| 2.20 | Illustration of <i>Computerized Adaptive Test Initialization</i> | 74 |
| 2.21 | Illustration of <i>Computerized Adaptive Testing Loop</i> | 75 |
| 2.22 | Item illustrating the replay feature (work in progress) (html ib). . . | 83 |
| 2.23 | Item in the style of a Raven's Progressive Matrices test (html ib). . . | 87 |
| 2.24 | Item illustrating result feedback (html ib). | 87 |
| 2.25 | Item illustrating task progress feedback (html ib). | 88 |
| 3.1 | <i>Main Menu, Toolbar</i> and left area of the CBA ItemBuilder user interface (<i>Project View, Embedded HTML Explorer, Renderer</i> and below the <i>Component Edit</i>). | 97 |
| 3.2 | <i>Context Menu</i> in the <i>Project View</i> clicked on the root (left) and on a page (right) | 97 |
| 3.3 | Five steps for opening a newly created page in the <i>Page Editor</i> and open the <i>Palette</i> | 100 |
| 3.4 | <i>Properties</i> view of a selected component of type <i>Panel</i> | 102 |
| 3.5 | <i>Properties</i> view showing the tab <i>Appearance</i> | 103 |
| 3.6 | <i>Properties</i> view shows <i>Rules & Grid</i> if the <i>Page</i> is selected. | 104 |
| 3.7 | <i>Task</i> editor in the right area of the user interface. | 104 |
| 3.8 | <i>Variables</i> editor in the right area of the user interface. | 105 |
| 3.9 | <i>Value Maps</i> editor in the right area of the user interface. | 105 |

| | | |
|------|--|-----|
| 3.10 | Rename Project-Dialog accessible form context menu in <i>Project View</i> . | 107 |
| 3.11 | Warning about allowed characters for project names. | 108 |
| 3.12 | Dialog asking to save the changes in the current project. | 109 |
| 3.13 | Dialog asking for a preview of the current project before saving the project file is possible. | 109 |
| 3.14 | Request to preview the current project before saving the project is possible. | 109 |
| 3.15 | CBA ItemBuilder <i>Preferences</i> to define the <i>CBA Presentation Size</i> for new projects. | 111 |
| 3.16 | CBA ItemBuilder <i>Global Properties</i> to define the <i>CBA Presentation Size</i> for existing projects. | 111 |
| 3.17 | Video: Hands-on section 3.3.1 (html ib). | 114 |
| 3.18 | Output of hands-on section 3.3.1 (html ib). | 117 |
| 3.19 | Video: Hands-on section 3.3.2 (html ib). | 117 |
| 3.20 | Output of hands-on section 3.3.2 (html ib). | 121 |
| 3.21 | Video: Hands-on section 3.3.3 (html ib). | 121 |
| 3.22 | Output of hands-on section 3.3.3 (html ib). | 124 |
| 3.23 | Video: Hands-on section 3.3.4 (html ib). | 125 |
| 3.24 | Output of hands-on section 3.3.4 (html ib). | 128 |
| 3.25 | Video: Hands-on section 3.3.5 (html ib). | 128 |
| 3.26 | Output of hands-on section 3.3.5 (html ib). | 130 |
| 3.27 | Video: Hands-on section 3.3.6 (html ib). | 130 |
| 3.28 | Output of hands-on section 3.3.6 (html ib). | 132 |
| 3.29 | Toolbar icon and context menu to create a new page. | 133 |
| 3.30 | CBA ItemBuilder <i>New Page</i> -Dialog (left without and right with <i>Advanced</i> properties) | 134 |
| 3.31 | Item illustrating different <i>Page Types</i> (html ib). | 135 |
| 3.32 | Tag for creating a new page as <i>XPage</i> | 136 |
| 3.33 | Checkbox to tag a page created using <i>Template Browser</i> as <i>XPage</i> . . . | 136 |
| 3.34 | Dialog for changing <i>Page Settings</i> | 137 |
| 3.35 | Empty <i>Page Editor</i> if no <i>Frame</i> is defined. | 139 |
| 3.36 | Video illustrating manual definition of <i>Frame</i> and <i>Panel</i> (html ib). . | 140 |

| | | |
|------|--|-----|
| 3.37 | Message of a failed <i>Preview</i> because every <i>Page</i> requires a <i>Frame</i> . . . | 140 |
| 3.38 | Dialog <i>Configure Select Groups</i> for single- and multiple-choice select groups. | 142 |
| 3.39 | Context menu in the <i>Component Edit</i> for the <i>Frame</i> Component to open the dialog <i>Configure Select Groups</i> | 142 |
| 3.40 | Example for <i>Panel</i> s and <i>PageArea</i> s nested within <i>Frame</i> s (html ib). . . | 143 |
| 3.41 | Context menu entry <i>Set Auto Layout</i> for selected <i>Panel</i> s to open the dialog <i>Auto Layout Panel Properties</i> | 145 |
| 3.42 | Dialog <i>Auto Layout Panel Properties</i> showing an example with merged cells. | 146 |
| 3.43 | Dialog <i>Component Edit</i> of a page with <i>Auto Layout Panel</i> | 147 |
| 3.44 | Message after the attempt to <i>Set Auto Layout</i> for a <i>Panel</i> that already contains components. | 147 |
| 3.45 | Message after <i>Duplicate</i> was called on the content of a <i>GridArea</i> . . . | 147 |
| 3.46 | Item illustrating scrollbars of <i>PageArea</i> s (html ib). | 148 |
| 3.47 | Item illustrating how to use content in <i>PageArea</i> s multiple times (html ib). | 149 |
| 3.48 | Item illustrating how to add a <i>PageArea</i> if a <i>Panel</i> fills the <i>Frame</i> completely (html ib). | 149 |
| 3.49 | Context menu for components of type <i>PageArea</i> | 150 |
| 3.50 | Link <i>Embedded Page</i> dialog for <i>PageArea</i> s. | 150 |
| 3.51 | Warning when pages are larger than the defined <i>CBA Presentation Size</i> | 151 |
| 3.52 | Empty <i>Tasks</i> - view of the CBA ItemBuilder. | 152 |
| 3.53 | <i>Tasks</i> - view showing a complete <i>Task</i> -definition. | 153 |
| 3.54 | Dialog <i>Execution layout settings</i> to define <i>xPage</i> -layouts. | 154 |
| 3.55 | Item illustrating an <i>xPage</i> -layout with enabled slider (html ib). . . | 155 |
| 3.56 | Entry <i>Global Properties</i> in the <i>Context Menu</i> of the project name in the <i>Project View</i> | 156 |
| 3.57 | <i>Project Settings</i> to change the <i>CBA Presentation Size</i> | 156 |
| 3.58 | Schema for components of the <i>CBA Presentation Size</i> | 157 |
| 3.59 | Example for navigation with <i>Runtime Commands</i> (html ib). | 158 |
| 3.60 | <i>Main Menu Edit</i> showing <i>Undo</i> and <i>Redo</i> | 160 |

| | | |
|------|---|-----|
| 3.61 | Section Position of the <i>Properties</i> view. | 161 |
| 3.62 | Context menu for a selection in the <i>Page Editor</i> showing the entry <i>Duplicate</i> | 163 |
| 3.63 | <i>View Clipboard</i> illustrating the visualization of elements in the clipboard. | 164 |
| 3.64 | <i>Component Edit</i> view allows to select any component in the <i>Page Editor</i> | 165 |
| 3.65 | <i>Properties</i> view section Identification to define a <i>UserDefinedId</i> . . . | 166 |
| 3.66 | Message when a provided <i>User Defined Id</i> is not unique. | 167 |
| 3.67 | Dialog informing requesting to close all editors before the requested operation can be executed. | 167 |
| 3.68 | <i>Main Menu Project</i> showing the entry <i>Edit all User Defined IDs</i> . . | 168 |
| 3.69 | Dialog to <i>Edit User Defined ID</i> of a <i>Project File</i> | 168 |
| 3.70 | Item illustrating components of type <i>Line</i> and <i>Rectangle</i> (html ib). . | 170 |
| 3.71 | Context menu for design-time <i>Z-order</i> in the <i>Page Editor</i> | 170 |
| 3.72 | Item illustrating the use of custom <i>Cursors</i> (html ib). | 171 |
| 3.73 | Context menu for defining the <i>Cursor</i> in the <i>Page Editor</i> | 171 |
| 3.74 | Dialog <i>Set Cursor</i> | 171 |
| 3.75 | Item illustrating the use of <i>Tab Index</i> (html ib). | 172 |
| 3.76 | Overview of components to display text. | 173 |
| 3.77 | Section <i>Appearance</i> in the <i>Properties</i> view to define <i>Fonts and Main Colors</i> | 174 |
| 3.78 | Example illustrating components to display text (html ib). | 174 |
| 3.79 | <i>Input Source Configuration Editor</i> to define the <i>Input Source</i> of <i>SimpleTextFields</i> | 174 |
| 3.80 | Item illustrating options to define <i>Input Source</i> for <i>SimpleTextFields</i> (html ib). | 175 |
| 3.81 | Context menu entries to link components with the option <i>Input Field</i> . . | 175 |
| 3.82 | <i>HTML Text Editor</i> for the content of <i>HTMLTextField</i> components. . . | 176 |
| 3.83 | Dialog to <i>Configure Embedded Link</i> in <i>HTMLTextFields</i> | 177 |
| 3.84 | Example illustrating the use of text highlighting (html ib). | 178 |
| 3.85 | Example illustrating the use of text <i>MathJax</i> (html ib). | 179 |
| 3.86 | Illustration of line breaks in text. | 179 |

| | |
|--|-----|
| 3.87 Overview of components to collect responses. | 181 |
| 3.88 Example illustrating components to collect text responses (html ib). | 181 |
| 3.89 Filter Show Advanced Properties of the <i>Properties</i> view. | 183 |
| 3.90 Figure 1 from Moon et al. (2019) illustrating different item formats (html ib). | 184 |
| 3.91 Item illustrating <i>RadioButtons</i> (html ib). | 185 |
| 3.92 Context menu to <i>Configure Select Groups</i> in the <i>Component Edit</i> | 188 |
| 3.93 <i>Configure Select Groups</i> dialog to define <i>Frame Select Groups</i> | 189 |
| 3.94 Assignment of a component to a <i>Frame Select Group</i> in the <i>Properties-view</i> | 189 |
| 3.95 Item illustrating the use of <i>Frame Select Groups</i> (html ib). | 189 |
| 3.96 Item illustrating <i>ComboBoxes</i> and <i>Lists</i> (html ib). | 190 |
| 3.97 Context menu for <i>ComboBoxes</i> in the <i>Page Editor</i> | 191 |
| 3.98 Dialog to configure a component of type <i>ComboBoxItem</i> | 191 |
| 3.99 Context menu for <i>ComboboxItems</i> in the <i>Component Edit</i> | 192 |
| 3.100 Advanced properties for <i>Panels</i> in the <i>Properties-view</i> to define <i>selectable: true</i> | 193 |
| 3.101 Item illustrating the <i>setInputValue()</i> -operator (html ib). | 193 |
| 3.102 Item illustrating components of type <i>MenuBar</i> and <i>Menu</i> (html ib). | 194 |
| 3.103 Overview of components for special purposes. | 195 |
| 3.104 Item illustrating <i>Tables</i> (html ib). | 196 |
| 3.105 Dialog to configure <i>Table</i> -components. | 196 |
| 3.106 Item illustrating <i>Trees</i> to collect click-responses (html ib). | 197 |
| 3.107 Configuration of <i>Tree</i> -component in the <i>Component Edit-view</i> | 198 |
| 3.108 Context menu for <i>[Nodes]</i> in the <i>Component Edit-view</i> of a <i>Tree</i> -component. | 198 |
| 3.109 Edit values for <i>[Nodes]</i> shown in the <i>TreeView</i> -component. | 199 |
| 3.110 Item illustrating 'ImageMaps' (html ib). | 200 |
| 3.111 Item illustrating advanced <i>ImageMaps</i> (html ib). | 203 |
| 3.112 Item illustrating the use of images (html ib). | 204 |
| 3.113 <i>Rendering</i> (left) and <i>Page Editor</i> (middle) and <i>Properties</i> view showing item <i>ImageExamples.zip</i> (html ib) | 204 |

| | |
|---|-----|
| 3.114 Icon <code>Browse</code> resources in the CBA <code>ItemBuilder Toolbar</code> | 206 |
| 3.115 <i>Resource Browser</i> to manage embedded resources. | 206 |
| 3.116 Item illustrating the use of different image file formats (html ib). . . | 207 |
| 3.117 Item illustrating the use of different audio file formats (html ib). . . | 208 |
| 3.118 Item illustrating the use of different video file formats (html ib). . . | 208 |
| 3.119 Item illustrating the difference scaling of <code>ImageFields</code> and images as background in <code>Panels</code> (html ib). | 210 |
| 3.120 <i>Link Audio</i> dialog for <code>Audio</code> components. | 210 |
| 3.121 Item illustrating the use properties <code>Hide Controls</code> and <code>Max Play of</code> <code>Audio</code> and <code>Video</code> components (html ib). | 211 |
| 3.122 <i>Set Media Raised Events</i> dialog for <code>Audio</code> components. | 212 |
| 3.123 Item illustrating audio recording using the <code>Audio</code> component (html ib). | 213 |
| 3.124 Item illustrating audio recording using the <code>Video</code> component (html ib). | 214 |
| 3.125 Example for <i>Links</i> and <i>Conditional Links</i> (html ib). | 215 |
| 3.126 Schematic display of links in Figure 3.125. | 215 |
| 3.127 Example for components of type <code>Link</code> (html ib). | 217 |
| 3.128 <i>Context Menu</i> for components of type <code>Link</code> | 217 |
| 3.129 Dialog <i>Configure a Multiline Text</i> | 218 |
| 3.130 Dialog <i>Link Page</i> used for components of different type. | 218 |
| 3.131 Item illustrating the component <code>Button</code> (html ib). | 219 |
| 3.132 <i>Context-Menu</i> for components of type <code>Button</code> | 220 |
| 3.133 Component of type <code>Button</code> configured as <i>Standard Button</i> | 220 |
| 3.134 Component of type <code>Button</code> configured as <i>Image Button</i> | 221 |
| 3.135 Dialog <i>Set Background Color</i> for <code>Buttons</code> (<i>Standard Buttons</i>). | 221 |
| 3.136 Item illustrating links in different components (html ib). | 223 |
| 3.137 Item illustrating links with <i>xPage</i> -layouts (html ib). | 224 |
| 3.138 Context menu to assign a <i>Runtime Command</i> | 226 |
| 3.139 Dialog <i>Set Runtime Command</i> | 226 |
| 3.140 Property <i>Command</i> in section <i>Component Interaction</i> | 227 |

| | | |
|-------|---|-----|
| 3.141 | Preview message when calling the <code>NEXT_TASK</code> -command in the last Task. | 228 |
| 3.142 | Item illustrating Task-related Commands (html ib). | 228 |
| 3.143 | Item illustrating fullscreen-related Commands (html ib). | 229 |
| 3.144 | Item illustrating clipboard-related Commands (html ib). | 230 |
| 3.145 | Hierarchy of components for different Page Types. | 232 |
| 3.146 | Item illustrating page type <code>TabFolder</code> (html ib). | 233 |
| 3.147 | Item illustrating page type <code>Taskbar</code> (html ib). | 233 |
| 3.148 | Item illustrating page types <code>WebBrowser</code> and <code>WebChild</code> (html ib). | 234 |
| 3.149 | Overview of components for web browser. | 235 |
| 3.150 | Item illustrating 'xPage'-layout with browser pages (html ib). | 235 |
| 3.151 | Context menu of <code>ExternalPageFrame</code> -components. | 236 |
| 3.152 | Dialog <i>Set URL</i> to configure <code>ExternalPageFrames</code> | 237 |
| 3.153 | Context menu of the <i>Embedded HTML Explorer</i> | 238 |
| 3.154 | HTML document opened in an <i>HTML Editor</i> | 238 |
| 3.155 | Warning shown by the <i>Embedded HTML Explorer</i> | 238 |
| 3.156 | Item illustrating dialog pages (html ib). | 241 |
| 3.157 | Definition of a <code>Frame</code> as <code>DIALOG</code> in the <i>Properties</i> view. | 241 |
| 3.158 | Item illustrating modal dialogs (html ib). | 242 |
| 3.159 | Item illustrating dialog pages with links (html ib). | 244 |
| 4.1 | Example for a hint button disabled using states (html ib). | 246 |
| 4.2 | Screenshots of the <i>State Machine Debug Window</i> (html ib). | 246 |
| 4.3 | Screenshot of the <i>Auto-Completion</i> feature in syntax editors. | 249 |
| 4.4 | Editor for <i>Variables</i> | 253 |
| 4.5 | <i>Set Variable attributes</i> dialog. | 254 |
| 4.6 | <i>Set Named Value attributes</i> dialog. | 254 |
| 4.7 | Item illustrating the use of variables and value inputs (html ib). | 256 |
| 4.8 | Context menu in the <i>Page Editor</i> to <i>Link Variables</i> | 256 |
| 4.9 | Dialog for assigning a <code>Button</code> to a <code>ScaleValueInput</code> | 256 |
| 4.10 | Item illustrating layout option for <code>VariableValueDisplays</code> (html ib). | 257 |

| | | |
|------|---|-----|
| 4.11 | Editor for <i>Value Maps</i> | 258 |
| 4.12 | Dialog <i>Set value map detail entry attributes</i> | 258 |
| 4.13 | Dialog for defining <i>Guards</i> using the dialog <i>Set value for column Guard</i> | 259 |
| 4.14 | Item illustrating different variable inputs (html ib). | 259 |
| 4.15 | Property <i>Value Display Type</i> for components of type <code>MapBasedVariableDisplay</code> in the <i>Properties</i> view. | 260 |
| 4.16 | Item illustrating layout option for <code>MapBasedVariableValueDisplay</code> (html ib). | 260 |
| 4.17 | Item 1 from Jiang et al. (2021) illustrating Drag-and-Drop (html ib). | 261 |
| 4.18 | Item 2 from Jiang et al. (2021) illustrating Drag-and-Drop (html ib). | 262 |
| 4.19 | Drag-and-drop example illustrating the use of variables (html ib). | 262 |
| 4.20 | Item illustrating free drag and drop (html ib). | 263 |
| 4.21 | Item illustrating drag and drop modes (html ib). | 263 |
| 4.22 | Item illustrating drag and drop groups (html ib). | 264 |
| 4.23 | Example for <i>Dynamic Text</i> in <code>HTMLTextFields</code> (html ib). | 265 |
| 4.24 | Item illustrating the use of <i>Conditional Links</i> (html ib). | 267 |
| 4.25 | Dialog <i>Link Page</i> with buttons <i>Edit Condition</i> and <i>Drop Condition</i> | 268 |
| 4.26 | Item with contextualized examples of <i>Conditional Links</i> (html ib). | 268 |
| 4.27 | Item illustrating the advanced use of <i>Conditional Links</i> (html ib). | 270 |
| 4.28 | Item illustrating the use of <i>Conditional Links</i> with <i>Operators</i> (html ib). | 279 |
| 4.29 | Item illustrating the <code>setEmbeddedPage()</code> -operator (html ib). | 280 |
| 4.30 | Empty <i>State Machine Tree View</i> of the CBA <i>ItemBuilder</i> | 282 |
| 4.31 | Main menu <i>Statemachine Editor</i> and context menu in the <i>State Machine Tree View</i> to define a state | 283 |
| 4.32 | Newly created <i>State</i> and <i>Configure State</i> dialog. | 284 |
| 4.33 | <i>State Machine Tree View</i> with four states and <i>Properties</i> view. | 285 |
| 4.34 | <i>Link Raised Event</i> in the context menu of the <i>Page Editor</i> | 287 |
| 4.35 | Item illustrating events triggered by user interactions with components (html ib). | 288 |
| 4.36 | Item illustrating timed events (html ib). | 289 |
| 4.37 | Item illustrating different <i>Start Rules</i> for projects with multiple <i>Tasks</i> (html ib). | 290 |

| | | |
|------|--|-----|
| 4.38 | Item illustrating simple <i>Transitions</i> (html ib). | 293 |
| 4.39 | Item illustrating a delayed activation of a button using a timed event and the <code>unsetFrozen()</code> -operator (html ib). | 296 |
| 4.40 | Item illustrating contextual dependency of events (html ib). | 297 |
| 4.41 | Item illustrating the combination of condition in <i>Rules</i> (html ib). | 299 |
| 4.42 | Item illustrating the operators <code>set()</code> and <code>reset()</code> (html ib). | 299 |
| 4.43 | Item illustrating the operators <code>setFrozen()</code> and <code>unsetFrozen()</code> (html ib). | 300 |
| 4.44 | Item illustrating the <code>setHidden()</code> - / <code>unsetHidden()</code> -operator (html ib). | 303 |
| 4.45 | Item illustrating the <code>focus()</code> -operator (html ib). | 303 |
| 4.46 | Item illustrating the <code>insertText()</code> -operator (html ib). | 304 |
| 4.47 | Item illustrating the <code>setActive()</code> -/ <code>unsetActive()</code> -operator (html ib). | 305 |
| 4.48 | Item illustrating operators for text highlighting (html ib). | 306 |
| 4.49 | Item illustrating the <code>setInputValue()</code> -operator (html ib). | 307 |
| 4.50 | Item illustrating the Operators for <code>Frame Select Groups</code> (html ib). | 307 |
| 4.51 | Example for the task-related operators (html ib). | 309 |
| 4.52 | Example for the <code>openDialog()</code> and <code>closeDialog()</code> -operators (html ib). | 310 |
| 4.53 | Example for the <code>scrollEmbeddedPage()</code> -operator (html ib). | 311 |
| 4.54 | Item illustrating restriction to play media components (html ib). | 311 |
| 4.55 | Item illustrating operators for media components (html ib). | 312 |
| 4.56 | Example for Calculator-operators (html ib). | 313 |
| 4.57 | Example for <code>setEmbeddedPage()</code> -operator (html ib). | 314 |
| 4.58 | Example for <code>raise()</code> and <code>initFSM()</code> -operator (html ib). | 315 |
| 4.59 | Example for <code>raise()</code> and <code>initFSM()</code> -operator (html ib). | 316 |
| 4.60 | Example illustrating the <code>elapsedTime()</code> -operator (html ib). | 318 |
| 4.61 | Example for transitions <code>internal</code> , <code>entry</code> and <code>exit</code> (html ib). | 320 |
| 4.62 | Example for multiple nested <i>Finite-State Machines</i> (html ib). | 321 |
| 4.63 | Item illustrating <i>Timed Events</i> and <i>Regions</i> (html ib). | 321 |
| 4.64 | Example for navigation with states (html ib). | 322 |
| 4.65 | Context menu <i>Configure State</i> to assign a <i>Page</i> to a <i>State</i> . | 322 |
| 4.66 | Dialog <i>Configure State</i> showing <i>Page to open</i> to assign a <i>Page</i> to a <i>State</i> . | 323 |

| | |
|---|-----|
| 4.67 States <i>Linked to Pages</i> in two <i>Regions</i> as implemented in the item shown in Figure 3.137. | 323 |
| 4.68 Example for changing <i>Page</i> and <i>X-Pages</i> simultaneously (html ib). | 325 |
| 4.69 Example for a timed presentation using states and timed events (html ib). | 325 |
| 4.70 Item illustrating component of type <code>Timer</code> (html ib). | 326 |
| 4.71 Item illustrating component of type <code>Timer</code> (html ib). | 327 |
| 4.72 Item illustrating <i>Task Initialization Syntax</i> (html ib). | 328 |
| 4.73 Template for content developed for <code>ExternalPageFrames</code> (html ib). | 332 |
| 4.74 Item Illustrating the API for <code>ExternalPageFrames</code> (html ib). | 333 |
| 4.75 Example Item Illustrating JavaScript calls from FSM (html ib). | 336 |
| 4.76 Example Item Illustrating JavaScript calls for Persistence (html ib). | 338 |
| 5.1 <code>UserDefinedIds</code> defined for example item shown in Figure 5.2. | 341 |
| 5.2 Example for scoring a multiple choice item (html ib). | 342 |
| 5.3 <i>Task-Editor</i> with one <i>Task</i> and three <i>Hit</i> -conditions for the item shown in Figure 5.2. | 342 |
| 5.4 <i>Class Definition Dialog</i> for the item shown in Figure 5.2. | 343 |
| 5.5 <i>Condition Syntax</i> for the <i>Hit</i> -condition <code>Correct</code> of the item shown in Figure 5.2. | 343 |
| 5.6 Screenshot of the <i>Scoring Debug Window</i> in a preview of the item shown in Figure 5.2. | 344 |
| 5.7 Example for scoring a Likert-style item into two variables (html ib). | 345 |
| 5.8 Item illustrating scoring with <code>result_text()</code> -operator (html ib). | 346 |
| 5.9 Example item illustrating scoring with variables (html ib). | 347 |
| 5.10 Hit definition with logical expressions (<code>and</code> , <code>or</code> , <code>not</code> ; html ib). | 349 |
| 5.11 Different <i>Hit</i> - definitions using the <code>matches()</code> -operator (html ib). | 351 |
| 5.12 Use of <i>FSM-Variables</i> in <i>Scoring-Conditions</i> with the <code>variable_in()</code> -operator (html ib). | 352 |
| 5.13 Using <i>set of values</i> and <code>visited_all_values_of_variable()</code> -operator in <i>Scoring-Conditions</i> (html ib). | 353 |
| 5.14 Example for scoring free drag and drop using the <code>panel_position_range()</code> -operator (html ib). | 354 |
| 5.15 Example for using events for scoring (html ib). | 356 |

| | | |
|------|--|-----|
| 5.16 | Item illustrating scoring when <i>PageAreas</i> are used (html ib). | 360 |
| 5.17 | Item illustrating <i>Missing Value Coding</i> for a multi-paged item (html ib). | 361 |
| 6.1 | Example item illustrating scoring with regular expressions (html ib). | 368 |
| 6.2 | Example item illustrating the different approaches to check empty input (html ib). | 369 |
| 6.3 | Property Input Validation Pattern in the <i>Properties</i> -view. | 370 |
| 6.4 | Item illustrating different Input Validation Pattern (html ib). | 371 |
| 6.5 | Item illustrating the use of Input Validation Events (html ib). | 372 |
| 6.6 | Example item illustrating transparent images in the PNG-format (html ib). | 374 |
| 6.7 | Example item illustrating ImageFields with alpha transparency and different Z-Order (html ib). | 374 |
| 6.8 | Example showing images resized to different sizes (html ib). | 375 |
| 6.9 | Tabs <i>Translation</i> and <i>Icons</i> in the <i>Global Properties</i> of CBA ItemBuilder Project Files. | 378 |
| 6.10 | Example item illustrating the use of CSS Styles (html ib). | 379 |
| 6.11 | Tab <i>Meta-Data</i> in the <i>Global Properties</i> of CBA ItemBuilder Project Files. | 380 |
| 6.12 | Example for sequence of audio files with multiple states (html ib). | 383 |
| 6.13 | Item illustrating <i>with-task</i> adaptivity using <i>Conditional Links</i> (html ib). | 384 |
| 6.14 | Adaptivity in the <i>with-task</i> adaptivity example shown in Figure 6.13. | 384 |
| 6.15 | Example item illustrating the use of <code>setHidden()</code> / <code>unsetHidden()</code> -operators in <i>Conditional Links</i> (html ib). | 386 |
| 6.16 | Example item illustrating the different ways to show additional information (html ib). | 386 |
| 6.17 | Example item illustrating a time limit for multiple tasks (html ib). | 387 |
| 6.18 | Example item illustrating the different navigation restrictions (html ib). | 388 |
| 6.19 | Example item illustrating an interrupted video with embedded questions (html ib). | 389 |
| 6.20 | Example for using operators in <i>Conditional Links</i> to align <i>Single-LineInputFields</i> and <i>RadioButtons</i> (html ib). | 390 |
| 6.21 | Example for contextualized multiple-choice with images (html ib). | 391 |

| | | |
|------|--|-----|
| 6.22 | Example for shuffle response options with <code>ValueMaps</code> (html ib). | 392 |
| 6.23 | Example for a calculator based on 'Finite-State Machine'. | 394 |
| 6.24 | Example for <code>ExternalPageFrame</code> with calculator. (html ib). | 396 |
| 6.25 | Continuous Audio Play with <code>ExternalPageFrames</code> (html ib). | 397 |
| 6.26 | Using <code>math-field</code> (from <i>Mathlive</i>) within <code>ExternalPageFrames</code> (html ib). | 398 |
| 6.27 | <code>CKEditor</code> within <code>ExternalPageFrames</code> (html ib). | 398 |
| 6.28 | <code>TextArea</code> within <code>ExternalPageFrames</code> (html ib). | 399 |
| 6.29 | Browser-based Speech Recognition using <code>Vosk</code> and <code>ExternalPageFrames</code> (html ib). | 399 |
| 6.30 | Server-based Speech Recognition using <code>window.SpeechRecognition</code> and <code>ExternalPageFrames</code> (html ib). | 400 |
| 6.31 | Reading Task using Server-based Speech Recognition and <code>ExternalPageFrames</code> (html ib). | 400 |
| 6.32 | Example for the integration of H5P content in an 'ExternalPageFrame'. | 401 |
| 6.33 | Integration of GeoGebra content in a <code>ExternalPageFrame</code> (html ib). | 402 |
| 6.34 | Integration of QTI content using <code>QTI.js</code> and <code>ExternalPageFrames</code> (html ib). | 403 |
| 6.35 | Integration of Questionnaires using <code>SurveyJS</code> and <code>ExternalPageFrames</code> (html ib). | 404 |
| 6.36 | Screenshot of the Embedded HTML Explorer using <code>SurveyJS</code> content. | 404 |
| 6.37 | Example adaptive number series test (html ib). | 406 |
| 6.38 | Item illustrating Window Management in CBA <code>ItemBuilder</code> (html ib). | 408 |
| 6.39 | 'CBA Item Fonts' setting of the CBA <code>ItemBuilder</code> . | 410 |
| 6.40 | Color selector (<i>Choose Color</i>) of the CBA <code>ItemBuilder</code> allows storing <i>Recent-colors</i> . | 410 |
| 6.41 | Item for Color Conversion of CBA <code>ItemBuilder</code> 's internal Color Representation (html ib). | 411 |
| 6.42 | CBA <code>ItemBuilder</code> Error Messages (html ib). | 411 |
| 6.43 | Item illustrating overlapping components in CBA <code>ItemBuilder</code> (html ib). | 413 |
| 6.44 | Context Menu in the <i>Project Tree</i> . | 415 |

| | | |
|------|---|-----|
| 6.45 | Dialog to specify the name of a <i>Template</i> | 416 |
| 6.46 | Dialog to specify the name of a <i>Template</i> | 416 |
| 6.47 | Example screen shot running two CBA ItemBuilder in parallel. . . | 417 |
| 7.1 | Output of an Adaptive Test Created with ShinyItemBuilder and catR. . | 434 |
| 7.2 | Repository fastib2pci shows the button Use this template after log-in to github.com. | 437 |
| 7.3 | Example repository settings showing activated <i>GitHub Pages</i> | 439 |
| 8.1 | Illustration of an <i>Assessment Cycle</i> that includes <i>Transfer</i> | 455 |
| 8.2 | Example Illustrating the Distribution of Items to Tasks (html ib). . | 464 |
| 8.3 | SVN: Repository and Working Copies (Mason, 2006). | 467 |
| 8.4 | GIT: Remote Repository and Local Repositories. | 471 |
| 8.5 | Lifecycle of file satus in git. | 472 |
| B.1 | Example item illustrating <i>Raw Log-Events</i> (html ib). | 538 |

List of Tables

| | | |
|------|--|-----|
| 3.1 | Example for <i>Hit</i> -definitions of a single-choice item | 120 |
| 3.2 | Example for ‘Hit’-definitions of a multiple-choice item | 123 |
| 3.3 | Example for ‘Hit’-definitions of a text-entry item | 127 |
| 3.4 | Basic page type <i>Simple Page</i> of the CBA ItemBuilder and <i>xPages</i> . . | 134 |
| 3.5 | Settings for <i>xPage</i> layouts | 155 |
| 3.6 | Visual feedback (mouse pointer) in the <i>Page Editor</i> | 160 |
| 3.7 | Comparison of components to display text | 179 |
| 3.8 | Guide for creating <i>ImageMaps</i> in the CBA ItemBuilder | 200 |
| 3.9 | Supported File Formats for Resources | 205 |
| 3.10 | Overview of advanced page types of the CBA ItemBuilder | 231 |
| 4.2 | Example for ‘ValueMap’-definition ‘M_Feedback’ | 277 |
| 4.3 | Example for ‘ValueMap’-definition ‘M_Frequency’ | 277 |
| 7.1 | Data Reported by all <i>SCORM</i> Components with CBA ItemBuilder Tasks | 447 |
| 7.2 | Data Reported by <i>SCORM</i> Components With Scoring | 448 |
| 8.1 | Workflow for <i>Overall Planing and Preparation</i> | 457 |
| 8.2 | Workflow for Item Development | 459 |
| 8.3 | Steps for Test Design and Assembly | 462 |
| 8.4 | Content of CBA ItemBuilder <i>Project Files</i> required at <i>Runtime</i> | 474 |
| 8.5 | Content of CBA ItemBuilder <i>Project Files</i> required at <i>Design-Time</i> . . | 474 |
| 8.6 | Steps for Testing | 477 |
| 8.7 | Steps for Test Administration / Data Collection | 481 |
| 8.8 | Steps for Data Preparation / Reporting / Feedback | 481 |

| | |
|---|-----|
| 8.9 Steps for Documentation | 485 |
| A.1 Glossary of (Technical) Terms used in this Book | 497 |
| B.1 Overview of the File menu. | 508 |
| B.2 Overview of the Edit menu | 508 |
| B.3 Overview of the Diagram menu | 509 |
| B.4 Overview of the Project menu | 510 |
| B.5 Overview of the Template menu | 511 |
| B.6 Overview of the Utilities menu | 511 |
| B.7 Overview of the Help menu | 511 |
| B.8 Operators for <i>State Machine Variables</i> | 513 |
| B.9 Operators for <i>Component States</i> | 515 |
| B.10 Operators for <i>Trees</i> | 517 |
| B.11 Operators for <i>Evaluations</i> | 518 |
| B.12 Finite-state machine operators for <i>Task Management</i> | 522 |
| B.13 <i>Logical</i> expressions in the domain specific language (DSL) | 523 |
| B.15 <i>Comparisons</i> in the domain specific language (DSL) | 526 |
| B.16 <i>Arithmetics</i> in the domain specific language (DSL) | 527 |
| B.17 Miscellaneous operators in the domain specific language (DSL) | 527 |
| B.18 Regular Expression Symbols | 529 |
| B.21 CBA ItemBuilder Versions (with React-based runtime) | 532 |
| B.22 Register of all Components | 534 |
| B.23 Log events for basic elements | 539 |
| B.24 Log events for component selection | 540 |
| B.25 Log events for answer-changes | 540 |
| B.26 Log events for text entry components | 541 |
| B.27 Log events for value inputs | 542 |
| B.28 Log events for audio/video components | 543 |
| B.29 Log events for tree components | 544 |
| B.30 Log events for advanced components | 545 |
| B.31 Log events for text highlighting | 547 |
| B.32 Log events for test deployment | 547 |

Preface

This book describes the CBA ItemBuilder, a tool implemented over the last years for developing and delivering complex items and tasks. The CBA ItemBuilder is a development environment for assessments enabling researchers, domain experts, and teachers to develop complex items without requiring any specific programming skills. Possible item types include but are not limited to: web environment simulations (including browser, search engines, etc.), desktop application(s) simulation and simulations using finite-state machines. The CBA ItemBuilder has been used in several national and international projects and is constantly being improved. It is available free of charge for non-commercial projects. This book describes how to use the CBA ItemBuilder for different user groups involved in the use of technology-based assessments as part of Open Educational Resources.



The latest version can be found online as [pdf](#), [epub](#) or as [html](#) at [github](#).

Suggested Citation:

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

Why read this book



This book is for item authors and researchers preparing and organizing assessments or analyzing the result data or log data from computer-based assessments.

For the following use-cases we have created section that allow to reach a particular goal without the need to read the entire book:

- Quick start section 1.4 describes how CBA ItemBuilder project files that contain one or multiple tasks with single items or units can be previewd. If you are new to computer-based task authoring tools, you can download existing ItemBuilder projects, open them in the CBA ItemBuilder and use the preview function to view the items ‘live’. This allows, for instance, item reviewers to use the CBA ItemBuilder to view (and review) items.
- Quick start section 1.5 provides the necessary information to explore the item scores, that are created by a particular task defined in an CBA ItemBuilder project file. This allows, for instance, domain experts to verify the scoring of items, that is to relate test-taker’s responses to result variables provided by the items.
- Quick start section 1.6 shows how identifiers that can be found in (log) data can be linked to parts of the instrument, as it is required for scientists after obtaining data collected with the CBA ItemBuilder. This allows, for intsance, using *mock items* to explore interactive computer-based tasks.
- Quick start section 1.7 explains how to *revise* existing items (e.g., adapt, adjust, modify or improve content). The ability to adapt the content of computer-based items without having to consult with the programmers or designers of the item material is one of the important prerequisites for open computer-based assessments.
- Quick start section 1.8 summarizes how programmers can integrate content generated with the CBA ItemBuilder into their Web application. Moreover, it is explained how HTML content can be used as part of CBA ItemBuilder projects.¹
- Quick start section 3.3 gives hands-on instruction how to create simple single-page items using the CBA ItemBuilder. This section is intended for users who want to familiarize themselves with the CBA ItemBuilder while using the software.
- Quick start section 7.1 provides an hands-on example how to use the assessment components created in section 3.3 to realize a typical offline deployment.

All quick start sections are cross-referenced to the relevant sections in the main text, allowing readers to dive deeper into the subject matter if needed.

¹The CBA ItemBuilder is software designed for item authors who want to use or create components for computer-based assessment. Therefore, the CBA ItemBuilder is not a tool for programmers who can create web-based assessments with their development environments without the CBA ItemBuilder.

Structure of the book



For researchers, teachers, lecturers, content experts, students and Ph.D. students, student assistants, and interested readers who want to go deeper into the use of the CBA ItemBuilder, the book starts with a short primer on computer-based assessments in chapter 2.

Building on the objectives and some basic vocabulary, chapter 3 then describe the implementation of assessment components using static components provided by the CBA ItemBuilder. The static content mainly covers the visible elements of items, instruction pages, etc., and includes the development of items using multiple pages and dialogs.

If the displayed task content should change either time-controlled or in reaction to observed test-taker behavior, the items become *dynamic*. Chapter 4 deals with the enrichment of assessment components, such as items, with dynamic content.

Chapter 5 of this book is then devoted in detail to the evaluation of answers, the so-called *Scoring* of computer-based items.

For users who would like to plan and create new assessments, chapter 6 provides further tips and suggested solutions for practical challenges in creating computer-based assessments using the CBA ItemBuilder.

Chapter 7 summarizes possibilities to combine single CBA ItemBuilder project files to complete assessments and introduces software tools and platforms to deliver assessments in practice.

Chapter 8 finally deals with workflows, i.e., typical processes for preparing, testing, delivering, and archiving computer-based assessments.

The book concludes with a discusses of the CBA ItemBuilder as a tool in the context of *open science* and *reproducible research*.

Contributors



Starting with version 9.0 of the CBA ItemBuilder, item-projects created with this authoring tool can be flexibly integrated into various HTML-based execution environments without the need for special server components. I have taken this milestone as an opportunity to combine the existing documentation and the CBA ItemBuilder Wiki in a new user book-length documentation.

This book would not be possible without the support of

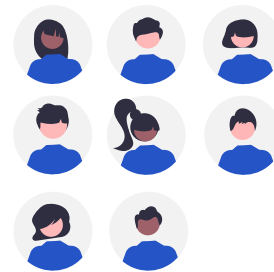
- all previous and current CBA ItemBuilder users,²
- Frank Goldhammer (Center for Technology-based Assessment), Eckhard Klieme (retired Director of the department of Educational Quality and Evaluation and Research Fellow at DIPF) and Marc Rittberger (Director of the Information Center for Education),
- Robert Baumann, Ingow Barkow, Paul Libbrecht and Daniel Schiffner (IT engineers at the Center for Technology-Based Assessment),
- Margit Mikula, Marisa Herrmann, Gabriele Gissler and Britta Upsing (CBA Item author support and early users of the CBA ItemBuilder at the Center for Technology-Based Assessment),
- Rachel Ghebrehawariat and Angelika Sichma (project assistants at the Center for Technology-Based Assessment),
- all former and current doctoral students at the Center for Technology-Based Assessment,
- student assistants at the Center for Technology-Based Assessment,
- and many others.

The idea of creating an authoring tool for the implementation of complex interactive items goes back to Jean Paul Reef and was supervised conceptually and organizationally by Heiko Rölke in the first years at the Center for Technology-based Assessment at the DIPF (Rölke, 2012).

²You can contribute to the book by making edits and pull requests on [github](#).

Special thanks go to the long-standing cooperation partners Softcon GmbH / nara-gro AG (Michel Dorochevsky, Constantin von Kirschten and the entire team).

Acknowledgments



For the development of the CBA ItemBuilder, in addition to the programmers and users, special thanks must also be given to the organizations and projects that have used and funded the development.



1

Introduction



Computer-based assessment (CBA)¹ has gained importance in various research fields that investigate human education, cognition, affect, experience, and human behavior. The success of CBA is due to many advantages for testing in general and for measuring competencies and skills in particular (see chapter 2). However, the development of computer-based tests requires either programming skills or software tools. Various successful software tools exist for computer-based surveys and questionnaires (e.g., [limesurvey](#)), for *IMS QTI*-based tests (e.g., [TAO](#)) and for interactive e-learning content (e.g., [H5P](#)). Data collection can also be implemented using software for electronic data capture (e.g., [REDcap](#)), for psychological experiments (e.g., [PsychoPy](#) or [Open Sesame](#)) or using the open source statistical software environment R (e.g., [Concerto](#), [formr](#) or even [Shiny](#)). The type of assessments that can be created is often limited by the software tools or the developers' resources and programming skills.

However, the implementation of CBA can be laborious and expensive. As a result, applications using CBA make less use of the possibilities of the existing technology, and *how* constructs are measured is affected by the available features of existing tools. Moreover, when content experts write items on paper and technical experts implement the assessment material on digital platforms, it becomes more challenging to unlock the diagnostic potential of CBA.

CBA ItemBuilder: In this book we will introduce the *CBA ItemBuilder*, an authoring tool for computer-based assessment that enables content experts to become test authors and to design different item types with a graphical editor. The CBA ItemBuilder is created for persons with no, or with no specific programming experi-

¹Synonyms are computer-based testing (CBT), technology-based assessment (TBA) or technology-based testing (TBT).

ences. Researchers, (PhD)-students, content experts and teachers with no specific background in web development can use the CBA ItemBuilder to create and implement complex items, after following some of the quick start tutorials provided in this book, and after exploring some of the examples in the CBA ItemBuilder.

The CBA ItemBuilder, as a specialized development environment for assessment material, provides a graphical page designer. The CBA ItemBuilder allows creating items with multiple pages by placing elements (such as texts, radio buttons, buttons, etc.) to specific locations in a visual editor (called *Page Editor*, see section 3.1.3).²



The CBA ItemBuilder is an open-source tool used in various contexts to create assessment content. It is particularly suitable for implementing complex interactive item formats (so-called technology-enhanced items, see section 2.3) but can also be utilized for simple item types. The content created with the CBA ItemBuilder can be used together with various tools, such as TAO (see section 7.4).

Item Authors: Item authors, i.e., content experts or researchers, can use this graphical editor to design one or several items by creating pages with images, texts or videos, implementing special features for test-takers like timers or instant feedback, defining rules for (automatic) scoring and create complex items with dynamic behavior (see section 2.11.1 for a more detailed discussion of the possible division of labor for the creation of computer-based assessment content by item developers).

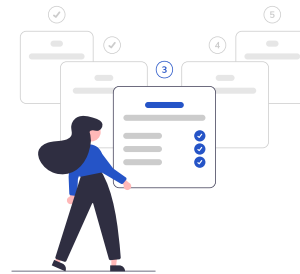
The available features of the CBA ItemBuilder are permanently extended and updated. Features described in this book match [version 10.0.0](#) of the CBA ItemBuilder and covers many of the features available in this version.³

Software Developers: The CBA ItemBuilder supports not all options to create interactive items. However, the lack of functionalities or features does not necessarily mean that the CBA ItemBuilder can not be used for large parts of creating computer-based assessment content. Instead, software developers can be involved in implementing the missing functionality as an *HTML5/JavaScript*. *HTML5/JavaScript* can be integrated and used via *iframes* (or in the component model of the CBA ItemBuilder *ExternalPageFrames*). Moreover, items generated with the CBA ItemBuilder can also be embedded into other environments that render content as *HTML5/JavaScript*. Section 1.8 summarizes the important things developers need to know about these two use cases.

²The *Page Editor* is not yet WYSIWYG (i.e., “What you see is what you get”), but gives a rough visual impression of the page content, and the internal *Renderer*-view (see section [@ref\(ui-project-view-component-edit-embedded-html-explorer\)](#)) and the (external) *Preview* show the final layout (see section [1.4.2](#)).

³For further information about the different versions of the CBA ItemBuilder, see appendix [B.5](#). Note that some of the screenshots might show older versions of the CBA ItemBuilder.

1.1 Installation & Requirements



While the items created with the CBA ItemBuilder can be used in different web-based environments (see chapter 7), a desktop application is necessary for designing, authoring, and editing the CBA ItemBuilder project files. An *install-wizard* as shown in Figure 1.1 is provided to install the CBA ItemBuilder to a default location that does not require local administrator privileges.⁴

CBA ItemBuilder Installation Wizard

After starting "CBAItemBuilder-{version}-windows-installer.exe", the application is installed in five steps. No admin rights are required because the software is installed in the appdata user directory.

File: InstallDemoWinVersion9_0.zip

Step 4: Wizard copies all required files to the target directory

FIGURE 1.1: Item illustrating *Installation Wizard* ([html](#)|[ib](#)).



Before using the CBA ItemBuilder, the program must first be downloaded and installed locally. No administrator rights are required for the installation, but currently, CBA ItemBuilder is provided only for Windows computers.

⁴For a user with the user name `USER_NAME` the CBA ItemBuilder of a particular `CBA_IB_VERSION` is installed here: `C:\Users\USER_NAME\AppData\Local\CBA_IB_VERSION\IB\cba-itembuilder.exe`. Note that for previous versions of the CBA ItemBuilder it was suggested to avoid *white spaces* in path names and to keep path names as short as possible.

System Requirements: Current Windows operating systems (Windows 8, 10 and 11) are supported. A computer with an SSD is recommended for fast and smooth operation. Minimal required disk space is 2 GB. Dual core CPU and at least 4 GB Ram are recommended.

Items are being created with the CBA ItemBuilder, a desktop application that requires a Windows operating systems. However, the created assessment components (so-called *Tasks* stored in CBA ItemBuilder *Project Files*) can be used in typical *Web Browsers* on any operation system and device type (including mobile devices). A build-in internal *Rendering* (see section 3.1.2) shows how pages will look like in a *Web Browser* and content created with the CBA ItemBuilder can be displayed in any *Web Browser* installed on the local computer using the *Preview*-feature (see section 1.4). The use of either Chrome, Firefox or the Chrome-based Edge browser is suggested.



To many technical terms? Note that this book contains a *Glossary of Terms* with short descriptions (see appendix A).

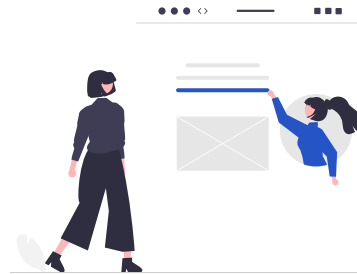
Technical Configuration: The CBA ItemBuilder is configured using an so-called *ini*-file (*cba-itembuilder.ini*). This allows activating and deactivating specific features and running the CBA ItemBuilder in different environments. By default, the CBA ItemBuilder is expected to run without modifying the *ini*-file. Details can be found in the appendix (see appendix B.4).

De-Installation: The de-installation of a previous version of the CBA ItemBuilder is (technically) not necessary. Copies of different versions of the CBA ItemBuilder can be installed on a computer in parallel. However, to de-install a previous version, just delete the CBA ItemBuilder folder or use the shortcut `Uninstall CBA ItemBuilder` created during the installation.

Manual Installation: For developers the CBA ItemBuilder is also delivered as zip archive without installer. To manually install the CBA ItemBuilder, extract the content of the zip archive to a local drive. The plain CBA ItemBuilder will run after unzipping the content, after starting the executable *cba-itembuilder.exe*. The CBA ItemBuilder will require a free port (by default 7070). If this port is not available, adjust the configuration in the file *cba-itembuilder.ini* (change: `-Djetty.port={free port}`), see appendix B.4 for detail).

Multiple Versions: Multiple instances of CBA ItemBuilder can be installed in parallel on one computer (see section 6.8.8 for use cases).

1.2 Contact & Support

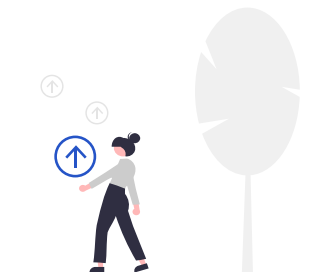


The CBA ItemBuilder is provided free of charge for non-commercial projects in the field of education (i.e., educational research, psychological diagnostics and related areas) by the [Centre for Technology-Based Assessment \(TBA\)](#), a scientific research and infrastructure center at [DIPF | Leibniz Institute for Research and Information in Education](#).



To obtain the latest version of the CBA ItemBuilder, see information at the homepage of the [Centre for Technology-Based Assessment](#).

1.3 Quick Start: Get the Right Version



The CBA ItemBuilder is continuously developed, so there are different program versions available. In order to reproduce the exact display of existing CBA ItemBuilder project, it might be necessary to use the exact same CBA ItemBuilder version and, if possible, the identical browser. The development of the software follows the

paradigm that a) older CBA ItemBuilder project files can be opened with newer versions of the CBA ItemBuilder, but b) older versions of the CBA ItemBuilder cannot open project files saved with a newer version of the software. If project files created with an older version of the CBA ItemBuilder are opened in a newer version (a), the items are internally migrated to the newer version (see section 3.2.1 for details about this *Migration*).

Which Version to Choose? If you are starting from scratch and using CBA ItemBuilder for the first time, install the latest version. If you have existing items that you want to view in the preview (section 1.4), whose scoring you want to view (section 1.5) or whose log events you want to explore (section 1.6), you could also consider using the version with which these items were created and tested.⁵ If you are planning to update or revise existing items (see section 1.7) for an upcoming assessment, you might want to use the latest version of the CBA ItemBuilder.

How to Start: After installing the CBA ItemBuilder, it can be started by double-clicking the icon or the executable file `cba-itembuilder.exe` in the installation directory (see section 1.1). The CBA ItemBuilder will allow editing of one CBA ItemBuilder project file at a time, and typically only one instance of the CBA ItemBuilder will be opened at the same time.⁶

Main Window: The *Main Window* of the CBA ItemBuilder consists of at different areas (see Figure 1.2): On the left side the *Project View* is located, with the *Component Edit* below. The gray area right to the *Project View* is reserved for various editors, like the *Page Editor*. If requested, additional views can be shown on the right part of the main window.

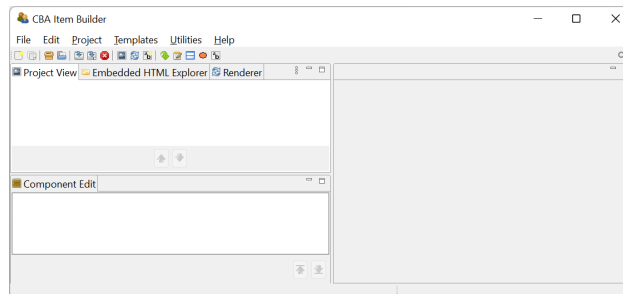


FIGURE 1.2: CBA ItemBuilder *Main Window*.

The *Help* menu of the main menu contains the entry *About*, that shows the version information (see Figure 1.3).

Migration: It is important that CBA ItemBuilder projects are never opened with pre-

⁵How to find out which version an item was created with is described in the subsection 8.3.3.

⁶Note: An error message is displayed, when a second instance of the CBA Item Builder is started. In this case the second instance of the CBA Item Builder terminates. If configured properly, new CBA ItemBuilder versions (starting with version 9.4) can be used in parallel, see section 6.8.8 for details.

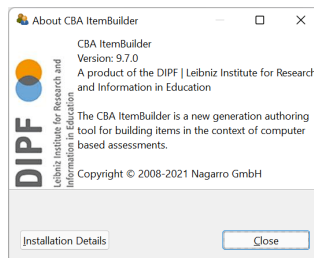
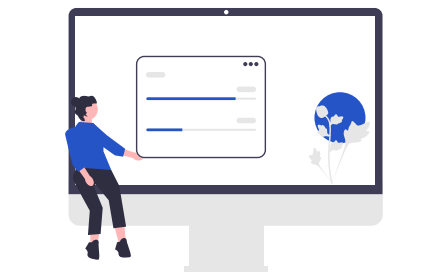


FIGURE 1.3: CBA ItemBuilder *About*-Dialog.

vious version, i.e. versions that are older than the last version that was used to save the CBA ItemBuilder project file. However, opening CBA ItemBuilder projects with newer version is possible. If CBA ItemBuilder projects are opened and saved with a newer version of CBA ItemBuilder, a so-called *Migration* is done automatically.⁷ Once a project file is migrated (i.e, opened and saved) to a particular newer version, no version of the CBA ItemBuilder older than that particular version can open the migrated project file.

1.4 Quick Start: Preview Items







Once the CBA ItemBuilder has been started, *Project Files* can be opened and presented using the so-called *Preview* feature. To preview an item means opening the project file within the CBA ItemBuilder desktop application and requesting the preview feature, which will bring up a web browser that shows a particular task.

⁷Migrations might require changes and corrections if the functionality of CBA ItemBuilder changed (see section 3.2.1).

Example Item

Do you know these types of fruit?
Please assign the appropriate names to the pictures by drag-and-drop!

Sharon fruit
Medlar
Quince
Greengage

Not sure about the answers?

Check Answers

Want to try again?

Reset

Submit

FIGURE 1.4: Example item to test *Preview* ([html](#)|[ib](#)).



Please note the two links in the figure caption of Figure 1.1. The link **html** gives access to the HTML representation of the CBA ItemBuilder item shown in the particular Figure 1.1 and the link **ib** refers to the CBA ItemBuilder project file that can be downloaded and opened with the CBA ItemBuilder.

In the following, we assume that you have either already received CBA ItemBuilder project files with item material. You could have obtained the CBA ItemBuilder project file from a collaborator, or a research data center (e.g., [DIPF FDZ](#)) or that you are starting to explore example projects from this book that can be downloaded from links provided in the figure's captions ([html](#)|[ib](#)).

1.4.1 Basic Terminology

The CBA ItemBuilder is used to create and manage item projects. In order to implement a typical computer-based assessment, multiple *project files* are created and edited with the CBA ItemBuilder. If an assessment is created by several people, the individual project files can be shared and edited with one instance of the CBA Item-

Builder of the appropriate version. To see how the content of a *project file* actually looks like in the assessment, the *Preview* function can be used.

Project File: *Projects files* are zip archives that can be opened, edited, or modified and previewed with the CBA ItemBuilder desktop application to see the content of the assessment material that is included in that file. All content that belongs to an item project is stored in the zip archives, and the zip archives should be modified only with the CBA ItemBuilder as editor.

Project files contain all the information that belongs to a particular part of an assessment (e.g., an item, a unit, an instruction, etc.). More specifically, *project files* include:

- The visual design of the item in terms of one or multiple pages, input elements, graphics etc. is included in the *project file* (see section 3.3).
- At least one starting point (referred to as the entry-point defined as a so-called *Task*, see section 3.6 for details) must be defined. If multiple parts of the assessment are defined in one *project file*, multiple *Tasks* can be defined.
- The internal logic controlling the behavior of the item (using so-called *finite-state machine*, see section 4.4) are included directly in the *project file*.
- The definition of potential scoring rules (see chapter 5) are embedded in the *project files*.
- Additional resources (such as images, sound, and video files, see section 3.10.1) required to render the assessment can be included in the *project files*.

Assessments implemented with the CBA ItemBuilder use one or multiple *project files*. The option of previewing CBA ItemBuilder *project files*, presented in this section, allows to view and inspect parts of assessments outside and independently from the delivery platform.

After a *project file* has been saved locally on the computer on which the CBA ItemBuilder is installed, it can be opened to start the preview. A typical file menu (see Figure 1.5) gives access to the basic functions for creating new projects (New project) opening existing CBA ItemBuilder project files (Open project), for saving a project file (Save), for renaming project files (Save As ...), and for closing the current project file (Close project).⁸

Project Files, i.e., zip archives containing parts of assessments, can be copied, moved, and shared like ordinary files. *Project files* can also be managed using version control tools (e.g., GIT or SVN, see section 8.3.2).

One specific feature needs to be mentioned briefly (see section 3.2.1 for more information):

⁸Note that a *project file* has a file name and an internal *project name* (see section 3.2.1.)

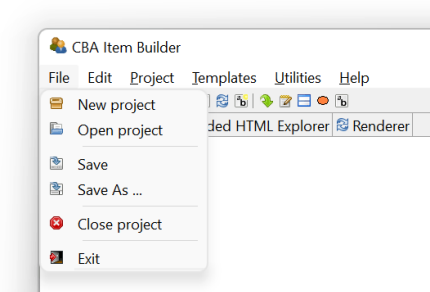


FIGURE 1.5: CBA ItemBuilder File-Menu.



CBA ItemBuilder *Project Files* are zip archives with a file name. In addition, each project has a *Project Name*. *Project* and *File Name* can be different, but in some software tools used for test assembly and deployment (see chapter 7) might require that the *Project Name* is identical to the *File Name* (see section 3.2.1).


The zip archives created by the CBA ItemBuilder as *Project Files* contain everything needed to develop and use a particular assessment component, including the generated configuration files required for deployment (see chapter 7 for details). This configuration or code is automatically generated when the item is saved using the CBA ItemBuilder, and it is specific to a particular version of the CBA ItemBuilder.

1.4.2 Preview Item in Browser

The best way to get to know the CBA ItemBuilder is with concrete examples. In this manual all embedded items are linked as CBA ItemBuilder project files. As soon as you have downloaded a concrete CBA ItemBuilder project file and saved it as a zip archive locally on your computer, it can be opened with the CBA ItemBuilder. This [link](#) provides access to the CBA ItemBuilder project file shown in Figure 1.4.

Preview: The content of a CBA ItemBuilder project file can be displayed in a web browser in the way, the content will be presented in the actual assessment. This feature of the CBA ItemBuilder is called *Preview*.

There are several ways to request the preview for an opened CBA ItemBuilder *project file*. The easiest way is to use the main menu (Project > Preview project see first entry in Figure 1.6).

After calling the preview via the menu (or using the icon  of the *Toolbar*, see section 3.1.1), the *Preview* dialog appears (see Figure 1.7). This dialog allows selecting which part of the project file should be previewed. Typically a task is previewed, and often a *project file* contains only one task. In this case, the default selection (the first *Task*

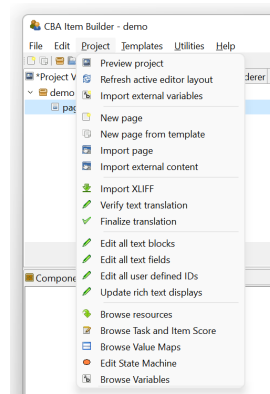


FIGURE 1.6: CBA ItemBuilder Project-Menu.

will be previewed, see `Task01` in the example shown in Figure 1.7) must be confirmed by pressing `OK`.

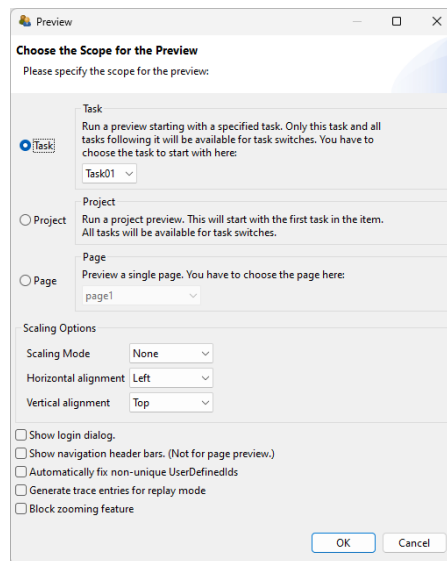


FIGURE 1.7: CBA ItemBuilder Preview-Dialog.

CBA ItemBuilder project files can contain multiple entry points, called *Tasks* (see section 3.6 for details). Each *Task* can select a different parts of an CBA ItemBuilder project file by specifying the starting *Page*.⁹ If multiple *Tasks* are defined, the *Preview* follows navigation request for next and previous *Tasks* and shows the *Tasks* accord-

⁹If the task uses a so-called *X-page layout*, the first two pages that appear on screen are defined in the task definition (see section 3.6.2).

ing to the order, in which they are defined within the CBA ItemBuilder *project file*. The *Preview* will apply proportional scaling if requested using the *Scaling Options* (see section 3.2.2 for details).



CBA ItemBuilder project files contain one or multiple *Tasks*. *Tasks* are the entry points that are used for assessments. Hence, the *preview* of a particular *Task* is typically the right choice.

The preview dialog also can be used to preview selected pages directly. This option is only of importance during the development of assessment material (see chapter 3). Moreover, the dialog also offers the possibility to preview the project if no task is defined, using the first defined page as the (temporary) entry point.

Which (Web-)Browser should be use for Preview? The preview of the CBA ItemBuilder uses the so-called default browser of a computer to show the assessment materials such as items, if configured as *Systems Default Browser* in the dialog *Preferences* in the section *CBA Preview* (see Figure 1.8):

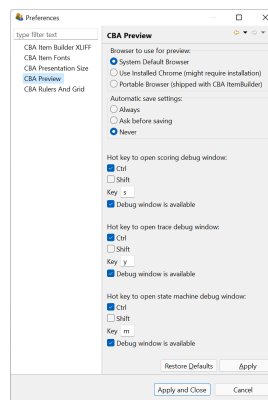


FIGURE 1.8: CBA ItemBuilder *Preferences*-Dialog.

The dialog shown in Figure 1.8 also shows additional settings, such as the auto-save configuration (see section 3.2.1) and the hot keys used for *Scoring Debug Window* (see section 1.5.2), the *Trace Debug Window* (see section 1.6.2) and the *State Machine Debug Window* (see chapter 4).

If different browsers are expected, given a particular delivery strategy, it should be ensured that the items, including the externally integrated components, are compatible with the used browsers. Section 8.4.1 describes how the preview can be used in different browsers to check the cross-browser compatibility locally.

Automatic Start (Autostart): Web browsers prevent audio and video files from playing before the first user interaction. This security setting is not a restriction in practice since a login or welcome page is usually displayed before the first item. After

clicking a button on this page, for example, audio and video components can then be started automatically on all subsequent pages. However, this does not apply to the preview, which uses a new browser or browser tab. The *Preview* can be requested with a log-in dialog (see the checkbox `Show login dialog` in Figure 1.7) to allow the preview audio and video components with auto-play. Otherwise, a warning message is shown as displayed in Figure 1.9.

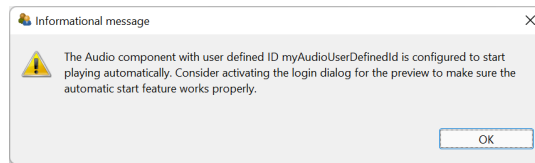
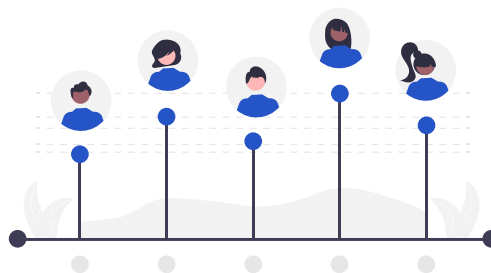


FIGURE 1.9: Warning about autostart of audio and video components.

Summary: With the CBA ItemBuilder, assessment components can be created as *project files*. Complete assessments with multiple tests or booklet designs with rotations (see section 2.7.2) usually consist of a sequence of many assessment components (i.e., several *project files* with the corresponding *tasks* as the entry points). Using the preview function described in this section, the individual assessment components stored in a particular *Project File* can be displayed and checked directly from the CBA ItemBuilder. Thus, the components of assessments can be viewed even outside the sometimes complex rotation designs.

1.5 Quick Start: Explore Scoring



This section is intended for readers who already have CBA ItemBuilder project files and want to preview and check scoring with CBA ItemBuilder.

Viewing assessment content created with the CBA ItemBuilder in the preview (see

section 1.4) also allows to view the built-in automatic scoring. A possibility to access the automatic scoring of tasks is useful, for example, to check which scoring results from a specific input or a certain test-taking behavior.



A possibility to examine the automatic scoring of computer-based items is essential for creating assessments and interpreting data from computer-based tests.

In the following, it is assumed that assessment content created with the CBA Item-Builder is available that contains automatic scoring rules. This section aims to provide the required information for exploring the implemented scoring in *project files*, either for testing or understanding available instruments.

1.5.1 Basic Terminology

A majority of assessments conducted for the different use cases (see chapter 2) can be described as a data collection based on responses or test-taking behavior within a computerized environment. Accordingly, it can be useful to think of the result of assessments as a data set, typically with persons in rows and variables in columns.

Result Variables vs. Score Variables: For each person, the data set should contain the scored final answer to each item (e.g., a score variables either 0 for wrong, 1 for correct). The data set could also include a variable for the selected answer option (i.e., a raw result variable). Result variables and score variables can be defined for simple single choice tasks (see section 2.2) simply by naming the possible and correct answer option(s). For more complex computer-based items, the definition of a result variable may already rely on several input elements or may require the use of information from the test editing process.

Typically, score variables assign credits (no credit and full credit for dichotomously scored items and one or multiple additional partial credits for polytomously scored items) to possible result variables' values. In other words, once all possible values of result variables are defined, the definition of the scoring variables is also almost done. Unfortunately, for polytomously scored items, defining partial credits is not necessarily possible without taking into account empirical data to make sure the intended order of partial credits holds empirically.

Manual Scoring vs. Automatic Scoring: An important distinction concerns the difference between automatically scored answers and manually scored answers. The integrated handling of scoring definitions implemented in CBA ItemBuilder concerns automatic scoring of mainly closed answers. This refers to responses by selecting an input element, selecting a page, entering a particular state, etc. For free text answers, the automatic scoring is limited to the identification of given text by means of regular expressions.

Classes as Variables: Each variable, i.e., each column in the result data set, is defined in the CBA ItemBuilder as a so-called *class*. Each part of an assessment, which is defined as a CBA ItemBuilder *project file*, can generate any number of result variables and the corresponding *classes* are defined within the *tasks*. Accordingly, the classes form the columns of the result data set. Thereby variables can either have categorical values (called hit-conditions) or take over character strings into the result variables (called result-texts).

Hit-Conditions as Values: Hit-conditions are logical expressions, which combine one or more possible inputs of test-takers. If, for instance, the correct answer option is selected in a single choice item, this can be indicated by a first hit-condition. If a wrong answer option is selected, this can be indicated by a second hit-condition. If the two hit conditions are defined so that either one or the other condition is active, then the two hit-conditions form the values of one variable. The assignment of hits to classes specifies this relationship of hit-conditions as the variable's values. The CBA ItemBuilder provides a variety of options to define hit-conditions (see section 5.3).

Missing Values: Besides the hit conditions defined for specific constellations of responses, missing answers can find special consideration. For distinguishing reasons why replies or responses are not available for a particular test-taker in the result data set, hits can also be defined for so-called *missing values* (see section 2.5.2 for more details).

Text Responses: Beyond categorical variables, whose values can be defined by hit conditions, there are also answer variables that contain the text answers. For such variables, the sorting definition consists of specifying from which input elements the result text should be taken (see section 5.3.10).

Codebook vs. Scoring Definition: For the result data set of a computer-based assessment, all used variables should have so-called *variable labels*. For importing the data sets into statistical programs like SPSS, Stata, or R, the definition of a data type for each variable can also be useful. This information per variable is typically stored in so-called codebooks. Codebooks for CBA ItemBuilder *tasks* should define for each *class* whether it should be used as a categorical variable (i.e., hit conditions are used as values) or whether it uses the result text (i.e., the variable represents a text response). For categorical variables, the value labels can then also be defined in the codebook. The possible values of categorical variables are defined as hit conditions. For each hit condition, a value label and, if necessary, a numerical value must be defined in the codebook, with which the values of the categorical result variables should be represented in the final result data set. For CBA ItemBuilder tasks in assessments, the codebook represents a translation of the scoring definitions into result data sets. The next section describes how the scoring definition for CBA ItemBuilder *tasks* can be checked and displayed directly in the embedded preview, using the *Scoring Debug Window*.

1.5.2 Scoring Debug Window

Direct access to verify the scoring definition based on classes and hit-conditions for a running item is provided as the so-called *Scoring Debug Window*. If configured, this window can be called directly by a key combination (default is `Strg / Ctrl + S`, see appendix B.4 for details) as soon as the preview for a *Task* or a *Project* is requested as described in section 1.4.2. The *Scoring Debug Window* is available when a *Task* or *Project* is previewed using the CBA ItemBuilder.¹⁰

Scoring Debug Window

Score result: true

Execution Time: 1578

Total hits/Total Weight: 3/3

Credit Class: Variable2Score

Reaction Time: 1387

Total misses/Total Weight: 0/0

Credit Weight: 1

Nb. of Interactions: 1

Result text:

Hits:

| No. | Name | Weight | Class | Result text |
|-----|-----------------|--------|----------------|-------------|
| 1 | Question1_Wrong | 1 | Variable1Score | |
| 2 | Question2_Wrong | 1 | Variable2Score | |
| 3 | Question3_Wrong | 1 | Variable3Score | |

Misses:

| No. | Name | Weight | Class | Result text |
|-----|------|--------|-------|-------------|
| - | | | | |

FIGURE 1.10: CBA ItemBuilder *Scoring Debug Window*.

Figure 1.10 shows a screenshot of the *Scoring Debug Window* for the example item shown in Figure 1.11. The *Scoring Debug Window* is organized into three sections:

- The first section displays summaries for the entire task. This information can be useful for simple items (see section 5.4). Regarding the general description of automatic scoring using the CBA ItemBuilder, provided in this section, the task summaries are not relevant.
- The second section contains the list of all hits assigned to classes and, if used, the *result text*. This list contains the information described in the previous section. The interpretation of these outputs is described in detail by means of examples in the following section.
- Finally, the third section, not described here in detail (see section 5.3.2), contains potentially existing miss-conditions.

The three lines of the hits table in figure 1.10 can be read as follows: Line 1 shows that the *hit* `Question1_Wrong` is active, which is assigned to the *class* `Variable1Score`. Line 2 says that the *hit* `Question2_Wrong` is active, which is assigned to the *class* `Variable2Score`. Line 3 finally shows that the *hit* `Question3_Wrong` is active, which is assigned to the *class* `Variable3Score`. As the hit names suggest, neither `Question1`, `Question2` nor `Question3` was answered correctly. For all three variables in the data

¹⁰No *Scoring Debug Window* is present for the preview of single *Pages*.

set, the expected information would be that the respective question were answered wrong.

Figure 1.10 shows the *Scoring Debug Window* for the CBA ItemBuilder project file shown in Figure 1.11. For the shown *task* three questions are defined on one page. Question 1 is answered correctly if option C is selected. Question 2 is answered correctly if option A and B are selected. The correct answer for question 3 is D.

With the help of the *Scoring Debug Window*, it can now be checked that the correct answer also causes the hit, which shows a correct answer to be activated.



The *Scoring Debug Window* can be used to check which *hit* is active for each *class* at any given time. The *Scoring Debug Window* is available in the *Preview* (see section 1.4.2) and in the examples embedded in the [online version](#) of this book.

The active hit or miss as shown in the column *NameOf* of the *Scoring Debug Window* or the provided *Result text* is used as value for a variable (see column *Class*).

Examples: The CBA ItemBuilder shown in Figure 1.11 can be used to play with the *Scoring Debug Window*. In this project, the scoring definition differentiates wrong vs. correct responses. Accordingly, with respect to the differentiation in score variables and result variables, Figure 1.11 only contains score variables. To understand this property, select different wrong answers to the three questions for the following item and check the scoring with the *Scoring Debug Window*. As you can see, there is a distinction between correct and wrong answers. Which wrong answers are selected for the three questions is not taken into account in the scoring.

Figure 1.12 contains the identical three questions, but now with a scoring that contains result variables as well as score variables. A deployment platform that can handle CBA ItemBuilder tasks is expected to provide the variable as defined within the tasks. If the result data should contain variables that contain which answers were selected in addition to the variables for correct versus wrong answers, the result variables can be defined as part of the CBA ItemBuilder scoring.

For the *Single Choice* task (question 1), a hit shows the different possible choices (compare the hits *Question1_OptionA*, *Question1_OptionB*, *Question1_OptionC*, and *Question1_OptionD* which are all assigned to the class *Question1*).

Note, however, that for the *Multiple Choice* task (question 2), one class (i.e., one variable) is defined for each option. Accordingly, two hits for the class *Variable2A* show if option A for question 2 is selected (*Question2A_Selected*) or not selected (*Question2A_NotSelected*). One result variable is required for each option of a *Multiple Choice* task (see the classes *Variable2A*, *Variable2B*, *Variable2C*, and *Variable2D*). Moreover, as already shown in Figure 1.11, the score variable for question 2 indicates if the two required options are selected (see hits *Question2_Wrong* and *Question2_Correct* for class *Variable2Score*).

Quick Start: Explore Scoring - Example 1

Example with 3 questions on one screen:

Question 1: Single-Choice Task - Select C

☐ Option A
☐ Option B
☐ Option C
☐ Option D

Variable1Score

Question 2: Multiple-Choice Task - Select A and B

☐ Option A
☐ Option B
☐ Option C
☐ Option D

Variable2Score

Question 3: Constructed-Response Task - Enter D

Variable3Score

File: QuickStartScoringExample1.zip

Click here and press Ctrl + S (Strg + S) to open the Scoring Debug Window.

FIGURE 1.11: Scoring-example (score-variables, [html](#)|[ib](#)).

Finally, for the *Constructed Response* task (question 3), the class `Question3` only has one assigned hit (`Question3`). This hit, however, uses the so-called `result_text()`-operator (see section 5.3.10 for details). Using the `result_text()`-operator causes the entered character to be displayed in the *Result text* column of the *Scoring Debug Window*. This illustrates that the text entered in the input field (here only one letter) can then be used for the result data set.

Summary: The definition of automatic scoring of items is described in detail in Chapter 5. Accordingly, this section did not describe how to define scoring. But it was shown that with the help of the *Scoring Debug Window* in the preview of the CBA ItemBuilder, the defined scoring can be checked and displayed for different inputs. In order to check for existing ItemBuilder project files whether the integrated scoring works correctly, it is not necessary to check the various rotations of a test delivery with a booklet design. The checking of the scoring can be done separately for the individual assessment components.

Quick Start: Explore Scoring - Example 2

Click here and press Ctrl + S (Strg + S) to open the Scoring Debug Window.

Question 1: Single-Choice Task - Select C

☐ Option A

☐ Option B

☐ Option C

☐ Option D

Variable1

Variable1Score

Question 2: Multiple-Choice Task - Select A and B

☐ Option A

☐ Option B

☐ Option C

☐ Option D

Variable2A

Variable2B

Variable2C

Variable2D

Variable2Score

Question 3: Constructed-Response Task - Enter D

Variable3

Variable3Score

File: QuickStartScoringExample2.zip

FIGURE 1.12: *Scoring*-example extended (with result-variables, [html|ib](#)).

1.6 Quick Start: Explore Log Events





This section is intended for readers who want to interpret log data of an assessment carried out with the CBA ItemBuilder. To be able to view live which log events are provided for which user interaction, the preview can also be used.

An important challenge in documenting log data is to enable secondary researchers to understand which user interactions within the assessment led to which (basic) log events.

To interpret log data of computer-based tests, some form of documentation is necessary. For assessments created with the CBA ItemBuilder, the CBA ItemBuilder project files can be used for documentation purposes (see section 8.7.2 for details), after sensitive item content is replaced with dummy text (image or media files).

1.6.1 Basic Terminology

If CBA ItemBuilder project files are available, researchers can inspect not only the scoring (see section 1.5) in the preview (see section 1.4), but also get an impression of the additional behavioral data created during the task execution.



Tasks shown in the CBA ItemBuilder *Preview* can also be used to inspect the recent *events* collected as *log data*.

Log Events: The following basic understanding of log events is necessary to assign the processing behavior to the log data. Log data consists of single log events. Log events can result from user interaction (e.g., clicks) or internal system changes (e.g., timers). In the context of assessments, log events always occur with an assignment to a person. There is always a *person identifier* associated with a log event (that can have different names such as actor, pupil, student, test-taker, respondent, etc.). Log events also always occur at a particular time. Accordingly, at least one *timestamp* is always associated with log events, indicating when the event occurred. Other timestamps that show, for instance, when the event was transmitted or received are possible but not mandatory for log events. The meaning of log events is encoded in the so-called *event type* (sometimes also denominated as event name). How events of a certain event type can be interpreted should be comparable for assessments using the same assessment platform. However, the meaning of identically named events may differ between different assessment platforms. An event of a particular event type that occurred at a specific time can be the central information in particular log data. However, events of a particular event type can also provide additional information. For example, a mouse-click event could inform about whether the right or left mouse button was clicked. This additional information is called *event-specific data*. If the event-specific data can be described by enumerating key-value pairs, we also

speak of event-specific attributes. Event-specific data can be mandatory (i.e., every event of this event type provides it) or optional (i.e., events of that event type can provide this information but do not have to). Finally, log events always have an identifier for a part of the assessment (*instrument identifier*). In the context of log data collected with the CBA ItemBuilder the name of the CBA ItemBuilder project (i.e. the name of the *project file*) is often sufficient.

Original Items: If the original items are available, i.e., the ItemBuilder *project files* are available as they were used in a delivery platform (see chapter 7), the preview of the CBA ItemBuilder can be used to explore the triggered log events using the *Tracing Debug Window* described in the next section. Suppose the original *project files* can be used. In that case, it is automatically ensured that the so-called `UserDefinedIds` of the individual components used in the CBA ItemBuilder, which can be affected by user interactions, can be identified in the event-specific data of the log events. The `UserDefinedIds` are necessary for the interpretation of log events as soon as several components of one type within an ItemBuilder project have to be distinguished in order to assign the log events exactly (see section 3.7.4 for details).

Mock-Items: Unfortunately, for some instruments it is necessary to keep the task content secret. The stimulus and question texts, the concrete tasks, pictures, audio and video files, etc. must then be kept confidential. However, the confidential item contents are not necessary to explore the computer-based instrument concerning log events that are triggered by different user interactions. Mock-items are items in which the protected item contents have been replaced, keeping the structure and especially the `UserDefinedIds` unchanged (see section 8.7.2 for more details).



If for interpretation of log data the original items or carefully created mock-items for an instrument are available in form of CBA ItemBuilder *project files*, the log events can be explored with the *Tracing Debug Window*.

Since the CBA ItemBuilder versions can differ concerning the collected log data, it is suggested to use the identical CBA ItemBuilder version if this feature is used to make sense of log data from existing data collections (see section 1.3).

1.6.2 Trace Debug Window

Like the *Scoring Debug Window*, the *Tracing Debug Window* is a functionality provided by the CBA ItemBuilder during the preview of tasks. To use this feature, an original item or mock item must first be opened with the CBA ItemBuilder in the appropriate version. Afterward, as described in section 1.4, the preview must be requested. As soon as a *task* is previewed, a key combination (default is `Strg / Ctrl + Y`, see appendix B.4 for details) requests the *Tracing Debug Window*.

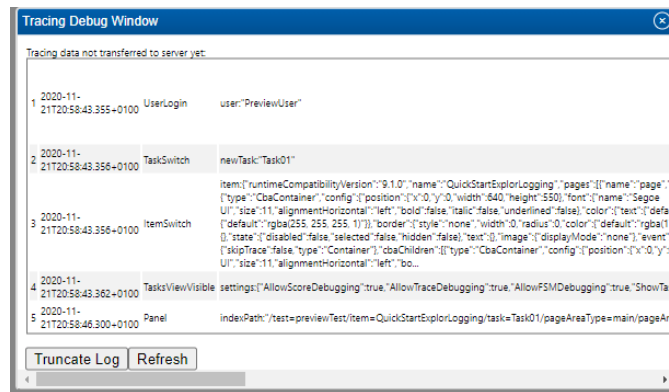


FIGURE 1.13: CBA ItemBuilder *Tracing Debug Window*.

Figure 1.13 shows a screenshot of the *Tracing Debug Window* for the example item shown in Figure 1.14.

Select the answer by clicking on one of the buttons. The *Tracing Debug Window* displays the past log events of the interaction with the currently displayed item:

- The first column contains a counter for the log events.
- The second column shows the timestamp.
- The third column is filled with the event type.

All following columns contain event-specific data, including the `UserDefinedId`. The *Tracing Debug Window* can be used to display the recent log events. After some time, the entries are deleted, i.e., only the most recent log events are displayed. Note that the input focus must be *within* the item before the keyboard shortcut is detected. The text *click here* in the example item (see, for instance, Figure 1.14) reminds you to set the input focus before pressing `Ctrl + v`. While the Trace Debugger is open, no further log events are triggered respectively registered.

Examples: The CBA ItemBuilder project file shown in Figure 1.14 gives an insight into how the *Tracing Debug Window* works. In this example simple buttons are used to implement a *Single-Choice Item*. The buttons are arranged in an identical way on different pages. With the help of the `UserDefinedId` of the buttons the selection behavior can be traced.

Summary: The *Tracing Debug Window* of the CBA ItemBuilder’s preview can be used to display which log events a particular user interaction has within the computer-based assessment components. The complete list of implemented log events of the current version of the CBA ItemBuilder can be found in appendix B.7. Additional log events can be added by the execution environment (see chapter 7). Log events are generated by default (i.e., no specific configuration is required to enable logging, see section 2.8).

Quick Start: Explore Log Events

Click here and press Ctrl + Y
(Strg + y) to open the Trace
Debug Window.

Example Task: Which of the following three options is correct?

Select the answer by clicking on one of the buttons.

Option A

Option B

Option C

Please note that the key combination `Ctrl + Y` or `Strg + Y` works only if the input focus (cursor) is inside the item. Therefore, you should left-click on *click here* before you enter the key combination.

File: QuickStartExplorLogging.zip

FIGURE 1.14: *Button*-example for single-choice response format ([html](#)|[ib](#)).

1.7 Quick Start: Revise Item Content



This quick start section is intended for use cases where there are already existing

CBA ItemBuilder projects that require only minor adjustments or changes. If you want to create new items, jump to the quick start section [3.3](#)).

A primary motivation for developing an authoring tool for computer-based assessments was that item content should be easy to create and adopt without programming knowledge. Content experts should use the CBA ItemBuilder to develop assessment components and modify, adapt, and improve existing content. The subsequent section deals with this use case. In the following, it is assumed that assessment components (i.e., tasks or items) already exist in the form of CBA ItemBuilder *Project Files*, which are to be adapted, revised, or changed.

There may be several reasons why existing assessments might be adapted and modified. The ability to make minor changes is an essential part of *Open Assessments*. Potential reasons why existing instruments (i.e., assessment components in the form of instructional pages, tasks, and questions) should be adapted, could include:

- An instrument computerized using the CBA ItemBuilder is not available in the required language and should be translated by a content expert into a new language.
- An instrument was translated by an expert, and the translation was verified. However, since different languages require a different amount of space, so-called layout adaptations might be required for the assessment components created with the CBA ItemBuilder.¹¹
- Computer-based instruments include additional components besides the actual test created by tasks and questions that embed and frame the assessment into a particular context. However, wording that was appropriately and accurately chosen for a particular occasion may be disruptive, obstructive, or misleading for further application of a test or instrument. This may relate only to how the target persons are addressed, to the wording in the consent form for participation, or other parts of the text used within assessment components.
- Within items, tasks or questions, individual phrases may need explanation for a particular target audience or may need to be replaced with other phrases that are more familiar or common to target audiences.
- Finally, perhaps some questions or tasks should be excluded or changed. As far as the changes require adjustments within *Project Files*, these can be implemented with the CBA ItemBuilder. Changes that affect the composition of assessments from multiple CBA ItemBuilder *Project Files* must then be configured in the test delivery software (see chapter [7](#)).

¹¹Note that in previous projects that used the CBA ItemBuilder in multilingual assessments, an automated translation process was used using the XLIFF format (see section [6.9](#)). The current version [version 10.0.0](#) might not fully support XLIFF. please contact the support, see section [1.2](#)).

Test and questionnaire development of psychometric instruments requires a scientific approach and in-depth understanding of the construct itself and the process of item development, test theory, and validation that goes far beyond the technical implementation in an assessment software. When items of computer-based tests are developed and scaled in terms of calibration, changes to item content and significant changes to response formats and item presentation may need to investigate and, in case, re-scale the modified items (see section 8.7.4 for a detailed discussion).



Modifications to psychometrically developed tests and questionnaires can invalidate the instruments or at least require empirical studies to investigate psychometric properties of modified instruments.

Hence, the following section can only describe from a purely technical perspective how CBA ItemBuilder *Project Files* at hand can be used as a starting point for *sensitive* changes to existing assessment components. CBA ItemBuilder *Project Files* could be available, for instance, because they were provided by a research data center or shared by other scientists.

1.7.1 Basic Terminology

In order to modify and adapt assessment components created with CBA ItemBuilder, a rough understanding of the structure of CBA ItemBuilder *Project Files* is required first. *Project Files* can be thought of as a *workspace* that contains all content that can be edited with the CBA ItemBuilder. This content is divided into the following units:

- **Pages:** Content is placed on pages and one or more pages are displayed simultaneously. In the *Project View* (see section 3.1 for a full description of the user interface) all pages are listed. If you right-click a page and select the entry *Preview Page*, you can view each page in isolation in the browser as it will then look during the runtime of the assessment. The pages have names assigned by the item author. Do not rename pages, because they are used as references for links and in the possibly configured dynamic parts. If all the contents of a *Project File* are to be translated, all pages must be edited. If only one part is to be adjusted or changed, the appropriate page must first be found. Pages can be previewed, or opened by double-clicking in the *Page Editor*.
- **Components:** The *Page Editor* shows a design view for editing pages. Pages are designed by placing components in the *Drawing Area* of the *Page Editor*. The design view is not 100% the same as how a page will look at runtime, but it reflects which component is placed at which position. Components are hierarchically nested on pages and can only be inserted in certain orders.



In the *Page Editor* elements can be selected but also moved by drag and drop. Be careful and undo unwanted changes by using the *Undo* function (see section 3.7).

Two views of the CBA ItemBuilder are central for editing components:

- *Component View*: Without going into detail about the design of pages using components (see section 3.7), the so-called *Component Edit* provides direct access to all components of a page. The *Component Edit* is the tab next to the *Project View*. It is possible to navigate within the tree view of the *Component Edit* to see nested components. To modify the properties of existing components, the context menu of components that opens after right-click on components in the *Component Edit* can be used.
- *Property View*: In order to be able to change the visible parts of components, it is first necessary to identify what type of component it is. The type of a component is displayed directly in the *Component Edit* in square brackets. To view and change further details about the components, the entry *Show Properties View* can be selected from the context menu of the component in the *Component Edit*. This opens the *Properties* view in the right area of the CBA ItemBuilder. The *Properties* view allows to edit properties of the component that is currently selected in the *Drawing Area* of the *Page Editor* or in the *Component Edit*. Most properties, with the exception of editing formatted text, can be changed directly in the *Properties* view.

Adaptations may also be necessary for images, audio, or video files that are used to design pages of assessment components in *Project Files*:

- *Resource Browser*: Images, videos and audio files are not created with the CBA ItemBuilder. Instead, media files are embedded into the *Project Files* and components link to the required resources. All resources included in a *Project File* can be viewed (and exchanged) via the *Resource Browser*, which can be opened via the menu *Project* and the entry *Resource Browser*.

With the help of the above-mentioned features of the CBA ItemBuilder, items can be customized and, for example, translated, as described in the next section.

1.7.2 Change Content in Existing Project-Files

To make changes to an assessment component created with the CBA ItemBuilder, the *Project File* must be opened. The CBA ItemBuilder requires additional configuration to allow opening two parallel programs on one computer (see section 6.8.8). As

described in detail in the chapter 3, content can be edited using the *Page Editor* and assessment components created using the CBA ItemBuilder can be modified. For some selected purposes, the specific dialogs are available.



The CBA ItemBuilder provides the *Edit all text fields* function (in the *Project* menu), which can be used to make text changes and translations easily.

Change Texts using the *Edit Text Occurrence-Dialog*: For changing text, the CBA ItemBuilder provides a function that displays all components with text. The text can be changed immediately by double-clicking, without the need to open single pages. To use this function a *Project File* must only be opened. Then the option *Edit all text field* can be accessed via the menu *Project*. In the dialogue opening thereby (see illustration 1.15) all components are indicated, which have a text property.

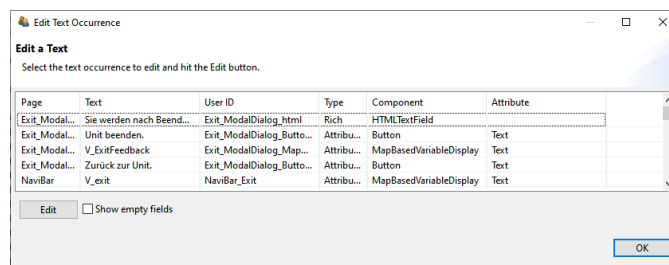


FIGURE 1.15: CBA ItemBuilder *Edit Text Occurrence* Dialog.

The page on which a component with a particular text is placed is shown in the first column of the dialog shown in Figure 1.15. A preview of the text is shown in the second column, followed by the *UserDefinedId* (i.e., the user-defined internal identifier of the component). The columns *Type*, *Component* and *Attribute* inform about the types of the component the text belongs to. Do not translate texts that you don't expect to be shown in the assessment to test-takers (such as texts used as variable names for components of type *MapBasedVariableDisplay*).



Texts that were included via images when designing assessment components with the CBA ItemBuilder must be replaced by replacing the images.

Depending on the design of the items by the item authors, it may be necessary to modify components manually, as described in the following. In general, the way how to change properties and the content of component depends on the type of component. The type of each component can be identified when the component is displayed in the *Component Edit*.

Change Text in Property View: For all components that only show text in one particular font size and font family (e.g., `TextField`, `Button`, `Link`, `RadioButton`, `CheckBox`, ...), the text can be changed by changing the value of this property in the *Properties* view. For components that allow text to be formatted differently (formatted text), the CBA ItemBuilder provides internal editors (see the next subsection). To change simple text using the *Properties* view, the page must first be double-clicked for editing in the *Project View*. Then you can switch to the *Component Edit* (the next tab right to the *Project View*). Expanding the tree in the *Component View* is possible using the small arrows left to the text naming the [component type]. With the help of the right mouse button and the entry *Show Properties View* the *Properties* view can be shown. The *Properties* view always shows the properties of the component selected in the *Component View / Page Editor*. The *Properties* view is a powerful tool within the CBA ItemBuilder, allowing for instance, to adjust the displayed text of many components changing the value in the section *Text* of the *Properties* view.



If more space is needed by changing the text, the size of each component can be adjusted using the 'Width' and 'Height' properties. You may also need to adjust the position via the *x* and *y* properties.

The font and font size used to display components with a text property can be adjusted in the *Appearance* section of the *Properties* view.

Change Formatted Text: Components of type `TextField`, `HTMLTextField` and `ImageTextField` do not use the *Text* Property. Instead, the text for these components can each be edited with a separate text editor, which also provides functions for formatting text sections. To open this editor these components can be double clicked in the *Page Editor*. `TextFields` and `ImageTextFields` also provide the entry *Edit Rich Text* in the context menu of the *Component Edit*. After an editor is opened for editing, the text can be changed. The changes are applied when the editor is closed via the *Save* and *Close* button.

Change Images, Videos and Audios: Components that show images or play videos or audio files refer to the media files embedded in the *Project Files*. The references are shown in the *Properties* view. To change or adjust media files, the reference can remain unchanged if the file name and file extension remains identical. It is sufficient to update the media files themselves (see section 3.10.1). Note that in case of images and videos the updated files require to have the identical size. Images are used with many different components for designing CBA ItemBuilder *Pages*, for instance, as background in containers of the type `Panel` (see section 3.10 for details).

If text is included in many images (instead of having images in the background and *transparent* components to show text on top within the CBA ItemBuilder, see section 3.7.5), we suggest the following: First export all images from a CBA ItemBuilder *Project File* that contain any text. This can be done manually by clicking on each image in the *Resource Preview* of the *Resource Browser* and exporting it via *Save Image* (or

similar). Alternatively, the CBA ItemBuilder *Project File* can also be opened using a program for ZIP archives to copy the image files from the sub-directory *resources* (see section 8.3.3 for details).

Translated or modified image files, audio files or videos can be easily replaced by *Add* in the *Resource Browser*. To do this, close all open pages beforehand (so that no *Page Editor* is visible anymore).

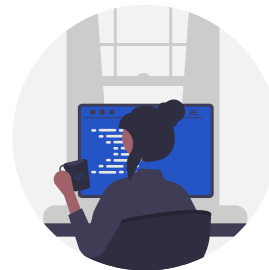
Change Text Mapped to Numbers: The CBA ItemBuilder also supports the display of dynamic content with various components. For this purpose, so-called *Value Maps* (see section 4.2.4) are used to assign texts, images or videos to numerical values (*Guards*). The images used there can be exchanged with the help of the *Resource Browser*, as described. Texts, however, must be edited in the *Value Map Details*. For this purpose, the entry *Browse Value Maps* can be opened via the menu *Project*. If a *Value Map* contains text, then this can be edited after selection of the *Value Map* in the *Value Map Details* with the help of the *Edit* button.



A step-by-step approach is recommended for the modification of assessment components. Check adjustments with the help of *Preview* before making further changes.

The CBA ItemBuilder automatically checks the consistency of the configuration when a large number of changes are made and displays an error message if necessary. A collection of error messages and possible solutions can be found in section 6.8.4.

1.8 CBA ItemBuilder for Software-Developers





This section is not intended for *item authors* who want to use the CBA ItemBuilder to computerize assessment content but for software developers that were asked to support a particular assessment project that considers using the CBA ItemBuilder.

The CBA ItemBuilder supports mainly two use cases that require software developers.



The CBA ItemBuilder is not only freely available for non-commercial use (see section 1.2). It can also be integrated into existing contexts and allows the integration of existing HTML/JavaScript implementations of assessment content- In this sense, the CBA ItemBuilder is an *open* tool for developing computer-based assessments.

Embedded Content into CBA ItemBuilder Tasks: Content can be embedded into CBA ItemBuilder items using *iframes* (used within CBA ItemBuilder designed *Pages* with components called *ExternalPageFrames*, see section 3.14). A simple interface allows the embedded *HTML5/JavaScript* to interact with the CBA ItemBuilder task, to store data, and to re-store its state on re-visit is available.

Software developers can find all required information in section 4.6.



HTML5/JavaScript programmers can add missing features to CBA ItemBuilder items by implementing content that is embedded as *iframes*, and that can interact with the content created by item authors using a simple API based on *Post Messages*.

Open source repositories with content generated for *ExternalPageFrames* are welcome.¹²

Embedding CBA ItemBuilder Tasks into other Applications: Items created with the CBA ItemBuilder can also be embedded into other web-based delivery software. For this use case the so-called *TaskPlayer API* is provided.

Software developers can find all required information in section 7.7.



Developers can integrate the content generated with the CBA ItemBuilder by including the provided runtime (called *TaskPlayer API*) into web applications, that allow loading and displaying of CBA ItemBuilder *Tasks*. No special server technology is required.

¹²See, for instance, [msk-oc-externalpageframes](#).

Open Source Contribution to the CBA ItemBuilder: Currently, open-source contributions to the CBA ItemBuilder's source code (i.e., the desktop application) and the React runtime cannot be managed and considered for integration by DIPF/TBA.



2

Principles of Computer-Based Assessment



Computer-based assessment (CBA) is based on principles and ideas of psychometric measurement and diagnostics (e.g., [Parshall, 2002](#), [Bartram and Hambleton \(2006\)](#), [Lane et al. \(2015\)](#)), developed in various disciplines, such as psychology, social sciences, and educational research, over many years. Since data collection was done for a long time using paper-based tests and questionnaires, previous research still influences how CBA is designed. However, from the beginning of CBA research, it was anticipated that the change in the technology used for testing, although started with the transfer of existing paper and pencil tests, would also impact the instrument development and the theory of test responses (e.g., [Dann et al., 1991](#)).

Educational Large-Scale Assessments: Beyond the traditions developed for paper and pencil instruments, in particular the implementation of current educational assessments are also influenced by the testing industry (e.g., [Bridgeman, 2009](#)) and international large-scale assessment (ILSA) programs such as PIAAC (*Programme for the International Assessment of Adult Competencies*, e.g., [Kirsch and Lennon, 2017](#)), PISA (*Programme for International Student Assessment*, e.g., [Naumann and Sälzer, 2017](#)), TIMSS (*Trends in International Mathematics and Science Study*, e.g., [Fishbein et al., 2018](#)), ICILS (*International Computer and Information Literacy Study*, e.g., [Ihme et al., 2017](#)). These ILSA's, as well as many national assessment programs such as, for instance, NAEP (*National Assessment of Educational Progress*, e.g., [Bennett et al., 2008](#)) in the U.S. and NEPS (*National Educational Panel Study*, e.g., [Kroehne and Martens, 2011](#)) in Germany, have changed from paper-based to computer-based assessment in the past year(s).

Assessment of and for Learning: CBA can be used for *summative* and *formative* assessment (e.g., [Tomasik et al., 2018](#)) and, when combined with feedback (see section 2.9), is also helpful for self-assessments. It thus goes beyond assessments of learning and can also be used in assessments *for* learning ([Wiliam, 2011](#)).

Testing on the Internet: Computer-based assessment, of course, does not only occur within large-scale educational assessments. Instead, cognitive and non-cognitive instruments in various disciplines of the social and behavioral sciences are administered computer-based, online via the internet, or with the help of mobile devices in research and application. While from a technical perspective, different frameworks for creating (native) user interfaces exist for the various platforms, *HTML5/JavaScript* has emerged as a universally available concept. So, although there are also specific requirements for each of the different delivery formats (see chapter 7) in the context of assessments, a considerable overlap can be observed for the presentation of assessment content in *HTML5/JavaScript*, in the tradition of what was called *psychological testing and assessment on the Internet* (e.g., [Naglieri et al., 2004](#)).

In the following, advantages of computer-based assessments in comparison to paper-based testing (section 2.1) are described, followed by typical challenges, in particular, with respect to the comparability of assessments using classical response formats (section 2.2). This will be followed by a section on innovative item types (section 2.3), meaning item types there are only available for computer-based assessments. Different forms of item presentation and navigation between items are presented in section 2.4).



How to do this with the CBA ItemBuilder? In this chapter, small sections under the heading ‘How to do this with the CBA ItemBuilder?’ refer to other text parts in the following chapters that help to relate the theoretical concepts to practical applications.

2.1 Advantages & Benefits of CBA



Using CBA for the measurement of individual differences or group differences in various personal characteristics such as skills, competencies, personality dimensions, motivation, or attitudes is often motivated by potential benefits that will be briefly mentioned in this section (see, for instance, [Haigh, 2010](#)).

Standardization: Among the apparent advantages of CBA is that the comparability of assessments can be increased through an increased degree of standardization (e.g., [Jurecka, 2008](#)). In a computer-based assessment, the item authors can design diagnostic situations, defining which information is visible and accessible at which point (navigation restriction) and how test-takers can interact with the content to answer questions or construct responses. Carefully designed items allow standardizing, for instance, in which sequences tasks can be answered or if questions linked to an audio stimulus are accessible before the audio stimulus was played (navigation restriction). Item authors can also improve the common understanding of response formats by providing animated and interactive tutorial tasks (instructions). Audio and video components might also be used to reduce the reading load of items for all test-takers or as test-accommodation provided for selected test-takers only. Even simple answer formats can be designed in the computer-based assessment in such a way that invalid responses are impossible and test-takers are informed about possible handling errors, e.g. the selection of several answers in a single-choice task.



How to do this with the CBA ItemBuilder? Details about the use of audio and video elements can be found in section [3.10.3](#). More about animated instructions can be found in section [6.4.1](#) and an example illustrating the idea of *navigation restrictions* can be found in section [6.4.6](#).

Scoring: Various additional advantages are achieved by the possibility of instant scoring of closed response formats and selected text responses. (e.g., number inputs or text inputs that can be scored using simple rules like regular expressions and open text responses that can be scored using advanced NLP techniques). Automatically scored information derived from already administered items can be used for various purposes, either during the assessment or to simplify the post-processing of assessment data.



How to do this with the CBA ItemBuilder? Key for using the results of answered items directly is the so-called scoring definition, that can be defined for CBA ItemBuilder items within the tasks (see chapter [5](#)).

Instant Feedback: During a computer-based assessment, instant feedback is possible on the results, processes and time (see section [2.9.1](#)), the presentation of prompts, and a combination of tasks as scaffolding can improve data quality or implement *formative assessment* and assessment *for learning* (i.e., formative assessment). Immediately following the completion of an assessment, result reports can be generated and made available. Feedback can also refer to missing values in the assessment, for instance to reduce accidental overlooking of individual subtasks.



How to do this with the CBA ItemBuilder? Instant feedback and sequences of questions within tasks can be implemented using the conditional link feature (see section 4.3) or with the help of the so-called *Finite-State Machine* (see section 4.4). Feedback across tasks can be provided as part of the deployment software (see section 7.3.5 for an example).

Adaptivity & Measurement Efficiency: If scoring is already available for at least some of the items or questions during the test administration, various forms of adaptivity can be realized. The spectrum of possibilities ranges from hiding particular non-relevant items, simple skip rules, and filters to psychometric approaches such as multi-stage testing and one- or multidimensional adaptive testing as strategies of *Automated Test Assembly* that can result in an increased *Measurement Efficiency* (see section 2.7).



How to do this with the CBA ItemBuilder? Adaptivity within instruments can be implemented directly in the CBA ItemBuilder (see section 6.7 for an example), strategies of *Automated Test Assembly* require the use of specific deployment software (see section 7.2.7).

Innovative Items / Technology-Enhanced Items (TEI): Computer-based assessment promises to provide benefits for validity and construct representation of assessments using innovative item formats (e.g., Sireci and Zenisky, 2015; Wools et al., 2019) and technology-enhanced items (TEI, e.g., Bryant, 2017), using capabilities provided by the digital environments used for assessment (e.g., Parshall, 2002). Item formats that were not possible in paper-based assessment include drag-and-drop response formats, digital hypertext environments (Hahnel et al., 2022), performance-based assessment in simulated environments and authentic assessment (e.g., Frey et al., 2012) to game-based assessments and stealth assessment (e.g., Shute et al., 2016).



How to do this with the CBA ItemBuilder? Starting from freely designable pages, items can be contextualized in the CBA ItemBuilder and enhanced with a variety of interactive components, including hypertexts (see section 3.13.2) and drag-and-drop (see section 4.2.6). A large number of innovative items can be implemented directly with the components provided by the CBA ItemBuilder. In addition, the CBA ItemBuilder allows the integration of external content, e.g. to enable interactive response formats that cannot yet be created with the components in the CBA ItemBuilder (see section 3.14). Furthermore, items designed with the CBA ItemBuilder can also be integrated into other web-based content (see section 7.7).

Log Data & Process Indicators: Computer-based assessment as a method also provides insight into test processing through behavioral data (Goldhammer and Zehner, 2017), i.e., log data (gathered in the form of events) from which process indicators can be derived (Goldhammer et al., 2021). While log data can be collected using technical tools even with paper-based assessments (see, e.g., Dirk et al., 2017; Kroehne et al., 2019c), the availability and use of log-data from computer-based assessment has developed into a unique area of research (e.g., Zumbo and Hubley, 2017).



How to do this with the CBA ItemBuilder? Items created with the CBA ItemBuilder by default collect log events (see section 1.6.2 for the live preview in the *Trace Debugger*), and custom log events can be added if required. The CBA ItemBuilder aims at *replay-completeness* (as defined in Kroehne and Goldhammer, 2018) and the analysis of log data gathered with the CBA ItemBuilder is possible, for instance, with the LogFSM package (see section 2.8).

Response Times: A typical kind of information, which can also be understood as a specific type of process indicator, is the *Response Time*. Suppose the task design meets specific requirements (e.g., the *one item one screen*, OIOS, Reips, 2010), response times can be easily identified and may already be part of the programming of computer-based assessments. Response times can be used for various purposes, including improving the precision of ability estimates (e.g., *Time on Task* as used in Reis Costa et al., 2021). However, even when multiple tasks are administered within a unit, time information is available. In that case, item-level response times can either be derived using methods for analyzing log data (Kroehne and Goldhammer, 2018), or at least the *total time* for the entire unit or screen can be derived from computer-based assessments. A specific process indicator that can be derived using response times that allows the identification of disengaged test-taking and careless insufficient responding is *Rapid Guessing* and *Rapid Responding* (see section 2.5.3), a threat to validity, in particular, for low-stakes assessments. Response times allow monitoring test-taking engagement and can be used to investigate differences in test-taking motivation (e.g., Kroehne et al., 2019b).



How to do this with the CBA ItemBuilder? The CBA ItemBuilder runtime automatically provides the total time for each task and user-defined time measures can be computed during the assessment using an operator measuring the elapsed time (see section 4.4) or extracted from the log data (see section 2.8).

Online & Mobile Deployment: The manifold possibilities of Internet-based assessment were recognized early on (e.g., Buchanan, 2002; Bartram, 2005). Since the early years, the possibilities to conduct online assessment under similar conditions

have technically improved. For example, it is now possible to carry out assessments in full-screen mode and to register and record exits or interruptions in the log data, if not to prevent them. At the same time, however, the heterogeneity of Internet-enabled devices, tablets, and especially mobile phones has increased. Reliable and secure online and mobile assessments are therefore still a topic of current research and (further) developments.



How to do this with the CBA ItemBuilder? The CBA ItemBuilder runtime can be used to deliver tasks under a variety of online, mobile, and offline scenarios (see chapter 7 for more details on test delivery options).

CBA also results in changed costs for the assessments since the effort to create and test computer-based assessments can be higher (in particular for testing, see section 8.4), but the costs for the distribution of the computer-based administered instruments and the scoring of closed response formats, in particular, can be lower. However, most importantly content created for computer-based assessments can be shared and duplicated without additional costs. While these options obviously do not change the requirements for item protection and confidentiality (see section 2.10), especially concerning assessment content from large-scale assessments, they change how developed assessment instruments from research projects can be distributed and applied in practice (see section 8.7.4). All the potential benefits of CBA come with, for instance, practical challenges (e.g., Mills, 2002; Parshall, 2002), some of them will be discussed in section 6.


2.2 Standardized Response Formats



The existing standard *Question and Test Interoperability* (QTI)¹ defines simple items with one point of interaction. These simple items can be understood as the standardized form of classical response formats (see Figure 2.1 for an illustration).

¹Que (2022)

Illustration of selected interaction types as defined by the *Question and Test Interoperability* (QTI) standard:

| | |
|---|--|
| Choice Interaction (Multiple Responses) | Choice Interaction (Multiple Responses) |
| Choice Interaction (Single Response) | The Centre for technology-based assessment (TBA) is located at DIPF Leibniz Institute for Research and Information in Education, Frankfurt am Main, Germany. TBA is responsible for the CBA ItemBuilder, an item-authoring tool for computer-based assessment (CBA). |
| Text Entry Interaction | At which places is the DIPF located? |
| Extend Text Entry Interaction | <input type="checkbox"/> Bamberg <input type="checkbox"/> Berlin <input type="checkbox"/> Kiel <input type="checkbox"/> Frankfurt |
| GapMatch Interaction |  |
| Associate Interaction | |
| Order Interaction | |
| Match Interaction | |
| HotText Interaction | |
| Inline Choice Interaction | |
| Slider Interaction | |

Important note: The CBA ItemBuilder does not use the IMS QTI format, but you can create items that work analogously to the QTI interaction types.

[View as single page with scrolling ...](#)

FIGURE 2.1: Item illustrating QTI interactions for simple items ([html5lib](http://html5lib.org)).

Choice Interaction: The QTI *Choice Interaction* presents a collection of choices to a test-taker. The test-takers response is to select one or more of the choices, up to a maximum of `max-choices`. The choice interaction is always initialized with no choices selected. The behavior of QTI *Choice Interactions* regarding the number of selectable choices is described with the attributes `max-choices` and `min-choices`.



How to do this with the CBA ItemBuilder? Instead of `max-choices` and `min-choices` the CBA ItemBuilder differentiates between `RadioButtons` (single-choice, see section 3.9.2) and `Checkboxes` (multiple choice, see section 3.9.3), and the more general concept of `FrameSelectGroups` (see section 3.5.1).

The QTI standard only differentiates between orientation with the possible values `horizontal` and `vertical`, while the CBA ItemBuilder allows you to create any visual layout (including tables) with either `RadioButtons` and/or `Checkboxes` (see section

3.5.3).² Moreover, *QTI* allows to define the option `shuffle` to randomize the order of choices.³

(Extended) Text Entry Interaction: The *QTI Text Entry Interaction* is defined as an inline interaction that obtains a simple piece of text from the test-taker. The delivery engine (i.e., assessment platform) must allow the test-taker to review their choice within the context of the surrounding text. An example illustrating an item from Striewe and Kramer (2018) is shown in Figure 2.1. *QTI* uses specific so-called *class attributes* to define the width of text entry component. *QTI* defines the *Extended Text Interaction* for tasks where an extended amount of text needs to be entered as the response.



How to do this with the CBA ItemBuilder? Text responses can be collected with input fields (either single line or multiple lines, see section 3.9.1)⁴, while size and position can be defined visually in the *Page Editor* (see section 3.7). Regular expressions can be used to restrict possible characters (see section 6.1.3).

Gap Match Interaction / Associate Interaction / Order Interaction: *QTI* defines various interactions, that can be realized using drag and drop, such as *Gap Match Interaction*, *Associate Interaction*, and *Order Interaction* (see Figure 2.1). The interactions are defined by *QTI* for text and graph, depending on the nature of the elements.



How to do this with the CBA ItemBuilder? A concept that allows drag and drop of text, images and videos is implemented in the CBA ItemBuilder (see section 4.2.6). Using this approach, the interactions described by *QTI* can be implemented by a) visually placing the source and target elements on a page and b) configuring the drag and drop to either switch, copy or move values.

Match Interaction: Another interaction that not necessarily needs to be realized using drag and drop is the *Match Interaction*, that can also be computerized using components of type *Checkbox* as used for multiple choice responses formats (see Figure 2.1).

HotText / Inline Choice Interaction: For choice interactions embedded in context (e.g. text, image, etc.), the *QTI* standard defines two different interactions. Two possible implementations, with buttons and *ComboBoxes*, are shown in Figure 2.1. However, the response formats can also be implemented with other components,

²Note that also other components can be used to implement choice interactions, see section 3.9.7 and section 3.9.10 for examples.

³The `shuffle`-option is, if required, possible only with the help of dynamic components of the CBA ItemBuilder, as described in chapter 4 (see section 6.4.10 for an example).

for example, Checkboxes (multiple-choice) or RadioButtons (single-choice) for hot text interactions.

Slider Interaction: Response formats in which the possible answers cannot leave a predefined value range contribute to the standardization of assessments. QTI defines such a response format as *Slider Interaction* (see Figure 2.1).



How to do this with the CBA ItemBuilder? The CBA ItemBuilder has specific components that can be linked directly to variables (see section 4.2.2).

Additional Interactions Defined by QTI: The QTI standard defines additional interactions not illustrated in Figure 2.1. The *Media* interaction allows to add audio and video components to items, including a measurement how often the media object was experienced.



How to do this with the CBA ItemBuilder? The use of components to play audio and video files is described in section 3.10.3.

Hotspot interaction and *Position Object* interaction are graphical interactions that allow to select or position parts of images, while the *Drawing* interactions describes items in which test-taker can construct graphical responses.



How to do this with the CBA ItemBuilder? Simple graphical interactions can be implemented using *ImageMaps* (see section 3.9.10). For more advanced graphical response formats, the CBA ItemBuilder provides the concept of *ExternalPageFrames*, to embedd HTML5/JavaScript content (see section 3.14 and section 6.6 for examples).

QTI also defines the *Upload* interactions, that is more suited for learning environments and not necessarily for standardized computer-based assessment since uploading material from the computer might violate typical requirements regarding test security (see section 2.10).

PCI Interaction: As shown in Figure 2.1, the CBA ItemBuilder can be used to create items that correspond to the interactions defined by QTI. Figure 2.1 shows a single CBA ItemBuilder task, illustrating two different ways of navigating between interactions. The *Innovative Item Types* described in section 2.3 below show additional examples beyond the possibilities defined in the QTI standard. To use such innovative items, e.g., tasks created with the CBA ItemBuilder, in QTI-based assessment platforms, QTI describes the *Portable Custom Interaction (PCI)*.



How to do this with the CBA ItemBuilder? CBA ItemBuilder tasks can be packaged in a way that allows using them in QTI-based assessment platforms, such as TAO (see section 7.4).

Combination of Interactions: Not all of the interactions standardized by QTI were available in paper-based assessments (PBA) mode. However, in particular, single- and multiple-choice interactions (for closed question types) and text entry interactions (for constructed written responses) were used extensively in PBA, meaning printed and distributed across multiple pages.



How to do this with the CBA ItemBuilder? The CBA ItemBuilder uses the concept of pages on which components are placed to implement individual interactions to structure assessments into items, tasks and units (see section 3.4). This makes it possible to implement *Choice Interactions* and *Text Entry Interactions*, among others, in any order, without the QTI interactions themselves having to be provided by the CBA ItemBuilder (see Figure 2.1).

Beyond simple items, *Items* in general are defined by QTI as a set of interactions:⁵

For the purposes of QTI, an item is a set of interactions (possibly empty) collected together with any supporting material and an optional set of rules for converting the candidate's response(s) into assessment outcomes.

Distribution of Items on Pages: The QTI standard also provides some guideline how to split content with multiple interactions into items.⁶

To help determine whether or not a piece of assessment content that comprises multiple interactions should be represented as a single assessmentItem (known as a composite item in QTI) the strength of the relationship between the interactions should be examined. If

⁵https://www.imsglobal.org/question/quiv2p2/imsqti_v2p2_impl.html#h.lkeh7elhv2n

⁶http://www.imsglobal.org/question/quiv2p2/imsqti_v2p2_impl.html#3.1

they can stand alone then they may best be implemented as separate items, perhaps sharing a piece of stimulus material like a picture or a passage of text included as an object. If several interactions are closely related then they may belong in a composite item, but always consider the question of how easy it is for the candidate to keep track of the state of the item when it contains multiple related interactions. If the question requires the user to scroll a window on their computer screen just to see all the interactions then the item may be better re-written as several smaller related items.



How to do this with the CBA ItemBuilder? The CBA ItemBuilder provides a high degree of design freedom. Items do not have to be broken down into individual interactions. Components to capture responses can be freely placed on pages, and item authors can define how test-takers can switch between pages. However, the division of assessment content into individual components (referred to as *Tasks* in the context of the CBA ItemBuilder) is still necessary (see in detail in section 3.6).

Two key points for the computerization of assessments can be derived from the QTI standard:

- 1) The QTI standard defines basic interaction types. However, the combination of multiple items requires either *scrolling* (if all items are computerized using one page) or *paging* (i.e., additional interactive elements are required to allow the *navigation* between items, see section 2.4). The button `buttonView as single page with scrolling ...` in Figure 2.1 illustrates the two possibilities.



How to do this with the CBA ItemBuilder? As shown in Figure 2.1, the CBA ItemBuilder technically supports *paging* and *scrolling*. However, *paging* is preferred as it allows item designs in which only one task is visible at a time. Moreover, the structure created by item content, such as *Units*, is important to consider when distributing items across pages and deciding on the required navigation.

- 2) The standardization of computerized items goes beyond the definition of individual interactions. For instance, typical QTI items provide a *submit* button at the end of the page that contains one or multiple interactions

(and the submit button is essential to acknowledge, for instance, regarding the precise definition of response times, see section 2.2.2).⁷



How to do this with the CBA ItemBuilder? Assessment components created with the CBA ItemBuilder cannot be described with the QTI standard. However, the project files can be shared and archived, either with the actual task content or only as functional mockups with replaced content (referred to as *Mock Item*).

2.2.1 Mode Effects and Test-Equivalence

Items designed for collecting diagnostic information were printed on paper, and simple response formats such as choice interactions and text entry interactions were used that capture the products of test-takers answering items in paper-based assessments. As described in the 2.1 section, there are a number of advantages of computer-based assessment that distinguish this form of measurement from paper-based assessment. Until these advantages are exploited to a large extent and to confirm that existing knowledge about constructs and items holds, research into the comparability of paper- and computer-based assessment is essential (e.g., Bugbee, 1996; Clariana and Wallace, 2002).

Properties of Test Administration: As described by Kroehne and Martens (2011), different sources of potential mode effects can be distinguished, and a detailed description of *properties of test administrations* is required, since also different computerization are not necessarily identical. Hence, instead of the comparison between new (CBA) versus old (PBA), the investigation of different properties that are combined in any concrete assessments is required, for instance, to achieve reproducibility.

Item Difficulty and Construct Equivalence: A typical finding for mode effects in large-scale educational assessments is that items become more difficult when answered on a computer (e.g., for PISA, Robitzsch et al., 2020). From a conceptual point of view, a separation between the concept of mode effects and *Differential Item Functioning* (Feskens et al., 2019) might be possible, since properties of the test administration can be created by researchers and conditions with different properties can be randomly assigned to test-takers. Consequently, mode effects can be analyzed using random equivalent groups and assessments can be made comparable, even if all items change with respect to their item parameter (Buerger et al., 2019). When advantages available only in computer-based assessment are utilized, the issue of mode effects fades into the background in favor of the issue of construct equivalence (e.g., Buerger et al., 2016; Kroehne et al., 2019a).

Mode effects might affect not only the response correctness, but also the speed

⁷See the QTI standard for the *End Attempt* interaction.

in which test-taker read texts or, more generally, work in computer-based tests (Kroehne et al., 2019c), and mode effects can also affect rapid guessing (e.g., Kroehne et al., 2020), and might occur more subtly, for instance, concerning short text responses due to the difference in writing vs. typing (Zehner et al., 2018).

Further research and a systematic review regarding mode effects should cover typing vs. writing (for instance, with respect to capitalization, text production, etc., Jung et al., 2019), different text input methods such as hardware keyboard vs. touch keyboard, different pointing devices such as mouse vs. touch, and scrolling vs. paging (e.g., Haverkamp et al., 2022).

2.2.2 Response Times and Time on Task

Various terms are used in the literature to describe how fast responses are provided to questions, tasks, or items. Dating back to Galton and Spearman (see references in Kyllonen and Zu, 2016). *Reaction Time* measures have a long research tradition in the context of cognitive ability measurement. Prior to the computer-based assessments, response times were either self-reported time measures or time measures taken by proctors or test administrators (e.g., Ebel, 1953).

In recent years and in the context of computer-based assessment, *Response Time* is used to refer to time measures that can be understood as the time difference between the answer selection or answer submission and the onset of the item presentation (see Figure 2.2). However, a clear definition of how response times were operationalized in the computer-based assessments is missing in many publications (e.g., Schnipke and Scrams, 1997; Hornke, 2005). If log data are used to measure the time, the term *Time on Task* is used (see, for instance, Goldhammer et al., 2014; Naumann and Goldhammer, 2017; Reis Costa et al., 2021).

Response Time Operationalization Example

Response Time for single items and single visits as the time an item is visible on screen. Using the time in which an item is visible can result in time measures when no answer was given.

Example 1

A *Response Time* could also be defined as the time until the last response change. If no answer is given, this *Response Time* operationalization is NA.

Example 2

Further considerations on operationalization and specific definitions regarding the response time are necessary if items can be visited several times or if several items are presented on one page.

File: ResponseTimeOperationalizationExample.zip

FIGURE 2.2: Example illustrating different operationalizations of *Response Time* ([html](#)|[ib](#)).

Extreme Response Times and Idle Times: Response times have a natural lower limit. Short response times faster than expected for serious test-taking can be flagged using time thresholds if necessary (see section 2.5.3). Very long response times occur, for instance, if the test-taking process is (temporarily) interrupted or if the test-taking process takes place outside the assessment platform (in terms of thinking or note-taking on scratch paper). In particular, for unsupervised online test delivery (see section 7.2.1), long response times can also be caused by test-takers exiting the browser window or parallel interactions outside the test system. In order to gather information to allow informed trimming of long response times, it may be useful to record all interactions of test-takers that imply activity (such as mouse movements, keyboard strokes). Log events can then be used to detect *idle times* that occur when, for whatever reason, the test-taking is interrupted (see section 2.8).

Response Times in Questionnaires: As described in Kroehne et al. (2019a) without a precise definition, response times cannot be compared, and alternative operationalizations, for instance, the time difference between subsequent answer changes are possible. In survey research the term *Response Latency* is used (e.g., Mayerl, 2013), both for time measures taking by interviewers or by the assessment software. However, as described by Reips (2010), the interpretation of time measures require to know which task, question or item a test-taker or responded is working on, and additional assumptions are required if test-taker can freely navigate between tasks or see multiple questions per screen. With additional assumptions, item-level response times can, however, be extracted from log data, as illustrated for the example of item-batteries with multiple questions per screen in Figure 2.3.

The screenshot displays a web-based assessment interface. On the left, a questionnaire is shown with a question: "How much do you agree with the following statements?". Below the question, there is a list of statements and a Likert scale. On the right, a table titled "Extracted Measures (Time on State)" is visible. The table has two columns: "State" and "Time". The "State" column lists states such as "LOADED", "ANSWERCHANGE", and "UNLOADED". The "Time" column lists corresponding times. A highlighted yellow area in the table illustrates how collected log events can be used to distinguish between two types of states using a finite-state machine. Below the table, there is a section titled "The Average Answering Time (AAT)" which explains how AAT is calculated as the average of the times in the visited states Q_i . The text also mentions "Rapid Responding (RR)" and "Careless Responding and Insufficient Effort Responding (C/IER)". At the bottom of the interface, there are buttons for "(Reset)", "Additional explanation ...", and "Show link to this example as QR code".

FIGURE 2.3: Item illustrating *Average Answering Time* for item batteries ([html|ib](https://ib)).

Time Components from Log Data: Since there are now countless different computer-based tests, many software tools, and assessment implementations, the

concept of response times requires a more precise operationalization. One possible definition of time measures uses log events, as suggested as *Time on Task*. Various response time measures can be extracted from log data using methods developed for log file analyses (see section 2.8). Depending on the item design, all response time measures may require assumptions for interpretation. For example, if items can be visited multiple times, the times must be cumulated over visits. However, this rests on the assumption that the test-taker thinks about answering the task each time a task is visited. If multiple items are presented per screen and questions can be answered in any sequence, an assumption is necessary that each time a test-taker thinks about a question, this will be followed by an answer change.

2.2.3 Time Limits (Restrict Maximum Time)

Traditionally, in paper-based large scale assessments, *time limits* for tests or booklets were mainly used. Restricting the time for a particular test or booklet has the practical advantage that this type of time limit can also be controlled and implemented in group-based test administrations. A similar procedure can also be implemented in computer-based assessment. A time limit is defined for the processing of several items (e.g., for the complete test or a sub-test or section), and after the time limit has expired, the tasks for a particular test part can no longer be answered.

In contrast, however, computer-based testing also allows the implementation of time limits for individual tasks or small groups of items (e.g., units). The advantage is obvious: While time limits at the test level or booklet level can result various reasons to large inter-individual differences in the number of visited items (for instance, due to individual test-taking strategies or individual items that a test person is stuck on), time limits at item level can be implemented in such a way that all persons can see each item for at least a certain amount of time. Computer-based assessment allows to differentiate between time limits at the item level and time limits at test level. In between, time limits for item bundles, e.g., units, can be created. If the comparability of psychometric properties of an assessment to an earlier paper-based form is not necessary or if this comparability can be established, for example, on the basis of a linking study, then time limits can be used purposefully in the computer-based assessment to design the data collection. For example, if a test is administered in only one predetermined order (i.e., no booklet design), time limits at the test level will result in not-reached items depending on the item position.

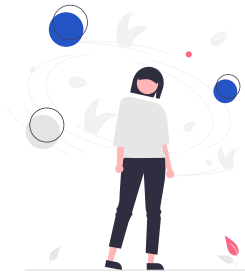


How to do this with the CBA ItemBuilder? Time limits for tasks (i.e., single items or group of items) can be implemented with the CBA ItemBuilder, for example (see section 6.4.5). Time limits across multiple items can be defined at the level of the test delivery (see section 7.2.8).

Time limits do not only restrict test-taking. Time limits also give the test-takers

feedback on their individual chosen pace of working on the task (e.g., [Goldhammer, 2015](#)). Different possibilities to give feedback during the assessment are described in section 2.9.1). The item design of *Blocked Item Response* (see section 2.4.1) can be used to force a minimum time for individual items.

2.3 Innovative Item Types / Technology-Enhanced Items (TEI)



New item formats for computer-based assessment are called *Innovative Item Formats* (e.g., [Sireci and Zenisky, 2015](#); [Wools et al., 2019](#)) or *Technology-Enhanced Items* (TEI, e.g., [Bryant, 2017](#)), and early attempts defined innovative item formats as items using capabilities not available in paper-based assessment (e.g., [Parshall, 2002](#)). Innovations through computer-based assessment were also described along the dimensions of *Complexity* (A=less complex to D=more complex) and *Constraintness* (1=fully selected to 7=fully constructed) (see Figure 2.4 and Figure 1 in [Scalise and Gifford, 2006](#)).

Since the early attempts to define innovative item formats, research practice has produced many forms of simulation-based, authentic, and interactive assessments in recent years. Examples created with the CBA ItemBuilder include:

- Assessment of ICT literacy with simulations (see, for instance, Fig. 6.2 in [Goldhammer and Kroehne, 2020](#))
- Evaluation of online information (see, for instance, Fig. 1 in [Hahnel et al., 2020](#), see also [Hahnel et al. \(2023\)](#))
- Complex problem solving (see, for instance, Fig 1 in [Tóth et al., 2014](#))
- Technical problem solving (see, for instance, Fig 1 in [Stemmann, 2016](#))
- Highlighting as response format (e.g., for PIAAC, [Schnitzler et al., 2013](#); and for the National Educational Panel Study, NEPS, [Heyne et al., 2020](#))
- Multiple Document Literacy (e.g., Fig 1 in [Hahnel et al., 2019](#))
- ... (to be continued) ...

Typical item formats are composed by single components, including Checkboxes, Radiobuttons, InputFields and ImageMaps.

Type 1A (Figure 2)

Type 1C (Figure 4)

Type 1D (Figure 5)

Type 2B (Figure 7)

Type 2C (Figure 8)

Type 2D (Figure 9)

Type 3A (Figure 10)

Type 3B (Figure 11)

Type 3C (Figure 12)

Type 4A (Figure 14)

Type 4B (Figure 15)

Type 4D (Figure 17)

Type 5A (Figure 18)

Type 5B (Figure 19)

Type 6D (Figure 25)

Examples from Scalise & Gifford (2006):

Type 1A: True/False

Every linear equation can be written in slope-intercept form.

☐ TRUE

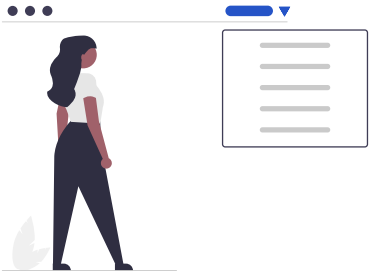
☐ FALSE

Next

Scalise, K., & Gifford, B. (2006). Computer-based assessment in E-learning: A framework for constructing "intermediate constraint" questions and tasks for technology platforms. *The Journal of Technology, Learning and Assessment*, 4(6).

FIGURE 2.4: Item illustrating different item formats described in (Scalise and Gifford, 2006, [html|ib](#)).

2.4 Item Presentation and Navigation



The term *Navigation* (or interface navigation) within assessments refers to the provided possibilities for test-takers to switch back and forth between items. Software tools and assessment platforms provide different interfaces for navigating between

Tasks, and as the ICT/ATP *Guidelines for Technology-Based Assessment* suggest,⁸ test-takers should be informed about navigating between tasks and have opportunities to practice navigation before the assessments.

Screen Layout: In order to be able to describe the options for navigation within assessments, one must first be aware of how the instructions and items in a computer-based assessment are presented on screen. Either only one item is displayed in full screen (*Full Page Items*, see section 2.4.1), or one or more additional visual components, for instance, for navigation, progress display, etc., are presented together with the item (*Integrated Item Presentation*, see section 2.4.6) on screen.

Combining multiple text entries, for instance, in *C-tests* (see section 2.4.3) or more general as so-called *Cloze* tests or in an *Embedded Answers* response format results in screen layout with multiple interactions per page. Text entry response formats (and other response formats as well) typically define one focused element, resulting in a *Within-Item Navigation*. Combining multiple items, for example, items with a common stimulus can also result in dependencies between assessment items. For this reason, it may be helpful to further distinguish between navigation within related items (referred to as *Units*, see *Within-Unit Navigation* in section 2.4.4) and navigation between items (see *Between-Unit / Test-Level Navigation* in section 2.4.5). Finally, for operational reasons, some assessment components have a unique role. Examples of this are so-called stop items, which are used in group tests to synchronize test processing in terms of time (see section 2.4.2).

Presentation Size: The planned screen layout determines how much space the assessment content can occupy (i.e., the required *Size* of items on screen). Screen size is usually specified in *pixels*. In addition, the *Resolution* (also measured in *Pixels*) must be distinguished from the actual size of the display, which depends on the physical size of the screens (usually specified in *Inches* of the diagonal). Finally, for the translation from *Pixels* (resolution) to *Inches* (display size), the pixel density as the amount of pixels per inch (often specified as *dpi*, i.e., dots per inch) and potential zoom factors must be taken into account. Zoom factors can be device-specific (i.e., a magnification implemented, for instance, by the operating system) or in browser-based deliveries also implemented by the web browser.



How to do this with the CBA ItemBuilder? A property *CBA Presentation Size* is defined for each assessment component created with the CBA ItemBuilder. As discussed in section 3.2.2, the *CBA Presentation Size* defines the aspect ratio of item material as presented on screen, while the test deployment software (see chapter 7) can proportionally scale the content to fit on screen).

Aspect Ratio and Screen Orientation: The ratio of the width to the height of a screen

⁸“Planning for TBAs should include the design of tutorials for navigation of test elements.”(International Test Commission and Association of Test Publishers, 2022)

is called the aspect ratio, resulting in terms like 4:3 (e.g., screens with a resolution of 1024x768 pixels), 16:9 (e.g., screens with a resolution of 1920x1080 pixels), etc., which are also used to characterize screens. Finally, the aspect ratio also determines the orientation of screens. If the width is larger than the height, one speaks of *Landscape Format*, if the height is larger than the width of *Portrait Mode*.

Windows Size and Browser Size: For web-based delivered assessments or other forms of delivery that are not standardized using a kiosk solution, the window size or the size of the browser must also be considered. In the worst case, this can cause the user to change the display size during test processing. On some operating systems, even for web-based deliveries, the browser full-screen mode can at least prevent the window size from changing.

Proportional Scaling: When assessments are not run on uniform hardware, determining the exact *Item Size* is often tricky in practice. It is even often impossible, especially if the item contents are only displayed embedded (see section 2.4.6) or if the available display area may even change due to the window size. This is countered by the demand to design as precisely as possible what the test-takers see in the context of a standardized assessment at a given time and how they can interact with the item material. One possible solution to this challenge is to implement *Proportional Scaling* of item content. This uses the available space on the screen while maintaining the aspect ratio until either height or width is exhausted.

Scrolling: An obvious way to display more content on a screen than the screen size allows is to use scrollbars. In fact, in HTML-based implementations of computer-based assessments, these often appear automatically. Vertical scrolling, in particular, should be used with caution for diagnostic reasons (see section 2.2.1). For detailed analyses, it may also be necessary to infer later from the data what a test-taker saw at a point in time (see section 2.8). Finally, it must be considered whether the entire screen or only parts of the display should scroll, for example, if a navigation area is to be permanently displayed on the screen.

Responsive Design: A more typical way for websites to deal with different resolutions and screen sizes is the use of so-called *Responsive Designs*. In *Responsive Designs*, the arrangement and display of the content are adapted to the actual available area (*View Port*). In the context of computer-based assessments, the use of *responsive designs* must be weighed up against the extent to which this can be reconciled with the goal of standardizing assessments.

2.4.1 Full Page Items

Full Page Items represent an item design in which minimal non-solution interactions with the assessment platform are required. *Full Page Items* can be implemented without scrolling and with scrolling, as shown in Figure 2.5. The central feature of *Full Page Items* is that they are presented exclusively on the screen, without being sur-

rounded by a navigation area. Necessary navigation elements are displayed within the item when they should be available.

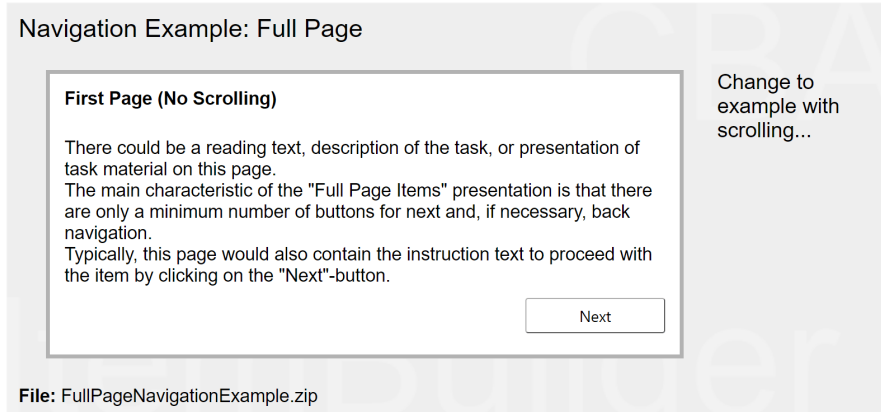


FIGURE 2.5: Example for full page navigation. ([html](#)|[ib](#)).

The design and presentation of items should always clarify whether they are instructional text or a question. In addition, it should be explained (either in the instruction at the beginning of the assessment or through short instructional texts) how the item can be processed and answered. If multiple pages are used, the navigation buttons must be repeated on each page.



How to do this with the CBA ItemBuilder? The creation of *Full Page Items* is the standard procedure for the CBA ItemBuilder when only the necessary navigation elements for switching between pages and finishing the task (see *Next-Task* command in section 3.12.1) are placed on each page.

Forced Choice: A unique response format in which items are typically displayed without navigation is *Forced Choice*. In this format, which is also used for questionnaires, the test takers are not offered the option of not answering. As shown in the examples in Figure 2.6, the system automatically navigates to the next question as soon as an answer is given.

Multiple forced choice questions can also be presented on one page and still be administered in a fixed order, as the example in Figure 2.6 shows.

Blocked Item Response In cognitive and non-cognitive assessments, rapid responding (see *Rapid Guessing* and *Rapid Responding* in section 2.5.3) is phenomena that compromise the validity of responses. A naive way to limit response elicitation is to disable the buttons for a limited time. Example 3 in Figure 2.6 shows this concept in combination with the forced-choice response format, Figure 2.8 shows this concept with other response formats (see also Persic-Beck et al., 2022).

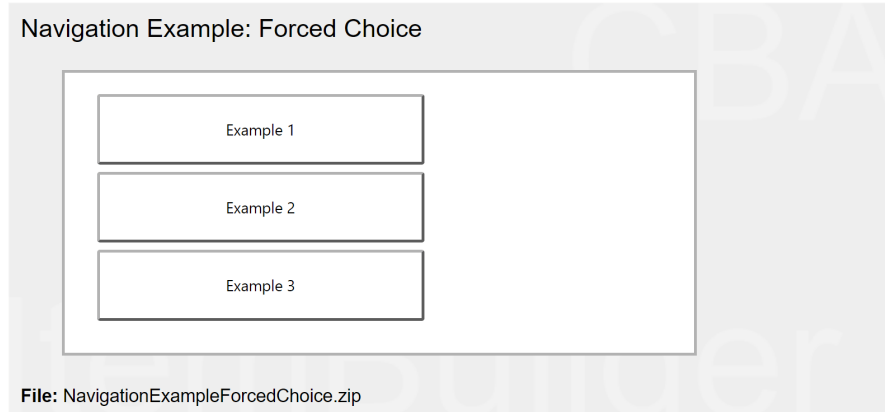


FIGURE 2.6: Example for forced choice navigation. ([html](#)|[ib](#)).

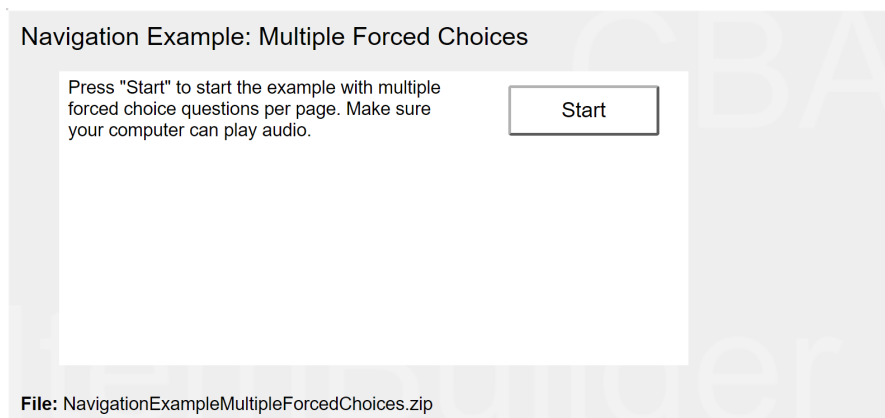


FIGURE 2.7: Example for multiple forced choice questions per page. ([html](#)|[ib](#)).

2.4.2 Breaks / Stopp Items

For different operational reasons, it may be necessary to interrupt and pause the flow of computerized instruments or to insert positions where test-takers should take a break. According to the logic described in this section, these pause pages are *Full Page Items*, since no further navigation should be possible.

Magic Word: In school assessments, test administration is often organized as on-site group tests with a test administrator present in the room. If the assessment is administrated using secure offline environments (e.g. in a kiosk mode, see 7.2.3), breaks for the group can be easily implemented with a *Magic Word* (i.e., a piece of information that functions as a password), which is only announced by the test administrator at the time when every test-takers should start with a following section.

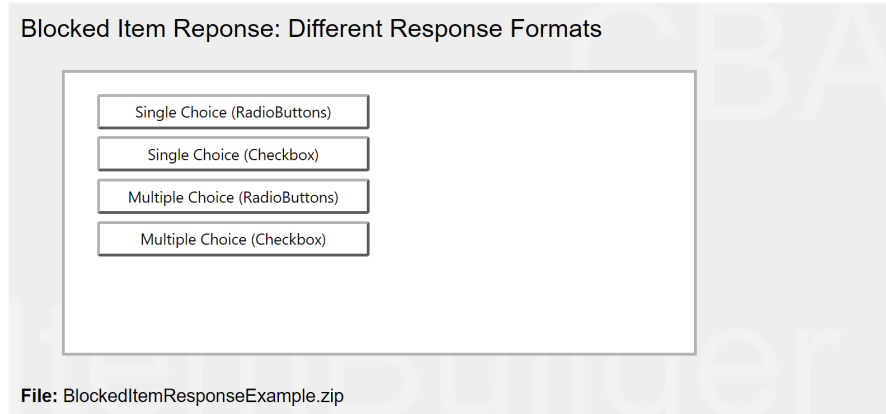


FIGURE 2.8: Examples for *Blocked Item Response* with different response formats ([html](#)|[ib](#)).

Dashboards: Computer-based assessments are also often administered in network environments (e.g., web-based), enabling centralized control across test-takers. So-called *Dashboards* for test management can show, for example, which task a test-taker is currently working on or whether a test taker needs support. Dashboards can also be used for the control of group testing or for monitoring online test administrations, for instance, by remote interviewers.

2.4.3 Within-Item Navigation

As soon as several input fields or components with an input focus are placed on a page, the possibility or the necessity for a *Within-Item Navigation* arises. Only particular response formats, such as *Point and Click* response formats (see, for instance, section 3.9.10 for so-called *ImageMaps*) and *Drag and Drop* response formats (see section 4.2.6) do not work with input focus.

C-Test (Text Completion Test): In the C-Test (Text Completion Test) in Figure 2.9,⁹, the text entries can be made in any order. Nevertheless, switching from one gap to the next with the tab key is also possible to allow fast editing.



How to do this with the CBA ItemBuilder? If many input fields are combined on one page, as shown in the example in Figure 2.9, then controlling the tab order and setting your input focus to the first text field is recommended. Item authors can define the tab order (see section 3.7.7), and input focus can also be set with a particular operator (see section 4.4.6).

⁹The content for this example is taken from Baghaei and Tabatabaee (2015)

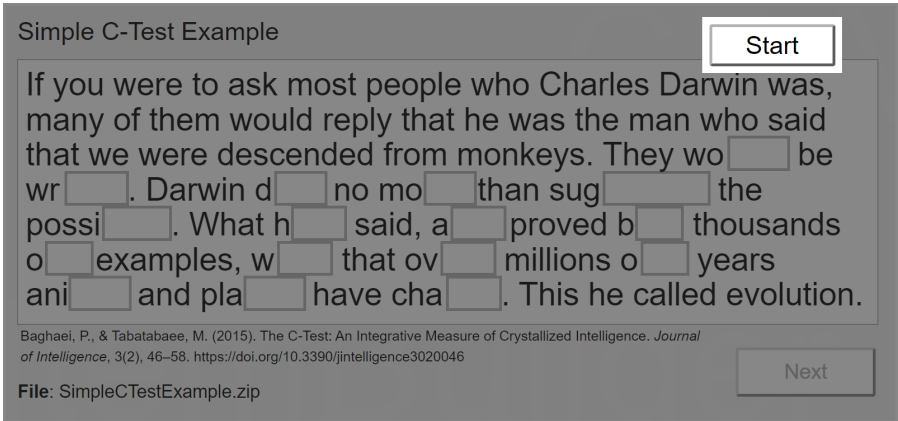


FIGURE 2.9: Example for a C-Test / Text Completion Test ([html](#)|[ib](#)).

If more than one answer can be given on a page, it may also make sense not to restrict the processing order due to construct-related considerations. An example of a test originally developed for paper-based assessment is shown in Figure 2.10. In this so-called *Digit Symbol Substitution Test* (representation based on Jaeger, 2018), it is implicitly assumed that the test is processed in sequential order.

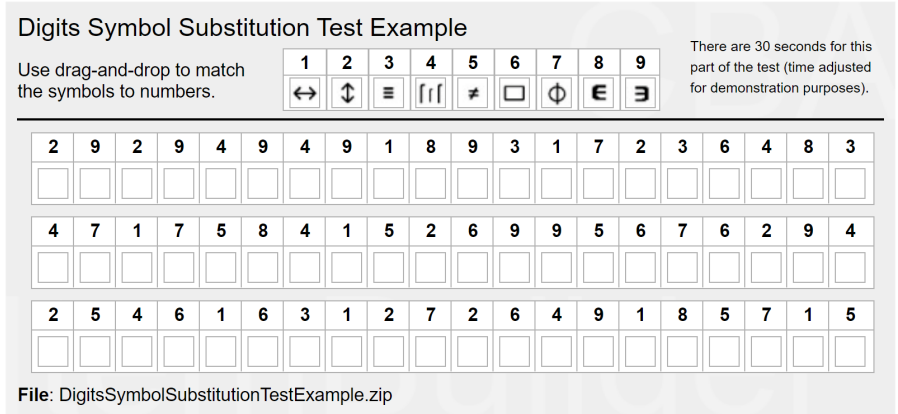


FIGURE 2.10: Example for a *Digits Symbol Substitution Test* ([html](#)|[ib](#)).

In the computerization with the CBA ItemBuilder shown in Figure 2.10, the processing can happen in any order and it must be additionally explained in the instruction whether only the consecutive items should be evaluated.

2.4.4 Within-Unit Navigation

When multiple tasks share a common stimulus, the tasks and the stimulus material are often referred to as a *Unit*. Within *Units*, dependencies can arise, i.e., answering individual subtasks is influenced by preceding or subsequent subtasks. Accordingly, computerization often allows subtasks to be answered and re-answered in any order, which leads to so-called *Within-Unit*-navigation. Figure 2.11 illustrates an example for within-unit navigation based on the separation between *Question and Stimulus*, as used, for instance, in the *Programme for International Assessment of Adult Competencies* (PIAAC, see Figure 2.2. in OECD, 2019, that shows the PIAAC user interface).¹⁰

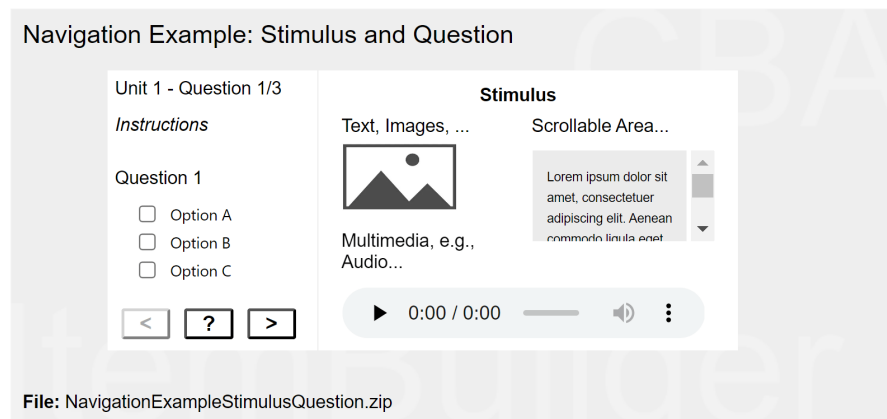


FIGURE 2.11: Example for question and stimulus navigation ([html](#)|[ib](#)).

The left side in Figure 2.11 shows the individual questions, which can be navigated between using the > and < buttons. The split stimulus can be seen permanently on the right side in Figure 2.11. The stimulus itself can provide further opportunities for interaction for complex interactive items. The ? button indicates how additional editing help can be integrated into the unit.



How to do this with the CBA ItemBuilder? The separation of items into different areas, e.g., a stimulus area and individual task-pages as shown in Figure 2.11, is possible in the CBA ItemBuilder by using either special page types (e.g., *X-Pages*, see section 3.4.2) or by using partial pages (e.g., *PageAreas*, see section 3.5.4).

¹⁰The example item [NavigationExampleStimulusQuestion.zip](#) is created using simple pages (see section 3.4.1), components of type *PageArea* (see section 3.5.4) and a navigation implemented completely in the finite-state machine (see section 4.4).

2.4.5 Between-Unit / Test-Level Navigation

For technical reasons, there is no need to restrict navigation within computerized tests. Just as test-takers could flip back and forth in paper-based test booklets, they could do the same in computer-based assessments.

Figure 2.12 shows schematically how free navigation could look, illustrated for a unit with only one task, a unit with multiple tasks and scrolling, and a unit with a text stimulus and multiple tasks.

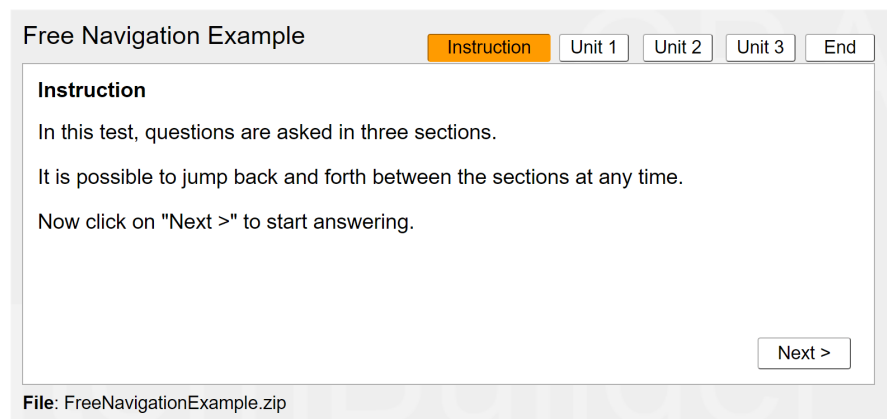


FIGURE 2.12: Examples for free *Between-Unit* navigation ([html](#)|[ib](#)).



How to do this with the CBA ItemBuilder? For implementing assessments with the CBA ItemBuilder, it must be acknowledged that completely free navigation with the means of the CBA ItemBuilder is only possible within *Tasks*. Each CBA ItemBuilder *Task* can request navigation to a next and a previous *Task* (see *Runtime Commands* in section 3.12), but the delivery software is responsible for responding to these requests. However, delivery software can also provide an additional user interface that can be used to drive any CBA ItemBuilder *Tasks* (see *Integrated Item Presentation* in the next section).

From a psychometric/diagnostic perspective, it should be considered what reason free navigation is necessary and whether the advantages of this item presentation outweigh the disadvantages already present with paper-based assessment. Suppose test-takers can freely navigate between a more significant number of items. In that case, it can be assumed that the influence of *test-taking strategies* such as skipping items when needed and returning later, is increased.

In practice, the options for navigation are often restricted, either due to psychometric considerations (for instance, to support adaptive testing), or because of opera-

tional considerations (for instance, to simplify the implementation of time restriction). For instance, backward navigation across *Units* (i.e., the possibility to go back to a previous unit) can be disabled in CBA (see, for instance, [Cochran et al., 2020](#)). As shown in Figure 2.13, test-takers can move freely only *within* a unit. When test-taking requests to navigate forward from the last task or page of a unit, a pop-up message is displayed informing the test-taker that it will be impossible to return to the previous tasks.

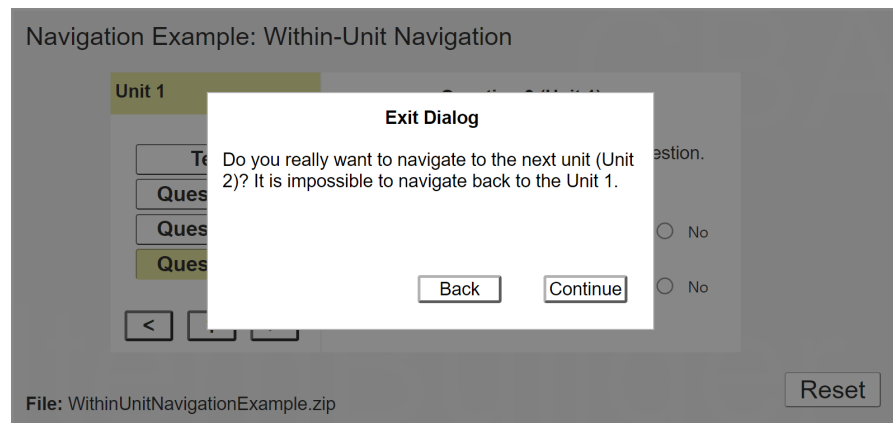


FIGURE 2.13: Examples for restricted *Between-Unit* navigation ([html](#)|[ib](#)).



How to do this with the CBA ItemBuilder? Feedback to test-takers about potential navigation restrictions is just one of the many options for providing targeted information during test-taking (see 2.9.1). They can be realized with the CBA ItemBuilder, for instance, using dialog pages (see section 3.15), so-called *Page Areas* (see section 3.5.4), or by so-called *Conditional Links* (see section 4.3).

2.4.6 Integrated Item Presentation

In various contexts, items are displayed and integrated within other web application or web-based deliveries. This includes, for instance, the integration of CBA ItemBuilder items within QTI-based deployment software (such as [TAO](#)) as *Portable Custom Interaction (PCI)* (see section 7.4), or more general embedding CBA ItemBuilder items using the so-called *TaskPlayer-API* (see section 7.7). Another possible scenario is the integration of assessments within learning management systems (e.g. Moodle) based on the LTI interface (see section 7.5.8).

In scenarios where assessment components are integrated into other deliveries, the following points need special consideration:

- **Presentation Size:** The size of the item displayed within other software environments is often smaller than the available space on the screen. Therefore, the items may need to be made smaller so that they can be answered reasonably in the embedded presentation. This can also result in the display of scroll bars on different screen sizes if the web application adjusts the size of the integrated items (responsive design).
- **Re-Visits / Restore:** The integration of items within external applications may result in items being terminated and re-entered by external navigation. To ensure a coherent picture for the test-takers and that the content shows the last working state on each new visit, the items must save their state before exiting.
- **Full Screen:** Full-screen presentation control must take into account when items are integrated into other web applications.



How to do this with the CBA ItemBuilder? The *CBA Presentation Size* must be defined for each CBA ItemBuilder project (see section 3.2.2). The *TaskPlayer-API* allows to store the item state at each point in time (see section 7.7). Commands to controll full screen presentation (see section 3.12.3) might not work in *Integrated Item Presentation*.

Suppose assessment components, which themselves allow inner navigation or should not be terminated at any time, are integrated into other environments (e.g., via PCI in TAO, see section 7.4). In that case, the navigation in the external application must be blocked.

2.5 Scoring and Calibration



Many psychometric CBA applications rests on principles of psychological and educational measurement (see, e.g., Veldkamp and Sluiter, 2019) and typical design principles for tasks/items and tests that also apply to non-technology-based assessments

are still valid (see, e.g., [Downing and Haladyna, 2006](#)). For instance, approaches to increase the measurement efficiency use a particular item response theory (IRT) model and (pre-)calibrated item parameters (see also section 2.7):

- Difficulty Parameter or Threshold Parameters,
- Discrimination Parameter (Loading) and
- (Pseudo)Guessing Parameter.

Which item parameters are used to model the probability of a correct response (or a response in a particular category) as a function of a one- or multidimensional latent ability depends on the choice of a concrete IRT model (see, for instance, [Embretson and Reise, 2013](#), for a general introduction). [Bolt \(2016\)](#) list the following IRT models for CBA:

- Models for innovative item types (polytomous IRT models and multidimensional IRT models) and models related to testlet-based administration (see section 2.5.1)
- Models that attend to response times (see section 2.5.6)
- Models related to item and test security (see section 2.10.1)

Further areas of application for (IRT) models in the context of CBA include:

- Models to deal with missing values (see section 2.5.2)
- Models to deal with rapid guessing (see section 2.5.3)
- Models for automated item generation (see section 2.6)

Another class of IRT models that is often used with CBA data are *Cognitive Diagnostic Models* (CDM, see, e.g., [George and Robitzsch, 2015](#), for a tutorial). Additional information, available when data are collected with CBA, such as response times and more general *process data* (see section 2.8) can be used in cognitive diagnostic modeling (e.g., [Jiao et al., 2019](#)).

2.5.1 Scoring of Items

Computer-based assessment consists of a sequence of assessment components, i.e., instructions and between-screen prompts, and purposefully designed (digital) environments in which diagnostic evidence can be collected. The atomic parts for gathering diagnostic evidence are called items, typically consisting of a prompt (or a question) and the opportunity to provide a response. Items can include introductory text, graphics, tables, or other information required to respond to the prompt, or questions can refer to a shared stimulus (creating a hierarchical structure called units). The *raw response* provided to items is typically translated into a numerical value called *Score*.

Dichotomous vs. Polytomous Scoring: For the use of responses to determine a person's score, a distinction is commonly made between dichotomous (incorrect vs. correct) and polytomous (e.g., no credit, partial credit, full credit). While this distinction is essential, for example, for IRT models, these two kinds are not mutually exclusive concerning the use of information collected in computer-based assessment. Identical responses can be scored differently depending on the intended use. For instance, a polytomous score can be used for ability estimation, while multiple dichotomous indicators for specific potential responses can provide additional insight in formative assessment scenarios.



How to do this with the CBA ItemBuilder? The concept implemented to score responses provided in CBA ItemBuilder tasks (see chapter 5) allows to define so-called *Scoring Conditions* (see section 5.3.2). These conditions provide the basic evidence, that can be used to create either dichotomous (only one out of two possible conditions) or poloytomous (one out of multiple possible conditions) variables (called *Classes* in the CBA ItemBuilder terminology, see also section 1.5).

Multiple Attempts & Answer-Until-Correct: In computer-based assessments, the *process* of the test-taking and responding can also be included in the scoring. This allows scenarios in which test-takers can answer an item multiple times (e.g., Attali, 2011) or until correct (e.g., DiBattista, 2013; Slepko and Godfrey, 2019). While this goes beyond the simple IRT models, it can be included in addition to a traditional scoring of the (first) attempt or the final response prior to a provided feedback (see section 2.9).



How to do this with the CBA ItemBuilder? The scoring options provided by the CBA ItemBuilder are rich and might look complex or confusing at first sight. The perceived complexity is because the CBA ItemBuilder disentangles the components used to create the interactive environments (required for test-takers to respond, see chapters 3 and 4) from the formulation of scoring conditions (required to translate the provided responses into scores, see chapter 5).

Constructed Response Scoring: Scores can be calculated automatically for closed response formats (i.e., items that require selecting one or multiple presented response options). For response formats beyond that, scoring can require human raters, pattern recognition (e.g., using *Regular Expressions*, see section 6.1), or *Natural Language Processing* (NLP) techniques and machine learning (typically using some human-scored training data, see, for instance, Yan et al., 2020).



How to do this with the CBA ItemBuilder? The CBA ItemBuilder can use *Regular Expressions* for scoring text entry (see section 5.3.4) and allows to convert text entry into result variables (see section 5.3.10) that can be used (outside) of the CBA ItemBuilder *Runtime* to score constructed response items.

Scoring of Complex Tasks: Special scoring requirements may arise for complex tasks that consist of multiple subtasks or where multiple behavior-based indicators are derived. Possible approaches for scoring multiple responses include aggregating the total number of correct responses, used, for instance, to score C-tests as shown in Figure 2.9 (see, for instance, [Harsch and Hartig, 2016](#), for an example). If (complex) items share a common stimulus or if for any other reason responses to individual items influence each other, dependencies may occur that require psychometric treatment (see, for instance, the *Testlet Response Theory* in [Williamson et al., 2006](#)).



How to do this with the CBA ItemBuilder? For scoring complex items, the CBA ItemBuilder allows the define pieces of evidence in terms of (many different) properties of the test-taking process and responses to the different interactive elements. Multiple individual *Scoring Conditions* can be combined using logical operators to formulate (intermediate) scoring variables.

The scoring of answers is, among other things, the basis for automatic procedures of (adaptive) test assembly (see section 2.7) and for various forms of feedback (see section 2.9), among others regarding the completeness of test processing. For this purpose, a differentiated consideration of missing responses is also necessary, as described in the following sub-section.

2.5.2 Missing Values

Scoring can also take the position of items within assessments into account. This is typically done when differentiating between *Not Reached* items (i.e., responses missing at the end of a typically time-restricted section) versus *Omitted* responses (i.e., items without response followed by items with responses). Computer-based assessment can be used to further differentiate the *Types of Missing Responses*, resulting in the following list:

- **Omitted responses:** Questions skipped during the processing of a test lead to missing values typically described as *Omitted Responses*. How *committed responses* should be taken into account when estimating item parameters (see section 2.5.4) and estimating person parameters (see section 2.5.5) depends on the so-called *missing-data mechanisms* (see, for instance, [Rose et al., 2017](#)).

- **Not reached items:** If there is only a limited amount of time provided to test-takers to complete items in a particular test section, answers may be missing because the time limit has been reached. These missing responses are called *not reached items*. The arrangement of the items and the possibilities of navigation within the test section must be considered for the interpretation of missing values as *Not Reached*.
- **Quitting:** An example that tasks can be incorrectly classified as *Not Reached* is described in [Ulitzsch et al. \(2020\)](#). Missing values at the end of a test section can also indicate that the test was terminated if testing time was still available.
- **Not administered:** Missing responses to items that were not intended for individual test-takers, for instance, based on a booklet design (see section 2.7.2). Since these missing values depend on the test design (see section 2.7.2), they are often referred to as *Missing by Design*.
- **Filtered:** Items may also be missing because previous responses resulted in the exclusion of a question.

Missing Value Coding: In computerized assessment, there is no reason to wait until after data collection to classify missing responses. Features of interactive test-taking can be considered during testing to distinguish missing responses using the described categories as part of scoring (see chapter 5).



How to do this with the CBA ItemBuilder? The classification of missing values for assessments implemented with the CBA ItemBuilder requires differentiating two levels: Within individual assessment components, the scoring must be implemented in a way allowing to differentiate between *Not Reached* vs. *Omitted* responses. If applicable, missing responses due to filtered questions must also be coded accordingly. In addition, the software used to deliver the assessment components is expected to classify missing responses as *missing by design* (or *not administered*) if items in selected assessment components were not intended for individual test-takers. Finally, the delivery software must also classify missing responses as *not reached* due to a time limit, using the information about the test design and the scheduled item sequence.

Use of Log-Data for Missing Value Coding: An even more differentiated analysis of missing responses is possible by taking log data into account. The incorporation of response times for the coding of omitted responses (e.g., [Frey et al., 2018](#)) is one example for the use of information extracted from log data (see section 2.8). Response elements that have a default value require special attention. For instance, *checkboxes* (see section 3.9.3) used for multiple-choice questions have an interpretation regarding the selection without any interaction (typically de-selected). Log data can be used to differentiate whether an item with multiple (un-selected) checkboxes has been solved or the item should be coded as a missing response.



How to do this with the CBA ItemBuilder? Information from individual events during the test-taking process can be included using so-called scoring operators, which can also account for variables (see section 4.2) and internal states of the item (see *Finite-State Machine* in section 4.4). Since these events are also captured in CBA ItemBuilder log events (see section 2.8), a fine-grained differentiation of missing values is also possible as part of a log data analysis.

Missing responses can provide additional information regarding the measured construct, and their occurrence may be related to test-taking strategies. As described in the next section, *rapid* missing responses may also be part of a more general response process that is informative about test-taking engagement.

2.5.3 Rapid Guessing

Computer-based assessment can make different types of test-taking behaviors visible. A simple differentiation into *solution behavior* and *rapid guessing* was found to be beneficial (Schnipke and Scrams, 1997), that can be applied, when *response times* (as discussed in the previous section 2.2.2) are available for each item or when an item design is used that allows interpreting individual time components (see section 2.8.2). Rapid guessing is particularly important for low-stakes assessments (such as PISA, Rios and Soland, 2022, and PIAAC, Goldhammer et al. (2017)).

While solution behavior describes the (intentional) process of deliberate responding, a second process of very fast responding can be observed in many data sets. Since both processes can often be clearly separated when inspecting the response time distribution, a *Bimodal* has become a central validity argument (see, for instance, Wise, 2017) for focusing on *Rapid Guessing* as a distinguished response process. Using the bimodal response time distribution, a time threshold can be derived, and various methods exist for threshold identification (e.g., Soland et al., 2021), using either response time and or (in addition to) other information-based criteria (e.g., Wise, 2019).



How to do this with the CBA ItemBuilder? The interpretation of time measures requires an appropriate item design (see section 2.8.2). If time components can be identified as *response time*, time thresholds can be used to classify responses as *Rapid Guessing*, either during the assessment or as part of the data post-processing (see section 8.6). If the implemented between-item navigation (see section 2.4) allows to measure time for item omission, *Rapid Omission* can also be classified as additional indicator for low test-taking engagement.

Alternatively to simple time thresholds, mixture modeling (e.g., Schnipke and

Scrams, 1997; Lu et al., 2019) can be used to differentiate between solution behavior and rapid guessing when post-processing the data. Treatments of rapid responses include response-level or test-taker-level *filtering* (see Rios et al., 2017, for a comparison). However, similar to missing values (see above), a treatment of responses identified as rapid guessing might require to take the missing mechanism into account (e.g., Deribo et al., 2021). Further research is required regarding the operationalization of rapid guessing for complex items (see, e.g., Sahin and Colvin, 2020, for a first step in that direction) and validating responses identified as *Rapid Guessing* (e.g., Ulitzsch et al., 2021a). Another area of current research is the transfer of response-time-based methods to identify *Rapid Guessing* to non-cognitive instruments and the exploration of *Rapid Responding* as part of *Careless Insufficient Effort Responding* (CIER), either using time thresholds or based on mixture modeling (e.g., Ulitzsch et al., 2021b).

2.5.4 Calibration of Items

After constructing a set of new assessment tasks (i.e., single items or units), the items are often administered in a pilot study (often called *calibration study*). Subsequently, a sub-set of items is selected that measures a (latent) construct of interest in a comparable way, where the selection of items is typically guided in the context of the *Item Response Theory* (see, e.g., Partchev, 2004) regarding *Item Fit*, and so-called *Item Parameters* are estimated. Different tools and, for instance, R packages such as TAM (Robitzsch et al., 2022) can be used to estimate item parameters and to compute (item) fit indices.



How to do this with the CBA ItemBuilder? Item parameters are not stored within the CBA ItemBuilder project files because identical items might be used with different parameters, for instance, to compensate for *Differential Item Functioning* (DIF) or to acknowledge *Parameter Drift* (PD).

Missing values can be scored in different ways for item calibration and ability estimation (see Robitzsch and Lüdtke, 2022, for a discussion), depending, for instance, on assumptions regarding the latent missing propensity (see, for instance, Koehler et al., 2014). The treatment of *rapid guessing* can improve item parameter estimation (e.g., Rios and Soland, 2021; Rios, 2022).

IRT models exist for dichotomous and polytomous items (see section 2.5.1). When multiple constructs are collected together, multidimensional IRT models can increase measurement efficiency (see, e.g., Kroehne et al., 2014).

Known item parameters are a prerequisite for increasing measurement efficiency through automatic test assembly and adaptive testing procedures (see section 2.7), and techniques such as the *Continuous Calibration Strategy* (Fink et al., 2018) can help to create new *Item Pools*.



How to do this with the CBA ItemBuilder? The accuracy of item parameters can be relevant, for instance, if adaptive testing is used that assumes known (and fixed) item parameters. For these applications, the calibration study should be performed with similar tools as the primary assessment to avoid mode effects due to different administration modes (i.e., paper vs. computer) or due to different computerization (see section 2.2.1).

Item parameters are only valid as long as the item remains unchanged. This limits the possibilities for customizing items, even if they are shared as *Open Educational Resources* (OER, see section 8.7.4).

2.5.5 Ability Estimation

While the estimation of item parameters is typically done outside the assessment software as part of test construction, the computation of a raw score (e.g., the number of items solved) or the estimation of a (preliminary) person-ability (using IRT and based on known item parameters) is a prerequisite for the implementation of methods to increase measurement efficiency (multi-stage testing or adaptive testing, see section 2.7). Rapid guessing (see section 2.5.3, e.g., [Wise and DeMars, 2006](#)) as well as informed guessing can be acknowledged when estimating person parameters (e.g., [Sideridis and Alahmadi, 2022](#)).



How to do this with the CBA ItemBuilder? Classes with assigned scoring conditions are defined in the CBA ItemBuilder without specifying how the resulting variables will be used to score the test-taker's ability. For this purpose, an additional codebook should be part of the test delivery software, which assigns numerical values to the (nominal) scoring conditions. By separating the (nominal scaled) scoring conditions defined within CBA ItemBuilder tasks and the nominal or ordinal scaled weights, CBA ItemBuilder projects can remain unchanged even when switching between different ways of using item scores based on empirical results (e.g., model comparison).

2.5.6 Incorporation of Response Times

A long research tradition deals with the incorporation of response times in psychometric models. Based on the *hierarchical modeling of responses and response times* ([van der Linden, 2007](#)) response times can be used, for instance, as collateral information for the estimation of item- and person-parameters ([van der Linden et al., 2010](#)). Response times (and more generally, *Process Indicators*, see section 2.8) used to improve

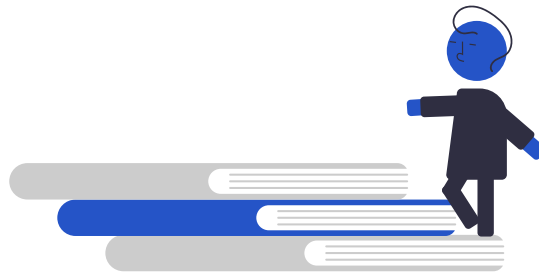
item response theory latent regression models (Reis Costa et al., 2021; Shin et al., 2022). In combination with *missing responses* response-time related information (in terms of *not reached* items) can also be included in the ability estimation using polytomous scoring (Gorgun and Bulut, 2021).

2.6 Automated Item Generation



Automatic item generation (AIG) is used to describe the process of generating items using computer technology, typically using some kind of models (e.g., cognitive models, Gierl et al., 2012). A *template-based* approach (Gierl and Lai, 2013) formulates an item model (also called, for instance, schema or blueprint) containing the components of a task that can be varied to generate items. Item models can be described regarding the number of layers in which item *clones* (i.e., generated items) differ from a source. While AIG from an IRT perspective, for instance, generating items *on the fly* was suggested more than ten years ago (e.g., Embretson and Yang, 2006), current research incorporating machine learning techniques such as deep learning (e.g., von Davier, 2018) and models developed for natural language processing can be expected to provide promising new methods and applications (see for a review, e.g., Das et al., 2021; and for an example Attali et al., 2022).

2.7 (Automated) Test Assembly



When more items are available than can or should be completed by a test-taker, the term *Test Assembly* is used to describe the psychometric process of combining items to test *Booklets* or *Rotations*. The test assembly process usually requires items with known item parameters (see section 2.5.4) and can be performed manually or automatically (van der Linen, 2006).

The research literature on automated test composition provides insight into criteria that are considered when assembling tests. The primary criterion is typically provided by item response theory, i.e., the selection of items to optimize the measurement by taking already available information about the anticipated test-takers (for instance, the expected ability distribution) into account.

Constraints: Approaches that formalize the test assembly (e.g., Diao and van der Linden, 2011) can also incorporate additional criteria (i.e., constraints for the test assembly when conducted, for instance, in R, see Becker et al., 2021), such as:

- **Content:** Content areas or domains of requirements, defined as test specification in relation to an underlying assessment framework (i.e., the *test blueprint*)
- **Response Format:** Response format or number of response alternatives, or the position of the correct responses
- **Item Position:** Balancing the position of items or keeping the position of certain items constant (e.g. link items)
- **Response Time:** Expected time to solve the item (can be used to assemble tests that with comparable time limits)

Constraints can be considered in test composition to make different individual tests comparable or to balance and account for item properties at a sample or population level. Moreover, constraints might also be used to achieve further operational goals, such as the interpretability of adaptive tests at the group level for formative assessment purposes (e.g., Bengts et al., 2021).

2.7.1 Fixed Form Testing

Assessments with a fixed set of items can be seen as the typical use case for test deployments, either in the preparation of IRT-based applications of tailored testing (i.e., to collect data for item calibrations) or as the final output of test development. As shown in Panel A of Figure 2.14, a *fixed form testing* requires the administration of assessment components as linear sequence.

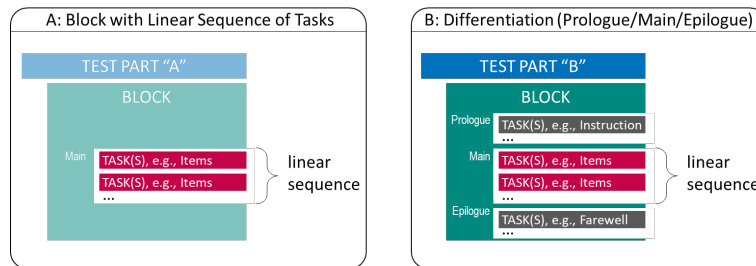


FIGURE 2.14: Fixed Form Testing with linear sequence of Tasks



How to do this with the CBA ItemBuilder? Simple uses of the TaskPlayer API (see section 7.7) or the integration of CBA ItemBuilder items to *Portable Custom Interactions* (PCI, see section 7.4) allow the use of one or more CBA ItemBuilder tasks administered in a predefined order.

Criteria for item selection, optionally taking into account constraints, is reflected in the selection of *Tasks* that are included in the linear sequence.

A first differentiation of the structure also of *Fixed Form* test assemblies (see Panel B in Figure 2.14) concerns the distinction in assessment components which are administered BEFORE the actual tasks (*Prologue*), the tasks themselves (*Main*), and the assessment components which are administered AFTER the main tasks (*Epilogue*). The subdivision made can be helpful if, for example, a time limit is required for a part of the assessment components (*Main*),¹¹ but the time measurement does not begin until the instruction is completed (*Prologue*) and a uniform test-taker enactment is to be implemented (*Epilogue*).

Test designs using *Fixed Form Testing* can also incorporate multiple tests, domains or groups of *Tasks* as shown in Figure 2.15, by repeating multiple *Test Blocks*.

The difference between Panel C (multiple *Test Blocks* within one *Test Part*) and Panel

¹¹While time limits within assessment components created with the CBA ItemBuilder can be defined directly with *Timed Events* using the *Finite-State Machine* as part of item implementation (see section 4.4.3, time limits that are to work across different *Tasks* must be implemented with the test deployment software (see section 7.2.8).

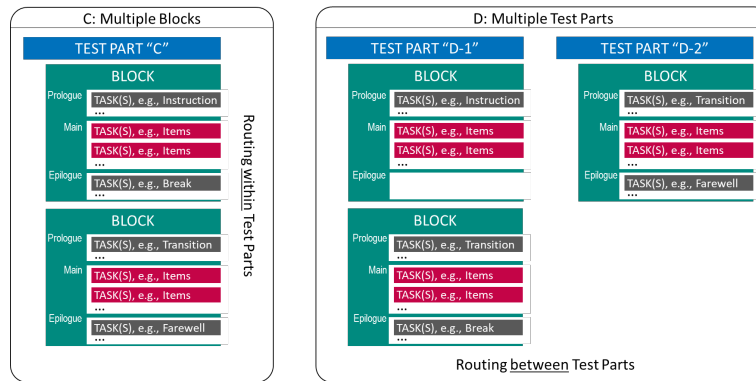


FIGURE 2.15: Fixed Form Testing with multiple *Blocks* or *Parts*.

D (multiple *Test Parts*) in Figure 2.15 is only cosmetic, as long as *Test Parts* are also administered in a linear sequence. However, test deployment software might add the possibility to define the *Routing* between *Test Parts* differently than *Routing* within *Test Parts*. Moreover, as soon as different technologies come into play, *Test Parts* might use test content created with different tools (if supported by the test deployment software). In a typical educational assessment, a specific part (often administered at the end of a session) is dedicated to an additional questionnaire, that could serve as the content of the *Block* shown in the right part of Panel D in Figure 2.15.

2.7.2 Booklet Designs and Rotations

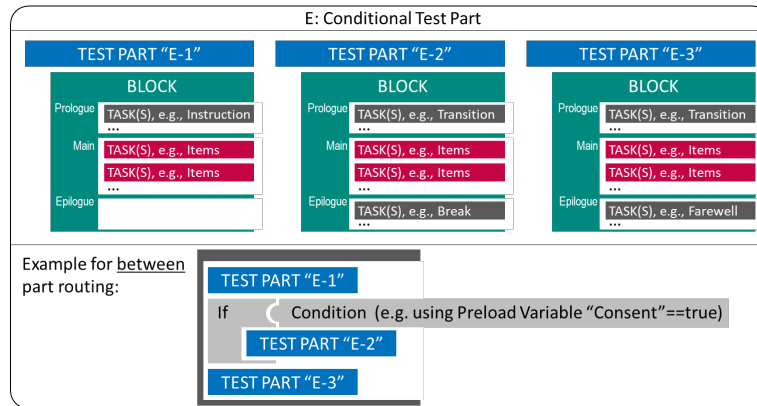
For various operational reasons, it may be necessary to define which *Test Parts* of a study definition are administered under which condition. In this way, for example, consent to test participation in different parts of an assessment can be incorporated, or the order of domains can be balanced or controlled.

Rotations of Test Content: One possibility to support such scenarios with a potential test delivery software is to allow conditional skipping (i.e., filtering) of *Test Parts*. A condition can be, for instance, a *Preload*-variable (i.e., a variable that contains information available about test-takers prior to an assessment; see section 7.5.3).

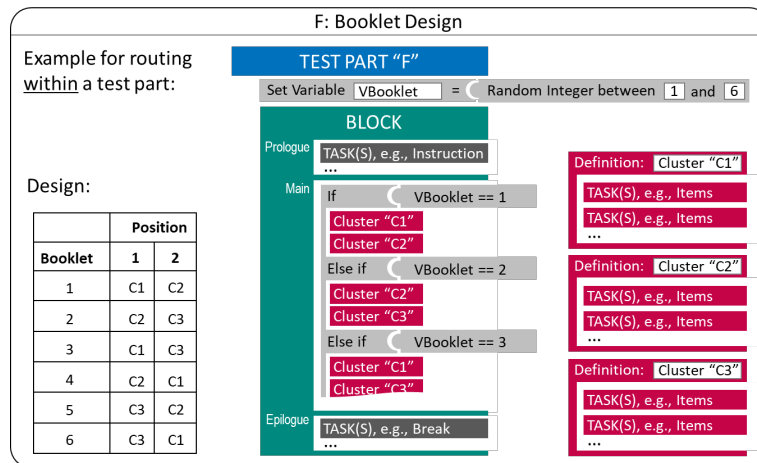
Figure 2.16 shows an example where a *Test Part* (“E-2”) is only administered if a hypothetical *Preload*-variable “Consent” has the value “true” (i.e., if, for example, parents have given their consent for a child to answer questions combined as *Test Part* “E-2”).

Using multiple *If-else-if*-conditions (or a *Switch*-condition), multiple *Rotations* can be implemented, for instance used to make identical *Test Parts* usable in different sequences.

Booklet Designs: In large-scale assessments, multiple test forms or *Booklets* are also used to balance items across students, for instance, to ensure content coverage (e.g.,

**FIGURE 2.16:** Conditional *Test Part* using between-part Routing.

Frey et al., 2009) or to link a huge amount of items (e.g., Yousfi and Böhme, 2012). Defining individual *Test Parts* for items or combination of items (called, for instance, *Clusters*) can become cumbersome. Instead, test deployment software can make use of the underlying structure that provides rationale for creating booklets, for instance, balancing the position of clusters in a permutation design (see Figure 2.17).

**FIGURE 2.17:** Example for a simple *Booklet Design* using within-part Routing.

The booklet design illustrated in Figure 2.17 has a random component (i.e., a random number is created during runtime for each test-taker and the value is assigned to the variable "VBooklet") that is used to select the order in which two *Clusters* are administered. The clusters are created statically by listing tasks in a particular sequence in a separate definition that is re-used in the test assembly.

Booklets with Targeted Difficulty: One use case for multiple booklets is to align test difficulty or length with prior information about the test-takers. For this purpose, variable(s) used in condition(s) to select the *Clusters* or *Tasks* to be administered can contain information about test-taker, provided to the test deployment software as so called *Preload*-variables. If the *Preload*-variables contain information gathered in longitudinal designs in a previous assessment, a simple form of multi-stage testing can be implemented (Pohl, 2013).

2.7.3 Multi-Stage Testing

If an (intermediate) scoring of at least some responses of previously administered items is feasible at runtime (i.e., if tasks contain items that can be automatically scored, see section 2.5.1), tailored testing becomes possible. A typical goal for multi-stage tests is to tailor the items' difficulty to the test takers' ability. Suppose there is no prior information that can be used as *Preload*-variables. In that case, this goal can be achieved by evaluating the test-taker's capability after administering a first set of tasks (a first stage in a test the combines multiple stages). As shown in Figure 2.18, as soon as a list of *Tasks* that constitute the first stage are administered, a variable "VStage1Score" can be computed that serves as the condition for a subsequent stage. In the most simplest form, a raw score is used as criterion, allowing to select the second stage by comparing the raw score of the first stage to a cut-off value.

Administration of *Tasks* within a *Stage* can allow test-takers to Navigate *between* units (see section 2.4.5), since the scoring is only done after the administration of all *Tasks* that create a *Stage*. In Figure 2.18 this is made explicit by illustrating the variable "VStage1ResultList", a list that contains all results gathered when administering the *Tasks* of the first *Stage*.

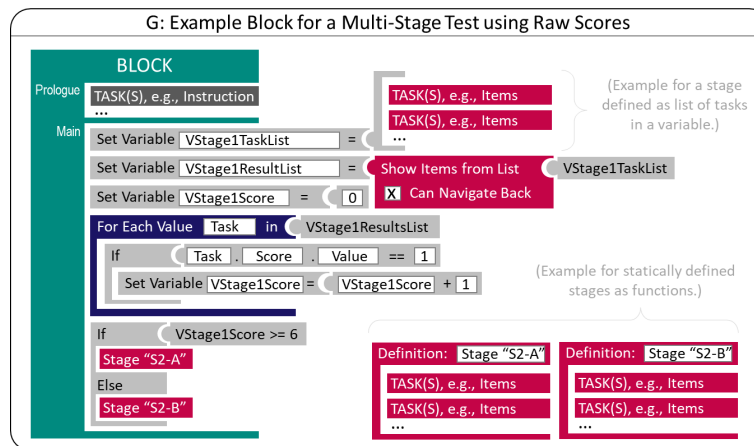


FIGURE 2.18: Basic Principle of Multi-Stage Testing

The tasks used for a particular stage¹² can be defined statically in the test deployment software (see the definition for Stage "S2-A" and Stage "S2-B" in Figure 2.18) or stored in a variable (i.e., a list with at least one *Task*, see "VStage1TaskList"). Having the stage definition not static in the test specification (i.e., the configuration of the deployment software) allows for advanced approaches such as *On-the-Fly Multistage Testing* (e.g., Zheng and Chang, 2015). A function that returns a list of tasks (selected from a larger pool of candidate items) based on the provisional estimation (temporary) or expected ability is required (see next section 2.7.4 about *Computerized Adaptive Testing*).

The list of results shown in Figure 2.18 can also be used in an IRT-based function for ability estimation (see section 2.5.5) if the raw score (e.g., the number of solved items) is not sufficient for routing between stages. An IRT ability estimate (i.e., the return value of an IRT-based function for ability estimation) is either a scalar representing a uni-dimensional ability estimate or a vector representing a multidimensional ability estimate.

2.7.4 Computerized Adaptive Testing

Computerized adaptive testing (CAT) is a method to increase measurement efficiency (see, for instance, Weiss, 1982) based on *Item Response Theory* (IRT). Either single items or sets of items (*Item Stages*) are selected with respect to an item selection criterion such as the *Maximum Fisher Information* for dichotomous items (see van der Linden and Glas, 2000, for an introduction), typically for a specific *Provisional Ability Estimate*. Adaptive testing can be illustrated as flow diagram as shown in Figure 2.19, based on a sequence of steps embedded into the *CAT Loop*.

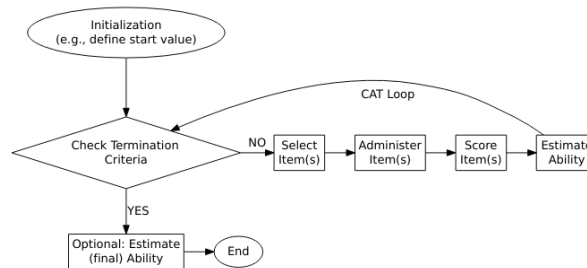


FIGURE 2.19: Simplified Illustration of *Computerized Adaptive Testing*.

CAT Algorithms (i.e., algorithms used for adaptive tests) administer items until a particular *Termination Criterion* is reached. *Termination Criteria* are created based on the

¹²The [QTI 3.0 Specification](#) refers to a list or batch of one or more items from the *Item Pool* as *Item Stage*.

Test Length as the number of administered items (resulting in *Fixed Length* test) or the accuracy of the ability estimate (resulting in *Variable Length* test) or combinations. Hence, after initializing the adaptive test, a loop (see keyword *While* in Figure 2.20) is used to make sure the adaptive algorithm is terminated not before the termination criteria are met. In operational adaptive tests multiple criteria (including, for instance, that no suitable item was found in the *Item Pool*) can be used.

Depending on the select IRT model used to calibrate the items in the *Item Pool* (see section 2.5.4), a *unidimensional* (i.e., a scalar) or *multidimensional* (i.e., a vector) ability estimate is used as *Start Value*, as *Provisional Ability Estimate* and as the *Final Ability Estimate*. During the *Initialization* of an adaptive test, prior information can be used to adopt the *Start Value* (i.e., the ability estimate that is used to select the first item(s) of the adaptive test). *Preload*-variables can be used, for instance, to assign group-specific *Start Values* (see Figure 2.20 for an example).¹³



How to do this with the CBA ItemBuilder? Multi-stage testing for simple use-cases can be implemented *within* CBA ItemBuilder *Tasks* (see, for instance, section 6.7.1). Multi-stage testing across different *Tasks* requires a test deployment software that provides the required *IRT* functions for ability estimation and, if used, for item selection (see, for instance, section 7.5) or the integration of statistical software such as R [see section 7.3].

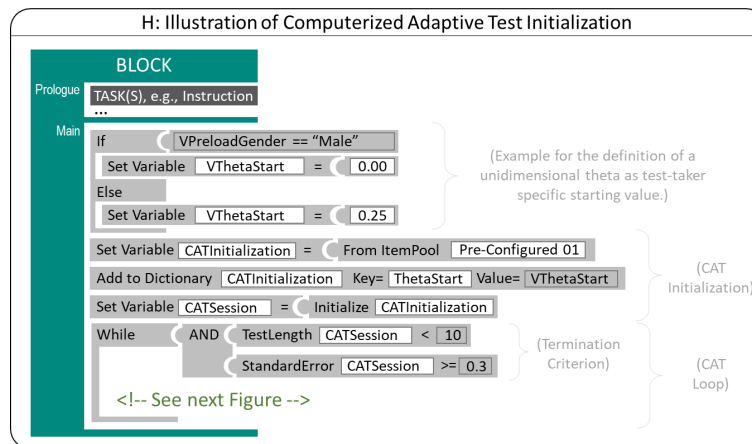


FIGURE 2.20: Illustration of *Computerized Adaptive Test Initialization*

Items are selected from an *Item Pool* (i.e., a list of *Tasks* with known *Item Parameters*, see section 2.5) and item selection algorithms can incorporate additional constraints (see section 2.7). For constraints management (see, for instance, [Born and](#)

¹³The general structure of information used to initialize CAT algorithms is described, for instance in the [QTI CAT Specification](#) using key-value pairs.

Frey, 2017), additional parameter stored in an *Item Pool* can be required, for instance, for *Exposure Control* (e.g., Simpson-Hetter-Parameter, Hetter and Simpson, 1997) and for *Content Balancing* (e.g., answer keys, see Linden and Diao, 2011).

Item selection either results in one single item (*Item-by-Item* adaptive testing) or a list of items (*Item Stages*), similar to *On-the-fly Multi-Stage Testing*. As described above, a list of items (with at least one entry) can be used to store the selected items used for test administration (see "SelectedItems" in Figure 2.21).

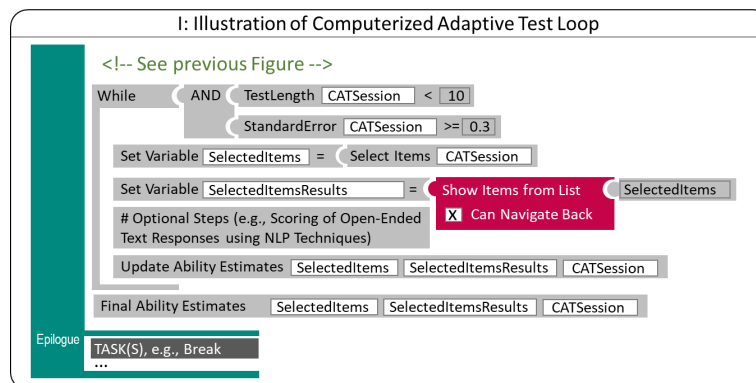


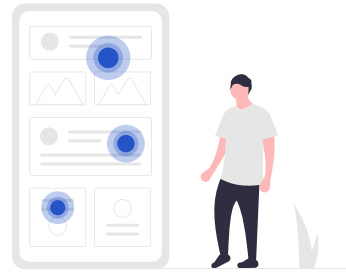
FIGURE 2.21: Illustration of *Computerized Adaptive Testing Loop*

Navigation between *Tasks* of an *Item Stage* can be allowed since item selection only takes place after administering all *Tasks*. Scoring of all or selected administered items (see section 2.5.1) is required for the subsequent update of the ability estimation. Scoring can take place inside of the item or based on the list of result data (see "SelectedItemsResults" in Figure 2.21).



How to do this with the CBA ItemBuilder? Adaptive testing requires a test deployment software that either provides the required *IRT* functions for item selection and ability estimation (see, for instance, section 7.5) or the integration of statistical software such as R (e.g., Scalise and Allen, 2015, see also section 7.3).

2.8 Log and Process Data



Computer-based assessments provide the opportunity to collect not only the final work product (i.e., raw responses and scored responses, see section 2.5.1) but also to allow the collection of so-called *log data* that origin from students' interactions with the computer-based assessment platform (e.g., clicked buttons, selected radio buttons or checkboxes, entered text, mouse-moves, etc.) or internal system changes (e.g., timers). The examination of these log data from cognitive ability testing has gained increased attention (Goldhammer and Zehner, 2017), for instance, in educational research, since the computer-based assessment was introduced in large-scale assessments (e.g., Greiff et al., 2016).

2.8.1 Basic Terminology

In the context of computer-based assessment, using log data is still a relatively new field of research. Different terms like stream data, log file data, event data, process data, and others are used (and sometimes mixed). In order to illustrate the meaning and use of log data for the theory-based construction of process indicators and to provide guidance concerning potential implementations in the CBA ItemBuilder, a conceptual clarification follows first.

Paradata: Additional information about the test-taking process and the data collection can be understood as part of the so-called *paradata*, commonly used in social science research (e.g., Kreuter, 2013). Kroehne and Goldhammer (2018) summarizes categories of access-related, response-related and process-related *paradata*.

- *Response-related paradata* include all user interactions (pointing device like mouse click or touch events, keyboard entries of hardware keyboards or soft keyboards) together with all state changes of components that have different states (components like checkboxes, radiobuttons etc. that can be selected and deselected) and internal states (like timers etc.)

- *Process-related paradata* cover, for instance, information related to the navigation within assessment instruments (see section 2.4) as well as interactions not directly related to item responses
- Additional *assess-related paradata* can occur when administering computer-based assessments. For example, this data can inform when an assessment is started, continued, or completed with what type of device.

Describing possible paradata with a taxonomy cannot cover all possible future applications of the additional information available in technology-based assessments. For a deeper look, it is worth considering the underlying nature and origin of this additional information, the emergence of which can be conceptualized in terms of events that create what is called *Log Data*.

Log Events: *Log Data* are generated and stored by assessment platforms as events, that inform about how a platform providing assessment material was used and how a provided platform changed (see section 1.6). For the data to be events, we can assume without further limitation that each event contains the following information:

- *Time stamp:* When did something take place?
- *Event name:* What has taken place?

The *Time Stamp* can represent an absolute date and time, or it can represent a relative time difference. The *Event Name* (or *Event type*) is only a first nominal distinction of different events. As described in section 1.6, log events in the context of *assessments* can be expected to contain the following additional information:

- *Person identifier:* Which test-taker can it be assigned to?
- *Instrument identifier:* Which part of an assessment can it be assigned to?

The assignment to a person is made by a reference (e.g., in the form of an identifier), and this personal reference must be taken into account in the context of using log data as research data (e.g., in the form of an ID exchange, see section 8.6). The reference to a part of the instrument can be established, for example, by an item identifier or a unit identifier or by describing the level at which a log event occurred (e.g., test-level).

- *Event-specific attributes:* What additional information describes what happened?

The *Event Name* describes various possible log events distinguished by an assessment platform. Each *Event* can provide specific further information, which in addition to the *Event Name* form the actual content of the log data. Depending on the event type, the event-specific attributes can be optional or required, and attributes can have a data type (e.g., String, Boolean, or some numeric type). If the information provided

by the assessment platform with an event-specific attribute is not in atomic format (i.e., if it is not a single piece of information but a data structure, see Kroehne, 2010, for details), storing log data in rectangular data set files becomes more challenging (see section 2.8.4).

- **Raw Log Events:** From a technical perspective, events in digital environments like web browsers are required and used for programming and implementing digital (interactive) content, such as assessment instruments. Accordingly, a basic layer tries to connect at a low level to make those events available and usable for diagnostic purposes. The resulting log events not specific to any concrete task or assessment content are called *Raw Log Events*. *Raw Log Events* have event types that relate the captured information to the event's origin (e.g., button click, mouse move, etc.). Raw log events are not necessarily schematically identical to the events of the used technological environment in which the (interactive) assessment content is implemented (such as, for instance, HTML5/JavaScript for browser-based content). However, raw log events are platform specific (i.e., different software implementations of identical content can provide different raw log events). Hence, the assessment software defines which raw log events are captured (and how).



How to do this with the CBA ItemBuilder? Assessment components created with the CBA ItemBuilder automatically provide *Raw Log Events* (see appendix B.7 for a description of all log events). For interpreting the log events it is crucial to define *User Defined Ids* (i.e., identifiers for interactive components, that are used as references for the events data provided as trace logs to the components used in the *Page Editor* to design the assessment content, see section 3.7.4).

- **Contextualized Log Event:** Based on the assessment content, a second kind of log event can be described: Events that inform about an event concerning a particular action or change in a concrete task or a particular item. These events can be called *Contextualized Log Events*, and instrument developers (i.e., item authors) need to define which particular action or internal change has which particular meaning. The event name (or event type) can encode the semantics of contextualized log events, and contextualized log events fit (as raw log events) into the concept of log events as described above.



How to do this with the CBA ItemBuilder? The definition of specific *Contextualized Log Events* as part of the implementation of assessment materials with the CBA ItemBuilder is possible (see *Operators to Create Trace Messages* described in section 4.4.6) and recommended if the derivation of the events based on the *Raw Log Events* is either laborious or if theoretically defined *Contextualized Log Events* are already

defined as part of the instrument construction. HTML5 / JavaScript assessment content that is included in CBA ItemBuilder projects (see section 3.14 for a description of `ExternalPageFrames`) can provide custom log-entries (*Raw Log Events* or *Contextualized Log Events*) via the API described in section 4.6.3.

Feature Extraction: Tagging or labeling selected *Raw Log Events* as *Contextualized Log Events* can be understood as an example of *Feature Extraction* (i.e., the derivation of *Low-Level Features* using the raw log events, see Kroehne and Goldhammer, 2018). In this context, *Contextualized Log Events* are *Actions* (i.e., *Low-Level Features* that occur at a point in time but do not have a time duration). More generally, *Actions* are contextualized information that can be extracted from the log data. So-called *States* (i.e., *Low-Level Features* that have a time duration) supplement the possible features that can be extracted from log events. As a rule, log events indicate the beginning and end of a *States*, while *Actions* represent specific *Log Events* that occur within *States*.



How to do this with the CBA ItemBuilder? The R package *LogFSM* described in section 2.8.5 can be used to analyze log data provided by the different deployment software tools (see chapter 7).

Process Indicators: Information about emotional, motivational, and cognitive processes during assessment processing may be contained in log data. Their interpretation in the context of assessments is guided by psychometric concepts such as validity (e.g., Goldhammer et al., 2021) and scientific principles such as reproducibility, replicability, and (independent) replication of empirical research.

Raw log events are platform-specific and are not suitable for defining indicators since if an assessment is re-implemented in a different technical platform, it cannot be assumed that the *Raw Log Events* will arise identically. Accordingly, the definition of process indicators that can become the subject of empirical validation is based on low-level features (*Actions* and *States*), where *Actions* also include *Contextualized Log Events*.

In this context, *Process Indicators* are aggregates of *Low-Level Features* (e.g., the number of occurrences of a particular *Action* or the aggregated time in a particular *State*), meaning values of person-level variables that can be derived from low-level features, and for whose interpretation theoretical arguments (e.g., in the sense of evidence identification) and empirical support can be collected. Psychometric models (e.g., measurement models) can be used, for instance, to investigate the within-person relationship of process indicators across different items or tasks and their relationship to outcome measures.



How to do this with the CBA ItemBuilder? Low-level features extracted with LogFSM can be used to compute *Process Indicators* in R.

2.8.2 Item Designs and Interpretation of Log Data

In line with the terminology described in the previous section, [Kroehne and Goldhammer \(2018\)](#) describe a framework for analyzing log data. The core of this framework is the decomposition of the task processing into sections (called *States*), which can be theoretically described regarding an assessment framework.

The presentation of an assessment component (i.e., an item or unit, for example) always begins in a designated start state. *Raw Log Events* collected by an assessment platform can be used to mark the transition from one state to another state. As described above, *Raw Log Events* can also indicate specific *Contextualized Log Events* (i.e., *Actions* with a task-related interpretation). This way, two identical *Raw Log Events* can be interpreted differently depending on the current state (called *Contextual Dependency of Log Events*, see [Kroehne and Goldhammer, res](#)).

Decomposition of Test-Taking Processes: The theoretical framework can also be used to describe item designs with respect to the interpretability of log data. For this purpose, it is helpful first to consider what creates *States*. According to [Kroehne and Goldhammer \(2018\)](#) the meaning of *States* is constituted by combining the displayed information (i.e., what is presented to test-takers on screen) with the possibilities to interact (i.e., what can test-taker do and how can test-taker interact with the content).

Suppose the presented information changes (i.e., a page change or a modification of the visible area of a scrollable page) or the opportunities change how the test-taker can interact with the assessment content. In that case, it may be helpful to describe the test-taking process using two different *States*. A log event (or events) can mark the transition between the old and new state (e.g., a page-change-event or scroll-event). If the interpretation of the two states differs meaningfully, then the interpretation of the involved log event(s) follows from the difference between the two states.

Log events can, for example, represent the selection of a response from pre-defined response options (i.e., events that can be categorized as *answer-change-event*). Suppose a *State* contains the view of an item stem, question, and the possibility to respond, for instance, by selecting a radio button. In that case, these *answer-change-events* can distinguish two states, BEFORE_RESPONDING and AFTER_FIRST_RESPONSE. While the assessment is ongoing, meaning while the item is on screen and the test-taker still has the opportunity to change the response, the state AFTER_FIRST_RESPONSE cannot be further decomposed into WHILE_RESPONDING (the time between the first and the last answer-change event) and AFTER_RESPONDING, as it is not yet decided whether the test-taker will select an answer only once (meaning, only one answer-change-event), or the

test-taker will change to a different answer by clicking on different radio button. However, the situation will be different if log data from concluded assessments are analyzed. Either way, the interpretation of the state BEFORE_RESPONDING rests on the premise that the item design allows assigning this time component to one question. This is only possible with assumptions when multiple questions are presented in parallel on one screen.

Process Indicators for Item Analysis: Based on the decomposition of test-taking processes into individual sections (i.e., *States*), which are subdivided by log events, an in-depth item analysis can be performed, for example. An example of using frequency-based aggregates of low-level features (e.g., number of *Actions* meaning events tagged as answer-change events) and low-level features within states (e.g., number of visits of an item after selecting a final response), as well as time-based aggregates (e.g., total time after selecting the final response), can be found in [Lahza et al. \(2022\)](#).

States with Dedicated Meaning: Theory-driven created, interpretable process indicators are also possible if dedicated *States* can be crafted with a specific interpretation regarding the measured construct. An example of this idea can be found at [Hahnel et al. \(2019\)](#), based on making additional information necessary for solving a task of the task solution visible only after a test-taker interaction. The additional information about the source of a document is placed on a dialog page, and buttons that create log events need to be clicked to open and close the dialog page.

A similar concept underlies the analysis of navigation behavior in hypertext reading when relevant pages (i.e., *States* reconstructed based on navigation behavior, on which information essential for the task solution is presented) are distinguished from irrelevant page visits (see e.g. [Naumann, 2015](#)).

2.8.3 Completeness of Log-Data

Which *Raw Log Events* are provided by an assessment platform depends on the respective programming, and *Contextualized Log Events* are each related to concrete item content. Hence, both forms are not suitable to describe whether the programming of a computer-based assessment provides all (relevant) log events. [Kroehne and Goldhammer \(2018\)](#), therefore, describe different completeness conditions.



How to do this with the CBA ItemBuilder? The log data collected with the CBA ItemBuilder can be inspected live during a *Preview* of the assessment using the *Trace Debug Window* (see section 1.6).

Log Data versus Result Data: The starting point for differentiating different completeness conditions of log data is the review of the relation between log data and result data of a (computer-based) assessment. The result data contain for each item

the raw responses (for instance, the text entered into text fields and the selection of choice elements such as radio buttons and checkboxes), and if implemented within the assessment platform, the scored item responses (see section 2.5.1). Although result data can be missing coded (see section 2.5.2) already when provided by the assessment software, we ignore missing value coding for the following explanation.

Response completeness: Suppose a result variable that contains the final selection of, for instance, a group of radio buttons (e.g., A, B and C). The value for this variable is an identifier for the finally selected radio button or a transformation of this identifier to a numeric value using a simple mapping (e.g., 1=A, 2=B, and 3=C). Log data of the intuitive type answer-change are generated each time the selection of the radio button group is changed. If A is selected first, followed by a selection of C, two answer-change log events are expected, one for the first selection (A) and one for the second selection (C). Taking both log events together in the correct order allows us to reconstruct the final selection and, thus, the value of the result variable (C or 3). Hence, if all answer-change events are collected, the result data can be re-constructed from the log data. If this is possible, all answer-change events are logged (whatever technique is used to collect the responses), and log data are called *response complete*. To achieve this property, answer-change events need to be ordered by timestamp. However, we do not need real time measures, it is sufficient that the order of log events is maintained by the logging.



How to do this with the CBA ItemBuilder? Although the collection of log data is well developed for the CBA ItemBuilder, scoring defined within the CBA ItemBuilder tasks is only evaluated when test-taker end the task (using a runtime command, see section 3.12). Accordingly, only the raw input can be reconstructed from log events provided by the CBA ItemBuilder runtime in the current version and result data and log data are stored in parallel. Note, however, that using the TaskPlayer API (see section 7.7) deployment software that uses the CBA ItemBuilder runtime can request scoring data at any time (and multiple times).

Progress-completeness: If the answer-change events can not only be sorted, but all answer changes are logged immediately with sufficient accuracy, then log data can also be *Progress Complete*. To check this property, it must be ensured that the result variables can be determined from the log events at any time (and not only after an assessment component, i.e., an item or a unit, has ended). This property can be easily illustrated through text input. If the changes in a text input field are only logged when the input focus changes, then *Progress Completeness* is not satisfied because the values of the result variables can be reconstructed from the answer-change events only at the times when the test-taker leaves the input field. To achieve *Progress Completeness* all text changes (e.g., in connection with the key-down or key-up-events) would have to be logged.

State-completeness: The completeness conditions described so far are agnostic regarding the planned use of log data. This is different for the condition described as

State Completeness. Consider a use case in which we want to replicate findings from a specific study that used a particular set of *States* (or specific *Actions* or *Contextualized Log Events*). To verify that this replication will be possible using the assessment software under consideration, *State Completeness* needs to be checked regarding this differentiation. For that purpose, all transitions between distinguished *States* need to be identified with available *Raw Log Events*. Note that the *Raw Log Events* used for the re-implementation can be different from the original implementation as long as all required transitions (as well as *Actions* and *Contextualized Log Events*) can be recognized from the log data with specific *Raw Log Events* that are provided by the new platform.

Replay-completeness: Verifying log data with respect to *State Completeness* is especially helpful if a concrete set of *States* and *Actions* (or *Contextualized Log Events*) is known. If one wants to ensure that log data is as complete as possible so that all changes based on user interactions and internal state changes, such as timers, etc., are included in the log data, then *Replay Completeness* is helpful. *Replay Completeness* is fulfilled when a screencast (like a video)¹⁴ can be recreated from the collected log data. Figure 2.22 provides an example.

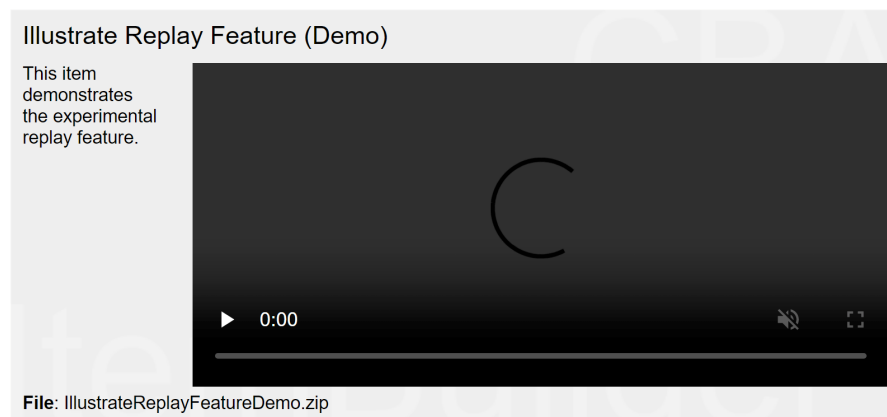


FIGURE 2.22: Item illustrating the replay feature (work in progress) ([html](#)|[ib](#)).



How to do this with the CBA ItemBuilder? The development of the CBA ItemBuilder starting with Version 9.8 aims to achieve *Replay Completeness*. Note, that in the current preview the pointing device (i.e., the mouse pointer) is not included and, hence, invisible for the replay.

An understanding and objective analysis of the completeness of log data (i.e., what

¹⁴Note that develops the the *Text Replay* suggested by [Gobert et al. \(2013\)](#) further.

interactions and system state changes can be inferred from the log event data) is also crucial for making valid statements about *Idle Times* and for interpreting time differences in log data analyzed as time-stamped action sequences (e.g., [Ulitzsch et al., 2022](#)).

2.8.4 Data Formats for Log Data

Log data collection often requires specific programming and presents additional requirements that must be specified, implemented, and tested (see, for instance, section 8.4 for details about *Testing* of CBA projects).

Even though based on the definition of log events described above (see section 2.8.1), the structure of the event data can be derived, the data of software developers in concrete assessment software tools do not necessarily have to be stored in log files in a structured way. Often log data is stored mixed with other data (including paradata and metadata), and functional requirements (regarding presenting assessment content and collecting final responses) might be prioritized for the assessment software in comparison to a transparent separation of different data forms.

For various reasons, data from an assessment platform may initially be stored in a preliminary (and proprietary) data format, and additional steps of data post-processing (see section 8.6 may be necessary to extract the *Raw Log Event* data (or *Contextualized Log Events*). The data formats for log event data described briefly below must therefore be generated from the data from the preliminary data formats used by particular assessment software. It does not matter whether the assessment software stores the data internally in a relational or document-based database or whether it is based on the generic markup language XML or the JSON serialization commonly used in web contexts.

Flat and Sparse Log Data Table: Starting from a long-format with one event per line, the storage of log data in the form of a one big *Flat and Sparse Log Data Table* (FSLDT) is possible ([Kroehne, tion](#)). As long as the minimal conditions described above (see section 2.8.1) are fulfilled (i.e., each log event is assigned to a person, has an event type or name, and a timestamp), the corresponding columns in the FSLDT are filled in for each line. Suppose many different event types that contain different required or optional event-specific data. In that case, the FSLDT contains many missing values and can become large and messy. Moreover, additional specifications are required for non-atomic event-specific data (i.e., to handle nesting, see [Kroehne, tion](#)).

Universal Log Format: Log data can be stored clearly and efficiently using simple relational database concepts. For this purpose, the data is stored in tables per event type. Each of these data tables thus contains fewer missing values, and the semantics and data types of the event-specific attributes (i.e., columns) can be defined and documented (see section 8.7). Missing values only occur for optional event-specific attributes, and additional specifications can be used to handle nested data types in the form of additional tables.

The individual tables per event type can be combined and sorted again based on the time stamps to create an FSLDT. The individual tables can be saved as data sets in the common data set formats (CSV, Stata, SPSS, ...) and thus easily managed by research data centers since standard procedures (e.g., for the exchange of identifiers, see section 8.6) can also be applied.

eXtensible Event Stream (XES): Developed to achieve interoperability for archiving event logs data, the [IEEE 1849-2016 XES Standard](#) is the most attractive format for storing log data in a way that different tools can read. As described in [Kroehne \(2016\)](#), the XES format combines information about the log data (i.e., how the data are stored) and the data, making this standard very flexible and valuable for log data from computer-based assessments. However, although the data are stored in XML format, researchers unfamiliar with the XES standard cannot read or verify the data without additional tools.



How to do this with the CBA ItemBuilder? When post-processing data collected with CBA ItemBuilder content with the R package *LogFSM* (see section 2.8.5), log data are processed and provided as (compressed) XES file and in the *Universal Log Format* (in either Stata, SPSS or CSV tables). When reading and merging all event-specific tables from a ZIP archive containing data in the *Universal Log Format*, a *Flat and Sparse Log Data Table* can be created in R.

Learning Analytics: Log data gathered in assessment can also be described and stored using the concepts developed in the domain of learning analytics (for instance, the *experience API* statements, xAPI). The test-taker, required for log events after data preparation as *person identifier* corresponds to the *actor* in an xAPI statement. The event name (if necessary in combination with one or multiple event-specific attributes) can create a *verb* (e.g., *clicked*). The *object* is specified by the instrument identifier, and when necessary, further specified by event-specific attributes. Finally, *context* information of an xAPI statement can refer to access-related para-data (e.g., the location where an assessment takes place) or to *metadata* or linked data (such as the instructor of a course, in which an assessment is conducted).

Note that other standards (such as [IMS Caliper](#) and [Hao et al., 2015](#)) exist that might be worse to consider.

2.8.5 Software Tools

The development of generic tools for analyzing log data from computer-based assessments is still in its infancy. Often, log data are only analyzed in a study-specific way, for example, by creating specific programs for analysis (cf. PIAAC Log Analyzer, [Goldhammer et al., 2020](#)).

LogFSM: A generic tool for analyzing log data based on algorithmic processing of

log events (*Raw Log Events* or *Contextualized Log Events*) using finite-state machines is the R package *LogFSM*, which implements the framework for feature extraction suggested by Kroehne and Goldhammer (2018). *Finite-State Machines* (FSM, e.g., Alagar and Periyasamy, 2011) are used, for instance, in the CBA ItemBuilder to implement dynamic interactive items (see section 4.4, and Rölke, 2012; Neubert et al., 2015). Similar principles are also useful for the analysis of log file data (e.g., Kroehne and Goldhammer, 2018).

Further R Packages: Additional R packages for analyzing log data include, for instance, *LOGAN* (Reis Costa and Leoncio, 2019) / *LOGANTree*, *ProcData* (Tang et al., 2021), and *TraMineR* (Gabadinho et al., 2011).

2.9 Feedback

Computerized assessment allows for a variety of different forms of feedback. Feedback can relate to the answering process and the answers themselves.

2.9.1 Feedback during the Assessment

Different feedback forms can also be distinguished, either always displayed during the assessment or retrieved optionally. Some of the basic options are briefly described in this section.

Real-Time Feedback while Responding: Interactive items can be designed so that the cognitive operations for completing a task change as part of the response format. This type of feedback during test-taking is illustrated in the item in Figure 2.23 for an example matrices task. In this example, once the answer is chosen, it is already presented in the context of the stimulus, supporting verifying the submitted answer without explicitly providing feedback about task correctness.

Real-Time Feedback about Results: Closed response formats or automatically scored items allow explicit feedback about the results provided either immediately or delayed (Shute, 2008). As discussed in van der Kleij et al. (2012), the operationalization of immediately (e.g., feedback given during the completion of an item) and delayed (e.g., feedback given directly after completion of all the items in the assessment) differs across research. Different types of feedback are investigated and used in practice and real-time performance feedback is considered (e.g., scaffolding feedback, Finn and Metcalfe, 2010), in particular, attractive for formative assessments (DiCerbo et al., 2020).

Feedback about (Remaining) Time: Especially when moderate time limits are used for larger test sections, it is helpful to integrate feedback about the remaining time

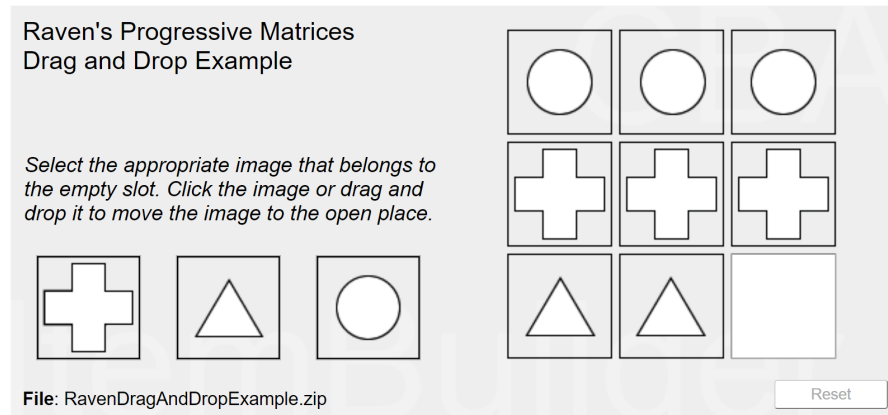


FIGURE 2.23: Item in the style of a Raven's Progressive Matrices test ([html](#)|[ib](#)).

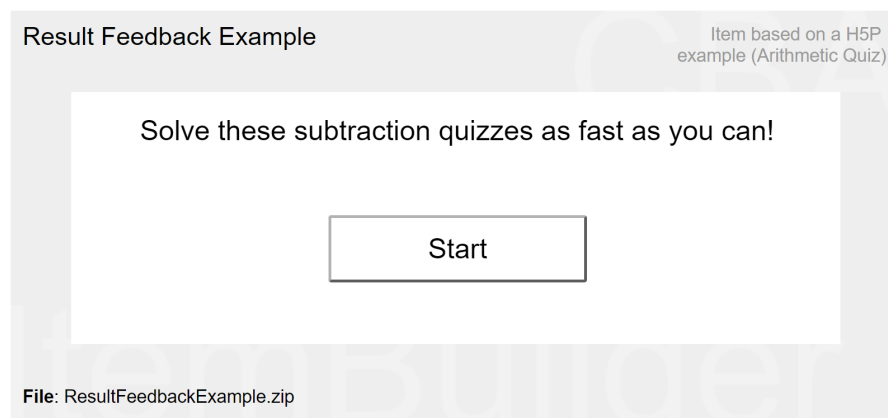


FIGURE 2.24: Item illustrating result feedback ([html](#)|[ib](#)).

into the items to make the information available to all test takers in a comparable (standardized) form.

The feedback can be implemented by displaying the remaining time (numerically) or by using a graphical visualization that visualizes the remaining time only roughly (in order not to create unnecessary time pressure).

Alternatively, as illustrated in the item in Figure 6.37, feedback on the remaining time can be provided, for example, at the item level by displaying a hint if an item still needs to be processed after a predefined time threshold.

Feedback about Task Progress: In addition to the remaining time, it is also helpful to provide feedback on overall test completion in computer-based assessments where progress cannot be inferred from the printed test booklet, as shown in Figure 2.25.

Task Progress / Completion Example

Feedback about Task Progress

1: Visited 2: Not Visited 3: Not Visited 4: Not Visited

☒ Task Progress
☐ Task Completion

Item 01

☐ Option A
☐ Option B
☐ Option C
☐ Option D

Next

File: TaskProgressExample.zip

Reset

FIGURE 2.25: Item illustrating task progress feedback (<http://html5lib.org>).

Feedback on Task Completion: The feedback about the processing status can also distinguish between visited pages and answered items. In that case, it is also possible to read how many tasks will still have to be processed from a progress display. A simple example is included in Figure 2.25. The graphical design can be more elaborate if multiple items are combined on individual pages.

Feedback about Navigation: As shown already in Figure 2.13, as soon as test-takers can not navigate back after leaving a section, unit, or page, a feedback dialogue or popup message is often used to inform the test-taker about the consequence of continuing.

Missing Value Feedback: The number of unfinished tasks can be displayed continuously or when leaving a unit. Suppose test takers cannot navigate back to a previous section. In that case, it makes sense to display a warning that can be designed differently for the case that all items have been answered (feedback about navigation) or that items still need to be answered (missing value feedback).



How to do this with the CBA ItemBuilder? The CBA ItemBuilders capabilities for the dynamic content can be used within tasks to realize the different feedback possibilities during the assessments (see chapter 4).

Rapid Guessing Feedback: Feedback about test-taking behavior, such as rapid guessing, can be provided to test-takers, either automatically [see, e.g., N.N.] or by test administrators (see, e.g., Wise et al., 2019).

Feedback on Consistency: Feedback about responding (too) quickly or unexpectedly fast is just one of many possibilities. If the software allows, for instance, person-fit measures can also be used to give feedback about inconsistent answers.

2.9.2 Feedback after the Assessment

Once data collection is completed for a single test-taker, there are multiple opportunities for further use of data for feedback purposes. Steps that can be relevant for creating feedback after an assessment include the scoring of responses (see section 2.5.1) and the ability estimation (using calibrated items, see sections 2.5.4 and 2.5.5).

Technical Platform for IRT-Methods: When an IRT model is used to model the measured construct, appropriate psychometric algorithms must be used for ability estimation, using pre-calibrated item parameters (i.e., the number of correct responses is not enough if the raw score is not a sufficient statistic). The necessary functions for ability estimation with various IRT models are freely available within software packages for data analysis (for instance, in the form of the R package TAM, [Robitzsch et al., 2022](#)). In order to use these functions also for operational computer-based assessments, the corresponding platform must be available (e.g., by using R with the help of the environments [ShinyProxy](#), [OpenCPU](#) or the R package [Plumbr](#)), or the delivery environment must provide the required IRT functions.

Technical Platform for Report Generation: A technical solution for the automatic creation of feedback is also necessary to create texts and, if necessary, graphics and combine them into HTML, PDF, or other text documents [e.g., R and the package [kntir](#), [Xie \(2015\)](#); see section 7.3.5 for an example].

Technical Inclusion of Process Indicators: In addition to outcome variables, process indicators representing aggregations of *Low-Level Features* that can be extracted from log data using algorithms (see section 2.8) can be helpful in generating feedback. A technical platform is also required for this necessary analysis of the log data, in which, for example, the R package LogFSM (see section 2.8.5) can be integrated.



How to do this with the CBA ItemBuilder? The derivation of result variables for feedback after an assessment can be prepared by using the scoring functions of the CBA ItemBuilder (see chapter 5). However, parts of the required technical platform must be provided by the deployment software (see chapter 7).

2.10 Item and Test Security



Computer-based testing is also connected in different ways with the concept of item and test security.

2.10.1 Protection of Items and Tests

Test security is first of all related to the protection of item content, of particular importance for high-stake assessments or for low-stakes assessments that aim to measure trends by using so-called link items. The core idea is that test instruments should only be accessible to a limited group of users to protect item contents from becoming known.

2.10.2 Secure Test Deployment

Whether and in what way item contents can be made known depends mainly on the type of test delivery. If, for example, a proctor is present on-site during a test session and the test is carried out in a so-called kiosk mode (see section 7.2.3), then the proctor and the software can help to manage the possibility of uncontrolled dissemination of item content.

From a diagnostic perspective, test security also has a second meaning: An assessment platform should implement a certain degree of restrictions so that the assessment can be carried out with as little disruption as possible. Since *a certain degree* is difficult to verify from an IT perspective, we work with the following operational definition of *operational test security* of assessments in terms of usability for diagnostic purposes:



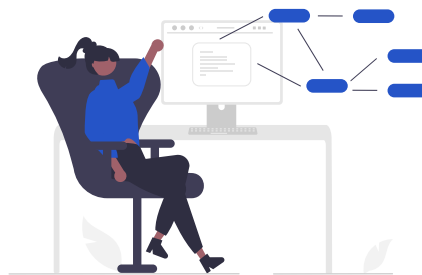
No *unintentional* mishandling of the assessment platform or any unintended test-taker behavior should lead to an interruption of the assessment.

The usability of computer-based assessments and the protection of items and tests are related, as both aim at restricting the possibilities of how the test-takers can interact with the test delivery while answering items and working on assigned tasks. The following examples illustrate for web-based deployments how a computer-based assessment can be challenged regarding the usability (and related to test security):

- Accidental closing of the entire browser can interrupt the assessment or result in losing data.
- Navigation using browser *Back*-, and *Forward*- buttons (instead of the buttons within the assessment content) might interrupt the assessment.
- Drag and drop operations on content that requires scrolling to be completely visible can occur unintendedly, for instance, if test-taker can reduce the size of a browser window.
- The attempt to open an assessment in a second window or tab can lead to unwanted interruptions or, in the worst case, inconsistencies in the data.

Thus, the question of the delivery of computer-based assessments can have a significant effect on the validity of measurements. Collecting log data, for instance, for so-called off-task behavior, can help identify and ultimately quantify the problem. However, interpreting scores at the individual level may be challenging or, analogous to disengaged response behavior and rapid guessing responses, impossible.

2.11 Design Principles of the CBA ItemBuilder



2.11.1 Content Experts as Item Developers

Creating good assessment items depends primarily on creating task contexts in which understanding can be applied and, where appropriate, knowledge can be

demonstrated. As general as the concepts of computer-based testing and psychometric assessments are, as specific are the application areas and domains in which diagnostic questions are to be answered. The CBA ItemBuilder is therefore designed in such a way that content experts can use the software to create, test, and ultimately use items for computer-based assessments. The technical platform should support a wide range of possibilities of more innovative, technology-enhanced item formats, which is why the CBA ItemBuilder goes beyond the simple, although standardized item formats of IMS QTI and allows the design of multi-page interactive items.

2.11.2 Model-based Representation

The CBA ItemBuilder uses a *component model* to enable the graphical design of assessment contents. Starting from individual pages, the position of elements and their properties are defined with the help of a graphical editor. In the current version of the CBA ItemBuilder, this graphical editor is a program that can be installed on the computer (see section 1.1). With this program the abstract *component model* becomes the basis for the creation of assessment content. The visual part of the *component model* comprises *pages* (of different page type, see section 3.4), and the graphical *Page Editor* allows to add components from the so-called *Palette* to the pages. The components have various properties, such as the position (in pixels x and y), the size (in pixel `width` and `height`) and many more (depending on the type of a particular component). The component model is stored within the item definition (in so-called *Project Files*) and used to generate source code (or a configuration file that can be used with a particular piece of software, called *TaskPlayer API*) that is used to use CBA ItemBuilder content in assessments.

2.11.3 Separation of Layout and Logic

The model-based representation of the assessment contents (using the *component model* as describe above), as implemented in the CBA ItemBuilder, is also the basis for a possible long-term archiving strategy of computer-based tests (see section 8.7.3). The basic idea here is that the runtime code can be updated from the model if a newer version of the CBA ItemBuilder can open *project files* of previous versions (see 3.2.1).

2.11.4 Containers and Nesting

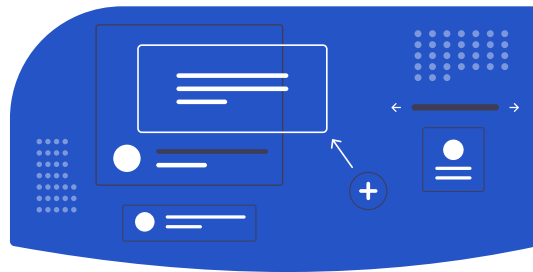
Containers define which components can be added as child elements (i.e., only components of a particular type can be nested within containers of a particular type). Moreover, using the concept of containers, the order in which components are displayed (the so-called *Z-Order*, the sequence in which components are rendered on top of each other if they overlap) is defined for pages created with the CBA Item-

Builder. Containers are always rendered below their child elements. If a component is a container, the *Z-Order* of its nested components is defined by the order of the components *within* the container (see section 6.8.5 for details on overlapping components). Hence, the order in which components are added (i.e., the order within the *Component Edit* view) defines the order in which components are displayed on top of each other during *runtime* (i.e., in the *Preview* and when assessment components created with the CBA ItemBuilder are used in assessments).



3

Designing Items Using Static Content

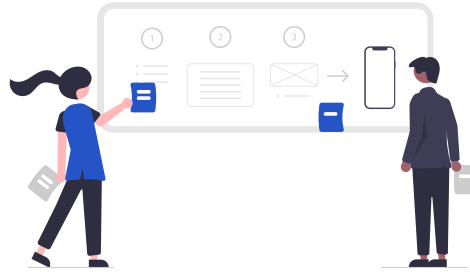


The following two chapters describe creating new items with the CBA ItemBuilder. Static content (chapter 3) and dynamic content (chapter 4) are introduced separately, which helps describe the CBA ItemBuilder's functional scope. This separation will become less critical for practical work with the authoring tool since the two areas are deeply integrated.

In this chapter, the features and topics are introduced in sections 3.1-3.7 in the order that follows the steps required for item implementation, and cross-references acknowledge the relationships between the various topics. The introduction is followed by a second part in sections 3.8-3.15 that systematically introduces all components authors can use to design assessment content.¹

¹Note that all components can be easily found by using the register of all components included in Table B.22 in the appendix (see appendix B.6).

3.1 Overview of the User Interface



The CBA ItemBuilder is built based on an open-source *development environment* ([Eclipse](#)). Hence, working with the CBA ItemBuilder differs from using modern apps and web-based tools. The user interface is first described in detail for all users to find their way around quickly.



Tip: Using the CBA ItemBuilder requires the *context menu*, opened via the right mouse button.

3.1.1 Top: Main Menu and Toolbar

After starting the CBA ItemBuilder, a program window opens with a structure as shown in Figure 3.1. The arrangement of the different areas can be adjusted and configured according to individual needs and preferences (see section 6.8.1 for details). The different areas of the user interface have particular meanings, described in the following, as shown in the default configuration.

The CBA ItemBuilder can be operated via a *Main Menu*, located in the top of the application (see File, Edit, ... Help in Figure 3.1). Essential commands are also directly accessible with icons in the *Toolbar*, below the *Main Menu*.²



Tip: The available entries in the *Main Menu* of the CBA ItemBuilder are context-sensitive (e.g., selected entries are only available if a tab with a certain content is displayed in the *Page Editor*, see section 3.1.3).

²A complete list of all functions can be found in the appendix in Tables B.1-B.7.

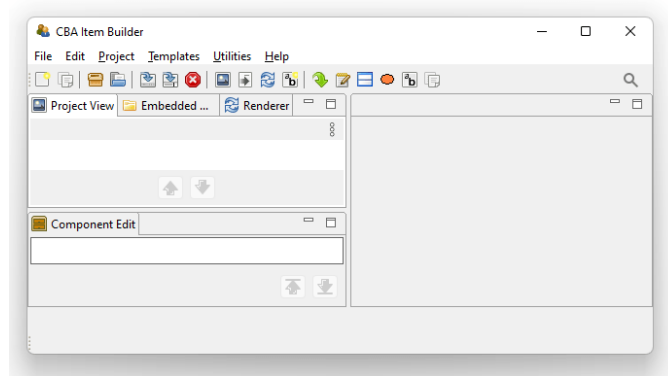


FIGURE 3.1: Main Menu, Toolbar and left area of the CBA ItemBuilder user interface (Project View, Embedded HTML Explorer, Renderer and below the Component Edit).

3.1.2 Left: Project View, Component Edit, Embedded HTML Explorer and Design Pages with Basic Components

In the default configuration, the left area of the CBA ItemBuilder contains the following four tabs (see Figure 3.1) with specific functions as described in the following: The *Project View*, the *Component Edit*, the *Embedded HTML Explorer*, and the internal *Rendering*. It is possible to switch between the tabs at any time, and the tabs can also be un-docked and re-sized (see section 6.8.1).

Project View: The *Project View* is organized as a tree and is empty by default. As soon as a project is created (see section 3.2.1, the *Project View* shows the current project's name as the root element, which is also shown in the name of the main window. When a project contains pages, the pages are listed under the root element. Right-click on the root element (i.e., the project name) or a page (i.e., one of the elements in the tree) opens different context menus, as shown in Figure 3.2.

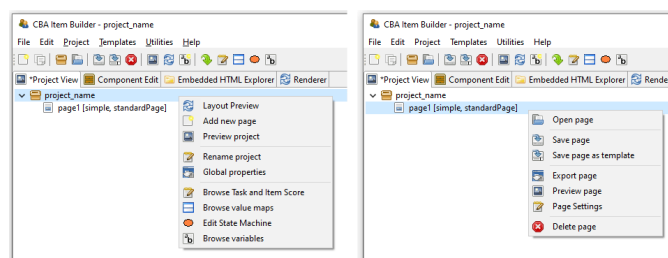


FIGURE 3.2: Context Menu in the Project View clicked on the root (left) and on a page (right)

Entries in the tree of the *Project View* represent pages in the current project. The order of pages can be sorted with the buttons  and .³

The context menu of the *Project View* clicked on the root (i.e., right-click on the project name) allows to create new pages (see *Add new page* in the left part of Figure 3.2). Double-click a page in the *Project View* (or right-click a page name and select *Open Page*) allows to open a page in the *Page Editor*. The remaining functions accessible using the context-menu of the *Project Name* (right-click the root element in the *Project View*) and the *Pages* (right-click a page name in the *Project View*) will be described in context of the related features of the CBA ItemBuilder.



The *Project View* provides **two different context menus**, namely a menu with functions to the current project (available when the project name, i.e., the root element is right-clicked) and a menu with functions to a selected page (available when a page in the *Project View* is right-clicked).

Component Edit: Beyond the *Project View* that is open by default, the left area of the CBA ItemBuilder's user interface contains three more tabs. The first tab is headed with *Component Edit*. After opening a page in the *Page Editor* the *Component Edit* lists all components of that page. The *Component Edit* view is essential for selecting components in the graphical *Page Editor*, for instance, when components are placed on each other or are too small to be easily selected by point-and-click.



The *Component Edit* view allows inspecting and selecting elements of the current page in the *Page Editor* (see section 3.7.3 for details).




CBA ItemBuilder Version 10.0 will support changing the order of components in the *Component Edit* view.

Embedded HTML Explorer: The CBA ItemBuilder is designed as an *open* tool for researchers to create interactive assessment components (items, instructions, etc.). Content that can be created with the core components presented in this chapter is defined independently of concrete programming technique (see detailed about the model-based representation in section 2.11.2). However, CBA ItemBuilder items can be extended with HTML5/JavaScript content. If features are not available using the

³Sorting pages can be helpful in large projects, but the order of pages has no functional meaning: The first page that is shown is defined in the *Task*-definition (see section 3.6), and the navigation between pages is defined using links (see section 3.11), using specific finite-state machine operators (see section 4.4.6) or by assigning pages to states (see section 4.4.9).




core components or fragments are already available or programmed for browser-based environments, HTML and JavaScript can also be used along with core components. For this purpose, existing material can be imported as files and folders in the CBA ItemBuilder *Project Files* using the *Embedded HTML Explorer* (see section 3.14.2).

Rendering: The *Drawing Area* in the *Page Editor* of the CBA ItemBuilder (see section 3.1.3) is not *What you see is what you get*. To verify the final layout of a page, the internal *Rendering* (or the full *Preview*, see section 1.4) can be used. The *Rendering* tab in the left part of the CBA ItemBuilder is preferred since this preview of the currently opened page of the *Page Editor* is updated automatically if major changes are applied to the page or the button  is pressed (see section 3.7.1 for details). The *Rendering* tab is empty, if no page is opened.




The tab *Rendering* provides an instant feedback about the page layout without requiring an external Browser. The tab can be undocked (and docked to the left, middle and right part of the user-interface to fit the need of item authors, see section 6.8.1).



The tabs on the left side can be minimized and hidden in the margin with the icon  Minimize. The icon  Maximize enlarges the area by minimizing all other areas. The icon  Restore returns to the original view.

3.1.3 Middle: *Page Editor* and Other Editor-Tabs

The middle region is the main working area of the CBA ItemBuilder. This area is structured in the form of tabs, each of which is opened to edit specific content (i.e., a *Page Editor* to edit pages, an *HTML Text Editor* to edit components of type `HTML-TextField`, etc.). Changes to individual parts of the current *Project Files* made in such a tab must typically be applied, before other parts of the CBA ItemBuilder can consistently work with the changes. The CBA ItemBuilder indicates unchanged contents with a small * in the tab name.

Page Editor and Palette: The core component for designing pages is the *Page Editor*. The *Page Editor* is opened for a particular page by double-clicking on the existing page in the *Project View*. By clicking on the small icon on the right , the context-sensitive *Palette* is displayed (see Panel E in Figure 3.3).

If not requested differently, new pages are by default created with `Frame` and `Panel` (see section 3.4). The *Palette* allows selecting components that can be added in the *Page Editor* in the current context, meaning *within* the currently selected component. After creating the first page, the CBA ItemBuilder, the *Page Editor* is not yet open (see A

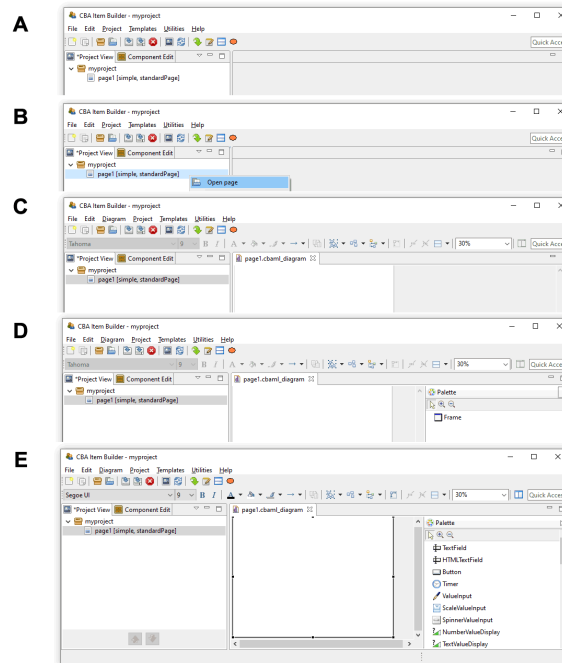




FIGURE 3.3: Five steps for opening a newly created page in the *Page Editor* and open the *Palette*.

in Figure 3.3). To open the page, double-click the page name in the *Project View* or use the context menu (right-click on the page name, see B in Figure 3.3, and select *Open page*). After opening the page the *Page Editor* appears as a new tab, showing the page with *Frame* and *Panel* in the *Drawing Area* (see D in Figure 3.3). Clicking on the small icon  opens the context-sensitive *Palette* (see D in Figure 3.3). However, as long as no component is selected, the *Palette* contains only the *Frame* component. To see all components in the *Palette* that can be added inside of components of type *Panel*, click the *Panel* that is by default created when new pages are added (see E in Figure 3.3).



The available components for the current selection in the *Page Editor* are displayed in the context-sensitive *Palette*.


The different components possible in the current context are shown in the *Palette* (according to the currently selected component in the *Drawing Area*). To insert a new component, select that particular component in the *Palette* and then click and drag to place it in the parent component in the *Page Editor* (see section 3.7 for detailed instructions).

Changes in the *Page Editor* are applied when the *Page Editor* is closed via the small cross next to the page name in the tab title  and the *Save Resource* dialog is confirmed with OK. Also, the saving of the entire *Project Files* takes over the change one all *Page Editors*.



Changes in the *Page Editor* and other editor tabs must be applied so that the CBA ItemBuilder user interface is always consistent with the contents of the current *Project File*. A small * in the title bar marks changes that have not yet been applied.

Other Editor Tabs: The CBA ItemBuilder's main area is also used for other editors. These include editors for formatting text (for components of type `HTMLTextField` see section 3.8.2, and for components of type `TextField`, see section 3.8.3). In addition, the *Resource Browser* (see section 3.10.1) for importing images and media files into *Project Files* is also shown in this area. Editors for editing dynamic content (see chapter 4) and defining the scoring items (see chapter 5) are also presented in this section. This includes syntax editors for defining *Hit Conditions* (see section 5.3.2), *Task Initialization* (see section 4.5), *Conditional Links* (see section 4.3) and for defining *State Machine Rules* (see section 4.4.4). Finally, the definition of *States* and *Variables* is also conducted in the *State Machine Tree View*, shown in this part of the CBA ItemBuilder user interface (see section 4.4.1).

Zoom: For the graphical editing of assessment components (i.e., for the design of pages in the *Page Editor*), the CBA ItemBuilder provides a zoom function. A zoom factor can be entered or selected in the toolbar (). This zoom factor only influences the display in the *Page Editor* during the creation of the pages and has no influence on the final display in the *Preview* or the test delivery.⁴ *Page Editors* for multiple pages can be opened simultaneously.

3.1.4 Right: Properties, Tasks, Variables, Value Maps and Clipboard View

The right part of the CBA ItemBuilder window also has content that in the default configuration always displayed there, and also this right area is organized with the help of tabs. A tab appears if one of the following editors are requested: *Properties* view, *Tasks*, *Variables* or *Value Maps*. Changes in the editor are saved either when the project is saved (see section 3.2.1) or when the tab is closed.


Properties view: A central part of the user interface for defining components in the *Page Editor* of the CBA ItemBuilder is the so-called *Properties* view. In this tabular display, individual properties of the component currently selected in the *Page Editor*

⁴If a mouse with wheel is available, the shortcut `Ctrl+Mouse Wheel` (`Strg+Mouse Wheel`) changes the zoom level.

can be specified and modified. The *Properties* view is hidden by default. The *Properties* view can be shown with the entry *Show Properties View* from the *Context Menu* of components in the *Page Editor*



To specify detailed *Properties* of components, the *Properties* view can be opened via the context menu and the entry *Show Properties View*.

Once the *Properties* view is opened in the right area of the CBA ItemBuilder, it shows the properties of the currently selected component in the *Page Editor*. The headline of the *Properties* view always show the components' type. The component selection in Figure 3.4 is, for instance, of type *Panel* (as can be seen from the headline:  **Panel**).



To use the *Page Editor* it is essential to know what type a selected component in the *Drawing Area* is. The type is automatically displayed in the *Properties* view and the *Component Edit* view (see section 3.7.3 for details).

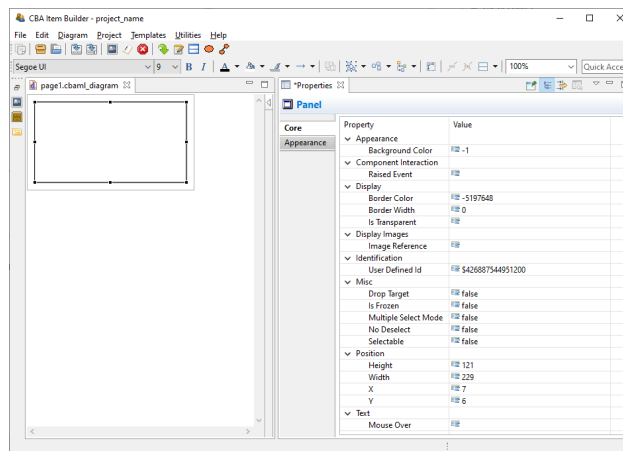


FIGURE 3.4: *Properties* view of a selected component of type *Panel*.

The *Properties* view is divided into sections (*Appearance*, *Component Interaction*, *Display*, ...) and has two tab-pages (*Core* and *Appearance*). Most properties are shown on the tab *Core*. Which properties can be changed depends on the component's type.

Section Position in the Properties View: All components that can be freely placed in the *Page Editor* have a section *Position* in the *Properties* view. The section *Position* allows the *width* and *Height* as well as the *x*- and *y*-coordinate to be defined exactly. The upper-left corner serves as the origin with the coordinates $x=0$ and $y=0$.

Tab Appearance in the Properties View: For components that display text (except for components `TextField` and `HTMLTextField`, which support different formatted text, see section 3.8), font name, font size and font color (A), bold font (A), italic font (A), and underlined font (A) can be configured in the *Appearance* tab of the *Properties* view (see Figure 3.5). To facilitate the use of consistent fonts and to narrow the fonts used in web deliveries, the available fonts can be restricted (see section 6.8.2).

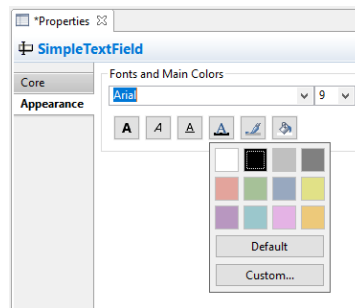


FIGURE 3.5: *Properties* view showing the tab *Appearance*.

For all components the fill color (A) is only applied, if the component is not configured to be `Is Transparent=true` in the section *Display* of the *Properties* view. Finally, for all components the border color (A) can be defined. However, a border is only shown if the property `Border Width` in the section *Display* of the *Properties* view is defined (default is 0).



Background color is only used for components that are not transparent and line color is only used, if the `Border Width` for a component is a positive number larger than zero.

Section Identification in the Properties View: The section *Identification* should be mentioned, which is important for the creation of assessment components with the CBA ItemBuilder. As described in detail in section 3.7.4, components required for scoring or dynamic parts must be named with a unique `UserDefinedId`. For that purpose, a string literal can be entered as value of the property `UserDefinedId`.

Rulers & Grid in the Properties View: The CBA ItemBuilder can align components in the *Page Editor* using a grid. Settings for spacing and visibility of the grid can be made using the *Properties* view when the *Page* is selected (see Figure 3.6). To select the *Page*, open the *Page* and click outside of the `Frame` in the *Page Editor*.

Editing the *Rulers & Grid* options in the *Properties* view is only possible, if a page is open and the page itself (and no component) is selected in the *Page Editor*.⁵

⁵Note that the configuration of the CBA ItemBuilder defines whether settings for *Rules & Grid* can be

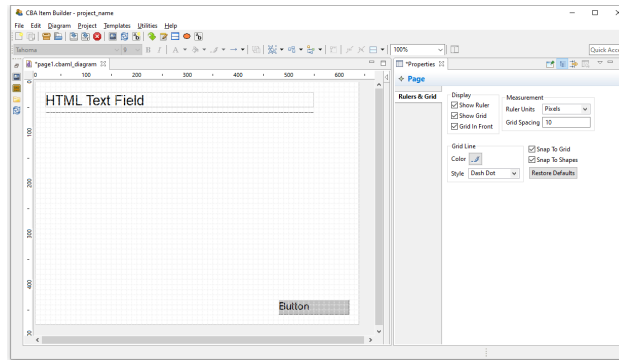


FIGURE 3.6: Properties view shows Rules & Grid if the Page is selected.



To minimize the need for entering exact coordinates when designing items, it is recommended to activate the Snap To Grid and Snap To Shape functions and to define a meaningful Grid Spacing (e.g., 10 for the Ruler Units=Pixels) or to use *Auto-Layout Panels* (see section 3.5.3).

Task Editor: A specific editor to define the entry points that are provided by a *Project File* is the *Task Editor* (see section 3.6). The editor is requested using the icon (or the entry Browse Task and Item Score from the Project menu) and allows also to define the scoring rules for each task (see section 5.3). The *Task Editor* is also displayed in the right part of the user interface (see Figure 3.7, in which the left and the middle part of the user interface are minimized).

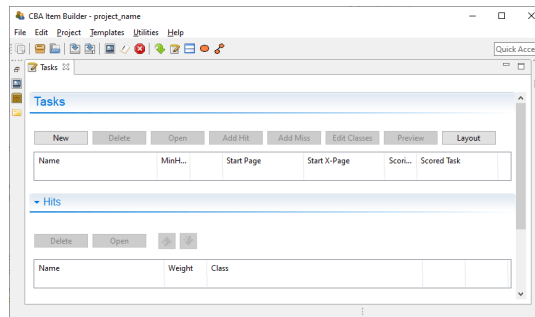


FIGURE 3.7: Task editor in the right area of the user interface.

Variables: Variables can be used in the logic-layer of the CBA ItemBuilder (to control the behavior of items using *finite-state machines* and to show dynamic content, made separately for each page or apply to all pages in the CBA ItemBuilder (the menu Utilities the item Preferences gives access to the tab CBA Rules And Grid).

see section 4.2). Variables can also be used to store results or information from content embedded via so-called `ExternalPageFrames` (see section 3.14) and variables can be used for scoring responses. As shown in Figure 3.8, variables can be declared with different `type`, need to have a `name` and a (default) `value`.

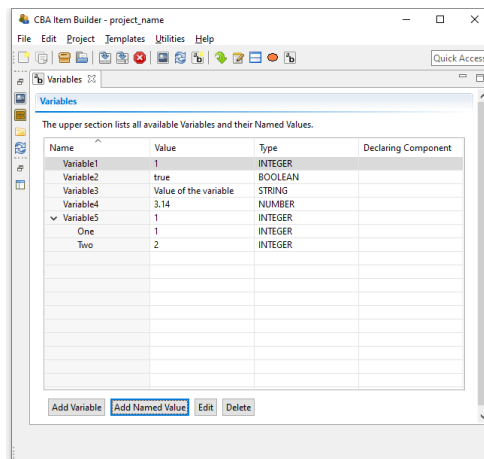


FIGURE 3.8: Variables editor in the right area of the user interface.

Value Maps: Another part of the CBA ItemBuilder's user interface shown in Figure 3.9 is the editor for so-called *Value Maps* (see section 4.2.4). *Value Maps* are tables for translating variable values (e.g., 1, 2, 3-5, see section 4.2) into pre-defined texts and media (i.e., images, audio or video files embedded in the project, see section 4.2.4).

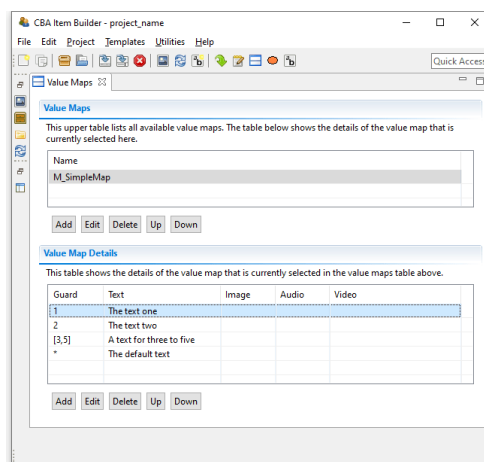


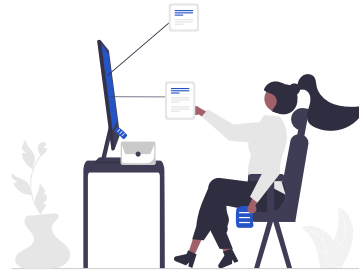
FIGURE 3.9: Value Maps editor in the right area of the user interface.

After defining a *Value Map* to translate variable values or value ranges (so-called

Guards) to texts, images or audio-/video files (see section 4.2.4 for details), map-based value-displays can be embedded on pages to show the mapped resources (using `MapBasedVariableDisplays`, see section 4.2.5). Value maps are used, for instance, to visualize FSM-variable values, to adjust the visual presentation within items dynamically and to implement drag-and-drop response formats (see section 4.2.6).

Clipboard View: The right pane of the CBA ItemBuilder can also display the *Clipboard View*, enabling components and their contents to be copied and transferred within pages, between pages, and between CBA ItemBuilder Project Files (see section 3.7.2 for details).

3.2 CBA ItemBuilder Projects Files



After describing the user interface of the CBA ItemBuilder, this section turns to the content that can be created with the authoring tool. The CBA ItemBuilder uses *Project Files* that are plain ZIP-archives. The *Project Files* (with the file extension *.zip) contain the task specification, settings, pages, syntax for scoring and the finite-state machine as well as all resources (e.g., graphics, videos) used by the project. Moreover, the ZIP archives contain the generated data required for rendering the item during runtime (see section 2.11.2).



The files with assessment content that can be created and edited with the CBA ItemBuilder are *ZIP archives*. These must not be unpacked but can be opened directly as *ZIP archives* in the CBA ItemBuilder desktop application.

3.2.1 Working with Project Files

As already described in the section 1.4.1, the CBA ItemBuilder is an editor for item projects, which are stored as *ZIP archives*. The *Project Files* are not created using the *Explorer*. Instead, *Project Files* are created inside of the CBA ItemBuilder.

Creating new Projects: After the CBA ItemBuilder has been started, a project must either be created, or an existing project can be opened. To start with an empty project, the menu `File` contains the entry `New project` (📁). To create a new project, the input of a valid *project name* is mandatory. The name selected here is subject to some restrictions:



The name of a project (*Project Name*) must not contain neither white spaces, special characters nor start with a number and using the name `template` for CBA ItemBuilder projects is not allowed. The name of the *Project File* (i.e., the ZIP archive) is by default the *Project Name*, but projects can be renamed (see Figure 3.10) and project files can be renamed (by changing the name of the ZIP archive outside of the CBA ItemBuilder or by using `Save As...`) independently.

The file name is by default identical to the *Project Name*. To save a *Project File* with a different file name, use `Save as...` (see section 3.2.1). To rename a project, use the entry `Rename project` (see Figure 3.10) in the context menu of the *Project View*.

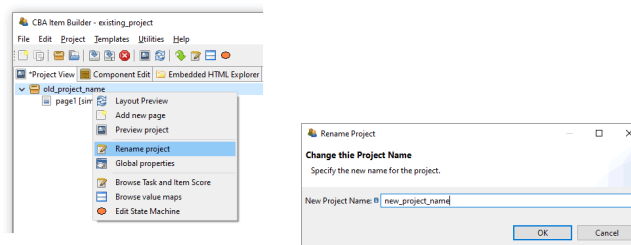


FIGURE 3.10: Rename Project-Dialog accessible from context menu in *Project View*.

Valid Project Names: If an *invalid* project name is entered, the CBA ItemBuilder displays the message shown in Figure 3.11 next to the ❌-icon. In this case, change the name to comply with the regulation and confirm the dialog with OK.

Download Project Files: Only local files can be opened in CBA ItemBuilder. Therefore, CBA ItemBuilder *Project Files* must be downloaded as ZIP archives when shared over the Internet. Web browsers have a habit of opening ZIP archives after download. Make sure you open *Project Files* unchanged as a ZIP archive in the CBA ItemBuilder.⁶

Rename Projects Files (save as...): By using `Save as` from the `File` menu and choos-

⁶For CBA ItemBuilder versions prior to 9.4: If a file with a particular file name already exists in your download folder, browsers often create copies with modified file names (e.g. `my_project (1).zip` instead of `my_project.zip`). Make sure that files are saved with their original name when you download CBA ItemBuilder *Project Files*.

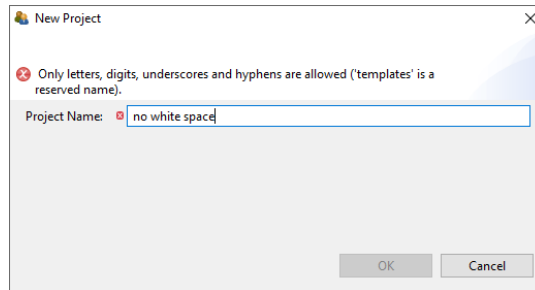


FIGURE 3.11: Warning about allowed characters for project names.

ing a different name, the CBA ItemBuilder renames the file but not the project. Renaming the ZIP file is therefore not sufficient to rename the project (see section 3.2.1).⁷

Use of CBA ItemBuilder Project Files: CBA ItemBuilder project files (i.e., ZIP archives) can be used directly in delivery software (see section 7). For the definition of tests (i.e., the so-called test assembly), assessment components stored in CBA ItemBuilder project files are referenced via the *project name* and the entry point (*Task*, see section 3.6). It is recommended not to change the file names of different versions and use tools for version management instead of files with different file names (see section 8.3.2).

The configurations required at runtime (see section 2.11.2) are automatically generated with each normal save and *Project Files* can be used directly after saving if no error message was displayed.⁸

Save Projects: Saving projects is possible using either *Save* from the *File* menu (or the icon in the toolbar).



It is possible to configure the CBA ItemBuilder so that a current project is automatically saved when a *Preview* is requested (see settings for the *CBA Preview* in section 1.4.2).⁹

If a project containing unsaved changes is to be closed, the dialog shown in Figure 3.12 is displayed.

Inconsistent Project States: Saving CBA ItemBuilder projects shows a warning if the runtime code (i.e., the definition of the item used for deployments) is invalid. Hence, a *Preview* (see section 1.4) might be required, as shown in Figure 3.13.

⁷ Prior to 9.4: Do not rename your project by just changing the name of the ZIP file on your computer without opening it in the CBA ItemBuilder. If you rename a project file in the file explorer or finder, the project will be destroyed and cannot be used any longer with the CBA ItemBuilder.

⁸ Prior to 9.0: Previous versions of the CBA ItemBuilder have provided an additional function *Generate & Save*. This is no longer necessary in the newer CBA ItemBuilder versions.

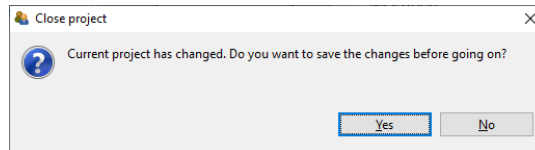


FIGURE 3.12: Dialog asking to save the changes in the current project.

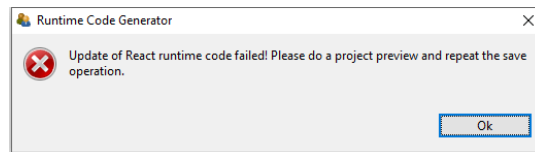


FIGURE 3.13: Dialog asking for a preview of the current project before saving the project file is possible.

The runtime code is created, whenever a project is previewed. However, there are selected configuration conflicts (i.e., inconsistencies) that make the creation of this runtime code and the preview impossible and therefore also prevent the saving of the items. In these cases the CBA ItemBuilder gives an error message and points to the place in the item definition which prevents saving (see Figure 3.14 for an example). This issue will occur as rare as possible in practice, if changes are regularly previewed and saved and if configuration inconsistencies (e.g. incorrect or empty `UserDefinedID`'s in the scoring definition) are corrected as soon as possible.

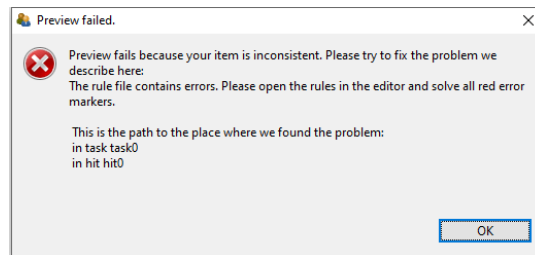


FIGURE 3.14: Request to preview the current project before saving the project is possible.

As shown in Figure 3.14, the CBA ItemBuilder provides a detailed description where the inconsistency is located (i.e., in this example, previewing and saving the project would be possible after changing the `hit0` in `task0`, see chapter 5 for details about scoring). Inconsistencies can also at other locations (task definition, conditional links, value maps, finite-state machines), but the CBA ItemBuilder will always provide a useful hint (i.e., a path) where to spot the issue.



Only consistent projects can be saved. It is therefore recommended to **preview** and **save** projects regularly and thus check their *consistency*.

Migration and use of Multiple Versions (Details): The CBA ItemBuilder is updated at regular intervals. The software development tries to ensure that project files of previous CBA ItemBuilder versions can be used in newer software versions. For this purpose, files from older versions can be opened in the more recent CBA ItemBuilder. The internal data structures will be automatically transferred to the new data formats if necessary and possible (called *Migration*). Updating old items might require migrating in several steps.



When using different CBA ItemBuilder versions, it is essential to know that more recent CBA ItemBuilder versions are expected to read projects created with previous versions (and migrations should be possible), but older program versions **can-not** read files created with **newer** CBA ItemBuilder versions.

3.2.2 CBA Presentation Size

CBA ItemBuilder projects are designed for a particular size (*CBA Presentation Size*), defined in pixel height and width. This *CBA Presentation Size* is expected to fit the expected average screen size (in pixels) to avoid raster images with too low or too high resolution.



If there is no particular other cause, an *CBA Presentation Size* of 1024 (Width) x 768 (Height), which researchers dreamed of years ago ([Bartram and Hambleton, 2006](#)), has proven to be still a good minimum size. This *CBA Presentation Size* is set as default in the CBA ItemBuilder.

Item Design: The *CBA Presentation Size* is used to define the actual size of the assessment component or at least the proportional size (i.e., the aspect ratio of width and height) used to position content in the form of components. Depending on the configuration of the deployment software, if the *CBA Presentation Size* is proportionally scaled and the *CBA Presentation Size* is the size of the content at 100% zoom-level.

The *CBA Presentation Size* for new items can be configured in the dialog *Preferences*, which can be found in the menu *Utilities* of the CBA ItemBuilder located in the menu entry *Open preferences* (see Figure 3.15). To define the item size that is used for newly created items, select the element *CBA CBA Presentation Size* on the left and adjust height and width of the item in pixels.¹⁰

¹⁰The order of height (first) and width (second) attribute in the dialog *Preferences* might be unfamiliar.

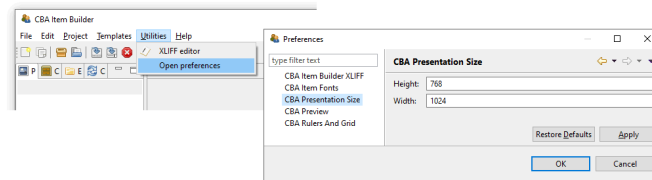


FIGURE 3.15: CBA ItemBuilder Preferences to define the *CBA Presentation Size* for new projects.

For existing CBA ItemBuilder projects, the *CBA Presentation Size* (see section 3.6.2) is part of the *Global Properties* (see section 6.3 for details).¹¹ Hence, if the preview of your item is not correctly showing the item with the correct size, the *CBA Presentation Size* for the project can be changed in the *Global Properties*. After opening an existing project, the entry *Global Properties* can be selected in the context menu of the *Project Name* in the *Project View* using the right mouse button as shown in the following Figure 3.16 and the *CBA Presentation Size* must be adjusted in the dialog that then opens:

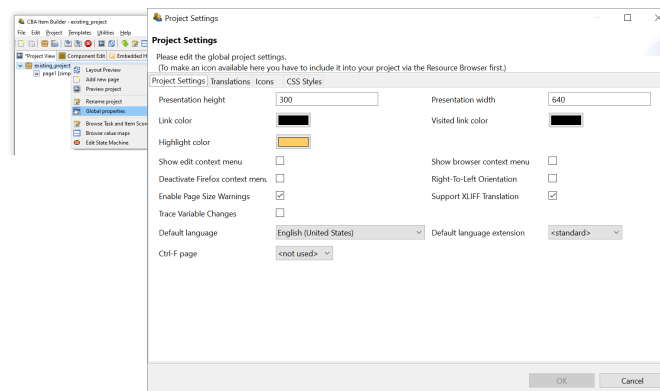


FIGURE 3.16: CBA ItemBuilder *Global Properties* to define the *CBA Presentation Size* for existing projects.

Item Deployment: For the use of CBA ItemBuilder generated items, the *Presentation Size* defines the display size and aspect ratio for which the items are designed. Within the space available for the delivery in a web browser or browser component, the deployment software is expected to provide options to customize the behavior: *Alignment of the content* in horizontal (left, center, right) and vertical (top, middle,

¹¹The *ini*-file (see appendix B.4) can be used to configure the default *CBA Presentation Size* of the CBA ItemBuilder.

bottom) direction as well as optional *Proportional Scaling of the content* (un-scaled, up-and-down, down).¹²

Tip: When starting a new project, the following questions can help to find the best *CBA Presentation Size*:

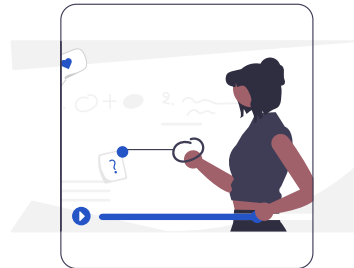
- Should the items be optimized for a 4:3 display (e.g., 1024x768 pixels) or for a 16:9 display (e.g., 1366x768 pixels)?
- Should the items be displayed in portrait format (width > height) or landscape format (height > width) with as few white borders as possible?
- Are low-resolution screens (e.g., notebooks with 1024x600 pixels), medium-resolution (e.g., computers with FullHD monitors), or high-resolution screens (e.g., modern tablets or computers with 4K resolution) expected?
- Should the items be delivered in full-screen mode, or is additional space on the screen necessary for either navigation or the browser window?
- Is extra space covered by an on-screen keyboard, or do the items either not require a keyboard, or can a hardware keyboard be assumed?

Scaling Options: Proportional scaling should ensure that the items can still be displayed even if answers to this question change or if the test is performed on heterogeneous hardware. However, the display and usability may then be less than ideal. Deployment software should provide the following options, that also can be requested by item authors for task *Preview* (see also section 1.4.2):

- **None:** Content is presented at 100%, and either a scrollbar appears (if the *CBA Presentation Size* of a task is larger than the effectively available size on screen) or space on screen remains empty. The position of the unused space will depend on the configuration for *Horizontal* and *Vertical alignment*.
- **Up:** If the *CBA Presentation Size* is smaller than the effectively available size on screen, the space will be filled, but content will not be scaled down.
- **Down:** If the *CBA Presentation Size* is larger than the effectively available size on screen, items will be scaled to fit the screen, but content will not be scaled up.
- **Both:** If the *CBA Presentation Size* is smaller than the effectively available size on screen, the space will be filled and if the *CBA Presentation Size* is larger than the effectively available size on screen, items will be scaled to fit the screen.

¹²According to this procedure, multiple item variants should normally only be created if, for example, item contents are to be distributed differently on pages in portrait format (vs. landscape format, see section 2.4).

3.3 Quick Start: Create Single Page Items



While the several chapters in this book describe the individual functions, components, and concepts for creating *complex assessment components* with the CBA ItemBuilder in detail, simple one-page items can be easily made. A simple guide to how to do this is the subject of this *Quick Start* section.



This section provides step-by-step instructions for creating single-page items. Open the CBA ItemBuilder to implement items based on the instructions. The steps described to create the different items are provided in Figures 3.17, ... as videos. If you do not want to complete this hands-on, continue reading in section 3.4.

First, it must be defined what is to be understood as a *Single Page Item*. *Single Page Items* are items that fit on a computer screen, i.e., that can be displayed without page switching. When possible, scrolling should be avoided. *Single Page Items* are typically composed of a question stem or stimulus and one single response format. The response formats *Single-Choice*, *Multiple-Choice*, and *Text Entry* are considered in this section. Moreover, CBA ItemBuilder projects containing tasks for the general instructions (embedding a video), a closing page (containing an image) will be created as *Single Page Items*.

To give a realistic impression of working with the CBA ItemBuilder, we will start with creating a master project, which we will then adapt for the different item types.

3.3.1 Create Master Project

At the beginning of preparing an assessment project, you have to decide on the screen orientation (portrait or landscape) and aspect ratio (e.g., 16:9 or 4:3) of the

Hands-on: Create Master Project -- Start


Create Master Project

- 10 Steps to create a CBA ItemBuilder Project
- The project will be used in subsequent hands-ons as starting point ("Master Project")

Idea

- Read the instruction on screen
- Try to implement the steps
- Watch the video if necessary

This hands-on will take 15 minutes.



FullScreen

Let's Start

FIGURE 3.17: Video: Hands-on section 3.3.1 ([html](#)|[ib](#)).

computer-based material (see section 2.4). The size that should be supported minimally without scrolling or scaling. As mentioned in the section 3.2.2, 1024x768 can often be a reasonable choice, which is the current default of the CBA ItemBuilder.

1. Check Settings: Before preparing multiple project files, it is recommended to check the settings of the CBA ItemBuilder. For instance, it is suggested to consider the appropriate *CBA Presentation Size* right at the very beginning. For that purpose, open the main menu *Utilities* and select *Open preferences*. The *CBA Presentation Size* should be defined as *Height: 768*, and *Width: 1024*. Change to the section *CBA Rulers And Grid* and select *Apply changes to all pages*. Finally, go to the section *CBA Item Fonts*, click *Deselect All*. Afterwards, select the two *Font Names* *Arial* and *Courier New*. Close the *Preferences* dialog with *OK*. Confirm the message *Any open editors must be closed before these changes are applied* with *OK*.




2. Create a new Project: With the main menu *File* using the entry *New project*, a new CBA ItemBuilder *Project File* can be created. It is required to enter a valid project name, for instance, *SinglePageItems_MasterProject*. As soon as the (empty) project is created, the entry *Save* from the main menu *File* can be used to specify the location where the *Project File* is stored. Select a folder and keep the file name *SinglePageItems_MasterProject.zip*.

3. Add New Page: Since creating the project, the project name *SinglePageItems_MasterProject* is shown in the *Project View* in the left part of the CBA ItemBuilder. Right-click on this project name in the *Project View* and select the entry *Add new page* in the context menu. In the dialog that appears keep the suggested name *page1* and confirm the dialog with *OK*.

4. Define Rulers & Grid: The *Project View* now contains the newly created page *page01*.

Double-click this page in the *Project View* to open the *Page Editor* in the main area of the CBA ItemBuilder window. In the *Page Editor* that opens, right-click in the gray area and select *Show Properties View*. Select *Show Ruler* and *Show Grid* in the section *Display*, change the *Rulers Units* to *Pixels* and change to 20 for the *Grid Spacing* in section *Measurement*. Finally, check the option *Snap To Shapes*.


5. Define Border: Click in the white part of the *Page Editor*. This should select the `Panel` that was automatically generated by the CBA ItemBuilder for that `page1`. The type of the selected element is shown in the headline of the *Properties* view. If the *Properties* view was closed, right-click anywhere in the *Page Editor* and select the entry *Show Properties View* again in the context menu. Find the property *Border Width* in section *Display* and enter the value 5.




6. Add `HTMLTextField` as Headline: Click again in the white part of the *Page Editor* to select the `Panel`. When the `Panel` is selected, the *Palette* should show a long list of components that could all be added to the selected `Panel`. If the *Palette*  is not displayed, it can be shown at the right window border with the small icon . Select the `HTMLTextField` entry () and draw a rectangle in the *Drawing Area* (inside the `Panel`) to add the `HTMLTextField`. Leave two boxes of the grid blank on the left, right and top and draw the rectangle 4 grid boxes high while holding down the mouse button. This way it should be 940 pixels wide and 80 pixels high, and placed at X=40 and Y=40.

Check the following values in the *Position*-section of the *Properties* view: *Height*: 80, *Width*: 940, *X*: 40, and *Y*: 40. If the *Properties* view was closed in a previous step, open it using the context menu. If the headline of the *Properties* view is showing a different component type than `HTMLTextField`, select the `HTMLTextField` in the *Page Editor*.


The `HTMLTextField` is now aligned relative to the `Panel` in the top. To enter text, double-click the `HTMLTextField` and enter the text *Headline* in the *HTML Text Editor*. Select the text and change the font to *Arial* and the size to 30. Finally, save the changes using the button *Save and Close*.



7. Add `HTMLTextField` for Content: Repeat step 5 and create a second `HTMLTextField`. Start by selecting the `Panel` in order to see the entry `HTMLTextField` in the *Palette*. Position the second `HTMLTextField` aligned below the first and draw the `HTMLTextField` with 80 pixels height and leave 20 Pixel distance. You can also change the position in the *Page Editor* or enter the values in the section *Position* of the *Properties*-view: *Height*: 80, *Width*: 940, *X*: 40, and *Y*: 140. Double-click the new `HTMLTextField` and enter the text *Content* in the *HTML Text Editor*. Change the font to *Arial* of size to 20 and close the editor with the *Save and Close* button.

8. Add and Configure *Next-Button*: Buttons can be added to components of type `Panel`. Select the `Panel` and find and click the entry *Button*  in the *Palette*. When the component type is selected in the *Palette* a new button can be added to the page by drawing a rectangle into the *Drawing Area*. It should be two three boxes (i.e., 60 pixels) height, ten boxes (i.e., 200 pixels) width and two full boxes left and below in the lower right corner of the `Panel`. You can also use the *Properties* view and enter

Height: 60, Width: 200, X: 780, and Y: 660 in the section *Position*. Find the tab *Appearance*  and click it, and change the formatting of the button. Select *Arial* as font and set the font size to 20. Moreover, set the font color to white using the button . Now set the background color using the button  and select *Custom...* in the small color dialog. In the *Choose Color* change to *RGB* and enter Red: 50, Green: 100 and Blue: 200 and confirm with *OK*. Navigate back to the tab *Core* in the *Properties* view and change the *Properties* Use Default Link Color to *false* and Use Same Color For Visited Reference to *true*.



Next, double-click the button and enter the text *Next* in the dialog *Configure a Multi-line Text*, before closing the dialog with *OK*. Finally, right-click the button in the *Page Editor* and select the entry *Set command* in the context menu. In the dialog *Set Runtime Command* click on the entry *NEXT_TASK* and confirm the selection with *OK*.

9. Define Task: Open the *Task Editor* using the main menu *Project* and the entry *Browse Task and Item Score* . In the right area of the CBA ItemBuilder main window a tab titled *Tasks* will appear. Click the first button *New* below the headline *Tasks*. This will move the cursor to the first row to the column titled *Name*. Keep the default *Name* *task0* and select the name of the page created in step 3 (page1) as the *Start Page*.

10. Save and Preview: The project can now be saved, for instance, using the main menu *File* and the entry *Save* . Confirm the message box *Save Resource* that informs that *page01.cbaml_diagram* has been modified with *Yes*. Finally, in the preview (main menu *Project* the entry *Preview Project* ) the item should now look like shown in Figure 3.18. In case of a problem, download and open the CBA ItemBuilder project file [SinglePageItems_MasterProject.zip](#).

Using the master item created in the ten steps, it is possible to create various single page items. For this purpose, the components used to capture the response must be added. The required steps will be described for the different response formats *Single Choice*, *Multiple-Choice*, and *Text Response*, in a subsection each.

3.3.2 Create Single-Choice Item

1. Open Master and Save as SinglePageItems_01SC.zip: Open the master item created in the previous section 3.3.1 (or download [SinglePageItems_MasterProject.zip](#)) using the icon  in the CBA ItemBuilder (or use the main menu *File* and the entry *Open project*). Save the *Project File* with the new name *SinglePageItems_01SC.zip* using the main menu *File* and the entry *Save as...* (or use the icon ).

2. Rename the Project: After saving the project to a new file, the project itself needs to be renamed as well. For that purpose, right click on the old project name *SinglePageItems_MasterProject* in the *Project View* and select *Rename project*. Enter the name *SinglePageItems_01SC* and confirm the dialog with *OK*.

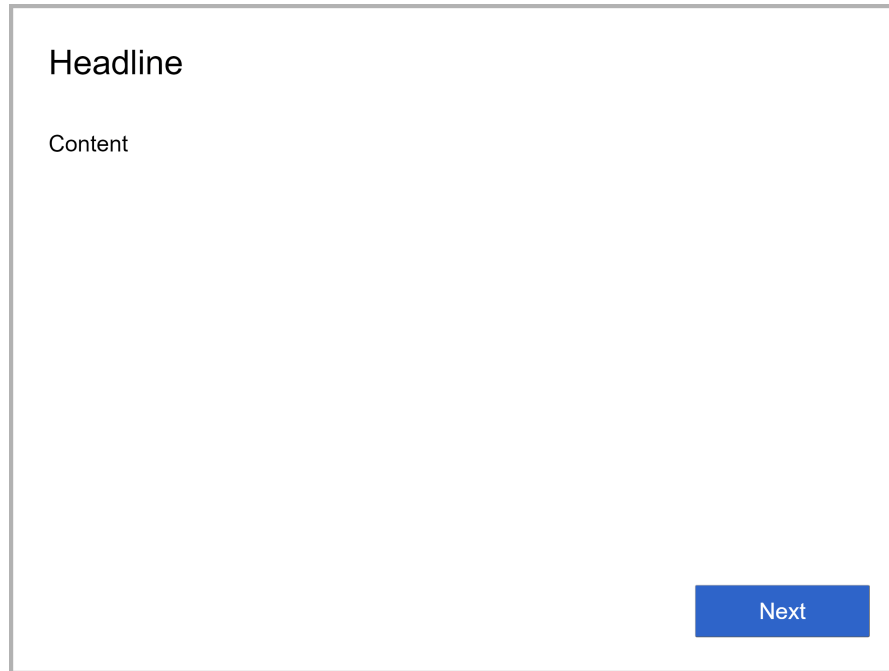


FIGURE 3.18: Output of hands-on section 3.3.1 ([html|ib](#)).

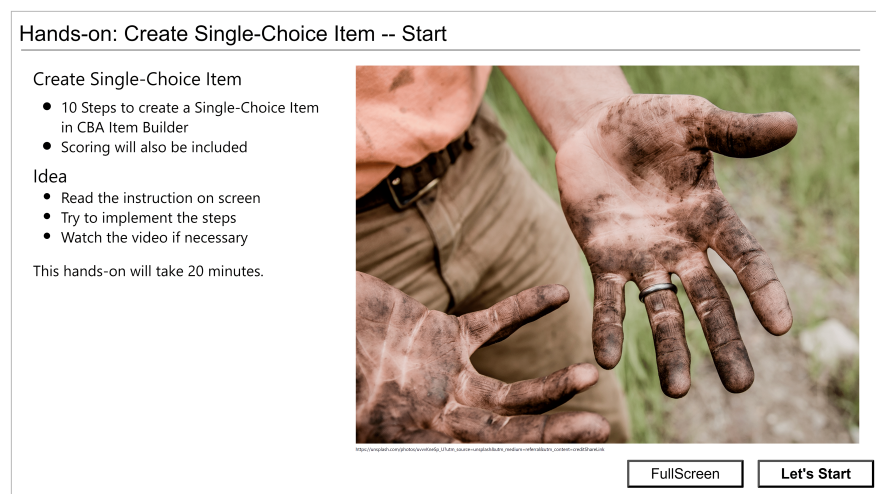





FIGURE 3.19: Video: Hands-on section 3.3.2 ([html|ib](#)).

3. Update Headline and Text: The most visible customization necessary to turn the master project into an item is editing the `HTMLTextFields` that are used for the heading and page content. To make this change, first open the page `page01` in the *Page Editor* by double-clicking on `page01` in the *Project View* (alternatively, you can also use the context menu in the *Project View*, which can be opened with a right-click on the page and which contains the entry *Open page*). Once the page is opened in the *Page Editor* (i.e., the tab in the middle region of the CBA ItemBuilder titled `page01.cbaml_diagram`), the text in the `HTMLTextFields` can be edited. Replace the text *Headline* with *Single-Choice Task* and insert as Content the text *Select the option that answers the following question: (italic) and then, after two line breaks (i.e., with an empty line in between): Which answer is correct? Close the HTML Text Editors each time with Save and Close.*


4. Add RadioButtonGroup: In the next step, a so called `RadioButtonGroup` will be inserted (see section 3.9.2). A `RadioButtonGroup` is necessary to define the connection between the components of the type `RadioButton`, inserted in the next step. To add the `RadioButtonGroup`, select the *Panel* in the *Page Editor* first. Afterwards, find and select the icon `RadioButtonGroup` ( `RadioButtonGroup`) in the *Palette*. To add a `RadioButtonGroup` to the page, click in the *Drawing Area* within the *Panel* below the second `HTMLTextField` and hold the left mouse button clicked while moving the mouse to draw a rectangle. Add the `RadioButtonGroup` to fill the remaining space with 40 pixels border around. After releasing the mouse button open the *Properties* view if necessary using the context menu (right-click on the `RadioButtonGroup` and select the entry *Show Properties View*). Check the following values in the *Position* section: *Height*: 360, *Width*: 940, *X*: 40, and *Y*: 260.

5. Add the first RadioButton: Adding `RadioButtons` is only possible within components of type `RadioButtonGroup` (see section 2.11.4). In the *Palette* the icon `RadioButton` ( `RadioButton`) can therefore only be selected if a `RadioButtonGroup` is selected in the *Page Editor*. Select the `RadioButtonGroup` in the *Page Area* and the `RadioButton` in the *Palette* and add a first `RadioButton` by drawing a rectangle within the area that is covered by the `RadioButtonGroup` with 20 pixels border to the top, left and right and 60 pixels height. Subsequently, check the position in the *Properties* view to *Height*: 60, *Width*: 900, *X*: 20, and *Y*: 20.


Change the values for the properties *Control Item Size* and *Label Distance* in section *Misc* to 30. Change to the tab *Appearance* () and select *Arial* as font and 20 as font size. It is also possible to enter *Arial* (including capitalization) and 20 directly in section *Appearance* into the properties *Font Name* and *Font Size*, respectively. Finally, double-click the `RadioButton` in the *Page Editor* and enter the text *option A* into the dialog *Configure a Multiline Text*, that must be confirmed with *OK*.

6. Duplicate the RadioButton: After creating the first `RadioButton`, copies of this component can be created using the function *Duplicate*. For that purpose, the `RadioButton` that has just been created must be right-clicked to open the context menu that contains a sub-menu *Edit* with the entry *Duplicate*.¹³ After duplicating the `RadioButton` the first time, arrange it with 20 pixels border below the first or change the *Position*

¹³If the sub-menu *Edit* does not contain the entry *Duplicate*, the text property of the component was

in the *Properties* view to Y: 100, and X: 20 and change the text property to Option B (via double-click on the duplicated `RadioButton` or using the small icon  in section *Text* of the *Properties* view). Repeat these procedure for a third `RadioButton` (Y: 180, X: 20, Text: Option C*) and a fourth `RadioButton` (Y: 260, X: 20, Text: Option D).

7. Provide `UserDefinedIDs`: Before the scoring can be defined, the created components of type `RadioButton` must be uniquely named. For this purpose, each `RadioButton` must be selected in the *Page Editor*, and an identifier must be entered in the section *Identification* of the *Properties* view as `UserDefinedId`. To keep the scoring syntax as simple as possible, we use the following schema to name the `RadioButtons`. First: `UserDefinedID: a`, second `UserDefinedID: b`, third `UserDefinedID: c`, and fourth: `UserDefinedID: d` (without white spaces).

8. Add *Classes* as *Variables* for Scoring: With the help of the `UserDefinedID`'s, the scoring can now be created exemplary for this item. Let's assume that *Option C* is correct. Then we could expect either a true/false-coding in the result data set (i.e., a variable `ScoredResponse`) or a raw-response coding (i.e., a variable `RawResponse`). In this example, we create both variables. Variables are defined in the CBA *ItemBuilder* as *Classes*. The definition of *Classes* can be requested in the *Task Editor*, which can be opened via the menu *Project* and the entry *Browse Task and Item Score* or the icon . To create two new *Classes*, open the *Task Editor* select task `task0`. Open the dialog *Task Classes Editor* with the button *Edit Classes*. In that dialog create two classes using the button *Add new class* and enter the class name `ScoredResponse` for the first class and `RawResponse` for the second class. Finally, confirm the *Task Classes Editor* with the button *OK*.



9. Define *Hit-Conditions*: Each hit represents a condition to be distinguished as value of a (categorical) variable. The value of the `RawResponse` variable should indicate which option was selected (resulting in four potential values, defined as *Hits* `RawA`, `RawB`, `RawC` and `RawD`). To define a *Hit*, first a task must be selected in the *Task Editor* (in this example the task with the name `Task01` as defined above). A click into the first table in the *Task Editor* directly on the name `Task01` selects the first task. When the task is selected, a new hit condition can be created with the *Add Hit*-button. Then, the cursor jumps to the table in the *Hits* section of the *Task Editor*. Enter as Name the name `RawA` for the first *Hit*, enter as *Weight* the value 1 and select the *Class* `RawResponse`. Then click the *open*-button or double-click the newly created *Hit* to open the *Conditions* editor in the main CBA *ItemBuilder* window. For the first *Hit* just enter here the `UserDefinedID` of the first `RadioButton` (i.e. `a`) and close the editor using the small x in the tab-title (✕). Confirm to save the changes. Continue for the remaining *Hits* using the information presented in Table 3.1. Copy the string provided in the column *Conditiona-Syntax* exactly, including all brackets (see section 4.1.3 for more information).

selected. In that case, just select another component in the *Page Editor* and then right-click on the `RadioButton` again.



TABLE 3.1: Example for *Hit*-definitions of a single-choice item

| Name | Weight | Class | Condition-Syntax |
|--------------|--------|----------------|------------------|
| RawA | 1 | RawResponse | a |
| RawB | 1 | RawResponse | b |
| RawC | 1 | RawResponse | c |
| RawD | 1 | RawResponse | d |
| ScoreWrong | 1 | ScoredResponse | ((a or b) or d) |
| ScoreCorrect | 1 | ScoredResponse | c |

To complete the scoring, two more *Hit*-conditions are needed for the coding of missing responses: A *Hit* `RawOR` (assigned to the *Class* `RawResponse`) and a *Hit* `ScoreOR` (assigned to the *Class* `ScoredResponse`). The abbreviation OR is used for *Omitted Response* (see section 2.5.2). Click the button *Open* to enter the following syntax as *Condition*:
 (((not a and not b) and not c) and not d).

10. Save, Preview and Test Scoring: The project can now be saved () and previewed (). Probably the most error-prone part of creating this item is the scoring definition. With a key combination (default is `ctrl + s`) the scoring can be checked directly in the preview (see section 1.5). The scoring is defined correctly if exactly one hit is active for each *Class* (= *Variable*) at any time. If no answer is selected, the *Hits* `ScoreOR` and `RawOR` should be active. Once a response is selected, the *Hit* to the *Class* `RawResponse` should indicate which `RadioButton` was selected. Only if *Option C* is selected, the *Hit* `ScoreCorrect` assigned to the *Class* `ScoredResponse` should be active.

3.3.3 Create Multiple-Choice Item

1. Open Master and Save as `SinglePageItems_02MC.zip`: Open the master item created in section 3.3.1 (or download [SinglePageItems_MasterProject.zip](#)) using the icon  in the CBA ItemBuilder (or use the main menu `File` and the entry `Open project`). Save the *Project File* with the new name `SinglePageItems_02MC.zip` using the main menu `File` and the entry `Save as...` (or use the icon ).

2. Rename the Project: After saving the project to a new file, the project itself needs to be renamed as well. For that purpose, right click on the old project name `SinglePageItems_MasterProject` in the *Project View* and select `Rename project`. Enter the name `SinglePageItems_02MC` and confirm the dialog with `OK`.

3. Update Headline and Text: Open the page `page01` in the *Page Editor* by double-clicking on `page01` in the *Project View* and edit the `HTMLTextFields`. Replace the text *Headline* with `Multiple-Choice Task` and insert as *Content* the text *Select all alternatives that apply: (italic)* and then, after a line break: *For which of the following entries a hypothetical rule applies? Close the HTML Text Editors each time with Save and Close.*

Select the option that answers the following question:
Which answer is correct?

- ☐ Option A
- ☐ Option B
- ☐ Option C*
- ☐ Option D

Next

FIGURE 3.20: Output of hands-on section 3.3.2 ([html](#)|[ib](#)).

Create Multiple-Choice Item

- 11 Steps to create a Multiple-Choice Item in CBA Item Builder
- Scoring will also be included

Idea

- Read the instruction on screen
- Try to implement the steps
- Watch the video if necessary


This hands-on will take 20 minutes.





FullScreen

Let's Start


FIGURE 3.21: Video: Hands-on section 3.3.3 ([html](#)|[ib](#)).

3. Add the first checkbox: Adding Checkboxes is possible within components of type Panel. In the *Palette* the icon Checkbox ( `CheckBox`) can be selected if the Panel is selected in the *Page Editor*. Select the Panel in the *Page Area* and the Checkbox in the *Palette*. Add the first Checkbox by drawing a rectangle at with 40 pixels border top, left and right and a height of 60 pixels. Subsequently, check the position in the *Properties* view, which should be Height: 60, Width: 940, X: 40, and Y: 260.

Change the values for the properties Control Item Size and Label Distance in section Misc to 30. Go to the tab *Appearance* () and select Arial as font and 20 as font size. Finally, double-click the Checkbox in the *Page Editor* and enter the text Entry 1* into the dialog *Configure a Multiline Text*, that must be confirmed with OK.

4. Duplicate the checkbox: After creating the first Checkbox, create copies of this component again using the function *Duplicate*. For that purpose, the Checkbox that has been created in step 3 must be right-clicked to open the context menu that contains a sub-menu Edit with the entry Duplicate. After duplicating the Checkbox the first time, change the *Position* in the *Properties* view to X: 40, and Y: 340 and change the text property to Entry 2 (via double-click on the duplicated RadioButton or using the small icon  in section Text of the *Properties* view). Repeat these procedure for a third Checkbox (X: 40, Y: 420, Text: Entry 3*) and a fourth Checkbox (X: 40, Y: 500, Text: Entry 4).

5. Provide UserDefinedIDs: Each Checkbox must obtain a unique name as UserDefinedID. Since UserDefinedIDs are not allowed to start with a number, the following schema is used to name the Checkboxes: First: UserDefinedID: e1, second UserDefinedID: e2, third UserDefinedID: e3, and fourth: UserDefinedID: e4 (without white spaces). To assign the UserDefinedIDs select each Checkbox in the *Page Editor*, and enter the identifier in the section *Identification* of the *Properties* view.

6. Add Classes as Variables for Scoring: With the help of the UserDefinedID's, the scoring can now be created exemplary for this item. Let's assume that Entry 1 and Entry 3 are required for a correct response. Since the item is in multiple-choice format, we can either define a true/false-coding in the result data set for each choice, or a combined score variable. In this example, we create both variables. Variables are defined in the CBA ItemBuilder as *Classes*. The definition of *Classes* can be requested in the *Task Editor*, which can be opened via the menu Project and the entry Browse Task and Item Score or the icon . To create two new *Classes*, open the *Task Editor*, select Task task0 and use the button Edit Classes. In the dialog *Task Classes Editor* use the button Add new class and enter the class name ScoredResponse for the first class and RawResponse1, RawResponse2, RawResponse3, RawResponse4 for four remaining classes. Finally, confirm the *Task Classes Editor* with the button OK.

7. Define Hit-Conditions: After defining the *Classes* the definition of Hit-conditions is necessary. Select the Task with the name Task01 in the *Task Editor*. To add a Hit-condition use the button Add Hit and type the name this Hit in the first column: Entry1Selected. Press the Tab key and maintain the default 1 as Weight in the second column and select the class RawResponse1 in the third column. Once the Hit is created, double-click the Hit or use the button Open to enter the condition-syntax e1 into the


Tab titled `Condition - <Entry1Selected>`. Close the editor and confirm to save the changes. After creating the first *Hit*, continue with the information presented in Table 3.2.



TABLE 3.2: Example for ‘Hit’-definitions of a multiple-choice item

| Name | Weight | Class | Condition-Syntax |
|-------------------|--------|----------------|--------------------------------------|
| Entry1Selected | 1 | RawResponse1 | e1 |
| Entry1NotSelected | 1 | RawResponse1 | not e1 |
| Entry2Selected | 1 | RawResponse2 | e2 |
| Entry2NotSelected | 1 | RawResponse2 | not e2 |
| Entry3Selected | 1 | RawResponse3 | e3 |
| Entry3NotSelected | 1 | RawResponse3 | not e3 |
| Entry4Selected | 1 | RawResponse4 | e4 |
| Entry4NotSelected | 1 | RawResponse4 | not e4 |
| ScoreWrong | 1 | ScoredResponse | ((e2 or e4) or not e1) or not e3) |
| ScoreCorrect | 1 | ScoredResponse | ((e1 and e3) and not e2) and not e4) |

8. Define Missing-Values: Typically, in `RadioButtonGroups` (used for the *single-choice* item in section 3.3.2) no `RadioButton` is pre-selected. Hence, the response is *missing* until one `RadioButton` of a `RadioButtonGroup` is selected. Components of type `Checkbox` (used for the *multiple-choice* items) can not distinguish whether a response was not given or the `Checkbox` was intentionally not selected. Accordingly, defining missing values for `Checkboxes` is either impossible or must take additional information into account. In this example, we only apply missing value coding to the score variable. For this purpose, we use the operator `user_interactions()` that counts the number of interactions in the current task. Suppose this number is smaller than the minimal number of interactions (one for clicking the *Next*-button). In that case, we consider the value of the variable `ScoredResponse` to be the *Hit* for an *omitted response*. In order to implement this approach for missing-value coding, select the *Task* labeled `Task01` in the *Task Editor*, add an additional *Hit* using the button `Add Hit` and name this *Hit* `ScoreOR` (Weight: 1; Class: `ScoredResponse`). Edit the *Hit* using the button `Open` (or double-click the *Hit* `ScoreOR`) and enter the following syntax: `[user_interactions()==0]`. Close the *Condition*-editor using the small x in the tab-title (✕) and confirm to save the changes. Note that a more specific missing-value coding counting only answer-change events would be possible using the CBA ItemBuilder, but is omitted here in the quick start example.

9. Re-Define Scoring for the `ScoredResponse`-Variable: What, if a test-taker clicks the *Next*-button without selecting any `Checkbox`? The *Hit* `ScoreWrong` will be active, since the defined *Hit*-condition `((e2 or e4) or not e1) or not e3` is fulfilled. It is not true, that the `Checkboxes` with the `UserDefinedId`'s `e1` and `e2` are selected. However, at

the same time the *Hit ScoreOR* is active, since the number of user-interactions would be zero. To observe this prediction open the *Preview* (main menu *Project* the entry *Preview Project* or the icon ) , click into the item and hit the key combination (default is *Ctrl + S*). You can verify that as long as the number of interactions is zero, both *Hits* are active at the same time. Reload the page in the preview (typically by hitting *F5* to try again). To resolve this issue, we can make use of the fact that at least two interactions (selecting any *Entry 2* and pressing the *Next*-button) are necessary. Hence, we can adjust the scoring syntax and use the scoring-syntax `((e2 or e4) or not e1) or not e3` and `[user_interactions()>=2]` for the *Hit* that correspond to a wrong response. Using this adaptation the condition is mutually exclusive with `[user_interactions()<2]`, the condition used for the *Hit* that correspond to an omitted response on the *ScoredResponse*-variable. To adjust the scoring, double-click the *Hit ScoreWrong* and copy the following condition syntax in to the editor provided by the CBA ItemBuilder: `([user_interactions()>1] and not (e1 and e3))`. Finally, also adjust the *Hit ScoreOR* to the following syntax: `[user_interactions()<2]`

10. Save, Preview and Test Scoring: This concludes the *multiple-choice item* and the project can now be saved () and previewed () again. Check the scoring using the *Scoring Debug Window* in the preview (see section 1.5).

Multiple-Choice Task

Select all alternatives that apply:

For which of the following entries a hypothetical rule applies?

☐ Entry 1*

☐ Entry 2

☐ Entry 3*

☐ Entry 4

Next

FIGURE 3.22: Output of hands-on section 3.3.3 ([html](#)|[ib](#)).

3.3.4 Create Text-Entry Item

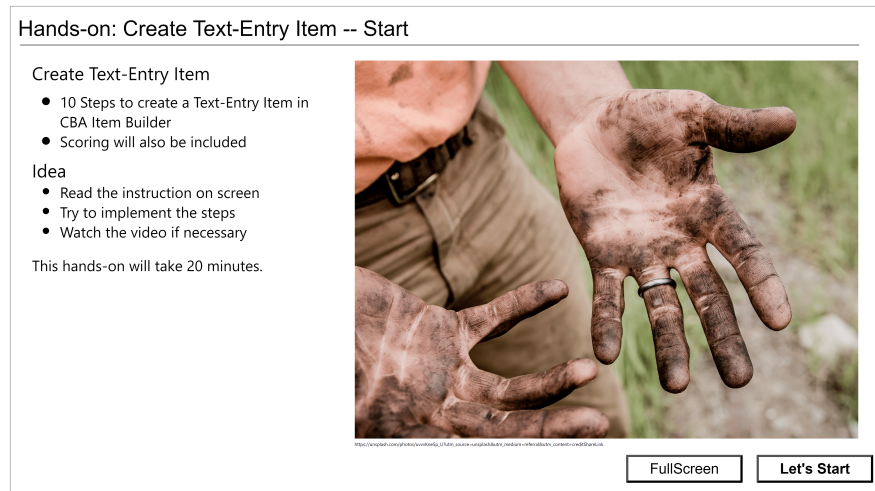





FIGURE 3.23: Video: Hands-on section 3.3.4 ([html](#)|[ib](#)).



1. Open Master and Save as SinglePageItems_03TXT.zip: Open the master item created in section 3.3.1 (or download [SinglePageItems_MasterProject.zip](#)) using the icon  in the CBA ItemBuilder (or use the main menu File and the entry Open project). Save the Project File with the new name SinglePageItems_03TXT.zip using the main menu File and the entry Save as... (or use the icon .

2. Rename the Project: After saving the project to a new file, the project itself needs to be renamed as well. For that purpose, right click on the old project name SinglePageItems_MasterProject in the Project View and select Rename project. Enter the name SinglePageItems_03TXT and confirm the dialog with OK.

3. Update Headline and Text: Open the page page01 in the Page Editor by double-clicking on page01 in the Project View and edit the HTMLTextFields. Replace the text Headline with Text-Entry Task and insert as Content the text Use the keyboard to provide an answer to the following question: (*italic*) and then, after a line break: What is the "Answer to the Ultimate Question of Life, the Universe, and Everything"? Close the HTML Text Editors each time with Save and Close.


4. Add two HTMLTextFields to Provide Context for the Text-Response: Two more components of type HTMLTextField are necessary to embed the text input into a response phrase. Select the Panel in the Page Editor. Then select the HTMLTextField entry () in the Palette and draw a rectangle in the Drawing Area (inside the Panel) to add the HTMLTextField to the page. Use the Properties view to specify the position: Height: 40, Width: 170, X: 40 and Y: 300. Double-click the HTMLTextField and enter the text The answer is (font Arial and font size 20). Save and Close the HTML Text Editor.

Add a second `HTMLTextField` with text . (font Arial and font size 20) at the position: Height: 40, Width: 40, X: 320 and Y: 300.

5. Add a `SingleLineInputField`: `SingleLineInputField` can be added to components of type Panel. In the *Palette* the icon `SingleLineInputField` ( `SingleLineInputField`) is available if a Panel is selected in the *Page Editor*. Select the Panel in the *Page Area* and the `SingleLineInputField` in the *Palette*. Add the `SingleLineInputField` to the page by drawing a rectangle in the free area between the two `HTMLTextFields` added in step 4. To make sure the `SingleLineInputField` is precisely adjusted check the position and change the Y coordinate in the *Properties* view (Height: 40, Width: 110, X: 210, and Y: 295). Moreover, define a Border Width: 2 in the section *Display*. Finally, change to the tab *Appearance* () and select Arial as font and 20 as font size.

6. Define the Input Validation Pattern: Text fields without input restrictions can be a challenge for privacy and scoring. Therefore, and because we expect a number to be the answer, we can use an Input Validation Pattern to configure that only digits can be entered. For that purpose select the `SingleLineInputField` and enter the string `[0-9]*` in the section *Misc* of the *Properties* view.

7. Provide `UserDefinedID`: The scoring definition needs an identifier for the `SingleLineInputField`. Enter the property User Defined Id: txt in the *Identification* section of the *Properties* view.

8. Add Classes as Variables for Scoring: As a result of an item with text input, two variables can be distinguished again. A `RawResponse` variable should contain the entered text, and a `ScoredResponse` variable can, if the string can be evaluated automatically using, for instance, regular expressions, indicate whether the answer is correct or incorrect. First, two *Classes* must be created to prepare the definition of the corresponding scoring syntax. The definition of *Classes* is possible in the *Task Editor*, which can be opened via the menu Project and the entry Browse Task and Item Score or the icon . To create two new *Classes*, open the *Task Editor*, select the Task task0 and use the button Edit Classes. In the dialog *Task Classes Editor* use the button Add new class and enter the class name `ScoredResponse` for the first class and `RawResponse` for the second class. Finally, confirm the *Task Classes Editor* with the button OK.



9. Define *Hit*-Conditions: *Hit*-conditions are required for the two *Classes* `ScoredResponse` and `RawResponse`. Assume the correct response is 42. The scoring-operator `matches(txt, "42")` (see section 5.3.4) can be used to compare the text entered into the `SingleLineInputField` with the `UserDefinedId: txt` with the string for a correct response. Select the Task with the name Task01 in the *Task Editor*. Add a *Hit*-condition using the button Add Hit and type the name for this *Hit* in the first column: `ScoreCorrect`. Press the Tab key and remain 1 as the `Weight` in the second column and select the class `ScoredResponse` in the third column. Once the *Hit* is created, double-click the *Hit* or use the button Open to enter the condition-syntax `matches(txt, "42")` into the Tab titled Condition - <ScoreCorrect>. Close the editor and confirm to save the changes. After creating the first *Hit*, continue with the information presented in Table 3.3.

TABLE 3.3: Example for 'Hit'-definitions of a text-entry item



| Name | Weight | Class | Condition-Syntax |
|---------------|--------|----------------|--|
| ScoreCorrect | 1 | ScoredResponse | matches(txt,"42") |
| ScoreWrong | 1 | ScoredResponse | (not matches(txt,"") and not matches(txt,"42")) |
| RawResultText | 1 | RawResponse | (not matches(txt,"") and result_text(txt)) |

As shown in Table 3.3, the *Hit* ScoreWrong of the class ScoredResponse uses the matches()- operator two times, combined as logical expression (see section 4.1.3). The condition-syntax for the *Hit* RawResultText of the class RawResponse uses the result_text()-operator, also as part of a logical expression. The result_text()-operator copies the text of the component (see the UserDefinedId provided as argument) to the variables *Result text*.

To complete the scoring of the *text-entry item*, two more *Hit*-conditions are needed for the coding of missing responses: A *Hit* RawOR (assigned to the *Class* RawResponse) and a *Hit* ScoreOR (assigned to the *Class* ScoredResponse). Click the button *Open* to enter the following syntax as *Condition*: matches(txt,"") to both *Hit*-conditions.

10. Save, Preview and Test Scoring: This concludes the *text-entry item* and the project can now be saved () and previewed (). Check the scoring using the *Scoring Debug Window* in the preview (see section 1.5).

3.3.5 Create a Closing Page

1. Open Master and Save as SinglePageItems_END.zip: Open the master item created in section 3.3.1 (or download [SinglePageItems_MasterProject.zip](#)) using the icon  in the CBA ItemBuilder (or use the main menu File and the entry Open project). Save the *Project File* with the new name SinglePageItems_04END.zip using the main menu File and the entry Save as... (or use the icon ).

2. Rename the Project: After saving the project to a new file, the project itself needs to be renamed as well. For that purpose, right click on the old project name singlePageItems_MasterProject in the *Project View* and select *Rename project*. Enter the name SinglePageItems_END and confirm the dialog with OK.

3. Update Headline and Text: Open the page page01 in the *Page Editor* by double-clicking on page01 in the *Project View* and edit the HTMLTextFields. Replace the text *Headline* with *Thank You* and insert as *Content* the text *You have reached the end of this short assessment.* and then, after a line break: *Click "Finish" to end the test.* (italic). Close the *HTML Text Editors* each time with *Save and Close*.

4. Use the Resource Browser to Insert Image: As an example on how to use images,

Text-Entry Task

Use the keyboard to provide an answer to the following question:

What is the "Answer to the Ultimate Question of Life, the Universe, and Everything"?

The answer is .

Next

FIGURE 3.24: Output of hands-on section 3.3.4 ([html](#)|[ib](#)).

Hands-on: Create a Closing Page -- Start


Create a Closing Page

- 7 Steps to create a Text-Entry Item in CBA Item Builder

Idea

- Read the instruction on screen
- Try to implement the steps
- Watch the video if necessary



This hands-on will take 10 minutes.




FullScreen



Let's Start

FIGURE 3.25: Video: Hands-on section 3.3.5 ([html](#)|[ib](#)).



an image should be displayed on the last page. For this purpose, the picture must be added using the *Resource Browser* to the *Project File* first. Open the *Resource Browser* over the main menu *Project* and the entry *Browse resources* (or use the icon ). Any image in one of the supported file formats (see section 3.10.1) can be used.¹⁴ Unpack this ZIP archive and then click the *Add* button in the *Resource Browser* of the CBA ItemBuilder to add the file `ExampleImage_min.png` to the list of *Available resources*. Close the *Resource Browser* via the small cross in the tab title (.

5. Add a `ImageField` and Link the Image: Once resources have been added to the *Project File*, they can be used in components. Next, add an `ImageField`. Components of type `ImageField` can be inserted into `Panel`s. To add an `ImageField`, open the page `page01` by double-clicking in the *Project View* for editing in the *Page Editor* and select the already existing `Panel`. Use the headline of the *Properties* view (right mouse button in the *Page Editor* and then the entry *Show Properties View* in the context menu) to check that you have selected the `Panel`. Then select the `ImageField` ( `ImageField`) in the *Palette* and click inside the `Panel` in the drawing area of the *Page Editor* with the left mouse button while moving the mouse so that a small rectangle is created. Click the `ImageField` with the right mouse button and select the entry *Link Image*. In the dialog *Select Image* that will appear, select the file name of the picture added in step 3 (e.g., `ExampleImage_min.png`). Finally, adjust the position of the `ImageField` by entering the following numbers in the section *Position* of the *Properties* view: *Height*: 380, *Width*: 944, *X*: 40, and *Y*: 240.

6. Update Button-Text: Finally, change the text of the `Button` from *Next* to *Finish*. To edit the text, double click the `Button`.

7. Save and Preview: This concludes the assessment component prepared as the last page. The project can now be saved () and previewed (). Verification of a scoring-definition is not necessary since the last page does not contain any response elements.

3.3.6 Create an Instruction Page

1. Open Master and Save as `SinglePageItems_INSTR.zip`: Open the master item created in section 3.3.1 (or download [SinglePageItems_MasterProject.zip](#)) using the icon  in the CBA ItemBuilder (or use the main menu *File* and the entry *Open project*). Save the *Project File* with the new name `SinglePageItems_01INSTR.zip` using the main menu *File* and the entry *Save as...* (or use the icon .

2. Rename the Project: After saving the project to a new file, the project itself needs to be renamed as well. For that purpose, right click on the old project name `SinglePageItems_MasterProject` in the *Project View* and select *Rename project*. Enter the name `SinglePageItems_INSTR` and confirm the dialog with *OK*.

¹⁴If you don't have a picture at hand, a sample image can be downloaded here: [SinglePageItemsResources.zip](#).

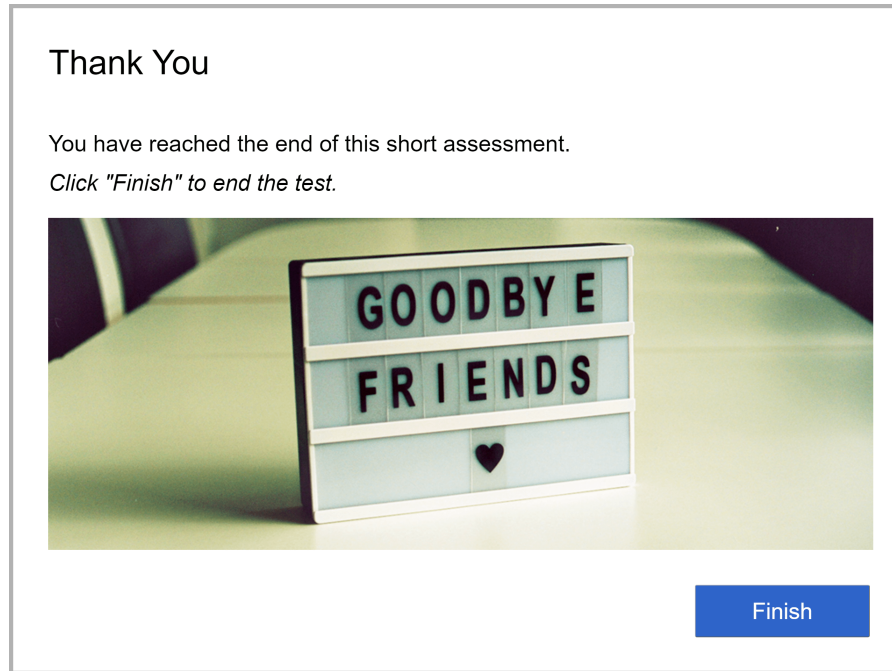


FIGURE 3.26: Output of hands-on section 3.3.5 ([html](#)|[ib](#)).

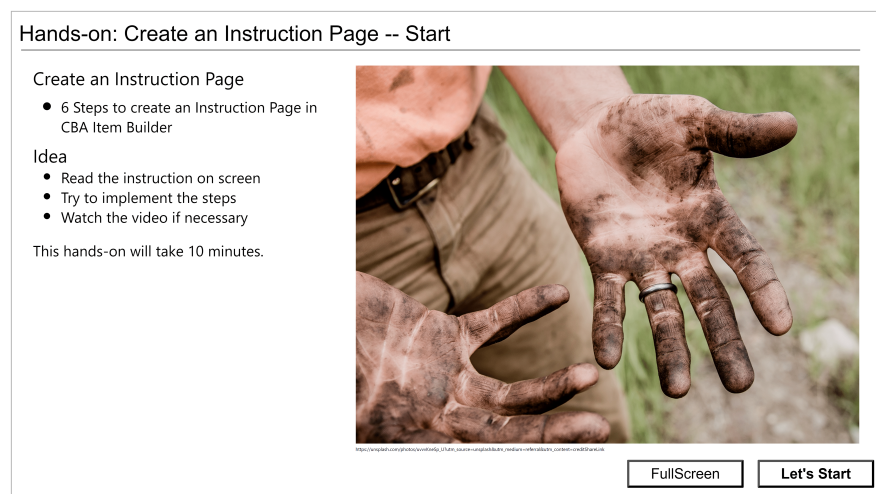








FIGURE 3.27: Video: Hands-on section 3.3.6 ([html](#)|[ib](#)).

3. Update Headline and Remove `HTMLTextField` Content: Open the page `page01` in the *Page Editor* by double-clicking on `page01` in the *Project View* and edit the `HTML-TextField`s. Replace the text `Headline` with `Welcome`. After closing the *HTML Text Editor* with the button `Save` and `Close` select the second `HTMLTextField` at `X: 40` and `Y: 140` in the *Page Editor*. After selecting the `HTMLTextField` press the delete key or right click and select `Delete` from *Model* () to delete the second `HTMLTextField`.

4. Use the *Resource Browser* to Insert Video: As an example on how to use videos, a small video should be displayed on this first page. For this purpose, the video must be added using the *Resource Browser* to the *Project File* first. Open the *Resource Browser* over the main menu *Project* and the entry *Browse resources* (or use the icon ) . Any video in one of the supported file formats (see section 3.10.1) can be used. If you don't have a picture at hand, a sample image can be downloaded here: [SinglePageItem-Resources](#). Unpack this ZIP archive and then click the 'Add' button in the *Resource Browser* of the CBA ItemBuidler to add the file `ExampleVideo.mp4` to the list of *Available resources*. The video was created using Microsoft PowerPoint's export feature. The video size used for this export was 852x480 pixel. A *.mp4-file was created by PowerPoint that can be embedded and shown on a page, after it was added to the *Resource Browser*. Close the *Resource Browser* via the small cross in the tab title () .

5. Add a *Video-Component* and *Link the Video*: The video added to the *Project File*, can now be used in components. To play videos, the CBA ItemBuilder provides the `Video`-component. `Video`-components can be added to `Panels`. To place the component of type `Video`, open the page `page01` by double-clicking in the *Project View* for editing in the *Page Editor* and select the already existing `Panel`. When the `Panel` is selected, components of type `Video` can be selected in the *Palette*. Select the `Video` () in the *Palette* and click inside the `Panel` in the drawing area of the *Page Editor* with the left mouse button while moving the mouse so that a small rectangle is created. Click the `Video` with the right mouse button and select the entry `Link Video`. In the dialog *LinkVideo* that will appear, click the button *Browse for Internal Media* and select the video's file name added in step 3 (e.g., `ExampleVideo.mp4`). Adjust the `Video` position by entering the following numbers in the section *Position* of the *Properties* view: `Height: 480`, `Width: 852`, `X: 40`, and `Y: 140`. Finally, find the section *Misc* in the *Properties* view and configure `Automatic Start: true` and `Hide Controls: true`.

6. *Save and Preview*: This concludes the assessment component prepared as the first page. The project can now be saved () and previewed () . Note that the CBA ItemBuilder will inform you when requesting the *Preview*, that an automatically started audio or video might not be previewed correctly until the first user-interaction took place in the browser (see section 1.4.2 for details). Check the box `Show login dialog`. In the preview, a small dialog appears, asking for a `Username`. Use any `Username`. Because a user-interaction took place for entering the `Username`, the video will automatically start.

Summary: This hands-on section described how to create single-page items using the CBA ItemBuilder. Items made according to this template can be used for

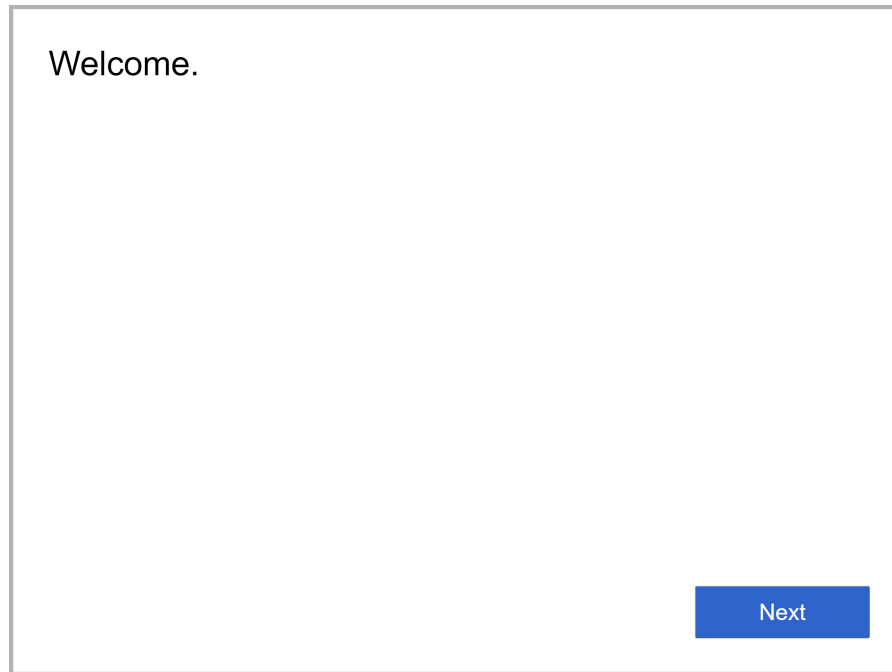


FIGURE 3.28: Output of hands-on section 3.3.6 ([html](#)|[ib](#)).

computer-based assessments. The quick-start section 7.1 picks up the created CBA ItemBuilder *Project Files* again and shows how to combine the *Tasks* as simple of-line delivery. The sample items illustrate the three most common response formats (single-choice, multiple-choice, and short text responses). Only one item was placed on each page. The welcome page was used to illustrate how images can be embedded, and the instruction page gives an impression of how to include videos into assessment components created with the CBA ItemBuilder. The created ItemBuilder *Project Files* each contained only one *Task* with only one page of the type `Simple Page`. Dynamic content that will be introduced in chapter 4 was not required to implement the simple single page items.

3.4 Pages and Page Types




Assessment components created with the CBA ItemBuilder, such as items and instructions, are composed by *Pages*. Each page has a *Page Type*, which is defined when the page is created. The type of a page determines how the pages can be used to design items.

3.4.1 Basic Page Type simple Page



Pages in the CBA ItemBuilder have a *Page Type*, for instance, *Simple Page*, which is set when the page is created. The *Page Type* of a page can be changed after the page is created using the dialog *Page Settings*.

Depending on the intended task design, pages of different *Page Type* are required. Section 3.4 gives an overview of the types available in the CBA ItemBuilder and explains the primary purpose for each *Page Type*.

New Page Dialog: Creating new pages in a project is done by clicking the icon  or by selecting the entry *Add new page* in the context menu (see Figure 3.29).

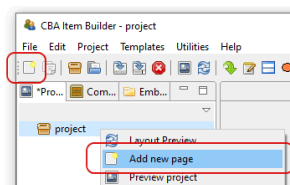


FIGURE 3.29: Toolbar icon and context menu to create a new page.

Both possible methods open the *New Page*-dialog (see panel A in figure 3.30), where

the name for the new page to be created must be entered. Page names must not start with a digit and may only contain characters and digits, without special characters (except for `_`).

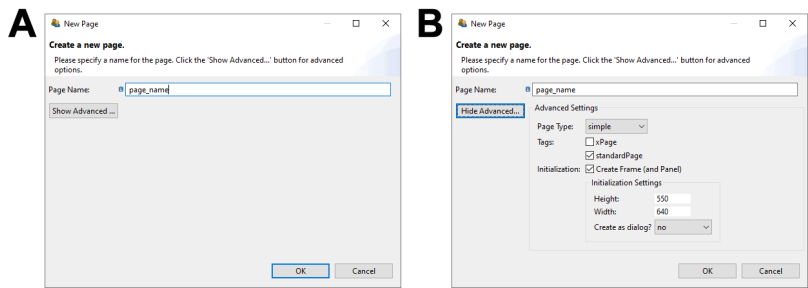


FIGURE 3.30: CBA ItemBuilder *New Page*-Dialog (left without and right with *Advanced* properties)

By clicking the `Show Advanced` button in the *New Page* dialog, additional properties of new pages to be created can be set (see panel B in figure 3.30).

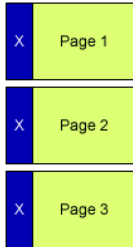
The *page type* can be selected (see section 3.4), pages can be tagged as `standardPage` or `xPage` (see section 3.4) and it can be deselected that components of type `Frame` and `Panel` are automatically created when the page is created (see section 3.5). If this default is kept, then it can be specified in which size `Frame` and `Panel` are created and whether the `Frame` (and thus the page) is created as *Dialog* (see section 3.15).



Most assessment components created with the CBA ItemBuilder consist of one or more pages of the type *Simple Page* (see Table 3.4).

TABLE 3.4: Basic page type *Simple Page* of the CBA ItemBuilder and *xPages*

| Page Type | Description |
|--------------------|--|
| <i>Simple Page</i> | Pages of the type <code>simple page</code> can be used to implement assessment components such as items or instruction pages with the CBA ItemBuilder. By default, simple pages are shown in CBA ItemBuilder <i>Tasks</i> separately on screen, filling the available space up to the defined <i>CBA Presentation Size</i> . Simple pages can be linked to other simple pages. Simple pages are the primary page type that can also be used for advanced applications, such as dialogs or as page components included in <i>PageAreas</i> -components. |
| <div>Page 1</div> | |
| <div>Page 2</div> | |
| <div>Page 3</div> | |

| Page Type | Description |
|---|---|
| <i>xPage</i>  | A page that is displayed simultaneously with another page in a <i>Task</i> is called an <i>xPage</i> (see section 3.6.2). <i>xPages</i> are typical of type <code>Simple Page</code> , and <i>xPages</i> defined for a <i>Task</i> can remain visible while navigating between different non <i>xPages</i> in the main area of the task. For example, <i>xPages</i> can be used to implement a common instruction or a navigation area that is visible during the complete task. Note that all other page types can also be marked as <i>xPage</i> and that <i>xPages</i> can be combined with pages other than <code>Simple Pages</code> . |

Example Item Illustrating all Page Types using Different Tasks

Task name: Task01
Page type: Simple Page
Page name: page 1

Click [here](#) for a summary of tasks and page types.

Goto Task02 (Illustrating an xPage Layout)

File: ExampleTasksIllustratingAllPageTypes.zip

FIGURE 3.31: Item illustrating different *Page Types* ([html](#)|[ib](#)).

The item in Figure 3.31 uses multiple tasks to illustrate different page types and their use. The additional page types shown in Figure 3.31 and the components that can be added to pages of particular type will be described in subsection 3.13.

3.4.2 Pages Flagged as *xPage*

xPages are required to implement so-called *xPage*-layouts. An *xPage*-layout combines two pages simultaneously on screen (one regular page and one *xPage*). Pages can be changed using *Links* (see section 3.11 and *Conditional Links*, see section 4.3, and by assigning *State* to pages, see section 4.4.9). The *xPage* layout briefly described in Table 3.4 (see section 3.6.2 for details) requires that pages of the appropriate size for the particular area be linked in the page-area and the *xPage*-area. To ensure that this distinction fundamental to as *xPage*-layouts can be automatically maintained when creating *Links* using the CBA ItemBuilder's user interface, regular pages, and pages with the *xPage* tag are consequently separated. *xPages* are defined when creat-

ing new pages (see Figure 3.32) in the Advanced-section of the New Page-Dialog of the CBA ItemBuilder.



The *xPage* flag is used to help creating complex items for *Tasks* using an *xPage*-layout (see section 3.6.2).

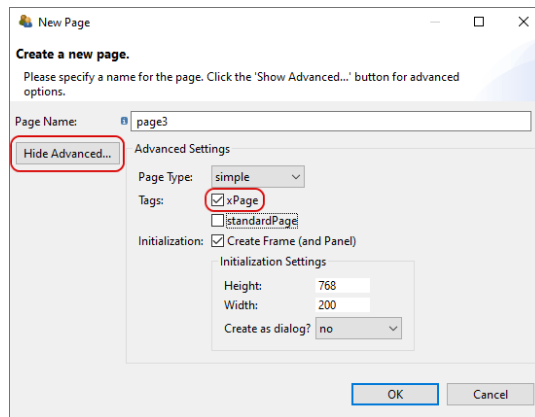


FIGURE 3.32: Tag for creating a new page as *XPage*.

Pages can also be flagged as *xPage* (see Figure 3.33, when pages are imported from the *Template Browser* (see section 6.8.7 for details).

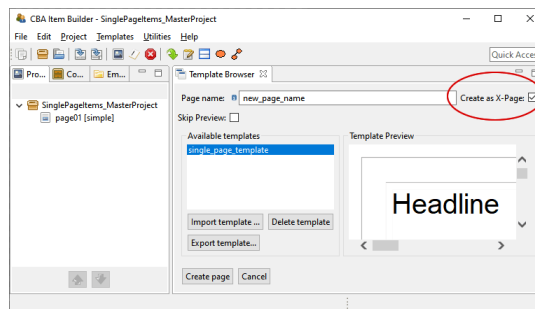


FIGURE 3.33: Checkbox to tag a page created using *Template Browser* as *XPage*.

The distinction between regular pages and *xPages* simplifies and structures complex assessment components created with the CBA ItemBuilder. For the definition of links, pages and *xPages* are not accidentally mixed. Section 3.11.4 describes how this classification is also applied to dialogs and popups.

The tagging of a page as *xPage* and / or *standardPage* can be changed again at any time

via the Page Settings-dialog (see figure 3.34). The dialog can be called via the context menu of a page in the *Project View*.

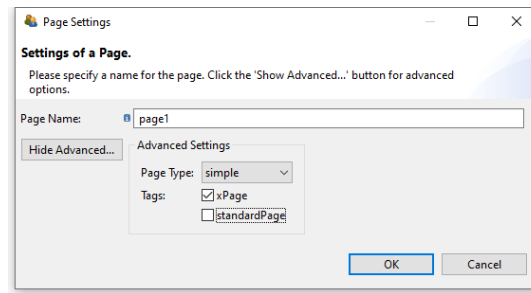


FIGURE 3.34: Dialog for changing *Page Settings*.

The page type `simple page` is the default page type for creating assessment components using the CBA ItemBuilder. The size of `simple page` is defined by the a component of type `Frame` used as root element (see section 3.5.1). Each page needs a `Frame` and only one `Frame` can be defined each page. Pages of type `simple page` can be shown in `PageAreas` (see section 3.5.4) and the `Frame` of a `simple page` can contain one or multiple `PageAreas`. Pages of `simple type page` are also the basis for creating *Dialog*-pages (i.e. pop-up pages within the item, see section 3.15).

The CBA ItemBuilder also provides additional page types. The page type is central to the functions and features of the CBA ItemBuilder:

- Selected components can be added only to pages of a particular type. Unless otherwise specified, the different components described in this chapter are intended for pages of type `simple page`. The *Component Register* (see Appendix B.6) provides a complete listing of which components can be added to pages of which type.
- To implement a specific task layout, pages flagged with the `xPage` attribute are required. To implement an `xPage-Layout`, a regular page and an `xPage` must be combined (see section 3.6.2 for details).
- The page type and the `xPage` attribute defines which pages can be linked to each other using simple *Links* and *Conditional Links* (see section 3.11). For example, pages defined as `xPages` can only link to other `xPages`. Similarly, links within *Web Child Pages* can only be made to other *Web Child Pages* (see section 3.13.2 for details).

The task definition (see section 3.6.1) is crucial for defining the layout of an assessment component created with the CBA ItemBuilder. The task definition defines if an `xPage-Layout` is used (see section 3.6.2) and defines the page (and thereby the page type) of the page used as the *Start Page*.

3.5 Basic Containers (Frame, Panel and PageArea)



The graphical design of assessments in the CBA ItemBuilder is done via pages, each of which contains first a `Frame` and then usually a `Panel`.




Since version 9.3 of the CBA ItemBuilder, `Frame` and `Panel` are automatically created by default when a new page is created. Only if this option is disabled (see section 3.4), `Frame` and `Panel` have to be added as components manually.

This section describes how to start from blank pages yourself (section 3.5.1), and how to nest multiple `Panels` (section 3.5.2) and how to use advanced features of `Panels`, such as automatic table layout (see section 3.5.3). Finally, the section ends with a description of `PageAreas` that can be used to re-use pages multiple times (see section 3.5.4).

3.5.1 Top-Level Component: Frame

A component of type `Frame` is required as highest component in the hierarchy for pages of all types. Exactly one `Frame` must be defined on each page, and `Frames` serve as the container for all other components.

Preparation for Empty Pages: Pages must always have a component of type `Frame` at the highest hierarchical level. If the item design does not use an *xPage* (see the description of *xPages*, above in the section 3.4), then the size of the `Frame` should correspond to the size defined as *CBA Presentation Size* (see section 3.2.2).

To define a `Frame` manually, the page must first be opened in the *Project View* by double-clicking. Then the *Page Editor* opens, as shown in figure 3.35. To add a frame, first select the component `Frame` in the *Palette* (icon  `Frame`). Then use the mouse to draw a rectangle in the empty area of the *Page Editor*, where you place one corner by

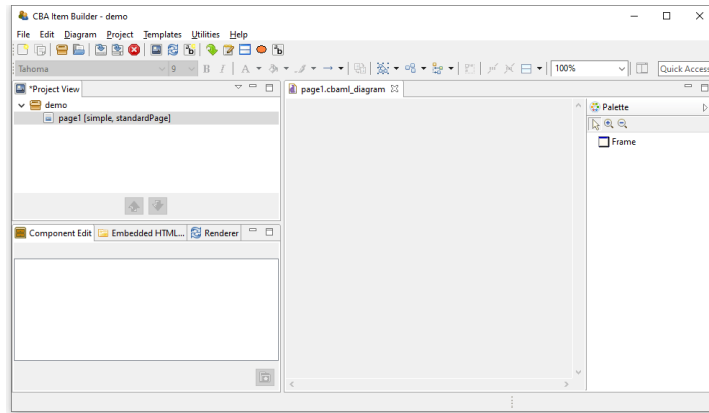



FIGURE 3.35: Empty *Page Editor* if no *Frame* is defined.

clicking, and then, keeping the mouse button pressed, define the size of the *Frame* by moving the mouse over to the opposite corner. Then release the mouse button.



If the *Palette* is not visible in the current workspace of the CBA ItemBuilder, you can show it with the small button  (see section 3.1.3 for details about hiding and showing the *Palette*).

The CBA ItemBuilder item shown in Figure 3.36 illustrates with a video the necessary steps to a) create a new page without *Frame* and *Panel* and b) adding manually a *Frame* and a *Panel* which fill the complete *CBA Presentation Size* (in this example 1024x768 pixels).

Frames are created as regular components in the *Page Editor*. After the component *Frame* has been selected in the *Palette* and added to a page using the point-and-click interface, it is recommended to configure the size and position exactly via coordinates (see 3.7.1). If no additional borders are intended, the *Frame* should be set to the coordinates $x=0$ and $y=0$, with a *Height* and *Width* matching the *CBA Presentation Size* (see section 3.6.2 for details).



The preparation of a *Frame* is necessary for each page and automatically performed, if the *Initialization* is not de-activated.

For this reason, previewing a project with a pages that do not yet contain a *Frame* lead to the error message shown in Figure 3.37.

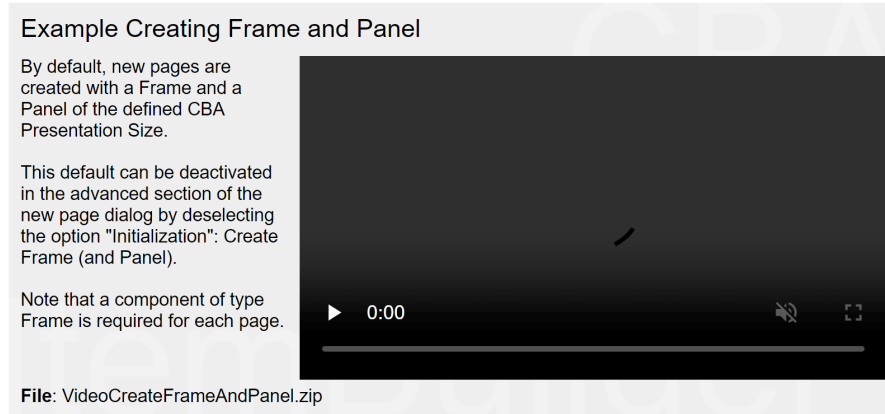


FIGURE 3.36: Video illustrating manual definition of `Frame` and `Panel` ([html|ib](#)).

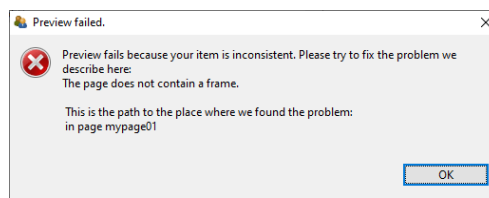


FIGURE 3.37: Message of a failed *Preview* because every *Page* requires a *Frame*.

Since components of the type `Frame` represent the highest containers in the hierarchy, no further components can be placed on a page outside a `Frame`.



The `Frame` of a page contains the complete page content. Accordingly, deleting the `Frame` of a page will remove all defined components.

To cancel accidental deletion or replacement of components in the *Page Editor*, the *Undo* function can be used (see section 3.7).

Size, Identifier and Appearance of `Frames`: Like all visible elements, components of type `Frame` can be configured using the properties in the section *Position* of the *Properties* view (*x*, *y*, *Height* and *Width*, see section 3.7.1). `Frames` can be uniquely named for the connection with dynamic item contents with a *User Defined Id* (see section 3.7.4).



The configuration of the size of a `Frame` requires special attention, because the `Frame`

defines the size of a page. Scrollbars can appear if the height or width of the `Frame` of a page are larger than the height or width of the *CBA Presentation Size*.

`Frames` can be configured with a border, which is displayed with a width of `Border Width` pixels in the color `Border Color`. By defining a border with a `Border Width` different from zero, the `Frame` and thus the required space for a page will be enlarged. If a `Frame` is defined with a width of 100 pixels and an border is defined with a `Border Width` of 10 pixels, 120 pixels are required for the complete display of the page. The inner size of `Frames` therefore always remains unchanged.



The space required for a `Frame` increases when a `Border Width` larger than zero is defined.

The color of the border will be changed in the *Properties* view on the tab *Appearance* via the stroke color (see section 3.1.4). Without affecting the size of a `Frame`, the background color of a `Frame` can also be changed using the fill color in tab *Appearance* of the *Properties* view.

To change the colors (`Background Color` and `Border Color`) directly in the *Properties* view, the color values must be known in the internal representation of the CBA ItemBuilder. Since this is usually not the case, edit the colors in the tab *Appearance* and not in the tab *Core*. However, you can copy color values that you have defined using the color selection dialog box as numerical values in the *Properties* view and insert them, for example, in the properties of other components.

Additional Properties of `Frames`: `Frame`-type components have additional properties that are covered in detail in later sections of this book. `Frames` can be configured as dialogs. Therefore the properties `Dialog` and `Closable` can be changed, located in segment *Misc* of the *Properties* view. It should be noted that the property `Closeable` is only relevant if the property `Dialog` is not configured as `UNDEFINED` (default). For full details on popups and dialogs, see section 3.15.

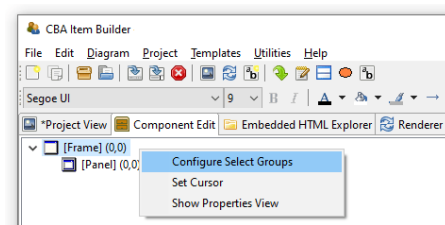
When a `Frame` is clicked, an event can be raised, which can be processed by a *Finite-State Machine* in the dynamic part of the CBA ItemBuilder (see chapter 4). For this purpose, `Frames` provide the property `Raised Event` in the segment *Component Interaction* of the *Properties* view, which can be configured as described in section 4.4.3. If the `Frame` contains other components, these components will absorb this event.

Moreover, the property `Mouse Over Text` in the segment *Text* of the *Properties* view allows for `Frames` to specify which text is displayed as a tool tip when the mouse is moved over the `Frame`.

`Frames`-Select Groups: The CBA ItemBuilder currently provides two ways to group components for single- and multiple-choice responses. As described in section

[illegible]

Select Groups are defined for *Frames* using the dialog shown in Figure 3.38. This dialog is accessible from the context menu of the *Component Edit* (see section 3.1.2), when the *Frame* is selected (see Figure 3.39).




For a complete description of all the functions of *Select Groups*, see section 3.9.4.

3.5.2 Containers of Type Panel

Each page requires a `Frame` (see previous section 3.5.1), that can contain *one* or *multiple* `Panel`s. At least one `Panel` is required for most pages. Exceptions are pages that are

designed using `PageAreas` (see section 3.5.4). Accordingly, components of type `Frame` allow adding one or more components of type `Panel` and `PageArea`.

To define a `Panel`, the page must first be opened in the *Project View* by double-clicking. Then the *Page Editor* opens. To add a `Panel`, first select the component of type `Frame` in the *Page Editor*. If the `Frame` is selected, the *Palette* allows to select the component `Panel` ( `Panel`). Use the mouse to draw a rectangle within in the area of the selected `Frame`, and then, keeping the mouse button pressed, define the size of the `Panel` by moving the mouse over to the opposite corner. Then release the mouse button. The main properties of `Panels` can be defined using the *Properties* view.

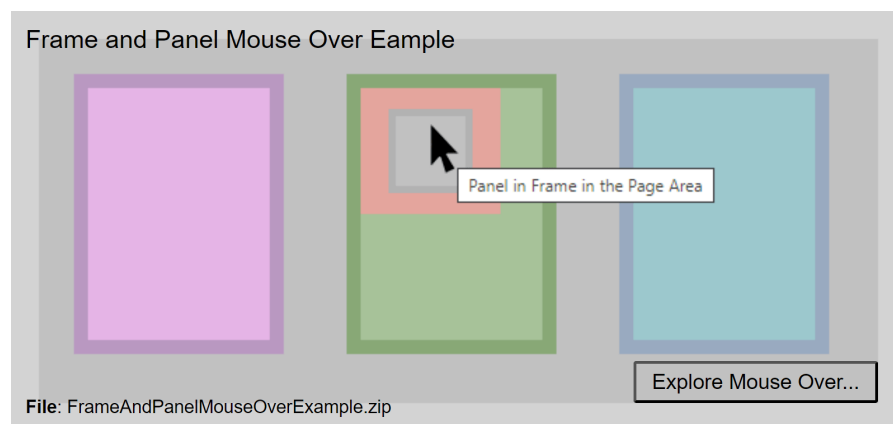


FIGURE 3.40: Example for `Panels` and `PageAreas` nested within `Frames` ([html](#)|[ib](#)).

Figure 3.40 shows how `Frames` can contain components of type `Panel` (pink rectangle on the left and blue area on the right) and of type `PageArea` (green rectangle in the middle). All components are configured with border (10 pixel for `Panels` and the `PageArea` and 20 pixels for the gray `Frame` at the top level). Use the button `Explore Mouse Over...` to see the type of each component as mouse over text. As you can see, the `PageArea` contains a `Frame` (red) that itself contains a `Panel` again (gray).

Technical Use of `Panels`: `Frames` are containers that can host `Panels`, and `Panels` can serve as containers for many other components (see Figure 3.145 in the previous section 3.5.1).



A first `Panel` is required for most pages for technical reasons (to host components), that can be also used to design the page.

Most of the possible components that can be added to `Panels` will be briefly described in this chapter, starting with components for displaying text (see section 3.8), components for collecting responses (see section 3.9), components for multimedia con-

tent (audio and video, see section 3.10), and components for linking between pages and triggering events (see section 3.11).



Deleting components of type `Frame` or `Panel`s removes all nested components.

Nesting Panels: `Panel`s can contain `Panel`s. This feature is called *nesting*, since the inner `Panel` is nested in the outer `Panel`. The use of nested `Panel`s is useful for two reasons: Additional `Panel`s can be used for graphical design of pages (e.g., by adding a background to the panel). Moreover, additional `Panel`s can facilitate page editing and can be used for grouping components. Components grouped together in `Panel`s can be moved and placed together in the *Page Editor* (see section 3.7.1).



If (nested) components like `Frame` and `Panel` are perfectly overlapping (i.e., $x=0$ and $y=0$ coordinates for the nested child component and identical `Width` and `Height` of parent and child component), only the nested inner components can be selected in the graphical *Page Editor*. To select the outer components, the *Component Edit* view must be used (see section 3.7.3).

Layout Type: Components of type `Panel` provide two different *Layout Types*. By default `Panel`s have the `Layout Type=ABSOLUTE`, meaning that the position and size of components within a `Panel` are defined by coordinates x , y and by `Height` and `Width`. This relates to the general layout approach that uses absolute coordinates (in pixels) to define the exact position of elements. The size and position of a `Panel` within a `Frame` or within another `Panel` with `Layout Type=ABSOLUTE` is defined by the coordinates x , y and by the `Height` and `Width` of the `Panel`. All size-related numbers can be defined accurately by entering numbers in the *Properties* view, or by drag and drop of components in the *Page Editor*. Within `Panel`s using the `Layout Type=ABSOLUTE`, elements can also be moved with the arrow keys of the keyboard. Make sure to adjust the `Snap to Grid` setting and the `Grid Spacing` (see section 3.1.4) to get best results. If the first `Panel` within a `Frame` is defined with some distance to the outer `Frame` (i.e., $x > 0$ and $y > 0$), it can be used to create margins for pages. In this case, make sure the `Width` and `Height` of the `Panel` are smaller than the `Width` and `Height` of the `Frame`. The border of `Panel`s can be designed to show a visible rectangle around the `Panel` (see `Border Width` and `Border Color` described in section 3.7.5). `Panel`s can be used to show a (background) image (see section 3.10).

A second `Layout Type=GRID` can be configured to create table-based layouts using multiple nested `Panel`s. This feature is called *Auto-Layout* in the CBA ItemBuilder and is only available for components of type `Panel`. The position of the first `Panel` using either `Layout Type=GRID` or `Layout Type=ABSOLUTE` is defined with x and y within the `Frame` component of a page, and the size of the first `Panel` is defined by the properties `Width` and `Height`. The `Layout Type=GRID` can be used to change the positioning of nested elements for `Panel`s as described in section 3.5.3.

3.5.3 Auto-Layout-Panels

Pages should look consistent and alike across multiple parts of an assessment. For example, padding and margins should be similar if various pages are designed for one or several *Tasks* in an assessment project. The CBA ItemBuilder supports this requirement by providing so-called *Auto-Layout Panels*. The configuration of *Auto-Layout Panels* starts with the selection of the entry *Set Auto Layout* for a selected *Panel*, as shown in Figure 3.41 for example in the *Component Edit*.

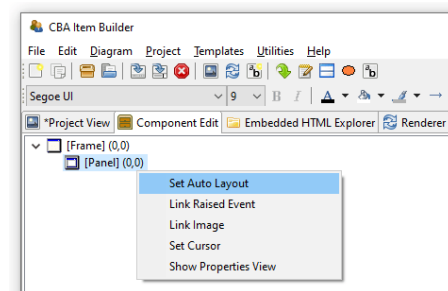


FIGURE 3.41: Context menu entry *Set Auto Layout* for selected *Panels* to open the dialog *Auto Layout Panel Properties*.

The dialog *Auto Layout Panel Properties* allows defining the number of rows and columns that are used for the automatic table layout. *Columns* and *Rows* can be of different type (absolute, percent, fill and auto) and the type defines the height and width of rows and columns.

- **Absolute:** A row or column will take exactly the defined height or width in pixel.
- **Percent:** The height of row or the width of columns will be forced to match the defined percent.
- **Fill:** The row height or column width will fill the available space in the *Auto-Layout Panel*.
- **Auto:** The height of row or the width of columns will be created to match the required size of its content.

As shown in Figure 3.42, cells can also be merged and for each cell the vertical and horizontal alignment of the content can be defined.

Panels with *Auto-Layout* then contain as many child elements of type *GridArea* as cells are configured over the rows and columns. Each element of type *GridArea* contains a single component (e.g., an *HTMLTextField* or a *SingleLineInputField*) or a *Panel* that can contain multiple components.

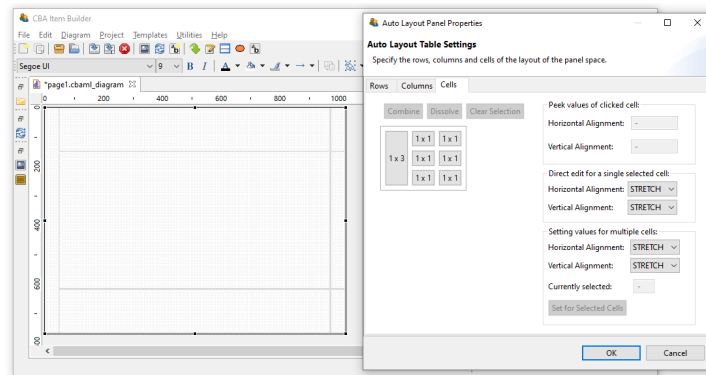


FIGURE 3.42: Dialog *Auto Layout Panel Properties* showing an example with merged cells.



When using the `Layout Type=GRID`, a table structure for components of type `Panel` can be created using the `Set Auto Layout` context menu item. The structure is defined with `Rows` and `Columns` and the size and position of the resulting table cells of type `GridArea` are automatically calculated and cannot be changed in the *Page Editor*.

The calculation of cell sizes for *Auto Layout Panels* is done at design time of the item and is performed in the built-in *Rendering-view*. Based on the defined rows and columns and the combined cells, the resulting `GridArea` are created that can be selected in the *Component Edit*. To delete or change `GridArea`, the dialog *Auto Layout Panel Properties* must be used. It is also accessible from the context menu in the *Component Edit* after selecting the `Panel` that provides the *Auto-Layout*.

`Panels` with *Auto-Layout* can have cells that again contain `Panels` with *Auto-Layout*. However, a `Panel` can only be defined as `Layout Type=GRID` (i.e., to use *Auto-Layout*) if the `Panel` does not already contain components. Otherwise, an warning is displayed (see Figure 3.44).



The definition of *Auto Layouts* only works if the page can be displayed in the *Renderer* view.

It is required to fix possible configuration errors in the page before calling *Set Auto Layout*. Only if the page is displayed without error in the *Renderer* view, the computation of positions for the `GridArea` components is possible (see numbers in parenthesis in the *Component Edit* in Figure 3.43).

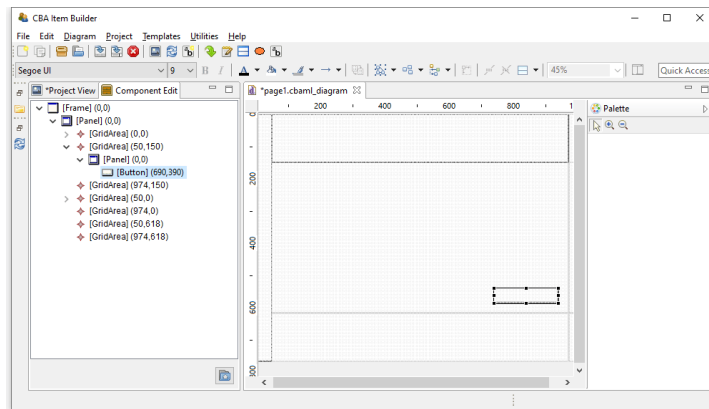


FIGURE 3.43: Dialog *Component Edit* of a page with *Auto Layout Panel*.

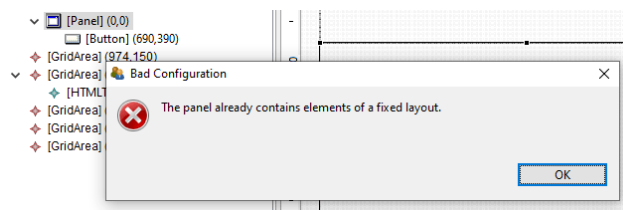


FIGURE 3.44: Message after the attempt to *Set Auto Layout* for a *Panel* that already contains components.

Each *GridArea* (i.e. each cell in a *Panel* with *Auto Layout*) may contain only one child component. If several components are to be inserted, a *Panel* is always necessary first, which may then contain several components. If content of a *GridArea* is duplicated (see section 3.7.2 for a description of the way to *Duplicate* content), the item definition becomes invalid (see Figure 3.45). This mis-configuration can be fixed by moving the duplicated content to a still empty *GridArea* in the *Page Editor*.

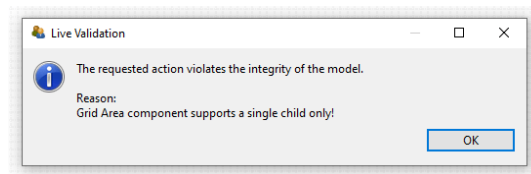


FIGURE 3.45: Message after *Duplicate* was called on the content of a *GridArea*.

3.5.4 Embedded Pages in PageAreas

An alternative approach to increase page similarity is to embed and re-use pages multiple times. Pages can be embedded into pages using `PageAreas` as shown in Figure 3.145. In addition to `Panels` also `PageAreas` can be placed within `Frames` of `Simple Pages`. Components of type `PageAreas` (and similarly also components of the type `ChildArea` and `WebChildArea`) are placeholders that define the location and size of embedded page's content in the frame-component of a page.

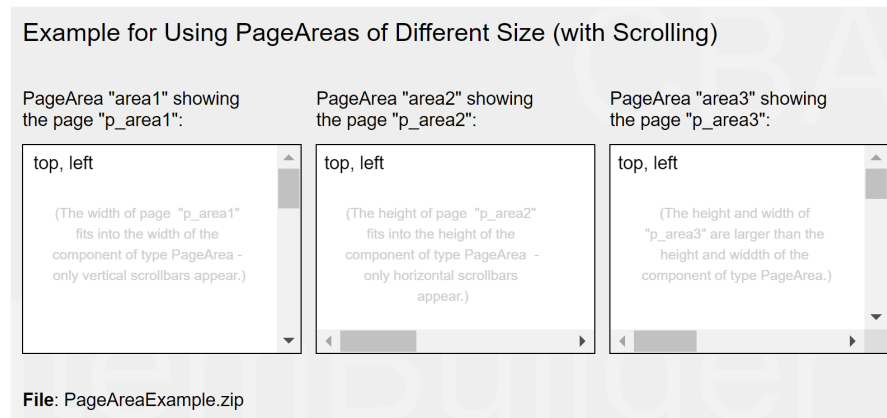



FIGURE 3.46: Item illustrating scrollbars of `PageAreas` ([html](#)|[ib](#)).

Multiple `PageAreas` are possible on one page, as shown in Figure 3.46. The size of the visible area (*View Port*) corresponds to the size of the `PageArea` as defined in the *Properties* view with the properties `Width` and `Height` and the location of the `PageArea` is defined using the properties `x` and `y`.

Use of Embedded Pages: Inserting pages allows a variety of possibilities that can only be listed here. The spectrum of possibilities ranges from the implementation of scrollable regions (as shown in Figure 3.46), over the reuse of page compositions as illustrated in Figure 3.47 to the dynamical display of additional content (see, for instance, section 6.4.4).

Definition of Embedded Pages: Using components of type `PageArea` (and analogously also `ChildArea` and `WebChildArea`) requires two steps: In the first step the `PageArea` is inserted and placed in the *Page Editor*. Since `PageAreas` can only be added to containers of type `Frame`, the `Frame` must first be selected for insertion in the *Page Editor*, so that the icon for `PageAreas` is displayed in the *Palette* ( `PageArea`). Note that as illustrated in the video embedded in Figure 3.48, a `Panel` that already exists within the `Frame` must be resized before the `Frame` can be selected.

After selecting the icon `PageAreas` in the *Palette*, use the mouse to draw a rectangle in the empty area of the *Page Editor*. Although `PageAreas` can only be inserted into

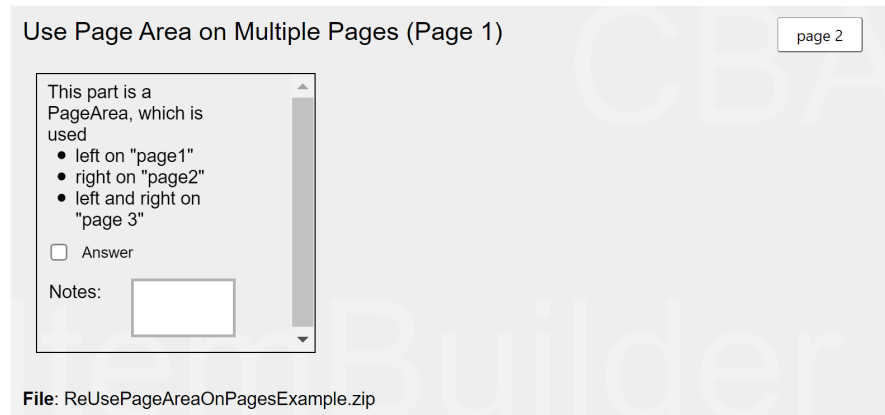


FIGURE 3.47: Item illustrating how to use content in PageAreas multiple times ([html](#)|[ib](#)).

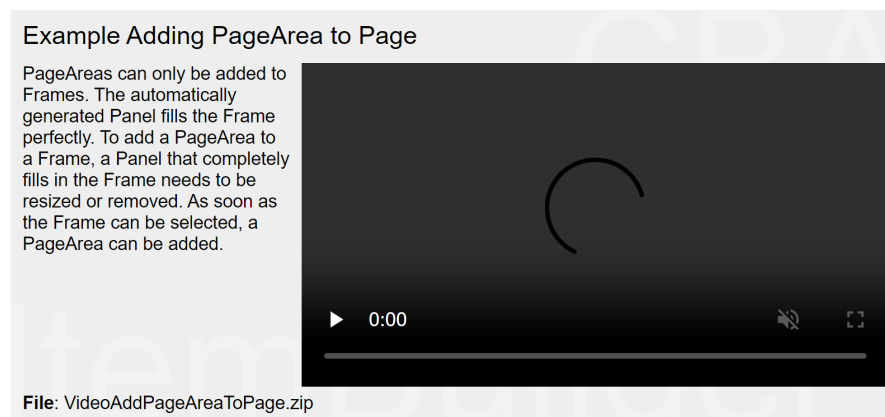


FIGURE 3.48: Item illustrating how to add a PageArea if a Panel fills the Frame completely ([html](#)|[ib](#)).

components of type `Frame`, an additional `Panel` can be used to design the page as a whole. Position and size of `PageAreas` can be adjusted via the *Properties* view.

Once a `PageArea` has been added to the `Frame`, it can be right-clicked in the *Page Editor* to bring up the context menu shown in Figure 3.49.

In the context menu of `PageAreas` the entry *Link Embedded Page* can be used to open the dialog shown in Figure 3.50. The dialog lists all pages that have been defined in the current *Project File* and which can be embedded in the current context in the `PageAreas`. To embed a page it has to be selected in the list and the dialog has to be closed with `OK`.

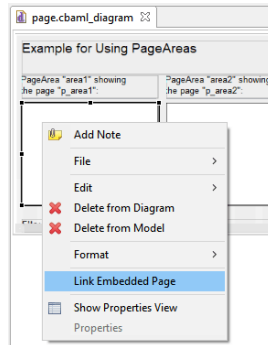


FIGURE 3.49: Context menu for components of type `PageArea`.

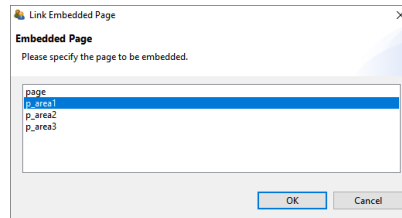


FIGURE 3.50: *Link Embedded Page* dialog for `PageAreas`.

An initial assignment of an embedded page is necessary, even if the assignment is then changed with conditional links (see section 4.3.4) or in the finite-state machine (see section 4.4.6).

User Defined IDs and `PageAreas`: With the help of `PageAreas` pages can be used multiple times in assessment components created with the CBA ItemBuilder. It is therefore necessary that each `PageArea` has its own unique naming.



The `UserDefinedId` of `PageAreas` must be defined (see section 3.7.4) to be able to address page content embedded with `PageAreas` uniquely. Note that in all syntax components of the CBA ItemBuilder (see section 4.1) all references to content of `PageAreas` must include the `UserDefinedId` of the `PageArea` (see section 4.1.4).

CBA Presentation Size and `PageAreas`: After calling the preview, the CBA ItemBuilder checks if the defined pages are larger than the specified CBA Presentation Size (see section 3.6.2). If this is the case, a warning message is displayed, as shown in figure 3.51. When using `PageAreas`, scrollbars appear automatically, so the warning can be ignored (or de-activated in the *Global Properties*, see section 6.3).

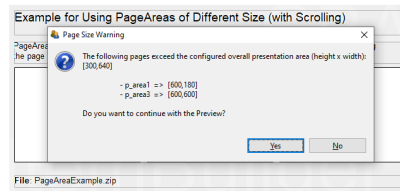


FIGURE 3.51: Warning when pages are larger than the defined *CBA Presentation Size*.

3.6 Tasks as Entry Points



After the required pages are defined, the definition of a so-called *Task* is necessary. *Tasks* fulfill different functions in CBA ItemBuilder projects. Most importantly, tasks are the *entry points* to an item for the *Preview* (see section 1.4) and when the item is used in a *test assembly* (see chapter 7).




During the preparation of assessment components with the CBA ItemBuilder, the content is distributed to different *Tasks*. *Tasks* can be viewed directly with the built-in *Preview* (see section 1.4.2) and can be understood as the basic building blocks for complex assessment designs (e.g., booklets, rotations etc.).

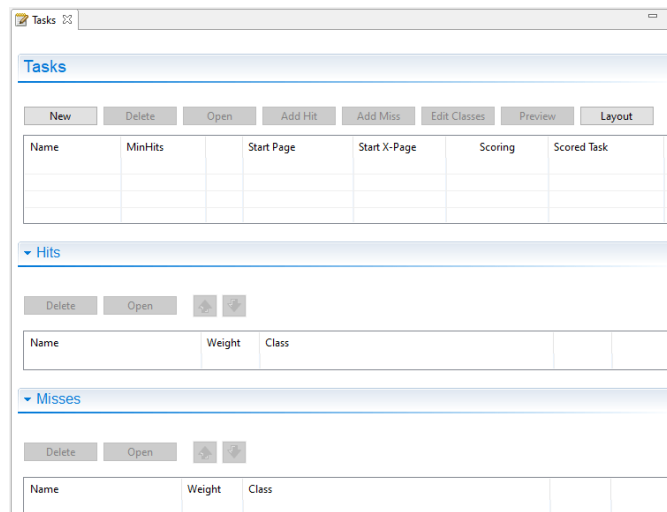
As an entry point, the *Task* specifies which page should be used as *Start Page*, and if specified in the *Layout Settings*, which *xPage* should be used as *Start xPage*. By defining one of the existing pages as the *Start Page*, the *Task* also specifies the page type of page. Beyond this function, *Tasks* are also used to structure scoring within CBA ItemBuilder projects. Since existing pages and item content can be used differently in different tasks, scoring rules are defined per task (see chapter 5 for details about the scoring of CBA ItemBuilder tasks).

3.6.1 Task Definition in the Tasks-View





To use a CBA ItemBuilder project as assessment component, at least the definition of one *Task* is mandatory as entry point.

The necessary steps and an overview of the central options to configure tasks are described in the following. Using the menu `Project > Browse Tasks` or with the toolbar icon , the *Tasks View* in the right panel of the CBA ItemBuilder can be opened. If no task is defined yet, it should look as shown in Figure 3.52.





| Name | MinHits | Start Page | Start X-Page | Scoring | Scored Task |
|------|---------|------------|--------------|---------|-------------|
| | | | | | |
| | | | | | |
| | | | | | |

Hits

Delete Open  

| Name | Weight | Class |
|------|--------|-------|
| | | |

Misses

Delete Open  

| Name | Weight | Class |
|------|--------|-------|
| | | |

FIGURE 3.52: Empty *Tasks* - view of the CBA ItemBuilder.

A new *Task* can be created with the `New` button and existing *Tasks* can be edited in the *Tasks* editor by changing the entries in the particular row. The button `Delete` removes the *Task* in the row selected in the *Tasks* editor.

Task Name: After creating a new task via the `New` button, the default task name can be changed in the first column (`Name`). Task names can only contain selected characters: Letters, digits and underscores are allowed and the task name must start with a letter.¹⁵

MinHits: The minimum number of hits (default 1) can be entered in the second column (`MinHits`) and a positive number is expected.

First Page: Each task must define the first page (i.e., the page that is loaded first,

¹⁵These restrictions can also be presented as a regular expression (see section 6.1.1).

when the task is shown). To define the first page, select an existing page from the list of pages. A completely defined task (with the name `Task01` and the start page `page`) should look as shown in Figure 3.53.

| Name | MinH... | Start Page | Start X-Page | Scori... | Scored Task |
|--------|---------|------------|--------------|----------|-------------|
| Task01 | 1 | | page | | |
| | | | | | |
| | | | | | |

FIGURE 3.53: *Tasks* - view showing a complete *Task*-definition.

The green symbol in Figure 3.53 indicates that the task definition is correct and that the specified page is part of the current project. A red symbol would be shown in case of a mis-configuration.¹⁶

If multiple *Tasks* are defined, the order of *Tasks* can be adjusted using the buttons and .

Task Initialization: Only for the implementation of items with dynamic content (see chapter 4) settings can be prepared for the logic layer of the CBA ItemBuilder using a syntax language (see 4.1). If this is necessary, syntax for the *Task Initialization* can be defined using the button `Open`. Afterwards an editor for the currently selected task will open for to entry syntax for the *Task Initialization*. Details on this syntax and on using the *Task Initialization* can be found in the 4.5 subsection in chapter 4.

Task Scoring: The three buttons `Add Hit`, `Add Miss` and `Edit Classes` are provided for defining the scoring for a selected task. The defined *hit* and *mis* conditions are then displayed in the *Tasks*-view. Further information about the scoring capabilities of the CBA ItemBuilder can be found in chapter 5.

The remaining two buttons serve the following functions: With `Preview` the preview can be started for the currently selected task (see 1.4 for details). The `Layout` button can be used to set the layout for defined tasks (see section 3.6.2 below).

3.6.2 Layout Settings for Tasks

A layout can be defined for each defined task. If the layout setting is not changed for a particular task, only one page is displayed at a time by default. The dialog *Execution layout settings* (see Figure 3.54) can be opened in the *Tasks*-view using the button `Layout`. If multiple task are defined in an CBA ItemBuilder project, the settings are applied for the task selected in the list of task in the left part of the dialog.

¹⁶Note that it may be necessary to close the *Task View* once after configuration before it is displayed marked as valid.

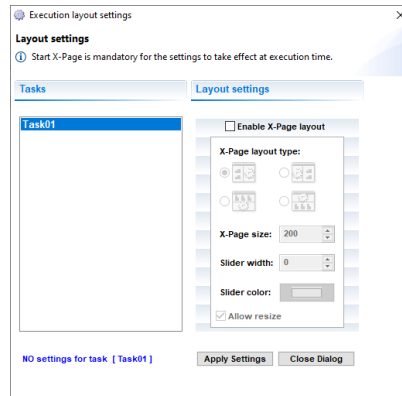


FIGURE 3.54: Dialog *Execution layout settings* to define *xPage*-layouts.

To save the changes, the `Apply Settings` button must be clicked before closing the dialog.

xPage Layout: The so-called *xPage Layout* can be activated using the dialog *Layout setting* (see Figure 3.54).



If the *xPage Layout* is enabled for a task, an *Start xPage* must be defined in the *Task View* together with a *Start Page*. For that purpose, an existing *xPage* must be selected in the particular column of the *Tasks*-table.

As shown in Figure 3.54, the *xPage Layout* allows to specify the *xPage layout* for a task using a checkbox. *xPage* layouts always composed by page and an *xPage*. The different layout types differ regarding the arrangement of these two components. Relative to the regular page the *xPage* can be *left* or *right* (*Horizontal xPage Layout*), *top* or *bottom* (*Vertical xPage Layout*). For *Horizontal xPage Layout* the property *xPage size* refers to the width of the *xPage* (full height as defined in the *CBA Presentation Size*), while for the *Vertical xPage Layout* the *xPage size* refers to the height of the *xPage* (full width as defined in the *CBA Presentation Size*).

The default size of the *xPage Area* can be changed at runtime during test taking if the `Allow resize` option is enabled. To make this possibility visible, a slider can be defined by setting a non-zero value for the 'Slider width' property. Finally, for visible sliders, the color can also be selected in the `Slider color` field.

Figure 3.55 shows an *Horizontal xPage Layout* with activated `Allow resize`-property. As can be seen in the example, scrollbars appear when the *xPage* (here left) or the *Simple Page* (here right) is larger than the visible area. By moving the *Slider* the visible size of the two pages can be changed. As you can see in the example, the size setting is retained when, for example, buttons are used to link to another page (see 3.11.4 for

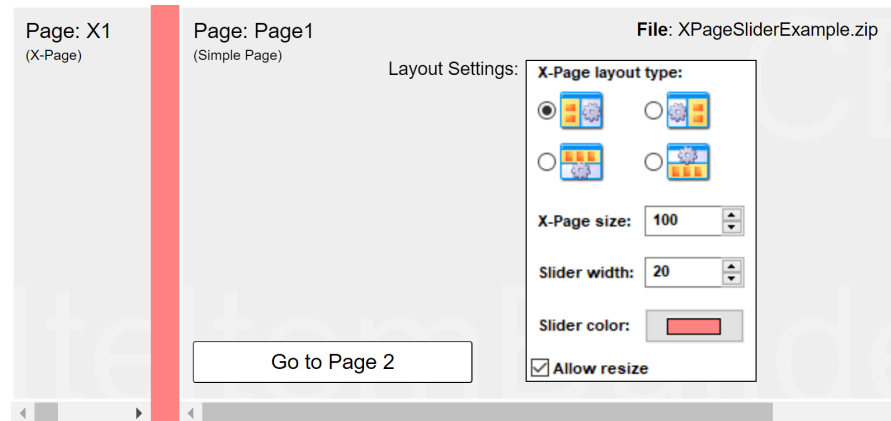






FIGURE 3.55: Item illustrating an *xPage*-layout with enabled slider ([html](#)|[ib](#)).

more details on using *Links* in combination with *xPages*). Table 3.5 summarizes all settings implemented in the CBA ItemBuilder for *xPage*-layouts. Table 3.5 summarizes the possible settings of the dialog *Layout setting* shown in Figure 3.54.

TABLE 3.5: Settings for *xPage* layouts

| Setting | Description |
|--------------------------------|---|
| Enable <i>xPage</i> layout | Checkbox to enable the <i>xPage</i> layout for the selected task. |
| Horizontal <i>xPage</i> layout | Arrangement of the pages in the <i>xPage</i> layout (left:  , right: ) |
| Vertical <i>xPage</i> layout | Arrangement of the pages in the <i>xPage</i> layout (top:  , bottom: ) |
| <i>xPage</i> size | Size of the <i>xPage</i> (width for type left and right, height for type top and bottom). |
| Slider width | Size of the slider, if <code>Allow resize</code> is selected. |
| Slider color | Color of the slider, if the slider size is different from zero. |
| Allow resize | Allow to resize the <i>xPage</i> during runtime. Scrollbars will appear, as soon as the available space is smaller than the size of the <code>Frame</code> . |

CBA Presentation Size in Detail: The *CBA Presentation Size* defines the space on screen, that is used either for a single page or the combination of two pages (*xPage-Layout*). The item size for new *Project Files* is defined in the dialog *Preferences* (see section 3.2.2). Once a *Project File* is created, the item size is part of the *Global Properties* of

that project. To edit the *Global Properties*, right-click on the project name and select *Global Properties* as shown in the Figure 3.56.

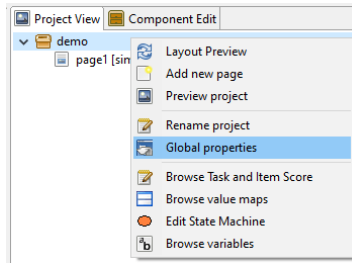


FIGURE 3.56: Entry *Global Properties* in the *Context Menu* of the project name in the *Project View*.

The entry *Global Properties* allows to edit the *Project Settings*, including the item size (*Presentation height* and *Presentation width*) for an existing CBA ItemBuilder *Project File* (see Figure 3.57).

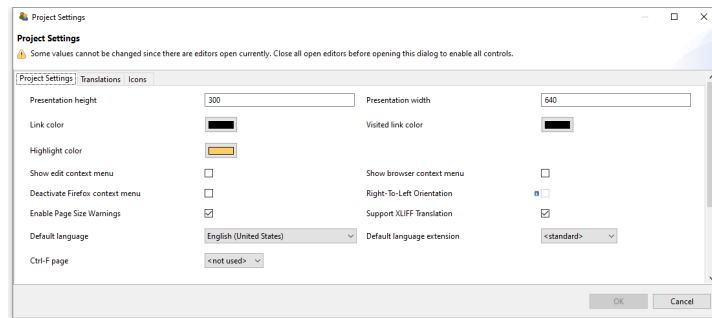


FIGURE 3.57: *Project Settings* to change the *CBA Presentation Size*.

The *CBA Presentation Size* and the size of pages defined as `width` and `height` property of the top-level container used as frame (see section 3.5.1) must fit each other so that no scrollbars appear. The critical factor here is whether an *xPage Layout* has been defined.

- *No xPage Layout*: The required size of an item results from the width of the `Frame` and the `Border Width` of the `Frame`. Accordingly, the *CBA Presentation Size* must match the *Frame Size* plus two times the `Border Width` to avoid scrollbars (see upper part of Figure 3.58).
- *xPage Layout*: If an *xPage* layout is enabled, then the *CBA Presentation Size* must be

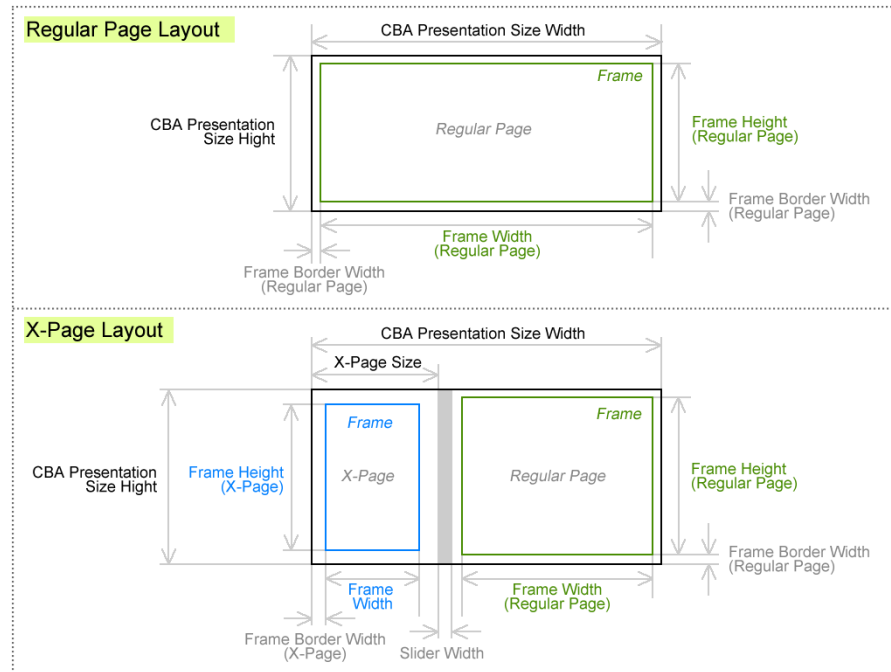


FIGURE 3.58: Schema for components of the *CBA Presentation Size*.

sufficient to span the `Frame` of the regular page and the `Frame` of the *xPage*. In addition, two times the `Border Width` of the page, two times the `Border Width` of the *xPage* and the width of the slider (if `Allow resize` is selected) must be taken into account (see lower part in Figure 3.58).



The size of a page is equal to the size of the `Frame` plus two times the width of the border (`Border Width` property of the `Frame`).

If a non-zero width of the slider (`Slider width`, see Figure 3.54) is set in an *xPage* layout, then the available size for the regular page is reduced.

3.6.3 Navigation Witin and Between Tasks

The CBA ItemBuilder provides with the concept of *Tasks* a way to structure assessment components into parts. *Tasks* allow the organization of assessments through the use of one or more CBA ItemBuilder *Project Files*.

Tasks as Entities: *Tasks* represent the smallest entity that must remain together in

test deployments. In this sense, *Tasks* provide the *Entry Points* into assessment components created with the CBA ItemBuilder, consisting of one or more pages. A *Task* can contain a non-interactive page that provides information or instruction only, a page with a single item, a page filled with multiple items, or even a complete unit or an entire test. Each *Task* is part of a CBA ItemBuilder *Project File*, and *Project Files* must contain at least one *Task*, but can also provide many *Tasks*.



The navigation *within Tasks* can be freely designed by the item author. Navigation *between Tasks* is done by the execution environment.

The assembly of assessment components into tests is handled by the software used for test delivery (see Chapter 7).

Multiple Tasks in the Preview : If several *Tasks* are defined within a *Project File*, then a linear sequence of *Tasks* can be viewed in the *Preview* of the CBA ItemBuilder. The navigation from one *Task* to the next or previous *Task* is triggered with so-called *Runtime Commands* (see section 3.12). Figure 3.59 illustrates the navigation in a linear sequence of *Tasks* using different layouts.

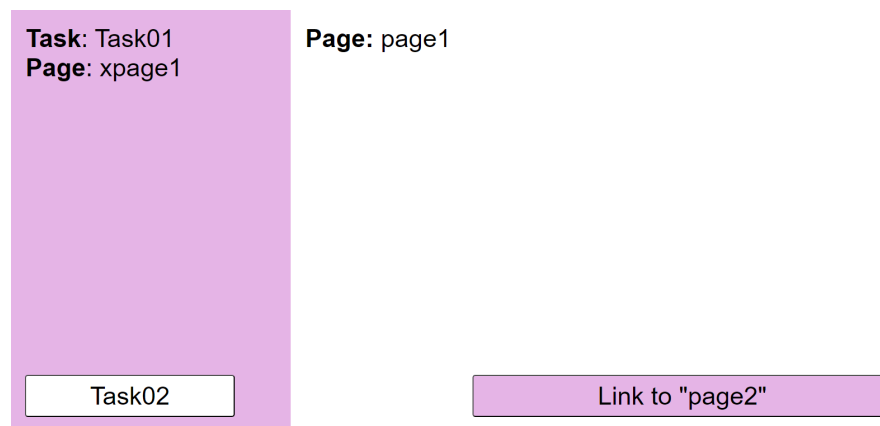


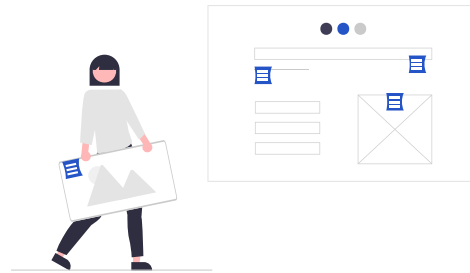
FIGURE 3.59: Example for navigation with *Runtime Commands* ([html](#)|[ib](#)).

The order in which *Tasks* of a *Project File* are displayed in the *Preview* corresponds to the order in the *Task Editor*.

Test Deployment Software: A list of CBA ItemBuilder *Project Files* with the *Tasks* as entry point then forms the *Item Pool*, which can be used for different test designs. If the *Tasks* are administered in a specific order one after the other, a *fixed form test* is the result. When several different sequences are combined into rotations, the result is an *booklet design*. If single or multiple *tasks* are selected depending on previous answers, forms of *adaptive testing* can be implemented.



Division of Content to Tasks: Using multiple *Tasks* within one *Project Files* allows re-using designed pages but requires additional considerations to ensure that the *Tasks* are independent. Dependencies can arise based on links (see section 3.11) and conditional links (see section 4.3), runtime commands (see section 3.12), the finite-state machine (see section 4.4), FSM variables (see section 4.2), and task initialization (see section 4.5). The issue of splitting content between different *Tasks* and different *Project Files* is discussed in section 8.2).

3.7 Layout Pages using Components



Pages are designed in the CBA ItemBuilder *Page Editor* within components of types *Frame* (see section 3.5.1) and a *Panel* (see section 3.5.2). Page content is usually added to *Panels* by selecting a particular component from the *Palette* and drawing a rectangular in the *Drawing Area* of *Page Editor*. Before the different components for designing static content are described in detail, this section first contains general notes that are intended to make working with the CBA ItemBuilder easier.

Adding Components: Before a component can be added to a *Page*, the *Page* must be opened first. To add a new element to the *Drawing Area* of an opened *Page*, first select the component *into* which the new element should be included. If a *Page* was created with *Frame* and *Panel* (the default), select the *Panel* as the container. After the container is selected, the *Palette* shows all components that can be added to this container. Select the component from the *Palette* (the *Palette* will only show entries that can be selected for the current selected element). Afterwards, draw a rectangle in the main area (click into the *Drawing Area* within the component that should contain the new component, hold the left mouse button pressed, move the mouse so that a small rectangle appears and release the mouse button at the opposite corner of rectangle).

Undo/Redo Actions: Changes in the *Page Editor* and in the *Properties* view can be undone with `Ctrl+Z` (or `Strg+Z`, ) respectively and redone with `Ctrl+Y` (or `Strg+Y`, ) , see also Figure 3.60).

If the *Project File* is saved, previous editing steps cannot be undone.

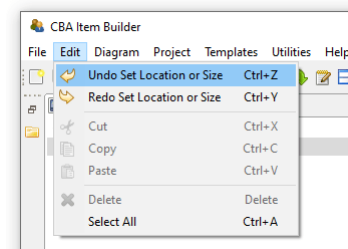


FIGURE 3.60: Main Menu Edit showing Undo and Redo.




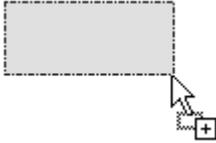
If it is configured in the settings, that a project is saved with each preview (see section 1.4.2), no changes can be undone afterwards.


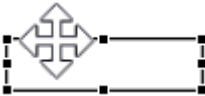
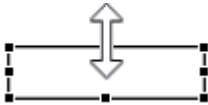




3.7.1 Positioning of Components

Within the *Drawing Area* of the *Page Editor* components can be positioned freely and precisely defined by integer-values used as coordinates.

Visual Editing: The CBA ItemBuilder allows to move components in the *Page Editor* for visual editing with the pointing device (e.g., mouse) and keyboard. To do this, the components must be clicked with the left mouse button and moved with the mouse button pressed. Moving components is only possible if, on the one hand, the text property of the component is not in edit mode (see Table 3.6). Components can be moved and re-sized using the anchor-points in the *Page Editor*. If the *Snap To Grid* feature is activated (*Rulers & Grid* in the *Properties* view described in section 3.1.4), the components will snap to the defined grid. Press the **Alt** key to move components freely. Selected components can also be moved using the arrow keys.

TABLE 3.6: Visual feedback (mouse pointer) in the *Page Editor*.

| Visualization | Description |
|---|--|
|  | The component selected in the <i>Palette</i> can be added by left-click in the <i>Page Editor</i> and moving the mouse to draw a rectangle, before releasing the mouse. |
|  | Visualization of adding a component (left mouse-button pressed). The size of the component is defined as the gray rectangle. Releasing the mouse to finally add the component to the page. |

| Visualization | Description |
|---|---|
|  | Visualization of the information, that the selected component cannot be placed into the current context. Select a different component or draw the component into the appropriate container to continue. |
|  | Component can be moved while holding the left mouse button pressed. Use the ALT-key if <i>Snap To Grid</i> is activated. |
|  | Height of the component can be changed by clicking and moving the mouse while holding the left mouse button pressed. |
|  | Width of the component can be changed by clicking and moving the mouse while holding the left mouse button pressed. |
|  | The Width and Height of the component can be changed by clicking and moving the mouse while holding the left mouse button pressed. |
|  | Lasso used to select one or multiple components. Click on an empty place in the <i>Drawing Area</i> of the <i>Page Editor</i> and move the mouse while keeping the left mouse button pressed to use this feature. |
|  | Edit mode of the text property is activated. Select another component in the <i>Page Editor</i> (or use the ESC-key) to close this mode. |

Positioning Using the Property View: Components selected in the *Page Editor* can also be moved by changing the values of the coordinates *x* and *y* as well as the values for *Width* and *Height* in the *Properties* view (see Figure 3.61 for an example).

| ▼ Position | |
|------------|-----|
| Height | 241 |
| Width | 613 |
| X | 12 |
| Y | 24 |

FIGURE 3.61: Section Position of the *Properties* view.



The left upper corner of all containers (`Frames`, `Panels`, `RadioButtonGroups`, ...) is defined as $x: 0$ and $y: 0$. Accordingly, increasing the x -coordinate of a component will move it to the right, and increasing the y -coordinate of a component will move it down.

In the *Page Editor*, scrollbars are displayed when components stick out beyond their container's boundaries. Such a case may occur when negative x or y coordinates are configured or when the total size (relative to x and y) exceeds the container's size.



Note that scrollbars in the *Page Editor* appear if a nested component perfectly fits into its container (i.e., if `Width` or `Height` of a nested component are identical to `Width` or `Height` of the container). Use the *Preview* to verify that the final layout matches the expectations.

For editing components in the *Page Editor*, it can be helpful to change the zoom setting of the *Page Editor*. Suppose components are perfectly overlapping or exactly nested. In that case, components can be selected using the *Component Edit* view (see section 3.1.2). After selecting a component in the *Component Edit*, it is possible to adjust the coordinates in the *Properties* view so that they can be selected and edited in the *Page Editor* again.

Group Components using Panels: Components of type `Panels` can be used for different purposes. Firstly, at least one `Panel` is required within the 'Frame' from a technical perspective. Secondly, `Panels` can be used for layout purposes, for instance to create space needed for margins, to show a border (see section 3.1.4) or a background image (see section 3.10.2). However, `Panels` can also be used to group components. x and y -coordinates of components nested within a `Panel` are defined relative to the left upper corner of the `Panel`.



To group components that have already been placed on a page together into a `Panel`, the components can be dragged and dropped in the *Page Editor* into the `Panel`.

3.7.2 Clipboard and Duplication of Components

One of the challenges for new users of the CBA ItemBuilder is the use of the clipboard. The *Page Editor* of the CBA ItemBuilder uses components of different types. Since components of a specific type can only embed other components of particular types, using the clipboard requires some comments.

Duplicate: Within pages, components can be selected and duplicated via the context

menu. To select multiple components, *Lasso-selection* feature of the *Page Editor* can be used (see Table 3.6). Similarly, components that are grouped, for instance, within a *Panel* (see section 3.7.1) can be duplicated together. Figure 3.62 shows the context menu for a selection of multiple elements.

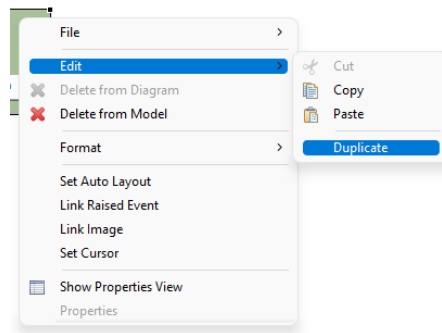


FIGURE 3.62: Context menu for a selection in the *Page Editor* showing the entry *Duplicate*.

The function to duplicate components is equivalent to executing `ctrl+c` and `ctrl+v` (or `strg+c` and `strg+v`) directly one after the other. The duplicated elements are added again in the same context and moved by 10 pixel in both *x* and *y*. If components are grouped in a *Panel*, they can be easily aligned together after duplication. Otherwise, the duplicated components must be moved individually to the final location.

View Clipboard: Single components with or without nested components can be copied, using the `ctrl+c`, the context menu entry *Edit > Copy* or the main menu entry *Edit > Copy*. The copy operation is available, after the component was selected in the *Page Editor*.



To support the use of the clipboard while maintaining the required nesting of components, the CBA ItemBuilder provides a specific *View Clipboard*.

Elements that are added to the clipboard are shown in the *View Clipboard* (see Figure 3.63), together with the information, which components in the *Page Editor* can be selected when requesting a *Paste* operation (`ctrl+v` or *Edit > Paste*). Multiple components can be used in the clipboard, and the clipboard remains active even after closing and opening a different project file.

Note that if the components are selected in the *Component Edit* (see subsection 3.1.2, it will be impossible to copy components until the components were selected (again) in the *Page Editor*.

Multiple Components: Duplicating elements with the context menu entry shown in

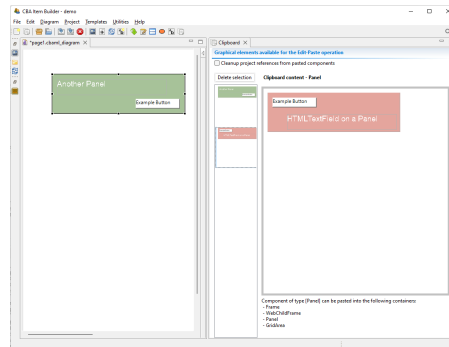


FIGURE 3.63: View Clipboard illustrating the visualization of elements in the clipboard.

Figure 3.62 is can handle multiple components at once, but is available only within pages. To copy multiple components across pages using the clipboard, components need to be nested into a single `Panel`. Note however, that if identical content is required on several pages, the concept of `PageArea` (see section 3.5.4) is often useful. If a complete page needs to be duplicated, this can also be achieved using *Page Templates* (see section 6.8.7).

Delete Components: The CBA ItemBuilder does not implement the *Cut* operation (i.e., `Ctrl+X`). To cut a selected component, copy the component first and use the context menu entry `Delete from Model` to delete the component, after it was added to the clipboard.



Remark: The clipboard of the CBA ItemBuilder will change in the next version (10.0). CBA ItemBuilder will allow to cut and paste components (including the nested components) maintaining the user-defined IDs.

3.7.3 Selecting Components using the *Component View*

In the *Page Editor* of the CBA ItemBuilder pages are designed within a component of type `Frame`. For this purpose, the components are selected in the *Palette* of the graphical *Page Editor* and then added within *Containers*. Within the *Page Editor* components are visualized with rectangles at the position defined with the `x` and `y` coordinate and with a size that reflects the height and width of components. Position and size are either defined in the *Properties* view, the result of visual editing (i.e., using drag and drop in the *Page Editor*) or are computed by the CBA ItemBuilder (if the `Layout Type=GRID` is used in *Auto-Layout-Panels*, see subsection 3.5.3).

The various possibilities can all result in components being perfectly nested, making it impossible to select the underlying component in the *Page Editor*.



Components can always be selected using the *Component View* (i.e., even elements perfectly nested into their parents can be picked).

An example of the *Component View* is shown in Figure 3.64.

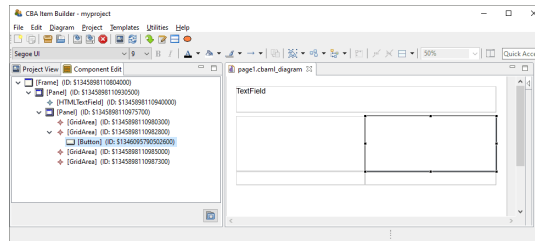


FIGURE 3.64: *Component Edit* view allows to select any component in the *Page Editor*.

To allow the selection of all components defined in a page, the CBA ItemBuilder provides the *Component View*. In the *Component View* (see figure 3.64) all components can be seen in their hierarchical structure and can be selected. If a component is not seen, then in the *Component View* the tree can be expanded and the components can be shown. The selection of components in the *Component View* corresponds to the selection of components in the *Page Editor* and also allows access to the *Context Menu* as well as the *Properties View*.

3.7.4 Naming Components with `UserDefinedIds`

Components that are used to create and design pages in the *Drawing Area* of the *Page Editor* have an automatic generated string identifier (i.e., a generated number that uniquely identifies the particular instance of a component). This automatic generated identifier can be overwritten so that components have more meaningful, human-readable identifiers.

Importance of `UserDefinedIds`: `UserDefinedIds` are provided and assigned by the item author. For that purpose, each component that can be selected in the *Page Editor* provides the property `UserDefinedId` in the section *Identification* of the *Properties* view. `UserDefinedIs` are used to refer to the components, for instance, for the definition of conditional links (see section 4.3), to change properties of components using the *Finite-State Machine* (see section 4.4), and to specify scoring-rules a *Task* (see chapter 5). `UserDefinedIds` also plays a central role with regard to the interpretation of log and process data, since log events as an attribute refer to the components from which they originated via the `UserDefinedId` (see section 1.6).



UserDefinedIds are human-readable identifiers for components that are used to connect the visual part (*Page Editor*) with the syntax components of the CBA Item-Builder.

The CBA ItemBuilder automatically generates IDs when saving *Project Files*, which start with a \$ character, followed by a unique random number. When naming the component, i.e. when item authors as users def IDs (UserDefinedIds), these automatically generated identifiers must be replaced.

Valid UserDefinedIds: Since the UserDefinedIds are used to create source code (see section 2.11), the following conventions and rules must be followed:¹⁷

- Only letters, digits and underscores (_) are allowed as characters.
- Each UserDefinedId must start with a letter and it is not allowed to use a digit or underscore as the first character.
- UserDefinedIds are case sensitive and upper and lower case letters need to be distinguished.
- No spaces and blanks are allowed in UserDefinedIds.

To edit a UserDefinedIds for a component selected in the *Page Editor*, change the entry of the same name in the *Properties* view (see Figure 3.65).

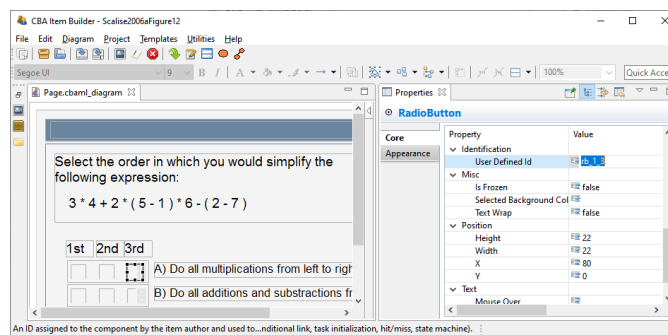


FIGURE 3.65: *Properties* view section Identification to define a UserDefinedId.



UserDefinedIds are entered in the *Identification* section of the *Properties* view. The CBA ItemBuilder checks the validity and uniqueness of entered UserDefinedIds.

¹⁷These restrictions can also be presented as a regular expression (see section 6.1.1).

The CBA ItemBuilder automatically checks whether the entered `UserDefinedId` is already used for another component in the current CBA ItemBuilder project and provides an error message, if the `UserDefinedId` is already used or invalid, as shown in Figure 3.66.

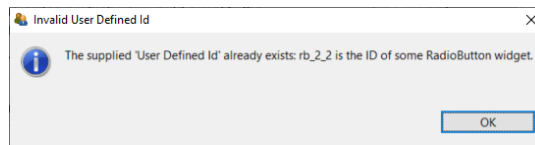


FIGURE 3.66: Message when a provided User Defined Id is not unique.

If a component with a `UserDefinedId` is deleted, this `UserDefinedId` can only be assigned again after saving the current *Project Files*.

Edit Multiple `UserDefinedIds`: Since providing `UserDefinedIds` of many components via the graphical editor can be very time-consuming, the CBA ItemBuilder provides a dialog that allows the `UserDefinedIds` of several components to be entered in tabular form. For using this feature all other editors must first be closed (see Figure 3.67).

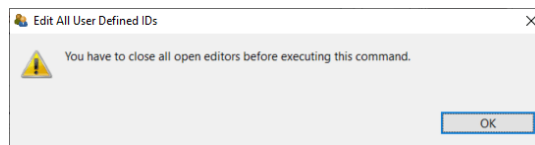


FIGURE 3.67: Dialog informing requesting to close all editors before the requested operation can be executed.

If there are no editors open in the main area of the CBA ItemBuilder, the entry *Edit all user defined IDs* can be opened via the main menu *Project* for an open *Project File* (see Figure 3.68).

The editor shown in Figure 3.69 lists all elements together with the assigned `UserDefinedIds`. Rows with generated `Defined IDs` are shown in gray. Using this dialog `UserDefinedIds` can be edited for a particular component shown as row in the table using the button *Edit* (or double-click the row). To be able to assign the individual elements without the *Page Editor*, the component type and selected properties are also displayed in this dialog.

By clicking on a column in the table header, the elements can be sorted in the *Edit User Defined IDs* dialog shown in Figure 3.69.



Since the `UserDefinedId` is used for many different functions to connect components to syntax, it is recommended in larger CBA ItemBuilder projects to use a

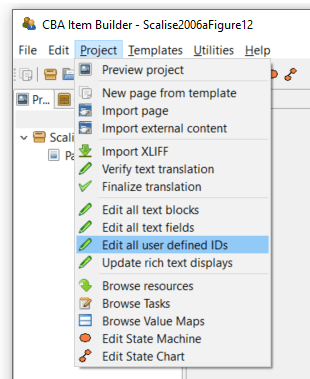


FIGURE 3.68: Main Menu Project showing the entry Edit all User Defined IDs.

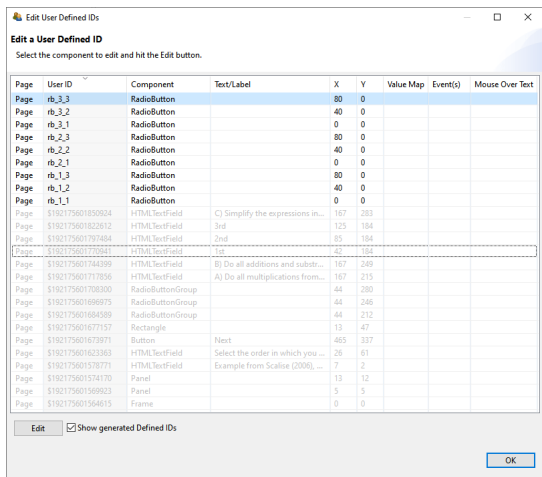


FIGURE 3.69: Dialog to Edit User Defined ID of a Project File.

consistent naming scheme for all UserDefinedIds . A systematic naming schemes for the UserDefinedIds can reduce the error-proneness of scoring definitions for complex items (see chapter 5).

3.7.5 Design Pages with Basic Components

The visual design of assessment components with the CBA ItemBuilder is mainly done by using images (and images can be used with several components (see section 3.10.2)).

Transparent Background: Transparent backgrounds of components can be defined with the property `Transparent=true`. In this case, the components do not hide components behind, which can, for instance, be designed using a component of type `Panel` with a linked (background) image.



Components (including components for displaying images, see section 6.2.1) and components for embedding external HTML content (`ExternalPageFrames`, see section 3.14.1) can be configured to be transparent, allowing to apply designs based on images behind components.

Components can be defined as `Transparent=true` in the section *Display* of the *Properties* view. As described in the 6.2.1 section, the CBA ItemBuilder also supports transparency within images.

Borders and Background in Components: Components to collect responses, for instance, components for entering text responses (see section 3.9.1) should be easily to recognize by test-takers. Accordingly, they should be formatted and marked consistently. Background color or a border can be used to emphasize components, such as `SingleLineInputFields` and `InputFields`. The `Border Width` property must be set to a positive, non-zero width to define a visual border.

The frame color can be configured in the *Appearance* area of the *Properties* view using a color selection dialog (see section 3.1.4).

Rectangles and Lines: Beyond images, the CBA ItemBuilder provides `Rectangles`, `Horizontal Lines` and `Vertical Lines` as components to design items graphically. These components are also inserted with `x`, `y`, `Width` and `Height` in the *Page Editor*. However, for components of type `Line`, the `Width` and `Height` is only relevant within the *Page Editor*. The `Line Width` property is used as a positive number to define the visual width of the line during runtime. Similarly, for components of type `Rectangle` also the `Line Width` property is used.

When components of type `Rectangle` are inserted late (i.e., when the item content is already added to the page) to create visual boundaries by frames, the rectangles are placed at the top of the editing order. In the *Page Editor*, it is then no longer possible to select underlying elements directly, but they must be selected with the help of the *Component Edit* (see section 3.1.2). Alternatively, the order in the *Page Editor* can also be adjusted using the context menu as shown in Figure 3.71. If the order has been changed (i.e., if a component of type `Rectangle` has been moved to the background in the editing order with `Format > Order > Send to Back`), then underlying components can be selected in the *Page Editor*.

It should be noted that the editing order does not influence the display order and the behavior of overlapping components at runtime (i.e., in the preview or the delivery of assessment composites created with the CBA ItemBuilder, see section 2.11.4). Moreover, components of type `Rectangular` can block the interaction with underlying components and prevent processing of assigned events (see section 6.8.5).

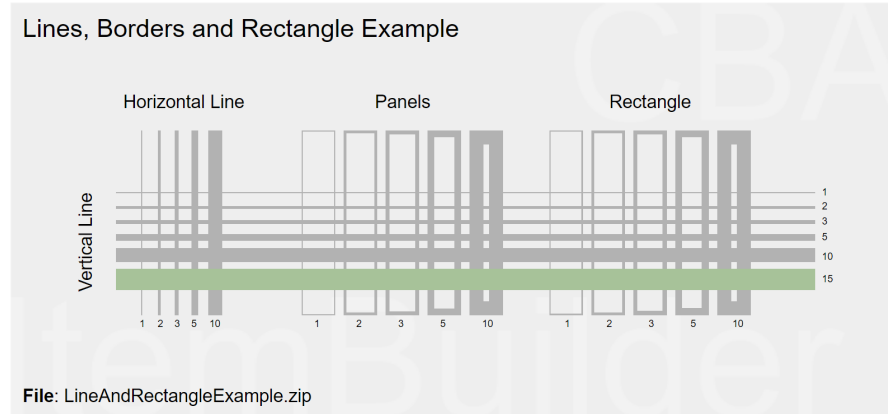


FIGURE 3.70: Item illustrating components of type `Line` and `Rectangle` ([html|ib](#)).

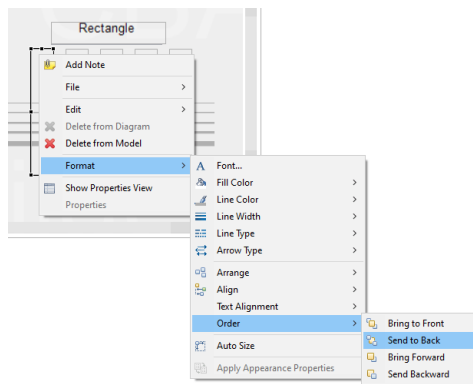


FIGURE 3.71: Context menu for design-time Z-order in the *Page Editor*.

3.7.6 Defining the Cursor of Components

When components are displayed at runtime, the cursor (i.e., the mouse pointer) often hints at what can be done with that component. For example, a text input cursor (`/text/`) appears for elements into which text can be entered. Each component has a default cursor. As shown in Figure 3.72, the default cursor can be overridden in the CBA ItemBuilder.

To change the cursor for a component selected in the *Page Editor*, the entry `Set Cursor` can be selected in the context menu (see Figure 3.73).

In the dialog *Set Cursor* (see Figure 3.74) you can either select an element from the list of cursors, use an image imported into the ItemBuilder project via the *Resource*

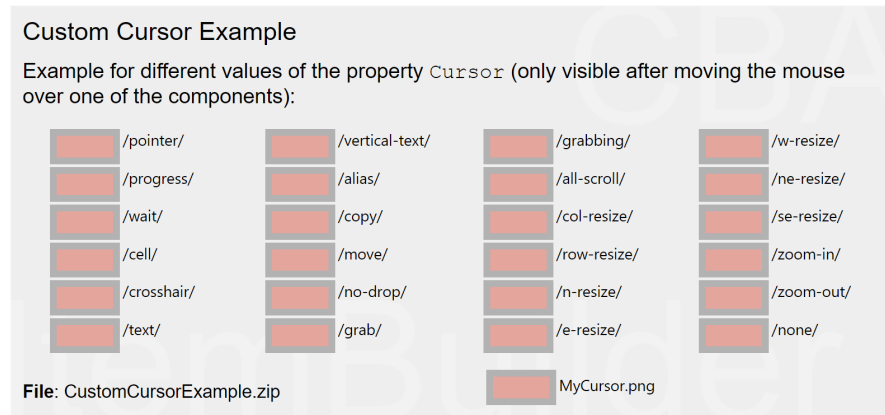


FIGURE 3.72: Item illustrating the use of custom `Cursors` (<http://html5lib.org>).

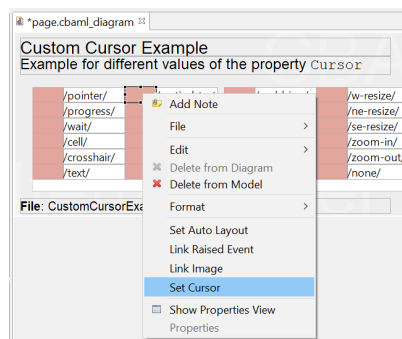


FIGURE 3.73: Context menu for defining the `Cursor` in the *Page Editor*.

Browser (see section 3.10.1) or use the default cursor. If no cursor is to be displayed, then the entry `/none/` can be selected as *Cursor Type*.

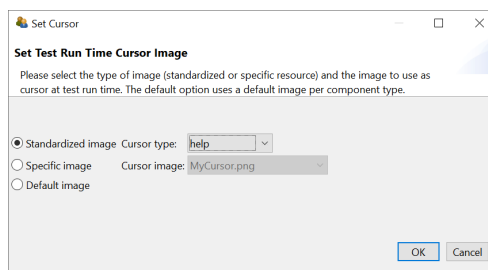


FIGURE 3.74: Dialog *Set Cursor*.

3.7.7 Defining the Tab-Order of Components

The `Tab Index` property in the *Properties*-view can be used to specify the order in which the input focus switches between components. Switching from one component to the next is possible with the *Tab*, if a positive integer number is defined for the `Tab Index` property.

The key combination *Shift+Tab* can be used to switch the focus back to the previous component (see Figure 3.75). The default value for the ‘*Tab Index*’ property is `-1`, and components with a `Tab Index` of `-1` are not included in the tab sequence.

Tab Index Example

| Component | Tab Index |
|---------------------------------------|-----------|
| <input type="text"/> | 1 |
| <input type="button" value="Button"/> | 2 |
| <input type="checkbox"/> Checkbox | 3 |
| <input type="text"/> | -1 |

File: TabIndexExample.zip

The `Tab Index` property can be used to specify the order in which the input focus switches between your components when **Tab** or **Shift+Tab** is pressed.

Components with a `Tab Index` of `-1` are not included in the tab sequence.

FIGURE 3.75: Item illustrating the use of `Tab Index` ([html|ib](#)).

3.8 Components to Display Text



This section describes individual components for displaying text on pages designed

with the CBA ItemBuilder. The components shown in Figure 3.76 can be added to containers of type Panel (on Simple Pages and Webchild Pages)¹⁸.

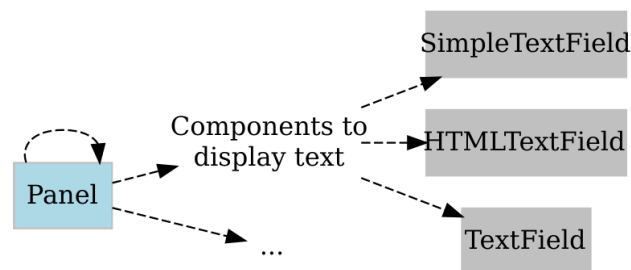


FIGURE 3.76: Overview of components to display text.

For adding static text to pages, the CBA ItemBuilder offers various components that differ in terms of design options and technical implementation. For all components for displaying text, you can restrict the list of available fonts, which is created in the CBA ItemBuilder based on the fonts installed on the system, to the fonts you require (see 6.8.2).

3.8.1 Text of Same Size: SimpleTextField

Text which should be displayed in a uniform font (i.e. the same font, font size and font color) can be added to a page using the SimpleTextField component. The configuration of the text is done by double-clicking on a SimpleTextField (or via the Edit Text entry in the context menu or the property Text in the Properties view).

The text defined for a component of type SimpleTextField can be single line or multi-line and the text is rendered with a vertical scroll bar if the text given the defined font size is larger than the defined size of the component. Not only the text can be modified using the property Text, also the font, the font size and the font color can be edited in the Appearance section of the Properties view (see Figure 3.77, and subsection 3.1.4).

Components of type SimpleTextField support the following values for the property Alignment in the section Appearance of the Properties view to configure the text alignment: LEFT, RIGHT, and CENTER. The item in Figure 3.78 illustrates the different components to display text.

Clicks on SimpleTextFields during assessments are logged as Trace events of type SimpleTextField.

¹⁸Pages of type WebBrowser Page allow to add HTMLTextField and TextField in containers of type WebBrowserToolbar.

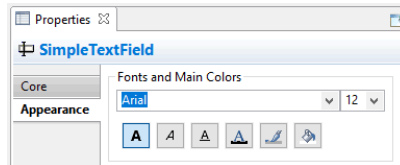


FIGURE 3.77: Section *Appearance* in the *Properties* view to define *Fonts and Main Colors*.

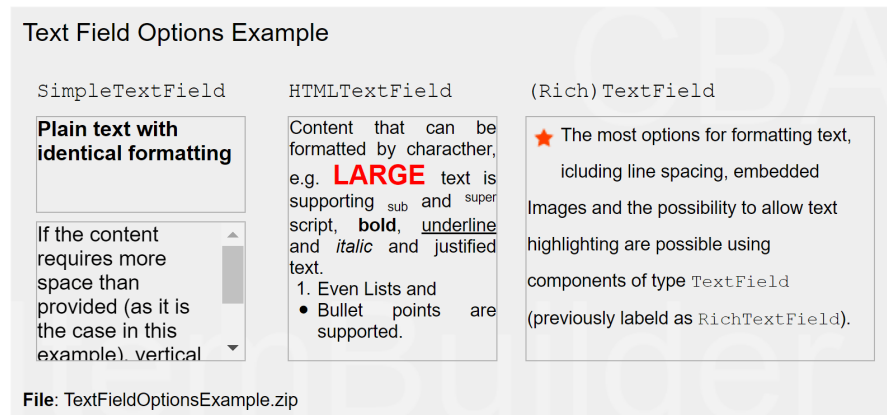


FIGURE 3.78: Example illustrating components to display text ([html](#)|[ib](#)).

SimpleTextFields as Input Source: Components of type `SimpleTextField` can also be connected to a source (using the context menu entry *Configure Input Source* or the property *Input Source* in section *Component Interaction* of the *Properties* view), for example, to display the current page in a simulated web browser or the result of a calculation with the built-in calculator engine. To configure the *Input Source* of a `SimpleTextField` provide the context menu entry *Configure Input Source* to open the *Input Source Configuration Editor* (see Figure 3.79).

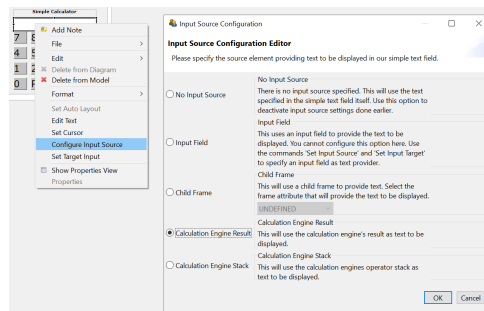


FIGURE 3.79: *Input Source Configuration Editor* to define the *Input Source* of `SimpleTextFields`.

The different options to specify the *Input Source* of a `SimpleTextField` are illustrated in the item shown in Figure 3.80.

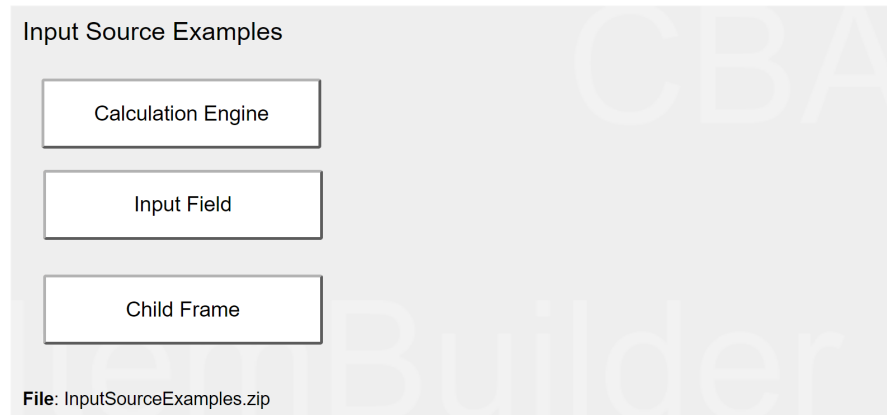


FIGURE 3.80: Item illustrating options to define *Input Source* for `SimpleTextFields` ([html](#)|[ib](#)).

The default is *No Input Source*, meaning that the `SimpleTextField` shows the text configured in the property `Text`. With the option *Input Field* components of type `SimpleTextField` can mirror the input provided into a `InputField`. To link both components internally, first select the option *Input Field* for the `SimpleTextField` using the dialog *Input Source Configuration* (see Figure 3.79). The configuration is completed after selecting the context menu entry *Set Source Input* at first for the `SimpleTextField` that should take the text entry (see left part in Figure 3.81) and then the context menu entry *Set Target Input* (see right part in Figure 3.81).

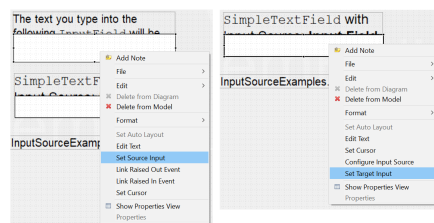


FIGURE 3.81: Context menu entries to link components with the option *Input Field*.

The option *Child Frame* configures `SimpleTextFields` shows either the *URL Text*, the *Page Description* or the *Tab Text* of a web child page (see section 3.13.2). Finally, the options *Calculation Engine Result* and *Calculation Engine Stack* can be used to link the two outputs of the calculation engine to a `SimpleTextField` (see section 6.5.1).

3.8.2 Formatted Text: HTMLTextField

Two additional components are provided to display text that can have different text formatting and multiple paragraphs. The `HTMLTextField`, which is described in this section, and the `TextField`, which is discussed in the next section.

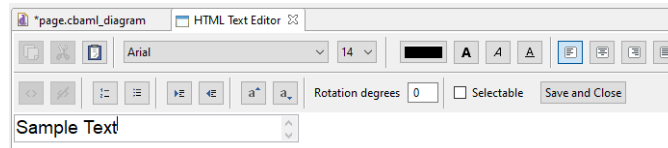


FIGURE 3.82: *HTML Text Editor* for the content of `HTMLTextField` components.




HTML Text Editor: Various options for formatting text are provided by the `HTMLTextField` by the *HTML Text Editor*.



- Font *family* (see section 6.8.2 for more information about the available font families) and font *size* (defined in `Images` and `Multimedia Components` pixels) can be specified for the current selection of text in the *HTML Text Editor*.
- Font *color* (see section 6.8.3 for more information about the definition of colors) and font decoration *bold*, *italic* and *underline* can be assigned to the current selection of text in the *HTML Text Editor*.
- Text *alignment* (can be defined for each paragraph using the icons for left, center, right and justify, see note on justify text in section 3.8.4) is supported.
- Rotation (defined in degrees from 0 to 360) can be used to rotate the entire content specified in an `HTMLTextField`.
- Superscript (a^{super}) and subscript (a_{sub}) can be formatted for selected texts.
- Enumerations (numbered) and listings (bullet points) are supported.
- Paragraph indentation can be controlled with in `HTMLTextFields`.



The visual representation of `HTMLTextFields` in the *Page Editor* of the CBA Item-Builder is only provided to give item authors a rough orientation. This text is updated only when the *HTML Text Editor* is closed and is not re-created when the size of an `HTMLTextField` is changed. The *Preview* or the *Renderer* view allow to check the exact display of text in the final item.

Save or Discard: Changes in the *HTML Text Editor* require to save the new text using the `Save` and `Close` button (see Figure 3.82). Closing the editor via the small cross (✖) in the title bar discards the changes.

Clipboard: To simplify the editing of longer texts in `HTMLTextFields`, the CBA ItemBuilder provides functions to copy, cut and paste text. These functions work in addition to the usual keyboard shortcuts:   .

Links: Beyond the formatting of text, individual sections of text in `HTMLTextFields` can be defined as embedded links, which can then refer to other pages within the CBA ItemBuilder project (see 3.11 for a description of the concept of linking between pages in the CBA ItemBuilder:  ). The color used for links and visited links can be configured in the *Project Settings* (see section 6.3). As shown in the dialog in Figure 3.83, conditions for links can also be defined (see section 4.3 for details on *Conditional Links*).

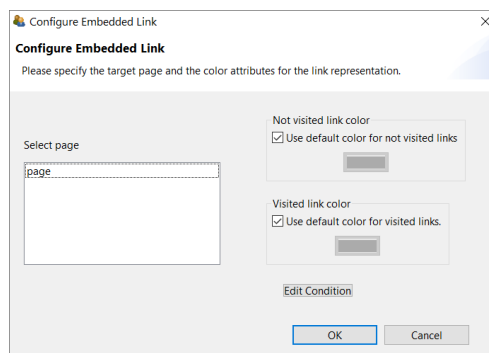


FIGURE 3.83: Dialog to *Configure Embedded Link* in `HTMLTextFields`.

3.8.3 Formated Text and Highlighting (`TextField`)

A second component for displaying differently formatted text is the `TextField`, which among other things supports *highlighting*.

Highlighting of Text: The CBA ItemBuilder supports the selection of characters and words as response format, labeled as *Highlighting*. Text highlighting is defined for components of type `TextFields` using the option `Highlightable`.

The following item in Figure 3.84 illustrates the *text highlighting* feature of the CBA ItemBuilder inspired by a PIAAC example (see OECD, 2013).

Note that the buttons *Green* and *Red* illustrate the used of multiple colors for text highlighting (see subsection 4.4.6 for details, the default color can be configured as *Global Property*, see section 6.3). More details on the use of *text highlighting* as response format (and, in particular, scoring of *text highlighting*) can be found in section

Highlighting Example

Text Highlighting (from PIAAC)

Look at the list of preschool rules. Highlight information in the list to answer the question below.

What is the latest time that the children should arrive at preschool?

Preschool Rules

Welcome to our Preschool! We are looking forward to a great year of fun, learning and getting to know each other. Please take a moment to review our preschool rules.

- Please have your child here by 9:00 am.
- Bring a small blanket or pillow and/or a small soft toy for naptime.
- Dress your child comfortably and bring a change of clothing.
- Please no jewelry or candy. If your child has a birthday please talk to your child's teacher about a special snack for the children.
- Please bring your child fully dressed, no pajamas.
- Please sign in with your full signature. This is a licensing regulation. Thank you.
- Breakfast will be served until 7:30 am.
- Medications have to be in original, labeled containers and must be signed into the medication sheet located in each classroom.
- If you have any questions, please talk to your classroom teacher or to Ms. Marlene or Ms. Tree.

File: HighlightingExample.zip

FIGURE 3.84: Example illustrating the use of text highlighting ([html](#)|[ib](#)).

5.3.8. Using components of type `ImageAreas` within so-called `ImageMaps` (see section 3.9.10) also allows the selection of predefined parts within images.

Mathematical Formulas: The `TextField` component allows to add formulas using the [MathJax](#)-syntax, as shown Figure 3.85.

To display a formula using `MathJax` in components of type `TextField`, the formulas must be entered between the two character keys `{tex}` and `{/tex}` in LaTeX format. The following fraction $\frac{1}{2}$ is created by the syntax:

```
{tex}\sqrt{\frac{1}{\frac{1}{2}}}{/tex}
```

3.8.4 Comparison of Components for Displaying Text

This section has described various components that can be used to represent text in assessment components and items. This surely leads to the question of which

MathJax Display in TextFields Example

$a \neq 0$

`{tex}a \neq 0{/tex}`

$x^n + y^n = z^n$

`{tex} x^n + y^n = z^n{/tex}`

$\sqrt{\frac{1}{\frac{1}{2}}}$

`{tex}\sqrt{\frac{1}{\frac{1}{2}}}{/tex}`

$2.5 \cdot 3 = ?$

`{tex}2.5 \cdot 3 = ?{/tex}`

File: MathJaxExample.zip

(See www.mathjax.org for more information on MathJax formulas).

FIGURE 3.85: Example illustrating the use of text MathJax ([html](#)|[ib](#)).

component should be used in which specific context. The answer to when to use a `SimpleTextField`, an `HTMLTextField` or a `TextField` depends on the required features. Table 3.7 summarizes the features of the different components to guide the selection.

TABLE 3.7: Comparison of components to display text

| Feature | HTMLTextField | TextField | SimpleTextField |
|--------------------------------|---------------|-----------|-----------------|
| Allows text highlighting | no | yes | no |
| Shows scrollbars automatically | no | no | yes |
| Allows justified text | yes | yes | no |
| Can be selectable | yes | no | no |
| Allows links | yes | yes | no |
| Allows images | no | yes | no |
| Allows formulas | no | yes | no |
| Can be rotated | yes | no | no |

Components of type `TextField` are also used in the `Table` component of the CBA Item-Builder (see subsection 3.9.8).

Notes on using justified text: If text is formatted as justified text, this formatting is only displayed at runtime (i.e. in the *Preview* and in the *Renderer* view) in the current version of the CBA ItemBuilder. Double blanks in justified text can cause problems and should be avoided. Justification only works for paragraphs, i.e. the text that is to be displayed block-justified must not contain any line breaks within paragraphs, as shown in the right part of Figure 3.86.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

FIGURE 3.86: Illustration of line breaks in text.

Other Components: Components of type `Link` can also display text (see section

3.9 Components to Collect Responses



The CBA ItemBuilder provides a variety of components that can be used to capture answers to questions and items. This section presents the various simple components that can be used for directly for designing static content (see Figure 3.87 for an overview).

Basic Components: First, two components for capturing *text responses* are described (see section 3.9.1). After that, components that can be used to capture *click responses* are described. Creating *multiple-choice* or *single-choice* response formats is possible, for instance, with components of type `RadioButton` with (and without) `RadioButtonGroups` (see section 3.9.2). *Multiple-Choice* items can also be created, for instance, using ‘Checkboxes’ (see section 3.9.3). Single and multiple selection of presented response alternatives is also possible with the `ComboBox` (single-choice) and `List` components (single- and multiple-choice, see section 3.9.5). Finally, selection within images can be implemented with components of type `ImageMap` to collect responses (section 3.9.10).

Advanced Components: Beyond text and click responses shown in Figure 3.87, the CBA ItemBuilder provides components that can be used to create advanced response formats (see, for instance, section 3.13 for components of type `Table`, `Menu` and `Tree`, that can be combined with simple click response-formats). Moreover, selectable components (see section 3.9.6) can be used to collect click responses. Finally, the combination of multiple pages connected via links and conditional links, *FSM Variables*, and *Finite-State Machines* with simple and advanced response-formats allows creating complex items that create traces of task processing that can be used for advanced scoring schemes.

3.9.1 Components for Text Responses

Components for capturing text responses are simply input fields in which text can be entered when they are focused. To focus an input field the test taker must click

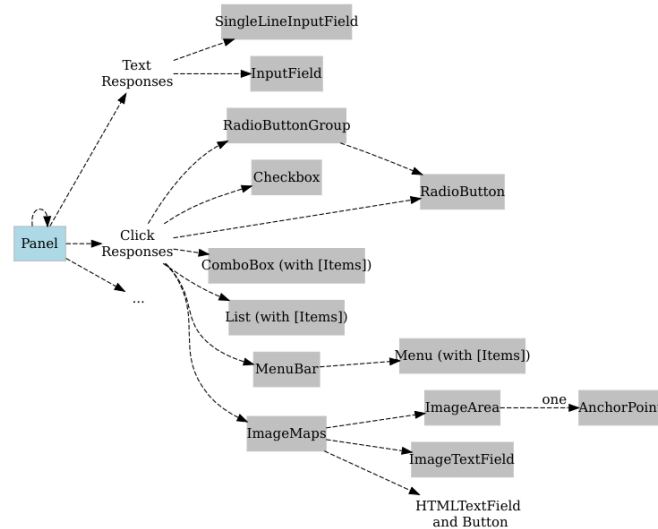


FIGURE 3.87: Overview of components to collect responses.

in the input field, or the focus is set automatically with the `focus()`-operator (see section 4.4.6).

As shown in Figure 3.87, the CBA ItemBuilder distinguishes mainly two components for capturing text responses.

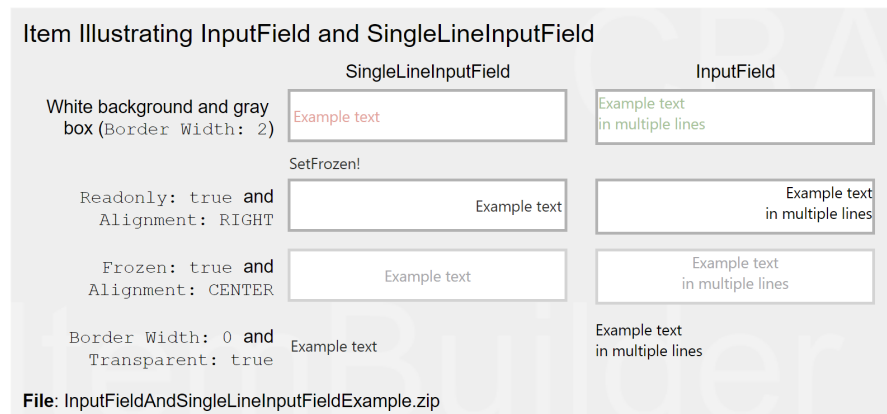


FIGURE 3.88: Example illustrating components to collect text responses ([html|ib](#)).

Single Line Text Input (SingleLineInputField): The component for single-line text in-

put is called `SingleLineInputField`. Line breaks (with the *Enter* or *Return* key) are not accepted by `SingleLineInputField` (i.e., only one single line can be used).

Multiline Text Input (`InputField`): For multiline text input, the CBA ItemBuilder provides the `InputField` component. Line breaks (with the *Enter* or *Return* key) are possible and scrollbars appear, if the text height becomes larger than the height of the `InputField` component.

Properties of `SingleLineInputField` and `InputField`: Both components can be configured and used in identical ways (see Figure 3.88). `SingleLineInputField` and `InputField` can be used within `Panels` on `Simple Pages` and `WebChild Pages`. `SingleLineInputField` can also be used within `WebBrowserToolbars` on `WebBrowser Pages`.



A border or background color must be configured to recognize components of type `SingleLineInputField` and `InputField` as elements of an item or page that allows text input (see last row in Figure 3.88).

Font size and font family can be configured in the tab *Appearance* in the *Properties* view (see Figure 3.5 in section 3.1.4). Components of type `InputField` and `SingleLineInputField` support the following values for the property *Alignment* in the section *Appearance* of the *Properties* view to configure the text alignment: `LEFT`, `RIGHT`, and `CENTER`.

Components of type `SingleLineInputFields` and `InputFields` support the *Read Only* property. If the value `true` is specified in the *Properties* view in section *Misc*, the text cannot be changed. The components then behave like `SimpleTextFields` (see section 3.8.1) with the difference, that the text can be selected.

All response capture components support the *Frozen* property, which locks the components for editing. Figure 3.88 shows how type `SingleLineInputFields` and `InputFields` components are displayed when *Is Frozen: true* is configured in the *Misc* section of the *Properties* view. The *Frozen*-property can also be modified using the operators `setFrozen()` / `unsetFrozen()` in *Conditional Links* and transitions triggered in the *Finite-State Machine* (see section 4.4.6).

Text color for `InputField` can be defined in the tab *Appearance* of the *Properties* view. By default the *Link Color* as defined in the in the *Global Properties* of a CBA ItemBuilder project file (see section 6.3) is used for `SingleLineInputFields`. To use the text color specified in the tab *Appearance* of the *Properties* view, the property *Use Default Link Color* needs to be specified as `false`. Note that the property *Use Default Link Color* might be invisible, until the filter *Show Advanced Properties* as shown in Figure 3.89 is clicked.

The length of the text that can be entered, as well as the characters that can be used, can be configured using regular expressions in the *Misc* section of the *Properties* view via the *Input Validation Pattern* property (see section 6.1.3 for examples).

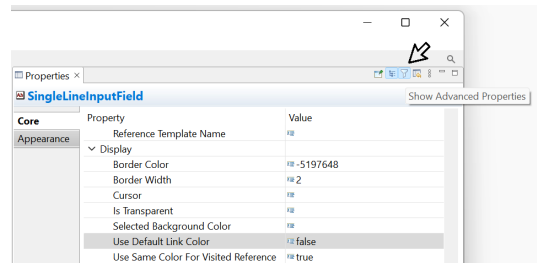


FIGURE 3.89: Filter Show Advanced Properties of the *Properties* view.

The two components for collecting text responses can raise *FSM events* (see section 4.4.3) when the text field receives input focus (Raised In Event) and when input focus is removed from the text field (Raised Out Event). If inputs are not accepted due to a regular expression, the Input Validation Event is triggered. The *FSM events* can be linked to the components using the context menu entries Link Raised In Event, Link Raised Out Event, and Link Input Validation Event.



The CBA ItemBuilder can be used to score *text-entry* tasks using regular expressions (see section 6.1.2), resulting in a score variable that can indicate if an expected answer was provided. The raw response to one or multiple `SingleLineInputFields` or `InputFields` can be copied to an additional variable using the `result_text()`-operator (see section 5.3.10 for details).

Components of type `SingleLineInputFields` and `InputFields` create *Trace events* of type `SingleLineInputField` and `InputFields` (when the input field is focused) and `SingleLineInputFieldModified` and `InputFieldModified` (when the text is changed), that are stored in the log data.¹⁹

3.9.2 Single-Choice Responses: `RadioButtonGroup` and `RadioButtons`

Even for simple item formats like single-choice and multiple-choice the item design (e.g., the affordance rendered by different item designs) can impact psychometric properties. This was, for instance, investigated by Moon et al. (2019, see Figure 3.90).

The creation of *single-choice* tasks is often done in computer-based assessment with components of the type `RadioButton`, while *multiple-choice* is usually implemented using `Checkboxes` (see subsection 3.9.3).

¹⁹Note that the current version of the CBA ItemBuilder is not logging every keystroke with time stamp and cursor position. If this information is required for your analyses plans, you have to implement text entry with advanced logging using JavaScript and `ExternalPageFrames`.

Different item formats can be created using the generic components of checkbox and radio buttons (in groups).

Examples from Moon et al. (2019):

NFC

FC

MSMC

DK

APO

Non-Forced-Choice (NFC)

Indicate wheather each property in the table is true for all rhombuses, all isosceles trapezoids, both all rhombuses and all isosceles trapezoids, or heither. Select all cells that apply.

| Property | True for all rhombuses | True for all isosceles trapezoids |
|---------------------------------------|--------------------------|-----------------------------------|
| Diagonals bisect each other | <input type="checkbox"/> | <input type="checkbox"/> |
| Diagonals are congruent to each other | <input type="checkbox"/> | <input type="checkbox"/> |
| All sides are congruent | <input type="checkbox"/> | <input type="checkbox"/> |
| The sum of interior angles is 360° | <input type="checkbox"/> | <input type="checkbox"/> |


Moon, J. A., Kaehner, M., & Katz, I. R. (2019). Affordances of Item Formats and Their Effects on Test-Taker Cognition under Uncertainty. *Educational Measurement: Issues and Practice*, 38(1), 54–62. <https://doi.org/10.1111/emip.12229>

FIGURE 3.90: Figure 1 from Moon et al. (2019) illustrating different item formats ([html](#)|[ib](#)).



Usually, the representation of components to provide click-responses are either round (called `RadioButtons` for single-choice) or rectangular (called `Checkbox` for multiple-choice). The CBA ItemBuilder follows this distinction, but also provides *Frame Select Groups* to define the behavior in more detail (see section 3.9.4).

As seen in Figure 3.87, `RadioButtons` in the CBA ItemBuilder are often not placed directly on pages within `Panels`. Instead, to insert `RadioButtons` in the *Page Editor* of a page, a component of type `RadioButtonGroup` can be used as container to create a group of related `RadioButtons`.

To add a `RadioButtonGroup` first be selected in the *Palette* ( `RadioButtonGroup`) and drawn in the *Drawing Area* (and then, if necessary, positioned and adjusted using the coordinates `x` and `y` as well as the properties `Width` and `Height` in the *Properties* view). The item shown in Figure 3.91 illustrates the use of `RadioButtons` within `RadioButtonGroups` and the properties `IsTransparent` and `BorderWidth`.

`RadioButtonGroup`: `RaidoButtonGroups` perform two functions. `RadioButtonGroups` are indispensable when used as response format n the CBA ItemBuilder to organize

| RadioButton-Examples illustrating the properties 'Border Width' and 'Is Transparent'. | | RadioButtonGroup | | RadioButton | |
|---|--|------------------|----------------|--------------|----------------|
| | | Border Width | Is Transparent | Border Width | Is Transparent |
| <input checked="" type="radio"/> Option A <input type="radio"/> Option B <input type="radio"/> Option C | | 2 | false | 2 | false |
| <input type="radio"/> Option A <input type="radio"/> Option B <input type="radio"/> Option C | | 2 | false | 0 | false |
| <input type="radio"/> Option A <input type="radio"/> Option B <input type="radio"/> Option C | | 2 | true | 0 | false |
| <input type="radio"/> Option A <input type="radio"/> Option B <input type="radio"/> Option C | | 2 | true | 0 | true |
| <input type="radio"/> Option A <input type="radio"/> Option B <input type="radio"/> Option C | | 0 | true | 0 | true |

File: RadioButtonExample.zip Set all RadioButtons frozen!

FIGURE 3.91: Item illustrating RadioButtons ([html|ib](#)).

the membership of RadioButtons to groups. All RadioButtons that are nested in a RadioButtonGroup form a common group. Exactly one RadioButton per group can be selected. By default (i.e., before a RadioButton of a group is clicked), no RadioButton is selected. Accordingly, a missing answers can be identified reliably. Once a RadioButton is selected, this selection can only be changed, but not undone. The second function of RadioButtonGroups concerns the graphical design. Using Border Width and background color for RadioButtonGroups with the property Is Transparent: false, RadioButtonGroups can be used to visualize how RadioButtons belong together. Alternatively, additional components can be used to design pages (see section 3.7.5), for example, to insert a background image. RadioButtonGroup can be added to Panels on Simple Pages and WebChild Pages.

RadioButton: The actual component for the *single choice* response format is RadioButtons. Each RadioButton represents a selectable option, which is represented with a label text (and an optional image). After selecting a RadioButtonGroup in the *Drawing Area* of the *Page Editor* only the icon RadioButton (🗉 **RadioButton**) can be selected in the *Palette*. RadioButtons can only be added to RadioButtonGroups. Once the icon RadioButton is selected in the *Palette*, a RadioButton can be added to the RadioButtonGroup. This is done by clicking inside of the RadioButtonGroup, followed by a mouse-move with pressed left mouse button. Use the zoom feature (see section 3.1.1), if the RadioButtonGroup is small. Coordinates *x* and *y* can be changed in section *Position* of the *Properties* view, but coordinates are specified relate to the origin (i.e., the upper left corner) of the RadioButtonGroup.

Properties of RadioButton: The size of RadioButton defined using Width and Height property must include the size for the label, that will appear for right-to-left languages right to the RadioButton. The label can be defined using the Text Property of RadioButtons, with font settings configured in the tab *Appearance* of the *Property Grid*. The label can be a single word, a multiline text (using the property Text Wrap: true) or

a simple linked images, as shown in Figure 3.112 in section 3.10. If the property `Use status image: true` in the section *Misc* of the *Property Grid* is configured, two images (*Activated Image* and *Pressed Image*) can be used to distinguish two images as label for *RadioButtons*.

RadioButtons can have *Background Color* (if `Is Transparent:false`) and border with nonzero *Border Width*. Using the context menu entry *Set Selected Background Color* (or entering a color number directly into the property *Selected Background Color* in section *Misc* of the *Property Grid*, see section 6.8.3 for more information about CBA ItemBuilder color codes) allows to define the background color used when a particular *RadioButton* is the selected (single) choice.



The *Text Property* provided by *RadioButtons* results in labels for *Single-Choice* response formats that are click-sensitive. If necessary, *Conditional Links* or the *Finite-State Machine* can be used to make additional components (such as *HTMLTextFields*) click-sensitive for *RadioButtons* (see section 6.4.8).

Radiobuttons can trigger *Events* when they are clicked to connect the *static content* with the *dynamic parts* of CBA ItemBuilder *Tasks* that are based on *Finite-State Machines*. For this purpose, a defined *Event* (see section 4.4.3) can be configured via the context menu and the entry *Link Raised Event* or in section *Component Interaction* of the *Properties* view (property *Raised Event*). For advanced uses, a second event name can also be defined only in the *Properties* view for the property *Raised Alternate Event*. While the *Raised Event* is triggered when a *RadioButton* is selected, the second *Event* (*Raised Alternate Event*) is triggered when an already selected *RadioButton* is pressed. From the *Finite-State Machine* (i.e., when a transition is triggered by a *event*, see section 4.4.4) or using a *Conditional Link* (see section 4.3.3), a *RadioButton* can be selected with the `setActive()`-operator (see section 4.4.6 for details).



For scoring *single-choice* tasks, the raw response of a *RadioButtonGroup* corresponds to a variable whose value represents the selected *RadioButton*. In addition, a *Score* variable can be defined, which indicates whether the correct 'RadioButton(s)' has (have) been selected (see section 5.3.2 for details).

Components of type *RadioButtonGroup* create *Trace events* of type *Container* (when clicked). Selecting *Radiobuttons* results in a log event of type *RadioButton* with an attribute `oldSelected` that informs about the previous state of clicked *RadioButton*.

3.9.3 Multiple-Choice Responses: Checkbox

In order to convert *multiple-choice* answer formats with the help of the CBA Item-Builder, the `Checkbox` can be used as a basic component. Checkboxes can be added to Panels, as shown in Figure 3.87. Each `Checkbox` can be selected and deselected.



For scoring *multiple-choice* tasks, the raw response of each `Checkbox` creates its own variable whose value represents the information if that particular `Checkbox` was selected. In addition, a *Score* variable can be defined which indicates whether particular pattern of multiple checkboxes has been selected (see section 5.3.2 for details).

To add a `Checkbox`, first select a `Panel` in the *Drawing Area* of the *Page Editor*. If a `Panel` is selected, the icon `Checkbox` (☒ `Checkbox`) can be selected in the *Palette*. To place the `Checkbox`, click into the `Panel` and move the mouse while keeping the left mouse button pressed.

Properties of Checkboxes: The position of `Checkboxes` can be precisely defined using the section *Position* of the *Properties* view. The size of `Checkboxes` must include the space required for the text label, that is displayed next to the `Checkbox` (on the right side for right-to-left languages). The label text can be defined by double-clicking on the `Checkbox`, using the context menu entry *Edit Text* or directly in the *Properties* view in section *Text*. Font and font size of the `Checkbox` labels can be configured using the *Appearance* tab of the *Properties* view. Analogous to `RadioButtons`, a background color for selected checkboxes can be defined (property *Selected Background Color* or context menu entry *Set Selected Background*) in addition to borders (*nonzero Border Width*) and the background color.



The *Text* Property provided by `Checkboxes` results in labels for *Multiple-Choice* response formats that are click-sensitive. If necessary, *Conditional Links* or the *Finite-State Machine* can be used to make additional components (such as `HTMLTextFields`) click-sensitive for `Checkboxes` (see section 6.4.8).

Components of type `Checkbox` can be linked to two different *Events*. The context menu entry *Link Select Event* allows to assign an *Event* that is triggered, when the `Checkbox` is selected, the entry *Link Deselect Event* can assign an *Event* for deselecting the `Checkbox`.

Components of type `Checkbox` create *Trace events* of type `Checkbox` for the log data, that provide the attribute `oldSelected` to inform about the previous state of the `Checkbox`.

Simple and *Complex Multiple-Choice* response formats or *Multiple-True-False* items can

be created by using components of type `RadioButtonGroups` and `RadioButton` (see section 3.9.2). However, since the use of `RadioButtonGroups` is not possible in combination with *Auto Layout*-Panels (see section 3.5.3), *Frame Select Groups* can also be used to define single- and multiple-choice response formats (see section 3.9.4).

More advanced *Single-Choice* and *Multiple-Choice* response formats can be implemented using the dynamic features of the CBA ItemBuilder (see, for example, section 6.4.9).

3.9.4 Single- or Multiple-Choice using *Frame Select Groups*

Using `RadioButtonGroups` to structure and group `RadioButtons` has a conceptual problem: `RadioButtons` of a group cannot be distributed across different containers, such as `Panels`. This problem is solved by so called *Frame Select Groups*. *Frame Select Groups* are groups defined at the `Frame`-level for `RadioButtons`, `Checkboxes` (and `ToggleButtons`) that specify the membership of these components in a group for click responses. *Frame Select Groups* can be created via the *Configure Select Groups* dialog, which can be called from the context menu of the `Frame` of a page. As shown in figure 3.92, this dialog can be invoked via the *Component Edit* by right-clicking on the `Frame`.

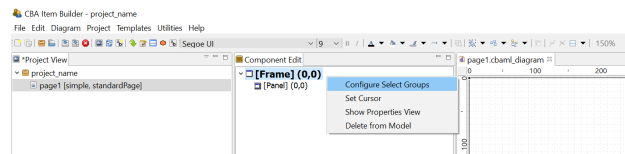


FIGURE 3.92: Context menu to *Configure Select Groups* in the *Component Edit*.

Frame Select Groups can be used together with *Auto Layout*-Panels (see section 3.5.3) to group components of type `Checkbox`, `RadioButton` or `Toggle-Button` into groups. Groups define the three important properties, that define *Single-* or *Multiple-Choice* response formats as groups (*Frame Select Groups*), instead of using `RadioButtonGroups`:

- *Selectable*: Can user interactions (i.e. clicks) on the components change the selection?
- *Multiple Select*: Is it possible to select more than one component of this group?
- *No Deselect*: Is an already selected component de-selected, if it is clicked again?

After right clicking the `Frame` of a page in the *Component Edit* (see subsection 3.1.2) the entry *Configure Select Group* opens the editor shown in Figure 3.93.

Frame Select Groups are addressed by their index. I.e. in each `Frame` multiple groups can be created, and each component is then assigned to a group by entering the index (starting with 0) in the *Properties*-view of a component. In Figure 3.93, 9 differ-

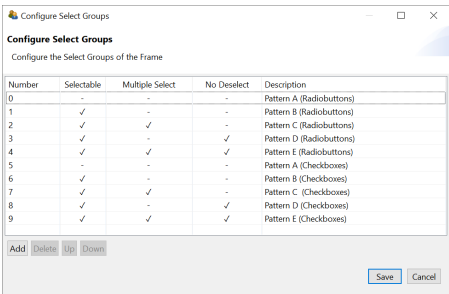


FIGURE 3.93: Configure Select Groups dialog to define Frame Select Groups.

ent groups are defined. All components belonging to each group are assigned to the same index (see the property *Frame Select Group* in section *Misc* in Figure 3.94).

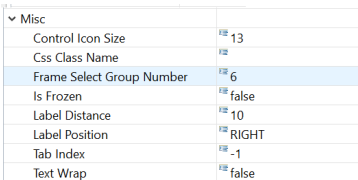


FIGURE 3.94: Assignment of a component to a *Frame Select Group* in the *Properties*-view.

The assignment of components to a previously defined *Frame Select Group* is defined by entering the number in the *Properties*-view, as shown in Figure 3.94. The meaning of the three properties (*Selectable*, *Multiple Select* and *No Deselect*) can be inspected with the example illustrated in Figure 3.95.

| Example for Single and Multple Choice using Select Groups | | | | | | | |
|---|-----------------|------------|-----------------|-------------|--------------------------|--------------------------|--------------------------|
| | Group / Pattern | Selectable | Multiple Select | No Deselect | Example | | |
| Radiobutton (Default is Group 3 / Pattern D) | 0 / A | no | no | no | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |
| | 1 / B | yes | no | no | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |
| | 2 / C | yes | yes | no | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |
| | 3 / D | yes | no | yes | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |
| | 4 / E | yes | yes | yes | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |
| Checkbox (Default is Group 9 / Pattern E) | 5 / A | no | no | no | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| | 6 / B | yes | no | no | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| | 7 / C | yes | yes | no | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| | 8 / D | yes | no | yes | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| | 9 / E | yes | yes | yes | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| File: SingleAndMultipleChoiceUsingSelectGroupsExample.zip | | | | | Buttons/ImageButtons | | |

FIGURE 3.95: Item illustrating the use of *Frame Select Groups* ([html](#)|[ib](#)).

3.9.5 Single- or Multiple-Choice using ComboBoxes and Lists

The CBA ItemBuilder provides two additional components for creating single and multiple selections of existing elements. Both components can be added to Panels (see Figure 3.87). ComboBoxes can be used to implement *Single-Choice* response formats. Only one element can be selected from each *Dropdown*-list. Lists support single- and multiple-choice using the property `Multiple Select Mode`. When closed, ComboBoxes display only the selected element, while Lists show all available elements within the available space. ComboBoxes require scrollbars for the selection of an item if the list of defined `Combobox Items` cannot be shown simultaneously on screen (and the CBA ItemBuilder provides the property `Visible Item Count` to control how many items are displayed during the selection before scrolling is required. With the configuration `Visible Item Count: 0` (default) as many items as possible are displayed). Lists show scrollbars, if the size of the `List`-component is small to show all elements (see Figure 3.96).

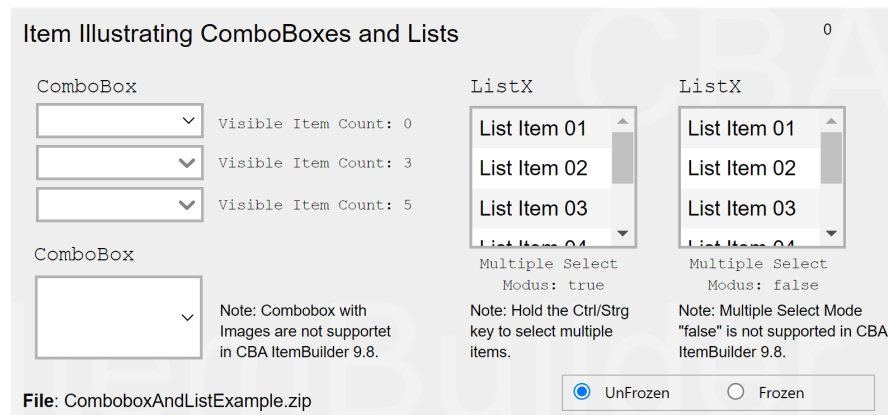


FIGURE 3.96: Item illustrating ComboBoxes and Lists ([html](#)|[ib](#)).

Components of type `ComboBox` and `List` are containers for entries added in the form of `ComboBox Item` (for ComboBoxes) `ListItem` (for Lists).



Entries of `ComboBoxes` (`ComboBox Items`) and `Lists` (`List Items`) are not displayed individually in the *Page Editor* and can only be edited via the *Component Edit* (see section 3.1.2).

To add an item to the `ComboBox`, the context menu (right-click a component of type `ComboBox` in the *Page Editor*) must be used, which contains the entry `Add Combo Box Item` as shown in Figure 3.97 (the analog entry is called `Add List Item` for components of type `List`).

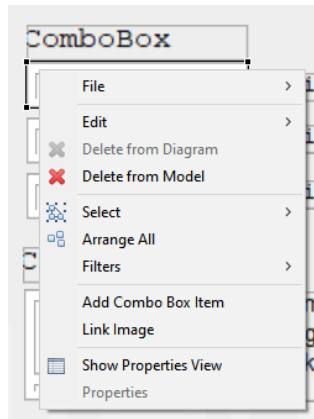


FIGURE 3.97: Context menu for ComboBoxes in the *Page Editor*.

After a new item has been created via the context menu, the central properties can be edited in the *Add Combo Box Item* dialog (see Figure 3.98). However, `Text`, `UserDefinedId` and `Mouse Over Text` can of course also be changed via the *Properties* view.

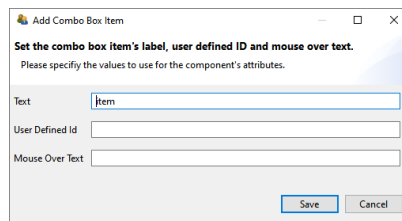


FIGURE 3.98: Dialog to configure a component of type `ComboBoxItem`.

To be able to edit an item of a `ComboBox` (or `List`), the `ComboBox` must first be selected as a container in the *Drawing Area* of the *Page Editor*. All defined items will then be displayed in the *Component Edit* section. To change the `Text` Property, the `UserDefinedId` or the `Mouse Over Text` the context menu in the *Component Edit* contains the entry *Set Basic Attributes* (see Figure 3.99). The context menu also contains the entry *Link Page* for defining a *Link* (see section 3.11) or a *Conditional Link* (see section 4.3).

Using the entry *Link Raised Event* that is part of the context menu when a `ComboBoxItem` or `List Item` in the *Component Edit* are clicked with the right mouse button, *FSM Events* can be added that are triggered when the item is clicked.

`Comobox`-items provide the context menu entry *Link Image* that is currently not working as expected.

To delete an item in the *Component View*, the context menu entry 'Delete Item' can be used (see Figure 3.99).

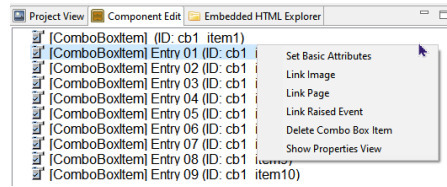


FIGURE 3.99: Context menu for `ComboBoxItems` in the *Component Edit*.

Moreover, the context menu of items in the *Component Edit* also allow to open the *Properties* view.

The formatting of entries in `ComboBoxes` and `Lists` is uniform for all items via the properties in the *Appearances* tab of the *Properties* view. A border is only displayed if a non-zero `Border Width` is defined.

The `SetFrozen()`-operator and the property `Is Frozen` is currently not working for `ComboBoxes` and not supported for `Lists`.

If entries in a `ComboBox` are selected a log entry `ComboBox` is written into the *Trace data*. If an entry is selected within a `List`, this can be traced by means of the entry `List Item` in the log data.

3.9.6 Selectable Components in Panels Or ImageMaps

In addition to specific components to collect click-responses, the CBA ItemBuilder allows to use additional components within `Panels` or `ImageMaps` to be selected.

Selectable Components: Different components such as `HTMLTextFields` provide the `Selectable` property, that can be used to allow components to be selected and de-selected by test-takers (☐ `Selectable`). If several `HTMLTextFields` are placed within a `Panel`, then the selection of the component (e.g., the `HTMLTextField`) can be used to capture a click response. If the option `Selectable: true` is enabled (default setting is `false`), then the entire `HTMLTextField` can be selected by the test-taker at item runtime, as soon as the `Selectable: true` is also defined for the hosting `Panel`. Note that the `selectable`-property is only visible if the icon *Show Advanced Properties* is clicked (see Figure 3.100).

Hosting Component: The selection of an `HTMLTextField` is done by clicking. The exact behavior of the selection can be defined as properties of the component, in which the selectable components (e.g., `HTMLTextFields`) are embedded (e.g., `Panels`). In addition to define `Selectable: true`, the `Panel` allows to define the additional properties `No Deselect` and `Multiple Select Mode` (see Figure 3.101 for illustrations).

An already selected `HTMLTextField` can be unselected by clicking on it, if `No Deselect: false`. Selected `HTMLTextFields` are highlighted with the color that is stored in the

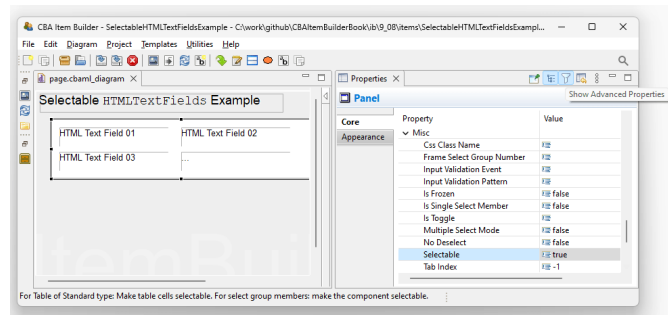


FIGURE 3.100: Advanced properties for Panels in the *Properties*-view to define `selectable: true`.

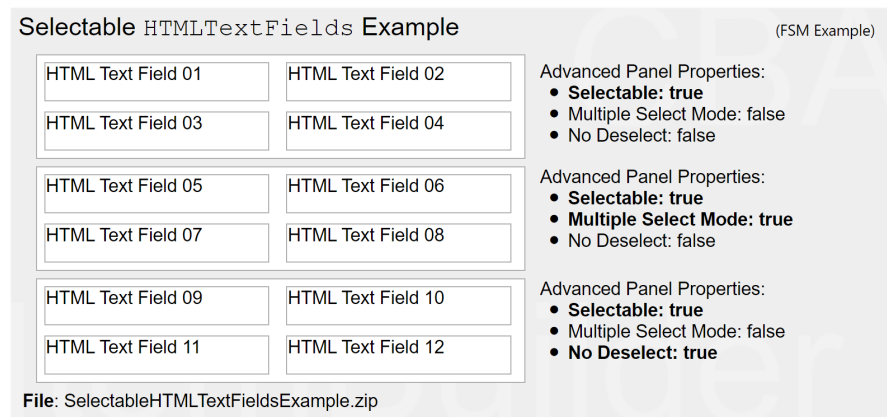



FIGURE 3.101: Item illustrating the `setInputValue()`-operator ([html|ib](#)).

global settings as `Highlight Color` (see section 6.3). The scoring of selectable components such as `HTMLTextFields` is described in section 5.3.2. Operators for finite-state machine and task initialization syntax to modify the selectable property of `HTMLTextFields` are described in section 4.4.6.

3.9.7 Single-Choices as MenuBar with Menu

A component for the implementation of *Single-Choice* tasks analogous to menus of computer programs is provided by the CBA ItemBuilder as `MenuBar` (see Figure 3.102).

MenuBar: Components of type `MenuBar` can also be used added to `Panels` on pages of type *Web Browser* within `WebBrowserToolbar`. To create a *Single-choice* selection as a menu, a `Panel` or a `WebBrowserToolbar` must first be selected in the *Page Editor*. This makes it possible to select the `MenuBar` icon in the *Palette* ( `MenuBar`). If `MenuBar` is se-

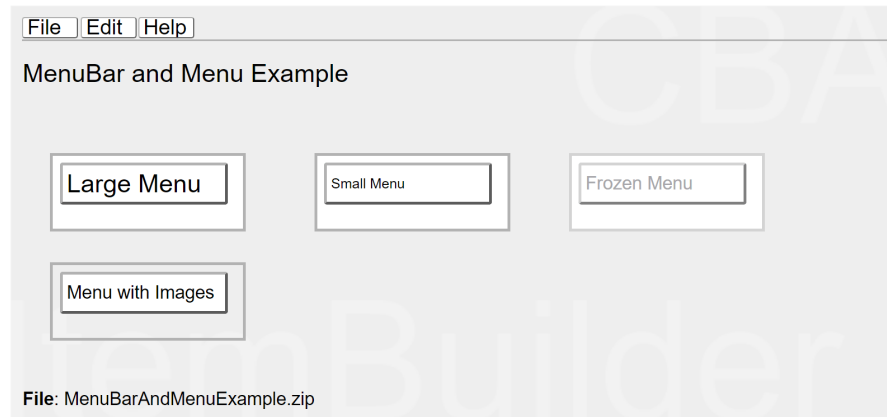



FIGURE 3.102: Item illustrating components of type `MenuBar` and `Menu` (<http://html5lib.org>).

lected in the *Palette*, a rectangle can be drawn within the `Panel` or within the `Web-BrowserToolbar` by clicking with the left mouse button and moving the mouse with pressed mouse button. This rectangle represents the typically invisible border within which one or more menus can be placed. The position of the `MenuBar` can be set exactly via the *Properties* view (properties `x`, `y`, `Width` and `Height`), as well as the border width and the border color (in the *Appearance* tab of the *Properties* view). Via the *Properties* view (property `Raised Event` in section *Component Interaction*) a *FSM Event* can be defined, which is triggered when a test-taker opens the `MenuBar`.


Menu: Unlike `ComboBoxes` and `Lists`, components of type `MenuBar` can provide multiple selections, which can be added to `MenuBars` as a component of type `Menu`. To add one or more `Menu` components to a `MenuBar`, the `MenuBar` must first be selected in the *Page Editor*. Then the icon `Menu` ( `Menu`) is available in the *Palette*. After this icon is selected, a `Menu` can be added inside the `MenuBar`. The size `Width` and `Height` can also be edited directly in the *Properties* view. The position `x` and `y` is not relevant for `Menu` components and is hidden in the default view of the *Properties* view.²⁰

Components of type `Menu` are containers for entries added in the form of `MenuItems` (using the context menu `Add Menu Item Of Menus`).



Entries of `Menus` (`MenuItems`) are not displayed individually in the *Page Editor* and can only be edited via the *Component Edit* (see section 3.1.2).

Images can be added to `Menu-items` using the context menu `Link Image` (or by entering

²⁰To have access to the hidden properties `x` and `y`, advanced properties can be displayed with the icon . However, it is important to keep in mind that the hidden properties are usually ineffective, not needed, or not supported.

the exact file name of the image resource added using the *Resource Browser*, see section 3.10.1, to the property *Image Reference* in section *Display Images of the Properties* view).

Menu-items can be used trigger *Links* (see section 3.11 for details) or *Conditional Links* (see section 4.3 for details), assigned using the context menu entry *Link Page* in the *Component Edit*. Moreover, Menu-items can trigger *Commands* (using the context menu entry *Set Command*, see section 3.12 for details) and *FSM Events* (using the context menu entry *Link Rasied Event*, see section 4.4.3 for details).

3.9.8 Collect Responses using Table and TableCellEditor

As shown in Figure 3.103, additional components can be added to *Panel*s which are needed for the implementation of specific item concepts. This concerns the use of tables (as input format) and so-called *Trees*, which can be used to design user interfaces as part of CBA ItemBuilder items.

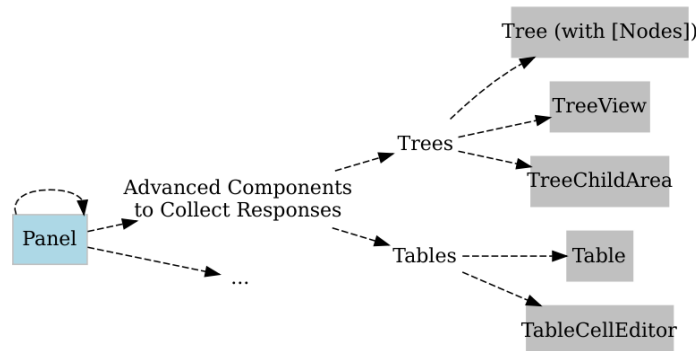


FIGURE 3.103: Overview of components for special purposes.

For the design of pages, the CBA ItemBuilder provides a *Table* component (see item in Figure 3.104 for an example).

Table: Tables can be used for different purposes. They can be used to arrange information on a page in tabular form. Tables can also be configured to mimic simple spreadsheet functions. After the place for a table has been defined in the *Page Editor*, tables can be initialized via the context menu and the entry *Configure Table*. When the dialog shown in Figure 3.105 is called for the first time, the type can be specified once. Besides the table type (*Standard* or *Spreadsheet*), the number of rows and columns and the default background color can be specified. These changes cannot be corrected once the dialog has been closed with *OK*.

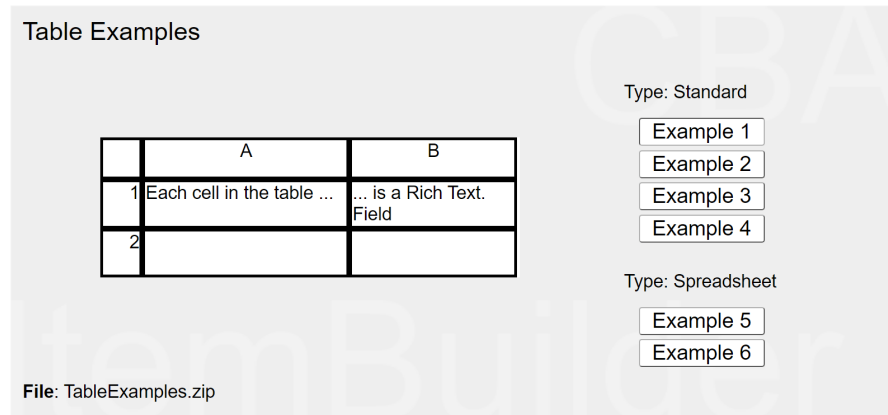


FIGURE 3.104: Item illustrating *Tables* ([html](#)|[ib](#)).

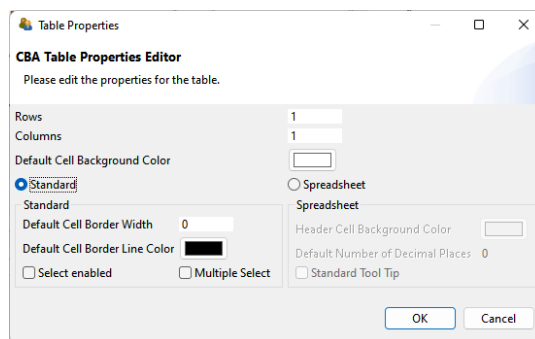


FIGURE 3.105: Dialog to configure *Table*-components.

After initializing a table, the table cells are automatically created and filled with components of type `TextField` (see subsection 3.8.3 for more information). Note that height, width and position of table cells can only be changed manually in the *Properties* view once the table is created.

Table cells are `TextFields` to which an image (context menu entry `Link Image`) and a page as link (context menu entry `Link Page`) can also be assigned. As shown in Example 1 in the item illustrated in Figure 3.104, tables of type `Standard` can be configured to be `Select enabled` (with single-choice or multiple-choice with the additional option `Multiple Select`, see Figure 3.105).

TableCellEditor: For tables of type `Spreadsheet` a component `TableCellEditor` is available, which enables the editing area of the currently selected cell, which is typical for spreadsheets (see figure 3.104). After adding a component of type `TableCellEditor` it can be assigned to a table configured as type `SpreadSheet` via the context menu entry `Attach Table`.

3.9.9 Collect Responses using Tree, TreeView and TreeChildArea

To create browsable folders with nested elements, the CBA ItemBuilder provides three components usually used together (see Figure 3.106 for an example):

- The `Tree`-component shows a tree, in which each element is represented by a label and the hierarchical structure is created at design-time by adding nodes in the *Component Edit*-view of the CBA ItemBuilder (see section 3.1.2).
- The `TreeView`-component that shows a list of all child-nodes of the node selected in the `Tree`.
- The `TreeChildArea`-component that can be used to display pages in a designated area (similar to *PageAreas*, see section 3.5.4), linked to the currently selected node in the `Tree`.

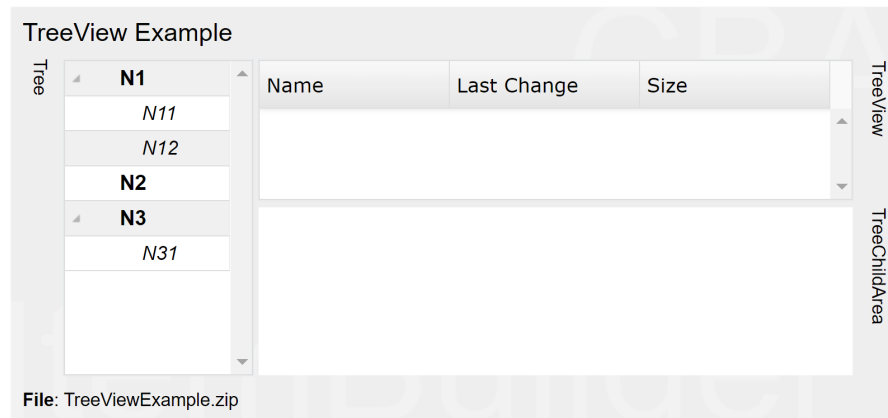


FIGURE 3.106: Item illustrating *Trees* to collect click-responses ([html](#)|[ib](#)).

Tree Configuration: The basic configuration of `Trees` is done in the *Component Edit*-view of the CBA ItemBuilder as shown in Figure 3.107. Each `Tree`-component provides three sections in that can be edited using the context menu. The section `[Type]` is used to define custom node types (`[TreeNodeType]`), for instance, to specify the font properties for all nodes assigned to this type. Each node must be assigned to a node type. The actual nodes that are to be displayed in the `Tree` (and the child nodes in the `TreeView`) are defined in the section `[Nodes]`. The columns shown in the `TreeView` are user defined and created in the *Component Edit*-view in the section `[Columns]`. Each child node in the section `[Columns]` corresponds to one column in the `TreeView`.

The configuration of nodes is done using the context menu in the section `[Nodes]` using the *Component Edit*-view of a `Tree`-component, as shown in Figure 3.108. Nodes are added using `Add Tree Node` and removed using `Delete Tree Node`. The node text (i.e., the displayed text in the `Tree` and `TreeView`-components), the *Mouse Over* text and

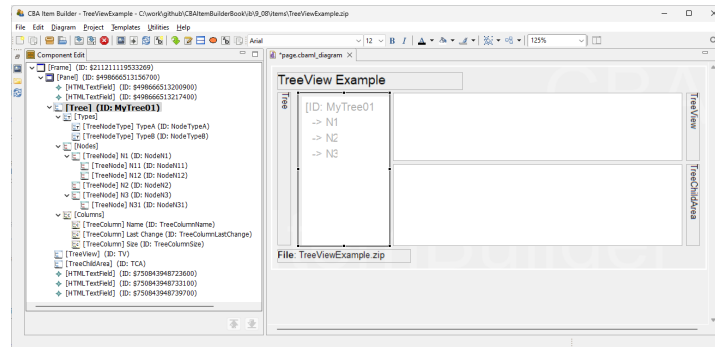


FIGURE 3.107: Configuration of Tree-component in the *Component Edit*-view.

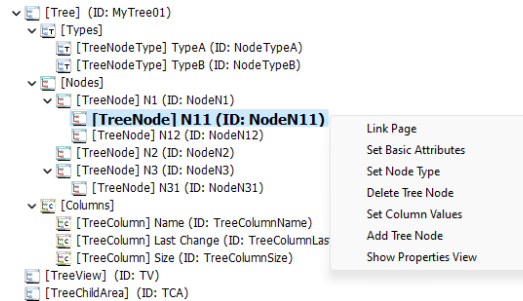


FIGURE 3.108: Context menu for [Nodes] in the *Component Edit*-view of a Tree-component.

the *UserDefinedId* are configured using the context menu entry *Set Basic Attributes* or by entering the values directly into the *Properties*-view.

Each node must be assigned to a node type using the entry *Set Node Type*. This requires to define at least one element in the section [Type], using the context menu entry *Add Tree Node Type* that is available via right-click on the element [Types] within the element [Tree] in the *Component Edit*-view. A node type is defined by a text, a *UserDefinedId* and an optional *Mouse Over Text*. Additional properties for nodes of that particular type can be defined in the *Properties*-view (e.g., details concerning the text presentation are used when *Use Tree Front*: false).

TreeView Configuration: The Tree component and the TreeView-component need to be linked. To link a TreeView to a Tree, right-click on the TreeView-component in the *Page Editor* provides the context menu entry *Set Tree*. Since Tree and TreeView are linked, the selection of a component in the Tree and in the TreeView are synchronized (i.e., selecting a node in the Tree selects the corresponding node in the TreeView and vice-versa).

The data about selected nodes shown in the TreeView are values for the columns, de-

defined for each node by selecting the context menu entry *Set Column Values* (see Figure 3.108). The values need to be entered in the dialog shown in Figure 3.109 according to the order in which the [Columns] are defined in the *Tree*-component.

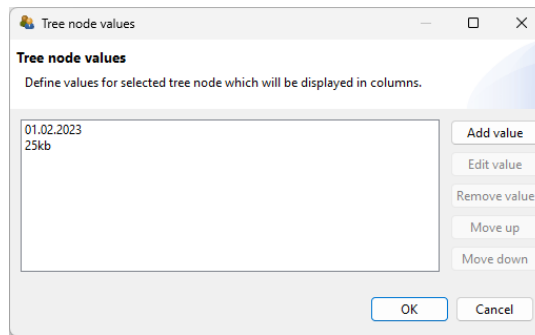


FIGURE 3.109: Edit values for [Nodes] shown in the *TreeView*-component.


Additional configurations (e.g., *Sortable*) can be specified in the *Properties* view of the *TreeView*-component.

TreeChildArea Configuration: The *Tree* component and the *TreeChildArea*-component need to be linked (similar to the *TreeView* as described above). Since *Tree* and *TreeChildArea* are linked, a child page can be linked to each node. When the node is selected in the *Tree* or the *TreeView*, the linked pages is shown in the *TreeChildArea*. Pages can be linked in the context menu of the *Tree*-component in the *Component Edit* view (see entry *Link Page* in Figure 3.107).

3.9.10 Graphical *Single- or Multiple-Choice* Formats using *ImageMaps*

Another component that can be used to computerize both *Single-* and *Multiple-Choice* response formats with the CBA ItemBuilder are *ImageMaps*. As the name suggests, these are images on which clickable areas can be defined. If the selection of a new clickable area deactivates an already selected area (*Multiple Select Mode: false*), then a *Single-Choice* response format is created. If additional elements are selected without deactivating already selected areas (*Multiple Select Mode: true*), a *Multiple-Choice* response format is created.

ImageMaps: *ImageMaps* that can be added to components of type *Panel* are shown in Figure 3.110).

To add an *ImageMap* to a page, the *Panel* must first be selected. After that the entry *ImageMap* ( *ImageMap*) is available in the *Palette*. If this icon is selected, it is possible to click inside a *Panel* in the *Drawing Area* of the *Page Editor* and draw a rectangle while holding down the mouse button. When releasing the left mouse button the *ImageMap* is added to the page. In the *Properties* view, the exact position and size of the *ImageMap* can then be adjusted in the *Position* section.

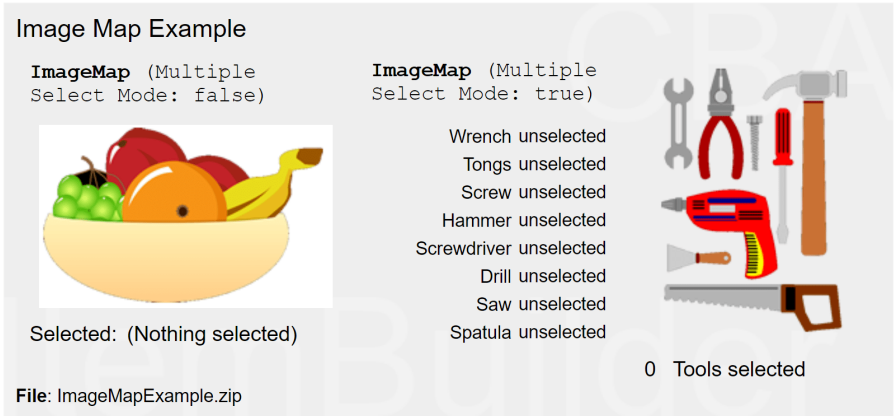



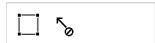


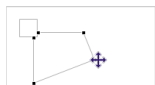
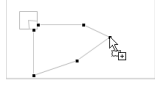
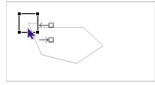
FIGURE 3.110: Item illustrating ‘ImageMaps’ ([html](#)|[ib](#)).

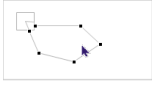
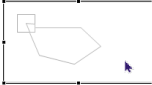
The graphical design of `ImageMaps` is usually done using a background image, which can be added using the context menu entry `Link Image` (see section 3.10). Alternatively, a background color can be configured in the *Appearance* tab of the *Properties* view (if `Is Transparent: false`). A border can be displayed if a non-zero `Border Width` is configured.

ImageArea: Components of type `ImageMap` are containers for clickable areas, which can be inserted as `ImageArea` Or `ImageTextField`. One or more `ImageArea(s)` can be created in the *Page Editor* inside `ImageMaps` (see table 3.8 for a detailed description). The size of `ImageAreas` is insignificant, since they are only used to anchor polygonal features, which must also be inserted as `AnchorPoint`.


TABLE 3.8: Guide for creating `ImageMaps` in the CBA ItemBuilder

| Page Editor | Description |
|-------------|--|
| | To add <code>ImageAreas</code> to an <code>ImageMap</code> , the <code>ImageMap</code> must first be selected in the <i>Drawing Area</i> of the <i>Page Editor</i> . |
| | When the <code>ImageMap</code> is selected, the <code>ImageArea</code> icon (<code>ImageArea</code>) can be selected in the <i>Palette</i> . After that a small rectangle (circa 25x25 pixels) needs to be drawn inside the <code>ImageMap</code> component, which represents the <code>ImageArea</code> . |

| Page Editor | Description |
|---|--|
|  | After the ImageArea has been added to the ImageMap, this component can be selected in the Page Editor. The displayed connector points () can be ignored. Once the ImageArea is selected, the icon AnchorPoint () can be selected in the Palette. |
|  | Unlike other components, AnchorPoints are added by a simple click. The CBA ItemBuilder shows that this is only possible by clicking on components of type ImageArea. |
|  | If the icon AnchorPoints is selected, then a selectable area for the ImageMap can be defined by a single click on the ImageArea. It is important to click inside the component of type AnchorPoints in the Drawing Area. |
|  | Adding the AnchorPoints by clicking on the ImageArea automatically creates a rectangle whose corner points can now be adjusted to the desired shape. The polygon path symbolized by the corner points is always anchored to the ImageArea. |
|  | The individual corner points of the polygon path can be moved in the Drawing Area with the mouse. The polygon path must be selected for this operation first. After that the mouse can be moved over a corner point. As soon as the mouse icon indicates that the corner point can be moved, the polygon path can be adjusted at the selected point using drag and drop. If the corner point falls exactly on the line between two already existing points by a drag and drop operation, then the corner point is removed. |
|  | A new corner point can be added by clicking on a line of the polygon course. The Page Editor shows this option by a small + at the mouse pointer. After the new corner point has been added, it can be moved again via drag and drop. |
|  | There are three different selections for ImageMaps. If the ImageArea is selected, properties for this polygon path can be changed in the Properties view. |

| Page Editor | Description |
|---|---|
|  | If the polygon path itself is selected, then the <i>Properties</i> view is not available, but individual points can be moved, added or deleted. |
|  | Properties of the whole ImageMap (e.g. the property <code>Multiple Select Mode</code>) can be edited if the ImageMap is selected in the <i>Page Editor</i> . |

ImageAreas are used to define the `UserDefinedId` for each click-sensitive path of an ImageMap, do define using the tab *Appearance* of the *Properties* view the background color and optional border (shown if `Border Width` is different from zero), to assign *FSM Events* (using the context menu entry `Link Select Event` and `Link Deselect Event`) and to define the `Transparency` (a value between 0 and 100 that defines the transparency of the defined background color, when the path is selected).

AnchorPoint: For each ImageArea exactly one component is required to anchor the polygon path. For that purpose, an `AnchorPoint` must be added to the ImageArea. The `AnchorPoint` is not drawn as rectangle in the *Page Editor*. Instead, adding the `AnchorPoint` is performed by selecting the ImageArea to get access to the icon `AnchorPoint` in the *Palette* (, `AnchorPoint`), followed by a single click with the left mouse button inside of the rectangle that represents the ImageArea in the *Page Editor*. After adding the `AnchorPoint`, the automatically generated polygon path can be adjusted as described in table 3.8. Once the ImageArea is added to the ImageMap, only the polygon path or the ImageArea can be selected in the *Drawing Area* of the *Page Editor*. `AnchorPoint` do not provide additional properties.

ImageTextField: As shown in Figure 3.87, ImageMaps can contain standard components of type `HTMLTextField` and `Button` as well as `ImageTextFields` in addition to `ImageAreas` with `AnchorPoints`. Components of type `Button` (see section 3.11.2) and `HTMLTextField` (see section 3.8.2) added to ImageMaps can contain *Links* (see section 3.11), while the specific component of type `ImageTextFields` are `TextFields` (see section 3.8.3) without the possibility to add *Links* and to use the *Highlighting* feature. Instead, `ImageTextFields` can be added to `ImageTextFields` as click-sensitive texts that integrate into the *Single-Choice* (`Multiple Select Mode: false`) or *Multiple-Choice* (`Multiple Select Mode: true`) behavior of `ImageTextFields` (see Figure 3.111 for an example).

Figure 3.111 illustrates that multiple ImageMaps can be added to a page simultaneously and that ImageMaps not necessarily must use a background image.

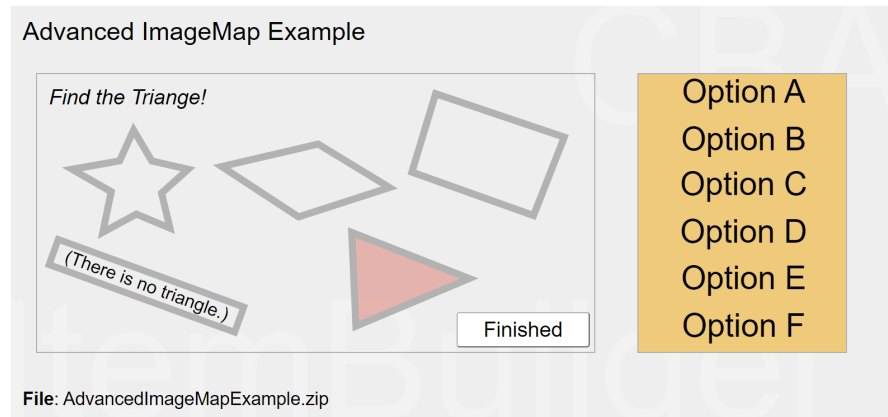
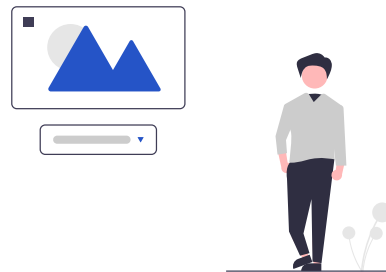


FIGURE 3.111: Item illustrating advanced ImageMaps ([html|ib](#)).

3.10 Images and Multimedia Components



Many components can be used to display images on CBA ItemBuilder *Pages*. The following example (see Figure 3.112) shows a selection of components that can be linked to images.

The component that can *only* display images is the `ImageField`. It can be used to display an image with a fixed size (width and height). The size of the image used should be preferably match the original size of the image in pixels (see 6.2.1 for details). Size and position can be changed in the *Page Designer*. To determine the exact position, the properties can also be defined in the *Properties* view (see Figure 3.113).

Images can also be used as background in `Panels`. A panel without any further element leads to the same result as an `ImageField`. However, if required, additional components can be placed in panels. In the example above, this approach was used for the `SingleLineInputField`, which was placed on a panel with a pattern. Images are frequently used for the design of buttons. As described in subsection 3.11.2 the CBA

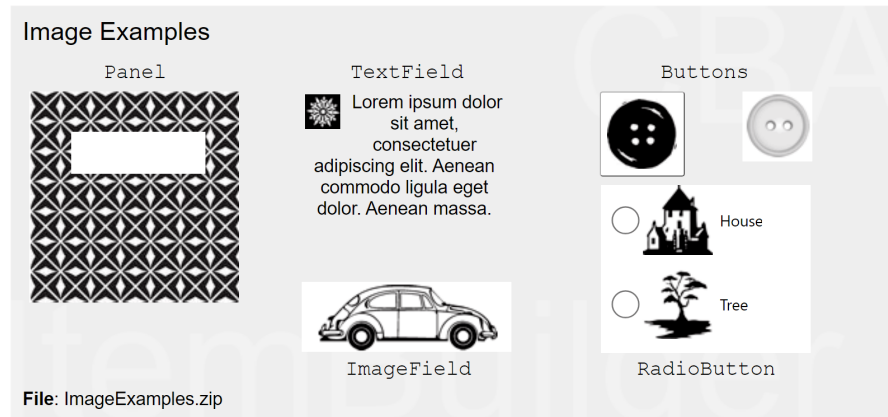


FIGURE 3.112: Item illustrating the use of images ([html](#)|[ib](#)).

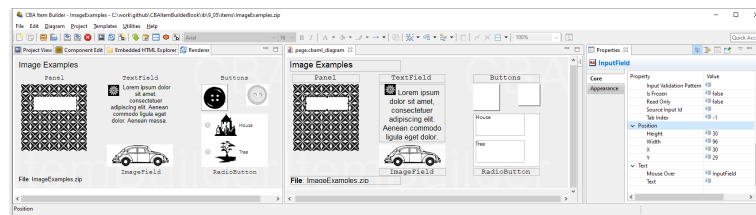


FIGURE 3.113: Rendering (left) and Page Editor (middle) and Properties view showing item ImageExamples.zip ([html](#)|[ib](#))

ItemBuilder distinguishes two configurations, namely *Standard button* (only one image can be used) and *Image button* (one image can be defined for each of the states *activated*, *deactivated*, *pressed* and *mouse over*). RadioButtons can also contain images. If the images are specified as part of the RadioButton, the corresponding option can also be selected by clicking on the image. Finally, the example in Figure 3.113 also shows that TextFields can contain images in addition to texts.

3.10.1 Manage Ressources

To use image files, audio data and video files for designing item they must be loaded into the CBA ItemBuilder *Project File* using the so-called *Resource Browser*. The *Resource Browser* supports the import of selected file formats that can typically be displayed in Web browsers.

Supported File Formats: The following Table 3.9 gives an overview of file formats for images, audio and video.²¹

TABLE 3.9: Supported File Formats for Resources

| Extension | Description | Type |
|-----------|---|-------|
| *.gif | GIF: Graphics Interchange Format, Lossless compression, transparency option, animation option, good browser support | Image |
| *_min.png | PNG: Portable Network Graphics, Lossless compression, transparency option, good browser support | Image |
| *.jpg | JPEG: Joint Photographic Experts Group, Lossy compression, good browser support | Image |
| *.tiff | TIFF: Tagged Image File Format, Lossless compression (not supported) | Image |
| *.mp3 | MP3: Lossy compression, good browser support | Audio |
| *.mp4 | MP4: Lossy compression, good browser support | Video |
| *.webm | WebM: Lossy compression, good browser support | Video |
| *.ogg | Ogg: Lossy compression, acceptable browser support | Audio |
| *.ogv | Ogv: Lossy compression, not supported by all major browsers | Video |
| *.wav | Wav: Waveform Audio File Format, good browser support | Audio |

Note that the XML-based vector image format for *Scalable Vector Graphics (SVG)* is currently not supported by the CBA ItemBuilder.

Resource Browser: To manage resources, the built-in *Resource Browser* provides the following three options: A) Add a resource file to the item project file, B) delete a selected resource file from the item project, and C) delete all unreferenced resources within a project.

The *Resource Browser* can also be opened using the entry `Browse resources` in the `Project` menu. To import a resource file, the *Resource Browser* once opened provides the button `Add`. This button opens an open file dialog that can be used to select one or multiple files of supported format for importing.

Resources whose file name contain special characters or blank space cannot be used in item project and displayed at runtime.²² Resources with file names that cannot be used at runtime are detected by the *Resource Browser* and the CBA ItemBuilder displays a warning message.

²¹Note that for historical reasons the CBA ItemBuilder also lists *.flv as import format for videos and *.bmp for images, which are no longer supported.

²²Only plain latin characters (A-Z and a-z, no diacritics), digits (0-9), dot (.), underscore (_), hyphen (-), plus sign (+) are allowed.

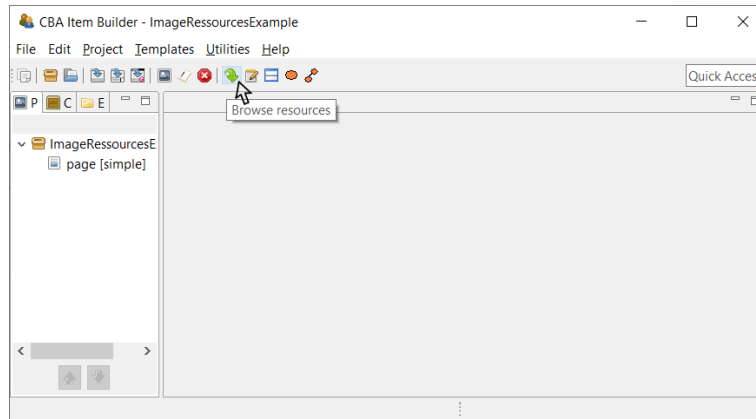


FIGURE 3.114: Icon Browse resources in the CBA ItemBuilder Toolbar.

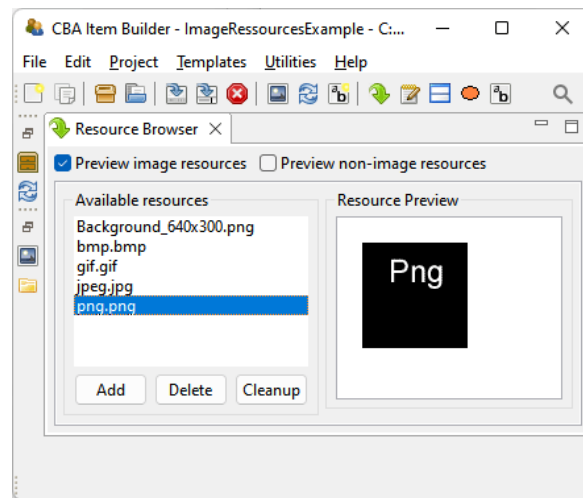


FIGURE 3.115: Resource Browser to manage embedded resources.

A resource entry selected in the list of *Available resources* (see Figure 3.115) can be removed, using the `Delete` button. If the automatic start of audio or video files should be suppressed, the checkbox `Skip Preview` can be marked.

As described in the following, resources are linked to components. Resources that are used in the current item project (i.e., images, audio files, and videos that are *linked* to components) cannot be deleted. Resources that *not* linked to a component (i.e., unused resource files) can be removed permanently from the item project using the button `Cleanup`.



Deleting unused resources can significantly reduce the file size of an item project. Many and large resources will slow down the processing of item project files at run-time.

Existing resources can be updated without linking them to components again by deleting the resource with the *Resource Browser* and adding a resource with the identical name. Note, however, that *Generate and Save* (see 3.2.1 below) will be necessary to apply the update.

Before you import resources in the item project, images, audio files, and videos should be prepared and converted into a suitable format. Images, for instance, should be prepared by image processing software and resized to the size required in the item project (see section 6.2 for further information).

A selection of file formats is commonly used for displaying images on the web. The CBA ItemBuilder supports image files in the formats JPEG, BMP, GIF and PNG.

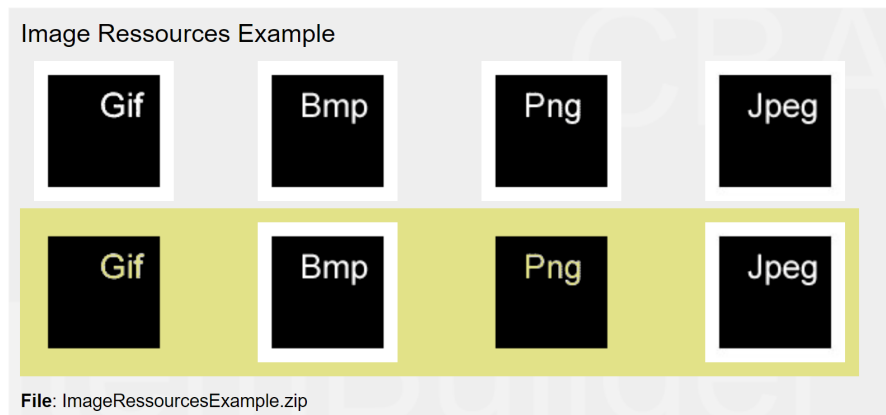


FIGURE 3.116: Item illustrating the use of different image file formats ([html|ib](#)).

Audio files can be added in MP3, OGG and WAV formats in the CBA ItemBuilder's *Resource Browser*. A tool to edit and convert audio files is [Audacity](#). For audio files to be played, the hosting may also need to be configured correctly (e.g. with regard to range headers). Therefore, it is recommended to thoroughly test the use of the audio format on the planned devices. In addition, audio output in the browser typically cannot occur until at least one user interaction (click, keyboard input, or similar) has taken place. The three audio formats are illustrated in the following item shown in Figure 3.117.

Video files can be added in MP4, OGV and webm formats in the CBA ItemBuilder's *Resource Browser*. A tool to convert video files is [VLC media player](#). The three video formats are illustrated in the following item shown in Figure 3.118.

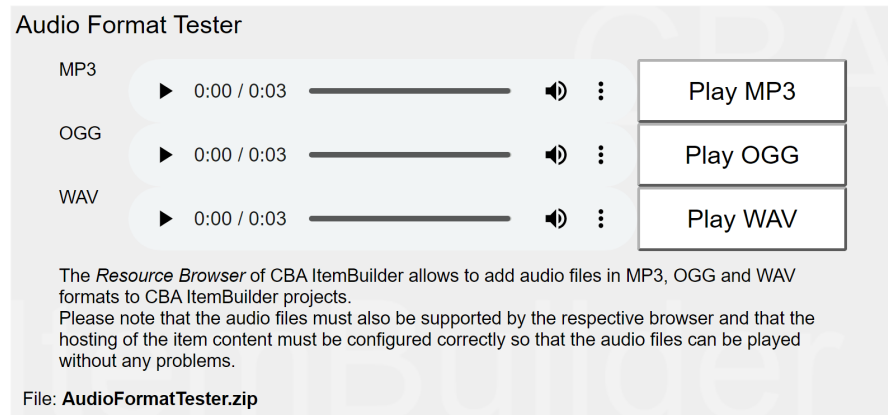


FIGURE 3.117: Item illustrating the use of different audio file formats ([html](#)|[ib](#)).

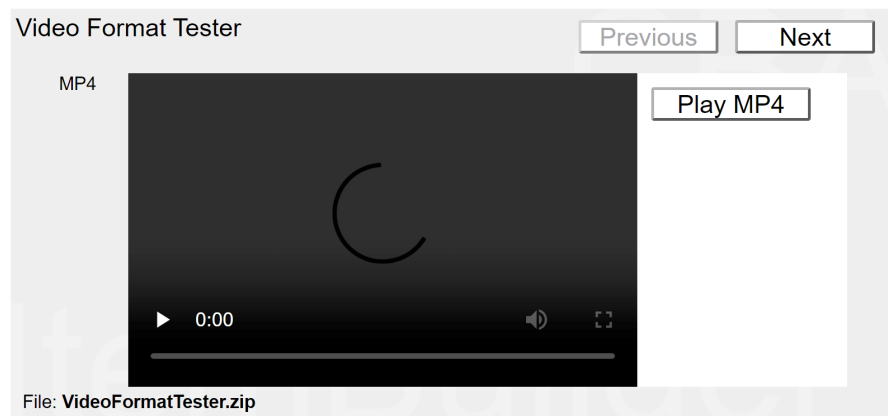


FIGURE 3.118: Item illustrating the use of different video file formats ([html](#)|[ib](#)).



Depending on the deployment method not all audio and video file formats are supported. Currently, the Ogg / Ogv or Mp3 / Mp4 are suggested formats to be tested first. For online use, MP4 with the codec H264 and WebM with the codec VP8 or VP9 are particularly suitable because of their high compatibility.

As a rule of thumb, MP4 should work in all common browsers, i.e., Android, Chrome, Internet Explorer, Edge, Firefox, Opera, and Safari - only Firefox on Linux requires a plugin (FFmpeg). Further, all universal browsers support WebM; the only exceptions are Internet Explorer and Safari.

Resource files of a supported format can be used in item projects by *linking* them to components, as described in the next section.

3.10.2 Components to Show Images

Importing images via the *Resource Browser* is only the first step. An image must also be assigned to a component in the *Page Editor* to be visible on a page, and all components that can show images provide the “Link Image” entry in the context menu.

Dedicated ImageFields: ImageFields are components dedicated to showing images on pages. If image file formats support transparency (i.e., *.gif and *_min.png), the property `Is Transparent` can be set to true and `Background Ccolor` can be used. Images can have borders (i.e., `Border Color` can be used when `Border Width` is different from zero) and ImageFields support text for `Mouse Over` (text that is displayed when mouse over the ImageField). Components of type ImageFields are typically added to Panels, but are also supported within WebBrowserToolbars.

The CBA ItemBuilder automatically resizes components of type ImageField and Panel to the native size of the image. Although the components can be resized afterward, resizing images either risk a loss in quality or a potential waste of resources, as item projects might require more bandwidth as needed (see section 6.2 for further information). A tool to create and edit images is paint.net.

Background Images in Panels: Images can also be used to create the background of a page. For this, no ImageFields have to be used in the CBA ItemBuilder, but images can also be linked directly to a panel. Note, however, that when images are added as background images to Panels via the `Link Image` option, they must be precisely the size of the panel in `Width` and `Height`. If this is not the case, then unlike ImageFields, they will be cut off (if the image is larger than the Panel) or displayed repeatedly (if the image is smaller than the Panel). This difference between ImageFields for displaying images and the use of images in the background of Panels is shown in Figure 3.119.

Other Components that allow to Link Images: TextFields (see subsection 3.8.3) can contain images (see Figure 3.112 in subsection 3.10 for an example). Images are necessary for components of type ImageMap (see section 3.10). Images can also be used to design components of type Button as so-called *Image Button* (see subsection 3.11.2). Moreover, TableCells (see subsection 3.9.8) can be linked to images. Finally, MapBasedVariableDisplay (see subsection 4.2.5) can be configured to show images assigned to values using so-called *Value Maps* (see subsection).

3.10.3 Components for Audio and Video Content

For audio and video content integration, the *Page Editor* provides components of Audio and Video type that can be placed into Panels. These two components can be configured with or without visible controls and are used to play hard-linked audio or

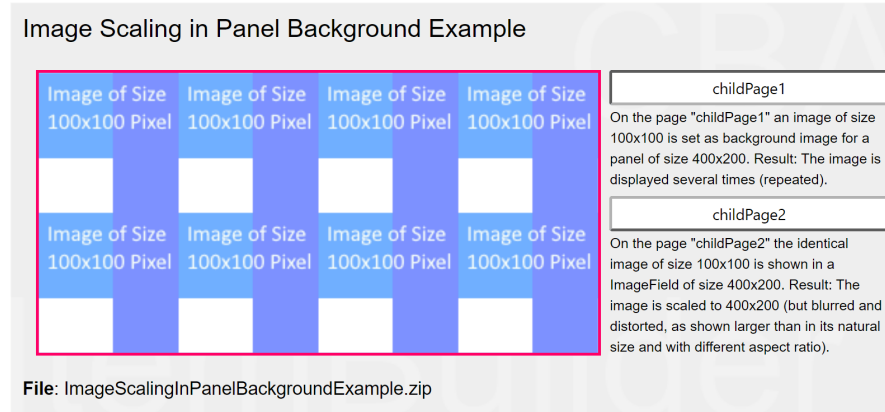


FIGURE 3.119: Item illustrating the difference scaling of `ImageFields` and images as background in `Panels` ([html](#)|[ib](#)).

video resources. Multiple `Audio` or `Video` components can be embedded if numerous audio or video resources are required on a page. For special use cases, the CBA ItemBuilder also provides additional functionality to play audio or video resources selected via a *Value Map* with a variable (see 4.2.5 for a description of using `MapBased-VariableDisplay` with the `Value Display Type` either `AUDIO` or `VIDEO`).

Link Audio / Video to Components: Similar to images, videos and audio files can be added as resources to the CBA ItemBuilder project. After inserting them in the *Resource Browser*, they can be linked to the components using the context menu item *Link Video* or *Link Audio* (as shown in Figure 3.120).

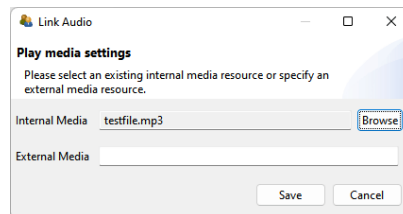


FIGURE 3.120: *Link Audio* dialog for `Audio` components.

Internal vs. External Media: In addition to internal resources that are part of the CBA ItemBuilder *Project Files* (referred to as *Internal Media*, audio and video files can also be inserted as URL's (referred to as *External Media*). It should be noted, of course, that the *External Media* must then also be available at the time of the test execution, which is typically not guaranteed for offline deliveries (see section 7.2).



To make audio and video resources part of the CBA ItemBuilder *Project Files*, they must be inserted via the *Resource Browser* and used as *Internal Media*.

Controls: The CBA ItemBuilder allows using audio and video resources with the default controls (i.e., a start and a stop button; property `Hide Controls`: `false`). With this setting, test-takers can also navigate within the audio and video files and thus reset the playback. Alternatively, the default buttons can be hidden with the `Hide Controls`: `true` option. For this use case, the `setMediaPlayer()` operator must then be used to start the output and pause or stop it if necessary (see section 4.4.6). This property `Hide Controls` is illustrated for audio and video components in the item shown in Figure 3.121.

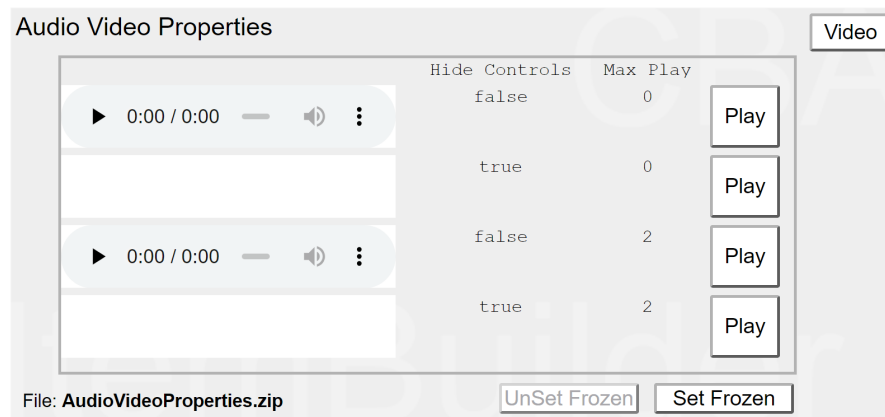


FIGURE 3.121: Item illustrating the use properties `Hide Controls` and `Max Play` Of Audio and Video components ([html|ib](#)).

Max Play: Variables and the finite-state machine can be used to control precisely when and how often multimedia resources can be played if the standard controls are not used to control audio and video output. For simple use cases, however, the `Max Play` property (the default value 0 means any number of times) can be used to specify how often an audio file or a video file can be played. When the number is reached, the default controls are automatically hidden (see the item in Figure 3.121 as an example). The `Max Play` value can be changed directly in the *Properties* view.

Note, however, that using a custom *Play* button (i.e., a component of type `Button` that triggers an event and uses the *Finite-State Machine* operator `setMediaPlayer()` to play the audio, see section 4.4.6) is not acknowledging the `Max play` setting (see Figure 4.54) for an implementation of a restriction for audio playback using *Finite-State Machine* variables).

The link between the components of type `Audio` and `Video` and the *Finite-State Machine*

is provided by a number of events that can be triggered (and processed by the *Finite-State Machine*, as described in section 4.4; see Figure 3.122).

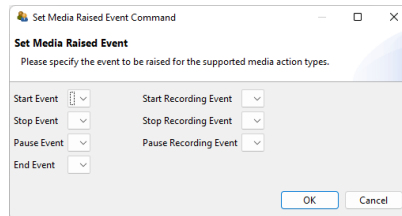


FIGURE 3.122: *Set Media Raised Events* dialog for `Audio` components.



Starting with CBA ItemBuilder 9.8, an `End Event` is also provided, and the events are triggered even if the playback state is changed by an FSM operator (see section 4.4.6).

Automatic Start: If an audio or video resource is to be played automatically when a page is visited, this can be configured with the `Automatic Start: true` property. The items created with the CBA ItemBuilder are rendered in web browsers. Thus they are subject to the security restrictions of the browsers. One of these security restrictions requires user interaction before audio output may be started. This is always not the case if `Automatic Start: true` (or via the *Finite-State Machine*) is used to start an audio output in an item before the user has clicked or otherwise interacted with the HTML output. This is not a problem for operative test execution, as user first interaction is usually triggered by entering a login name or confirming the privacy notices before the first audio output is required. However, the security restriction also applies to editing items and the *Preview* used for this purpose. Therefore, the *Preview* can also be started with a simulated *login dialog* (see *Show Login Dialog* option in section 1.4.2). If this option is not activated (and the warning message of the CBA ItemBuilder is ignored), the `Automatic Start` function cannot work in the *Preview*.

Audio Volume: Controlling the audio volume is possible with the components of type 'Audio' and 'Video' for the test-taker when the default controls are used (`Hide Controls: false`). If the start volume of the is to be set or if the volume of individual audio or video components is to be changed, an operator is available (see section 4.4.6) that can be used in the finite-state machine. Controlling the system volume of the device on which the web browser is executed within which the items created with the CBA ItemBuilder are rendered is not possible from within the item. The test delivery software can take over this functionality if the test execution takes place under controlled conditions (e.g., using kiosk mode with a USB stick deployment, see subsection 7.5.3 for an example).



The first audio output of an item within a test is delayed by a latency of about one second from the browser.

Alternate Video / Alternate Audio: Not all video formats (formerly also audio formats) are supported in all browsers. The CBA ItemBuilder, therefore, provides the functionality to link an *Alternate Video* to a component of type `Video` in addition to the primary resource (`Audio` analogously). The alternate media is used if a browser does not support the primary media.

Recording: The audio component can also be configured to record audio, using the property `Record Audio: true`. Starting (and ending) the recording is connected, for instance to buttons, as shown in Figure 3.123, using the dynamic features (i.e., *Finite State Machine Events*) described in Chapter 4.

Audio Recording Example

Instructions: First, press "Start Recording" to make an audio recording. Then, speak into the microphone and press "End Recording" at the end. After a recording, it is possible to listen to the recording again.

Start Recording

End Recording

Notes:

- The audio recording will only work if access to the microphone is granted.
- Due to a bug in CBA ItemBuilder 10.0, the recording is played twice after recording.

File: `AudioRecordingExample.zip`

FIGURE 3.123: Item illustrating audio recording using the `Audio` component ([html](#)|[ib](#)).

As Figure 3.124 shows, the component of type `Video` can also be used in the same way to record from the webcam.

Note that the audio (and video) recording will only work if access to the microphone (or webcam) is granted. Moreover, due to a bug in CBA ItemBuilder 10.0, the audio-track of the recording is played twice after recording.

A couple of additional features of `Audio` and `Video` components appear in the CBA ItemBuilder user-interface but are currently either deprecated, still under development or undocumented. The property `Use Audio` is currently not functional. Moreover, as illustrated in Figure 3.121 the `Frozen` property is currently not functioning, and the feature `VideoTextArea` is currently not used.

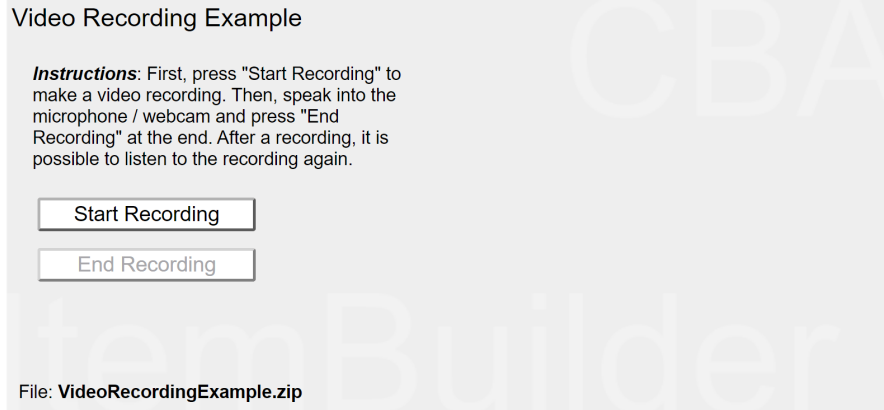


FIGURE 3.124: Item illustrating audio recording using the `Video` component ([html](#)|[ib](#)).

3.11 Links between Pages



Assessment components created with em CBA ItemBuilder can consist of a single page or of multiple pages. If multiple pages are used, then *Links* attached to components can be used to switch between pages.

Links vs. Conditional Links: Links between pages in the CBA ItemBuilder can either be static (i.e., the same target page is always addressed). Or the link's target is defined by conditions (i.e., the evaluation of the conditions decides which page is addressed). Accordingly, the CBA ItemBuilder differentiates between *Links* and *Conditional Links*.

The following example in Figure 3.125 illustrates simple *Links* and a *Conditional Link*.

Figure 3.126 graphically illustrates the distinction between *Links* and *Conditional Links* that is demonstrated in the item in Figure 3.125.

Links and Conditional Links Example

Page A

This page contains a button with a simple link to page B.

(Static) Link to Page B

The linked page B contains three links:

- A simple link to page A.
- A conditional link that always links to page A.
- A conditional link that links to page C if a checkbox is selected (or to page A).

File: LinksAndConditionalLinksExample.zip

FIGURE 3.125: Example for *Links* and *Conditional Links* ([html](#)|[ib](#)).

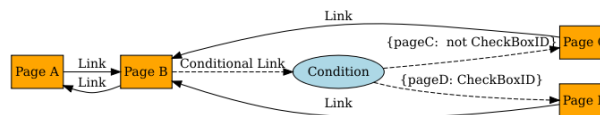


FIGURE 3.126: Schematic display of links in Figure 3.125.

The button on Page A always links to Page B. Similarly, the buttons on Page C and Page D always link back to Page B. The difference between these *simple links* and a *conditional link* is illustrated on Page B. Here, the button *Link to Page A* has a *simple link*, always referring back to Page A, while the button *Link to Page A or C* is configured with a *conditional link* that points either to Page C or to Page D.

The condition defined for this example has two parts, separated in the two lines:

```
{pageD: CheckBoxID}
{pageC: not CheckBoxID}
```

The first line defines the link to pageD (the page name of Page D), if the component of type `Checkbox` with the `UserDefinedId: CheckBoxID` is selected. If this `Checkbox` is selected, then the conditional link refers to pageD. If the component of type `Checkbox` with the `UserDefinedId: CheckBoxID` is not selected, the first line is ignored and the

second line is evaluated. Since `not CheckBoxID` is true, the conditional link will link to `pageC` (the page name of `Page c`) in this example.

Conditions for *Conditional Links* are stored as syntax in the CBA ItemBuilder in the form shown. This is introduced in detail in section 4.1 and described specifically for *Conditional Links* in section 4.3.

The next section 3.11.1 describes how *Links* for switching between pages can be connected to various components in the *Page Editor*. Links between pages and *xPages* are covered in section 3.11.4, including an overview of links between pages of different page type.

Connecting multiple pages using links is a central concept of the CBA ItemBuilder, which is needed to create complex items. Analogous to hypertext, it is used to switch (whole) pages. With the help of special page types (e.g. *WebChild* pages, see section Y) it can also be used to change sections of the visually presented information. Finally, links can also be used to display content in the form of dialogues (see section Y). Additional possibilities to switch between pages are:

- Pages can be linked to *States*. In this case, a page is displayed when the particular state that is linked to that page becomes the current state. More on this option can be found in section 4.4.9.
- Pages in *PageArea*-components (see section 3.5.4) can also be switched with a special operator (`setEmbeddedPage (PageArea,PageName)`). For more on this option, see section 4.3.4 (conditional links) and section 4.4.6.

Links vs. Commands: Links are used for switching between pages *within* a CBA ItemBuilder *Task* (see section 3.6 for task definition and section 8.2 for more information on splitting assessment components into individual tasks). To switch and navigate *between* CBA ItemBuilder tasks, *Runtime Commands* are used (see section 3.12).

Links vs. Events: Finally, the CBA ItemBuilder also allows to assign *Events* to components (see section 4.4.3). Events can be used to trigger *Transitions* in the *Finite-State Machine* of a project (see section 4.4), and operators within transitions (see section 4.4.6) or pages assigned to states (see section 4.4.9) can also be used to change pages.

3.11.1 Simple Components to Link Pages

Various components can be used to trigger switching between pages as links. Almost exclusively for switching between pages, the dedicated component `Link` is used, which is described first in the next section.

Dedicated Link-Component: During the execution of an assessment component created with the CBA ItemBuilder, components of type 'Link' are displayed as text. Clicking on the text will switch to the linked page. If a mouse is positioned over a link as a pointing device for a short time, a mouseover text can be displayed:

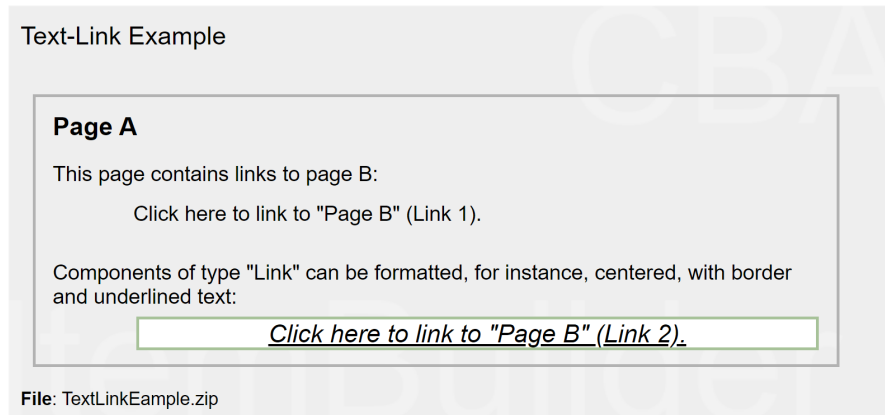



FIGURE 3.127: Example for components of type `Link` ([html](#)|[ib](#)).

Adding a component of type `Link` to a page in the *Page Editor* requires selecting the container in which the `Link` should be placed first. If a container allows hosting components of type `Links` (such as, for instance, `Panels`), the pallet will show the component `Link` (see section 3.1.3). After selecting the component `Link` (see the small symbol  in the palette, a rectangle can then be drawn in the *Drawing Area* and its position adjusted with the properties `x` and `y` (and `width` and `height`) if necessary. Font family, font size, font color of *Links* can be configured, as well as border and background (see 3.1.4).

The `Link` text and the target page can be configured using the *Context Menu* that is available using right-click on a component of type `Link`, as shown in Figure 3.128.

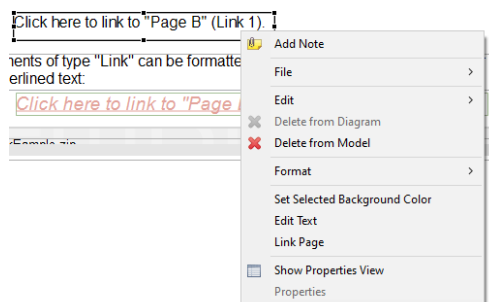


FIGURE 3.128: *Context Menu* for components of type `Link`.

Edit Link Text: To change the text that is shown as *Link*, the entry `Edit Text` in Figure 3.128 can be used. The built-in editor of the CBA *ItemBuilder* allows you to edit the single or multiline link text, as shown in Figure 3.129.

Link Page: Using the context menu shown in Figure 3.128 the dialog to *Link Pages*

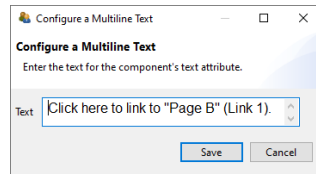


FIGURE 3.129: Dialog *Configure a Multiline Text*.

can be requested. The dialog for linking pages (see Figure 3.130) is structured the same for all components that support this feature. In the section *Select page* on the left side of the dialog, all available pages are listed that can be used as link-targets in the current context. Defining a *link* is possible by selecting one of the pages before confirming the dialog with OK. The selected page can be changed by selecting a new page. Removing a defined link is possible by de-selecting the select link target-page.

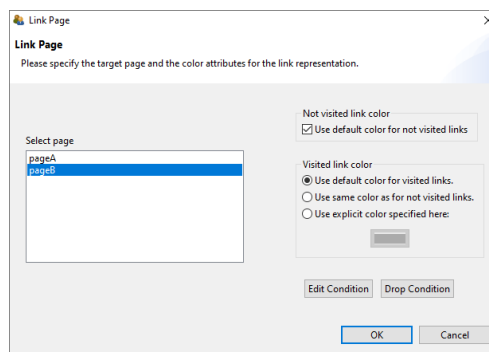


FIGURE 3.130: Dialog *Link Page* used for components of different type.


Visited Link Color: On the right side of the *Link Page*-dialog shown in Figure 3.130 different colors for visited and not-visited links can be defined. The default values for link colors can be defined in the *Global Properties* of a CBA ItemBuilder project file (see section 6.3).

Conditional Links: The buttons *Edit Condition* and *Drop Condition* in the lower right part of Figure 3.130 can be used to edit a *Conditional Link* as described in detail in section 4.3. It is important to note that if *Conditional Links* are used, the one or multiple target pages are not marked as selected in the *Select page* section of the dialog shown in Figure 3.130.

3.11.2 Button-Component

Components of type `Button` play an essential role in designing interactive assessment components with the CBA ItemBuilder. Typically, at least one button is used to

navigate to the following *Tasks* (connected to the `NEXT_TASK` command, see section 3.12.1). Buttons can not only be used for links but also to collect click responses and to trigger *FSM events* (see section 4.4.3).

Component for (Image)Buttons: General-purpose components that also can be used to link between pages are buttons. Adding a component of type `Button` to a page requires selecting a container that can host Buttons (e.g., a `Panel`) first. If the element selected in the *Drawing Area* can contain a button, the *Palette* shows the icon to add a Button (see the small symbol ).

The example item in Figure 3.131 illustrates for components of type `Button` two important properties: The use of images to style Buttons and the property `Toggle`.

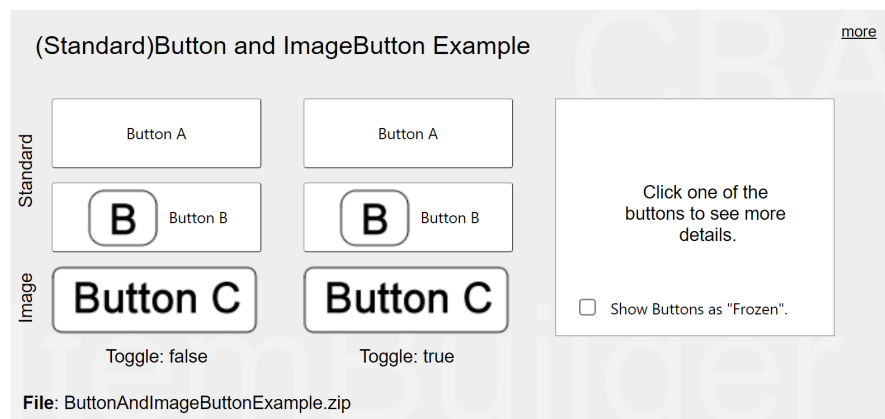


FIGURE 3.131: Item illustrating the component `Button` ([html](#)|[ib](#)).

As with all components, the context menu (right mouse button, see Figure 3.132) with the entry *Show Property View* can be used to edit properties.

Components of type `Button` can have a text property that can be edited via the context menu entry *Edit Text* (see Figure 3.132). The editor for editing the text property is identical to the editor of text properties for components of type `Link` already shown above (see Figure 3.129).

Toggle Buttons: By default, buttons can be clicked multiple times and buttons are only pressed as long as clicked by mouse or touch. However, buttons can also be configured to toggle between either pressed (i.e., *on*) and unpressed (i.e., *off*). So-called *Toggle Buttons* are defined by changing the property `Is Toggle: true`. Buttons with the `Is Toggle=true` property can remain in the pressed and non-pressed state, while buttons with the `Is Toggle=false` property automatically fall back to the non-pressed state after being pressed. *Toggle Buttons* can also be grouped by using the *Frame Select Groups* described in subsection 3.9.4 (see the item in Figure 3.92 for an example).

Standard Buttons vs. Image Buttons: Whether the `Text` property should be used

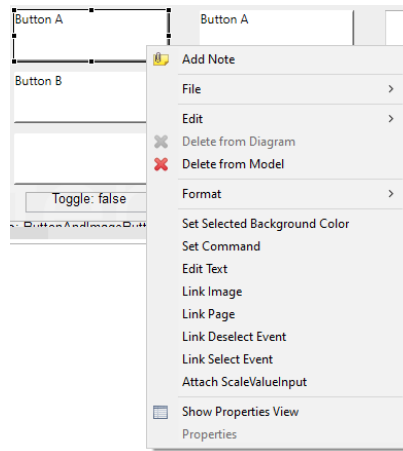


FIGURE 3.132: Context-Menu for components of type `Button`.

for components of the ‘`Button`’ type or not depends on the use of images for button design. The CBA ItemBuilder distinguishes between two configurations: *Standard Buttons* which consist of a text and an optional image, and *Image Buttons* which should be designed using images only.

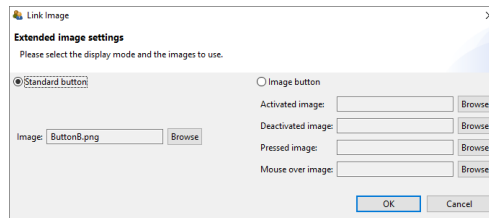


FIGURE 3.133: Component of type `Button` configured as *Standard Button*.

Figure 3.133 shows the dialog *Link Image* which can be accessed with the context menu item of the same name. The multiline text of `Buttons` can be styled using the properties font family, font size, font color, if a `Button` is configured as *Standard Button*. Moreover, border color and background color can be used to style *Standard Buttons* (see section 3.1.4). *Standard Buttons* can refer to one single image that is used as background image (see the buttons with the labels “Button B” in the example item provided in Figure 3.132).

If components of type `Button` are configured as *Image Button* (see figure 3.134), then four different images can be specified:

- **Activated Image:** This image is displayed when the button is active, i.e., it can be pressed. For buttons with the property `Is Toggle=true`, this image is displayed when the button is not toggled.

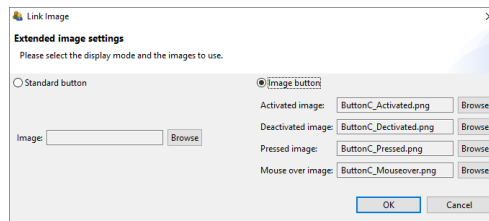


FIGURE 3.134: Component of type `Button` configured as *Image Button*.

- **Deactivated Image:** This image is displayed when the button is frozen. A button can be frozen at design time using the property `Is Frozen=true` or at runtime using the `setFrozen()`-operator. By default buttons are created with the property `Is Frozen=false` (identical to the `unsetFrozen()`-operator).
- **Pressed Image:** While a button is pressed, this image is displayed. For buttons with the property `Is Toggle=true` this image is displayed when the button is toggled.
- **Mouseover Image:** This image is displayed, when a mouse as pointing device is moved over a button.

Using the *Context-Menu* (see Figure 3.132 the background can be configured via the entry *Set Selected Background Color*) for standard buttons with the property `Is Toggle=true` (see Figure 3.135).

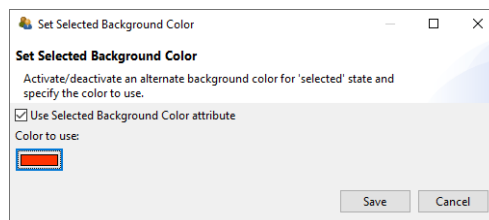


FIGURE 3.135: Dialog *Set Background Color* for Buttons (*Standard Buttons*).

Buttons are always defined as rectangles. If rounded corners are required, transparent images can be used, when the button is configured as *Image Button*. Components of type `Button` can be set frozen using the operators `setFrozen()` (and the revers operator `unsetFrozen()`, see appendix B.2) using either the *Finite-State Machine* (see section 4.4), *Conditional Links* (see section 4.3), or the *Task Initialization* (see section 4.5). The checkbox “Show Buttons as Frozen” in the example item in Figure 3.131 illustrates how frozen components of type `Button` look like.

Similar to components of type `Link` (see subsection 3.11.1) provides the context menu an entry *Link Page* (see Figure 3.132) that can be used to define regular (static) *Links* or to assign *Conditional Links* (see section 4.3) to components of type `Button`.

Buttons can be used in the CBA ItemBuilder when designing assessment components for very different purposes. Buttons can not only *Link Pages*, as described in the previous section 3.11.1 and switch between pages or open dialog pages (see section 3.15). Using the entry *Set Command* of the context menu (see Figure 3.132) enables to use buttons to control the navigation between tasks, close dialog pages, or change the full-screen mode on or off using *Runtime Commands* (see section 3.12).

Buttons can also execute operators. This is either possible using *Conditional Links* (see section 4.3) or with *Events* (see section 4.4.3). *Events* can be linked to selection (see menu entry *Link Select Event*) or deselection (see menu entry *Link Deselect Event*) of buttons using the context menu (see Figure 3.132). *Deselect Event* will only trigger for buttons configured as `Is Toggle=true`. If *Events* are linked to buttons, the events can be used to trigger rules in the finite-state machine (see section 4.4 for more details).

Finally, buttons can also be used to increase or decrease the value of components of type `ScaleValueInput` (see section 4.2.2 for a description of the entry *Attach ScaleValueInput*).

Text Color for Buttons: The text inside buttons works as a link, so the general link-color settings (see section 6.3) need to be considered. To set the text color in the *Properties* view, the properties `Use Default Link Color: false` and `Use Same Color For Visited Reference: true` must be set in the *Display* section.

Advanced Button Components: Not all functionality can be realized with simple buttons. For special requirements, the CBA ItemBuilder provides additional components that can be used, for instance, on particular page types (e.g., `TaskBarButton` and `TabButton`, see section 3.13.1). Finally, so-called map-based variable displays (see section 4.2.5) can be used instead of buttons, where, for instance, the displayed image can be changed depending on a variable with the help of *ValueMaps* (see section 4.2.4).

3.11.3 Links with other Components

Not only *Links* (see section 3.11.1) and *Buttons* (see section 3.11.2) can be used to link between pages. As can be seen in the item shown in Figure 3.136, a number of other components support the functionality of linking to pages directly.

Embedded Links: Components of type `HTMLTextFields` (see subsection 3.8.2) and `TextField` (see subsection 3.8.3) can embed links. By linking pages, the CBA ItemBuilder can be used to create navigation scenarios within assessment components and to design interactive tasks. Especially in combination with the special page types for web browser environments (`web browser page` and `web child page`, see section 3.13.2) typical hypertext environments can be created and used as part of computer-based assessments under controlled conditions.

Use Links on Answer-Selection: Components of type `ComboBox` and `List` (see section 3.9.5 for more details about `ComboBox`, `ComboBoxItem`, `List` and `ListItem`) and `MenuItem`

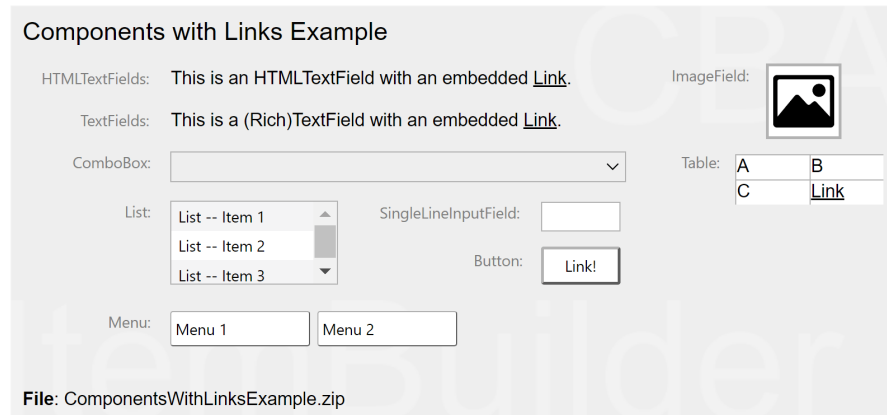


FIGURE 3.136: Item illustrating links in different components ([html](#)|[ib](#)).

(see section 3.9.7 for more details on `MenuBar`, `Menu` and `MenuItem`) can be used to link pages, when a response is given by selecting a particular item.

Use Links on Focus: Components of type `SingleLineInputField` can be used to trigger links when the components become focus. Since `SingleLineInputField` can also be defined to be `ReadOnly=true` (see section 3.9.1 for details), this can be very useful for using operators in *Conditional Links* without writing more complex *Finite-State Machine* rules (see section 4.3.3).

Change Page without Links: Not all components provide the entry *Link Page* in their context menu and can be used this way to switch to another page. Interactions with these components, for instance, mouse-clicks on components of type `ImageValueDisplay`, can nevertheless be used to switch between pages with the help of the finite-state machine. This approach based on the assignment of pages to states is described in detail in section 4.4.9.

3.11.4 Advanced Linking Scenarios

The CBA ItemBuilder offers unique flexibility in creating multi-page assessment components with different page types and various linking options. Pages can be of different types and can also be used as *xPage*, *Dialog Page* or *Page Area*. The following section is intended for advanced users of the CBA ItemBuilder who want to take advantage of the full range of possibilities when linking pages.

Link between Pages and xPages: When linking pages, it is crucial to consider the page type (see section 3.4) and the *xPage*-property (see section 3.4). For example, pages that were defined as *xPage* in the CBA ItemBuilder can only link to other *xPages*. Similarly, a page that is not defined as an *xPage* cannot link to an *xPage*. The separation of regular pages and *xPages* is also maintained for dialog pages (see 3.15

for more details on dialog pages). Again, if a dialog is linked from an *xPage*, it must necessarily be an *xPage*.

The following advanced example with two regular pages (Page1 and Page2), two *xPages* (X1 and X2), and two dialog pages (“Dialog1” and “XDialog1”) illustrates how *xPages* and *Pages* can be linked.²³

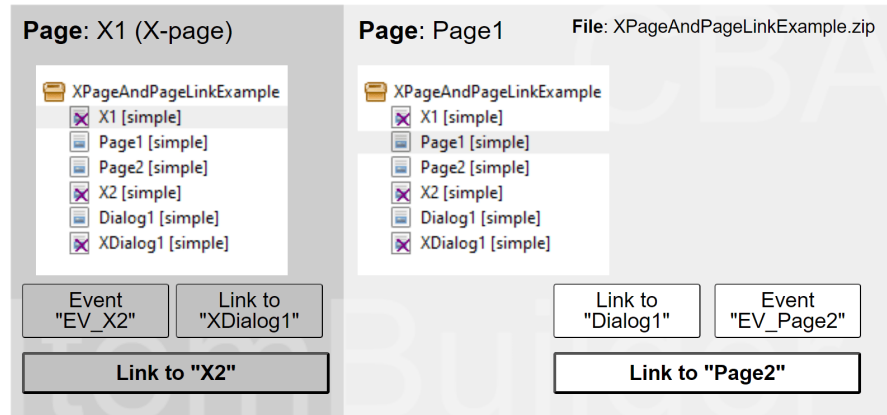



FIGURE 3.137: Item illustrating links with *xPage*-layouts ([html|ib](http://html5lib.org)).

Lets look at the button Go to "X2", which is connected to page X2 via Link Page. Since this page is an *xPage* (see the small symbol ) , the linked page is displayed in the *xPage* area of the item. Similarly, the link behind the Go to "X1" button on page X2 also causes the page X1 to be displayed in the *xPage* area. Accordingly, *xPages* are locked to remain in the *xPage* area of the item. The same is true for regular pages as well, as illustrated with the button Go to "Page2" (on Page1) and Go to "Page1" (on Page2). Components on pages that are not configured as *xPages* can link only to other regular pages.

The separation of pages by the *xPage*-flag is already implemented by the in the dialog to configuring links, i.e. depending on whether the link should be configured on a *Page* or an *xPage*, only all available *Pages* or *xPages* are listed.

The example item (see Figure 3.137 illustrates not only simple links between *Pages* and *xPages*, but also the use of links in dialog pages (see subsection 3.15 for more details) and the use of *Finite-State Machine Events* to switch *Pages* and *xPages* (see subsection 3.11.4 for more details).

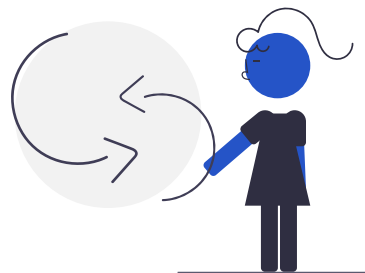
Links Between Pages of Different Type: The CBA ItemBuilder distinguishes two kinds of links: Regular (static) *Links* and *Conditional Links*. Links can be assigned to a variety of components using the context menu entry *Link Page*. Links are resolved

²³A simple example without the use of *Conditional Links* or *Finite-State Machine* can be found in the file [AdvancedLinkingExample.zip](#).

in the current environment and taking page types into account. To enable advanced linking across environments and page types, either *Conditional Links* or *Finite-State Machines* are required.

In addition to the simple links shown in the table, the `setEmbeddedPage()`-operator can also be used to link in `PageAreas` (see section 4.3.4).

3.12 Runtime Commands



Tasks created in CBA ItemBuilder project files (see section 3.6) are to be used with an execution environment, i.e., using a particular deployment software (see chapter 7). In most cases, deployments are a set of CBA ItemBuilder tasks, administered in a linear sequences. An authentication (i.e., login) of the test-takers is often required first before the items are shown in the defined sequence. Regardless of the administration mode, items are rendered in the test-takers web browser or in a browser-component provided as part of the deployment software.²⁴

Since the CBA ItemBuilder allows creating complex items with multiple pages, it is essential to distinguish the navigation *within Tasks* (implemented inside the CBA ItemBuilder) and the navigation *between Tasks*, which is handled by the deployment software. Both parts are linked with so-called *Runtime Commands* (short: *Command*), which the item author can define within CBA ItemBuilder tasks to allow test-takers to end working on a particular task.



Runtime Commands are used to send instructions to the deployment software. In order for a test participant to finish processing a CBA ItemBuilder task, at least one `NEXT_TASK` command is required in each task (see subsection 3.12.1).

²⁴A browser-component is an HTML browser control embedded in other software such as an Electron application.

The different *Runtime Commands* can be assigned to `Buttons` (see 3.11.2) and to `MenuItems` (see 3.9.7) using the context menu in the *Page Editor* (see Figure 3.138).

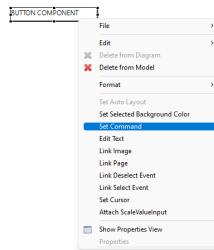


FIGURE 3.138: Context menu to assign a *Runtime Command*.

The dialog that opens (*Set Runtime Command*, see Figure 3.139) allows you to select one of the available *Runtime Commands*. Exactly one command (or `EMPTY` for none) can then be assigned to the currently selected `Button` or `MenuItem` (default is `EMPTY`).

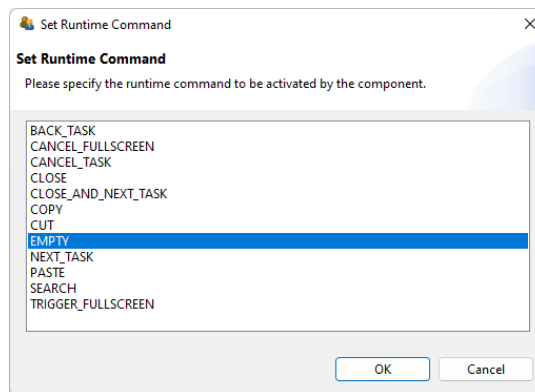


FIGURE 3.139: Dialog *Set Runtime Command*.

After specifying which command should be triggered by click or touch on the selected `Buttons` or `MenuItems` by selecting an entry in the dialog *Set Runtime Command* the button `OK` is used to conform the configuration. If a command is already assigned, it can be removed by selecting the command `EMPTY` or by deselecting the selected entry in the *Set Runtime Command* dialog.

Runtime Commands can also be defined directly in the *Properties* view (see property `Command` in Figure 3.140). The entry `EMPTY` can be selected to remove a defined *Runtime Command*.

Runtime Commands can also be triggered from the dynamic content of items (see subsection 4.4.6 for specific finite-state machine operators that trigger commands).

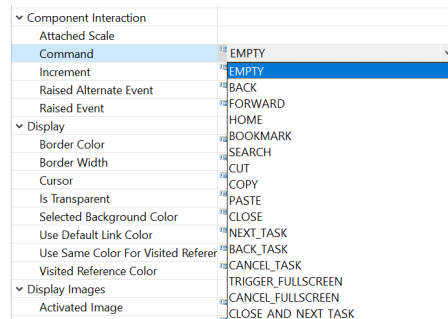


FIGURE 3.140: Property *Command* in section *Component Interaction*.



Some commands refer to the higher-level navigation between tasks and need to be tested not only in the *Preview* (see section 1.4) but also in the deployment software (see chapter 7).

3.12.1 Task-related Commands

A *Task* presented at runtime is always expected to be ended when one of the task-related commands is executed. Task-related *Runtime Commands* provide the possibility to end an assessment component form within the item.

NEXT_TASK: Ending a task and navigation to the next task is requested with the NEXT_TASK-command, that can be assigned, for instance, to a component of type Button.



A NEXT_TASK-command (either linked to a Button as described above in section 3.12 or triggered in the *Finite-State Machine* (see section 4.4.6) is required for each *Task* if the test-taker should be able to end the *Task*. Typically only tasks ended by a planned timeout implemented in the deployment software or tasks ended by an external event triggered by an interviewer or test-administrator have no NEXT_TASK-command.

If no next *Task* is defined within a CBA ItemBuilder *Project Preview* or a *Task Preview* was requested, the NEXT_TASK-command will trigger the message illustrated in Figure 3.141.

BACK_TASK / CANCEL_TASK: For controlling the flow between tasks in more depth, deployment software (see chapter 7) and the *TaskPlayer-API* (see section 7.7) can additionally support the following commands:

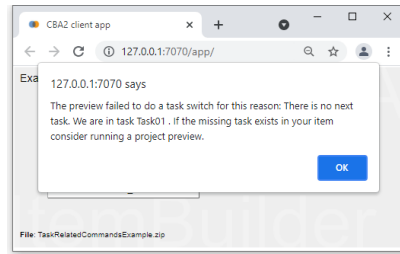


FIGURE 3.141: Preview message when calling the `NEXT_TASK`-command in the last *Task*.

- `BACK_TASK`: Navigates to the previous task
- `CANCEL_TASK`: Cancels the current part of an assessment

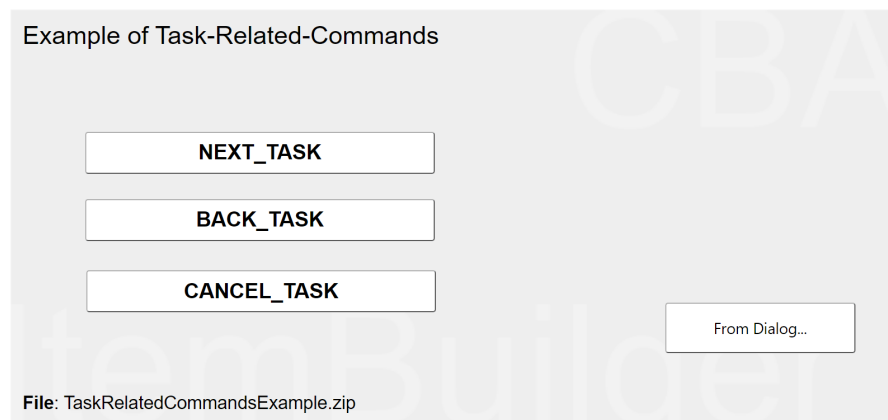


FIGURE 3.142: Item illustrating *Task-related Commands* ([html](#)|[ib](#)).

3.12.2 Dialog-related Commands

Pages can be used to design *Dialog* pages and *Popups* windows (see section 3.15) by configuring properties of the `Frame` (see Figure 3.156 for examples).

CLOSE / CLOSE_AND_NEXT_TASK: A specific `CLOSE`-command can also be used to close a currently open dialog page. The runtime command can be used to close a dialog page. Regular dialog pages can also be closed using the small “X” in the header if the attribute `Closable=true` is used. If `Closable=false` is configured, then the small “X” is not displayed in the header. Finally, the runtime command `CLOSE_AND_NEXT_TASK` combines `CLOSE` and `NEXT_TASKS`.

3.12.3 Fullscreen-related Commands

The presentation of assessment content created with the CBA ItemBuilder is done by deployment software (see chapter 7). It can be done in standard web browsers or with a specific web browser component (embedded into a deployment software and providing additional test security, for example, a kiosk mode; see section 7.2.3 for more details). The CBA ItemBuilder provides simple *Runtime Commands* to switch the display to full-screen mode intended for use directly in standard web browsers.

TRIGGER_FULLSCREEN/CANCEL_FULLSCREEN: The item shown in Figure 3.143 illustrates the use of the commands `TRIGGER_FULLSCREEN` and `CANCEL_FULLSCREEN`. If the execution environment supports it, the command `TRIGGER_FULLSCREEN` can be used to switch to full screen mode. Similarly, the full screen mode can be exited with the command `CANCEL_FULLSCREEN`. Depending on the integration of the item (see chapter 7) in a deployment system or task execution environment, these functions are identical to the full screen mode of the browser, or result in only the iframe containing the actual item being displayed in full screen.

Example of Fullscreen-Commands

TRIGGER_FULLSCREEN

CANCEL_FULLSCREEN

File: FullscreenCommandsExample.zip

FIGURE 3.143: Item illustrating fullscreen-related *Commands* ([html](#)|[ib](#)).



Whether the fullscreen-related commands can be used within the CBA ItemBuilder projects or whether the control over the fullscreen mode must be implemented by the execution environment depends on the deployment strategy and the deployment software used (see section 7.2.3 for more details).

3.12.4 Clipboard-related Commands

The commands `COPY`, `CUT` and `PASTE` allow to create buttons that can be used at runtime (i.e., for test-taker) to access the clipboard while answering tasks that use com-

ponents of type `InputField`. The example item shown in Figure 3.144 illustrates this functionality.

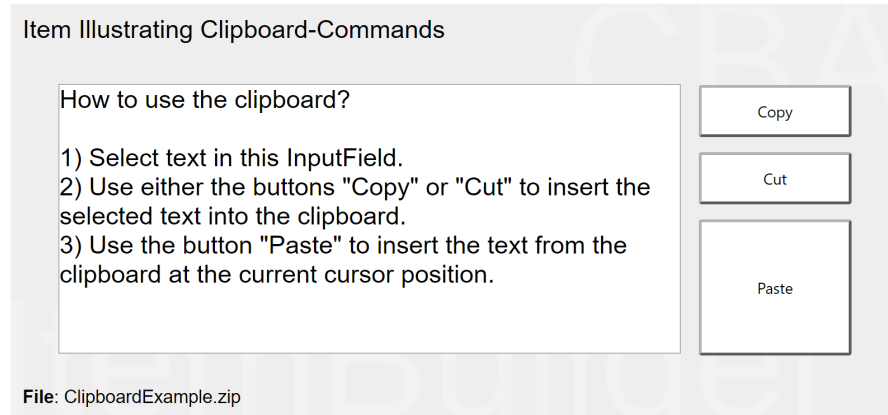
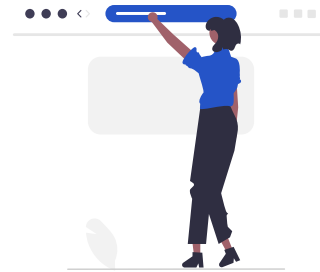


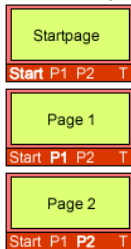
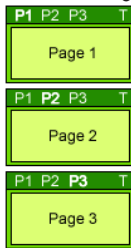
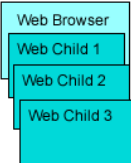
FIGURE 3.144: Item illustrating clipboard-related *Commands* ([html](#)|[ib](#)).

3.13 Components for Special Page Types



Regular pages of type `simple page` were described in section 3.4.1. As shown in Table 3.10 additional page types are available in the CBA ItemBuilder for specific purposes. These more specific pages types are listed in Table 3.10 and the components that can only be used with pages of that particular types will be described in subsections 3.13.1 and 3.13.2.

TABLE 3.10: Overview of advanced page types of the CBA ItemBuilder

| Page Type | Description |
|---|--|
| Task Bar Page  | <p>Several pages of type <code>simple page</code> can be connected with a <i>Task Bar Page</i> in such a way that the test-taker can switch between pages as if they were fullscreen applications. Buttons are associated with pages, and a home page is displayed when the button of type <code>TaskbarStartButton</code> is clicked, or when an activated button of type <code>TaskbarButton</code> is clicked again. Different layouts can be realized by altering the position of the components of type <code>TaskbarGroup</code> and <code>ChildArea</code>.</p> |
| Tab Folder Page  | <p>Several pages of the <code>simple page</code> type can be connected to a <i>Tab Folder Page</i> in order to simulate multiple tabs. Unlike the <i>Task Bar Page</i>, however, all pages are equal and remain open when clicked again. Different layouts can be realized by altering the position of the components of type <code>TabfolderGroup</code> and <code>ChildArea</code>.</p> |
| Web Browser Page & Web Child Page  | <p>A page type that acts as a frame for several subordinate pages is a <i>Web Browser Page</i>. A page of type <code>web browser</code> has an area in which a page of type 'web child' can be displayed. Transitions between <code>simple pages</code> and <code>web browser-pages</code> are not possible via links, but require the finite-state machine. Hence, the two page types are mainly used for specific task layouts with browser-like navigation between (simulated) web-pages.</p> |

For each page of a particular type, components of type `Frame` act as root element. As shown in Figure 3.145, `Frame` within pages of type `Simple Page` can contain `Panels` (see section 3.5.2) and `PageAreas` (see section 3.5.4). The hierarchy shown in Figure 3.145 illustrates the general concept of containers and nesting elements within container implemented in the CBA ItemBuilder (see section 2.11.4).

Figure 3.145 also shows that one specific component is required as the root element for all other pages. Pages of type `Tabfolder Page` need a `TabfolderFrame`, pages of type `Taskbar Page` are designed using a `TaskbarFrame` as the root element. The frame elements `WebBrowserFrame` and `WebChildFrame` are used only for pages of type `WebBrowser Page` and `WebChild Page`.

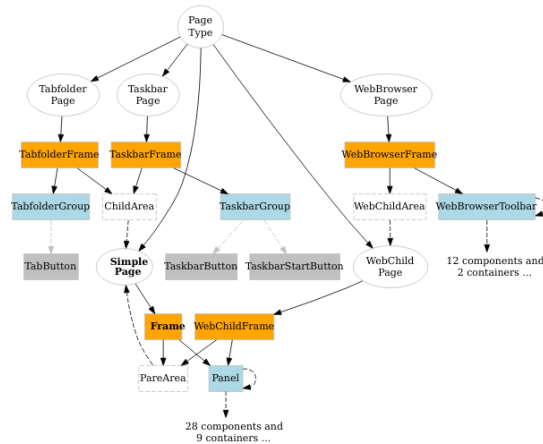


FIGURE 3.145: Hierarchy of components for different Page Types.



Depending on the *Page Type*, the appropriate top-level component must be used (i.e., *Frame*, *TabfolderFrame*, *TaskbarFrame*, *WebBrowserFrame* OR *WebChildFrame*). The CBA ItemBuilders context-sensitive *Palette* (see section 3.1.3) automatically shows the element that can be used as the top-level component if a page of a particular *Page Type* is clicked in the *Page Editor*.

Frames are containers that can host the components shown in Figure 3.145. The child elements of frames can themselves be containers (i.e., *Panel*s for *SimplePages* or the equivalent containers shown in blue for other page types) or components that embed other pages (*PageAreas*, *ChildAreas* OR *WebChildAreas*).

3.13.1 Tabfolder and Taskbar Pages

The page types *tabfolder* and *taskbar* are special pages for implementing navigation within items with multiple pages. Both page types require an area for displaying simple pages (see *ChildArea* in Figure 3.145) and an area for displaying the navigation buttons (each button is assigned a page via a link). The main difference between the two *Page Types* is the behavior of the buttons.

Tabfolder: An item using a page of type *tabfolder* with three tabs (*page1*, *page2* and *page3*) is shown in Figure 3.146. A page of type *tabfolder* can contain a component of type *TabfolderGroup* and a *ChildArea*. Only these two component types can be added to *TabfolderFrames* (i.e., the root elements of pages of type *tabfolder*). The *ChildArea*

links to a first page of type `simple page` and the `TabFolderGroup` can contain additional `TabButton`-components. The `TabButton` link to a `simple page`, so that clicking the `TabButton`-components changes pages in the `ChildArea`.

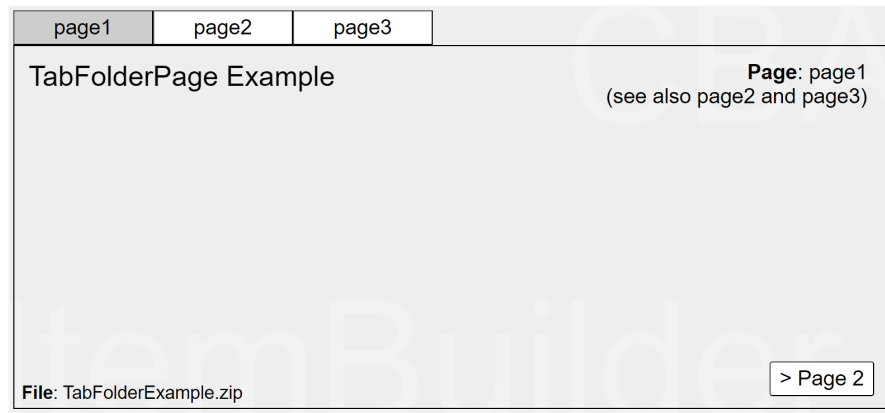


FIGURE 3.146: Item illustrating page type `TabFolder` ([html](#)|[ib](#)).

Taskbar: If an already selected button in a *Tab Folder Page* is clicked again, nothing happens. In contrast, clicking on an already selected button in a *Task Bar Page* causes the default page to be shown. This default page is called the start page in *Task Bar Pages*, linked to the corresponding button of type `TaskBarStartButton`. An item using a page of type `taskbar` with three pages (startpage, page1 and page2) is shown in Figure 3.147.

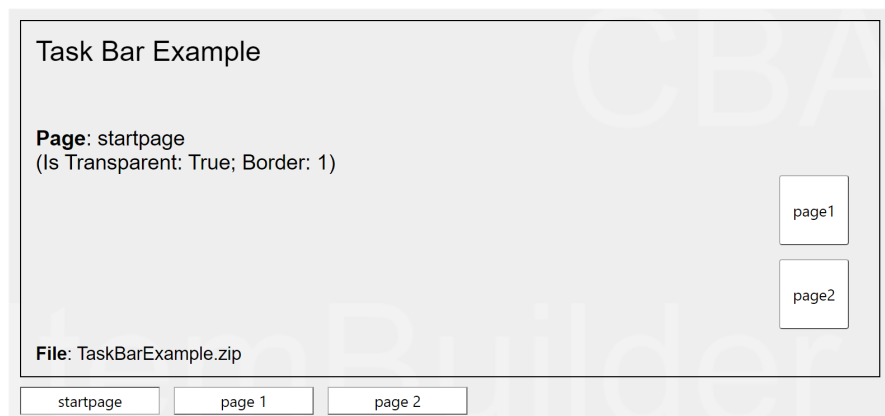


FIGURE 3.147: Item illustrating page type `Taskbar` ([html](#)|[ib](#)).

The contents of pages of the types `tabfolder` and `taskbar` are created as a module of

the type `simple page` (see section 3.3) and bound to buttons via links (see section 3.11). Further links between `simple pages` are also possible.

It is important to point out that the special page types `tabfolder` and `taskbar` are only simplifications for functionality that can also be implemented with `PageAreas` and the dynamic part introduced in chapter 4.

3.13.2 Browser Pages (`Web Browser` and `Web Child`)

The CBA ItemBuilder also supports the implementation of simulated hypertext environments with the page types `Web browser` and `Web child`. A page of type `Web browser` forms the outer frame on which typical browser buttons, an address bar, and an area for displaying pages of type `Web child` can be placed as components.

An example illustrating an assessment component implemented in the CBA ItemBuilder using *Browser Pages* can be found in Figure 3.148.

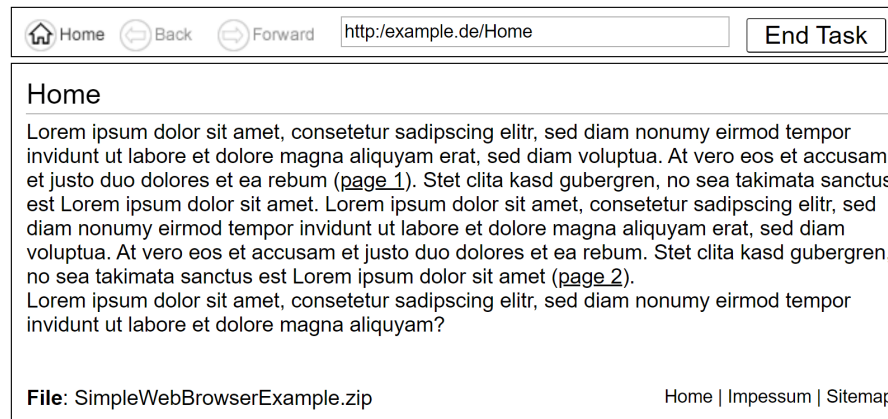


FIGURE 3.148: Item illustrating page types `WebBrowser` and `WebChild` ([html|ib](#)).

Web Browser: To create a simulated hypertext environment, a page of type `web browser` can be used that allows to host two types of components. The `WebChildArea` define the place at which the embedded pages of type `web child` are shown. The `Web-BrowserToolbar` component hosts the specific browser buttons that simplify the implementation of simulated hypertext environments. *Browser* pages bring a built-in browsing history, buttons for forward and back navigation, a home button and a bookmark function. Figure 3.149 summarizes all components that can be added to `WebBrowserToolbars`.

Often, the actual task or instruction should be displayed in addition to the simulated web environment. This example illustrates the use of *xPage* layouts well. The item in Figure 3.150. uses an *xPage* to display the *Task* and next to it a page of type `web`

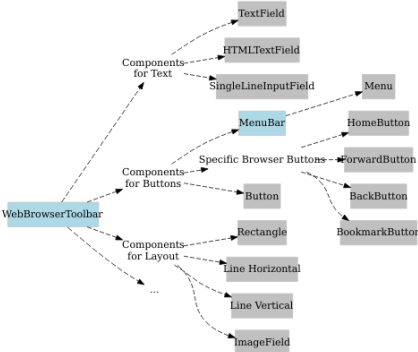


FIGURE 3.149: Overview of components for web browser.

browser. The actual simulated hypertext pages of type `web child` are embedded in the `Web browser page` via a component of type `WebChildArea`.

Task 1

Use the information on the website to answer the following question:

How often is the word "impsum" mentioned in the filler text?

times.

End Task

Home Back Forward http://www.example.org/Home

Home

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum (page 1). Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet (page 2).

File: WebBrowserAndXPageExample.zip Home | Impessum | Sitemap

FIGURE 3.150: Item illustrating ‘xPage’-layout with browser pages (html|ib).

Deprecated Features / Features under Development: The components of type `Web-BrowserFrame` (i.e., the root elements for pages of type `web browser`) provide the property `Is Tabbed` that can be used to implement tabbed browsing. However, the layout of tabbed browsers cannot be configured entirely, and links always open a new tab (hence, making the navigation buttons of `WebBrowserToolbars` obsolete).²⁵

²⁵ See more examples in file [TabbedWebBrowserExample.zip](#).

3.14 Embedding HTML Content

3.14.1 Component `ExternalPageFrame`



The CBA ItemBuilder allows the inclusion of HTML content in pages using the `ExternalPageFrame`-component. There are two options provided how HTML content can be included. Either, the HTML page and all used resources (e.g. images, audio/video files, CSS style sheets and JavaScript files) are added to the item project as embedded resources (*Local*). In this case, the CBA ItemBuilder items are delivered as *internal* resources. Embedded resources can also be used, for example, in an offline delivery (see section 7.2.2). Or the external contents are only integrated via a URL (*External*). In this case, it is necessary that the embedded resources are available at runtime and can be loaded from the browser via the Web. `ExternalPageFrames` can be defined to be transparent.

Components of type `ExternalPageFrame` can be selected from the palette as described above (3.7) and added to a container, e.g. a `Panel`. Size and position of embedded HTML content can be defined in the visual designer (see 3.7.1). The HTML content is assigned to the component via the context menu using the entry `Set Page Address` (see Figure 3.151).

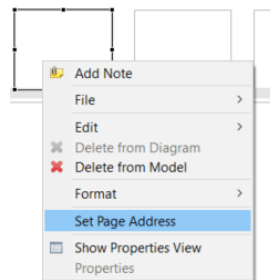


FIGURE 3.151: Context menu of `ExternalPageFrame`-components.

The dialog *Set the URL* allows specifying from which URL the content for this component is provided (see Figure 3.152). A *Local* URL can be defined if the content is provided as part of the item (included using the *Embedded HTML Explorer*), or the content references an *External* URL that needs to be accessible at runtime (see section 7.2.2).

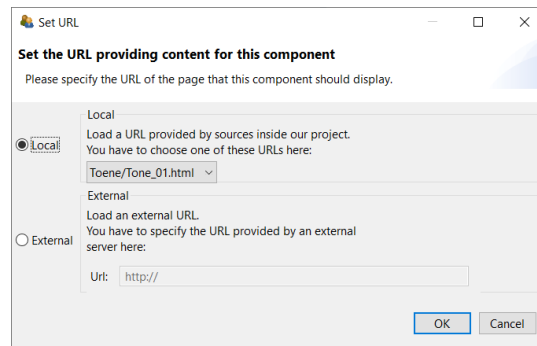


FIGURE 3.152: Dialog *Set URL* to configure `ExternalPageFrames`.

Regardless of the source from which the embedded HTML content is obtained at runtime, it is possible to exchange information between the embedded HTML content and the item. This is described in section 4.6.2. In the design view of the CBA ItemBuilder, the embedded HTML content is not displayed, that is, it will be visible only when previewing the item (see 1.4).

3.14.2 Using the Embedded HTML Explorer

To manage the local resources (i.e., the content in static files²⁶ that can be delivered with the item project), the CBA ItemBuilder provides the *Embedded HTML Explorer*. The HTML Explorer provides all necessary functions to add files as *external-resources*, organize them into folders, delete them, rename them and export them from the item project, if necessary (see context menu, shown in Figure 3.153).

To display the content of an HTML, CSS or JavaScript file, the entry *Open* or *Open With System Editor* can be selected. The screenshot in Figure 3.154 shows an HTML file opened in the *HTML Explorer* of the CBA ItemBuilder.

A warning is displayed if you try to open a file that cannot be opened in HTML Explorer (see Figure 3.155).

²⁶By default the CBA ItemBuilder supports *Text files* (.htm, .html, .css, .scss, .less, .js, .txt, .json), *Images* (.gif, .jpg, .jpeg, .min.png, .bmp) and *Media Files* (.mp3, .flv, .mp4, .webm, .ogg, *.wav) as external resources.

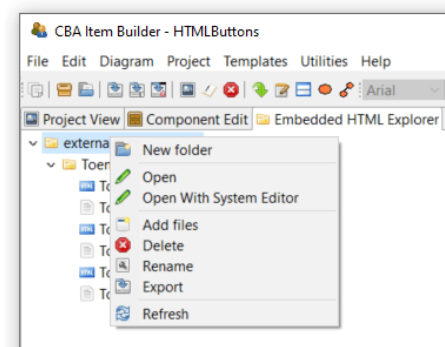


FIGURE 3.153: Context menu of the *Embedded HTML Explorer*.

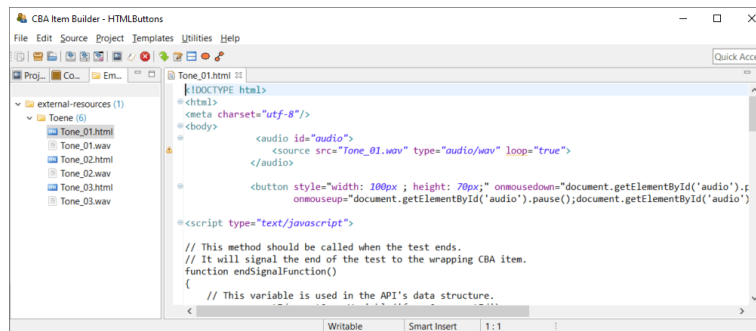


FIGURE 3.154: HTML document opened in an *HTML Editor*.



It is important to note that content inserted via `ExternalPageFrames` as HTML and JavaScript cannot be used for long-term storage and archiving of items and must be tested independently of the browsers supported by the CBA ItemBuilder.

Caution Note about the use of HTML5 and JavaScript in `ExternalPageFrames`: As described in section 2.11.2, the CBA ItemBuilder separates the assessment content and the technical implementation required for rendering the items. This separation is impossible when assessment content is only embedded with `ExternalPageFrames`. Ac-

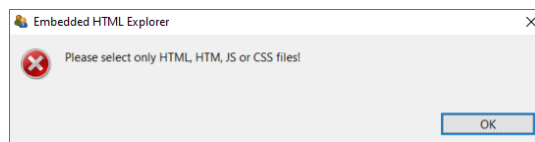


FIGURE 3.155: Warning shown by the *Embedded HTML Explorer*.

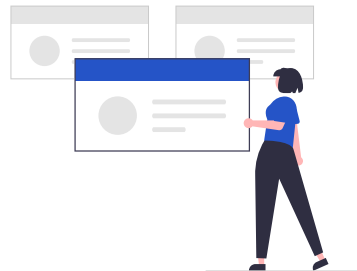
cordingly, it is not possible to adjust the display of items by updating the CBA Item-Builder, for example, if functionality can no longer be used in new browsers or errors are detected in the embedded content. Due to this restriction, the use of HTML5 and JavaScript in `ExternalPageFrames` is suggested only in scenarios where resources to update or modify the embedded content are available at the time when items are created and as long as the items are planned to be used or maintained (see also the list of pros and cons in section 4.6.1).

Pros and Cons of Embedded HTML Content: The use of embedded content can offer significant advantages, as it allows to implement item formats, dynamic behavior and interactivity within CBA ItemBuilder items of any kind, including the use of advance libraries, web assembly code and more. However, this huge flexibility also comes with limitations. Pros and Cons are briefly summarized in the following list:

- **Pro:** `ExternalPageFrames` can link to HTML content distributed with the CBA Item-Builder *project file* and to content included in online deliveries directly from external sources. If, for example, content can only be included by external links for legal reasons, the `ExternalPageFrame` is essential. Of course, if content is only integrated via URL, the delivery must be online and allow access to online resources. If the content is embedded as `ExternalPageFrame` is part of the CBA ItemBuilder *project file*, it will be deployed using the directory `external-resources` (similar to the resources used by the item, deployed using the directory `resources` of the *project file*).
- **Pro:** Parts and components that are difficult or impossible to implement with the objects and techniques of the CBA ItemBuilder can be quickly taken over from existing Web applications or easily programmed in HTML5 (i.e., with HTML and JavaScript). The possibility of including and integrating content as `ExternalPageFrame` within CBA ItemBuilder projects is therefore responsible for this tool's flexibility, extensibility, and openness.
- **Pro:** If assessment components already exist as HTML programs and do not require long-term maintenance, integrating these components as `ExternalPageFrame` is useful to enable the reuse of existing content. If more complex ideas are to be tried and tested, rapid prototyping practices can also be implemented using `ExternalPageFrames`.
- **Con:** Assessment content created with the pre-built components of the CBA Item-Builder automatically provides log events (see section 2.8). If external content is only embedded in assessments using `ExternalPageFrames`, the programming of the external content must independently capture necessary log events and integrate them into the log data of the test delivery using the provided interface (see section 4.6.2).
- **Con:** Content that is to be used via `ExternalPageFrames` in CBA ItemBuilder project files must be tested with regard to its display in all planned browsers (cross-browser testing, see section 8.4.1).

- Con: The assessment components created with the CBA ItemBuilder are currently executed in HTML environments. From the model defined by item authors (see section 2.11.2), a JSON configuration is generated for execution in the browser, which can be executed and displayed by a JavaScript runtime. This JSON configuration is updated by opening a *project file* of an older version of the CBA ItemBuilder in the latest version. Older contents (in the form of CBA ItemBuilder *project files*) then profit from possible further developments and error corrections, e.g., with regard to the requirements of modern browsers. This process forms the basis of the *migration* strategy (see section 3.2.1 and also section 8.7.3) and does not apply to content inserted directly as HTML / JavaScript via the `ExternalPageFrames`.
- Con: From the perspective of a tool for creating computer-based assessments by item authors, probably the most significant challenge for the long-term use of components as embedded HTML resources is the care and maintenance of these components. Depending on the integrated components' complexity, questions of knowledge management, and documentation of the source code may become necessary or relevant.

3.15 Pages as Dialogs (Popups)



The CBA ItemBuilder allows showing pages as *Dialogs* (Popups). Dialogs are usually designed using pages of type `Simple Pages` (see 3.4.1), and dialog-pages support all features and components that are available for their particular *Page Type*. Figure 3.156 illustrates the use of dialog pages for various purposes.

As shown in Figure 3.157, the definition of a particular page as dialog can be configured in the *Properties* view. To define a page as dialog page the property `Dialog` (in the section `Misc`) of a page's `Frame` must be specified either as `DIALOG` or `MODAL_DIALOG`²⁷.

²⁷By default, all frames are configured as `UNDEFINED`. Accordingly, to change a `Frame` configured as dialog back a regular page, select `UNDEFINED` as value of the property `Dialog`.

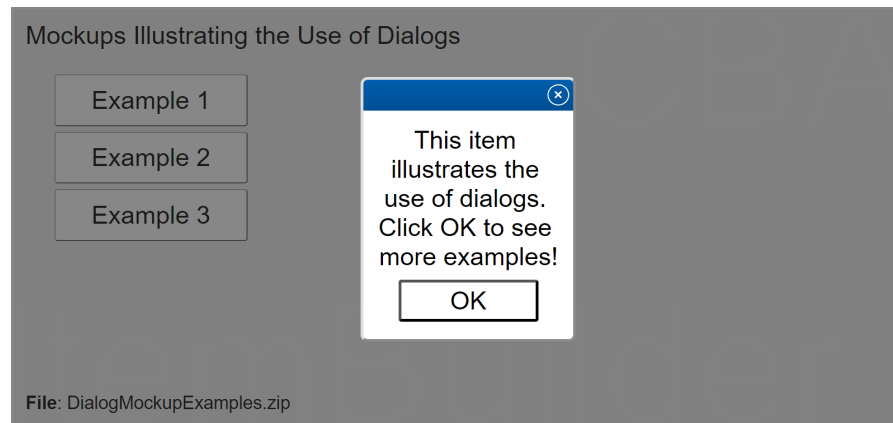


FIGURE 3.156: Item illustrating dialog pages ([html](#) | [ib](#)).



FIGURE 3.157: Definition of a `Frame` as `DIALOG` in the *Properties* view.



Pages can be configured as dialogs in the CBA ItemBuilder by setting the 'Dialog' property to the value 'DIALOG' or 'MODAL_DIALOG'.

3.15.1 Properties of Dialogs

The creation of dialog pages with the CBA ItemBuilder is done via properties of the 'Frame.' The dialog type (regular dialogs vs. modal dialogs), the display of a header with a button to close the dialog (closable dialogs), and the start position for dialog pages can be configured.

Modal vs. Non-Modal Dialogs: The CBA ItemBuilder distinguishes between two kinds of dialog pages (`DIALOG` and `MODAL_DIALOG`). A modal dialog is displayed exclusively on the screen. The content of the current page is still displayed as the underlying page, but is darkened. In contrast, a non-modal dialog allows further interaction with the content of the underlying page.

Figure 3.158 illustrates the difference between *Modal* and *Non-Modal* (regular) di-

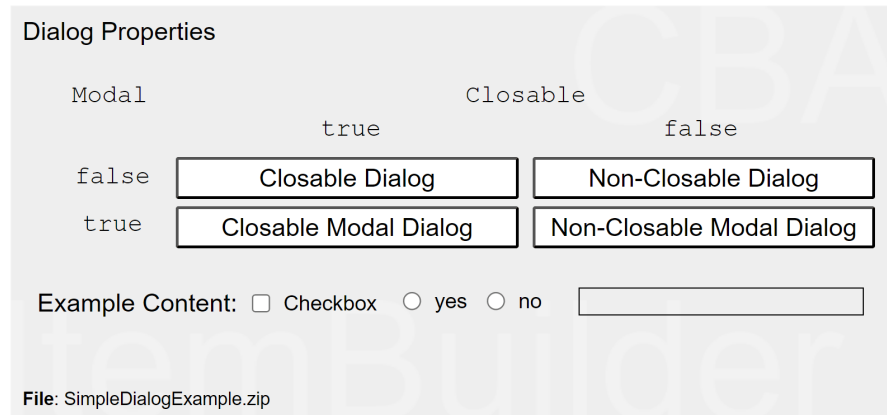



FIGURE 3.158: Item illustrating modal dialogs ([html](#)|[ib](#)).

dialogs.²⁸ Modal dialog pages are especially suitable for alerts and messages boxes that must be seen in any case. The presentation of modal dialog pages allows content of the item to be blocked. In contrast, non-modal dialog pages are useful if, for example, item content should remain visible and usable, as is the case with a calculator or for digital scratch paper.

The dialog pages that can be opened in Figure 3.158 using the buttons *Closable Dialog* and *Non-Closable Dialog* are not modal. Although they hide a part of the item, the item content can still be changed in the background. The checkbox can be selected and text can be entered into the `SingleLineInputField`. This kind of interaction is not possible for the dialogs that can be called with the buttons *Closable Modal Dialog* and *Not-Closable Modal Dialog*.

Closable vs. Not-Closable Dialogs: As shown in Figure 3.158, another property determines the appearance and behavior of dialog pages: `Closable`. Dialog pages that are configured as `Closable = true` can be closed using the small icon  located in the title bar, that is added by the CBA ItemBuilder to closable dialogs. In contrast, dialogs configured as `Closable = false` don't show a title bar. If and how dialog pages with the property `Closable = false` can be closed can be defined and controlled by the item author.



Components of type `Frame` provide the property `Closable` (`true` or `false`) that can be used to configure the behavior of dialogs. This property is only relevant, if the `Frame` is configured as either as `DIALOG` or `MODAL_DIALOG` in the property `Dialog`.

Start Position of Dialog Pages: Dialog pages appear at runtime within the displayed

²⁸For more examples see the project [DialogMockupExamples.zip](#) shown above, in which all modal dialogs are marked with *M*.

area of a CBA ItemBuilder project file defined by the *CBA Presentation Size* (see section 3.2.2). Pages configured as dialogs should therefore not be larger than the *CBA Presentation Size*. The position at which dialogs are displayed is determined by the *x* and *y* coordinates of the component of type `Frame`, which is inserted into a dialog page.

Transparent Dialog Pages: For advanced designs of complex assessment components with the CBA ItemBuilder, it can also be useful to design dialog pages transparently. This functionality is only available for dialogs with the property `closable=false` and can be configured by defining both `Frame` and `Panel` as `Transparent=true` (see Figure 3.156 for an example). In combination with transparent images as background (see section 6.2.1) this allows to implement dialogs with non-rectangular shape in the CBA ItemBuilder.

3.15.2 Opening and Closing Dialogs

Pages whose `Frame` is configured as dialog (i.e., `Frames` with a value `DIALOG` or `MODAL_DIALOG` in the property *Dialog*) can be displayed using different methods.

Open Dialog via Link: The easiest way to display a dialog in the CBA ItemBuilder is to link to a page that has been configured as a dialog. Using this approach, all components that can link to a different page (see section 3.11) can be used to open and show a dialog.

Open Dialog Pages with `openDialog()`-Operator or States: Two different mechanisms are available for displaying dialog pages from within the dynamic part of CBA ItemBuilder items. As described in section 4.4.6, dialog pages can be opened using a so-called operator (the `openDialog()`-operator). When this operator is executed (either within a *Transition* or as part of a *Conditional Link*), the page with the provided *Page Name* as argument will be shown as dialog. Moreover, pages (in general) as well as pages configured as dialog pages can be assigned to *States* of the *Finite-state Machine* (see section 4.4.9). If a *State* is assigned to a pages configured as dialog, the dialog will appear as soon as the particular *State* becomes the *Current State* (see section 4.4.2).

Depending on the configuration, dialog pages can also be closed or hidden in different ways.

User Interaction to Close Dialog: Dialog pages defined with the property `closable=true` can be closed at runtime at any time using the small icon in the dialog header.

Runtime Command 'Close': Dialog pages can also be closed using the *Runtime Command* `CLOSE` and `CLOSE_AND_NEXT_TASK` (see section 3.12.2). Using this technique, as shown in the example in Figure 3.158, `Buttons` can be defined to close dialog pages. For that purpose, a runtime command is assigned to the button.

For advanced applications, it may also be necessary to track in the logic layer of the

CBA ItemBuilder whether a dialog has been closed. Such an application must prevent unsupervised closing of dialog pages with the property `Closable=false`. Components of type `Button` can then be used to close the dialog (using the runtime command `CLOSE`) and trigger an *FSM event* (see section 4.4.3, which can then be processed in the logic layer of the CBA ItemBuilder.

Close Dialog Pages with `closeDialog()`-Operator: Dialog pages with a particular *Page Name* can also be closed from the finite-state machine with the help of the `closeDialog()`-operator (described in section 4.4.6).

3.15.3 Links in Dialogs

As described above, dialog pages are opened when a link directed to them is activated. According to the separation into pages of different areas, *xPages* can refer to dialogs defined as *xPage*. Regular pages cannot link to dialogs (and pages) that are not defined as *xPage*.

However, dialogs can contain links (see 3.11. Links to other dialogs are opened as dialogs, links to regular pages are opened as regular pages. Therefore, as Figure 3.159 shows, two dialog pages can be displayed at the same time: A dialog of an *xPage* and a regular dialog.

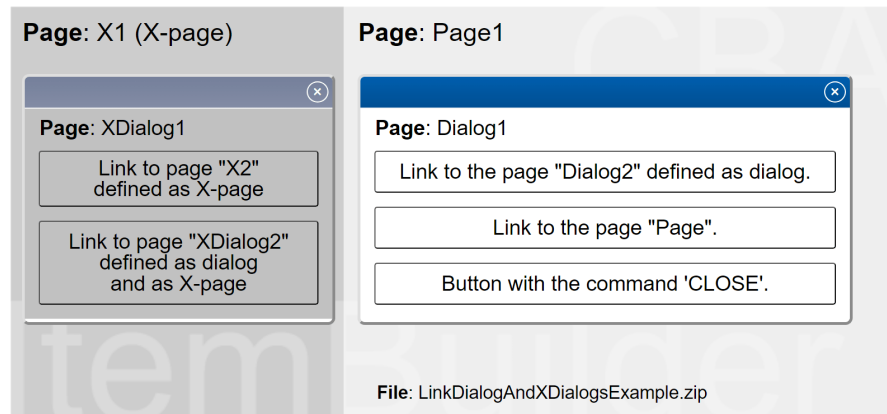


FIGURE 3.159: Item illustrating dialog pages with links ([html](#)|[ib](#)).

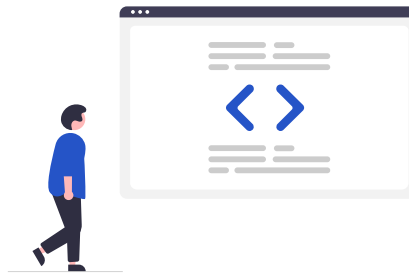


Dialog pages defined as *xPage* can link only to other *xPages* configured as dialogs. Likewise, dialogs not specified as *xPage* cannot link to other *xPages*.

The separation of regular pages (and regular dialogs) and *xPages* (and *xPage* dialogs) is maintained even if pages are linked to states (see section 4.4.9).

4

Enriching Items using Dynamic Content



The CBA ItemBuilder is an authoring tool that allows defining simple and complex items using different editors. In the previous chapter 3, the *Page Editor* was introduced for creating static content. The *Page Editor* together with the *Palette* allows to design pages uses a graphical representation of components and point & click techniques. Moreover, the *Properties* view was used to define and change detailed information for a selected component. The CBA ItemBuilder also uses text-based syntax to enrich the items with dynamic content, if necessary, for complex items.

Example: Before we describe the various uses of the syntax in detail, we start with an elementary example illustrating the CBA ItemBuilders' logical layer (i.e., the finite-state machine idea) that rests on this text-based syntax. Imagine, we want to improve the usability of a drag-and-drop item, to make the computer-based assessment more accessible to all students. For that purpose, we want to show a tiny hint on how to respond using the drag-and-drop user interaction. We use a modal dialog (see section 3.15) that is shown if a button with a question mark is clicked (see Figure 4.1). However, let's assume that we want to hide this button as soon as the student interacted with the item showing that they are familiar with the drag-and-drop technique. That means, as soon as one element was successfully moved in a drag-and-drop order interaction, we want to hide the button for requesting the hint. Accordingly, the item technically should distinguish two states: In the first condition (state `State_Hint_Visible`), we do not yet know whether the test-taker knows how to answer the question. Therefore, the hint button should be visible that explains how the items can be answered. As soon as the test-taker has shown that they can respond to the task using the drag-and-drop technique, this hint is no longer necessary. Accordingly, the item should change into a different condition (state `State_Hint_Invisible`). So in the state `State_Hint_Invisible`, the button should

be invisible, and the first drag-and-drop operation can be used to trigger the change from state `State_Hint_Visible` to state `State_Hint_Invisible`.

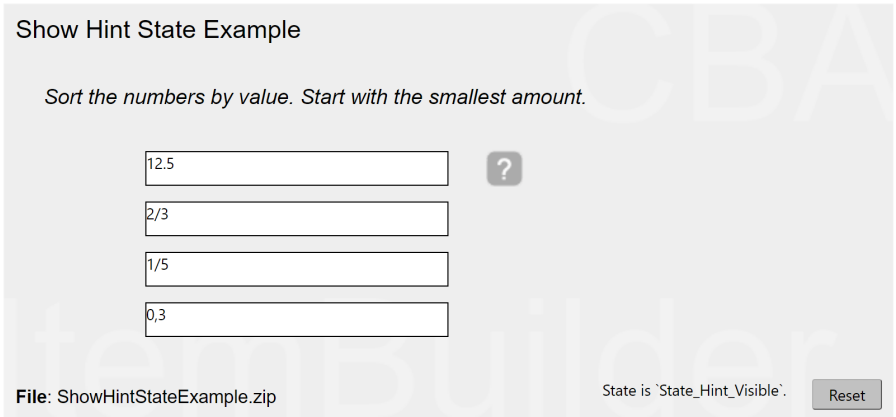


FIGURE 4.1: Example for a hint button disabled using states ([html](#)|[ib](#)).

To get an insight into the logic layer of the CBA ItemBuilder, which is designed using *Finite-State Machines*, at runtime, you can request support in the *Preview*.

State Machine Debug Window: For the dynamic content, the CBA ItemBuilder offers a built-in *State Machine Debug Window*. Similar to the *Scoring Debug Window* (see section 1.5.2) and the *Trace Debug Window* (see section 1.6.2), the *State Machine Debug Window* can be requested using a hotkey (default is `strg / ctrl + M`, see appendix B.4 for details).

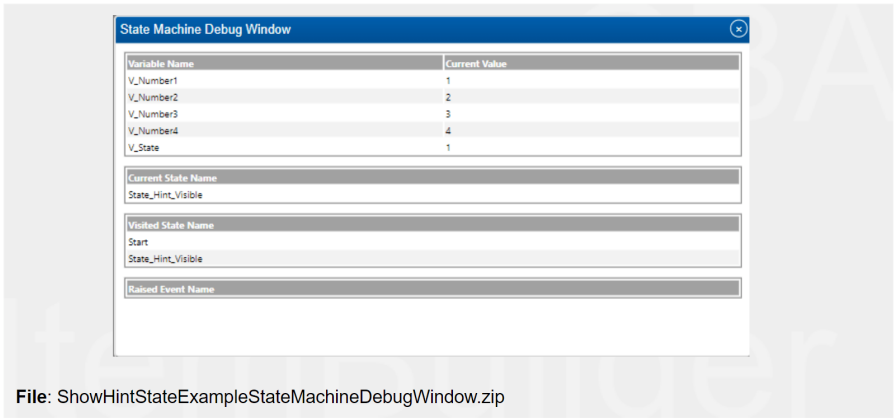
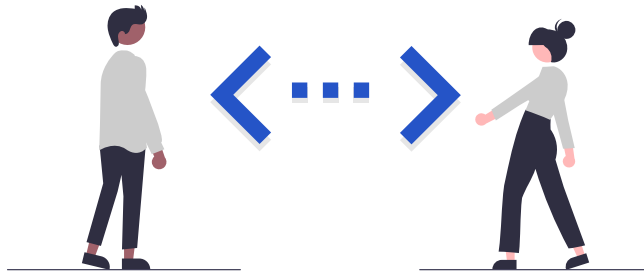


FIGURE 4.2: Screenshots of the *State Machine Debug Window* ([html](#)|[ib](#)).

The *State Machine Debug Window* shows states and variables that can determine at runtime the appearance and behavior of assessment components created with the

CBA ItemBuilder. Variables that can be defined (see section 4.2) are listed, together with the values of the variables at the time the *State Machine Debug Window* was opened. Below this, the *Current State* (for each *Finite-State Machine*, see section 4.4.2) is displayed, followed by the list of already visited *States*. Finally, the *FSM events* that have just occurred (see section 4.4.3) are displayed.

4.1 Syntax Overview



To implement dynamic item contents, the CBA ItemBuilder provides editors for various forms of text-based syntax. The different syntax types are designed in such a way that no specific programming knowledge is required and the different editors support editing and correcting syntax. Hence, although the ItemBuilder is primarily a graphical tool, some advanced features require the use of simple syntax expressions. Apart from the basic similarity of syntax variants, there are specific operators and syntax rules, which are described in the corresponding sub-sections:

- *Task Initialization* (i.e., syntax-based configuration of *Tasks*, see section 4.5)
- *Finite-State Machines* (i.e., internal logic layer of the CBA ItemBuilder capable to change behavior and visual presentation of items, see section 4.4)
- *Conditional Links* (i.e., links that change the link target according to conditions, see section 4.3)

Syntax will also be used in the next chapter to combine information from the response (e.g., component states from input components such as `RadioButtons`, `CheckBoxes`, `SingleLineInputFields`, etc.) and probably the response process (e.g., visited pages, occurred states, values of variables, etc.) to outcomes:

- *Scoring* (i.e., the definition of hits conditions using a scoring syntax, see chapter 5)

The graphical user interface allows the definition of the visible elements assessment

components such as items and instruction pages. However, computer-based assessments' full potential can only be realized with the CBA ItemBuilder by using the syntax components.

4.1.1 Basic Syntax Elements

The various definitions, which are made in the CBA ItemBuilder using a syntax that is easy to learn, are based on common design principles.

UserDefinedIds: To refer to components that have been created and placed in the *Page Editor* in the *Drawing Area*, the `UserDefineId` is always used (see section 3.7.4). Since these identifiers must always be unique within a CBA ItemBuilder *Project File*, each component can be referenced by the string that was defined as `UserDefinedId`. `UserDefineIds` can be used to name components or to query or evaluate the state of components.

If components can occur multiple times within a task, for instance, because a `PageArea` can be included multiple times (see section 3.5.4), the `UserDefinedId` of the container in which components are nested used must be included (see section 4.1.4).

Operators: In addition to the `UserDefineIds`, predefined keywords also form the syntax provided by the CBA ItemBuilder. These keywords mark *Operators* that can be used to apply changes to components or to perform more specific evaluations of component states in scoring conditions. Operators follow a common scheme: `name(arguments)`. The operators are distinguished by their `name` and the different operators can have arguments, but do not have to. The meaning of the arguments is determined thereby over the order and defined for each operator. Arguments are specified in quotation marks if they are strings. Multiple operators can be listed, depending on the context, using spaces (*Conditional Links*, see section 4.3.3; *Task Initialization* rules, see section 4.5), commas (*Actions* in finite-state machine rules, see section 4.4.6) or combined by *logical expressions* (*Scoring Conditions*, see section 5.3.2).



The exact spelling (including upper and lower case and underscores) must be followed for operators in CBA ItemBuilder syntax.

Auto-Completion: A useful helper for syntax creation is auto-completion. To enable this feature, press `Ctrl + Space` (`Strg + Space`) in a syntax editor of the CBA ItemBuilder. Available operators and existing `UserDefinedIds` are then displayed and selected by pressing `Enter`.

Auto-completion is available in the editor for creating conditional links (see section 4.3), in the editor for defining state machine events and rules (see section 4.4), in the editor for defining the task-initialization (see section 4.5), and in the editor for scoring rules (i.e., *Hit- / Miss-conditions*, see section 5.3).

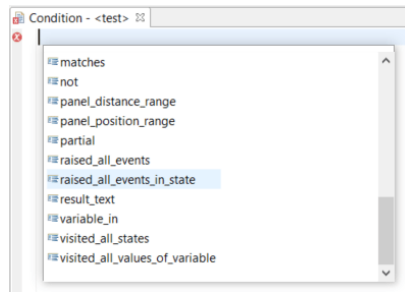


FIGURE 4.3: Screenshot of the *Auto-Completion* feature in syntax editors.



Auto-completion simplifies the definition of syntax in the CBA ItemBuilder since valid syntax elements (operators, UserDefinedIds, etc.) are automatically suggested after pressing Strg+Space / Cntr+Space.

With a small red icon, errors within syntax definitions are automatically detected and displayed.

4.1.2 Comments

In all syntax flavors *Comments* can be defined to document and describe purpose of particular part of the syntax. To mark a line as a comment in a syntax, add two slashes left to the text that should be treated as comment (e.g., (...) // My Comment). Everything in the remaining line of a syntax right to the // will be ignored.



Using the comment function to document syntax is highly recommended to structure syntax and to simplify the readability of syntax.

Multiple lines can be commented by starting either each line with // or using the syntax /* Comment (that can multi-line text) */. Everything between /* and */ is ignored as interpreted as comment, even if it spans multiple lines.

4.1.3 Logical Expressions and Bracketing

The CBA ItemBuilder syntax is not a complex programming language and requires only to learn few keywords (mainly, the so-called operators, see 4.4.6). However, three crucial points must be acknowledged to use the CBA ItemBuilder syntax capabilities efficiently, as described in the following.

Case Sensitive: All syntax statements are case-sensitive. This applies to `UserDefinedIds` (see section 3.7.4), to the operators (see section 4.4.6), the keywords `Events:` and `Rules` which are used to structure the *Finite-State Machine* syntax, and the *Logical Expressions*. Use the *Auto-Completion* (see section 4.1.1) to avoid invalid syntax due to wrong capitalization.

Logical Expressions: The CBA ItemBuilder can logically link or negate statements in the different syntax variants, e.g., logical conjunction (`and`), logical disjunction (`or`) and logical negation (`not`). For the formulation of logical expressions, the keywords `true` and `false` are additionally available. For example, if `true` is specified as a condition, this condition is always fulfilled.



Logical expressions must always be written in brackets with two components (pairs)!

Bracketing: Logical expressions in the syntax of the CBA ItemBuilder must always be formulated in such a way that not more than two expressions occur within one pair of brackets. For more than two statements, this requires nesting the statements, as illustrated with the following examples:

- Logical conjunction of two conditions: `(Condition1 and Condition2)`
- Logical disjunction of two conditions: `(Condition1 or Condition2)`
- Logical negation of a condition: `not Condition`
- Logical conjunction of three conditions: `((Condition1 and Condition2) and Condition3)`
 - Can be considered as two conjunction A: `(Condition1 and Condition2)`, B: `Condition3`,
 - combined to `(A and B)`.
- Logical negation of a conjunction of three conditions: `not ((Condition1 and Condition2) and Condition3)`
 - Can be considered as two conjunction A: `(Condition1 and Condition2)`, B: `Condition3`,
 - combined to C: `(A and B)`, and
 - negate as `not C`.
- Logical negation of a conjunction of three conditions (two with negation): `not ((Condition1 and not Condition2) and not Condition3)`

The CBA ItemBuilder has a built-in syntax checker that identifies errors with small icons and mouse-over texts. Writing syntax in several lines and using comments can help write maintainable valid syntax for complex statements.

4.1.4 UserDefinedIds of Nested Pages in Syntax

Special rules apply, if pages are nested. For instance, if components of the type `PageArea` are used for the design of pages, then the components defined on the embedded page can appear several times within a *Task* because using the same embedded page multiple times is possible. This means that the `UserDefinedId` is defined for the embedded page are no longer sufficient to uniquely identify a particular component. However, each `PageArea` itself has its own `UserDefinedId`. In order to be able to use components on nested pages such as `PageAreas` in the syntax of the CBA Item-Builder, the `UserDefinedId` of the container (i.e., the `PageArea`) must be used as a prefix:



References to components on embedded pages such as `PageAreas` must be specified in the form `UserDefinedId_of_the_Container.UserDefinedId_of_the_Component`.

The syntax editor does not check if a component is on a page referenced by a `PageArea`. Ignoring this rule is a common source of errors for conditions and operators in conditional links (see section 4.3.4), finite-state machine operators (see section 4.4.6), and scoring (see section 5.3.9).

4.1.5 Argument Indices

At selected points in the syntax, the CBA ItemBuilder allows so-called argument indices. For example, the operator `trace_text(Argument)`, which can be used to insert a text into the trace log of the CBAItemBuilder, can be used with an argument list (see section 4.4.6). If only a string argument is used, for instance, the string `My custom log information` can be inserted into the log data:

```
trace_text("My custom log information")
```

If an argument index is used in the first string argument with the syntax `%{Index}$s`, values of variables can be inserted into the created log entry. With this syntax any number of additional values can be inserted into the first string argument, based on an index starting with 1. In the following example, the values of the *Variables* (see section 4.2) `myVar1` and `myVar2` are used:

```
trace_text("My custom log with value %1$s for variable myVar1 and
          value %2$s for variable myVar2", myVar1, myVar2)
```

When this operator is called, and assuming the values 10 for `myVar1` and 5 for variable `myVar2`, the following text would be added to the trace log: "My custom log with value 10 for variable `myVar1` and value 5 for variable `myVar2`"

Argument lists can be used for the `trace_text()`-operator (see section 4.4.6 for an example) and for the `result_text()`-operator (see section 5.3.10 for an example).

4.2 Variables and Value Maps



Variables are placeholders for values of a particular variable *Type*, used in the logic layer of CBA ItemBuilder projects.



Variables in the CBA ItemBuilder are either of *type* `INTEGER` (i.e. can only contain numbers without decimal places), `NUMBER` (i.e. floating point numbers with a specific accuracy), `STRING` (i.e. text represented as characters) or `BOOLEAN` (i.e. logical values `true` or `false`).


Variables can be used for various purposes, for instance, in conditions of the finite-state machine rules (see section 4.4.5) or to adapt the presentation of text, numbers or images 4.2.5. To map values of *Type* `INTEGER` to text, images, audio and video files, *Value Maps* are used (see section 4.2.4).

Variables are created in the CBA ItemBuilder by specifying a unique *Variable Name* on the project level (see section 4.2.1) and by defining the *Type* of the variable and an initial value. Variables are globally available across different states of the finite-state machine(s) and can be used by different pages within a project. In this way, variables provide a way represent addition information across states and this information can be used to control the behavior and the visual presentation of complex items within *Tasks*. The initial value of variables is defined (see section 4.2.1) and can be adopted to *Task*-specific values using *Task Initialization Rules* (see section 4.5). Values of variables

can also be changed within transitions of the finite-state machine (i.e., using *Operators* as described in subsection 4.4.6). Moreover, variables can be linked to specific input components (see section 4.2.2).

Variables can be used at various places, for instance, to formulate conditional rules in the finite-state machine (see 4.4.5). Moreover, in connection with *Value Maps* (see section 4.2.4) and *Map-based Value Displays* (see subsection 4.2.5) variables play an essential role for the connection of static and dynamic content. In the following, Variables are introduced in more detail beginning with their definition (see section 4.2.1). Specific values of variables can be labeled by naming them (see section 4.2.1).

4.2.1 Introduction

Variables are defined in the *Browse Variables* view of the CBA ItemBuilder that can be requested using the entry *Browse Variables* of the menu *Project* (or using the icon ). After requesting *Browse Variables* an editor opens up in the right part of the CBA ItemBuilder as shown in Figure 4.4 (see also section 3.1.4).

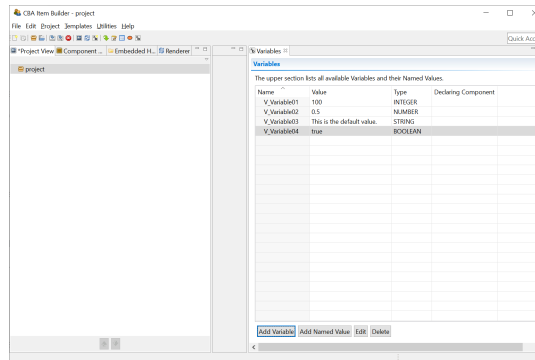


FIGURE 4.4: Editor for *Variables*.

Variable Definition: Adding new variables with the button *Add Variable* and editing existing variables using the button *Edit* opens the dialog *Set Variable attributes* as shown in Figure 4.5. The *Name* of variables must be unique and follow the rules already described for project names and page name (i.e., only letters, digits, and underscores are allowed and the first character must not be a digit).

Each variable requires a *Type* (either *INTEGER*, *NUMBER*, *STRING* or *BOOLEAN*) and the provided *Value* must fit to the variable *Type*. The value is only the default value of variables that can be changed in various ways.

Named Variable Values: Individual values of variables can be assigned to labels. These so-called *Named Values* help to make syntax that uses the variables easier to read and understand.

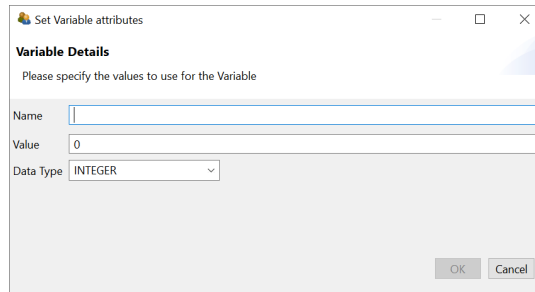


FIGURE 4.5: Set Variable attributes dialog.



Specific values can be defined as *Named Values* for a better readability of CBA ItemBuilder syntax.

For defined variables the button **Add Named Value** can be used to add new *Named Values* (and the buttons **Edit** and **Delete** can be used to modify or removed *Named Values*). *Named Values* are CBA ItemBuilder's approach to improve readability of syntax that implements different interpretation on specific values of variables. Instead of adding a comment that the value 1 represents, for instance, *Visited* and the value 2 of a variable represents *Not Visited*, named values can be defined for variable values.

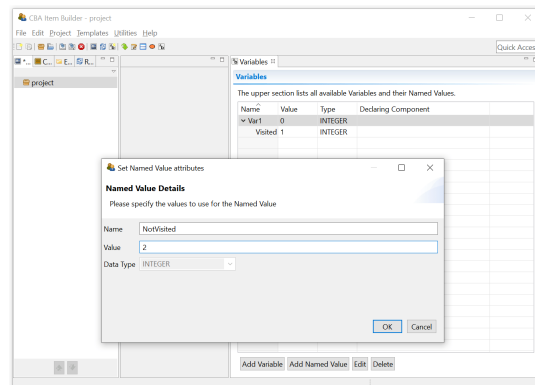


FIGURE 4.6: Set Named Value attributes dialog.

Named variables can be used to structure the syntax of finite-state machines (see section 4.4). To display different labels according to variable values, so-called *Value Maps* can be used as described in section 4.2.4).

Change Variable Values: The prerequisite to use variables for dynamic content is that values of variables can be changed at runtime, either in connection with a user interaction or based on triggering events, such as time-controlled finite-state machine

events. Values of variables can be changed with different mechanisms, which are listed in the following:

- Special input elements (so-called *Value Inputs*) can be bound directly to variables, so that a change to the input elements directly causes a change of the variable values. Details about *Value Inputs* are described in the section 4.2.2.
- Drag-and-drop operations can also be used to change the values of variables. The drag-and-drop fields (using so-called `MapBasedVariableDisplays`) are assigned to variables whose values are changed by the drag and drop user interaction. A detailed description of the possibilities for implementing drag and drop in the CBA ItemBuilder can be found in the section 4.2.6.
- Variable values can be set or changed using `set()`- and `reset()`-Operators in *Conditional Links* (see section 4.3.3) and in *Rules* of the *Finite-State Machine* (see section 4.4.6).

Variables keep their values across page (and state changes in the finite-state machine, below). Thus variables can be used to represent additional information that describe the behavior and visual presentation of items, in addition to the *Current State* of finite-state machine(s).

Use Variable Values: The value of variables can be used for different purposes, for instance, to display dynamic numbers, texts, images and videos (see section 4.2.5), to specific conditions for scoring (see section 5.3.5) or to create conditional *Finite-State Machine Rules* (see section 4.4.5).

4.2.2 Value Inputs

The CBA ItemBuilder provides some components, that change the value of *Variables* in line with user interactions. In particular sliders (`ScaleVariableInput`), spinners (`SpinnerVariableInput`) and input fields (`VariableValueInput`) are directly linked to variables of type `INTEGER` (see Figure 4.7).

Link Variables to Components: Components that can be linked to variables (e.g., *Value Inputs*) provide a context menu entry *Link Variable* as shown in Figure 4.8. Analogous to the definition of *Links*, the use images, audio and video files, the assignment of *Events* and *Value Maps*, the connection of components to variables is made via the context menu entry *Link Variable*.

Components of type `VariableValueInput` and `SpinnerVariableInput` only need to be linked to a variable. Changes in the input immediately change the value of the linked variable. Components of type `ScaleVariableInput` also need to be linked to a so-called *Value Map* (see section 4.2.4).

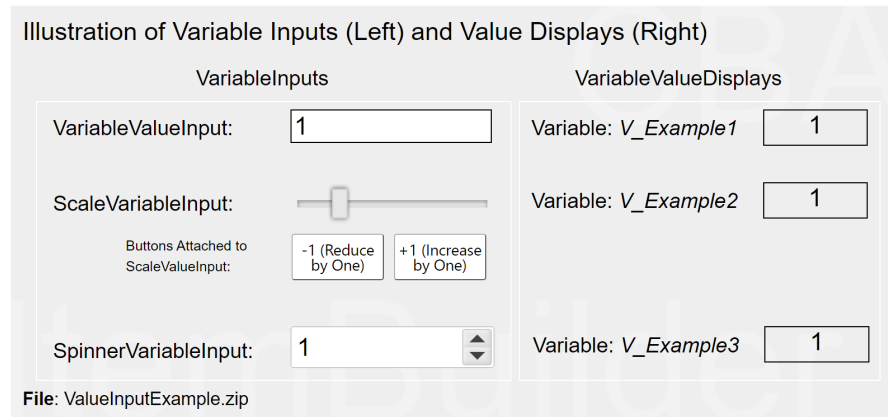


FIGURE 4.7: Item illustrating the use of variables and value inputs ([html](#)|[ib](#)).

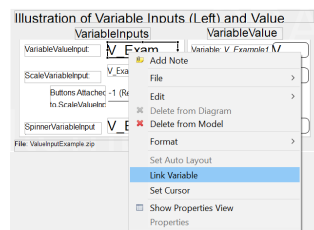


FIGURE 4.8: Context menu in the *Page Editor* to *Link Variables*.

Attach `scaleValueInput` to Buttons: Components of type `Button` (see section 3.11.2) can be linked directly to `ScaleValueInput` so that a click on a particular button increases or decreases the value of the linked variable. The numerical value to increase or decrease the variable must be specified as the property `Increment` in the *Properties* view (can be a positive or negative integer number, default is 0). To assign a `Button` to a `ScaleValueInput` use the entry *Attach ScaleValueInput* in the context menu of a `Button` in the *Page Editor* and select the `ScaleValueInput` in the dialog (see Figure 4.9).

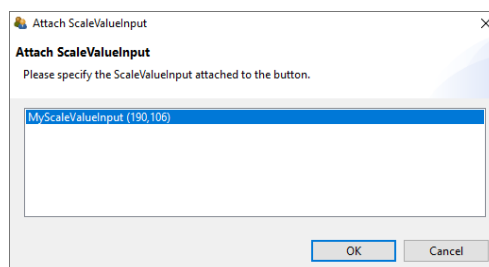


FIGURE 4.9: Dialog for assigning a `Button` to a `ScaleValueInput`.

4.2.3 Variable Value Displays

An essential use of variables is to display values within assessment components, i.e., on the side of an item. For this use case, `VariableValueDisplays` are provided as component that show the value of linked variables plain and unchanged. Examples of using components of type `VariableValueDisplay` can be seen in the right part of Figure 4.7 and in Figure 4.10.

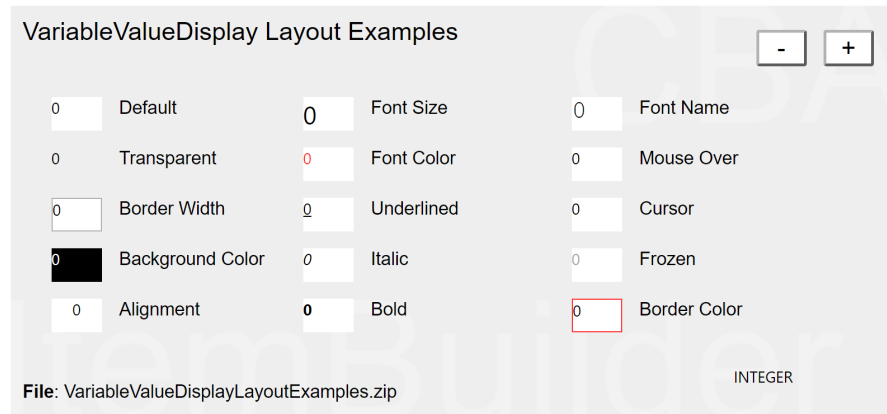



FIGURE 4.10: Item illustrating layout option for `VariableValueDisplays` ([html|ib](#)).

Layout: Font family, font size and font color, text alignment and text decoration as well as borders and transparency of `VariableValueDisplays` can be defined in the *Properties* view.

Events: `VariableValueDisplays` raise an *FSM Event* (see section 4.4.3 and Figure 4.35) and the *Variable Displays* as well as *Maps-based Variable Displays* (see section 4.2.5) can be used to implement *Drag-and-Drop* user interactions (with the additional *FSM Events*, see section 4.2.6).

4.2.4 Value Maps

Variable values can not only be displayed 1:1, but can also be displayed as images, audio or video files or as text with the help of translation tables. The assignment of values of a variable to other values is defined in the CBA ItemBuilder with the help of *Value Maps*.

Value Maps are defined in the *Value Maps* view of the CBA ItemBuilder that can be requested using the entry *Browse Value Maps* of the menu *Project* (or using the icon ). After requesting *Browse Value Maps* an editor opens up in the right part of the CBA ItemBuilder (see also section 3.1.4). The *Value Maps* view as shown in Figure 3.1.4

contains an upper table of all defined *Value Maps* with a *Value Map Details* view in the lower part.

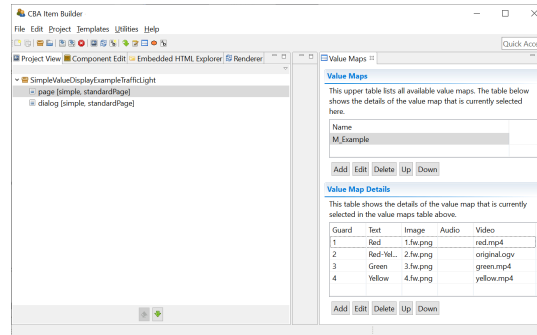


FIGURE 4.11: Editor for *Value Maps*.

Value Maps can be added using the **Add** button in the upper table, requiring a name that can be changed using the **Edit** button. The value maps in the upper table can be re-ordered using buttons **Up** and **Down**.

Once a *Value Map* has been created and selected in the upper table, new entries can be added with the **Add** button and modified with **Edit**.

In the example in Figure 4.11, the *Value Map* `M_Example` is selected in the upper pane, so the *Value Map Details* view shows the defined values. For each definition a *Guard* is necessary, together with at least one text, one image, one audio resource or one video resource. It is also possible to assign different resources directly to a *Guard*. In the example in Figure 4.11, Guards 1, 2, 3, and 4 are each assigned a text, an image, and a video.

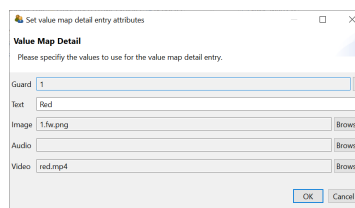


FIGURE 4.12: Dialog *Set value map detail entry attributes*.

Text resources can be defined directly in the editor in Figure 4.12 by typing. Images, audio, and video files refer to the resources imported via the *Resource Browser* (see section 3.10.1) in a CBA ItemBuilder *Project File*.

For the definition of *Guards*, a choice can be made between the *Default Value* (i.e., the value when no other *Guard* applies), a single numerical value (*Single Value*), and a range between two numerical values (*Interval*), see Figure 4.13).

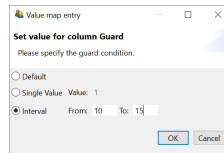


FIGURE 4.13: Dialog for defining *Guards* using the dialog *Set value for column Guard*.

Variables and *Value Maps* are permanently assigned to components when designing pages. However, both can be freely combined between different components, i.e., a *Variable* can be used with one *Value Map* in one component and with an additional *Value Map* in another component.

4.2.5 Maps-based Variable Displays

Components of type `MapBasedVariableDisplay` can be used with *Variables* for displaying dynamic content in the CBA ItemBuilder pages. A static example is shown in Figure 4.14 (see section 4.2.6 for the use of `MapBasedVariableDisplays` in combination with *Drag-and-Drop*).

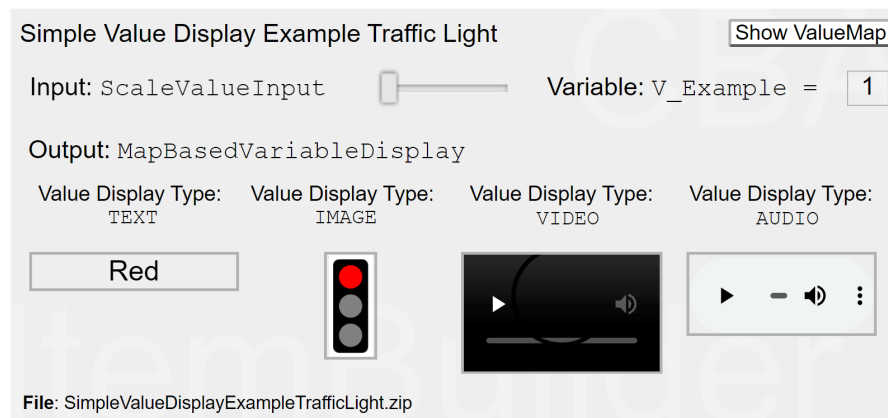


FIGURE 4.14: Item illustrating different variable inputs ([html](#)|[ib](#)).

The component `MapBasedVariableDisplay` can be used to display either text or images or to play audio or video files (to show the raw number of a *Variable* the component `VariableValueDisplay` as described in section 4.2.3 is provided). To define which part of the *Value Map* should be used, the property *Value Display Type* must be defined in the *Properties* view (see Figure 4.15).

Link Value Map to Components: All components that can use *Value Maps*, for instance, `MapBasedVariableDisplays`, provide an entry *Link Value Map* in the context

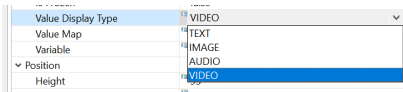


FIGURE 4.15: Property *Value Display Type* for components of type `MapBasedVariableDisplay` in the *Properties* view.

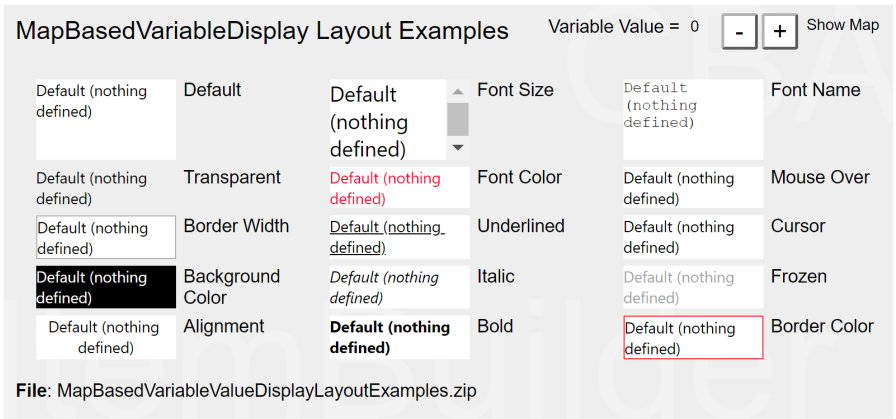


FIGURE 4.16: Item illustrating layout option for `MapBasedVariableValueDisplay` ([html|ib](#)).

menu. In the then opening editor *Set Value Map*, an existing *Value Map* can be selected. To remove an assigned *Value Map*, the *Set Value Map* editor allows deselection.

4.2.6 Drag-and-Drop

Drag-and-drop response formats can now be implemented using *Variables* (see section 4.2) and *Maps-based Variable Displays* (see section 4.2.5). Let’s start with two examples from real assessments (Jiang et al., 2021) shown in Figure 4.17 and 4.18 (see Gong et al., 2022, for an example how to analyze the log data collected with these items).



Drag-and-drop operations with fixed drop points are implemented using *Maps-based Variable Displays*, whose associated variables change their value on *drop* to the value that corresponds the *dragged* element.

Figure 4.19 illustrates how value of variables are changed with drag-and-drop operations. Inspect the example and first of all see, how the *Dragged* element corresponds

Each grid is made up of 100 small squares that are all the same size.

What part of each grid is shaded?

Drag a decimal into each box to show your answer.

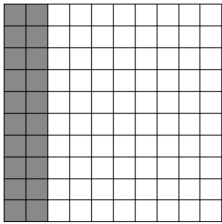
0.02

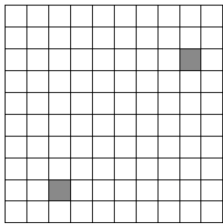
0.20

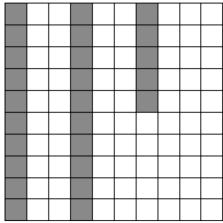
0.25

2.0

2.5







Clear Answer

Released drag-and-drop item administered to fourth-grade students (G4 item) in the 2017 NAEP mathematics assessment.

Next

Jiang, Y., Gong, T., Saldivia, L. E., Cayton-Hodges, G., & Agard, C. (2021). Using process data to understand problem-solving strategies and processes for drag-and-drop items in a large-scale mathematics assessment. *Large-Scale Assessments in Education*, 9(1), 2. <https://doi.org/10.1186/s40536-021-00095-4>

FIGURE 4.17: Item 1 from Jiang et al. (2021) illustrating Drag-and-Drop ([html](#)|[ib](#)).

to the variable value of the `MapBasedVariableDisplay` at which the drag-and-drop operation was started. The variable value is shown in the item, as soon as an element is dragged.

At the moment when a drag-and-drop operation is completed (i.e. at *drop*) the values of the variables are *swapped* in this example (drag-and-drop mode is `DROP_SWITCH`, see Figure 4.21 for alternative configurations).

Technical Details and Concepts For the description and definition of drag-and-drop, a distinction must be made between a drag-and-drop *source* and a *target*. Drag-and-drop is when something is moved from *source* to *target*, either with the mouse as pointing device or by touch.

Either the `MapBasedVariableDisplays` are dragged and dropped on fixed positions, or for free positioning, drag-and-drop allows to move the `MapBasedVariableDisplays` within a parent panel. Figure 4.20 illustrates both types of drag-and-drop functionality: On the left side, drag-and-drop of `MapBasedVariableDisplays` is possible within a Panel with the property `Drop Target=true` (free drag and drop). On the right side, drag-and-drop is possible between `MapBasedVariableDisplays` on fixed positions. For both modes, `MapBasedVariableDisplays` are used as *draggable/droppable* components.

Drag-and-drop only requires *Variables* and *Value Maps* together with `MapBasedVariableDisplays`. Dragging an element from a source position to the target position re-

Drag each of the digits 1, 2, 6 and 7 into the boxes to make the following multiplication problem correct.

Use each digit only once.

$$\begin{array}{r}
 \square \square \square \\
 \times \square \\
 \hline
 4,284
 \end{array}$$

1

2

6

7

Clear Answer

Released drag-and-drop item administered to eighth-grade students (G8 item) in the 2017 NAEP mathematics assessment.


Next

Jiang, Y., Gong, T., Saldivia, L. E., Cayton-Hodges, G., & Agard, C. (2021). Using process data to understand problem-solving strategies and processes for drag-and-drop items in a large-scale mathematics assessment. *Large-Scale Assessments in Education*, 9(1), 2. <https://doi.org/10.1186/s40536-021-00095-4>


FIGURE 4.18: Item 2 from Jiang et al. (2021) illustrating Drag-and-Drop ([html](#)|[ib](#)).

Drag-and-Drop Example
Illustrating the Use of Variables


Dragged Element: No Element.



Variable V_1=2



Variable V_2=3



Variable V_3=4

File: DragAndDropIllustratingVariablesExample.zip

FIGURE 4.19: Drag-and-drop example illustrating the use of variables ([html](#)|[ib](#)).

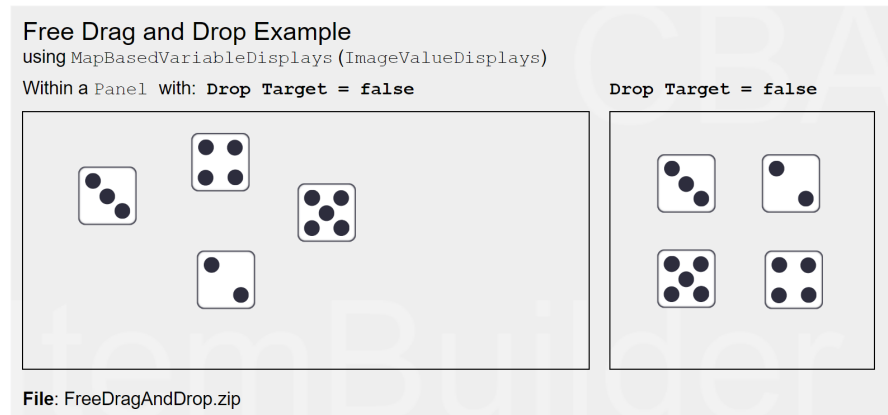


FIGURE 4.20: Item illustrating free drag and drop ([html](#)|[ib](#)).

sults in a change of values of the *variables* linked to the `MapBasedVariableDisplay`. The behavior of successful drag-and-drop operations of an object from a source to a target is configured by defining how variable values are changed. The semantic of the drag-and-drop behavior is reduced to modifying the values of the underlying variables, linked to the `MapBasedVariableDisplays`. This has the advantage that scoring can be done quickly based on the variable values and the various scoring operators (see chapter 5). The following three modes are available that define, how variables are changed after a successful drop-operation: `DROP_SWITCH`, `DROP_MOVE` and `DROP_COPY` (see Figure 4.21).

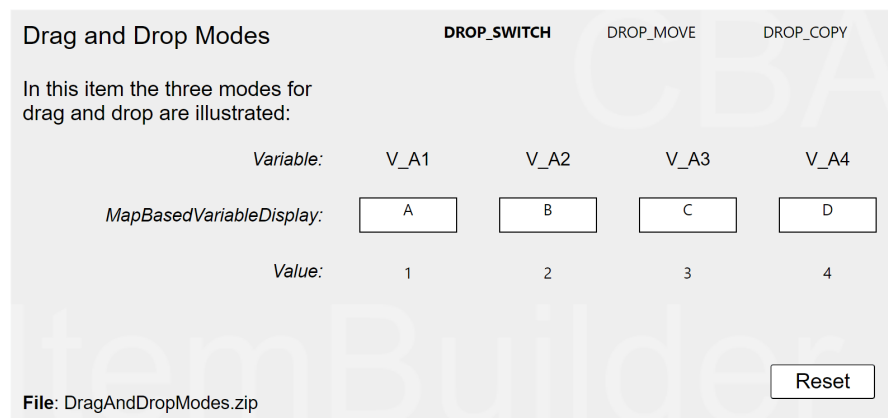


FIGURE 4.21: Item illustrating drag and drop modes ([html](#)|[ib](#)).

The variables are always assigned to a `MapBasedVariableDisplay`, but the different `MapBasedVariableDisplays` do not have to use the same *Value Maps*. This also allows drag-and-drop groups to be implemented, as shown in Figure 4.22.

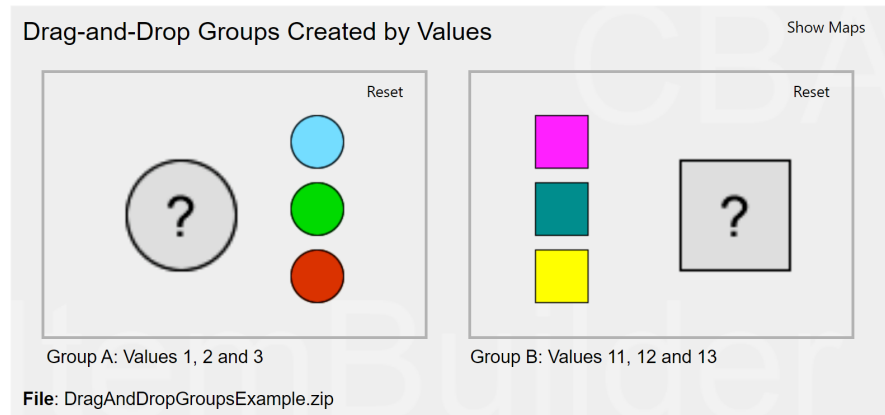


FIGURE 4.22: Item illustrating drag and drop groups ([html](#)|[ib](#)).

`MapBasedVariableDisplays` provide specific *Events* that can be used in the finite-state machine (see section 4.4). The default *Raised Event* is triggered, when a `MapBasedVariableDisplay` is clicked, a *Drag Event* is triggered together with *drag* operations and a *Drop Event* is fired when something is *dropped* at a `MapBasedVariableDisplay` (see Figure 4.35 for an example). Even complex pattern based on finite-state machine rules can be implemented using *Events*, for instance, to impose specific restrictions for allowed drag-and-drop operations.

4.2.7 Dynamic Text in `HTMLTextField`s

`HTMLTextField`'s can be used to display dynamic text. To do this, you must insert keywords in the content of the `HTMLTextField` using the following syntax. These keywords are at runtime translated and replaced with the string values of the referenced parameters:

```
${ParameterName}
```

The possible values for `ParameterName` are either names of FSM variables or references to the scoring result of a task. Accordingly, this mechanism can be used, for example, to implement simple feedback or, in the case of complex tasks, feedback on the scoring can be built into the item, for instance, for testing purposes.

Dynamic texts can be used in component of type `HTMLTextField` with the keywords following the `$`-sign in curly brackets. Dynamic texts are updated at *Runtime* each time the `HTMLTextField` is refreshed (for instance, when the pages change or the `HTMLTextField` is clicked, see Figure 4.23 for an example).

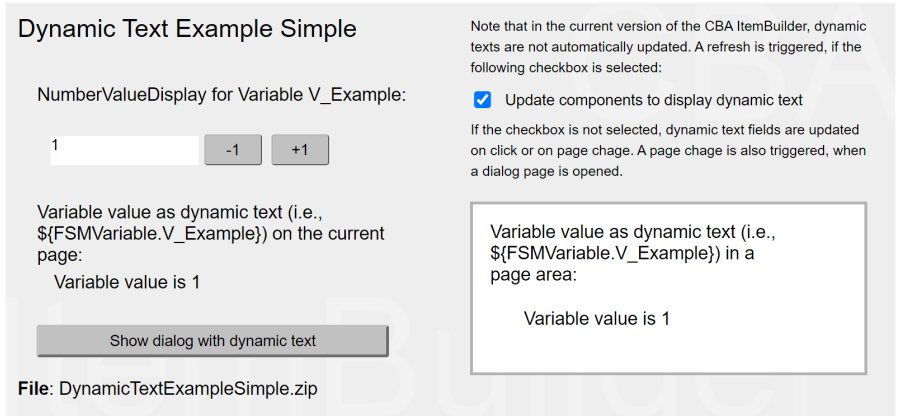


FIGURE 4.23: Example for *Dynamic Text* in `HTMLTextFields` ([html|ib](#)).

If a parameters of the item score is requested and the name of the task is not defined in the project, `-1` is shown at *Runtime*.

The following qualified values for `ParameterName` are available for dynamic texts:

| Qualified Parameter | Description |
|--|---|
| ItemScore .{TaskName} . {ScoreType} | Displays the value of the {ScoreType} for task {TaskName}. The task-name is defined in the task definition (see 3.6) and the {ScoreType} is one of the following keywords (see 5.4 for details): result, credit_Class, credit_weight, reactionTime, reactionTimeTotal, execTime, execTimeTotal, nb_Hits, nbInteractions, nbInteractionsTotal, Hit_weight, nb_Misses, Miss_weight |
| FSMVariable . {FSMVariableName} | Displays the value of the FSM variable {FSMVariableName} as string |

4.3 Conditional Links



4.3.1 Introduction

To design assessment components with multiple views, the CBA ItemBuilder uses the concept of *pages*. Navigation between pages is possible with dedicated *links*. Two links are distinguished: *Regular Links*, which statically always lead to the same page, versus *Conditional Links*, whose target page can change according to specific conditions. An example of a *Conditional Link* has already been shown in section 3.11. Another synthetic example illustrating various properties of *conditional links* is shown in Figure 4.24). The syntax for defining *Conditional Links* follows the simple schema:

```
{Page: Condition}
```

The syntax requires the use of curly brackets to define that a *Page* left of the colon is used as the target page of a link if the *Condition* right of the colon applies.

Figure 4.24 provides different synthetical examples that help to describe the *Conditional Link* feature of the CBA ItemBuilder:

- *Conditional Link 1* illustrates a *Button* that is combined with a condition that spans two lines: *Line 1* {*Target1: Checkbox1*} specifies that the target page *Target1* is used if the *Checkbox* with the *UserDefinedID: Checkbox1* is selected. *Line 2* then contains the statement that the *Button* should link to page *Target2*, if *Checkbox1* is not selected. The condition in *Line 2* uses the logical operator for negation *not* (see section 4.1.3).
- The syntax allows the specification of several combinations of pages and conditions, each included in a new line embedded in curly brackets. *Conditional Link 2* illustrates how the different conditions are processed. For this purpose, another line was added as a new first line: {*Target2: true*}. The condition *true* that was inserted here is always fulfilled (see section 4.1.3). Since the CBA ItemBuilder checks

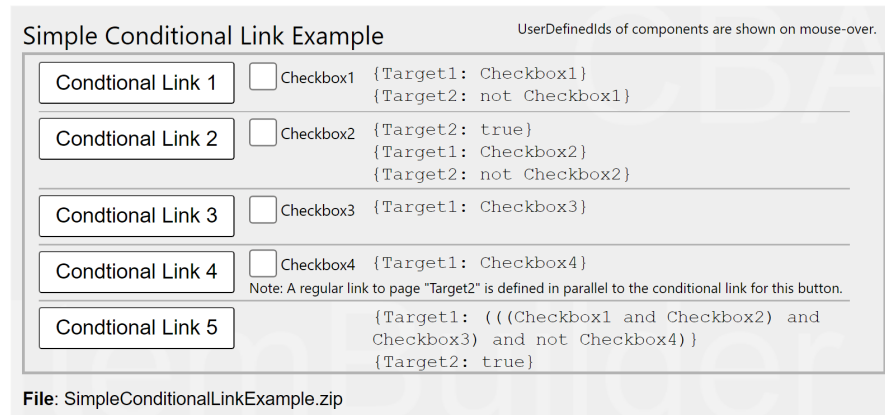


FIGURE 4.24: Item illustrating the use of *Conditional Links* ([html](#)|[ib](#)).

the defined conditionals one after the other, the Button with the text Conditional Link 2 will always refer to the page Target2, even if Checkbox2 is selected, because this combination of page and condition is defined after the reference to {Target2: true}.



Conditional links always link to the first page whose condition is met.

- Conditional Link 3 illustrates that if no condition is met, then no link is activated. In this example, if Checkbox3 is not selected, then clicking on the Button has no effect.
- The fourth example in Figure 4.24 shows how *Links* and *Conditional Links* relate to each other. For this, the *Conditional Link* was only defined that the condition Checkbox4 should lead to the page Target1. In addition, however, the regular *Link* to Target2 was specified via the *Link Page* dialog. The result first checks whether a *Conditional Link* applies. If this is the case (i.e. if Checkbox4 is selected), then the Button points to page Target1. If none of the conditions is true, then the regular link is used, i.e. Conditional Link 4 points to page Target2.
- The last example shows that conditions can also be combined if the rules for setting parentheses are followed. Conditional Link 5' refers, according to the definition, to the page 'Target1', if all checkboxes except Checkbox4 are selected. Then, in the second part of the syntax, if the first condition was not met, the link refers to page Target2, again with the keyword true (i.e., always).

The examples illustrated in Figure 4.24 illustrate that *conditional links* to references to pages can be used in a variety of ways. Of course, the pages do not necessarily have to

be dialog pages (see section 3.15). *Conditional links* can be used like *links* to all possible pages, but the respective context (e.g. *X-Page*, see section 3.11.4) must be considered. As described in the following section 4.3.3, *Conditional Links* can also be used to make changes to the item. For this purpose it may also be useful that *Conditional Links* point to the page on which they are defined.

Defining *Conditional Links* in the *Link Page* dialog: Components that can be used to link pages (see section 3.11) usually allow also the definition of *conditional links* in addition to the definition of regular *links*.

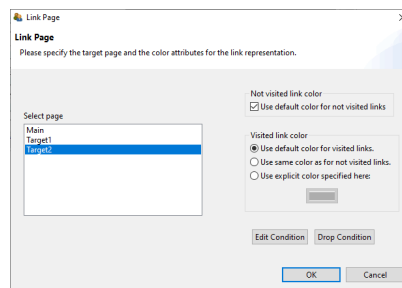


FIGURE 4.25: Dialog *Link Page* with buttons *Edit Condition* and *Drop Condition*).

The syntax editor required for defining *Conditional Links* can be invoked with the button *Edit Condition* (see Figure 4.25). This button allows adding new *Conditional Links* as well as editing existing *Conditional Links*. If an existing *Conditional Link* is to be removed completely, either the syntax can be deleted completely or the button *Drop Condition* is used.

Applied Conditional Link Examples

| | |
|----------------------|---|
| Feedback | Conditional links can be used to display feedback directly following the answer to a question or task. |
| Filtering | Conditional links can also be used to skip pages, for example, when answering questionnaires. This can be implemented with conditional links for forward and backward navigation, as shown in this example. |
| Answer-Until-Correct | Conditional links can also be used to implement response formats where multiple attempts are possible. |

File: AppliedConditionalLinkExamples.zip

FIGURE 4.26: Item with contextualized examples of *Conditional Links* ([html](#)|[ib](#)).

Applied Examples: Conditional links can be used for a variety of applications. Selected examples can be seen in Figure 4.26.

- The first example illustrates *adaptive feedback*, that informs about the correctness of a particular answer or about any other (intermediate) condition, that can be formulated as *Condition*. If *Conditional Links* are used to select a page that is shown, the information that is selected by conditions must be placed on different pages. Moreover, a component that can be used to *Link Pages* must be used to trigger the evaluation of the *Condition*. In the example in Figure 4.26, the decision of which feedback to display is made by the *Conditional Link* defined for the *Next* button. If the question is answered correctly (i.e. the radio button with the `UserDefinedID: rbGreen` was selected), then the *Next* button links to the page: `{feedbackCorrect: rbGreen}`. On the other hand, if a radio button with a different `UserDefinedID` is selected (`((rbRed or rbBlue))`), then the *Conditional Link* points to the `feedbackWrong` page.
- The first example in Figure 4.26 illustrates yet another important use of *conditional links*. The *Conditional Links* definition for the `next` button also handles the case where no selection has been made. This is done implicitly, since all previously defined pairs of pages and conditions cover all possible selections in the `RadioButtonGroup` (if the question was answered, then `rbRed`, `rbGreen` OR `rbBlue` must be selected). If none of the options is answered, then the *Conditional Link* is used to refer to a page which is defined as a modal dialog (see section 3.15). No specific condition is needed for this, i.e. the keyword `true` has been used so that this condition is always met: `{feedbackMissing:true}`. Since this last condition is only checked if the previously defined conditions were not fulfilled, it can be used to implement a feedback that informs about a *missing answers* for that question.



Conditional Links can be used to check and respond to the correctness or completeness of responses when navigating between pages.

- The second applied example in Figure 4.26 shows how *Conditional Links* can be used to show or skip additional pages with specific questions or tasks. By linking multiple pages via *next/back* buttons, the target person does not necessarily have to be informed about which components of an instrument have been skipped or filtered. In the example a *Conditional Link* was used in both directions, i.e. with the *Next* button a page with a specific question is only shown if the answer `Yes` (`rbYes`) was selected: `{filteringOptional: rbYes}`. If the radio button with the `UserDefinedID: rbNo` was selected (`{filtering2: rbNo}`), the page will be switched to `filtering2` immediately. The *back* button on page `filtering2` uses the same logic to link either to page `filtering1` or to page `filteringOptional` when navigating back.
- The third example shows how text responses can be included in the condition for *Conditional Links*. The operator `matches()` is used, which is also used for scoring text responses (see section 5.3.4). The operator is used in the condition for the *Conditional Link* at the *Next* button on page `answerCorrect1` to check if the correct response

has been entered. If this is not the case, i.e. 13 was not entered in the `SingleLineInputField` with the `UserDefinedId: txtResponse`, then the *Conditional Link* points to the same page again: `{answercorrect1: not matches(txtResponse, "13") /*...*/}`. If the question has not yet been answered correctly, then the page is not changed, i.e. an answer format *Answer-Until-Correct* can be implemented in this way. If the first condition `not matches(txtResponse, "13")` is not met (i.e. if `matches(txtResponse, "13")` is met), then the *Conditional Link* in this example points back to the home page `page ({page:true})`. The text `/*...*/` indicates how comments can be used within conditions of *Conditional Links* (see section 4.1.2). But it also indicates that in this example a further functionality was used, namely the use of additional operators that are executed when a certain *Conditional Link* is activated.

Beyond the selected application examples, various forms of interactivity in assessment components can be implemented with the help of *Conditional Links*, which cannot all be listed exhaustively here. However, *Conditional Links* can also be used as an alternative to regular links to trigger actions (e.g., to set a button *frozen*). *Conditional Links* are part of the standard repertoire for designing interactive assessment components that do not require the definition of a complete *Finite-State Machine* (see section 4.4). Often *Conditional Links* therefore also refer to the current page in practice, in order to trigger actions with the help of links, as described in the next section.

Regular Expression: The validation of text input in conditional links can include regular expressions using the `matches()`-operator:

```
{Page: matches(Component, RegularExpression)}
```

Advanced Conditional Link Example

| | | | |
|--------------------|----------------------|-------|---|
| Conditional Link 1 | <input type="text"/> | SLIF1 | {Target1: matches(SLIF1, "1")} {Target2: true} |
| Conditional Link 2 | <input type="text"/> | SLIF2 | {Target1: matches(SLIF2, "1")} {Target2: not matches(SLIF2, "")} |

File: AdvancedConditionalLinkExample.zip

FIGURE 4.27: Item illustrating the advanced use of *Conditional Links* ([html](#)|[ib](#)).

Deprecated Features / Features under Development: The use of the `getItemScore(Task, Calculation)`-operator is currently under development. The use of FSM



variables in conditions is currently not supported in conditional links.

4.3.2 Quick Start (Continued): Feedback Using Conditional Links for Single Page Items




In section 3.3 we created different single page items. In this quick-start, it will be illustrated how conditional links can be used to create different implementations of result feedback:


- Single Choice Item: Feedback using Dialogs
- Multiple-Choice Item: Feedback using PageAreas
- Text Entry Item: Feedback using MapBasedValueDisplays

4.3.2.1 Feedback using Dialogs


1. Open SinglePageItems_01SC.zip and Save as SinglePageItems_01SC_Feedback.zip: Open the item created in the section 3.3.2 (or download [SinglePageItems_01SC.zip](#)) using the icon  in the CBA ItemBuilder (or use the main menu File and the entry Open project). Save the *Project File* with the new name SinglePageItems_01SC_Feedback.zip using the main menu File and the entry Save as... (or use the icon ).




2. Create new Page: Make sure the project name SinglePageItems_01SC_Feedback is shown in the *Project View* in the left part of the CBA ItemBuilder. Right-click on this project name in the *Project View* and select the entry Add new simple page in the context menu. In the dialog that opens, a *Page Name*, e.g. FeedbackCorrect, must be entered. Confirm the dialog with OK (without ticking the checkbox Create as X-page?).

3. Add Frame and Panel: The *Project View* now contains the newly created page FeedbackCorrect. Double-click this page to open the *Page Editor* in the main area of the CBA ItemBuilder's main window. In the *Page Editor* that opens, the *Palette*  **Palette** is now required. If the *Palette* is not displayed, it can be shown at the right window border with the small icon . Select the **Frame**-component by clicking on  **Frame** in the palette. Draw the **Frame** in the empty drawing area of the *Page Editor* by first click and then move the mouse with the pressed left mouse button, so that a gray rectangular with dashed line appears. Release mouse and select the newly created **Frame** by clicking on the rectangular. Right-click on the **Frame** and select the entry *Show Properties View* in the context menu. Find the section *Position* and enter the following values: Height: 600, Width: 800, X: 112, and Y: 84. Change the property *Dialog* in section *Misc* to MODAL_DIALOG and change the property *Closable* to false.

Select the **Frame** in the *Page Editor* and create a new **Panel** on the page FeedbackCorrect. Select the **Panel**  **Panel** in the *Palette*. To add the **Panel** to the **Frame** click in the

drawing area of the *Page Editor* within the area that is covered by the *Frame*, hold the mouse button while moving the mouse and see a small rectangle appearing. Open the *Properties* view and enter the following values in the *Position* section: Height: 600, Width: 800, X: 0, and Y: 0.

4. Add a HTMLTextField: Select the *Panel* in the *Page Editor* and create a new HTML-TextField on the page *FeedbackCorrect*. Select the HTMLTextField  in the *Palette*. To add the HTMLTextField to the *Panel* click in the drawing area of the *Page Editor* within the area that is covered by the *Panel*, hold the mouse button while moving the mouse and see a small rectangle appearing. Open the *Properties* view and enter the following values in the *Position* section: Height: 100, Width: 750, X: 30, and Y: 30. Double-click the HTMLTextField and enter the text *Your answer was correct*. Select the text and change the font size to 28. Finally, save the changes using the button *Save and Close*.

5. Add a Button: Select the *Panel* in the *Page Editor* and create a new Button. click the entry Button  in the *Palette*. When the component type is selected in the *Palette* a new button can be added to the page by drawing a rectangle into the *Drawing Area*. Open the *Properties* view and enter Height: 45, Width: 180, X: 560, and Y: 500 in the section *Position*. Set the property *Border Width: 1* in the section *Display*. Now find the tab *Appearance*  and click it, and change the formatting of the button. Select *Arial* as font and set the font size to 16. Moreover, set the border color to black using the button . Next, double-click the button and enter the text *Continue* in the dialog *Configure a Multiline Text*, before closing the dialog with OK. Finally, right-click the button in the *Page Editor* and select the entry *Set command* in the context menu. In the dialog *Set Runtime Command* click on the entry *NEXT_TASK* and confirm the selection with OK.

6. Configure the button *Next* on page01: Open the page *page01* and find the button with the label *Next*. Right-click the button and select the entry *Set Command*. In the dialog *Set Runtime Command* select the element *Empty* to remove the command *NEXT_TASK*, that was previously assigned to this Button. After closing the dialog with OK, right-click the button *Next* again and select the entry *Link Page*. In the dialog *Link Page* don't select a page on the left part of this dialog, but click the button *Edit Condition*. This will open a syntax window with the title *Link \$. . .* In this dialog, enter the following conditional link syntax: `{FeedbackCorrect: c}` This will link to page *FeedbackCorrect*, if the button is pressed and the element with the *UserDefinedId: c* is selected. Close the link syntax editor via the small cross in the tab title (✕).

7. Duplicate page *FeedbackCorrect* to create page *FeedbackCorrect*: For the next step, close all open *Page Editors* first using the small cross in the tab title (✕). If no editor is open, right-click on the page *FeedbackCorrect* in the *Project View* and select the entry *Save As Template*. Define a template name (e.g., *dialog*) and hit OK in the dialog *Save As Template*.

If the current specification of the page *FeedbackCorrect* as saved as template *dialog*, create new page from template (using the entry *New page from template* in the *Project-*



menu) and select the template dialog. Define the page name as `FeedbackWrong` and create the new page. Open the *Page Editor* of the new page `FeedbackWrong` and change the text of the `HTMLTextField` to `Your answer was wrong..`

Right-click the button `Continue` and use the entry `Duplicate` in the context sub-menu `Edit` to duplicate the `Button`-component. Change the text to `Retry`. Use the *Properties* view to change the position of the `Button` (`X: 50` and `Y: 500`). Assign the command `CLOSE` using the context menu (right-click) and the option `Set Command`. Confirm the dialog `Set Runtime Command` with `OK` after selecting the command `CLOSE`.



8. Create Page `FeedbackMissing` from template: Use the option `create new page from template` (using the entry `New page from template` in the `Project`-menu) again. Create a new page from the template dialog created in step 7 and change the text of the `HTMLTextField` to `Please provide an answer`. Duplicate the `Button Continue` again (see step 7) change the text to `Retry`, change the command to `CLOSE` and the position to `X: 50` and `Y: 500`.

9. Update the link condition of Button `Next` on page01: Open page `page01` in the *Page Editor* and right-click on the `Button` with the text `Next`. Select the entry `Link page` and open the link condition syntax editor using the button `Edit condition`. The syntax defined in step 6 should be extended to the following three lines:




```
{FeedbackCorrect: c}
{FeedbackWrong: ((a or b) or d)}
{FeedbackMissing: true}
```


10. Save, Preview and Test: This concludes the *Feedback using Dialogs* and the project can now be saved () and previewed (). Check the feedback by selecting either nothing, the correct response (`c`) or any other response (and press `Next`).




4.3.2.2 Feedback using `PageAreas`


1. Open `SinglePageItems_02MC.zip` and Save as `SinglePageItems_02MC_Feedback.zip`: Open the item created in the section 3.3.3 (or download [SinglePageItems_02MC.zip](#)) using the icon  in the CBA ItemBuilder (or use the main menu `File` and the entry `Open project`). Save the *Project File* with the new name `SinglePageItems_01SC_Feedback.zip` using the main menu `File` and the entry `Save as...` (or use the icon ).


2. Create new Page: Make sure the project name `SinglePageItems_02MC_Feedback` is shown in the *Project View* in the left part of the CBA ItemBuilder. Right-click on this project name in the *Project View* and select the entry `Add new simple page` in the context menu. In the dialog that opens, a *Page Name*, e.g. `feedbackCorrect`, must be entered. Confirm the dialog with `OK` (without ticking the checkbox `Create as x-page?`).

3. Add Frame and Panel: The *Project View* now contains the newly created page `feedbackCorrect`. Double-click this page to open the *Page Editor* in the main area of the CBA ItemBuilder's main window. In the *Page Editor* that opens, the *Palette*  **Palette** is now required. If the *Palette* is not displayed, it can be shown at the right window border with the small icon . Select the `Frame`-component by clicking on  **Frame** in the palette. Draw the `Frame` in the empty drawing area of the *Page Editor* by first click and then move the mouse with the pressed left mouse button, so that a gray rectangular with dashed line appears. Release mouse and select the newly created `Frame` by clicking on the rectangular. Right-click on the `Frame` and select the entry *Show Properties View* in the context menu. Find the section *Position* and enter the following values: *Height*: 130, *Width*: 950, *X*: 0, and *Y*: 0.

Select the `Frame` in the *Page Editor* and create a new `Panel` on the page `feedbackCorrect`. Select the `Panel`  **Panel** in the *Palette*. To add the `Panel` to the `Frame` click in the drawing area of the *Page Editor* within the area that is covered by the `Frame`, hold the mouse button while moving the mouse and see a small rectangle appearing. Open the *Properties* view and enter the following values in the *Position* section: *Height*: 130, *Width*: 950, *X*: 0, and *Y*: 0.

4. Add Button and HTMLTextField: Buttons can be added to components of type `Panel`. Select the `Panel` created in step 3 and find and click the entry `Button`  **Button** in the *Palette*. When the component type is selected in the *Palette* a new button can be added to the page by drawing a rectangle into the *Drawing Area*. Open the *Properties* view and enter *Height*: 30, *Width*: 100, *X*: 40, and *Y*: 10 in the section *Position*. Set the property *Border Width*: 1 in the section *Display* and define the text as *Check* (either in the *Properties* view using the edit button ... in the *Text* property or by right-clicking on the `Button` in the *Page Editor* and using the entry *Edit Text*). Now find the tab *Appearance*  and click it, and change the formatting of the button. Select *Arial* as font and set the font size to 8. Moreover, set the border color to black using the button .

The `Panel` created in step 3 needs to be selected again in the *Page Editor*. If the `Panel` is selected, the palette allows to select the `HTMLTextField` entry ( **HTMLTextField**). Draw a rectangle in the *Drawing Area* (inside the `Panel`) to add the `HTMLTextField`. If the *Properties* view is visible, click on the `HTMLTextField` and enter the following values in the *Position* section: *Height*: 80, *Width*: 700, *X*: 160, and *Y*: 10. To enter text, double-click the `HTMLTextField` and enter the text *Correct* in the *HTML Text Editor*. Select the text and change the font size to 25. Finally, save the changes using the button *Save and Close*.

5. Duplicate page `feedbackCorrect` to create page `feedbackWrong` and `feedbackCheck`: For the next step, close all open *Page Editors* first using the small cross in the tab title (). If no editor is open, right-click on the page `feedbackCorrect` in the *Project View* and select the entry *Save As Template*. Define a template name (e.g., *area*) and hit *OK* in the dialog *Save As Template*. Repeat the step and create also the page `feedbackCheck` using template *area*.

6. Add PageArea to page page01: Open the page `page01` in the *Page Editor* and select the `Frame`. Use the context-menu entry *Show Properties View* to make sure you selected the `Frame`. If the `Frame` is selected, a component of type `PageArea` can be selected in the *Palette*. Once the `PageArea` is selected in the *Palette*, a rectangle can be drawn inside the `Frame` (but not inside the `Panel`). So, in order to add the `PageArea`, left-click in the margin area, that is the area of the `Frame` that is not covered by the `Panel`. Draw a small rectangle to add the `PageArea` and then select the `PageArea` to change its position in the *Properties* view (`X: 40, Y: 530, Width: 950, Height: 130`). Moreover, specify the `UserDefinedId` in section *Identification* of the `PageArea` to be `area`. Finally, right-click the `PageArea` and select the entry `Link Embedded Page`. In the dialog `Link Embedded Page` select `feedbackCheck` and confirm the dialog with `OK`.



7. Finish page feedbackCheck: Open the page `feedbackCheck` in the *Page Editor* and delete the `HTMLTextFiel` (for instance, using the context menu entry `Delete From Model`). Change the text of the `Button` to `Check` and define the following syntax as `Condition` for a link. For that purpose, right-click the `Button` and select the entry `Link Page`. In the dialog `Link Page` click on the button `Edit Condition`:

```
{page01: (e1 and e3) | setEmbeddedPage(area, feedbackCorrect) setFrozen(e1)
                        setFrozen(e2) setFrozen(e3) setFrozen(e4)}
{page01: true | setEmbeddedPage(area, feedbackWrong) setFrozen(e1)
                setFrozen(e2) setFrozen(e3) setFrozen(e4)}
```


8. Finish page feedbackCorrect: Open the page `feedbackCorrect` in the *Page Editor* and define the following syntax as conditional link (i.e., as syntax defined using context menu `Link Page` and the button `Edit Condition`):


```
{page01: true | setEmbeddedPage(area, feedbackCheck) unsetFrozen(e1)
                unsetFrozen(e2) unsetFrozen(e3) unsetFrozen(e4)}
```


9. Finish page feedbackWrong: Open the page `feedbackWrong` in the *Page Editor* and define the same syntax as in step 8 as conditional link (i.e., as syntax defined using context menu `Link Page` and the button `Edit Condition`). Finally, edit the text of the `HTMLTextField` and enter `Wrong`. Save the changes using the button *Save and Close*.

10. Save, Preview and Test: This concludes the *Feedback using Dialogs* and the project can now be saved () and previewed (). Check the feedback by selecting either nothing, the correct response (c) or any other response (and press `Check`).

4.3.2.3 Feedback using MapBasedValueDisplays

1. Open SinglePageItems_03TXT.zip and Save as SinglePageItems_03TXT_Feedback.zip: Open the item created in the section 3.3.4 (or download [SinglePageItems_03TXT.zip](#)) using the icon  in the CBA ItemBuilder (or use the main

menu **File** and the entry **Open project**). Save the *Project File* with the new name `SinglePageItems_03TXT_Feedback.zip` using the main menu **File** and the entry **Save as...** (or use the icon ).

2. Define States and Variables: For this example the definition of two *States* and two *Variables* is necessary. To define states and variables, open the *State Machine Tree View* using the icon  (or use the main menu **Project** and the entry **Edit State Machine**). In the *State Machine Tree View* right-click on the entry **Machine** and select the entry **State** in the sub-menu **New Child** to add the first state. After adding the state use the *Properties* view to change the name of the state from `<not set>` to `ST_Start`. Select **START** as Type. Repeat the same steps to add a second state (Name: `ST_Main`, Type: **NORMAL**).

Continue and add two variables and define their names as `V_Frequency` and `V_Feedback`.


3. Edit State Machine Syntax: Change to the tab **State Machine Rules** and copy the following text:


```
Events: EMPTY;
Rules: ST_Start -> ST_Main{true|set(V_Frequency,0), set(V_Feedback,99)}
```

Close both tabs (**State Machine Rules** and **State Machine**) and confirm to save the changes.

4. Add button with label check: Open page `page01` in the *Page Editor* and add a new **Button** to the **Frame**. Define the position (X: 500, Y: 350, Height: 30, Width: 120) and the **TextCheck**(Font: Arial, Font Size 8). Right-click the button and open the entry **Link Page**. In the dialog **Link page click Edit Condition** and enter the following syntax into the syntax editor:

```
{page01: matches(txt,"42") | set(V_Frequency,V_Frequency+1) set(V_Feedback,1)}
{page01: (not matches(txt,"42") and not matches(txt,"")) |
    set(V_Frequency,V_Frequency+1) set(V_Feedback,0)}
{page01: matches(txt,"") | set(V_Frequency,V_Frequency+1) set(V_Feedback,2)}
```

Close the syntax editor using the small x in the tab-title (.

5. Use the Resource Browser to Insert Images: As an example on how to use images in value maps, two image should be added. For this purpose, the picture must be added using the *Resource Browser* to the *Project File* first. Open the *Resource Browser* over the main menu **Project** and the entry **Browse resources** (or use the icon ). Any image in one of the supported file formats (see section 3.10.1) can be used. Two pictures are needed, one for a correct response and one for a wrong response. If you

don't have a picture at hand, a sample image can be downloaded here: [SinglePageItemFeedbackResources](#). Unpack this ZIP archive and then click the 'Add' button in the *Resource Browser* of the CBA ItemBuilder to add the files 'ThumbsDown_min.png' and 'ThumbsUp_min.png' to the list of *Available resources*. Close the *Resource Browser* via the small cross in the tab title (✕).


6. Add ValueMaps (M_Feedback and M_Frequency): The values of the two variables `V_Frequency` and `V_Feedback` should be mapped to text and images. This is possible using so-called value maps. To edit value maps, click the icon  (or use the main menu *Project* and the entry *Browse Value Maps*). In the upper part of the *Value Map Editor* use the button *Add* to create a first value map and enter `M_Feedback` as name. Select the value map `M_Feedback` and enter the following assignments using the buttons *Add* and *Edit* in the lower part of the *Value Map Editor*:

TABLE 4.2: Example for 'ValueMap'-definition 'M_Feedback'

| Guard | Text | Image |
|-------|---------|--------------------|
| 99 | | |
| 0 | Wrong | ThumbsDown_min.png |
| 1 | Correct | ThumbsUp_min.png |
| 2 | Empty | |

Repeat the same steps and create a second value map `M_Frequency` with the following mappings:

TABLE 4.3: Example for 'ValueMap'-definition 'M_Frequency'



| Guard | Text |
|-------|---|
| 0 | Press the check button to verify your answer. |
| [1,5] | You have checked a few times. |
| * | You have checked multiple times. |

7. Add ImageValueDisplay for Feedback: Open page `page01` in the *Page Editor* and select the `Panel`. Open the *Palette* and select the component with the name `ImageValueDisplay`. Draw a small rectangle within the `Panel` to add the `ImageValueDisplay` and define the following properties: `X: 750`, `Y: 250`, `Height: 200`, `Width: 200`. Right-click the `ImageValueDisplay` and use the entry *Link Variable* to link the `ImageValueDisplay` to the variable `V_Feedback`. Confirm the dialog *Set State Machine Variable*. Right-click the `ImageValueDisplay` again to link the value map `M_Feedback` using the context-menu entry *Link Value Map*. Close the editor *Set Value Map* with *OK*.

8. Add a first TextValueDisplay for Feedback: Add a component of type `TextValueDisplay` to the `Panel` and define the following properties: `x: 230`, `Y: 310`, `Height: 30`,

Width: 260. Right-click the `TextValueDisplay` and use the entry `Link Variable` to link the `TextValueDisplay` to the variable `V_Feedback`. Confirm the dialog `Set State Machine Variable`. Right-click the `TextValueDisplay` again to link the value map `M_Feedback` using the context-menu entry `Link Value Map`. Close the editor `Set Value Map` with OK.

9. Add a second `TextValueDisplay` for Frequency: Add a second component of type `TextValueDisplay` to the `Panel` and define the following properties: X: 230, Y: 350, Height: 30, Width: 260. Right-click the `TextValueDisplay` and use the entry `Link Variable` to link the `TextValueDisplay` to the variable `V_Frequency`. Confirm the dialog `Set State Machine Variable`. Right-click the `TextValueDisplay` again to link the value map `M_Frequency` using the context-menu entry `Link Value Map`. Close the editor `Set Value Map` with OK.

10. Save, Preview and Test: This concludes the *Feedback using ValueMaps* and the project can now be saved () and previewed (). Check the feedback by selecting either nothing, the correct response (42) or any other response (and press Check).

4.3.3 Trigger Actions in Links using Operators

If you take a closer look at the example *Answer-Until-Correct* in Figure 4.26 in a web browser or in the *Preview* of the CBA ItemBuilder, you will notice that the `Next` button is initially disabled. Only when you put the focus into the `SingleLineInputField` and start entering an answer, the `Next` button is activated. This behavior is also implemented using a *Conditional Link*. For this the `SingleLineInputField` links to the current page and removes the deactivation of the button with the `unsetFrozen()` operator.

The syntax of the *Conditional Links* is extended by a component after a pipe operator (`|`) to the following scheme:

```
{Page: Condition | Operator1() }
```

Again, embedded in curly brackets is first the page, followed by a colon. After the colon the condition is formulated. If several conditions are to be combined, this is again possible in compliance with the bracket rule (see section 4.1.3). If an additional action is to be added, which is executed when the *Conditional Link* is activated and the condition is met, an operator can be added after the vertical bar (`|`).

If multiple operators are to be executed, then they must be listed in the syntax for *Conditional Links* separated by spaces, as the following scheme for multiple operators illustrates:

```
{Page: Condition | Operator1() Operator2() }
```


In the CBA ItemBuilder, links can usually also be formulated as *Conditional Links*. A simple link is a *Conditional Link* with only one target page and the keyword `true` as condition.



Actions can be performed using operators when a link is activated. For this purpose the syntax of *Conditional Links* allows that besides a target page and a condition also one or more operators separated by spaces are specified.

It is possible to link to the same target page multiple times in *Conditional Links*. Conditions can then be used to define more precisely when which actions should be executed, i.e. which operator(s) should be triggered by a particular (*Conditional*) *Link*.

Use of Conditional Links with Operators

Use conditional links to set "Button 1" `setFrozen(Button1)` or `unsetFrozen(Button1)`.

Button 1

Use conditional links to set the value of variables: Increase the variable using the `set(Varname,Value)`-operator: `set(Var1,0)`, `set(Var1,Var1+1)` or `set(Var1,Var1-1)`.

0

In this example, a conditional link is used to select the radio button "RB_No" if the `SingleLineInputField` gets focused. This is possible since the operator `setActive(RB_No)` and `setActive(RB_Yes)` can be used in conditional links, and because `SingleLineInputField` allow to use conditional links.

☐ Yes. ☐ No. Please Explain: Click here to select RB_Yes.

Using the operators `setHidden(UserDefinedID)` and `unsetHidden(UserDefinedID)` also allows to hide and show selected components, for instance, an Panel.

Hide

File: UseOfConditionalLinksWithOperators.zip

FIGURE 4.28: Item illustrating the use of *Conditional Links* with Operators ([html](#)|[ib](#)).

Various *Operators* are illustrated in the following example (see figure 4.28):

- To enable and disable components, there is the `setFrozen(UserDefinedId)`-operator and its inverse `unsetFrozen(UserDefinedId)`. Only if the `Button` is not disabled, an action can be executed with it (i.e. a regular or *Conditional Link* can be activated or a *FSM event* can be triggered, see section 4.4.3). In the example, the button opens a modal dialog.
- To set the value of an *FSN variable* (see section 4.2), the `set(Variable)`-operator can be used. In the example, this `set()`-operator is used to implement various *conditional links* which set the value of the variable `Var1` to 0 (`set(Var1,0)`), increase the value by one (`set(Var1,Var1+1)`) and decrease the value by one (`set(Var1,Var1-1)`).
- Third, the item in Figure 4.28 illustrates the use of the `setActiv(Component)` operator. Selected components (e.g., `RadioButtons`) can be selected with it.

- Finally, the example contains *Conditional Links* at the `Hide` button (and, when pressed, the `Show` button that then appears). The operators `setHidden(UserDefinedId)` and `unsetHidden(UserDefinedId)` are used here to hide and show individual components.

4.3.4 Link PageAreas using Conditional Links

When links are used in the CBA ItemBuilder, they are always linked within the current page area. This approach makes it easy to implement links inside and outside X-Pages. If a page is included within a `PageArea`, then links within the `PageArea` lead to the page, which includes the `PageArea` being replaced. If you want to override this default behavior, you can do this with the help of *Conditional Links*. The following example illustrates how conditional links (to the current page) can be used to change the page displayed within the `PageArea` using the `setEmbeddedPage()`-operator.

Link PageAreas Using Conditional Links Example

This is the page "sub1" shown in the PageArea "main".

Link to Page "sub2".

Note: The link is defined as conditional link to page "mainpage" the following condition:

```
{mainpage: true|setEmbeddedPage(main,sub2)}
```

Link "Sub1"

Link "Sub2"

The white area is PageArea "main" on page "mainpage".

File: LinkPageAreasUsingConditionalLinksExample.zip

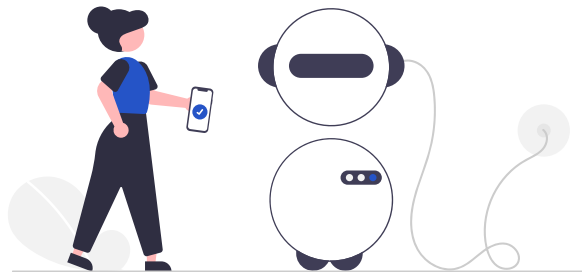
FIGURE 4.29: Item illustrating the `setEmbeddedPage()`-operator ([html](#)|[ib](#)).

The `setEmbeddedPage()`-operator can also be used as operator within the finite-state machine (see section 4.4.6 for details). A complete listing of all operators is included in the appendix B.2.



For more examples illustrating the potential of *Conditional Links* see section 6.4.

4.4 Finite-State Machine(s)



The creation of interactive assessment content with the CBA ItemBuilder can require a *logic layer* for dynamic content in addition to the design of static content (i.e., the *Page Editor* used for designing pages, see section 3.7), *Variables* (see section 4.2) and *Value Maps* (see section 4.2.4). Following the design principles of the CBA ItemBuilder (see section 2.11), this logic layer is not defined in a concrete interpreted or compiled scripting or programming language. Instead, this logic layer is implemented in the CBA ItemBuilder based on so-called *finite-state machines* (also called UML statechart). Abstracting the logic required within items from a concrete programming language to the generic approach of *finite-state machines* allows the concrete source code required for a runtime environment to run an actual item to remain separate from the definition of the required interactivity (see section 2.11.2). For people with programming experience, however, it requires a bit of rethinking, since typical concepts (such as loops or branching) are possible with *finite-state machines*, but may require different approaches than loops or vectorization.

4.4.1 Introduction

The following text describes the use of finite-state machines in the specific implementation of the CBA ItemBuilder, as far as they are needed to implement interactive assessment content. The goal of the presentation is to describe the functionality of the CBA ItemBuilder, and this goal should be achievable without the need for further literature on the general concept of finite-state machines or automata theory. In fact, the implementation in the CBA ItemBuilder is only metaphorically related to the formal idea of finite-state machines due to the possibility of using multiple finite-state machines within an item (see section 4.4.8) and the integration of variables in conditions (see section 4.4.5).


The following gives a brief introduction to the idea: A finite-state machine is something in the logic layer of the CBA ItemBuilder, that always starts in a specified state, called the *Start State*. Each finite-state machine can be in **exactly one** state at a time

and the number of states is limited (i.e., *finite*). A finite-state machine is, however, a machine that can change its state according to deterministic *Rules*. The *Rules* describe the *Transition* between the states and rules are triggered by *Events*. If no *Event* occurs, the finite-state machine stays in its current state, but events can be defined as timed events (see section 4.4.3), and timed events trigger automatically after a pre-defined time interval. Finally, each finite-state can reach one out of multiple *End States*.

The CBA ItemBuilder implementation allows to use multiple finite-state machines (based on *Regions*, see subsection 4.4.8). When *Transitions* are triggered successfully by *Events*, additional *Operators* can be executed (see subsection 4.4.6). And *Variables* can be used in conditions, so that the *Rules* can define *Transitions* between *States* that depend on variable values.



The finite-state machines that can be created in the CBA ItemBuilder allow the appearance and behavior of the item to be modified based on user interactions and temporal events.

User Interface for Finite-State Machines: To define and edit finite-state machines, the user interface of the CBA ItemBuilder provides two parts. Both can be opened at once by clicking on the icon  in the *Toolbar* (see section 3.1.1) or using the *Project* menu *Edit State Machine*). The first part of the user interface is titled *State Machine* and contains the *State Machine Tree View* as shown in Figure 4.30. A second tab with the title *State Machine Rules Editor* for state machine syntax, and both parts always open together.

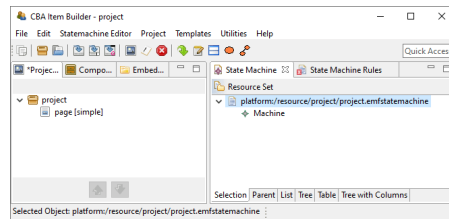


FIGURE 4.30: Empty *State Machine Tree View* of the CBA ItemBuilder.



It is possible to close the *State Machine Rules Editor* using the small \times next to the title of the tabs, so that only the *State Machine Tree View* is displayed. Note, however, that it does not work the other way around. Closing the *State Machine Tree View* automatically closes the *State Machine Rules Editor*.¹

State Machine Tree View The first component of the user interface is a tree representation of the defined states, optionally organized in regions (see Figure 4.30). After selecting the root, new states can be added to this tree by using either the menu *Statemachine Editor* (see left part in Figure 4.31) or the context menu in the *State Machine Tree View* (see right part in Figure 4.31).

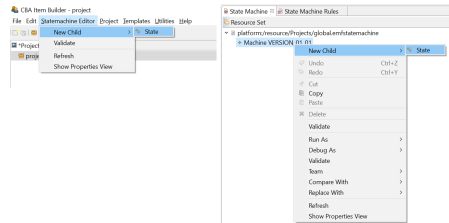


FIGURE 4.31: Main menu *Statemachine Editor* and context menu in the *State Machine Tree View* to define a state



States are defined and configured in the *State Machine Tree View* of the CBA Item-Builder (see section 4.4.2 for details).



State Machine Rules: Events and finite-state machine rules are defined using syntax, edited in the *State Machine Rules* editor. By default, the state machine rules file is empty.² The following example shows a valid rules file:

```
Events: EV_Name; /* Define events here, separated by comma.
                End the list with a semicolon. */
Rules:        /* Define rules here.
                Examples: */
ST_Start -> ST_First {true}
ST_First => ST_Second {EV_Dummy}
```

At least one event must be defined. Text within `/* ... */` or behind `//` is ignored as a comment (see section 4.1.2).

4.4.2 States

States are defined in the *State Machine Tree View* of the CBA ItemBuilder. After adding a new *State* using the context menu or the main menu as shown in Figure 4.31, the

²Technically, the empty rule definition is *invalid*, shown with the symbol  compared to . But this is of no importance, as long as the finite-state machine is not used in a CBA ItemBuilder project.

CBA ItemBuilder lists an undefined state in the *State Machine Tree View* as illustrated in the upper part of Figure 4.32. Continue by double-click on the new node <not set> to finish the configuration. This opens the *Configure State* dialog (see lower part of Figure 4.32).

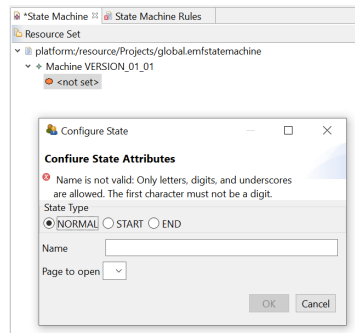


FIGURE 4.32: Newly created *State* and *Configure State* dialog.

For defining the state using the dialog *Configure State* it is essential to differentiate three different *State Types* (NORMAL, START and END).

Normal States (NORMAL): A finite-state machine typically differentiates multiple *Normal States*. The state in which a finite-state machine is (labeled the *Current State*) represents the main information of a finite-state machine. Transitions between normal states describe the functioning of the finite-state machine and the possible flow between the internal states. In the CBA ItemBuilder the transitions are triggered by *Events* (see section 4.4.3) and transitions can trigger actions (called *Operators*, see section 4.4.6). Most of the states are regular states.

Start State (START): However, every finite-state machine needs a starting state in which it is initially located. For this purpose, the state type *START* must be selected for exactly one state for each finite-state machine. A *Start State* is required for the very first *Start Rule* (\rightarrow , see section 4.4.4), that links the *Start State* to a first *Normal State*.

End States (END): The CBA ItemBuilder also allows defining states as *End States*. The use of *End States* is optional for the design of assessment components but can be helpful if, for instance, reaching a dedicated state in the logic layer itself is information for scoring (see section 5.3.7). While a start state and several normal states are needed for the typical use of the finite-state machine, the definition of end states is rarely necessary.



The state in which a finite-state machine is is called the *Current State*. Each state machine needs a start state and can have a remaining countable set of normal and end states.

State Names: After selecting a *State Type* in the dialog *Configure State* (see lower part of Figure 4.32), states require a unique *State Name*. *State Names* need to be a string literal without white spaces and without special characters (underscores, i.e., `_` is possible). Numbers are allowed but not as the first character. When defining states make sure that the state name is valid and not followed by white spaces.



After changing the state definition, close both editors (i.e., the *State Machine Tree View* and the *State Machine Rules*) to make sure the changes are applied internally.

Page to open: Finally, the dialog *Configure State* (see lower part of Figure 4.32) also provides the option *Page to open*. States can also be assigned to pages, so that the *Page* is shown, if the particular *State* is entered (see section 4.4.9 for details about the assignment of *Pages to States*).

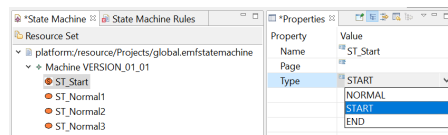


FIGURE 4.33: *State Machine Tree View* with four states and *Properties view*.

Note that states can be ordered within the *State Machine Tree View* using drag-and-drop. For a real application of finite-state machines in the CBA ItemBuilder, multiple states need to be defined, and ordering the states can increase the readability. As soon as the first state is defined, the CBA ItemBuilder also provides a context menu for states with the option to define states and so-called *Regions* as child elements. This feature is related to multiple (nested) finite-state machines, described in section 4.4.8. If only one finite-state machine is to be used, care must be taken when creating the states that all states created directly below the node `Machine VERSION_01_01`. As can also be seen in Figure 4.33, the *State Type* (and the *State Name* and assigned *Page to open*) can also be configured in the *Properties view*.

4.4.3 Events

After defining a set of *States* in the *State Machine Tree View* of the CBA ItemBuilder, *Events* are defined in the *State Machine Rules* editor that was automatically opened as described in section 4.4.1. Events are used to trigger *Transitions* in the *Finite-State Machine(s)*. There exist two type of events:

1. Events Linked to Components: Components used in the *Page Editor* to create visual parts of the assessment components provide slots to link *Events*. For instance, components of type `CheckBox` can trigger events when the user selects the `CheckBox` (called *Raised Event*) and when the user de-selects a previously selected `CheckBox`

(called *Raised Alternate Event*). Components that can raise one or different *Events* provide a context menu entry to link the event to events defined in the *State Machine Rules* syntax.

2. Timed Events: The finite-state machine concept of the CBA ItemBuilder also includes events that are automatically triggered repeatedly after a defined time interval. Such *Timed Events* can be used to change the behavior and appearance of items without user interaction by triggering *Transitions*. *Timed events* are also defined in the *State Machine Rules* syntax.

Events can also be triggered within transitions (see the `raise()`-operator in section 4.4.6), meaning that events that trigger transitions can be used to trigger an additional event.

Definition of Events: Before events can be linked to components in the *Page Editor*, they must be defined in the *State Machine Rules*. While *States* can be created in the *State Machine Tree View* with a graphical user interface, the definition of *Events* is only done by syntax. The structure of the syntax for *State Machine Rules* is:

```
Events: Event1, Event2, Event3; // Events defined as list
Rules: // Start rule
      ST_Start -> ST_First {true}
      // Additional rules
      ST_First => ST_Second {EV_Dummy}
      /*...*/
```

Since the syntax editor initially contains an empty document, times must enter the keyword `Events:` (incl. colon) first. Afterwards at least one event must be defined, if necessary as placeholder. The list of events must contain valid event names. Again, valid event names must not start with a number, must not contain spaces, and only letters or the `_` character are allowed.

The assignment of event names in the *State Machine Rules* syntax should be chosen in such a way that the meaning is clear from the event name. Only events must be defined which are used within the *Finite-State Machine*, e.g., for transitions between states or the triggering of operators. The specified name should be so unique that it can be identified in the *Page Editor* when using the dialog *Link Raised Event*.



Assigning events to components requires that the *State Machine Rules* changes are saved. After defining events by listing valid event names in the *State Machine Rules*, the item must be saved to apply all changes (or both the *State Machine Rules* and *State Machine Tree View* editors must be closed).

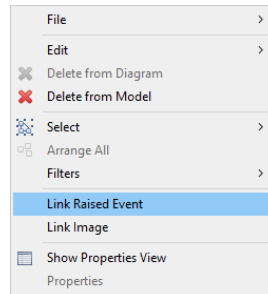


FIGURE 4.34: *Link Raised Event* in the context menu of the *Page Editor*.

To link an event to a particular component, the context menu in the *Page Editor* offers an entry called *Link Raised Event*, as shown in Figure 4.34.

Link Events to Components: For most components events can be assigned (i.e., *linked*) using the context menu in the *Page Editor*.³ Events are triggered by click by default (e.g., an event can be raised, if a panel is clicked, called *Raised Event*). Components that can be selected or deselected, components used for text entry and for components for audio/video content provide more specialized events are triggered by particular actions (see below). The CBA ItemBuilder item shown in Figure 4.35 illustrates which user-interactions trigger events that can be linked for most components.⁴

Components of different `type` provide different slots that can be used to link events. The meaning of the different slots is described mostly in chapter 3 together with the components itself.

Defining Timed Events: The definition of events that trigger automatically after a defined amount of time is done directly in the first section of the *State Machine Rules* syntax after the keyword `Events:`. If a number is specified there separated by a space after the event name, this number will be used as the time interval in seconds.

In the following example two *Timed Events* are defined. Event `E_T1` is fired after two 3.5 seconds, `E_T2` is fired after 10 seconds:

```
Events: E_ChangeState,
        E_T1 3.5,
        E_T2 10;
Rules: // ...
```

³Some events can only be assigned using the *Properties* view (i.e., some component, such as `Frames`, see section 3.5.1), do not allow to assign events using the *Page Editor*.

⁴More components support events, but are not shown in Figure 4.35, for instance, `Tree`, `Table`, `List`, and `Timer`.

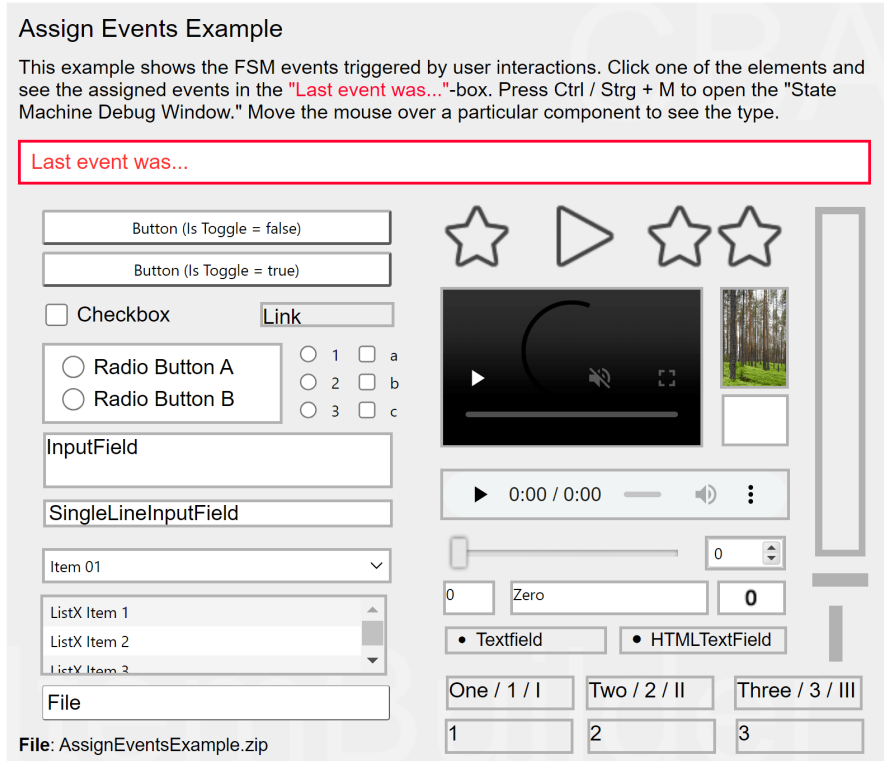


FIGURE 4.35: Item illustrating events triggered by user interactions with components ([html](#)|[ib](#)).

The definition of timed and regular (un-timed) events (e.g., `E_ChangeState`) can be mixed and time intervals can have decimal places. Timed events are started when the finite-state machine changes the *Current State*.



Timed events are started (and restarted) when the state machine changes state. To define a recurring timed event, a state must be restarted when the event is triggered (e.g., with a *Self-Transition* `State => State`, see section 4.4.7).

Timed events can be used to implement *Timers* (i.e., components that show the remaining time for a *Timed Event*, see section 4.4.10).

Since timed events are restarted when entering states, the processing of timed events in finite-state machine rules is critical to their behavior, as illustrated in Figure 4.36. To create independent finite-state machines, nested finite-state machines can be used (see section 4.4.8).

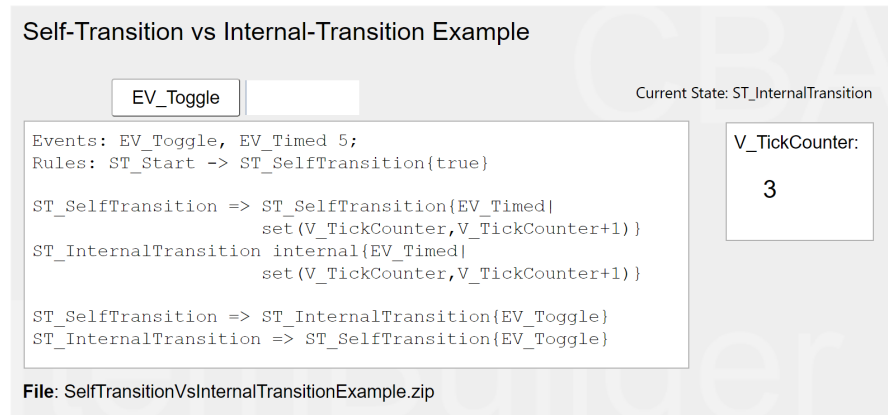


FIGURE 4.36: Item illustrating timed events ([html](#)|[ib](#)).

To fully understand the design possibilities with *Timed Events* shown in Figure 4.36, one must consider the difference between different rules for defining transitions. Transitions that lead to a change of the *Current State* are described in the next section 4.4.4, the processing of events without changing the current state is shown in section 4.4.7. More advanced scenarios can be implemented using so-called *Nested Finite-State Machines* (see section 4.4.8 and Figure 4.63 for an example).

4.4.4 Rules

The behavior of finite-state machines is described by rules that specify how a concrete event of a particular type is to be processed in a particular state. If no rule is defined that event of a particular type is to be processed in state, then the finite-state machine ignores events of that type in that state. If a rule is defined, then events of this type are accepted in a state and processed according to the rule.

Start Rule (Start Transition): At runtime, each finite-state machine is initially in its start state. A first rule with the syntax element `->` is therefore necessary to define which regular state the finite-state machine should enter first:

```
StartState -> StateA {true}
```

The definition of a state of type *Start State* (see section 4.4.2) and the specification of an initialization rule from the *Start State* to a first *Normal State* is mandatory. If this rule should always be executed, the keyword `{true}` is specified in curly brackets.

Advanced Use of Start Rules: When working with multiple *Tasks* within one CBA ItemBuilder project file (see section 3.6) or when combining multiple finite-state

machines (see section 4.4.8), some additional features can be used. Finite-state machine rules are defined for all *Tasks* of a CBA ItemBuilder project file. Instead of `true` specific conditions can be formulated to initialize a finite-state machine in a particular *Task*:

```
StartState -> StateA{isCurrentTask(Task01)}
StartState -> StateA{isCurrentTask(Task02)}
```

The condition `isCurrentTask(TaskName)` will be true, if the current task equals the specified `TaskName`, so that different initial states of a finite-state machine are possible if multiple initialization rules are defined.

It is also possible to use read the page defined in the *Task Definition* (see section 3.6.1) in a condition for the *Start Rule*, as the following syntax illustrates:

```
ST_Start -> ST_C{current_page(page2)}
```

If more than one start rule is defined, the order in which they are defined is decisive. For the example in Figure 4.37, four tasks are defined (`Task01-Task04`). All tasks except `Task03` use page `page1` as the *Start Page* (as defined in the *Tasks* view, see section 3.6.1).

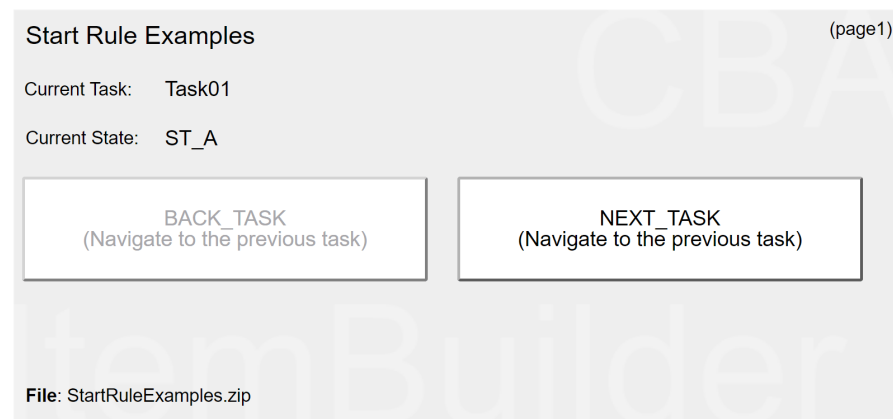


FIGURE 4.37: Item illustrating different *Start Rules* for projects with multiple *Tasks* ([html](#)|[ib](#)).

Here is the shortened finite-state machine syntax of the item in Figure 4.37. The order of the *Start Rules* is crucial, which can be easily seen by the fact that only for `Task04` the state `ST_D` is assigned. Although the condition `true` is always true, this last defined *Start Rule* is only applied if no previous condition is true.

```

Events: Placeholder; // No event is used in this example, but the syntax
                // requires the definition of at least one event.
Rules: ST_Start -> ST_A{isCurrentTask(Task01)| /* ... */ }
      ST_Start -> ST_B{isCurrentTask(Task02)| /* ... */ }
      ST_Start -> ST_C{current_page(page2)| /* ... */ }
      ST_Start -> ST_D{true| /* ... */ }

```

In simple CBA ItemBuilder projects, typically only one *Start Rule* is needed per finite-state machine. Besides the first transition in a regular state, further rules are used as described next.

Regular Rules (State Transitions): The possible connections between (normal) states are defined as transitions with rules using the syntax element `=>`:

```
StateA => StateB {TriggerEvent}
```

State Transitions can be read this way: From state *StateA* is changed to state *StateB* when event *TriggerEvent* occurs.



The CBA ItemBuilder distinguishes between different rule types in the finite-state machine implementation. The two most important rule types are a *Start Transition* (`->`) and multiple *State Transitions* (`=>`).

As *TriggerEvent* all events can be used, which are defined after the keyword *Events*: at the beginning of the *State Machine Rules* syntax. The use of *Timed Events* is possible as well as the use of *Events* created by user interactions with a component created in the *Page Editor*.

The use of user interactions in the finite-state machine of the CBA ItemBuilder is done via events that are first defined in the *State Machine Rules*. These events are assigned to components via *Link Raised Event* and the *Events* can be used within *Rules* for *State Transitions*.

In this way, the processing of user interactions becomes dependent on the *Current State* of a finite-state machine. Conditions (i.e. different processing of identical events) are realized by defining different rules for different *States* (see Figure 4.38 below).



An event can trigger only one rule for each finite-state machine. If more than one transition is defined for a state, which should be executed at an *Event*, the order in the syntax decides! For the readability and simpler interpretation this should be avoided!

Actions in Transitions (Operators): The *Current State* of a finite-state machine can be interpreted like the value of a categorical variable. Thus, it already represents information in itself. For the use of finite-state machines for the dynamic design of assessment components, however, the *Transitions* are also central, i.e. the transitions between *States* which are triggered by *Events*. With these *Transitions* changes can be made to the item, for instance, to the visual representation or to values of *Variables*, which form the dynamic parts of items depending on user interactions or *Timed Events*.

To trigger actions for a defined *Transitions* of a finite-state machine in state `StateA` when the *Event* `TriggerEvent` occurs, *Operators* can be called as seen in the following syntax:

```
StateA -> StateB {TriggerEvent | Operator1(), Operator2(), ..., OperatorN()}}
```

The names of the states (`StateA` and `StateB`) can be freely assigned when defining them in the *State Machine Tree View* (see section 4.4.2) and should be chosen sensibly in the context of the current CBA ItemBuilder project. Similarly, the names of *Events* can be freely specified by entering text in the *State Machine Rules* syntax (see section 4.4.3). In contrast, the operators inserted to the right of the `|` character in the syntax must be valid operators provided by the CBA ItemBuilder. `Operator1()`, `Operator2()` etc. in the example, are for illustration purposes only. Operators may require arguments, which are passed in single parentheses. If multiple arguments are passed, they must be separated by commas within the single parentheses. When entering operators, the auto-complete function helps (see section 4.1.1). Care must be taken to ensure exact spelling, including upper and lower case. A description of valid operators can be found in section 4.4.6.

The use of operators is also possible with *Start Rule*, as the following syntax illustrates:

```
StartState -> StateA {true | Operator1(), Operator2(), ..., OperatorN()}}
```

A selection of operators can also be used in the *Task Initialization* syntax (see section 4.5).

Self-Transitions: Transitions from a state `A` to the identical state `A` are called *Self-Transitions*. *Self-Transitions* restart the *Current State A*, and must therefore be distinguished from the use of operators without transitions (with the keyword `internal`) and the execution of operators when entering (with the keyword `entry`) and leaving (with the keyword `exit`) states (see section 4.4.7).



Self-Transitions (StateA => StateA {TriggerEvent|Operators()}) re-start the *Current State* and are not identical to internal processing of events (StateA internal {TriggerEvent|Operators()}).

In addition to operators, conditions can be defined for transitions, as described in the next section 4.4.5.

State Transition Example

Page: pageA
Current State: ST_A

```
ST_A => ST_A{EV_Previous|openDialog(dialogBegin, 240, 20)}
ST_A => ST_B{EV_Next|setEmbeddedPage(PA,pageB)}
```

Previous
(EV_Previous)

Next
(EV_Next)

Machine VERSION_01_01

- ST_Start
- ST_A
- ST_B
- ST_C

File: StateTransitionExample.zip

FIGURE 4.38: Item illustrating simple *Transitions* ([html|ib](http://html5lib.org)).

Figure 4.38 shows a simple item with four *States* and two *Events*. The two *Events* are linked to the buttons. ST_Start is the start state that always leads to state ST_A because of the first rule (ST_Start -> ST_A{true}):

```
Events: EV_Next, EV_Previous;
Rules: ST_Start -> ST_A{true}
// Transitions when FSM is in state ST_A
ST_A => ST_A{EV_Previous|openDialog(dialogBegin, 240, 20)}
ST_A => ST_B{EV_Next|setEmbeddedPage(PA,pageB)}
// Transitions when FSM is in state ST_B
ST_B => ST_C{EV_Next|setEmbeddedPage(PA,pageC)}
ST_B => ST_A{EV_Previous|setEmbeddedPage(PA,pageA)}
// Transitions when FSM is in state ST_C
ST_C => ST_C{EV_Next|openDialog(dialogEnd, 240, 20)}
ST_C => ST_B{EV_Previous|setEmbeddedPage(PA,pageB)}
```

In each state, the events EV_Previous and EV_Next are used in transitions. ST_A is the first in the sequence, if event EV_Previous occurs in state ST_A the openDialog()-operator (see section 4.4.6) is used to show the page dialogBegin at position $X = 240$

and $Y = 20$. If the event `EV_Next` occurs in state `ST_A`, the rule requests the finite-state machine to change to state `ST_B` and the `showEmbeddedPage()`-operator is used to show the page with name `pageB` in the `PageArea` with `UserDefinedID: PA`. Rules for state `ST_B` define transitions for both events. Event `EV_Next` is used to change to state `ST_C` and to show the embedded page `pageC`, and `EV_Previous` changes to state `ST_A` and shows the embedded page `pageA`. Hence, the same buttons linked to the events `EV_Previous` and `EV_Next` are used and the triggered events are processed differently, according to the *Current State* of the finite-state machine.

4.4.5 Conditional Rules (Guards)

Transitions in the finite-state machine are described by rules that specify how the finite-state machine should react to an event. Conditions (*Guards*) can be used to restrict that a transition is only executed, when a condition (typically formulated using *Variables*, see section 4.2.1) is fulfilled.

Conditions in Rules (Guards): The definition of conditions in transitions of the finite-state machine is introduced by a colon, followed by the condition in square brackets:

```
StateFrom => StateTo { EventName : [Condition] | Operator }
```

As usual, operators are optional (if no operators are required, the syntax simplifies to `StateFrom => StateTo { Event : [Condition] }`). Conditions are typically formulated by using variables (see section 4.2), and only variables of type `INTEGER` or `NUMBER` are currently supported in *guards*.

The conditions need not be mutually exclusive. However, it is important to note that the order of the rules can be relevant. The first condition that is fulfilled for the current state and for which a rule is defined for an event will be executed.

In the following definition, if the variable `v_Example` has the value 3, for example, the second transition would be executed (if event `EV_Example` was raised and the machine is in state `state1`), but not the third transition:

```
state1 => state2 {EV_Example : [V_Example>10] | /*...*/ } // Transition 1
state1 => state3 {EV_Example : [V_Example<5] | /*...*/ } // Transition 2
state1 => state1 {EV_Example : [V_Example<4] | /*...*/ } // Transition 3
```

The finite-state machine would then be in state `state3` after processing event `EV_Example`, if the value of `v_Example` would be 3.

Named variable values (see section 4.2.1) can be used to make the finite-state machine syntax more readable, and syntax comments are suggested (see section 4.1.2).


```
state1 => state2 { event : [Variable1 == Variable1.NameValue1] | /*...*/ }
state1 => state3 { event : [Variable1 == Variable1.NameValue2] | /*...*/ }
```

If conditions are used in particular to execute different operators on identical state changes, then a shortened notation can be used. The following long form defines two different conditions, for the identical *self-transition* that occurs within `state1`.

```
state1 => state1 { event : [Condition1] | Operator1() }
state1 => state1 { event : [Condition2] | Operator2() }
```

For a clearer presentation, the formulation can also be shortened as follows:

```
state1 => state1 { event : [Condition1] | Operator1() }
                  { event : [Condition2] | Operator2() }
```

Item Example: A content motivated example of a conditional transition can be seen in Figure 4.39. In this example, the *Next*-button should only be activated after 5 seconds (see *Blocked Item Response* in section 2.4.1 for more background). Timed events (see section 4.4.3) can be used for this purpose, in combination with the so-called `unsetFrozen()`-operator (see section 4.4.6 below for details).

In the example, the timed event `EV_TimeTick` is triggered every second. In a variable `V_TimeOnPage`, it counts how long the page has been visible, i.e., the variable is incremented by 1 every second if the maximum value has not yet been reached. If the maximum value is reached, the *Next* button is activated.

For illustration purposes there is also a reset button is added to the example, which can be used to reset the time.

The following listing shows the finite-state machine syntax including the conditions `[V_TimeOnPage<4]` and `[V_TimeOnPage>=4]` for the item shown in Figure 4.39:

```
Events: EV_TimeTick 1, EV_Reset;
Rules: Start -> page{true|raise(EV_Reset)}
page => page
// Condition 1
{EV_TimeTick:[V_TimeOnPage<4]|set(V_TimeOnPage,V_TimeOnPage+1)}
// Condition 2
{EV_TimeTick:[V_TimeOnPage>=4]|unsetFrozen(ButtonNext),set(V_TimeOnPage,5)}
{EV_Reset|set(V_TimeOnPage,0),setFrozen(ButtonNext)}
```

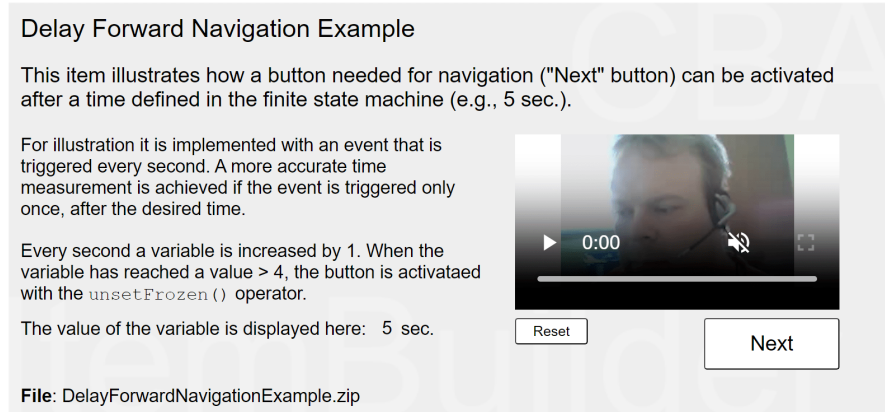


FIGURE 4.39: Item illustrating a delayed activation of a button using a timed event and the `unsetFrozen()`-operator ([html|ib](#)).

Processing the event `EV_TimeTick` is central to the illustration of conditions in state machine rules. If the variable `V_TimeOnPage` has a value less than 4, the variable `V_TimeOnPage` is increased by one with the statement `set(V_TimeOnPage,V_TimeOnPage+1)` (see section 4.4.6 for details on the `set()`-operator). Since condition 1 is fulfilled, the processing of the event 'EV_TimeTick' is thus terminated (the button remains deactivated). If condition 1 is no longer fulfilled, i.e. the variable `V_TimeOnPage` has a value that is not less than 4, the CBA ItemBuilder runtime checks whether condition 2 is fulfilled when the event `EV_TimeTick` occurs. This is always the case in this example. Connected to the transition defined for condition 2, the `ButtonNext`-button is then activated (i.e. the operator `unsetFrozen(ButtonNext)` is executed, see section 4.4.6). For cosmetic reasons the variable `V_TimeOnPage` is also set to the value 5, because in the example item it is displayed with a so called `NumberValueDisplay` (see section 4.2.5).

In addition, the syntax shown also shows the `EV_Reset` event, which is assigned to the reset button. This event is also triggered during initialization of the state machine, i.e. in the transition from the start state `start` to the state `page` using the operator `raise()` (see section 4.4.6). When the machine is in state `page` and the event `EV_Reset` is triggered, the variable `V_TimeOnPage` is set to the value 0 and the button is deactivated.

Note: In a real application, the timed event `EV_TimeTick` would be defined to fire after the required amount of time (without the additional variable that counts the number of ticks). By this change, the functionality illustrated in the example shown in Figure 4.39 can be achieved without conditional rules, and the `unsetFrozen()`-operator can be executed directly after the required amount of time.

Contextualization of Events: As the following example in Figure 4.40 illustrates, the meaning of events can change depending on the current state.

Combination of Conditions: A specific syntax is available to combine conditions to

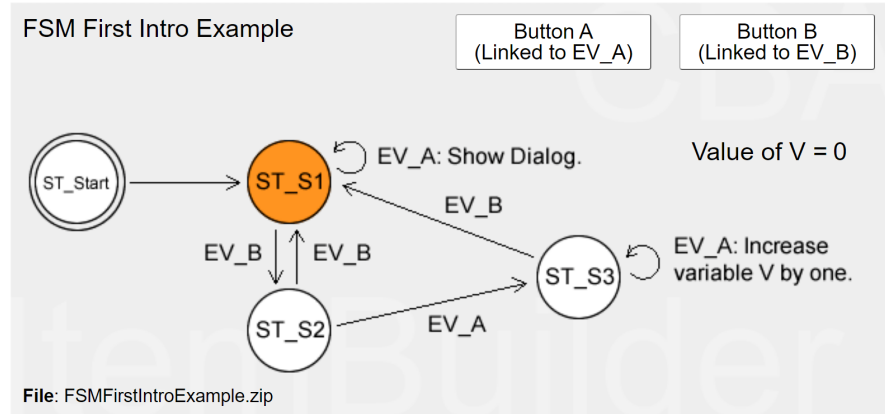


FIGURE 4.40: Item illustrating contextual dependency of events ([html](#)|[ib](#)).

more complex logical expressions.⁵ Note that logical expressions need to be in square brackets, and grouped into pairs of two (see also section 4.1.3).

```
// V1 == 1
state1 => state2 {EventName : [V1==0]}
```

```
// V1 == 1 and V2==2
state1 => state2 {EventName : ([V1==1] and [V2==2])}
```

```
// V1 == 1 or V2==2
state1 => state2 {EventName : ([V1==1] or [V2==2])}
```

```
// (V1 == 1 or V2==2) and (V3==3)
state1 => state2 {EventName : (([V1==1] or [V2==2]) and [V3==3])}
```

Current Task in Conditions: As already described for *Start Rules* (see section 4.4.4), conditions make use of the `isCurrentTask(Taskname)`-syntax. However, to define a valid condition for a conditional finite-state machine rule, it must be wrapped in an additional `ifthenelse(Condition, ExprTrue, ExprFalse)==ExprTrue`-block:

```
state1 => state2 {EventName : [ifthenelse(isCurrentTask(Task01),1,0)==1]}
```

The `isCurrentTask(Taskname)` evaluates to true if the current *Task* has the name specified as *Taskname*. In this case, the `ifthenelse(Condition, ExprTrue, ExprFalse)` returns

⁵Note that the `variable_in()`-operator, that is available for scoring purposes (see section 5.3.5) is not available in conditions of the finite-state machine.

`ExprTrue`, i.e., the value 1. This value is then compared with `==1` and the condition is true if `Taskname` is the current *Task*.

Current Page in Conditions: The identical procedure can also be used to formulate transitions between states with conditions that check whether a particular page is the current page:

```
state1 => state2 {EventName : [ifthenelse(current_page(pagename),1,0)==1]}
```

Text Input in Conditions: Specifically for text input only, the CBA ItemBuilder also provides the ability to use the `matches()`-operator to check whether specific text is entered into an input field (or whether the text matches a regular expression):

```
state1 => state2 {EventName :  
    [ifthenelse(matches(UserDefinedId,"text or regex"),1,0)==1]}
```

Mathematical Expressions in Conditions: Simple mathematical calculations using basic arithmetic (+, -, *, / and %) and some selected functions (`floor` / `ceil` / `trunc` and `round`) can be used in conditions:

```
state1 => state2 {EventName : [V1 >= V2]}  
state1 => state2 {EventName : [V1 + V2 == V3]}  
state1 => state2 {EventName : [round(V1/2) == V1*2]}
```

Elapsed Time in Conditions: The elapsed time (in milliseconds) during task execution of the current task can also be used in conditions:

```
state1 => state2 {EventName : [elapsedTime() < 5000]}
```

In the current version of the CBA ItemBuilder, scoring operators and access to other components (such as `CheckBox`, `RadioButton` etc.) are not provided for conditional rules (guards). However, in most instances, variables can be used instead. An example showing different conditional rules can be found in [Figure 4.41](#).

4.4.6 Operators

The transitions between states defined by *Rules* (see section [4.4.4](#)) and the internal processing of events without state changes as well as *entry* and *exit* of *States* (see

Conditional Rules Example

User-Defined Input:
V1=
V2=
V3=
sli:

Defined Rules:

```

ST_Regular => ST_Regular{EV_Click : [V1==0]}
ST_Regular => ST_Regular{EV_Click : ([V1==1] and [V2==2])}
ST_Regular => ST_Regular{EV_Click : ([V1==1] or [V2==2]) }
ST_Regular => ST_Regular{EV_Click : ([V1==5] or [V2==6] and [V3==3])}
ST_Regular => ST_Regular{EV_Click : [V1 >= V2]}
ST_Regular => ST_Regular{EV_Click : [V1 + V2 == V3]}
ST_Regular => ST_Regular{EV_Click : [round(V1/2) == V1*2]}
ST_Regular => ST_Regular{EV_Click : [ifthenelse(matches(sli,"dog"),1,0)==1]}
ST_Regular => ST_Regular{EV_Click : ([elapsedTime() > 50] and [V3==99])}
ST_Regular => ST_Regular{EV_Click : }

```

(see here which rule was activated)

File: ConditionalRulesExample.zip

FIGURE 4.41: Item illustrating the combination of condition in *Rules* ([html](#)|[ib](#)).

section 4.4.7) can be used to execute operators. The available operators of the CBA ItemBuilder are described next.

Operators for Variables: For variables defined in the *Browse Variables* view (see section 4.2.1), the `set()`-operator and a `reset()`-operator is available to be used either in the *Finite-State Machine* or in *Conditional Links* (see Figure 4.42).

```

set(Variable, Value)
reset(Variable, Variable, ...)

```

Example Illustrating the Change of Variable Values Using Operators

VariableValueDisplay linked to Variable "V_Example"

1

Buttons with Events

Finite-State Machine Syntax Triggered by Events with `set()`- and `reset()`-Operators

```

ST_Running internal
{EV_Minus|set(V_Example,V_Example-1)}

ST_Running internal
{EV_Plus|set(V_Example,V_Example+1)}

ST_Running internal
{EV_Reset|reset(V_Example)}

ST_Running internal
{EV_Value|set(V_Example,42)}

```

File: ChangeVariablesByOperatorsExample.zip Show Conditional Link Example

FIGURE 4.42: Item illustrating the operators `set()` and `reset()` ([html](#)|[ib](#)).

The `set()`-operator assigns the provided value to a particular variable and requires

two arguments: The first argument is the variable name (without quotes), the second argument is the value that should be assigned to the variable. The value can also be provided as formula, for instance, referring to other variables.



In the *Finite-state machine* syntax of the CBA ItemBuilder, the variable name is written in the operators `set()` and `reset()` without quotation marks directly in brackets.

The `reset()`-operator assigns the value 0 to the variable (or variables) provided as arguments. The `reset(Var1)`-operator is identical to the statement `set(Var1,0)`. The `set()`-operator is also available for conditional links, but the `reset()`-operator is provided for finite-state machine syntax only and must be re-written using the `set()`-operator for conditional links.

Operator to Freeze Components: Figure 4.43 demonstrates how components can be changed to *Frozen* using the `setFrozen()`-operator.

```
setFrozen(UserDefinedId)
unsetFrozen(UserDefinedId)
```

The `setFrozen()`-operator is illustrated in Figure 4.43 for *Finite-State Machines* and *Conditional Links*. In the *Finite-State Machine* the operator is triggered with an event `EV_SetFrozen` linked to the button *Set Frozen*. If the components are frozen, the button *Un-set Frozen* linked to the event `EV_UnsetFrozen` changes the components back to the default (`Is Frozen: false`).

Example for `setFrozen()` / `unsetFrozen()`-Operator

Finite-State Machine:

Conditional Link:

myButton

myInputField

mySingleLineInputField

myLink

Main Menu

A B C D

☐ myCheckBox

☐ myRadioButton01

☐ myRadioButton02

☐ myRadioButton03

Item 1

Option A

Option B

Option C

File: SetFrozenExample.zip

FIGURE 4.43: Item illustrating the operators `setFrozen()` and `unsetFrozen()` ([html|ib](#)).

The following listing shows the shortened finite-state machine syntax for the item shown in Figure 4.43:

```
Events: EV_SetFrozen, EV_UnsetFrozen;           // Definition of two events
Rules: Start -> ST_Unfrozen {true | /*...*/}    // Rule 1
ST_Unfrozen => ST_Frozen {EV_SetFrozen | //...   // Rule 2
    setFrozen(myButton),
    setFrozen(myInputField),
    setFrozen(mySingleLineInputField),
    //...
}
ST_Frozen => ST_Unfrozen {EV_UnsetFrozen | //... // Rule 3
    unsetFrozen(myButton),
    unsetFrozen(myInputField),
    unsetFrozen(mySingleLineInputField)
    //...
}
```

After the task is loaded the finite-state machine changes from state `start` to state `ST_Unfrozen` (Rule 1). In state `ST_Unfrozen` the components `myButton`, `myInputField` and `mySingleLineInputField` are not frozen (property `Is Frozen` in section `Misc` of the *Properties* view is `false`). The finite-state machine defines the two events `EV_SetFrozen` and `EV_UnsetFrozen`. The two events are triggered by the buttons `Set Frozen` and `Unset Frozen`. If the item is in the state `ST_Unfrozen` the event `EV_SetFrozen` triggers the transition to state `ST_Frozen` (Rule 2). In this transition, the `setFrozen()`-operator is used, to change the `Is Frozen` property of the components with the user defined `Ids` `myButton`, `myInputField` and `mySingleLineInputField` to `true`. Likewise, the event `EV_UnsetFrozen` triggers the transition to the state `ST_Unfrozen`, and in this transition the `unsetFrozen()`-operator is applied to the components (Rule 3).



In the *Finite-state machine* syntax of the CBA ItemBuilder, the `UserDefinedId` of the component to be changed is written directly in brackets in the operators `setFrozen(UserDefinedId)/unsetFrozen(UserDefinedId)` without additional quotation marks.

The `setFrozen()` / `unsetFrozen()` - operators can also be used in *Conditional Links* as shown in the lower part of Figure 4.43. The conditional link syntax refers to the current page (i.e., no page is changed when the *Conditional Link* is triggered. The literal `true` is used as the condition, meaning that no specific condition needs to be fulfilled, and the list of operators right to the `|` will be always executed.

```
{page: true | setFrozen(myButton)
  setFrozen(myInputField)
  setFrozen(mySingleLineInputField)
  /*...*/
  setFrozen(myFrame,0)}
```

The `setFrozen()` / `unsetFrozen()`-operators can be applied to input elements (i.e., components of type `Button`, `SingleLineInputField`, `InputField`, `Checkbox`, `RadioButton`, `ComboBox`, `List` and `Menu`) and the operators are also available in conditional links (see Figure 4.43). The operators can also be applied to `Frame Select Groups` (see section 3.9.4) when two arguments are provided: The `UserDefinedId` of the `Frame` and the index of the `Frame Select Group` (i.e., the `GroupNumber` starting with 0).

```
setFrozen(Frame, GroupNumber)
unsetFrozen(Frame, GroupNumber)
```

Operators to Hide/Show Components: Components can be not only frozen from the *Finite-State Machine* and using operators in *Conditional Links*, but also completely hidden:

```
setHidden(UserDefinedId)
unsetHidden(UserDefinedId)
```

As Figure 4.44 shows, the operators `setHidden()` and `unsetHidden()` can be used to hide and show components specified by a valid *User Defined Id*.

Components of type `PageArea`, `Table`, `ImageMap`, and `List` are not supported by the `setHidden()` / `unsetHidden()`-operator. Panels are supported, but components nested within panels are not affected by hiding / un-hiding Panels.

The operators can also be applied to `Frame Select Groups` (see section 3.9.4) when two arguments are provided: The `UserDefinedId` of the `Frame` and the index of the `Frame Select Group` (i.e., the `GroupNumber` starting with 0).

```
setHidden(Frame, GroupNumber)
unsetHidden(Frame, GroupNumber)
```

Operator to Focus Input Fields: In `SingleLineInputFields` and `InputFields` (see section 3.9.1) text can only be entered if these components are focused. The following operator can be used to set the input focus from the finite-state machine or from

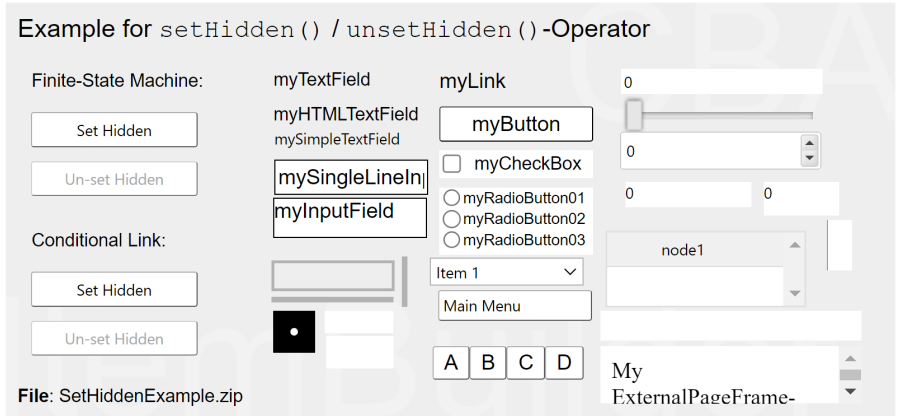


FIGURE 4.44: Item illustrating the setHidden() - / unsetHidden()-operator (html|ib).

conditional links to SingleLineInputFields Or InputFields with a named User Defined Id.

focus(UserDefinedId)

The item shown in Figure 4.45 illustrates the use of the focus()-Operator, used either in the Finite-State Machine or in Conditional Links.

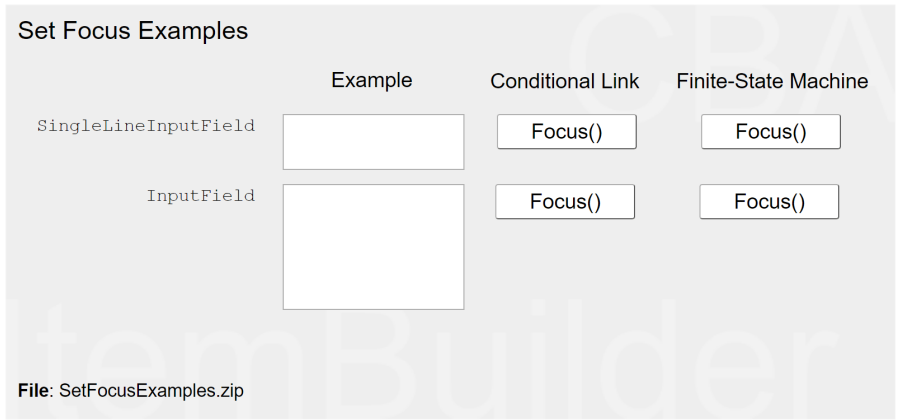


FIGURE 4.45: Item illustrating the focus()-operator (html|ib).

Operator to Insert Text into Input Fields: For technical reasons (e.g., when using touch screens) or for diagnostic reasons (e.g., when administering tests to children), it may be challenging to enter special characters or special characters in compo-

nents of type `SingleLineInputFields` and `InputFields` (see section 3.9.1). For these situations, CBA ItemBuilder provides an operator for inserting text:

```
insertText(InputField, TextToInsert, InsertPosition, DropLength)
```

The operator requires at least two arguments, the `UserDefinedId` of the component into which text is to be inserted (`InputField`) and the argument `TextToInsert` specified in quotes (e.g., " * "). The text can contain several characters. Figure 4.46 shows an example where the `insertText()`-operator is used together with the `focus()`-operator. Both operators can be used as operators in the finite-state machine and in conditional links.

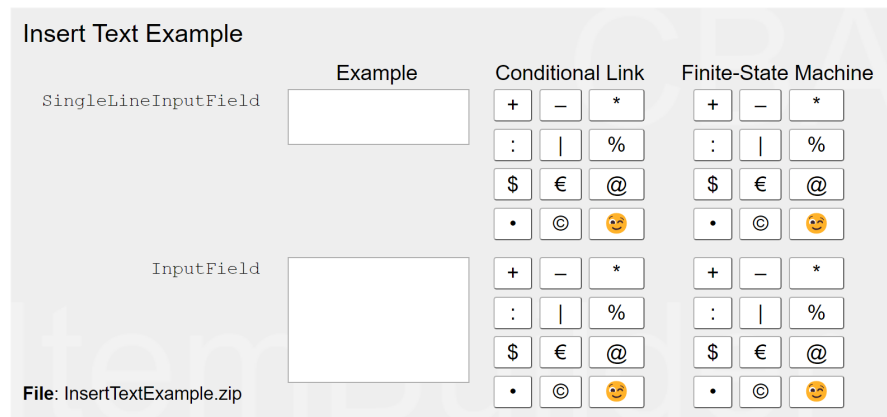


FIGURE 4.46: Item illustrating the `insertText()`-operator ([html](#)|[ib](#)).



Unicode characters can also be inserted into 'InputFields' and 'SingleLineInputFields' using the `insertText()` operator.⁶

If the argument `InsertPosition` is not specified, the value `-1` is used as default with the meaning that the text added to the end of the old text (i.e., the already existing text in the `InputField`). If `InsertPosition` is specified, an additional argument `DropLength` can be provided. If a value different from `-1` is provided for `DropLength`, the specified number of old characters starting with the `InsertPosition` to the end of the old text will be dropped. If not given it defaults to `-1` (i.e. drop all old characters after the insert position).

Operator to Select Components: Components that can be selected (Buttons with the property `Is Toggle: true`, Checkboxes and RadioButtons) can be selected (`setActive`) and deselected (`unsetActive`) from the *Finite-State Machine* or in *Conditional Links*:

```
setActive(UserDefinedId)
unsetActive(UserDefinedId)
```

The operators (see Figure 4.47) require a valid *User Defined Id* as an argument.

| Set Active Examples | | | | | |
|---------------------|--|------------------|-------------|----------------------|-------------|
| | Example | Conditional Link | | Finite-State Machine | |
| Button (Toggle) | <input type="button" value="myButton1"/> | setActive | unsetActive | setActive | unsetActive |
| | <input type="button" value="myButton2"/> | setActive | unsetActive | setActive | unsetActive |
| CheckBox | <input type="checkbox"/> myCheckBox1 | setActive | unsetActive | setActive | unsetActive |
| | <input type="checkbox"/> myCheckBox2 | setActive | unsetActive | setActive | unsetActive |
| RadioButton | <input type="radio"/> myRadioButton1 | setActive | unsetActive | setActive | unsetActive |
| | <input type="radio"/> myRadioButton2 | setActive | unsetActive | setActive | unsetActive |

File: SetActiveExample.zip

FIGURE 4.47: Item illustrating the `setActive()`-/ `unsetActive()`-operator ([html|ib](#)).

The operators `setActive()` and `unsetActive()` can be applied to frozen components. Constraints resulting from defined groups of components (i.e., `RadioButtonGroups`, see section 3.9.2, and `Frame Select Groups`, see section 3.9.4) are applied.

Operators for Text-Highlighting: The color for text highlighting can be defined using the following finite-state machine operators (see appendix B.2):

```
setGlobalProperty(highlight_color, "-5848680")}
```

Another operator is available for conditional links to enable and disable the highlighting-feature for text fields (see section 3.8.3 for more details):

```
setHighlightable(UserDefinedId)
unsetHighlightable(UserDefinedId)
```

The use of both operators is illustrated in Figure 4.48 (see also Figure 3.84 in subsection 3.8.3). To find the correct value for the third argument (i.e., the RGB color integer value), use the internal color editor to format, for instance, the background color of a panel. Copy then the value of the color property (from the *Properties* view) to the state machine syntax, and included it into the operator within double quotation marks (").

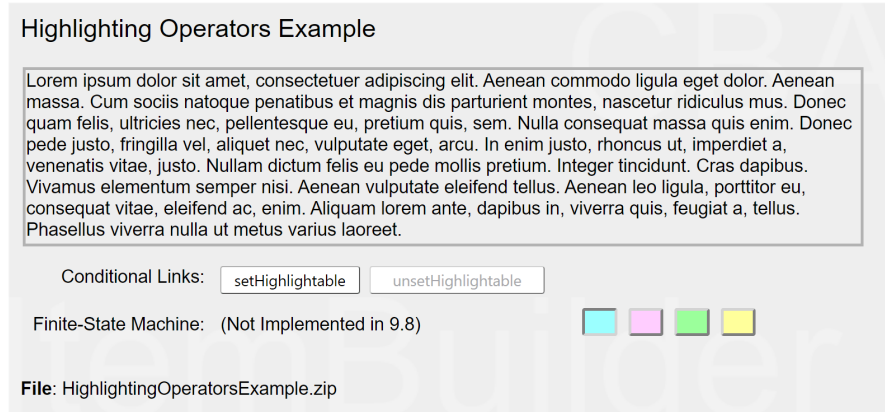


FIGURE 4.48: Item illustrating operators for text highlighting ([html5lib](http://html5lib.org)).

Operators to Set Values of Components: The default text of components to collect text responses (see section 3.9.1) can be defined using the property `text` (in the *Properties-view*). During runtime, text of input fields can also be changed using the `setInputValue()`-operator:

```
setInputValue(Source,Target)                                // Use-case 1
setInputValue(Source,Target,"New text for the source") // extended

setInputValue(Source,Source,"New text for the source") // Use-case 2
```

Source and Target are *UserDefinedIds* of `InputFields` or `SingleLineInputFields`. The `setInputValue()`-operator is illustrated in Figure 4.49. Use-case 1 shows how to copy the text from one input field (Source) to another (Target). The extended version shows how to specify an additional text, that is used as the new text in the source field, after the existing text is copied to the target field. Use-case 2 uses identical *UserDefinedIds* for source and target, but uses the additional text to set the value of a specific `InputField` or `SingleLineInputField`.

Operators for Frame Select Groups: As described above, components can be disabled with the `setFrozen()`-operator. The components will then remain visible but shown in a way that indicates that these components cannot be used. For components that allow selection (e.g., checkboxes, radio buttons, toggle buttons), it can also be helpful to disable the ability to select without displaying them visually differently. This functionality is suggested, for example, if radio buttons or checkboxes are shown in an instruction and when it is desired that the components are presented in the same way as they are used in the following tasks.

Frame Select Groups, as described in section 3.9.4, can be configured to be *not selectable*

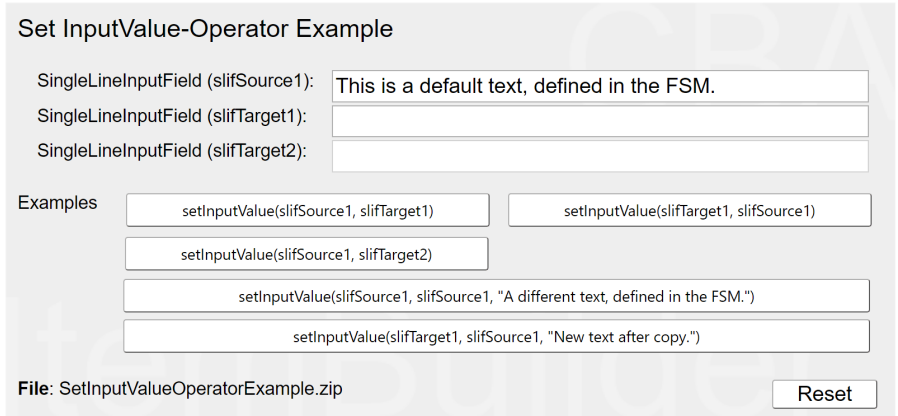


FIGURE 4.49: Item illustrating the setInputValue()-operator ([html](#)|[ib](#)).

at design time, and the operator unsetSelectable() can be used to change Frame Select Groups at runtime:

```
setSelectable(Frame, GroupNumber), unsetSelectable(Frame, GroupNumber)
```

The operator requires as the first argument the UserDefinedId of the Frame in which the Frame Select Group is defined. The second argument refers to the group index (i.e., the number corresponding to the Frame Select Group, starting with 0). The item shown in Figure 4.50 illustrates the use of the setSelectable()- and unsetSelectable()-operator in the *Finite State Machine*.

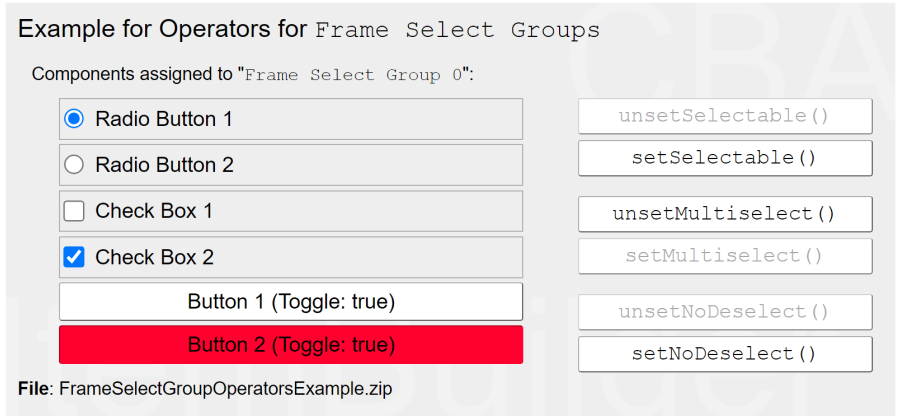


FIGURE 4.50: Item illustrating the Operators for Frame Select Groups ([html](#)|[ib](#)).

Also, the other properties of frame select groups (i.e., *Multiple Select* and *No Deselect*,

see section 3.9.4 for a description) can be modified at runtime using the following operators from the *Finite-State Machine*:

```
setMultiselect(Frame, GroupNumber), unsetMultiselect(Frame, GroupNumber)
unsetNoDeselect(Frame, GroupNumber), setNoDeselect(Frame, GroupNumber)
```

Note that previous versions of the CBA ItemBuilder used implicit select groups within containers (i.e. all components within a container such as `Panel`, `ImageMap`, `RadioButtonGroup`, etc.). For this reason, the syntax `setSelectable(Container or Table)`, `unsetSelectable(Container or Table)`, `setMultiselect(Container)`, `unsetMultiselect(Container)`, `unsetNoDeselect(Container)` and `unsetNoDeselect(Container)` is still valid. However, implicit select groups are deprecated and replaced by the `Frame Select Groups` (see section 3.9.4).

Operators for `MapBasedValueDisplays` (i.e., Drag-and-Drop): The ability to perform drag and drop operations with ‘`MapBasedValueDisplays`’ (see section 4.2.6) is activated by defining the ‘Drop Mode’ property different from `DROP_NONE`. With the help of the `setValueDisplayMode()`-operator this property can be also controlled from the *Finite-State Machine*:

```
setValueDisplayMode(UserDefinedId, Mode)
```

The followings keywords are defined for the argument `Mode`:

- `dd_none`: The component does not allow dragging or dropping.
- `dd_drag`: The component only allows dragging.
- `dd_drop`: The component only allows dropping.
- `dd_dragdrop`: The component allows both dragging and dropping.

Operators to Trigger Commands: Navigation between *Tasks* using *Commands* is essential for the use of assessment components created with the CBA ItemBuilder (see section 3.12). In parallel to *Commands* that can be linked directly to buttons, the CBA ItemBuilder provides operators for the *Finite-State machine* to enable navigation between *Tasks*:

```
next_task()
back_task()
cancel_task()
```

The operators are not available for *Conditional Links*.

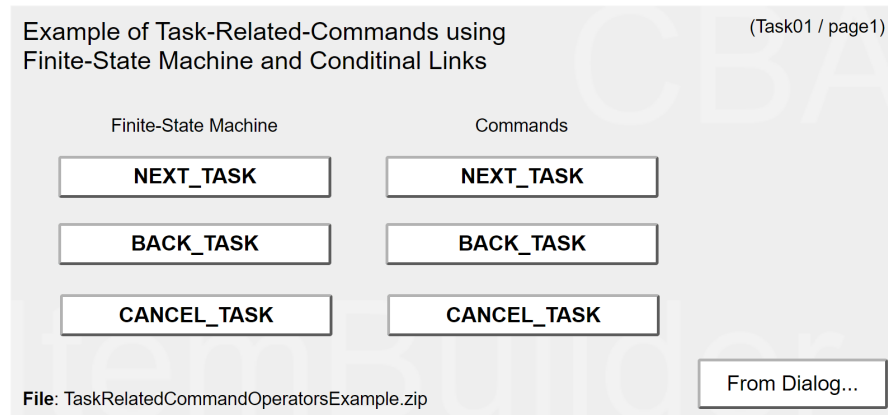


FIGURE 4.51: Example for the task-related operators ([html](#)|[ib](#)).

Note that, if supported by the deployment software (see chapter 7), the `next_task()`-operator is also prepared to take either a `TaskName` or a `TestName` and a `TaskName` as arguments.

Operators for Dialog Pages: The `openDialog()`-operator can be used to open a particular page as regular dialog at a specific position provided as *X* and *Y* coordinate (see Figure 4.52).

```
openDialog(PageName, X, Y)
```

Note that the page can only be opened once (either left or right) and that the modal dialog is configured using the frame as `Dialog=MODAL_DIALOG` and `Closable=false` (see section 3.15.1).

Dialog pages can be closed with the `closeDialog()`-operator (when visible):

```
closeDialog(PageName)
```

Instead of specifying the name, dialog pages can also be closed using the flags `isXPage` and `isModal`:

```
closeDialog(isXPage, isModal)
```

Operators for Scrolling: If necessary, the scrolling of pages can be triggered from

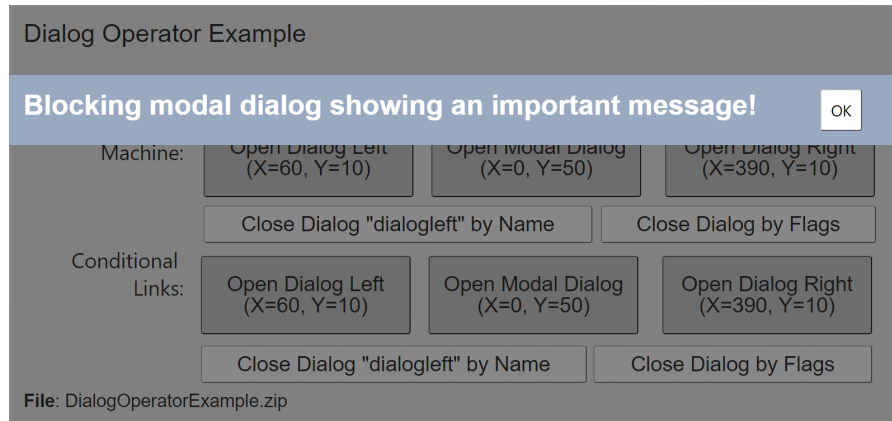


FIGURE 4.52: Example for the `openDialog()` and `closeDialog()`-operators ([html](#)|[ib](#)).

the *Finite-State Machine*. For main pages (regular pages or X-pages), the following operator is available:

```
scrollTopLevelPage(isXPage, PositionParameter, PositionParameter)
```

The argument `isXPage` can be set to `true`, if the *XPage* should be scrolled. The `Position-Parameter` can either be defined in % or in pixels, as shown in the following examples:

```
scrollTopLevelPage(false, yPosition=>0px)
scrollTopLevelPage(false, yPosition=>50%)
scrollTopLevelPage(false, xPosition=>30px, xPosition=>50%)
scrollTopLevelPage(false, xPosition=>0px, xPosition=>0px)
```

A second operator is provided to scroll embedded pages, illustrated in Figure 4.53.

```
scrollEmbeddedPage(PageArea-ID, PositionParameter, PositionParameter)
```

Operators for Media Components: An operator to start, stop and resume audio and video outputs can be used for multiple purposes. As shown in Figure 4.54, it can be used to limit the frequency of audio output. It is also possible to automatically start audio or video output when a page is visited, a state occurs, or an event is triggered. Finally, this operator is essential to replace the default control buttons for ‘Audio’ and ‘Video’ components.

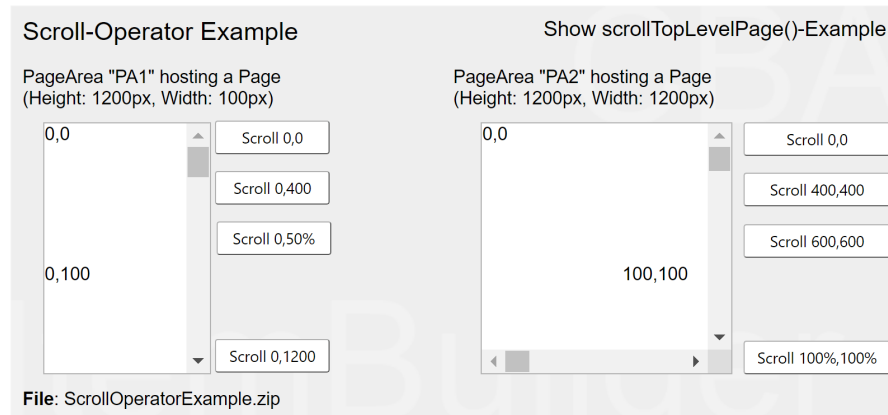


FIGURE 4.53: Example for the `scrollEmbeddedPage()`-operator ([html|ib](#)).

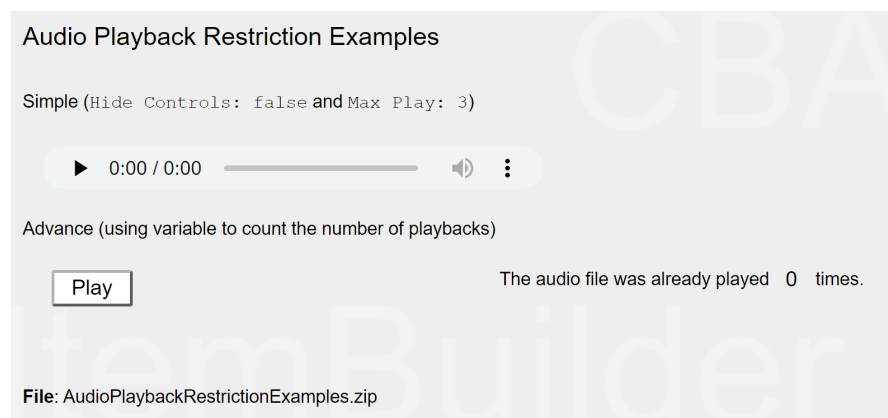


FIGURE 4.54: Item illustrating restriction to play media components ([html|ib](#)).

To start, stop and pause the audio or video output, the CBA ItemBuilder provides the `setMediaPlayer()` operator. This operator takes two arguments. Argument 1 specifies which audio or video component should be changed with the User Defined Id. Argument 2 is the action to be performed:

```
setMediaPlayer(ComponentID, Action)
```

The requested action can be either `mp_start`, `mp_stop` or `mp_pause`. The item in Figure 4.55 provides an example.

A similar operator can also be used to control the volume of audio or video outputs. The `setMediaPlayerVolume` operator also needs as the first argument the User Defined

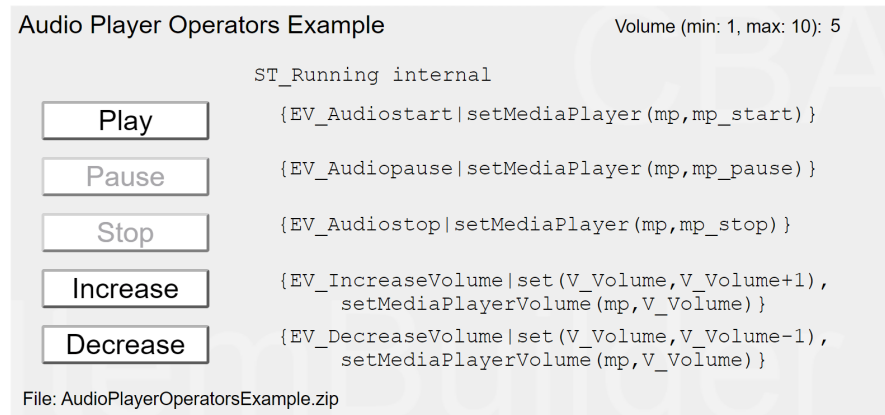


FIGURE 4.55: Item illustrating operators for media components ([html|ib](http://html5lib.org)).

id of the component to be changed and as the second argument the volume to be set. This operator can also be called before the first audio output to control the default volume:

```
setMediaPlayerVolume(ComponentID, Value)
```

The requested value is an integer value between 0 (not audible or silent) and 10 (maximum volume).

In the *Task Initialization* syntax (see section 4.5), the properties of media components can also be defined. The `initMediaPlayer()` operator is available for this purpose:

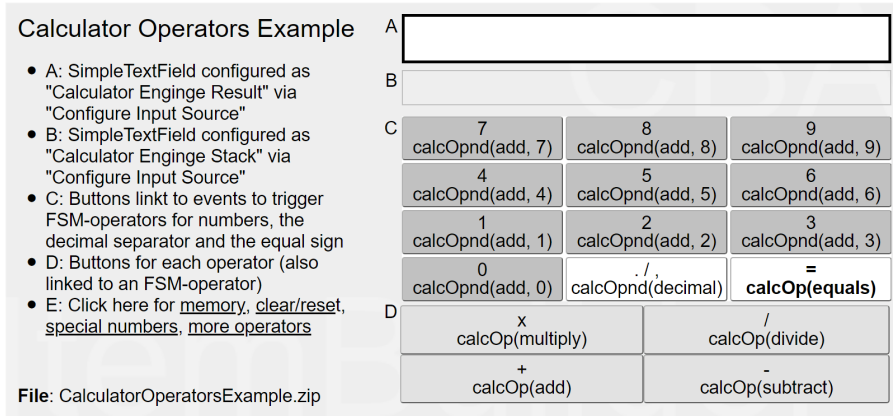
```
initMediaPlayer(ComponentID, Property, Property, ...)
```

The `ComponentID` is the `UserDefinedId` of a component of type `Audio` or `Video`, and the property values are specified as shown in the following example:

```
{page:true|initMediaPlayer(mp,automaticStart=>true, hideControls=>true, maxPlay=>2)}
```

Note that the `initMediaPlayer()`-operator is only available in the *Task Initialization* syntax and that the operators `'setMediaPlayer()'` and `'setMediaPlayerVolume()'` are only available in the *Finite-State Machine* syntax.

Operators for Calculator: The CBA ItemBuilder provides a *Calculator Engine*, which can be used to convert different calculators. The connection of the *Calculator Engine*



```
calcSettings(Parameter, Parameter, ...)
```

Operators for Embedded Pages: Links in pages embedded are processed *within* the `PageArea` of the source page, and the `setEmbeddedPage()`-operator that was already introduced in section 4.3.4 can be used to change an embedded page of any `PageArea` specified by the *User Defined Id*:

```
setEmbeddedPage(PageAreaUserDefinedId, PageName)
```

Within a given CBA ItemBuilder project file, the *User Defined Id* of all components must be unique and the CBA ItemBuilder ensures that no *User Defined Id* is used more than once (see section 3.7.4). Moreover, pages have unique names and the CBA ItemBuilder only allows the specification of page names not yet used within a project file (see section 3.4). However, multiple components of type `PageArea` (see section 3.5.4) can be used to embeds identical pages. Accordingly, the complete path of *User Defined Ids* is necessary to specify which page should be changed by the `setEmbeddedPage()`-operator. Figure 4.57 shows that the `setEmbeddedPage()`-operator can also be used to switch pages within pages (i.e., nested `PageAreas`), whereby the *User Defined Id*'s of the `PageArea`-components must be concatenated with a dot:

```
setEmbeddedPage(ParentPageArea.ChildPageArea, PageName)
```

Since this logic is a common source of errors when using nested pages, it has been explicitly mentioned again here (see also sections 4.1.4 and 5.3.9).

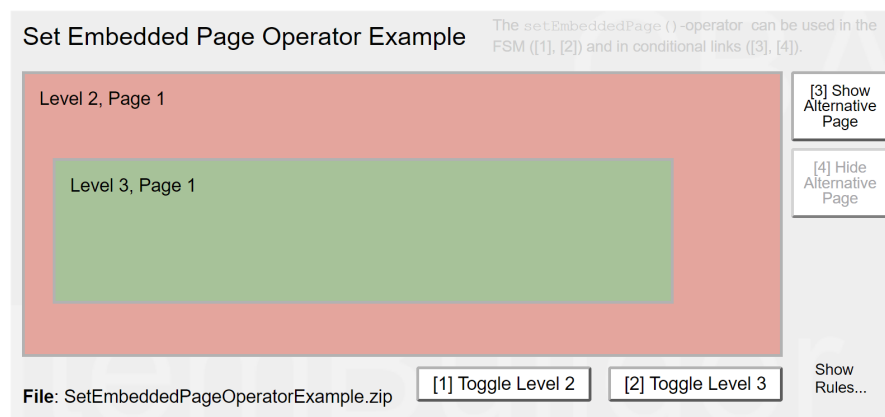


FIGURE 4.57: Example for `setEmbeddedPage()`-operator ([html](#)|[ib](#)).

As shown in Figure 4.57, the `setEmbeddedPage()`-operator can be used in both the finite-state machine and conditional links. When first called, the operator overwrites the static assignment made when configuring the `PageArea` via the context menu and the *Link Embedded Page* entry (see section 3.5.4).

Note that the `setEmbeddedPage()`-operator does not work for `TreeChildAreas` since these are controlled by their attached tree (see section 3.9.9).

Operator to Raise / Trigger Events: Mainly for technical reasons and to structure or simplify the finite-state machine syntax, additional events can also be triggered as operators. For this requirement the `raise()` operator is available in the finite-state machine syntax, which expects the name of a defined event as argument.

```
raise(EventName)
```

Events can also be triggered in *Conditional Links*. The following operator can be used for this purpose (see Figure 4.58 for an example):

```
initFSM(EventName)
```

Example for Operators `InitFSM()` and `raise()`

(page)

MyLink

For the component of type `Link` with the text "MyLink" a *Conditional Link* with the following *Condition* is defined:

```
{page:true|initFSM(EV_1)}
```

Since the link is already displayed on the "page", clicking on the component does not change the current page. However, the event `EV_1` is triggered via `InitFSM(EV_1)` and processed in the finite-state machine with the following syntax:

```
Events: EV_1, EV_2;
Rules: ST_Start->ST_Running{true}
ST_Running internal {EV_1 : {round(Var1/2)*2==Var1} | set(Var1,Var1+1),raise(EV_2)}
{EV_1 : {round(Var1/2)*2<>Var1} |set(Var1,Var1+1)}
{EV_2 | set(Var2,Var2+1)}
```

Var1=0

Var2=0

Only if the value of variable `Var1` is even, another event is triggered during the processing of `EV_1` with `raise(EV_2)`, which leads to the value of `Var2` also being increased by one.

File: RaiseFSMEventsExample.zip

FIGURE 4.58: Example for `raise()` and `initFSM()`-operator ([html](#)|[ib](#)).

Operators to Change Events: *Timed Events* as introduced in subsection 4.4.3 are triggered automatically after a specific amount of time. For that purpose, the interval is specified, when the *Timed Event* is defined in the *State Machine Rules File*. Remember that timed events are automatically started when a state is entered and that an *internal* transition is different from a self-transition (see subsection 4.4.7). Hence, *Timed Events* are often used multiple times (because, for instance, the current state of a *Finite-State Machine* is changed triggered by the *Timed Event*, meaning the timer

is restarted every time). A specific `setFSMEvent()`-operator is provided to modify the time interval for a *Timed Event*:

```
setFSMEvent(EventName, Time)
```

The item in Figure 4.59 illustrates the `setFSMEvent()` operator.

FIGURE 4.59: Example for `raise()` and `initFSM()`-operator ([html](#)|[ib](#)).

The item in Figure 4.59 provides an example also for another operator, that changes the page that is assigned to a state (see section 4.4.9 for the concept of *Pages* assigned to *States*):

```
setFSMState(StateName, PageName)
```

In the example in Figure 4.59 the button `Remain Page1` and `Alter Pages` trigger conditional links that contain the `setFSMState()`-operator. If the button `Remain Page1` is clicked, the state `ST_NavigationPage2` is assigned to the start page of the *Task* (i.e., `page1`). But if the button `Alter Pages` is clicked, the conditional link triggers the operator `setFSMState(ST_NavigationPage2, page2)` and changes the page that is assigned to the state `ST_NavigationPage2`.

The `setFSMEvent()`- and `setFSMState()`-operators are only available for *Conditional Links* and *Task Initialization* and only integer numbers can be assigned to *Timed Events*.

Operators to Create Trace Messages: *Events* in this chapter always refer to the events defined in the syntax (see section 4.4.3), which are used to control the finite-state machine. With the help of special operators, however, entries can also be written to the log data (i.e., trace messages can be created):

```
trace_text(text)
```

The `text` argument is interpreted as *Argument Lists* (see section 4.1.5), so that additional argument such as variables can be used to include additional values in the stored `text`.

```
trace_typed_text(type, text)
```



The assessment components created with the CBA ItemBuilder provide *Raw Log Events* without further effort. To create *Contextualized Log Events*, which indicate specific test-taker behavior (see section 2.8.1), the FSM operators `trace_text()` and `trace_typed_text()` can be used.

Item authors can use the following operators to trigger the creation of a snapshot:

```
trace_snapshot(TextSource)
trace_snapshot(TemplateSource, ValueSource, ValueSource, ...)
trace_typed_snapshot(Type, TextSource)
trace_typed_snapshot(Type, TemplateSource, ValueSource, ValueSource, ...)
```

Operators for Tree-Components: To make changes to `Tree` components, the CBA ItemBuilder provides the following two operators:

```
tree_move(Tree, NodeIdPattern)
```

```
tree_copy(Tree, NodeIdPattern)
```

Operator to Measure (Elapsed) Time: Timed events (as described in section 4.4.3) give item authors the ability to define actions or changes after a specified time. Hence, timed events allow various scenarios to be implemented, but no precise time measurement is possible when user interactions need to be incorporated. For this use case, the `elapsedTime()` operator can be used, which returns the elapsed time since the start of a task in milliseconds.

```
elapsedTime()
```

Figure 4.60 illustrates the use of the `elapsedTime()`-operator to measure the time between two clicks:

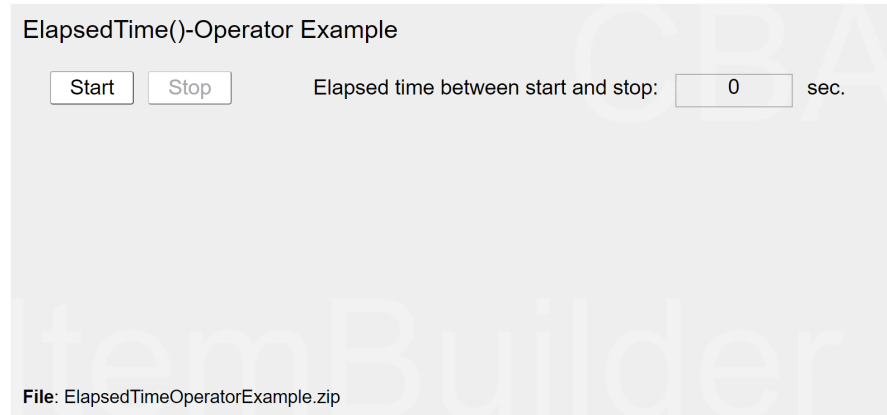


FIGURE 4.60: Example illustrating the `elapsedTime()`-operator ([html|ib](#)).

4.4.7 Trigger Operators without Transitions

Finite-state machine rules consist primarily of defined *Transitions* from a state A to a state B (see section 4.4.4). When a transition is triggered by an event, additional operators can be executed. Hence, this part of the finite-state machine concept of the CBA ItemBuilder focuses on transitions, i.e., state changes.

Operators can also be executed *without* transitions, either using *Conditional Links* (see section 4.3) or within the finite-state machine. For this purpose, the following syntax provides the possibility to define operators that are executed when a particular finite-state machine enters or leaves a specific state. These operators will be executed regardless of the transition that changed the state (i.e., the focus is on states, not transitions).

State Entry and State Exit: Operators that are to be executed when a finite-state machine changes to a state can be defined using the following syntax:

```
State entry {operator1(), operator2()}
```

The name `State` must be the name of a valid state, defined in the *State Machine Tree*

View (see section 4.4.2), followed by the keyword `entry` and a comma separated list of operators within curly brackets. The specification of an event name and a `|`-character are not necessary, since the operators are executed independently of the event or transition that brought the finite-state machine into the state specified as *State*.

Operators can also be defined that are always executed when a state is exited:

```
State exit {operator1(), operator2()}
```

If multiple state-entry or state-exit definitions are provided for a state, then all operators are executed, i.e., the following definition results in the same two operators executed when either *State1* or *State2* is entered:

```
State1 entry {operator1(), operator2()} // One statement

State2 entry {operator1()}             // Two statements
State2 entry {operator2()}
```

Internal Processing of Events: Processing events to execute operators without effectively changing the current state of a *Finite-State Machine* is possible, for instance, with the transition *State_A => State_A* (Self-Transitions). However, this is defacto a transition from *State_A* to *State_A* and state-entry and state-exit are triggered. If operators within a state should be triggered by a particular events without* the state being changed via a transition, this is possible with the keyword `internal`:

```
State internal {Eventname | operator1(), operator2()}
```

The *Self-Transition* *State => State {Eventname | operator1(), operator2()}* is only similar to the definition *State internal {Eventname | operator1(), operator2()}* with respect to the provided possibility to process events without defectively changing the state of a finite-state machine. However, only the internal processing of events defined with the syntax above is executed without triggering state-entry and state-exit (see Figure 4.61). Since *Timed Events* are restarted on state changes (and also on *Self-Transitions*), the distinction between *State internal {TimedEvent|/*...*/}* and *State =>State {TimedEvent|/*...*/}* is central to be able to control the behavior of *Timed Events* purposefully (see Figure 4.63 in section 4.4.3).



An event can trigger only one rule for each finite-state machine. If both `internal` processing and self-transitions to the same state are defined, the `internal` processing is prioritized. Only self-transitions trigger state `entry` and `exit`.

Keep in mind, that each event is only processed once by a rule that is defined for a finite-state machine in a particular state. Rules of type `internal` are interpreted prior to the rules pointing from a particular state to itself. Hence, in the example shown in Figure 4.61 the transitions `ST_A => ST_A` and `ST_B => ST_B` will never be executed (regardless of the order in the finite-state machine syntax definition) in mode `Implicit internal`, because the definitions `ST_A internal` and `ST_B internal` are processed first.

Transition Comparison Example

| | |
|-------------------|---|
| V Counter Entry A | 1 |
| V Counter Entry B | 0 |
| V Counter Exit A | 0 |
| V Counter Exit B | 0 |

| | |
|----------------------------|---|
| V Counter Internal A | 0 |
| V Counter Internal B | 0 |
| V Counter TransitionSelf A | 0 |
| V Counter TransitionSelf B | 0 |

Buttons: EV_A, EV_B, EV_Click

State: ST_A

File: TransitionComparisonExample.zip

Mode: State => State, Implicit internal, Explicit internal

Show Finite-State Machine Rules...

FIGURE 4.61: Example for transitions `internal`, `entry` and `exit` ([html](#)|[ib](#)).

The *Internal Processing of Events* can, analogous to *Conditional Rules* (see section 4.4.5) be restricted by *Conditions* (i.e., *Guards*, see section 4.4.5), which can be formulated with the help of variables.

Note that a single event can, however, be processed by multiple nested state machines, as described in the next section (see section 4.4.8).

4.4.8 Nesting Finite-State Machines (in Regions)

If several independent sets of states are to be distinguished or, for example, several timers are required within a *Task*, the limits of a single *finite-state machine* are reached quickly. For this reason, the CBA ItemBuilder provides the possibility to use several *Finite-State Machines*, which can be nested within each other. This functionality is illustrated in the item in Figure 4.62. The outer *Finite-State Machine* starts in the state `ST_OuterStart`, which is exited directly when the *Task* is loaded with the rule `ST_OuterStart -> ST_OuterMain{true}`. Within this state, two *Finite-State Machines* are defined in parallel. This is done by adding two regions (`Region1` and `Region2`) in the *State Machine Tree View*, as shown in Figure 4.62. Each region contains a set of states, including precisely one *Start* state and (one or) multiple *Regular* states.

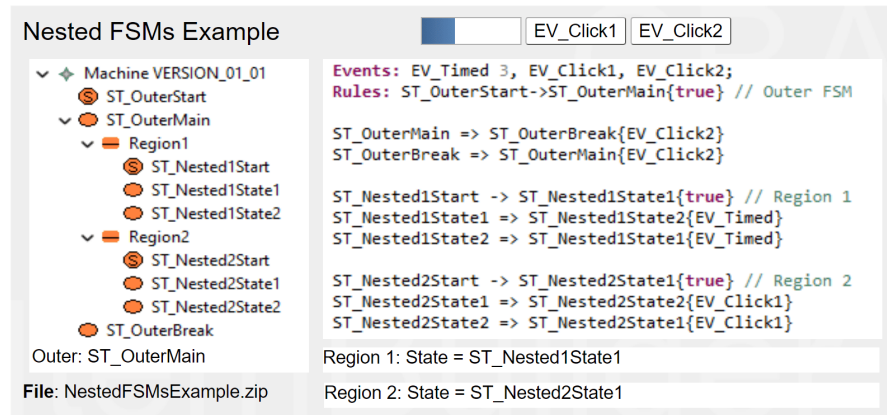


FIGURE 4.62: Example for multiple nested *Finite-State Machines* ([html](#)|[ib](#)).

An example illustrating the operation of nested finite-state machines in interaction with *Timed Events* is shown in Figure 4.63.

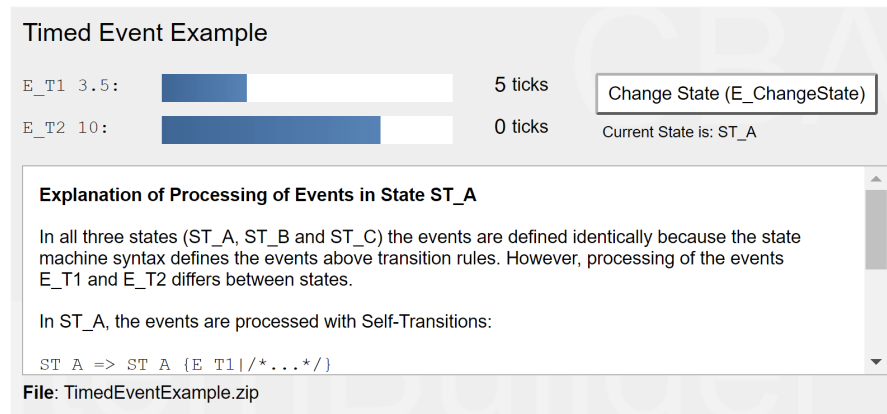


FIGURE 4.63: Item illustrating *Timed Events* and *Regions* ([html](#)|[ib](#)).

4.4.9 Assignment of Pages to States

Normal and end states can be connected with pages. If a transition is triggered that changes the *Current State* to a state assigned to a page, the linked page is shown.

Simple Example: The following example in Figure 4.64 illustrates how pages can be changed using state transitions in a finite-state machine. In the example, two states are defined: One state `ST_First` and one state `ST_Second`. Changing between states is operationalized by two buttons that are triggering the events `EV_GoToSecond` and

EV_GoToFirst, respectively. State ST_First is connected with page First and ST_Second is connected with page Second. In this setup, the pages are not linked to the buttons, but to the states:

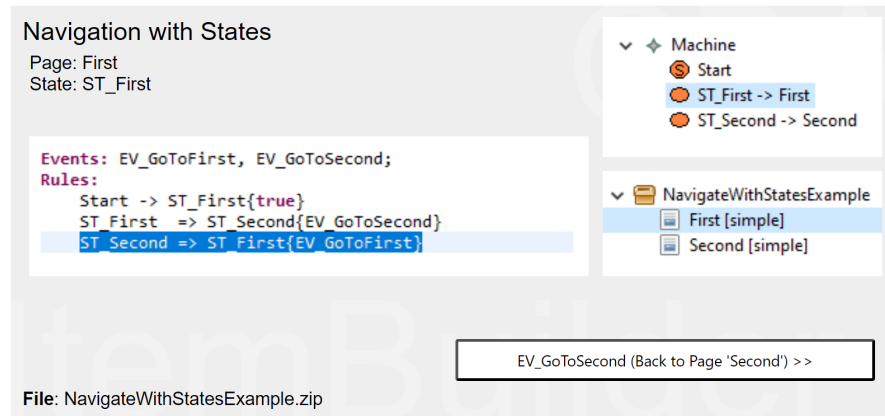


FIGURE 4.64: Example for navigation with states ([html](#) | [ib](#)).

In order to assign a page to a state, a state must be selected in the *State Machine Tree View* of the CBA ItemBuilder. Afterward, a right-click gives access to the context menu, which contains the entry *Configure States*.

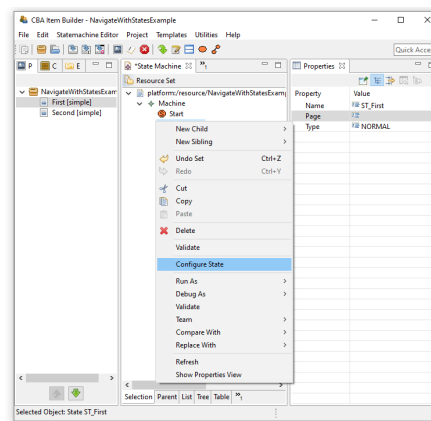


FIGURE 4.65: Context menu *Configure State* to assign a *Page* to a *State*.

The opening dialog *Configure State* allows to define which page should be opened when changing to this state. One of the defined pages can be selected from the combo box. An empty entry corresponds to the setting that the currently displayed page should not be changed when changing state.

After the dialog was closed with **OK**, the selection is also visible in the tree view of the

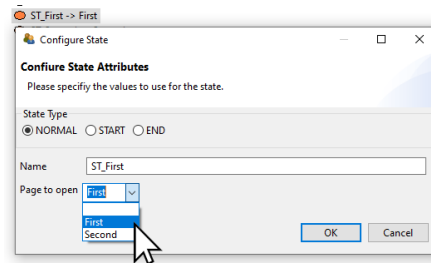


FIGURE 4.66: Dialog *Configure State* showing *Page to open* to assign a *Page* to a *State*.

finite-state machine. The arrow with the reference to a page name indicates that this state is linked to a page.



Tip: It is also possible to make the settings in the *Properties* view. Make sure that the spelling of the page name in the *Page*-property corresponds exactly to the name that is also displayed in the *Project View*.

Advanced Example using X-Pages and Pages Assigned to States: Let us now consider the example, which has already been described and discussed in subsection 3.11.4 to illustrated linking between pages and x-pages (see Figure 3.137).

In addition to buttons with links to pages and modal dialogs (see section 3.11.2), selected buttons are also linked to events of the finite-state machines. Using two nested finite-state machines (see section 4.4.8), one particular state always corresponds to the current page and one state corresponds to the current x-page. Note the region named *x* and the region named *Regular*. Within each region, a start-state and two states are defined, each assigned to a page or a X-Page (see Figure 4.67).

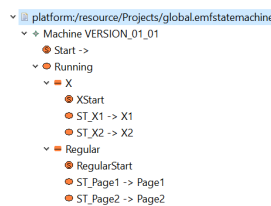


FIGURE 4.67: States *Linked to Pages* in two *Regions* as implemented in the item shown in Figure 3.137.

The buttons in the example item with the labels *EV_X1*, *EV_X2*, *EV_Page1* and *EV_Page2* are assigned to the events of the same name. The events are defined in the finite-state machine syntax and are used in the following rules:

```

Events:EV_X1, EV_X2, EV_Page1, EV_Page2; // Definition of events
Rules: Start -> Running{true}           // Start of the Region "Running"

XStart -> ST_X1{true}                    // (Nested) FSM for X-pages in
ST_X1 => ST_X2{EV_X2}                    // the region "X"
ST_X2 => ST_X1{EV_X1}

RegularStart -> ST_Page1{true}           // (Nested) FSM for pages in
ST_Page1 => ST_Page2{EV_Page2}           // the region "Regular"
ST_Page2 => ST_Page1{EV_Page1}

ST_X2 => ST_X2{EV_X2}                    // Necessary, since the page
ST_X1 => ST_X1{EV_X1}                    // is also changed via links
ST_Page1 => ST_Page1{EV_Page1}
ST_Page2 => ST_Page2{EV_Page2}

```

Finally, let's look at the use of state and page linking in the context of dialogs. If you click the `XDialog`-button in Figure 3.137, then the page `XDialog1` is displayed as a modal dialog (button `Dialog` and page `Dialog1` in parallel). These dialog-pages each contain a button that triggers an event in the finite-state machine, and the defined state transition rules lead to new states that are linked to pages again.

If, for example, on page `x1` (the finite-state machine will be in region `x` in state `ST_X1`), the event `EV_X2` is triggered, and the FSM will change to state `ST_X2` because of the rule `ST_X1 => ST_X2{EV_X2}`. Since the page `x2` is assigned to this state, it is displayed. However, page `x2` is not configured as a dialog, meaning that the underlying page will be changed even though the event was triggered from a dialog page. The dialog page remains visible. To close the dialog, you have to use the command `CLOSE` (see subsection 3.12.2 for more information about *Runtime Commands*), which is only assigned to the button `CLOSE` in this example. The `CLOSE` command can either be made available separately via an additional button (as it is implemented in `Dialog1` in the example) or the `CLOSE` command can be linked to the button in addition to the event (as it is implemented in `XDialog1` in the example). Not shown in this example: If a dialog is to be exchanged with a page linked via states, it is sufficient if the target page is configured as a dialog.

Change Page and X-Page Simultaneously: The separation of pages into the two categories (regular pages and X-pages) simplifies most scenarios, where *Links* as well as *Pages Linked to States* are either affecting the regular page area or the X-page area. To change a page and a X-Page simultaneously in an item at once, the connection of states and pages must be separated into two regions again, since each state can only be linked to one page.

Apart from this procedure, in which the pages are switched via the finite-state machine, the entire layout including the *X-Page* pages can also be changed by navigating

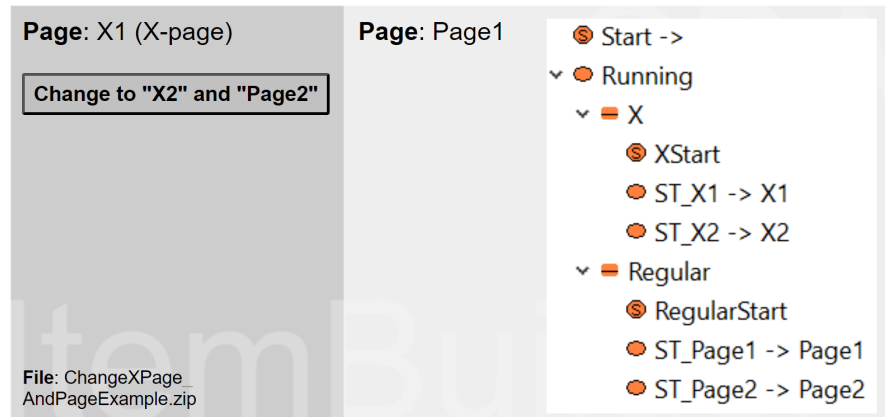


FIGURE 4.68: Example for changing *Page* and *X-Pages* simultaneously ([html](#)|[ib](#)).

to another task (see 3.6.2 for details on CBA ItemBuilder *Tasks* and see 3.12 for details on switching between tasks using runtime commands).

Change Page with States and Timed Events: The mechanism to change pages linked to states in the finite-state machine opens up a variety of design possibilities. For example, in combination with timed events (see section 4.4.3) items with controlled presentation time can be implemented.

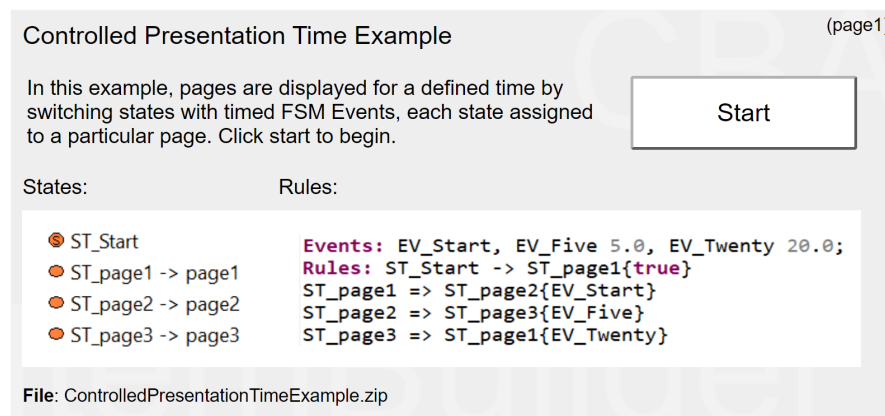


FIGURE 4.69: Example for a timed presentation using states and timed events ([html](#)|[ib](#)).

4.4.10 Timer Component

Time constraints in assessments can be implemented in different ways. The decisive factor is whether the restriction is to be implemented within a task or across several *Tasks*. In a nutshell: If a *Page* is to be displayed within an item after the timeout, the time limit must be implemented with the CBA ItemBuilder project. If another task is to be displayed afterward, the time limit must be implemented at the test delivery level. Therefore, in the concrete implementation, it is also essential to consider how assessment content is broken down into ItemBuilder projects (i.e., into individual *Tasks*, see section 8.2).



Time restriction *across* multiple tasks (i.e., CBA ItemBuilder project files and tasks used as entry-points) are to be implemented within the deployment software (see chapter 7).

Timed behavior of items and time constraints *within* tasks can be implemented with *timed FSM events* (see section 4.4.3). Timed events are invisible and defined as recurring events, automatically triggered and processed when rules are defined in the finite-state machine (see section 4.4.4). To visualize selected timed events, a specific component is provided by the CBA ItemBuilder that can be added to pages in the *Page Editor*. To visualize the remaining time of a timed FSM event (backwards) or the time since the last occurrence of a timed event (forwards), the *Timer*-component shown in Figure 4.70 can be used.

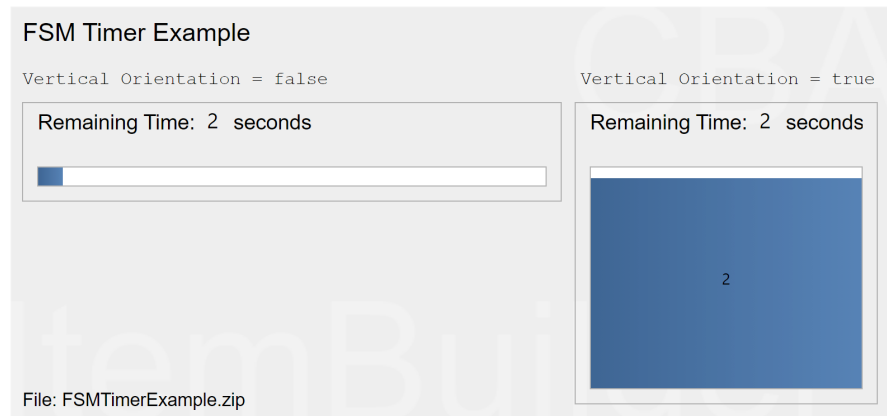


FIGURE 4.70: Item illustrating component of type *Timer* ([html|ib](#)).

Timer-components must be linked to a *timed FSM event* (assigned using the context menu entry *Link Timer Event*) and support the property *Vertical Orientation*, can show the remaining time automatically (*Show Remaining Seconds*), and can run either *Run Forward=true* Or *false*.

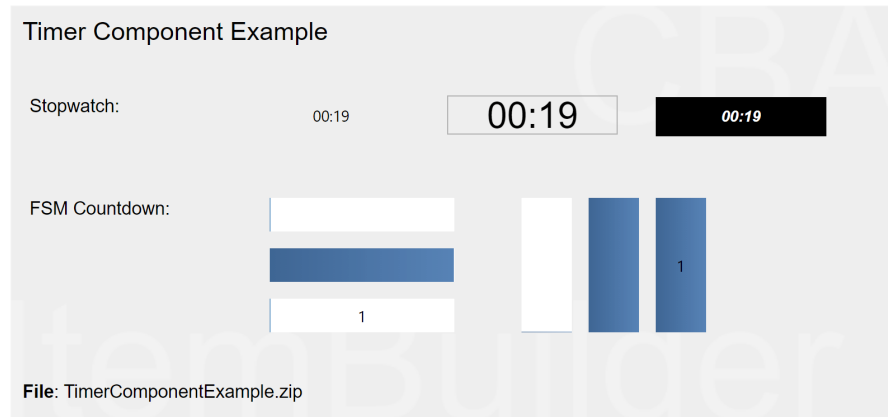


FIGURE 4.71: Item illustrating component of type `Timer` ([html|ib](http://html5lib.org)).

4.5 Task Initialization Syntax



CBA ItemBuilder project files must define at least one *Task* as the entry point (see section 3.6). However, they can also contain several *Tasks*. The delivery software can then determine which order of *tasks* taken from one or from multiple CBA ItemBuilder project files is used (see section 7.2.7). In the *Preview* (see section 1.4.2), the tasks are displayed according to the order in which they are listed in the *Tasks* editor, and the order can be adjusted there (see section 3.6). In the *Tasks* editor, it can also be defined which page and, if necessary, which additional *X-Page* is displayed at the beginning of a *Task*. However, all *Tasks* share the *Variables* and the *Finite-State Machine(s)* among themselves. As described in section 4.4.4, the *Task Name* can be used in conditions of the *Finite-State Machine*, for instance in the *Start Rule (Start Transition)*. With the *Task Initialization Syntax*, the CBA ItemBuilder provides an additional possibility to define *Task*-specific default values. Hence, the *Task Initialization* syntax is of particular importance if within one CBA ItemBuilder project several *Tasks* are defined.

The general syntax follows the syntax of *Conditional Links*:

```
{Page: Condition | Operator1() Operator2() }
```

Example

```
{ page1 : true | set(V_Task,2)}
```

Multiple operators can be defined, separate by white space.

Task Initialization Example 1

Page Name: pageTask01

☐ Use first active hit per class (applies to all tasks). ☐ Use first active miss per class (applies to all tasks).

| Name | MinHits | | Start Page | Start X-Page | Scoring | Scored Task |
|--------|---------|---|------------|--------------|---------|-------------|
| Task01 | 1 | ✔ | empty | | | |
| Task02 | 1 | ✔ | empty | | | |

File: TaskInitializationExample.zip NEXT-TASK Command

FIGURE 4.72: Item illustrating *Task Initialization Syntax* ([html](#)|[ib](#)).

4.6 Interactive Content in ExternalPageFrames

Assessments created with the CBA ItemBuilder can use existing *HTML5/JavaScript* content. As described in section 3.14, *HTML5/JavaScript* content can be embedded using components of type `ExternalPageFrame`. (i.e., components that are integrated into the CBA ItemBuilder-based items as `iframes` during runtime, see section 3.14).



This section is not intended for *item authors* who want to use the CBA ItemBuilder to computerize assessment content, but for software developers, that were asked to support a particular assessment project.

ExternalPageFrames are not suitable for item authors who want to create computer-based tasks with the CBA ItemBuilder without programming in a concrete programming language (see section 2.11.2). External content must be created in a way that this HTML-content supports a) the API of the CBA ItemBuilder for the transfer of data and b) can be used in the intended way in all browsers potentially used in deployments. Moreover, long-term archiving of content independently of the used rendering technology is not possible for content implemented directly in HTML5 / JavaScript. If HTML5 / JavaScript content is prepared appropriately, it can be integrated into the CBA ItemBuilder using the *Embedded HTML Explorer* as described in section 3.14.2.

4.6.1 Package External Content for CBA ItemBuilder

For software developers who want to create content for *ExternalPageFrames* without studying the CBA ItemBuilder in detail, the crucial aspects are summarized below:

- **(Screen) Size for External Content:** Items in the CBA ItemBuilder have a fixed size (in pixel, see section 3.2.2 for details). The deployment software used to deploy assessments can align the content in the view port of the web-browser (either in window mode or in full-screen mode). To allow deployments on heterogeneous hardware (with respect to screen size and resolution) while maintaining standardization of the presentation (no responsive design), proportional scaling can be activated. Proportional scaling is implemented as CSS transformation of the *iframe* that embeds the HTML/JavaScript runtime component of the CBA ItemBuilder. If the item itself is scaled, all components including the *ExternalPageFrames* are scaled.
- **Size and Margins:** The content is embedded as *iframe* (called *ExternalPageFrame* within the graphical *Page Editor* of the CBA ItemBuilder). The embedding is done with a fixed size defined in pixels. To avoid unwanted scrollbars at runtime, the provided HTML files have to convert the frames and size, e.g. via CSS, so that they fit into available size of a particular *ExternalPageFrames*. In practice, it has been proven that item authors find out the available space with a placeholder document, and then read the *Width* and *Height* of the *ExternalPageFrames* in the *Page Editor* and use them for the creation of the external content.
- **Embedding Files and Resources:** Assessments created with CBA ItemBuilder can be delivered online and offline (e.g., USB sticks). Hence, using external sources (i.e., libraries or content referenced from any URI) can be impossible or pose a potential privacy risk. Accordingly, although the use of online content is supported, the suggested *default* approach is to embed all content into the CBA ItemBuilder project files. Embedded content can be split into several files, and the importing of single files or folders into CBA ItemBuilder *Project Files* is possible. Embedded content becomes part of the CBA ItemBuilder project files (i.e., it can be found and even modified in the folder *external-resources* of the zip archives that are edited and previewed with the CBA ItemBuilder by item authors and that are used with

the deployment software for data collections). `ExternalPageFrames` require a file as an entry point (e.g., a file `index.html`). The content in the folder `external-resources` of the zip archives (i.e., the CBA ItemBuilder *Project Files*) can be updated and used as long as the entry points remain available.

- Save and Restore Content State:** `ExternalPageFrames` / `iframes` are included in the CBA ItemBuilder on pages, and if CBA ItemBuilder tasks contain multiple tasks, the test-takers typically can navigate between pages. If test-takers navigate to different pages, it is possible that the `ExternalPageFrames` / `iframes` is unloaded and reloaded when the test-taker navigates back to the page that contains the `ExternalPageFrames` / `iframes`. By design, the data or the editing state of `ExternalPageFrames` / `iframes` will be lost if it is not saved. In order to implement persistence of `ExternalPageFrame`'s / `iframes`'s state across page changes, the CBA ItemBuilder provides a simple mechanism: Before an `ExternalPageFrame` / `iframe` is unloaded, its state is queried with a JavaScript function call `setState()`. The data required for persistence is expected to be passed as a JSON serialized object so that the delivery software or the CBA ItemBuilder Runtime can take care of caching. In the opposite direction, the JavaScript function call `getState()` can be used to retrieve the last state. Therefore, when the created component is rendered, it should use `getState()` to query the JSON serialized object of the last state so that it can restore its state.
- Add Trace Messages (Provide Log-Data):** Everything that should be stored in the long term, e.g., answers and results for assessment components that are included as `ExternalPageFrame` / `frame`, can be passed via a `postMessage`-interface. The data is expected as JSON objects. The information provided using this API is stored as part of the log data (together with a time stamp and a person or session identifier). The type must be recognizable from the object itself if differentiation is necessary during data post-processing. This approach is mainly intended for the storage of log data, whereby the final result data can also result from all log data or the last log events of a type.
- Provide Result Variables:** Log data is typically not available for data preparation until the assessment is complete. Suppose answers or results from the `ExternalPageFrames` are already needed during the assessment or needed in the scoring (i.e., creating result variables) within the CBA ItemBuilder. In that case, they can be passed as variable-value pairs whenever the variables are changed within the `ExternalPageFrame`. The CBA ItemBuilder API also provides a mechanism for `ExternalPageFrames` to announce which variable values they will provide. This allows item authors to plan with these variables while creating the items and defining the scoring. At runtime, the `ExternalPageFrame` then passes the individual variable-value pairs via `postMessage`, making them available afterward in the logic layer of the CBA ItemBuilder (i.e., the finite-state machine) and for the final computation of item scores.
- Answer Calls from the Finite-State Machine:** The aforementioned logic layer of the CBA ItemBuilder (the finite-state machine) can also be used to pass information

into the `ExternalPageFrames / iframe`. The JavaScript code can respond to `postMessages` for this purpose, and change the display or functioning of the content provided via `ExternalPageFrames / iframes`.

- **Trigger Finite-State Machine Events:** Finally, the other direction of the message chain is also possible, i.e. actions in the logic layer of the CBA ItemBuilder can be triggered by making calls from the `ExternalPageFrames / iframes` that trigger events within the finite-state machine. For this, the item authors must define the events in the finite-state machine by assigning a unique name and an event with this name can then be triggered in the programming of the external content. This procedure is necessary, for example, if a navigation to a next item (i.e., in the terminology of this CBA ItemBuilder to a next task) is to be triggered from the external content.

Template A template file for programmers is provided and the following briefly describes what to look for when creating external content for use in `ExternalPageFrames` of the CBA ItemBuilder. The template is shown in Figure 4.73 and can be downloaded as a CBA ItemBuilder project file using the link in the figure caption. To view the template locally in the CBA ItemBuilder preview and to view the HTML / JavaScript source code of the template with the CBA ItemBuilder, the CBA ItemBuilder must be installed as described here (see section 1.1). After that the project file can be opened and the preview can be started as described in section 1.4 the *Preview* can be started. The prepared HTML / JavaScript files can be seen in the *Embedded HTML Explorer* described in section 3.14.2.

When creating a CBA ItemBuilder *Project Files*, an HTML file `example.html` is created automatically, this file contains JavaScript examples and can be edited and exported in the *Embedded HTML Explorer* (see section 3.14).

Examples: Additional examples how content can be used in `ExternalPageFrames` is provided in section 6.6. `ExternalPageFrames` can be used to collect responses, for instance, to measure how long a sound is played (see Figure 6.25). Moreover, `ExternalPageFrames` can be used to use online or offline voice recognition (see section 6.6.3), `ExternalPageFrames` can also be used to embedded content in other formats, such as *H5P* (see section 6.6.4), to collect mathematical input (see section 6.6.5) or even to embedd item content in the standardized [IMS Question & Test Interoperability \(QTI\) format](#) using, for instance, the [QTI.js](#) library (see section 6.6.6 for an example). Finally, offline questionnaires implemented, for instance, with the [SurveyJS](#) library can be embedded (see section 6.6.7).

ExternalPageFrame - Template

JavaScript API Example

Update state

--

--

Post trace
Post variable->3
Post get variable
Post FSM event

Check:

1) The button *Update state* increases a click counter by one at a time. After unloading the ExternalPageFrame by clicking *Go to "page2"* and returning to this page, the click counter must be restored.

2) The current value of VarA is:

0

Change the value and see how the button *Post get variable* can be used to read and the button *Post variable -> 3* can be used to set the variable. Click *Strg/Ctrl + M* to see the variable in the *State Machine Debugger*.

3) Click *Post FSM event* to trigger an event that opens a dialog.

4) Click *Post trace* and open the *Trace Debugger* by pressing *Strg/Ctrl + Y*.

Note: The keyboard shortcuts for the State Machine Debugger (*Ctrl/Cmd+M*) and the Trace Debugger (*Ctrl/Cmd+Y*) will only work if previously clicked in the white area outside the ExternalPageFrame.

Go to "page2"

Note: The red background color is defined inside the CBA ItemBuilder for the ExternalPageFrame. Width: 800 x Height: 600

File: ExternalPageFrameTemplate.zip

FIGURE 4.73: Template for content developed for ExternalPageFrames ([html](#)|[ib](#)).

4.6.2 Communication between with ExternalPageFrames (JavaScript) and Finite-State Machine



The simple JavaScript API for embedded content as ExternalPageFrame is described in depth in this section.

4.6.3 Provide Information to the Finite-State Machine from JavaScript

The communication between the embedded JavaScript and the CBA ItemBuilder runtime (and thus the *Finite-State Machine* as defined by item authors in the CBA ItemBuilder) is done via *postMessages*.

```
window.parent.postMessage(JSON.stringify(message), '*');
```

Depending on the configuration of the JSON string passed as *message*, *postMessages* can be sent for various purposes. The various use cases are illustrated in the item shown in Figure 4.74.

ExternalPageFrame API Illustration

Click here and press Ctrl + Y (Strg + Y) to open the Trace Debug Window or Ctrl + M (Strg + M) to open the State Machine Debug Window.

This ExternalPageFrame contains a button that is linked to a JavaScript function. The function sends a JavaScript-Injected log event. Set the focus to the item by clicking in the upper-right corner and press Strg/Ctrl+Y to see the log message in the Trace Debug Window.

Trace Log-Event

Set 'V_OutputInteger'

Set 'V_OutputString'

Set 'V_OutputNumber'

Set 'V_OutputBoolean'

V_OutputInteger = 0

V_OutputString = asdf234c34

V_OutputNumber = 7.11

V_OutputBoolean = true

Get 'V_OutputInteger'

Get 'V_OutputString'

Get 'V_OutputNumber'

Get 'V_OutputBoolean'

Trigger FSM-Event

This ExternalPageFrame contains buttons that trigger JavaScript functions to generate random values for the different variables and to transmit the values of the variables to the Finite-State Machine layer. The value of the variables is then shown using VariableValueDisplays. The value of the FSM variables can also be checked using the State Machine Debug Viewer.

This ExternalPageFrame contains buttons that trigger JavaScript functions to request the values for the different variables from the Finite-State Machine. After asking for the values, the transmission is performed asynchronously using post messages. The retrieved values are shown with a simple JavaScript alert.

This ExternalPageFrame contains a button that triggers a JavaScript function to trigger a Finite-State Machine event. The event is used in the Finite-State Machine to open a dialog.

File: ExternalPageFrameAPIIllustration.zip

FIGURE 4.74: Item Illustrating the API for ExternalPageFrames ([html5ib](http://html5ib.com)).

Create Custom Log-Entries: Log events (i.e., event-based data generated during test-taking) are stored by the delivery software and are available after the assessment. Only for components from the CBA ItemBuilder model are log events automatically created when the components are added to the design of pages (and thus the design of assessment components) via the *Page Editor*. For content that is

included as an `ExternalPageFrame`, care must be taken in the programming of the HTML5/JavaScript content to ensure that all relevant information is logged (see section 2.8.1 for completeness of log data).

If the payload transmitted as `postMessage` contains a `traceMessage` attribute, a log entry is created by the CBA ItemBuilder runtime:

```
var message = "Random message, e.g., " + new Date().toLocaleString();
var type = "CustomTraceType";

var pass_data = {
    indexPath: indexPath,
    userDefIdPath: userDefIdPath,
    traceMessage: message,
    traceType: type,
    traceCount : traceCount++
};
window.parent.postMessage(JSON.stringify(pass_data), '*');
```



Data passed from an `ExternalPageFrame` can be found in the CBA ItemBuilder log data with events of type `JavaScriptInjected`. They can contain log events from the embedded content. However, this interface can also be used to store result data from `ExternalPageFrames`.

The log entries can use their event types if the `traceType` attribute is also passed. The transmission of log data from `ExternalPageFrames` can be used to transmit arbitrary string serializable data and have them stored by the delivery environment.

Set FSM Variables: From a content included as `ExternalPageFrame`, also values of *FSM variables* can be set. This option allows, for instance, returning values from embedded content, which can also be used in scoring. In addition, arbitrary calculations at runtime of items are also possible via JavaScript and possible embedded libraries. To set a variable, the *payload* `microfinVariable` must be submitted as a JSON structure with the values `variableName` (name of the variable that will be set) and `newValue` (new value for the named variable). The type of the variable must be taken into account.

```
var varname = "V_OutputInteger";
var value = Math.floor(Math.random() * 100);

var pass_data = {
```



```

    indexPath: indexPath,
    userDefIdPath: userDefIdPath,
    microfinVariable: { variableName: varname, newValue: value },
    traceCount : traceCount++
};

window.parent.postMessage(JSON.stringify(pass_data), '*');

```

Request Value of FSM Variables: Besides setting *FSM variables*, the value of *FSM variables* can also be retrieved. To do this, the JavaScript sends a *postMessage* with the *payload* `getVariable`, in which the name of the desired variable and an (arbitrary) `callId` must be contained as `variableName`:

```

var callId = "ID" + new Date().toLocaleString();

var pass_data = {
    indexPath: indexPath,
    userDefIdPath: userDefIdPath,
    getVariable: { variableName: variableName, callId: callId},
    traceCount : traceCount++
};

window.parent.postMessage(JSON.stringify(pass_data), '*');

```

The CBA ItemBuilder runtime responds to this *postMessage* and transmits back the value of the variable, which can then be captured and further processed with an `EventListener`. In the example in Figure 4.74, the transmitted return JSON is displayed as `alert`.

Trigger FSM Event: *FSM events* defined in the *Finite-State Machine* syntax in the `Events:` section can be triggered from `ExternalPageFrames` when a *payload* with the `microfinEvent` attribute is submitted via *postMessage*:

```

var eventName = "EV_JSTriggeredEvent";
var pass_data = {
    indexPath: indexPath,
    userDefIdPath: userDefIdPath,
    microfinEvent: eventName,
    traceCount : traceCount++
};

window.parent.postMessage(JSON.stringify(pass_data), '*');

```

Using this part of the interface, transitions can be triggered in the *Finite-State Machine*, and operators can be used. If, for example, HTML5/JavaScript content included via an `ExternalPageFrame` should be able to end a task, then the `next_task()` operator (see section 4.4.6) can be integrated into the *Finite-State Machine* and placed into a *Rule*, which is triggered by an *FSM Event*. The *FSM Event* is fired when the JavaScript content sends a *postMessage* with the *payload* `microfinEvent` and the corresponding name of the *FSM Event*.

4.6.4 Call JavaScript-Function of `ExternalPageFrames` from Finite-State Machine(s)

Once an HTML5/JavaScript resource included as `ExternalPageFrame` is loaded, functions can be called using the *FSM* operator `callExternalPageFrame`:

```
callExternalPageFrame(UserDefinedId, ...)
```

Several `ExternalPageFrames` can be integrated in one page or in one *Task* of a CBA ItemBuilder *Project Files*. The first parameter must be the `UserDefinedId` of the particular `ExternalPageFrame` component that should process the JavaScript call. Additional optional parameters are possible. As illustrated in Example 4.75, the optional parameters can be used to pass static strings or values of *FSM* variables. The values of the arguments are then available in the called JavaScript function.

ExternalPageFrame Call JavaScript Example

| | |
|--|--|
| <div>Button 1</div> | <p>This button triggers an FSM-event ("EV_Button1") that uses the call <code>callExternalPageFrame("EFS1","StaticMessage")</code>-operator. The operator sends a <code>PostMessage</code> to the embedded HTML-File "SimpleExternalPageFrame1.html". In this HTML file, an <code>EventListener</code> waits for incoming <code>PostMessages</code> and shows an alert-window to print the received message JSON.</p> |
| <div>Button 2</div> <div>V_Input = <input type="text" value="0"/></div> | <p>This button triggers an FSM-event ("EV_Button2") that uses the call <code>callExternalPageFrame("EFS2",V_Input)</code>-operator. The operator sends a <code>PostMessage</code> to the embedded HTML-File "SimpleExternalPageFrame2.html". In this HTML file, an <code>EventListener</code> waits for incoming <code>PostMessages</code> and shows an alert-window to print the received message JSON.</p> |
| <div>Button 3</div> <div>V_Output = <input type="text" value="0"/></div> | <p>This button triggers an FSM-event ("EV_Button3") that uses the call <code>callExternalPageFrame("EFS3","")</code>-operator. The operator sends a <code>PostMessage</code> to the embedded HTML-File "SimpleExternalPageFrame3.html". In this HTML file, an <code>EventListener</code> waits for incoming <code>PostMessages</code>, generates a random number between 0 and 99 and sends a <code>PostMessage</code> back to assign the value to the <i>FSM</i> variable "V_Output".</p> |

File: ExternalPageFrameCallJSExample.zip

FIGURE 4.75: Example Item Illustrating JavaScript calls from FSM ([html](#)|[ib](#)).

As can be seen in the item in Figure 4.75, calls to JavaScript functions from the *Finite-State Machine* can be used for quite different purposes.

4.6.5 Provide ExternalPageFrames-State for Persistence (getState/setState)

HTML/JavaScript extensions inserted as `ExternalPageFrame` on pages inside CBA ItemBuilder projects lose their content when the page that contains the `ExternalPageFrame` is unloaded. Unloading occurs on page changes either within a task or when exiting a task. To allow `ExternalPageFrames` to save and restore their content, the API includes the `getState/setState`-functionality.

The following behavior is expected: When the `getState()` method is called, then a string (e.g., a JSON serialized object) can be passed from the `ExternalPageFrame` to the CBA ItemBuilder runtime. The call of `getState()` is done automatically before the page that contains the `ExternalPageFrame` is unloaded by the CBA ItemBuilder Runtime.



Content from embedded JavaScript/HTML5 will be lost if in a *Task* the *Page* is left, on which the `ExternalPageFrame` component is embedded. If the content is needed again when the page is revisited, it must be passed to the CBA ItemBuilder runtime via `getState()`. Furthermore, the embedded content needs to handle the call `setState()` and restore its content when requested.

```
function getState() {  
    try {  
        // Specific JavaScript Code Required  
        var json = JSON.stringify(/* your data */);  
        return json;  
    }  
    catch (e) {  
        console.debug(e);  
    }  
}
```

When navigating to a page that contains an `ExternalPageFrame` that has already been displayed, the CBA ItemBuilder Runtime calls the `setState()` method and passes the state serialized as a string that was passed the last time `getState()` was called.

```
function setState(stateString) {  
    try {  
        // Specific JavaScript Code Required  
        // ...  
    }  
    catch (e) {
```

```
        console.debug(e);  
    }  
}
```

The minimal example shown in Figure 4.76 illustrates the expected behavior.

ExternalPageFrame GetState / SetState Example

This is "page1".

myCheckBox ☐

myTextArea

The method `getState()` is called automatically. The method `setState()` is called before navigating to a different page (click button "Link to page2").

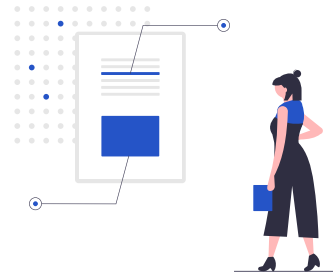
Link to "page2"

File: ExternalPageFrameGetStateSetState.zip

FIGURE 4.76: Example Item Illustrating JavaScript calls for Persistence ([html](#)|[ib](#)).

5

Scoring of Tasks



This chapter describes how scoring can be defined in CBA ItemBuilder *Projects Files*. Scoring is always defined on the task level (see section 3.6 for details on defining tasks). Accordingly, before scoring can be implemented in *newly created* CBA ItemBuilder projects files, a *Task* must be configured (see section 3.6 for details). A second prerequisite for defining the scoring of *Tasks* is to define names for all required components. That means that *User Defined Ids* (see subsection 3.7.4 for details) is required for the components used to gather responses. Human readable *User Defined Ids* are suggested to remember the meaning of particular identifiers when using them, for instance, in the syntax definition of scoring rules. Since this definition of scoring rules is based on the *User Defined Ids*, systematically defined and easily readable IDs simplify the creation and validation of scoring rules for item authors.

The definition of explicit scoring rules is not mandatory for the use of CBA ItemBuilder items, but it provides the most flexible way to combine evidence into scoring. Alternatively, 1) the so-called *component state*, that is, the values selected or entered for all input elements, can be automatically stored by the test assembly and deployment software (see chapter 7). Moreover, 2) the components can be linked to *FSM variables* (applies to version 10.0) and the last value of *FSM variables* when a task is exited can be used. Finally, 3), log data can also contain all changes of component values and it is often possible, to infer about the final response from the collected log events (see section 1.6, and Kroehne and Goldhammer (2018) for *response-completeness* of log data).

Motivation: Explicit automatic scoring of items can be necessary at runtime when scoring results that incorporate logical rules are required either for adaptive test assemblies (branched testing, multi-stage testing, or adaptive testing, see section 2.7.4), for the different feedback purposes (see section 2.9), or to monitor the test-

taking processes. Scoring of CBA ItemBuilder *Tasks* is evaluated at task switches (i.e., when tasks are changed from one task to another). Task switches can either be triggered from within the *Task* (using *Runtime Commands*, see section 3.12, or from outside (by the deployment software, for instance, because of a global timeout, see section 7.2.8). Having the scoring definition implemented within the assessment components created with the CBA ItemBuilder can also simplify data post-processing workflows (see section 8.6) and sharing of items (for instance, as *Open Educational Resources*, see section 8.7.4). Hence, automatic scoring provides essential advantages over above mentioned alternatives: It standardizes the scoring procedures and gives immediate access to the scored results.

Implementing the scoring rules within the CBA ItemBuilder project files comes with a second advantage. The scoring definition becomes independent from the deployment software (see chapter 7) and the approach used for data post-processing (see section 8.4.2). Scoring embedded in CBA ItemBuilder projects can be tested already during item development and will be available after distributing items (as CBA ItemBuilder projects). An essential tool for checking the scoring in CBA ItemBuilder items is the so-called *Scoring Debugger*, as described already in section 1.5. The *Scoring Debugger* can be used to inspect the scoring live during *Preview*.

5.1 Terminology, Concepts and User Interface



The core component for the definition of the automatic scoring is the design of syntax conditions, which can be evaluated based on inputs (i.e., *Component State* of elements) and operators (i.e., incorporating the states in the internal finite-state machine(s) and the visited pages).

User Defined Id's: The link between components and the syntax is provided by the User Defined Id's. Using the main menu Project > Edit all user defined IDs all named components of a CBA ItemBuilder *Project File* can be displayed. As shown in Figure 5.1, for a simple multiple-choice item, various components of type *Checkbox* can be defined, all of them with a unique *UserDefinedId*.

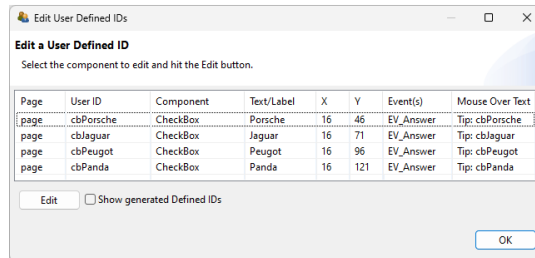


FIGURE 5.1: UserDefinedIds defined for example item shown in Figure 5.2.

Figure 5.2 shows a simple item to illustrate the scoring of a multiple-choice item. The item was created by adding a `HTMLTextField` (see subsection 3.8.2 for the instruction and the four `CheckBoxes` (see subsection 3.9.3). A simple scoring of this item might distinguish the conditions *Correct* (Jaguar and Panda) and *Wrong* (Jaguar and Panda are not selected or additional wrong options are selected). If a test-taker never selected any `CheckBoxe` at all, this might be called a *Missing* response (see section 5.3.11 for details).

The CBA ItemBuilder allows to define scoring-conditions using these `UserDefinedIds`. The conditions are labeled of *Hits* (i.e., *Hit*-conditions) and *Misses* (i.e., *Miss*-conditions) combining `UserDefinedIds` of components, and additional functional operators (see appendix B.2 for all operators), if necessary, combined with logical operators. Each condition represents a nominal conditions that is to be differentiated when computing the value of a scoring variable (called *Class*). The item contains one *Task* (defined in the *Task-Editor*, see subsection 3.6. The *Task-Editor* is also used to define the scoring (see Figure 5.3).

Class: Scoring is defined using nominal *Hit*-conditions. Each conditions corresponds to a potential value of a variable. To specify the relationship of values to variables, *Hit*-conditions (i.e., values) are assigned to *Classes* (i.e., *Classes* are equivalent to *Variables* in the final data set), as shown in Figure 5.4. The variable `score` can have the three potential values `Correct`, `Wrong` and `Omitted`.

Name: Each hit- and miss-condition requires a unique name, that is defined in the *Task-Editor* (shown in Figure 5.3). This name represents the nominal value of the variable, if the corresponding condition is met (i.e., if the hit is *active*).

Condition Syntax: Each *Hit*-condition is defined by providing a scoring syntax. The example item shown in Figure 5.2 contains three different *Hit*, and the syntax for the conditions are shown in the editor for *Conditions* in Figure 5.5. The first condition for the hit `Correct` combines the `User Defined Id`'s of the four `CheckBoxes` with logical operators (using the CBA ItemBuilder specific bracketing of expressions, see section 4.1.3).

However, the syntax for scoring rules (see section 5.3) also allows using scoring operators, for instance, to incorporate information from the dynamic part of CBA

Scoring Introduction - Example 1

Instruction
Select all animal(s).

☐ Porsche ID: cbPorsche
☐ Jaguar ID: cbJaguar
☐ Peugeot ID: cbPeugot
☐ Panda ID: cbPanda

(Click here to set the input focus and press Ctrl + S to open the "Scoring Debug Window".)

This example item illustrates the use of hit-conditions for scoring CBA ItemBuilder tasks. Three hits are defined:

- Correct: Jaguar and Panda are selected (and no other options).
- Wrong: Item answered (i.e., any checkbox selected), but wrong.
- Missing: No checkbox selected at any time.

The first hit (Correct) specifies the correct response:

Correct `((cbJaguar and cbPanda) and not cbPorsche) and not cbPeugot)` (1)

If this condition is not met, the second hit (Wrong) checks if the item is in the state "Answered" (i.e., if at any time a checkbox was selected and the assigned event `EV_Answer` was triggered):

Wrong `is_last_state(Answered)` (2)

Finally, the last condition is (trivial) defined as always fulfilled. This makes sense because the hits are checked in the defined sequence. If the condition "Correct" and "Wrong" are not met, the item is scored "Missing".

Missing `true` (3)

This example is based on the option "Use first active hit per class (applies to all tasks)".

File: ScoringIntroductionExample1.zip

Sequence Matters

FIGURE 5.2: Example for scoring a multiple choice item (html|ib).

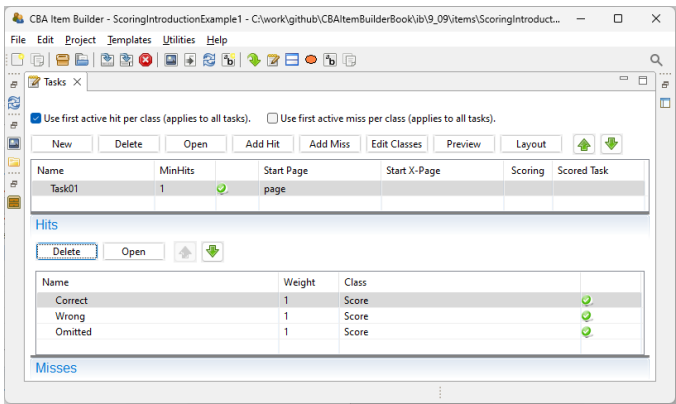


FIGURE 5.3: Task-Editor with one Task and three Hit-conditions for the item shown in Figure 5.2.

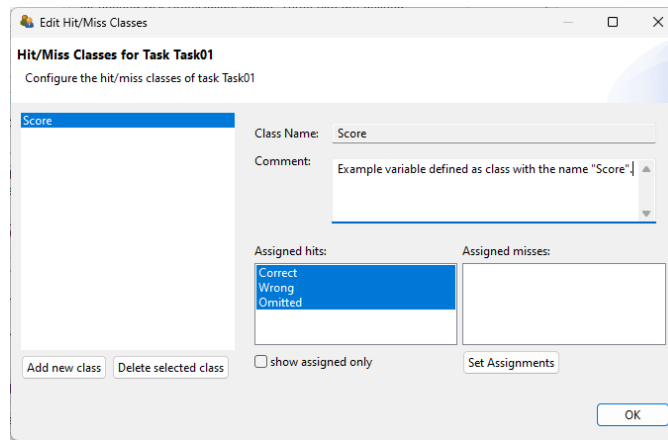


FIGURE 5.4: *Class Definition Dialog* for the item shown in Figure 5.2.

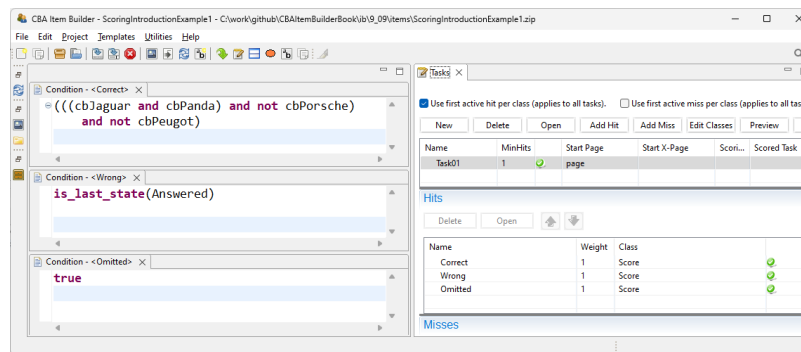


FIGURE 5.5: *Condition Syntax* for the *Hit*-condition *Correct* of the item shown in Figure 5.2.

ItemBuilder tasks. Operators are illustrated in Figure 5.5 in the syntax for the hit-condition for *Wrong*, which evaluates to true if the item's current state is *Answered*. Note that the state *Answered* is implemented using a simple finite-state machine (see section 4.4 for details).



Scoring conditions can either be defined mutually exclusive (default, see section 5.3.3), or the order of conditions is incorporated (as used in the example in Figure 5.2). This suggested option is activated by selecting *Use first active hit per class (applies to all tasks)*. (see right part of Figure 5.5).

Scoring Debug Window: The *Scoring Debug Window* (already introduced in section 1.5) can be used to explore the scoring of the example item shown in Figure 5.2, as

shown in Figure 5.6. The *Scoring Debug Window* can be requested during item development in the *Preview* and is also available in the examples embedded in the [online version](#) of this book.

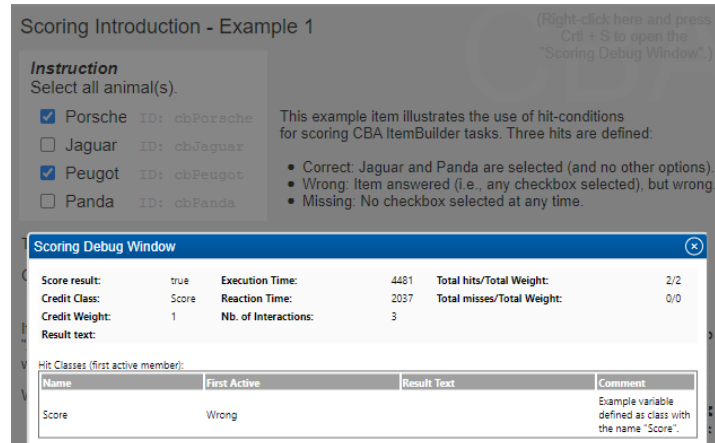


FIGURE 5.6: Screenshot of the *Scoring Debug Window* in a preview of the item shown in Figure 5.2.

Multiple Classes: If components are to be used only in a particular way to form outcome variables, defining scoring constraints may be more onerous than strictly necessary (see section 5.2 for an alternative). The full potential of CBA ItemBuilder scoring unfolds in the use cases when different summaries of answers to variables are to be used. This is illustrated in Figure 5.7 for a simple Likert-style item. Assume that two variables should be created: One variable containing the response (Class: *Response*) and one indicating agreement or disagreement [Class: *Style*, as used, for instance, in models to investigate response style; cf. Böckenholt and Meiser (2017) and others].

By introducing the layer of hit definitions (described in detail in section 5.3), the CBA ItemBuilder allows the creation of flexible scoring for responses that can combine multiple components and can result in multiple variables (i.e., classes). Use cases not only include questionnaires (as the example in Figure 5.7) but can also be found for cognitive assessment, e.g., when both the raw response and the automatically scored response (correct vs. incorrect) are to be stored or when dichotomous and polytomous scoring is to be considered. Use cases for explicit scoring with multiple classes also arise if, for example, time measures are included in the scoring of responses.

Result-Text: *Hit-* and *miss-*conditions can be used to define evidence in terms of categorical values, which can then be assigned to classes to be used as outcome variables. However, also entered text and numbers are required as result variables. To copy text responses to result variables, the CBA ItemBuilder provides the `re-`

Scoring Introduction - Example 2

Instruction: How do agree to a statement?

☐ Strongly Disagree ID: `rbStronglyDisagree`
☐ Disagree ID: `rbDisagree`
☐ Neutral ID: `rbNeutral`
☐ Agree ID: `rbAgree`
☐ Strongly Agree ID: `rbStronglyAgree`

(Click here to set the input focus and press Ctrl + S to open the "Scoring Debug Window".)

This example item illustrates how hit conditions can be used to create two classes (i.e., variables):

- Response: Selected answer
- Style: Agree vs. Disagree

| Hit (Name) | Class | Syntax (Condition) |
|--------------------------|----------|---|
| ResponseStronglyAgree | Response | <code>rbStronglyAgree</code> |
| ResponseAgree | Response | <code>rbAgree</code> |
| ResponseNeutral | Response | <code>rbNeutral</code> |
| ResponseDisagree | Response | <code>rbDisagree</code> |
| ResponseStronglyDisagree | Response | <code>rbStronglyDisagree</code> |
| StyleAgreement | Style | <code>(rbAgree or rbStronglyAgree)</code> |
| StyleNA | Style | <code>rbNeutral</code> |
| StyleDisagreement | Style | <code>(rbDisagree or rbStronglyDisagree)</code> |
| ResponseMissing | Response | <code>not (((rbStronglyDisagree or rbDisagree) or rbNeutral) or rbAgree) or rbStronglyAgree)</code> |
| StyleMissing | Style | |

File: ScoringIntroductionExample2.zip

Mutually exclusive hits per class
(order of hit conditions is irrelevant)

FIGURE 5.7: Example for scoring a Likert-style item into two variables ([html](#)|[ib](#)).

`sult_text()`-operator. The underlying idea is that each class (i.e., variable) can provide a *Result-Text* in addition to the name of the active hit. The condition defines which particular value is used as *Result-Text*. The (first) active *hit-/miss-condition* of a class defines which text is copied into the *Result-Text*.

The item shown in Figure 5.8 illustrates the use of the *Result-Text*. The first class (`Var1`) is used for question 1: Class `Var1` has only one hit-condition with the syntax `result_text(input1)`. This condition is always true, and whatever is entered in the `SingleLineInputField` with the *User-Defined Id* `input1` is copied to the *Result-Text* for `Var1`. For question 2, the class `Var2` is used with two hit conditions. When a text is entered into the `InputFiled` with the *User-Defined Id* `input2` (i.e., the text is not empty checked with the condition `matches(input2,"")`), the value is copied to the *Result-Text* using the operator `result_text(input2)` in the condition `Q2_Text`. When nothing is entered, the *Result-Text* is filled with the string `Missing` (see hit-condition `Q2_Missing`). The class `Var3` is used for question 3. The class contains either the selected option (A or B) in the *Result-Text* (see hit-conditions `Q3_A` and `Q3_B`). If neither A, B or Other is selected, the string `Missing` is copied to the * *Result-Text* * (see hit-condition `Q3_Missing`). Two hit-conditions are defined that deal with conditions that Other is selected. If no text is

entered into the `SingleLineInputField` with the *User-Defined Id* `input3`, the text `Other: Not Specified` is copied to the *Result-Text* (see hit-condition `Q3_OtherNotSpecified`). If a text is entered, the *Result-Text* is filled with the string `other:` followed by the provided text. This is achieved by using an argument list for the `result_text()`-operator (see 4.1.5). Note that `Var3` will not contain the text entered into `input3` if `A` or `B` is selected. This issue is addressed by defining `Var4` that contains the text entered in `input3`, even if `other` is not selected.

ResultText-Operator Example

Question 1:
(only numbers)

ID: input1

Question 2:
(multiline text)

ID: input2

Question 3:
(optional text)

☐ A

ID: A

☐ B

ID: B

ID: Other

☐ Other

ID: input3

(Click here to set the input focus and press Ctrl + S to open the "Scoring Debug Window".)

This example illustrates the use of the `result_text()`-operator for numeric inputs, (multiline) text responses and optional text responses.

| Name | Class | Syntax |
|-----------------------|-------|--|
| Q1_Number | Var1 | <code>result_text(input1)</code> |
| Q2_Text | Var2 | <code>(not matches(input2,"") and result_text(input2))</code> |
| Q2_Missing | Var2 | <code>(matches(input2,"") and result_text("Missing"))</code> |
| Q3_A | Var3 | <code>(A and result_text("A"))</code> |
| Q3_B | Var3 | <code>(B and result_text("B"))</code> |
| Q3_OtherNotSpecified | Var3 | <code>((Other and matches(input3,"")) and result_text("Other: Not Specified"))</code> |
| Q3_Other | Var3 | <code>((Other and not matches(input3,"")) and result_text("Other: %1\$s",input3))</code> |
| Q3_Missing | Var3 | <code>((not A and not B) and not Other) and result_text("Missing")</code> |
| Q3_EmptyText | Var4 | <code>matches(input3,"")</code> |
| Q3_Other_not_selected | Var4 | <code>((not Other and result_text(input3)) and not matches(input3,""))</code> |
| Q3_Other_selected | Var4 | <code>((Other and result_text(input3)) and not matches(input3,""))</code> |

File: ResultTextOperatorExample.zip

FIGURE 5.8: Item illustrating scoring with `result_text()`-operator ([html](#)|[ib](#)).

5.2 Scoring using FSM Variables

Using items provided by [Toplak et al. \(2014\)](#) the item in Figure 5.9 illustrates scoring using variables. In this example, the finite-state machine updates variable values, designed to allow immediate feedback (correct response, intuitive incorrect re-

sponses, and any other wrong response) and to compute the total score for all seven items of the *Cognitive Reflection Test*.

Cognitive Reflection Test

- (1) A bat and a ball cost \$1.10 in total. The bat costs a dollar more than the ball.
How much does the ball cost? cents
- (2) If it takes 5 machines 5 minutes to make 5 widgets, how long would it take 100 machines to make 100 widgets? minutes
- (3) In a lake, there is a patch of lily pads. Every day, the patch doubles in size. If it takes 48 days for the patch to cover the entire lake, how long would it take for the patch to cover half of the lake? days
- (4) If John can drink one barrel of water in 6 days, and Mary can drink one barrel of water in 12 days, how long would it take them to drink one barrel of water together? days
- (5) Jerry received both the 15th highest and the 15th lowest mark in the class.
How many students are in the class? students
- (6) A man buys a pig for \$60, sells it for \$70, buys it back for \$80, and sells it finally for \$90.
How much has he made? dollars
- (7) Simon decided to invest \$8,000 in the stock market one day early in 2008. Six months after he invested, on July 17, the stocks he had purchased were down 50%. Fortunately for Simon, from July 17 to October 17, the stocks he had purchased went up 75%. At this point, Simon has:
☐ a) broken even in the stock market ☐ b) is ahead of where he began ☐ c) has lost money

[Next](#)

Source: Toplak, M. E., West, R. F., & Stanovich, K. E. (2014). Assessing miserly information processing: An expansion of the Cognitive Reflection Test. *Thinking & Reasoning*, 20(2), 147–168. <https://doi.org/10.1080/13546783.2013.844729>

FIGURE 5.9: Example item illustrating scoring with variables ([html](#)|[ib](#)).



CBA ItemBuilder Version 10.0 will support a simple scoring definition by providing *default* scoring for components (e.g., Checkboxes) and groups (e.g., RadioButtonGroup and FrameSelectGroups) using *FSM Variables* (instead of so-called *Hit-/Mis-*conditions).

5.3 Definition of Explicit Scoring Rules



5.3.1 UserDefinedId's as String Literals

An important part of possible scoring rules are input elements, i.e. components for the design of items, which have a value (i.e., are either selected or un-selected). To refer to the value of a component in a *Hit*- or *Miss*-condition, it is sufficient to include the `UserDefinedId` of the respective component into the condition-syntax. For instance, for a checkbox with the `UserDefinedId`: `myCheckbox`, the string literal `myCheckbox` is interpreted as `TRUE` if the checkbox is selected, when the syntax is evaluated. If the checkbox is not selected, the string literal `myCheckbox` is interpreted with the value `FALSE`.



The checked/unchecked state of `CheckBox` - components, the selected/unselected state of `RadioButton` - components, the toggle state of Buttons in toggling mode and the selected/unselected-state of `ComboBoxItem` in a `ComboBox` can be used to define *Hit*- or *Mis*- conditions by simply referring to the `UserDefinedId` of the component.

5.3.2 Syntax for Scoring Rules

The item scoring mechanism implemented in CBA ItemBuilder goes beyond simple mapping of *Scoring Conditions* (i.e., hit- and miss conditions) to component states. This is enabled by providing the possibility to formulate conditions as arbitrary combinations of statements using a so-called *Domain Specific Language* (DSL, i.e., by using a specific syntax).

To combine `UserDefinedIds` of the components to logical expressions, the following *logical operators* can be used:

- A and B: true if A and B evaluate to true.
- A or B: true if A or B evaluate to true.
- not A: true if A is not true.

Flexible combinations of conditions are possible with the basic operators and, or and not. Use the *Scoring Debug Window* (Ctrl / Strg + S, see section 1.5) to explore the hit conditions in the item shown in Figure 5.10.

Hit-Definition AND & OR Example

(Click here to set the input focus and press Ctrl + S to open the "Scoring Debug Window".)

☐ A ID: A
☐ B ID: B
☐ C ID: C
☐ D ID: D

Select some of the Checkboxes and check the scoring using the "Scoring Debug Window".

Notice warning (red text) when multiple hits are active at a time.

File: HitdefinitionAndANDOr.zip

Hit Definitions:

- A_and_B: (A and B)
- A_or_B: (A or B)
- notB: not B
- all: (((A and B) and C) and D)

Scoring Debug Window

Score result: true Execution Time: 29279 Total hits/Total Weight: 1/1
 Credit Class: Reaction Time: 5511 Total misses/Total Weight: 0/0
 Credit Weight: 0 Nb. of Interactions: 6
 Result text:

Hits:

| # | Name | Weight | Class | Result text |
|---|------|--------|-------|-------------|
| | notB | 1 | | |

Hits:

| # | Name | Weight | Class | Result text |
|---|---------|--------|-------|-------------|
| ! | A_and_B | 1 | | |
| ! | A_or_B | 1 | | |

FIGURE 5.10: Hit definition with logical expressions (and, or, not; [html|ib](#)).

Notice the specific bracketing in the last hit condition shown in Figure 5.10: (((A and B) and C) and D). This condition illustrates that combining multiple Boolean expressions requires to include brackets so that the statement can be decomposed into pairs: A and B, (A and B) and C, and finally ((A and B) and C) and D.



It is important to note that the use of brackets is required to formulate statements with more than two conditions (see section 4.1.3).

For a number of scoring tasks, simply checking the Boolean value of components

is not sufficient. Therefore, the CBA ItemBuilder provides functions in the scoring syntax (so-called *operators*), which can be used within the scoring syntax to take into account properties of the current *task* for the formulation of *hit* and *miss* conditions.

5.3.3 Sequential Evaluation of Scoring Rules

By default (i.e., when not configured differently), *Hit*- and *Miss*-conditions are evaluated independently. If variables are created, i.e., hits are assigned to classes, a central condition must be met: *At any time, precisely one hit must be active for each class*. This condition follows directly from using hits as (categorical) values for variables. Consequently, hit conditions within a class must always be formulated in such a way that they are *mutually exclusive*. To support checking this condition, the CBA ItemBuilder's *Preview* of tasks provides the *Scoring Debug Window*, which contains a red exclamation mark once multiple hits are active for a class (see Figure 5.10).

However, a powerful alternative is to activate the sequential evaluation of scoring conditions in the *Task-Editor* by selecting the checkbox *Use first active hit/miss per class* (applies to all tasks). If this option is activated, the evaluation is performed sequentially, starting with the first hit condition of a class. Only if the first hit is not true, the second hit is evaluated. Accordingly, a last hit (when no other conditions evaluate to true) can be added, for instance, to simplify missing value coding (see the item shown in Figure 5.2 as an example).¹



Hit- and Miss-Conditions need to be mutually exclusive (i.e., at any time, precisely one hit must be active for each class), if the option *Use first active hit/miss per class* (applies to all tasks) is not activated.

5.3.4 Use of Text Responses in Scoring Rules

Text responses can be automatically scored inside of the CBA ItemBuilder using keywords or pattern. The provided `matches()`-operator takes two arguments: The `UserDefinedId` of the component used to collect the text response (see section 3.9.1) and a regular expression (see section 6.1 for details).

```
matches(UserDefinedId, RegularExpression)
```

The `matches()`-operator can be used with regular expressions (see section 6.1) and with concrete texts. Examples for using the `matches()`-operator are illustrated in Figure 5.11.

¹Note that *Hit*- and *Miss*-conditions can be ordered in the *Task Editor* of the CBA ItemBuilder.

Matches-Operator for Text Scoring Example

This example illustrates the scoring of text responses using the `matches()`-operator, which can use regular expressions.

(Click here to set the input focus and press Ctrl + S to open the "Scoring Debug Window".)

The `userDefinedId` of the `SingleLineInputField` is "TextID".

Hit Definitions (Ordered):

- **ExactMatchHALLO:** `matches(TextID, "HALLO")`
- **MatchUmlaut:** `matches(TextID, "ä|Ä|ö|Ö|ü|Ü")`
- **EmptyString:** `matches(TextID, "")`
- **OnlyWhiteSpaces:** `not matches(TextID, "^(?!\\s*$).+")`
- **HelloWorld:** `matches(TextID, "\\s*[Hh]ello [Ww]orld\\s*")`
- **HelloOrWorld:** `(matches(TextID, "\\s*[Hh]ello\\s*") or matches(TextID, "\\s*[Ww]orld\\s*"))`
- **Anything:** `not matches(TextID, "")`
- **Never:** `true`

In this CBA ItemBuilder project, the option "Use first active hit per class (applies to all tasks)" is activated, i.e. the last defined hit "Never" with the syntax `true` will never be active, because before that, the hit "Anything" with the syntax `not matches(TextID, "")` will always be active.

File: MatchesExampleScoring.zip

FIGURE 5.11: Different *Hit*- definitions using the `matches()`-operator ([html](#)|[ib](#)).

Note that the logical operators `and`, `or`, and `not` can be combined with several `matches()`-operators and other conditions. Hence, there is no need to formulate too complex regular expressions since multiple expressions can be combined using multiple `matches()`-operators.

5.3.5 Use of FSM-Variables in Scoring Rules

The value of *FSM-Variables* can be used within scoring rules (i.e., *Hit*- and *Mis*-conditions). This is achieved using the `variable_in()`-operator:

```
variable_in(FSMVariable, SetOfValues)
```

An examples for using the `variable_in()`-operator is provided in the Figure 5.12 for the scoring of a *Drag-and-Drop* response format, implemented using *FSM Variables* (see section 4.2.6 for the implementation of *Drag-and-Drop*).

Drag-and-Drop Scoring using variable_in()-operator

This item illustrates the formulation of hit conditions using FSM variables based on the variable_in()-Operator. The values of the FSM-variables can be changed with drag-and-drop interactions.

(Click here to set the input focus and press Ctrl + S or Ctrl + M.)

Put the dices in **descending** order.

V_1 = 2

V_2 = 3

V_3 = 4

V_4 = 5

Hit-Definitions

- NoDragAndDrop: variable_in(V_DropCounter,0)
- AscendingOrder: ((variable_in(V_1,2) and variable_in(V_2,3)) and variable_in(V_3,4)) and variable_in(V_4,5))
- DescendingOrder: ((variable_in(V_1,5) and variable_in(V_2,4)) and variable_in(V_3,3)) and variable_in(V_4,2))
- NoSpecificOrder: true

Drop-Counter

- Number of Drop-Events content in Variable
V_DropCounter = 0

Notes:

- In this CBA ItemBuilder project the option "Use first active hit per class (applies to all tasks)" is enabled, i.e. the hit conditions are evaluated considering the order.
- Use Ctrl / Strg + S to open the "Scoring Debug Window", which shows the active hits according to the hit definition shown above.
- Use Ctrl / Strg + M to open the "State Machine Debug Window", which shows the values of the defined FSM-variables as well as the raised event names.

File: VariableAndDragAndDropScoring.zip

FIGURE 5.12: Use of FSM-Variables in Scoring-Conditions with the variable_in()-operator ([html](#)|[ib](#)).

The visited_all_values_of_variable()-operator can be used to check whether a variable has taken one or more concrete values in the course of test-taking (see Figure 5.13 for an example):

```
visited_all_values_of_variable(FSMVariable,SetOfValues)
```

5.3.6 Use of Positions for Free Drag-and-Drop in Scoring Rules

As described in section 4.2.6, the CBA ItemBuilder supports free drag and drop. The panel_position_range()-operator can be used to score the position of drag-and-drop elements (see Figure 5.14 for an example):

Scoring using `variable_in()`-operator with multiple values

(Click here to set the input focus and press Ctrl + S or Ctrl + M.)

Scoring using the `variable_in()`-operator with multiple values:

Value of Variable
"V_FSMVariable":

0

Hit-Definitions "Var1"

- Hit_CurrentValue2or3:
`variable_in(V_FSMVariable,2,3)`
- Hit_CurrentValue4or5:
`(variable_in(V_FSMVariable,4) or variable_in(V_FSMVariable,5))`
- Hit_DefaultVar1 and Hit_DefaultHitVar2: true

Hit-Definitions "Var2"

- Hit_ValueAnyTime6or8:
`visited_all_values_of_variable(V_FSMVariable, 6, 8)`
- Hit_DefaultVar1 and Hit_DefaultHitVar2: true

Notes:

- Use Ctrl / Strg + S to open the "Scoring Debug Window", which shows the active hits according to the hit definition shown above.
- Use Ctrl / Strg + M to open the "State Machine Debug Window", which shows the values of the defined FSM-variables as well as the raised event names.
- The item uses the option "Use first active hit per class".

File: VariableSetsScoring.zip

FIGURE 5.13: Using *set of values* and `visited_all_values_of_variable()`-operator in *Scoring-Conditions* ([html](#)|[ib](#)).

```
panel_position_range(Container, [CheckNonMembers], XStart, XEnd, YStart, YEnd,
                        Center, Component, Component, ...)
```

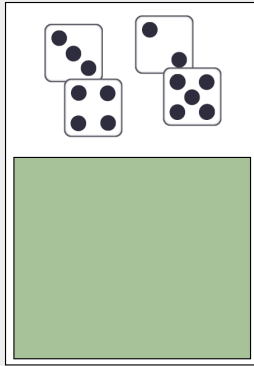
The operator evaluates to `true` if the (X,Y) positions of all given Components in the given Container are within the range given by XStart, XEnd, YStart and YEnd relative to the container's (X,Y) position. If the flag `CheckNonMembers` is not give or set to `true`, the operator only evaluates to `true` if the (X,Y)-positions of all other components in the given Container are outside the given range. The upper left corner of the component is used as (X,Y)-position of a Component if the flag `Center` is not provided as `true`.

Alternatively to the position of drag and drop element, the distance to score can also be evaluated. The `panel_distance_range()`-operator returns `true` if the mutual distance of all given Components in the given Container are within the given range between `MinDistance` and `MaxDistance`:

Free Drag and Drop Scoring Example

Free Drag & Drop can be scored using the Operator `panel_position_range()`.

Move all dice to the **green** area!



Hit-Definitions

- **Dice2Solved:**
`panel_position_range(PanalContainerID, false, 5,175,110,255,true,I_1)`
- **Dice3Solved:**
`panel_position_range(PanalContainerID, false, 5,175,110,255,true,I_2)`
- **Dice4Solved:**
`panel_position_range(PanalContainerID, false, 5,175,110,255,true,I_3)`
- **Dice5Solved:**
`panel_position_range(PanalContainerID, false, 5,175,110,255,true,I_4)`
- **AllDiceSolved:**
`((panel_position_range(PanalContainerID, false, 5,175,110,255,true,I_1) and panel_position_range(PanalContainerID, false, 5,175,110,255,true,I_2)) and panel_position_range(PanalContainerID, false, 5,175,110,255,true,I_3)) and panel_position_range(PanalContainerID, false, 5,175,110,255,true,I_4))`
- **OnlyDice2Solved:**
`panel_position_range(PanalContainerID, true, 5,175,110,255,true,I_1)`

(Click here to set the input focus and press Ctrl + S to open the "Scoring Debug Window".)

File: FreeDragAndDropScoring.zip

FIGURE 5.14: Example for scoring free drag and drop using the `panel_position_range()`-operator ([html|ib](#)).

```
panel_distance_range(Container, [CheckNonMembers], MinDistance,
                      MaxDistance, Center, Component, Component, ...)
```

5.3.7 Use of Events, States and Interaction in Scoring Rules

The CBA ItemBuilder provides various operators to incorporate events and the number of interactions into scoring conditions.

Number of Events: The number of events that have been raised during the execution of the current task can be used in scoring conditions. The CBA ItemBuilder considers an event to be raised even if it did not trigger a transition, and the count includes events raised by the `raise()`-operator:

```
raised_events()
```

If only the number of specific events should be counted, the `raised_nb_events()`-operator can be used:

```
raised_nb_events(SetOfEvents)
```

An even more advanced version of the `raised_nb_events()`-operator exist, that can be used to count how often one or multiple events were raised, while the item was in a particular state:

```
raised_nb_events_in_state(State, SetOfEvents)
```

Indicators for Events: In addition to the operators that count the events (of a particular type / within states), operators exist to check if an event was triggered. These operators evaluate to `true` (instead of returning the frequencies). The `raised_all_events(EventA, EventB)` return `true` if all events listed in the set of events (e.g., `EventA` and `EventB`) were raised:

```
raised_all_events(SetOfEvents)
```

Again, an more advanced version of the `raised_nb_events_in_state()`-operator exist, that can be used to check if one or multiple events were raised, while the item was in a particular state:

```
raised_all_events_in_state(State, SetOfEvents)
```

The following item shown in Figure 5.15 illustrates the use of the event-related operators.

Number of State Visits: The `visited_nb_states()`-operator return the number of the visits for a set of states during the execution of the current task:

```
visited_nb_states(State, State, ...)
```

Indicators State: The `is_last_state()`-operator returns `true` if the last state the finite-state machine is one of the given states in the `SetOfStates`:

Raised Event Scoring Example

Illustration of the operators `raised_events()` and `raised_all_events()`.

(Click here to set the input focus and press Crti + S to open the "Scoring Debug Window".)

Use the following buttons to trigger the assigned events and press Strg+S / Cntr+S to check the scoring using the "Scoring Debug Window":

| | |
|--------------------|---|
| Trigger EV_ButtonA | Event "EV_ButtonA" was triggered 0 times. |
| Trigger EV_ButtonB | Event "EV_ButtonB" was triggered 0 times. |
| Trigger EV_ButtonC | Event "EV_ButtonC" was triggered 0 times. |
| Trigger EV_ButtonD | Event "EV_ButtonD" was triggered 0 times. |

Hit Definitions ("Use first active hit per class" activated):

- `Hit_user_interactions_below5`: `[raised_events() < 5]`
- `Hit_raised_event_equals_zero`: `[raised_events() == 0]`
- `Hit_raised_eventsAandB_below2`: `[raised_nb_events(EV_ButtonA, EV_ButtonB) < 2]`
- `Hit_raised_all_events`: `raised_all_events(EV_ButtonA, EV_ButtonB, EV_ButtonC, EV_ButtonD)`
- `Hit_nothing_else`: `true`

File: RaisedEventScoringExample.zip

FIGURE 5.15: Example for using events for scoring ([html](#)|[ib](#)).

```
is_last_state(SetOfStates)
```

While the `is_last_state()`-operator refers to the last state of the finite-state machine, the `visited_all_states()`-operator can be used to check if all states listed in the `SetOfStates` were visited during the execution of the current task:

```
visited_all_states(SetOfStates)
```

Number of Interactions: A simple generic operator is provided that counts the number of user-interactions within the current task:

```
user_interactions()
```

Note that this operator counts the total number of interactions within the running

task. Counting specific interactions in *FSM variables* is possible using the finite-state machine (see section 4.4).

Elapsed Time: Another generic operator is provided that measures the elapsed time in the current task:

```
elapsedTime()
```

The scoring-operator `elapsedTime()` counts the total time in the current task. Measuring more specific time intervals is possible using finite-state machines (see section 4.4.6 and the example provided in Figure 4.60).

5.3.8 Use of Specific Operators in Scoring Rules

Tree Components: The scoring of response formats created using components of type `Tree`, `TreeView` and `TreeChildArea` (see section 3.9.9) is supported with the following operators:

- The operator `current_node()` allows to check if in a particular `Tree` a `RegularExpression` matches to the node path ID of the current node:

```
current_node(Tree, RegularExpression)
```

- The operator `exists_nodes()` returns number of nodes in the given `Tree` whose node path ID matches at least one of the given `RegularExpressions` (each node counts once only):

```
exists_nodes(Tree, RegularExpression, RegularExpression, ...)
```

- The operator `visited_nodes()` returns number of visited nodes in the given `Tree` whose node path ID matches at least one of the given `RegularExpressions` (each node counts once only):

```
visited_nodes(Tree, RegularExpression, RegularExpression, ...)
```

- The operator `matches_nodes()` returns the number of nodes in the given `Tree` whose node path ID matches the `NodeIdPattern` and whose column values match the specified `ColumnPatterns`. The first `ColumnPattern` corresponds to the node name, the second `ColumnPattern` to the first additional column, etc. (each node counts once only).

```
matches_nodes(Tree, NodeIdPattern, ColumnPattern, ColumnPattern, ...)
```

Pages: An operator `current_page()` is provided to check if a specified `Page` is currently displayed (or is displayed within the specified `PageArea`):

```
current_page(Page)
current_page(Page, PageArea)
```

For browser-components that support the bookmark function (see section 3.13.2) the following operator can be used to check if a page was bookmarked:

```
bookmarked(Page)
```

Spread Sheets: Operator to score value (or the computed formula) entered in a spreadsheet table with a given `UserDefinedId` (see section 3.9.8) as integer value:

```
integer_value(UserDefinedId, RoundingMode, Default)
```

The parameter `RoundingMode` can take the values `up`, `down`, `half_up` and `half_down`. If the text content is empty or does not represent a number, the `Default` value is returned.

To score the entered formula (instead of the value), the `matches()`-operator provides the additional argument `Selector`. If the value `formula` is requested, the operator evaluates the formula text of a spreadsheet table cell (instead of the formula value):

```
matches(Component, RegularExpression, Selector)
```

Highlighting: The following operators are provided to score the response format of multiple text highlighting (see section 3.8.3):


```
highlighted(RichText, RichText, ...)
```

```
complete(Selection, Selection, ...)
```

```
partial(Selection, Selection, ...)
```

5.3.9 Note on Scoring with PageAreas

PageAreas (see section 4.1.4) can be used to embed existing pages as content when designing pages. The CBA ItemBuilder allows that identical content can be re-used multiple times in different *PageAreas* on a single page, as illustrated in Figure 5.16. It is therefore generally necessary to add the *UserDefinedId* of the *PageArea* to all references to components displayed within *PageAreas*.



Components that are displayed in *PageAreas* need to be addressed using the following scheme: {UserDefinedId-of-PageArea}. {UserDefinedId-of-Component}.

5.3.10 Scoring Rules and Result Text

As shown in Figure 5.8, the CBA ItemBuilder integrates the handling of numerical and string responses into the *Scoring Rules* (i.e., *Hit*- and *Miss*-conditions) using the *result_text()*-operator. For each class, the active hit is determined first. If the option *Use first active hit/miss per class (applies to all tasks)* is activated (see section 5.3.3), this is the first condition within a class that applies. Otherwise item authors need to make sure that all conditions are mutually exclusive within each class. If the active hit contains a *result_text()*-operator, numerical or text input is provided as *Result-Text*.

5.3.11 Missing Value Coding for Tasks with Multiple Pages

The following examples shows, how to define scoring for single choice and multiple choice items including hits for *not reached items* and *omitted responses*. For this purpose, a variable is defined in the finite-state machine that counts how often a page was visited.

PageArea Scoring Example

This item illustrates how the same content can be inserted two times on a single page into two different PageAreas (PA1 and PA2) and how the components (i.e., the RadioButtons `rbA`, `rbB`, `rbC` and `rbD`) included in the PageArea can be used to create two different outcome variables (`ScorePA1` and `ScorePA2`).

ID: PA1

☐ Option A

☐ Option B

☐ Option C

☐ Option D

ID: rbA

ID: rbB

ID: rbC

ID: rbD

ID: PA2

☐ Option A

☐ Option B

☐ Option C

☐ Option D

ID: rbA

ID: rbB

ID: rbC

ID: rbD

(Click here to set the input focus and press Ctrl + S to open the "Scoring Debug Window".)

| Name | Class | Syntax |
|-------------|----------|---------|
| PA1_A | ScorePA1 | PA1.rbA |
| PA1_B | ScorePA1 | PA1.rbB |
| PA1_C | ScorePA1 | PA1.rbC |
| PA1_D | ScorePA1 | PA1.rbD |
| PA1_Missing | ScorePA1 | true |
| PA2_A | ScorePA2 | PA2.rbA |
| PA2_B | ScorePA2 | PA2.rbB |
| PA2_C | ScorePA2 | PA2.rbC |
| PA2_D | ScorePA2 | PA2.rbD |
| PA2_Missing | ScorePA2 | true |

File: PageAreaScoringExample.zip

FIGURE 5.16: Item illustrating scoring when *PageAreas* are used ([html](#)|[ib](#)).

Items without response on a not visited page are coded as *not reached* (NR), missing responses on visited pages are coded as *omitted response* (OR). For more details, see the item shown in Figure 5.17 and use the *Scoring Debug Window* (as described in section 1.5).

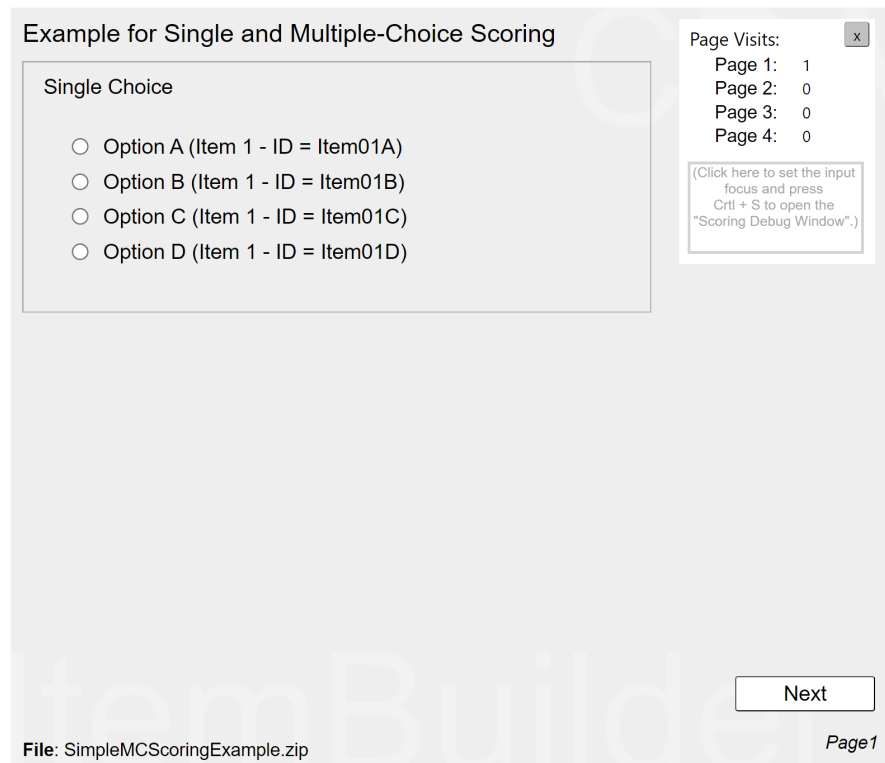


FIGURE 5.17: Item illustrating *Missing Value Coding* for a multi-paged item ([html](#)|[ib](#)).

5.4 Automatically Generated Variables



The CBA ItemBuilder runtime will create some selected variables automatically:

- *reactionTime*: Time (in milliseconds) between the start of the task execution and the first user interaction.

- *execTime*: Time in (milliseconds) since the start of this task execution.
- *nbInteractions*: Number of user interactions since start of the current task execution.

Deployment software for CBA ItemBuilder tasks (see chapter 7) can use identical tasks multiple times and can allow to re-visit tasks. For that purpose, the runtime also computes the cumulative variables:

- *reactionTimeTotal*: Accumulated time (in milliseconds) between the start of the task execution and the first user interactions in previous executions of the task (excluding the last execution, that can be found in the variable *reactionTime*).
- *execTimeTotal*: Accumulated time (in milliseconds) in previous executions of the task (excluding the last execution, that can be found in the variable *execTime*).
- *nbInteractionsTotal*: Accumulated number of user interactions in previous executions of the task (excluding the last execution, that can be found in the variable *nbInteractions*).

5.4.1 Scoring Complete Tasks with Weights



This scoring is included in the CBA ItemBuilder to maintain compatibility with old items. It is considered outdated, as it only allows to derive one score per tasks.

If only dichotomous scoring is required for the complete *Task*, the CBA ItemBuilder implemented a simple approach.

- *MinHits*: For each *Task* can be defined, how many *Hit*-conditions must be fulfilled, that the task is scored as `True`.
- *Weight*: Each *Hit*-/ and *Miss*-conditions is assigned to a *Weight*.
- *Class*: Each *Hit*-/ and *Miss*-conditions is assigned to a *Class* and to each *Class* either *Hit*-/ and *Miss*-conditions are assigned.

Each task provides the following results:

- *result*: Overall result (1 if the at least the number of hits is active that is defined as the property *MinHits*, 0 otherwise).
- *nb_Hits*: Number of (active) hits.
- *Hit_weight*: Total weight of hits.
- *nb_Misses*: Number of (active) misses.
- *Miss_weight*: Total weight of misses.
- *credit_Class*: Name of the *Class* with the highest value. The value is computed as the sum of weight for all active *Hits* (in classes with *Hits*) or all active *Misses* (in classes with *Misses*).
- *credit_weight*: Weight of the class with the highest class weight.

5.5 Checklist and Complete Workflow



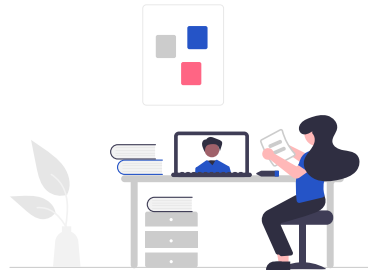
As a summary the following list describes the typical workflow that is required for implementing automatic scoring in the CBA ItemBuilder:

- Prepare the implementation of scoring by defining explicit *User Defined IDs* (see [3.7.4](#)) for all components that should be used for scoring. It is not possible to define scoring using the automatically generated *User Defined IDs* that start with a $\$$ -sign.
- Define a task as an entry point for the CBA ItemBuilder project. Since the scoring definition is done per task, a task must always be defined first (see section [3.6](#) for details).
- When tasks are defined, define *Classes*. For each variable that should be included in the result data for a particular *Task* define one variable. For all newly created items activate the option *Use first active hit per class* (applies to all classes).
- Define *Hit-Conditions*, that evaluate to `true` if the required conditions are fulfilled (see section [5.3.2](#). Order the *Hit*-conditions and add a default condition with the hit-syntax `true` as last condition. This will ensure that each class has one active hit (see section [5.3.3](#)). For a usual workflow *Miss*-conditions are not needed, neither are *Weights*.
- Extract string information using the `resultText()`-operator, if necessary. While hits are as values of categorical variables, the *Result-Text* can be used to capture numerical or text responses.
- Assign hits to classes. Classes fulfill the function of variables in the scoring of CBA ItemBuilder projects. Besides the unique name of the class (variable name), a description of the class can be entered in the *Class Comment* (variable description).
- Decide how to handle missing response and implement, if necessary, additional *Hit-Conditions* for omitted responses and not reached questions (see section [5.3.11](#)).

- Test the scoring implementation using the *Scoring Debug Window*. If the option `use first active hit per class` (applies to all classes) was not activated, make sure that exactly one hit (or miss) is active for each class at any point in time (see [8.4.2](#)).

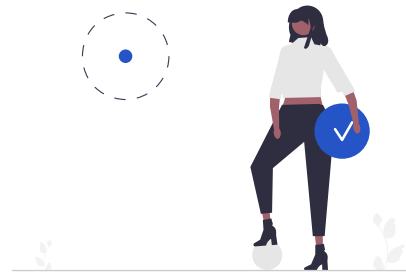
6

Recipes and Examples



This chapter provides concrete recipes and examples for creating innovative assessments using the CBA ItemBuilder. The first section 6.1, starts with tips about *regular expressions*, mainly used for scoring purposes and to restrict the possible text input. The subsequent sections focus on *resource files* (i.e., images, audio, and video files; see section 6.2) and on the *global properties* (i.e., definitions that are shared within a CBA ItemBuilder project file; see section 6.3). Examples illustrating the use of *Conditional Links* and *Finite-State Machines* syntax for different purposes are described in section 6.4. Section 6.5 focuses on *digital calculators* and the implementation using either *Finite-State Machines* or `ExternalPageFrames`. More examples for the use of *HTML5/JavaScript* content with `ExternalPageFrames` are provided in section 6.6. Section 6.7 summarizes possible approaches to implementing adaptive assessments using the CBA ItemBuilder. Finally, section 6.8 focuses on efficiency and tricks to implement content using the CBA ItemBuilder with less effort, and section 6.9 refers to creating items in multiple languages.

6.1 Regular Expressions



For different purposes, the CBA ItemBuilder supports the use of so-called *Regular Expressions*. Regular expressions are sequences of characters (i.e., strings) that can be used to formulate patterns that match text with specific properties. The CBA ItemBuilder allows regular expressions to restrict the input to `SingleLineInputFields` and `InputFields`. If a regular expression is defined in components of this type as `Input Validation Pattern`, only text that matches the defined pattern can be entered. The CBA ItemBuilder also supports the use of regular expressions for the scoring of text responses and for conditional links. The syntax for defining patterns as regular expressions that can be used for restricting input as well as for matching responses is listed in the Appendix (B.3).¹ In regular expressions used for scoring, conditional linking, or for definitions in any other syntax part of the CBA ItemBuilder, `\` need to be escaped (i.e., replaced with `\\`). Some characters have special meaning in regular expressions and must be escaped. All escaped characters begin with the `"` character. Within a regular expression, only `\`, `-`, and `]` need to be escaped.²

6.1.1 Valid `UserDefinedIds` as Regular Expression

To illustrate how regular expressions can be used to define patterns for valid text, we illustrate how the possible schema for `UserDefinedIds` (see section 3.7.4) can be translated into a regular expression:



Only strings matching this regular expression can be used as `UserDefinedId`: `([A-Za-z][A-Za-z_0-9]*)`

¹Note that the flavor of regular expressions used by the CBA ItemBuilder is based on Unicode: <http://www.unicode.org/reports/tr18/>

²For creating and testing regular expressions, there are a number of helpful web resources available, such as <https://regex101.com> and <http://regexr.com>.

According to the syntax of regular expressions (see table B.3 in the appendix for details) this expression defines the following pattern:

- Only uppercase or lowercase characters are allowed as first characters: `[A-Za-z]`. This matches the requirement that a User-Defined Id must start with a character and must have a length greater or equal one.
- Digits and underscores are allowed after the first position in addition to uppercase and lowercase characters: `[A-Za-z_0-9]*`, but no umlauts / vowel mutations are allowed.
- The number of characters is not limited, and white spaces are not allowed.

This restriction of allowed characters also applies to name of *Tasks* (see 3.6.1), name of Finite-State Machine *Variables* (see section 4.2) and *States* (see section 4.4).

6.1.2 Scoring (Text) Responses with Regular Expressions

Regular expressions are often used to define hit or miss conditions when scoring CBA ItemBuilder tasks. The item shown in Figure 6.1 illustrates with a synthetic example the use of regular expressions for scoring text responses.

Regular Expressions with Alternatives: The first hit (`Variable1_Correct`) defined in the example uses the syntax `[d|D]og` within a regular expression, which recognizes both upper and lower case forms of dog:

```
matches(txtVar1, "\\s?[d|D]og\\s?")
```

This syntax is supplemented by `\\s?`, i.e. an optional space before or after the searched words. As described above, the expression `\\s` (see appendix B.3) was thereby paraphrased as `\\s`.

Combination of `matches()`-operators: Alternatives can be formulated within regular expressions. However, it is often easier to combine several `matches()` operators in the scoring condition. The hit `Variable1_Wrong` is an example of this, where the already described case detection is combined with an operator that detects empty text fields:

```
(not matches(txtVar1, "\\s?[d|D]og\\s?") and not matches(txtVar1, ""))
```



When combining `matches()` operators with `and` and `or` the rules for bracketing multiple operands must be observed (see section 4.1.3). The negation with `not` can additionally be added as part of the operands.

Scoring Input Field Example

Enter correct or wrong text responses and check the "Scoring Debug Window". See table "Hits" for matched scoring conditions.

Click here and press Ctrl + S (Strg + S) to open the Scoring Debug Window.

Item 1: Text entry (correct = Dog)

Hit Definition for Class "Var1":

- Variable1_Correct: `matches(txtVar1, "\\s?[d|D]og\\s?")`
- Variable1_Wrong: `(not matches(txtVar1, "\\s?[d|D]og\\s?") and not matches(txtVar1, ""))`
- Variable1_Missing: `matches(txtVar1, "")`

Item 2: Number entry (correct = "3.5")

Hit Definition for Class "Var2":

- Variable2_Correct: `matches(txtVar2, "\\s?3[.]5\\s?")`
- Variable2_Wrong: `(not matches(txtVar2, "\\s?3[.]5\\s?") and not matches(txtVar2, ""))`
- Variable2_Missing: `matches(txtVar2, "")`

Item 3: Number entry (correct between "2.5 and 7.3")

Hit Definition for Class "Var3":

- Variable3_Correct: `(matches(txtVar3, "\\s?[2][.][5-9]\\s?") or matches(txtVar3, "\\s?[3-6][.][0-9]\\s?") or matches(txtVar3, "\\s?[7][.][0-33]\\s?"))`
- Variable3_Wrong: `(not ((matches(txtVar3, "\\s?[2][.][5-9]\\s?") or matches(txtVar3, "\\s?[3-6][.][0-9]\\s?") or matches(txtVar3, "\\s?[7][.][0-33]\\s?")) and not matches(txtVar3, ""))`
- Variable3_Missing: `matches(txtVar3, "")`

File: ScoringInputFieldExample.zip

FIGURE 6.1: Example item illustrating scoring with regular expressions ([html](#)|[ib](#)).

Empty Response: For encoding missing (text) responses, an empty string can be provided as argument for the `matches()`-operator to check whether a character was entered at all:

```
matches(userDefinedIDInputField, "")
```

This pattern was used in the item in Figure 6.1 for defining hits to identify missing responses. Because the CBA ItemBuilder's implementation of the `matches(userDefinedIDInputField, "")` (i.e., the use of "" empty expressions) is also triggered in multiline `TextFields` as soon as one empty line is included, it may also be useful to check if no characters have been entered with the following regular expression:

```
not matches(userDefinedIDInputField, "^(?!\\s*$).+)"`
```

Figure 6.2 illustrates the difference between empty expressions "" and the expression to match white spaces "(?!\\s*\$) .+".

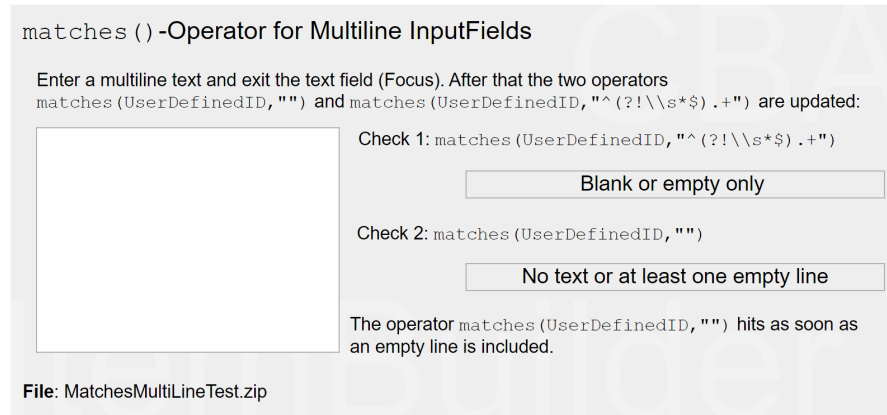


FIGURE 6.2: Example item illustrating the different approaches to check empty input ([html](#)|[ib](#)).

Decimal Numbers (with . or ,): Another application of a selection via | can be used to accept both dot and comma as decimal separators ([. |]). In cultures where a thousands separator is not typically used, this can be useful when checking decimal numbers:

```
matches(txtVar2, "\\s?3[. |,]5\\s?")
```

As shown in the second item in Figure 6.1, this can be used to check whether the correct answer (3,5 or 3.5) was entered in the `SingleLineInputField` with the `UserDefinedId: txtVar2`.

Numerical Ranges using Regular Expressions: By combining the components, it is also possible to check whether an entered decimal number is within a desired range:

```
((matches(txtVar3, "\\s?[2][. |,][5-9]\\s?") or  
 matches(txtVar3, "\\s?[3-6][. |,][0-9]\\s?")) or  
 matches(txtVar3, "\\s?[7][. |,][0-33]\\s?"))
```

Scoring conditions like this (see `Hit Variable3_Correct`) can also be negated and combined with a check for empty inputs (see `Hit Variable3_Wrong`).

6.1.3 Input Validation with Regular Expressions

Regular expressions are also commonly used when creating items with the CBA ItemBuilder to limit the possible inputs of text. A restriction of the possible characters that can be entered to text fields, such as `SingleLineInputFields` and `InputFields`, is a common requirement for the implementation of items with (short) text responses. To apply an input restriction using the CBA ItemBuilder, components that support this feature provide the property `Input Validation Pattern` in section *Misc* of the *Properties*-view.

To define an input validation, the component must be selected in the *Page Editor*. If necessary, the context menu can be used to open the *Properties*-view (entry *Show Properties View*). Regular expressions are entered in as *Input Validation Pattern* property (see Figure 6.3).

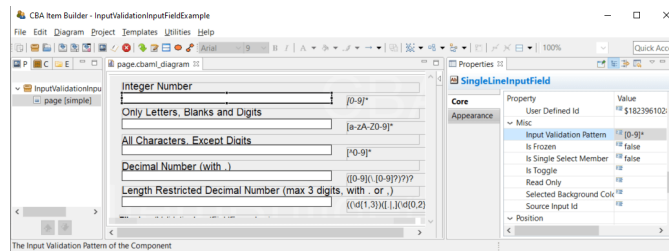


FIGURE 6.3: Property `Input Validation Pattern` in the *Properties*-view.

When using regular expressions to restrict text input, please note that the pattern must be valid not only for the final response but also for all intermediate steps in the input of the response.



The default value of the component must match the regular expression.

Since the input is made characterwise, the input `3.`, for example, must also be valid with respect to a particular pattern so that a decimal number `3.0` can be entered. This is not necessary for the identification of different free text responses with regular expressions (see 6.1.2).

The item shown in Figure 6.4 illustrates some of the regular expressions often used to restrict text entry for `SingleLineInputFields`.

Restricting the possible characters that can be entered into a `SingleLineInputFields` (or `InputField`) simplifies scoring for short text responses and also affects the design of tasks. The rejected characters are, however, included in the *Log Events* provided by components of type `SingleLineInputField` and `InputField`. In the following we provide commented and documented regular expressions for input validation that are used in the item shown in Figure 6.4.

Examples for Input Restrictions using Input Validation Pattern

Integer Number

/[0-9]*

Only Letters, Blanks and Digits

[a-zA-Z0-9]*

All Characters, Except Digits

[^0-9]*

Decimal Number (with .)

([0-9](\.[0-9]?)?)?

Length Restricted Decimal Number (max 3 digits, with . or ,)

((\d{1,3})([.](\d{0,2}))?)?

File: InputValidationInputFieldExample.zip

FIGURE 6.4: Item illustrating different Input Validation Pattern ([html](#)|[ib](#)).

Integer Numbers: Only the numbers 0 to 9 can be entered in an input field with the following Input Validation Pattern:

/[0-9]*

Empty strings are allowed (because of the *).

Only Letters, Blanks and Digits: Only the numbers 0 to 9, small letters a to z and capital letters A to Z can be entered in an input field with the following Input Validation Pattern:

/[a-zA-Z0-9]*

Only Single Letters: Only one single letter A-Z, or an empty string is allowed when using the following Input Validation Pattern:

/[A-Z]{0,1}?

All Characters, Except Digits: With the help of the following Input Validation Pattern it can be achieved that all characters except digits can be entered:

/[^\d]*

Decimal Number (with .): The input of decimal numbers is possible with this Input Validation Pattern, where both only . as decimal separator is allowed.

```
([0-9](\.[0-9]?)?)?
```

The expression allows one digit left to the digital delimiter (\.). The digital delimiter and one additional digit is optional. Empty strings are allowed.

Length Restricted Decimal Number (max 3 digits, with . or ,): If inputs are to be limited in length, this can also be implemented with regular expressions:

```
((\d{1,3})([. ,](\d{0,2}))?)?
```

In this example, only digits ‘(\d{1,3})’, one to three characters, are accepted before the decimal delimiter. A dot or comma are allowed as decimal delimiters ([. ,]). Up to three digits can be entered right to the decimal delimiter ((\d{0,2})). The second group “()?” is optional ([. ,](\d{0,2}))? and empty strings are allowed.

Feedback using Input Validation Events: If an Input Validation Pattern applies to the current input into a SingleLineTextField OR a InputField, an FSM Event can be triggered. The following example in Figure 6.5 illustrates the use of Input Validation Events, Raised In Events and Raised Out Events to inform test-takers about allowed characters and ignored inputs.

Examples for Input Validation Events

SingleLineTextField

[0-9]*

InputField

[0-9]*

Not focused: Click into one of the two white text fields and enter a number.

File: InputValidationEventExample.zip

FIGURE 6.5: Item illustrating the use of Input Validation Events ([html|ib](#)).

Regular Expressions are one of the standard approaches to search for patterns in text strings. In the CBA ItemBuilder, regular expressions are used at *Runtime* for scoring text responses and for restricting input. The examples described in this section can

only illustrate how regular expressions can be used. Concrete requirements can lead to more complex regular expressions, which can also be used to prevent the entry of sensitive information such as telephone numbers, for example. The regular expressions used must be tested in a systematic test strategy (see, for instance, 8.4.2) to collect reliable empirical data.

6.2 Ressources Files



Multimedia resources (images, videos, and audio files) are of great value in contextualizing interactive assessment, creating simulations and small experiments, and designing static and interactive content.

6.2.1 Preparing Image Files

Section 3.10.1 describes supported file formats. As it supports lossless compression, the PNG format (*Portable Network Graphics*) is often a good choice when preparing image files for designing items with the CBA ItemBuilder.

Some open source tools that might be helpful include:

- Simple image editor, e.g., [paint.net](https://www.paint.net/)
- More complex application, e.g., [gimp](https://www.gimp.org/)

For CBA ItemBuilder versions prior to Version 10.0, images were suggested to display text with non-web-safe fonts.³ In particular for online deployment (see section 7.2.1), image size can make a difference. If possible, images should be reduced to the

³The generation of many image files (e.g., to display texts in a non-web-safe font) can also be automated before the images are then inserted into the CBA ItemBuilder. [Here](#) is a simple example where texts are read from an Excel file and generated as small image files

required size, and stored as efficiently as possible (for instance, using vector quantization algorithms for PNG images, as provided by [pngquant](#)).

How to use transparent images? As shown in Table 3.9, transparency is also supported by the PNG format. To use images in the PNG format with transparent background, the property `Is Transparent` of the component that will be used within the CBA ItemBuilder to show the image, must be set to `true` (see Figure 6.6).

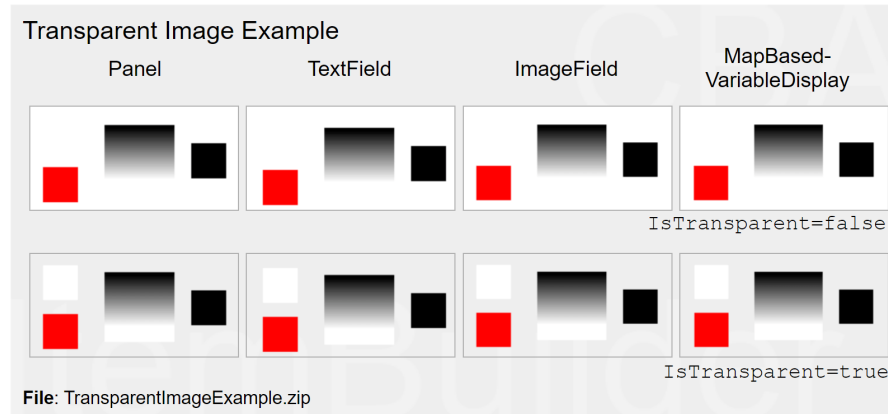


FIGURE 6.6: Example item illustrating transparent images in the PNG-format ([html](#)|[ib](#)).

How to create multiple layers? The example shown in Figure 6.7 illustrates that components of type `ImageField` can be arranged in different *Z-order* (see section 2.11.4 and also section 3.7.5 for an example).

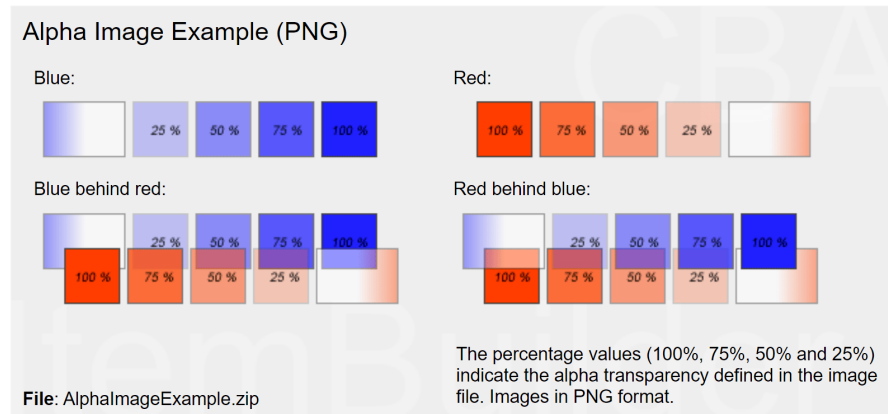


FIGURE 6.7: Example item illustrating `ImageFields` with alpha transparency and different *Z-Order* ([html](#)|[ib](#)).

Advanced scenarios combining and overlaying several images with semi-transparency currently need to be prepared outside the CBA ItemBuilder. To implement semi-transparency, please merge the images in a graphics tool and, if necessary, cut them into several pieces. The components described in section 3.10.2 can be used to design items with multiple images.

How to find the original size of an image in pixel? The size of images in pixels is relevant for the design of computer-based items (not only with the CBA ItemBuilder). If the images are too small, i.e., smaller than the size in which they are displayed, then a blurred impression results from the extrapolation during enlargement. If the images are too large, they must first be loaded entirely in the browser before they can then be reduced to the actual size for display. This consumes bandwidth unnecessarily.

To learn about the size of an image file, right-click the file in either *Explorer* or *Finder*, and select *Get Info (Mac)* or *Properties (Windows)* to see the physical size of images (in pixels). On Windows, a click either on *Details* or *Summary* can be required to access the information (the tab name will depend on the specific operating system).

To support the display in the size that matches the file size, the CBA ItemBuilder will automatically resize the component (e.g., `ImageFiled`) to the size of the image after linking the image to the component.

Why to resize images before adding them to the CBA ItemBuilder? It is recommended to reduce images for use in the CBA ItemBuilder to the size with which the images should actually be used. This does not only reduce the file size and thus potentially the loading time in online deliveries, it also allows a better control over the result of the scaling, as shown in Figure 6.8.

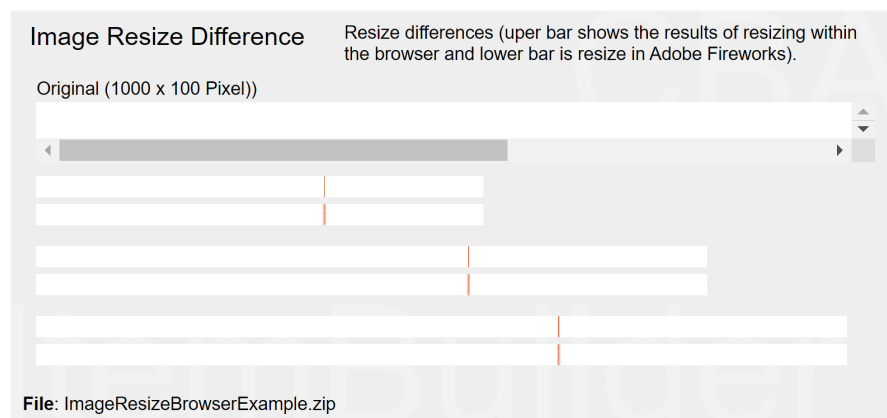


FIGURE 6.8: Example showing images resized to different sizes ([html](#)|[ib](#)).

Always use Proportional Scaling: If image files are resized in the CBA ItemBuilder, please make sure that the aspect ratio does not change (proportional scaling). The

easiest way to do this is to resize the image display component using the grid points at the corners (and not to use the grid points in the middle between the corners).



It is recommended to always insert images into CBA ItemBuilder project files in the size in which they will be displayed (to save disk space and bandwidth). For a professional look, images must never be compressed or stretched, i.e., the aspect ratio (or preferably the width and height in pixels) of an image file must be identical to the aspect ratio (or preferably the width and height in pixels) of the component displaying the images.

6.2.2 Preparing Audio and Video Files

Audio and video files can be used in CBA ItemBuilder *Tasks* as part of the instruction, as a stimulus (for example, for listening comprehension tasks), or for test accommodation (i.e., reading aloud test accommodation). In the current version, CBA ItemBuilder supports libraries for generating conversion from *Text to Speech* only via `ExternalPageFrames`. The default way is to create audio files and videos (analogous to images) before using them in the CBA ItemBuilder and import media resources via the *Resource Browser* (see section 3.10.1).⁴ Some open-source tools that might be helpful include:

- Tools for audio editing and converting, e.g., [audacity](#)
- Tools for video converting (e.g., [ffmpeg](#), [VLC player](#))
- Tools for video capturing (e.g., [OpenShot](#) or [OBS](#))

How to convert audio and video files formats? Various tools, such as the [VLC player](#), exist that support converting file formats. The CBA ItemBuilder supports the file formats listed in section 3.10.1 for audio and video files. For video resources, not only the file format but also the codec used must be taken into account. Whether an audio or video file in a particular format can be used for test delivery depends not only on the CBA ItemBuilder. The browser or web browser component used to display the item content must support the format used. Therefore, it is recommended to keep the sources, especially for self-created audio and video files, and to test the usability in the concrete setting before converting and integrating many files.

How to change the volume of the audio files? The volume of audio and video is fixed within the files and can, within certain limits, be increased or decreased as part of the pre-processing. For audio files, this can be done, for instance, with [audacity](#). The [VLC player](#) and other tools support this functionality for video files.

How test-taker can change the volume? The volume of audio resources within the

⁴A free tool to generate speech from text can be found here: <https://ttsmp3.com/>.

file is one of many factors determining how loud audio files are played during *Run-time*. The components used for embedding `Audio` and `Video` resources can also change the audio output volume (see section 3.10.3 for details), either using the visible *Controls* or using a *Finite-State Machine* operator.

6.3 Global Properties



Each CBA ItemBuilder project file has global settings that are used for all tasks (and thus for all pages) of the project.



If the settings of components cannot be changed or defined via the *Properties-view*, then they can be adjusted in the *Global Properties*.

6.3.1 Project Settings

As described in section 3.2.2, the context menu that is available using right-click on the project name in the *Project View* gives access to the *Project Settings* using the entry *Global Properties*.

Presentation height/width: The display size of an assessment component is set in pixels for a currently opened project file in the *Global Properties* (see section 3.2.2 and also section 3.6.2). Content created with the CBA ItemBuilder can be displayed in different ways (see section 7.2.1), including the use of *Proportional Scaling* (see section 2.4).

Link Color / Visited Link Color: General settings for the link color and the color of visited links can be set for the entire project in the *Global Properties*. These settings

Highlight Color: For the response format *Highlighting* (see section 3.8.3) the default color can be defined.

Page Size Warnings: Before executing a preview, the CBA ItemBuilder checks if all pages stay within the size defined as *CBA Presentation Size* (see section 3.2.2 and also section 3.6.2). If scrollable areas are to be created, e.g. with the help of components of the type `PageArea` (see section 3.5.4), the warning message (shown in Figure 3.51) can be ignored or deactivated via the setting in the *Global Properties*.

Deprecated Features / Features under Development: The tab *Project Settings* contains additional options that are kept for compatibility reasons, are currently not used or not documented so far. The options *Deactivate Firefox context menu*, *Ctrl+F - Page*, and *Right-To-Left Orientation and XLIFF Support* are not available, tested or supported in the current version of the CBA ItemBuilder. The *Default Language / Extension* should match the language of the test, but the setting is not yet used at many places.

6.3.2 Translations and Icons

The tab *Translations* (see Figure 6.9) can be used to provide translated texts for various English default texts that can be of relevant at runtime. Default icons that are used in different components can be linked to embedded resource files in a CBA ItemBuilder *project file* in the tab *Icons*. Note that the files must be added using the *Resource Browser* (see section 3.10.1) before the icons can be assigned.

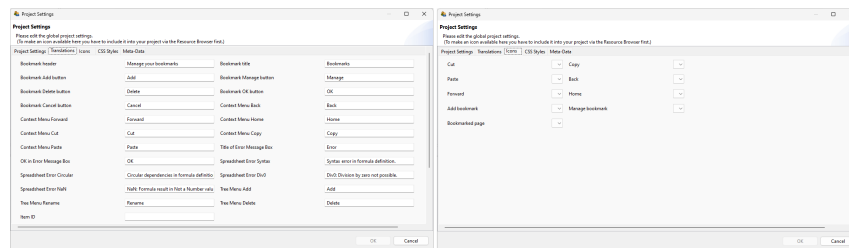


FIGURE 6.9: *Tabs Translation and Icons in the Global Properties of CBA ItemBuilder Project Files.*

6.3.3 CSS Styles

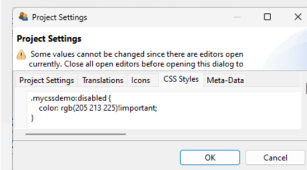
If items are to be designed according to precise specifications, selected properties can be missing in the CBA ItemBuilder. For example, in the current CBA ItemBuilder, the text color for disabled buttons cannot be defined via the Properties View. For such (rather extraordinary) use cases and refinements, CSS classes can be used for HTML5 generation. CSS styles are defined by entering valid CSS syntax into the text field in the tab *CSS Styles* of the *Global Properties*. CSS classes can be assigned by entering the class name into the field `css class name` in the *Properties*-view of components (see Figure 6.10 for an example).

In this example, the following CSS definition is used to define the text color of buttons when they are de-activated (frozen):

```
.mycssdemo:disabled {
    color: rgb(205 213 225)!important;
}
```

CSS Class Name Example

CSS (=Cascading Style Sheets) code can be defined in the section *CSS Styles* of the *Global Properties*:



File: CSSClassNameExample.zip

Components use the specific CSS class added to the *Properties View* (see: Misc / CSS Class Name):

| Property | Value |
|-------------------|---------------|
| Use Status Images | false |
| Identification | |
| User Defined Id | btDemoWithCSS |
| Misc | |
| Css Class Name | mycssdemo |

Example: The following button uses the CSS class "mycssdemo" to define the color for the text when the button is frozen.

Color defined in CSS:

btDemoWithCSS

Default Color:

btDemoWithoutCSS

unsetFrozen SetFrozen

FIGURE 6.10: Example item illustrating the use of CSS Styles ([html|ib](#)).

Please note that the use of CSS styles should be the exception since these CSS styles may not be adopted in case of a possible change of the delivery technology and that the CBA ItemBuilder does not check the validity of the CSS styles.



User-defined CSS styles can be a potent tool for customizing the appearance and behavior of items, which is, however, specific to the HTML output format of the CBA ItemBuilder used in the current versions. Please use this option with the necessary technical caution.

6.3.4 Metadata (about Content)

The CBA ItemBuilder provides an interface to define *metadata* for each *project file* (see Figure 6.11), that can be edited in the *Global Properties* (right-click the project name in the *Project View*), in the tab *Meta-Data*.

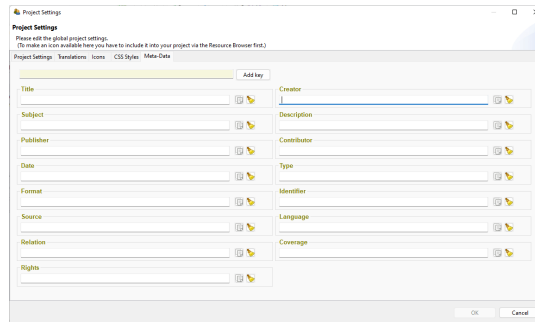


FIGURE 6.11: Tab *Meta-Data* in the *Global Properties* of CBA ItemBuilder Project Files.



The CBA ItemBuilder is a tool for creating and sharing assessment content. In the metadata, the information with which item authors want to enable the sharing of items can be stored in each project file.

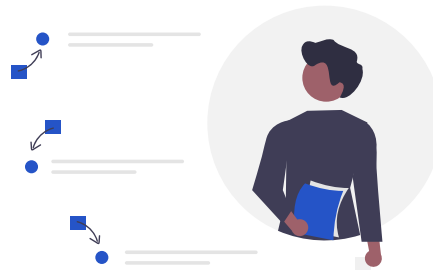
The entries are stored as key-value pairs, providing the following *Dublin Core* keys as default (definitions copied from the [Dublin core specification](#)):

- **Title:** A name given to the resource (i.e., the CBA ItemBuilder project file including one or multiple *Tasks*). Within a collection of CBA ItemBuilder items belonging to an instrument, test, or item pool, the usage should be consistent, e.g., the unit name, the item name, or even for several CBA ItemBuilder projects, the test name. If the identical title is used for multiple CBA ItemBuilder project files, the concrete items can be distinguished by the *identifier*.
- **Description:** A text-based account of the resource, describing, for instance, the measurement goals, the required competences, skills or sub-skills, etc.
- **Subject:** The topic of the items or material combined into a CBA ItemBuilder project file. If possible, a controlled vocabulary should be used.
- **Date:** A point or period of time associated with an event in the lifecycle of the CBA ItemBuilder project file (i.e., the year in which a particular assessment used the item or similar).

- *Language*: A language of the resource (i.e., the CBA ItemBuilder project file). Recommended general best practice is to use a controlled vocabulary such as RFC 4646 (Phillips and Davis, 2009)
- *Format*: The file format of the resource. For the CBA ItemBuilder it is suggested to include the URI of the CBA ItemBuilder and the version information (e.g., created with CBA ItemBuilder 9.8, <https://www.itembuilder.de>)
- *Type*: The nature or genre of the resource. For assessment content, the type could refer to the item type (e.g., multiple-choice task), the function of the component within the assessment (e.g., instruction, tutorial, cover page, etc.), or the assessment type (e.g., summative or formative assessment).
- *Identifier*: An unambiguous reference to the resource within a given context. For assessments this could be an Item-ID, or any other ID that is used to describe the assessment content stored in this CBA ItemBuilder project file.
- *Coverage*: The spatial or temporal topic of the resource, the spatial applicability of the resource, or the jurisdiction under which the resource is relevant.
- *Source*: A related resource from which the described resource is derived. The described resource may be derived from the related resource in whole or in part. Recommended best practice is to identify the related resource by means of a string conforming to a formal identification system.
- *Relation*: A related resource. Recommended best practice is to identify the related resource by means of a string conforming to a formal identification system.
- *Rights*: Information about rights held in and over the resource. Typically, rights information includes a statement about various property rights associated with the resource, including intellectual property rights.
- *Creator*: A person, an organization or a service primarily responsible for making the resources.
- *Contributor*: A person, an organization or a service that is responsible for making contributions to the resources.
- *Publisher*: An entity responsible for making the resource available. Examples of a Publisher include a person, an organization, or a service. Typically, the name of a Publisher should be used to indicate the entity.

Additional *Meta-Data* as key-value pairs can be defined by using the button *Add key* and defining the values of the user-defined *keys*.

6.4 FSM and Conditional Link Syntax Examples



6.4.1 Create Animated Instructions

In order to achieve a higher degree of standardization of assessments, e.g. if it is not ensured that all target persons can or want to read fluently, instructions are often read aloud. For this requirement the CBA ItemBuilder allows the use of components for the playback of audio files or videos (see section 3.10.3). The following example in Figure 6.12 illustrates how mp3 files are played back one after. The playback for each audio file `one.mp3`, `two.mp3` and `three.mp3` is started in a transition between two states and the transitions are triggered by either the button (`one.mp3`) or the audio *Stop Event* (see subsection 4.4.3) linked to the `Audio` component used to playback the preceding audio file.

Multiple *events* (see subsection 4.4.3) are used to create the example shown in Figure 6.12:

```
Events: EV_AudioOneFinished, EV_AudioTwoFinished,    /* Audio Stop Events */
          EV_AudioThreeFinished
          EV_Start, EV_Reset;                          /* Start and Reset */
```

The three audio stop events are used to trigger the transitions between the states *One*, *Two* and *Three*, the two additional events `EV_Start` and `EV_Reset` are used to start the instruction and to reset the illustration. Note that in real assessments the instruction would like be implemented to start automatically (using, for instance, the property `Automatic Start` of the first `Audio` component, see section 3.10.3 for details).

The remaining finite-state machine rules (simplified) are shown in the following listing:

6.4.2 Adaptivity Within Tasks using Conditional Links

Adaptive testing based on item response theory requires a scaled item pool. If assessments are not performed with the goal of IRT-based ability estimation using already known item parameters, skip rules using *conditional links* can also be used to implement explicitly specified branching.

The item in Figure 6.13 shows an example where tasks are presented following a pre-defined branching tree (see Figure 6.14) depending on the solution of previous tasks.

Adaptivity with Conditional Linking Example

This item illustrates a short sequence of 5 out of 9 tasks whose selection takes into account the tasks already solved in a fixed branching tree.

After the first three tasks, placeholders are built in, which could be replaced by instructions or motivational feedback, for example.

The last two tasks are administered without feedback and could represent a measurement of success.

Start

File: ConditionalLinkingAdaptivityExample.zip

FIGURE 6.13: Item illustrating *with-task* adaptivity using *Conditional Links* ([html](#)|[ib](#)).

In this hypothetical example, the assessment is done separately in two sections. In a first *learning* phase, additional pages are displayed after each incorrect answer, which could contain explanations or motivational feedback, for example. These additional pages are not displayed if a task has already been solved correctly. In the second *measurement* phase, no feedback provided, but the difficulty of the administered task take still previous responses into account.

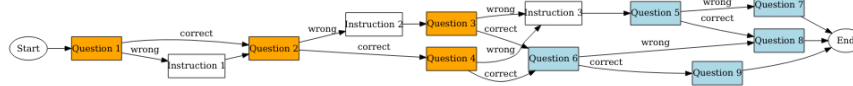




FIGURE 6.14: Adaptivity in the *with-task* adaptivity example shown in Figure 6.13.

The following tasks are used in the example, administered according to the logic shown in Figure 6.14:

- Question 1: What fraction of the square is blue?  (A: $\frac{4}{3}$, B: $\frac{5}{8}$, C: $\frac{3}{8}$, D: $\frac{7}{3}$, E*: $\frac{1}{2}$)
- Instruction 1: Some useful hint how to count the number of highlighted areas in a figure, using the plot of Question 1 as an example.
- Question 2: What fraction of the square is red?  (A: $\frac{1}{2}$, B: $\frac{2}{3}$, C: $\frac{4}{9}$, D*: $\frac{5}{9}$, E: $\frac{1}{5}$)
- Instruction 2: Some useful hint how to count the number of highlighted areas in a figure, using the plot of Question 2 as an example.
- Question 3: What is the denominator of the fraction $\frac{12}{25}$ (Short text: 25)
- Question 4: What is the numerator of the fraction $\frac{7}{8}$ (Short text: 7)
- Instruction 3: Some useful hint how the two components of a fraction are labeled (i.e., Fraction = $\frac{\text{Numerator}}{\text{Denominator}}$).
- Question 5: Express $3 \div 5$ as a fraction (A: $\frac{5}{3}$, B*: $\frac{3}{5}$, C: 15)
- Question 6: What fraction of one kilogram are 3 grams? (A: $\frac{3}{10}$, B: $\frac{3}{100}$, C*: $\frac{3}{1000}$, D: $\frac{3}{10000}$, E: 3)
- Question 7: How do we write two-thirds? (A: $\frac{3}{2}$, B*: $\frac{2}{3}$, C: $\frac{1}{3}$, D: $\frac{1}{2}$)
- Question 8: What fraction are 2 months of one year? (A: 2, B: $\frac{1}{12}$, C*: $\frac{1}{6}$, D: $\frac{2}{10}$, E: $\frac{1}{2}$)
- Question 9: What is the number that makes these fractions equal? $\frac{2}{7} = \frac{4}{12}$ (A: 4, B*: 6, C: 8, D: 3, E: 10)

In this example, forcing answers has been omitted and missing answers are treated as incorrect.

6.4.3 Hiding/Showing Components on Pages

The operator `setHidden(UserDefinedID)` can be used to hide the components with the `UserDefinedId`. Similarly, the `unsetHidden(UserDefinedId)` operator can be used to make a hidden component visible again. As can be seen in Figure 6.15, this can be used, for example, in *Conditional Links* to show or hide several components at the same time. The combination of several components is done by several `setHidden()` or `unsetHidden()` operators.

For hiding or un-hiding images, as an alternative to the `setHidden()` / `unsetHidden()` operators, a *Value Map* and a component of type `MapBasedVariableDisplay` can be used (see section 4.2). For this purpose, an empty or transparent image, for example, is assigned to a value in the *Value Map*.

Eliminate Wrong Choices Example

☐ A Text for distractor A hide

☐ B Text for distractor B hide

☐ C Text for distractor C hide

☐ D Text for distractor D hide

File: EliminateWrongChoicesExample.zip Next

FIGURE 6.15: Example item illustrating the use of `setHidden()` / `unsetHidden()` operators in *Conditional Links* ([html|ib](#)).

6.4.4 Approaches to Show Additional Content

Figure 6.16 illustrates different options to show additional information on request of test-takers. Example 1 displays additional information using a page configured as *Dialog Page*, linked to a button. The *Dialog Page* is configured as `closable: false` and positioned using the `x` and `y` coordinate of the `Frame`. The information is hidden using a button defined with the command `CLOSE` that is placed on the *Dialog Page*. Example 2 uses a `MapBasedVariableDisplay` to show either a question mark or a text depending on the value of a variable, and the value of the variable is changed using the *Finite-State Machine* and an event assigned to the `MapBasedVariableDisplay`.

Show Additional Information Examples

Example 1: Click the button to see more information (using a dialog without border).

more

Example 3: Click into the text field to see more information (using `PageAreas`).

 $3+7=$

Example 2: Click the question mark to see more information (using `MapBasedVariableDisplays`).

?

Example 4: Select a radio button to see more information, shown using `unsetHidden()`.

☐ A
☐ B

File: ShowAdditionalInformationExamples.zip

FIGURE 6.16: Example item illustrating the different ways to show additional information ([html|ib](#)).

Example 3 also uses events. The page shown in a `PageArea` is changed when the test-taker clicks into the `SingleLineInputField` using an event linked as *Raised In Event* and an empty page is shown in the `PageArea` when the *Raised Out Event* is triggered. Example 4 makes additional information visible using the `unsetHidden()`-operator, linked to the *Raised Events* of `RadioButtons`.

6.4.5 Implement Time Limits for Tasks

Limiting the available time for test parts in which test-takers may be able to navigate freely can be provided by the test delivery software. However, time limits are also doable within assessment components. In multi-page CBA ItemBuilder projects, time limits can be implemented with timed FSM events (see section 4.4.3). Figure 6.17 illustrates how a time limit can be implemented for multiple tasks. The example also uses *PageAreas* (see section 3.5.4) to implement a permanently visible outer page in which the time-limited tasks on sub-pages are embedded.

Time Limit Example

This item shows an example of a time-constrained test part. The time measurement starts as soon as the "Start" button is clicked. After that, three items can be answered, but only as long as the specified time of 60 seconds has not elapsed.

Start

Note that the timer runs continuously even when navigating between tasks.

File: TimeLimitExample.zip

FIGURE 6.17: Example item illustrating a time limit for multiple tasks ([html](#)|[ib](#)).

To understand the example in Figure 6.17, it is crucial to see that the timed event `EV_TimeLimit` only results in a timeout within the `ST_Started` state:

```
Events: EV_Start,                // Start time restricted section
        EV_TimeLimit 10,         // Timed event (10 seconds)
        EV_Task1, EV_Task2, EV_Task3; // Events for navigation

Rules: ST_Start->ST_NotStarted{true}

ST_NotStarted => ST_Started{EV_Start|setEmbeddedPage(PA,task1)}
```

```

ST_Started internal {EV_Task1|setEmbeddedPage(PA,task1)}
                  {EV_Task2|setEmbeddedPage(PA,task2)}
                  {EV_Task3|setEmbeddedPage(PA,task3)}

ST_Started => ST_Timeout{EV_Timelimit|setEmbeddedPage(PA,timeout)}

ST_Timeout => ST_NotStarted {EV_Start|setEmbeddedPage(PA,instruction)}

```

This desired behavior is achieved by defining a rule for the event `EV_Timelimit` only in the FSM state `ST_Started`.

6.4.6 Navigation Restriction

An essential goal for implementing computer-based assessments (especially for technology-enhanced items) is minimizing construct-irrelevant factors. Therefore, all possibilities for interaction with the item material must be examined and selected to determine whether they are necessary for authentic task presentation or more of a hindrance and distraction. The result of this consideration can be to enable certain functionalities only under clearly defined conditions. A typical example is the restriction of free navigation through task material or the time restriction of the availability of options. A small selection of possibilities, namely different options to restrict the navigation (i.e., the use of a button `Next`), are illustrated in Figure 6.18.

Navigation Restriction Examples

| | |
|-----------|---|
| Example 1 | Navigation is disabled until the audio file has been listened to. |
| Example 2 | A warning is shown if the task is not answered (and no navigation is possible). |
| Example 3 | A warning is shown if no text is entered (but continuing is still possible). |
| Example 4 | Navigation is not possible in the first 5 seconds. |

File: NavigationRestrictionExamples.zip

FIGURE 6.18: Example item illustrating the different navigation restrictions ([html](#)|[ib](#)).

6.4.7 Video with Built In Questions

Inspired by the *Interactive Video* format from [H5P](#), the following example shows how a video can be used split into multiple parts to combine the presentation of multimedia content and opportunities for interaction. Figure 6.19 illustrates how segments of a video can be shown, interrupted with questions.

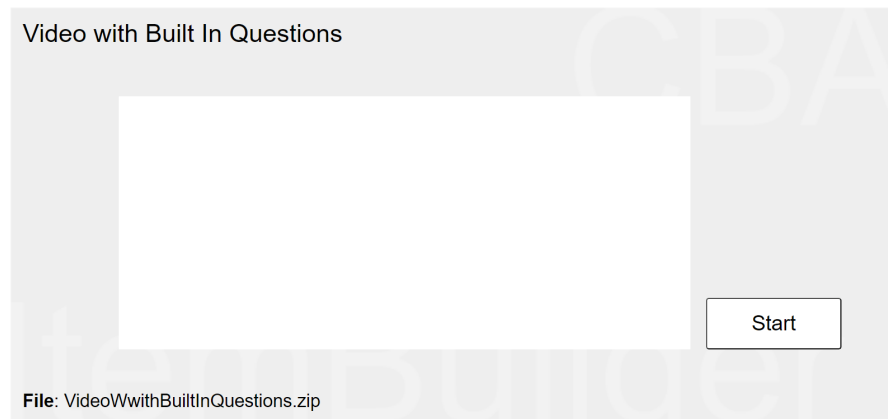


FIGURE 6.19: Example item illustrating an interrupted video with embedded questions ([html](#)|[ib](#)).

6.4.8 Click-Sensitive Labels

For `RadioButtons` (and `Checkboxes`) the label texts should be click-sensitive. That means, a click on the text next to a radio button or checkbox should select the corresponding element. This ensures that `RadioButtons` and `Checkboxes` are not only activated when the test-taker hits exactly the (small) control. As shown in the following example (A), this is the default behavior if the label text (i.e., the distractor text) is defined using the text property of the `RadioButton` (or `Checkbox`). If the text property is not used, a click on the additional component (for instance, an `HTMLTextField` or a `SingleLineInputField` with `ReadOnly=true` showing the distractor text) is not linked to the `RadioButton` (or `Checkbox`, see B). In this case, it is suggested to use the advanced features of the CBA ItemBuilder to make the components used to show the distractor text click-sensitive. This goal can be achieved with the CBA ItemBuilder using syntax operators either as part of *Conditional Links* (see section 4.3) or in transitions of a *Finite-State Machine* (see section 4.4).

Solution Using Conditional Links: A first and simple solution based on *Conditional Links* is shown in Figure 6.20. The example item illustrates the expected behavior (A) if the text property of `RadioButtons` is used. The item also shows the problem of labels that are not click-sensitive (B). To use of *Conditional Links* as additional click-sensitive

labels for `RadioButtons` (C) the labels were designed using `SingleLineInputFields` with the property `ReadOnly=true`:

Components as Labels for RadioButtons (using Conditional Links)

A (Default): Distractor text is defined as the property text of each component of type `RadioButton`.

- ☐ Option A
- ☐ Option B
- ☐ Option C
- ☐ Option D

B (Wrong): Distractor text is defined using additional components (e.g., `SingleLineInputField`) but not linked to `RadioButtons`.

- ☐ Option A
- ☐ Option B
- ☐ Option C
- ☐ Option D

C (Alternative): Labels are linked to the current page with a `Conditional Link` that contains `setActive()`-operators to select the `RadioButtons`.

- ☐ Option A
- ☐ Option B
- ☐ Option C
- ☐ Option D

File: `ComponentsAsLabelsForRadioButtonsConditionalLinksExample.zip`

FIGURE 6.20: Example for using operators in *Conditional Links* to align `SingleLineInputFields` and `RadioButtons` ([html|ib](#)).

As can be seen, when clicking on the text label for the different distractors for option C, each click on the text automatically activates the corresponding `RadioButton`. This behavior is implemented by assigning *Conditional Links* to each `SingleLineInputField`, as shown in the following example:

```
{page : true | setActive(rb1)}
```

The target page of the *Conditional Link* is the current *Simple Page* (with the name `page`). The Boolean value `true` is defined as the condition, meaning that this particular condition will always be executed. Next to the pipe symbol, the operator `setActive(UserDefinedID)` (see Appendix B.2 for details) is used to set the `RadioButton` with the *User-Defined-ID* `rb1` active (i.e., selecting the `RadioButton` and thereby, as the default behavior of components of type `RadioButton` de-selecting all other `RadioButtons` in this particular `RadioButtonGroup`).

Solution Using Finite-State Machine: A second more general solution uses the *Finite-State Machine* to align the `RadioButton` selection and clicks on the components used as labels.⁵ The example looks identical to Figure 6.20, but components of type `HTMLTextField` are used as labels. The reason is, that `HTMLTextFields` allow to assign *FSM Events* (see section 4.4.3).⁶ To implement option C with a *Finite-State Machine*, one *FSM Event* must be defined for each response option (i.e., for each `RadioButton`). These events can then be used in a particular state, to activate the `RadioButton` (or

⁵The example can be found in the file [ComponentsAsLabelsForRadioButtonsFSMExample.zip](#).

⁶Note that `HTMLTextField` could also be used with *Conditional Links* (see section 3.11.3).

Checkbox) using the `setActive`-operator (see Appendix B.2 for details), for instance, in an *internal* transition as shown in the following FSM syntax:

```
Events: EV_RB1, EV_RB2, EV_RB3, EV_RB4;           // One event for each RadioButton,
                                                    // triggered by the HTMLTextFields

Rules: Start -> Running {true}

Running internal {EV_RB1 | setActive(rb1)} // The setActive-operator is used
    {EV_RB2 | setActive(rb2)}              // to change the selected RadioButton,
    {EV_RB3 | setActive(rb3)}              // when the event that is linked to
    {EV_RB4 | setActive(rb4)}              // the HTMLTextField's is triggered.
```

Both approaches also work to create click-sensitive labels for other components such as Checkbox (see section 3.9.3). The choice between the two options is mostly a matter of taste. However, some components can only be linked with *Conditional Links* and other components only allow linking with *FSM Events*.

6.4.9 Contextualized Multiple-Choice Items

Figure 6.21 shows how components of type `ImageValueDisplays` and the *Finite-State Machine* can be used to contextualize a multiple choice answer format. In the example, the shown items shall be *striked through*.

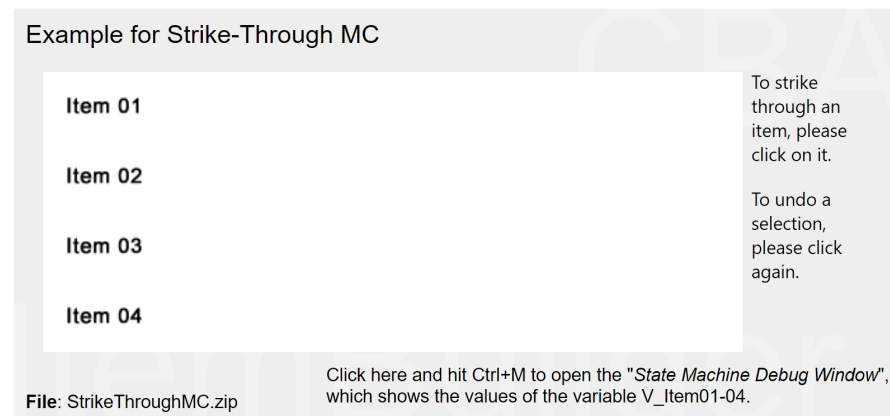


FIGURE 6.21: Example for contextualized multiple-choice with images ([html](#)|[ib](#)).

The visual design of the response format is created with `ImageValueDisplays` so that for each single choice an `ImageValueDisplay` shows either the selected or the un-selected option. Each `ImageValueDisplays` triggers one of the events `EV_Item01`, `EV_Item02`, `EV_Item03`, or `EV_Item04` and the following FSM is used to change the values of the variables:

```

Events: EV_Item01, EV_Item02, EV_Item03, EV_Item04; // Events for the options

Rules: Start -> Running {true|set(V_Item01,1), // Initialize the value of
                        set(V_Item02,1),      // variables, used to store the
                        set(V_Item03,1),      // selected state for each option
                        set(V_Item04,1)}      // (1:=deselected, 2:=selected)

Running internal
{EV_Item01 : [V_Item01 == 1] | set(V_Item01,2)} // Conditional internal
{EV_Item01 : [V_Item01 == 2] | set(V_Item01,1)} // rules for switching
{EV_Item02 : [V_Item02 == 1] | set(V_Item02,2)} // between selected and
{EV_Item02 : [V_Item02 == 2] | set(V_Item02,1)} // deselected state for
{EV_Item03 : [V_Item03 == 1] | set(V_Item03,2)} // each option.
{EV_Item03 : [V_Item03 == 2] | set(V_Item03,1)}
{EV_Item04 : [V_Item04 == 1] | set(V_Item04,2)}
{EV_Item04 : [V_Item04 == 2] | set(V_Item04,1)}

```

Use of the responses given by clicks on components of type `ImageValueDisplays` (stored internally in the variables `V_Item01`, `V_Item02`, etc.) for scoring the task is possible using the `variable_in()`-operator (see section 5.3.5).

6.4.10 Shuffle Response Options using *ValueMaps*

The CBA ItemBuilder does not explicitly support the representation of answer alternatives in random order (shuffling). However, the functionality can be implemented using `MapBasedVariableDisplays` and *ValueMaps* as shown in Figure 6.22.

Shuffle Response Options Example

Shuffle

The CBA ItemBuilder does not explicitly support the representation of answer alternatives in random order (shuffling). However, the functionality can be implemented using `MapBasedVariableDisplays`.

What is your favorite animal?

- ☐ Cat [Option 4]
- ☐ Horse [Option 3]
- ☐ Bird [Option 2]
- ☐ Dog [Option 1]

File: ShuffleResponseOptionsExample.zip

FIGURE 6.22: Example for shuffle response options with `ValueMaps` ([html](#)|[ib](#)).

In addition to `MapBasedVariableDisplays`, the example uses an invisible component

of type `ExternalPageFrame` to generate the random numbers in JavaScript. The random numbers are requested from the *Finite-State Machine* with the `callExternalPageFrame()`-operator (see section 4.6.4), calling a JavaScript code to set the variable values. Since the `ExternalPageFrame` is not yet loaded at the time of loading the item, initializing the variable values is triggered by calling the *FSM event* `EV_Shuffle` from the `ExternalPageFrame` as soon as it is loaded and ready.

6.5 Calculators Examples



The CBA ItemBuilder offers two different approaches to integrate a calculator into items. Digitalized calculators provide some advantages, e.g., the calculator can be enabled or disabled according to item-specific needs. Simple pocket calculators can be designed using visual components of the CBA ItemBuilder and directly integrated into items with the finite-state machine without any further programming knowledge. More specific and possibly more complex calculators can be integrated into CBA ItemBuilder projects via `ExternalPageFrames`.

6.5.1 Basic Calculator using Finite-State Machine

The following item in Figure 6.23 shows a basic calculator using the internal *Finite-State Machine*:

To implement the calculator using the CBA ItemBuilder *Page Editor*, components of type `Button` (see section 3.11.2) and a component of type `SimpleTextField` (see section 3.8.1) are placed on a page. The `Input Source` property of the `SimpleTextField` type component must be assigned to the `CALCULATION_ENGINE_RESULT` value. All components that have this `Input Source` are used by the CBA ItemBuilder to display the output of the calculations that are calculated by the calculator engine of the finite-state machine. If the property `Input Source Catch Focus` for this `SimpleTextField` is set to `true`, the keyboard input is automatically collected by the `SimpleTextField` and the calculator can be used using the keyboard. An optional second `SimpleTextField` can be used

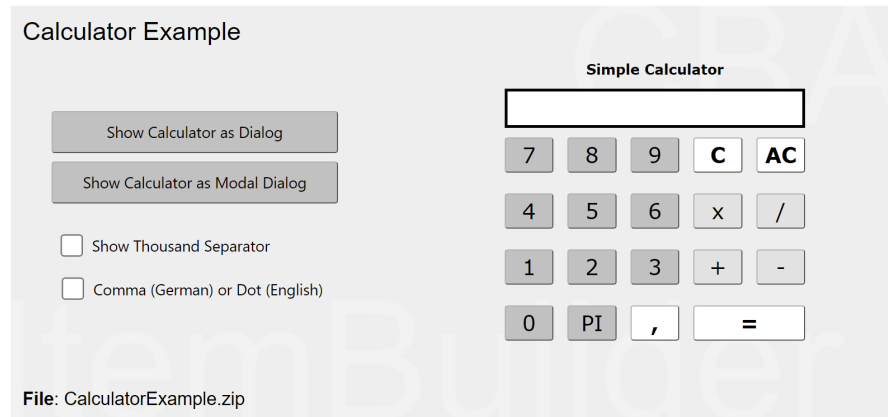


FIGURE 6.23: Example for a calculator based on 'Finite-State Machine'.

to show the calculator engine's stack (using the value `CALCULATION_ENGINE_OP_STACK` for the property `Input Source`).

The buttons must be assigned to *Events* so that in *Finite-State Machine Rules* the input can be passed to the calculator engine.⁷ An example FSM is shown below:

```
Events: /* One event is defined for each button: */
calc0, calc1, calc2, calc3, calc4, calc5, calc6, calc7, calc8, calc9,
calcPI, calcDecimal, calcMult, calcDiv, calcAdd, calcSub, calcRes,
calcC, calcAC;

Rules: /* Calculator settings are defined in very first transition */
Start -> Process{true | calcSettings(displayWidth => 12, scale => 24)}
/* Calculator operators are processed as internal rules */
Process internal { calc0 | calcOpnd(add, 0) }
Process internal { calc1 | calcOpnd(add, 1) }
Process internal { calc2 | calcOpnd(add, 2) }
Process internal { calc3 | calcOpnd(add, 3) }
Process internal { calc4 | calcOpnd(add, 4) }
Process internal { calc5 | calcOpnd(add, 5) }
Process internal { calc6 | calcOpnd(add, 6) }
Process internal { calc7 | calcOpnd(add, 7) }
Process internal { calc8 | calcOpnd(add, 8) }
Process internal { calc9 | calcOpnd(add, 9) }
Process internal { calcPI | calcOp(clear), calcOpnd(add, 3),
```

⁷Instead of buttons other components can be used for input if they can be assigned to an event (e.g. `ImageValueMaps`).

```

    calcOpnd(decimal), calcOpnd(add, 1),
    calcOpnd(add, 4), calcOpnd(add, 1),
    calcOpnd(add, 5) }
Process internal { calcDecimal | calcOpnd(decimal) }
Process internal { calcMult | calcOp(multiply) }
Process internal { calcDiv | calcOp(divide) }
Process internal { calcAdd | calcOp(add) }
Process internal { calcSub | calcOp(subtract) }
Process internal { calcRes | calcOp(equals) }
Process internal { calcC | calcOp(clear) }
Process internal { calcAC | calcOp(clearall) }

```

The actual processing of test taker input is done by the special operators for the calculator engine (see section 4.4.6). The user interfaces created as part of the CBA ItemBuilder *Project File*'s content can be customized and adapted by item authors to the needs of items and tasks by placing, adding, and omitting buttons.

6.5.2 Embedding Calculator using ExternalPageFrame

An example for an external calculator that can be embedded using the ExternalPageFrames (see section 3.14) is shown in Figure 6.24.⁸

Note that if logging of calculator actions is required, it can be implemented in JavaScript using the API for trace events (see section 4.6.2).

6.6 ExternalPageFrame Examples



The functionality of the CBA ItemBuilder does not limit the possibilities for design-

⁸The source can be found at github in the repository <https://github.com/DIPFtba/calculator>

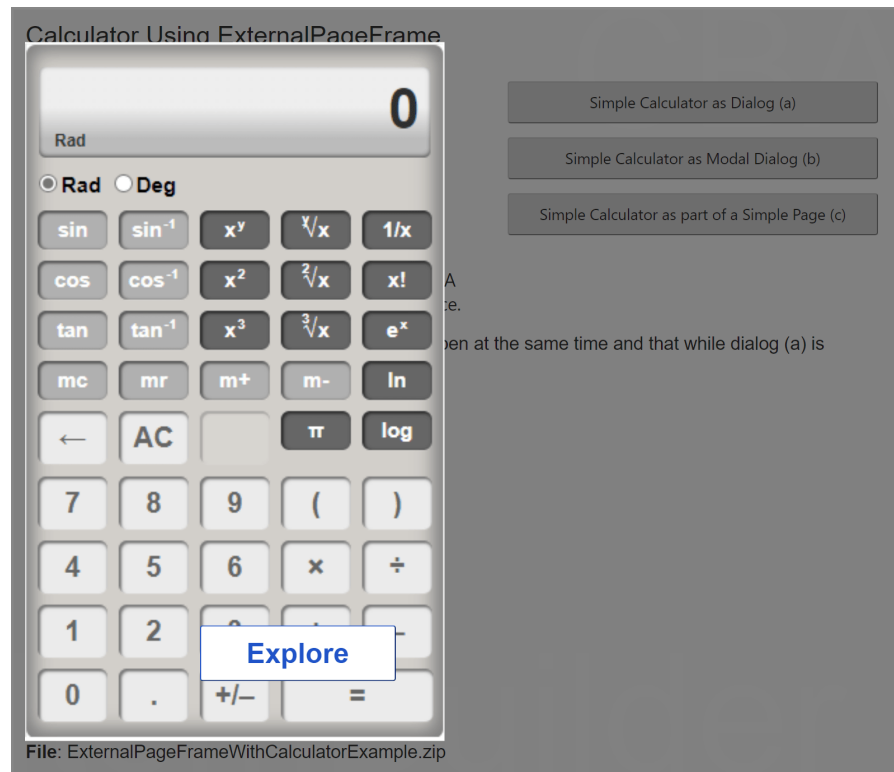


FIGURE 6.24: Example for `ExternalPageFrame` with calculator. ([html](#)|[ib](#)).

ing *Technology-Enhanced Items*. Instead, the CBA `ItemBuilder` can be understood as a platform where recurring elements are made available as components for item authors while new and innovative components provided by software developers can be integrated using the `ExternalPageFrames` (see section 4.6.2). In this section, some examples are shown to illustrate the possibilities. However, it is by no means a complete overview.

6.6.1 Continuous Sound using Buttons in `ExternalPageFrame`

The CBA `ItemBuilder` can play audio files. But it may not be able to repeat audio files continuously, as is necessary for playing notes on a (digital) instrument. Just because such a function is missing, the use of the CBA `ItemBuilder` does not have to be completely questioned. With the help of JavaScript and `ExternalPageFrames` extensions can be easily programmed. Figure 6.25 illustrates how such a missing functionality can be added using `ExternalPageFrames`.

In this project file, each of the three buttons is implemented separate `ExternalPage-`

Simple Example Showing the use of an ExternalPageFrame to Play Audio

With the three buttons below, you can create melodies. Please play one of the different tones by pressing and holding one of the buttons at a time!

Here you can play the three different tones:



File: ExternalPageFrameButtonExample.zip

FIGURE 6.25: Continuous Audio Play with ExternalPageFrames ([html](#)|[ib](#)).

Frame to keep the syntax as minimal as possible. The respective HTML files contain JavaScript code to play a wav file and a standard HTML button.

6.6.2 Alternative Editors as ExternalPageFrame

The CBA ItemBuilder does not natively support the input of mathematical formulas. The following example in Figure 6.26 illustrates the use of the [mathlive](#) library. *Mathlive* provides a virtual keyboard that also allows formula input on touch devices. The library can be included and used via an ExternalPageFrame component.

In the same way as a specific editor for mathematical input, a text editor can also be included as an ExternalPageFrame, as shown in Figure 6.27.

Displaying the number of words already written (or the number of words remaining, if specified) and, for example, detailed JavaScript-based logging of keystrokes is also possible with an ExternalPageFrame, as illustrated in Figure 6.28.

6.6.3 Adding Speech Recognition using ExternalPageFrame

Speech recognition functions can also be embedded in ExternalPageFrames. A generic variant that should work in many browsers and delivery options is illustrated in Figure 6.29.

Offline: The following item shows an offline capable speech recognition. For this, a large language model must be loaded before spoken language can then be translated into characters without transferring the data to the server. For this example, the model of one language is loaded, i.e. the speech recognition in the CBA ItemBuilder item in Figure 6.29 recognizes only German speech.

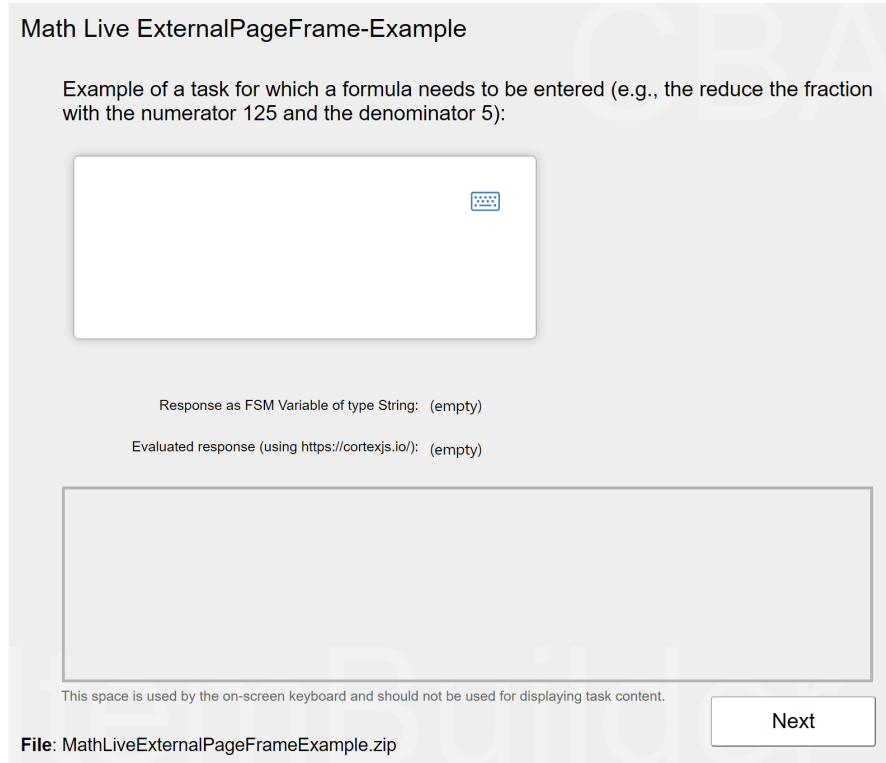


FIGURE 6.26: Using `math-field` (from *Mathlive*) within `ExternalPageFrames` ([html](#)|[ib](#)).

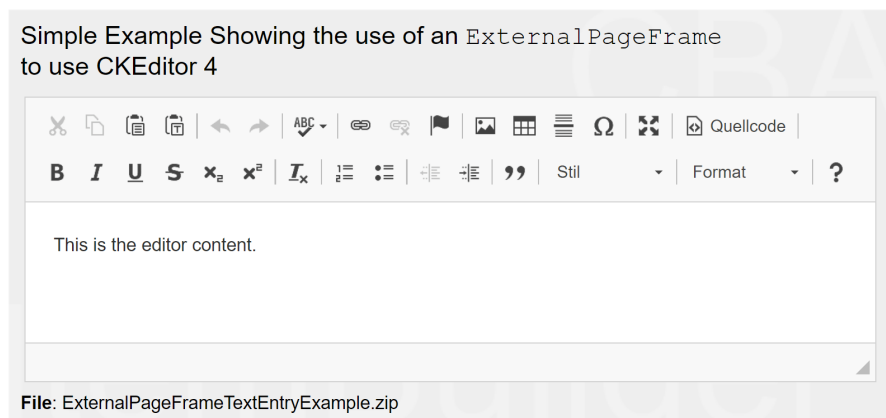


FIGURE 6.27: CKEditor within `ExternalPageFrames` ([html](#)|[ib](#)).

Word Counter ExternalPageFrame - Example

The following example shows a TextArea that also records single keystrokes as log-events:

Total word count: 0 words.

File: WordCounterExternalPageFrameExample.zip

FIGURE 6.28: TextArea within ExternalPageFrames ([html](#)|[ib](#)).

Speech Recognition using Vosk in an ExternalPageFrame

Press the button "Start Recognition (Language=German)" and give the browser access to your computers microphone.

Start Recognition (Language = GEMAN)

File: ExternalPageFrameVoskExample.zip

<https://alphacephei.com/vosk/>

FIGURE 6.29: Browser-based Speech Recognition using Vosk and ExternalPageFrames ([html](#)|[ib](#)).

Online: As an alternative to client-side recognition, some browsers also provide access to server-side speech recognition. The following example in Figure 6.30 will only work in the Chrome browser and requires an internet connection:

6.6.4 Showing HTML5 Package (H5P) using ExternalPageFrame

Using the MIT licensed [H5P standalone player](#) it is possible to embed [HTML5 Package \(H5P\)](#) into CBA ItemBuilder projects as interactive content. The following example shows how to display H5P content without using a web server inside the



FIGURE 6.30: Server-based Speech Recognition using `window.SpeechRecognition` and `ExternalPageFrames` ([html|ib](#)).

Example using an `ExternalPageFrame` and `window.SpeechRecognition` (Chrome) / `window.webkitSpeechRecognition` (Safari)

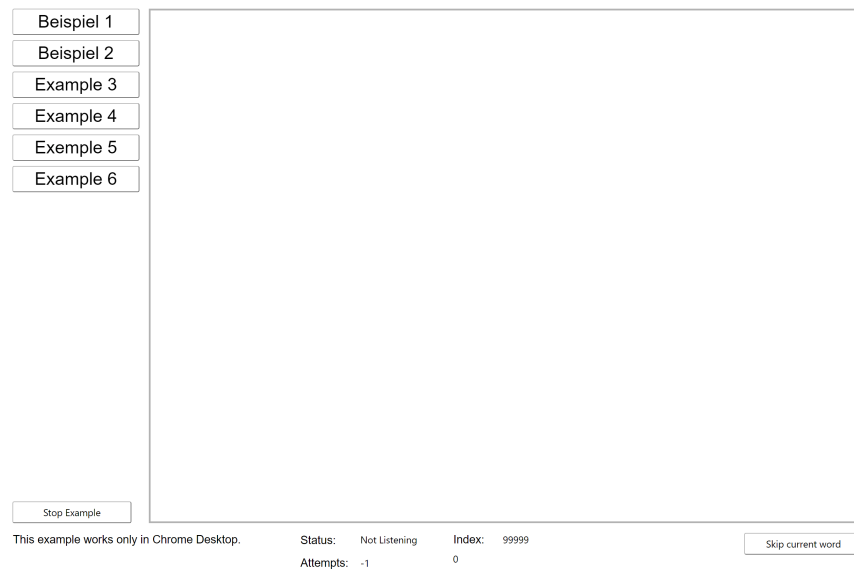


FIGURE 6.31: Reading Task using Server-based Speech Recognition and `ExternalPageFrames` ([html|ib](#)).

CBA ItemBuilder using `ExternalPageFrames` (for more details, open [ExternalPageFrameH5PIntegrationExample.zip](#) in the CBA ItemBuilder):

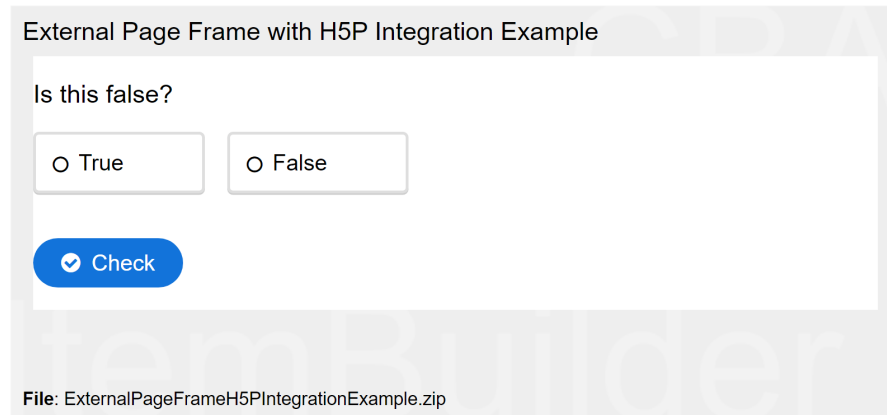


FIGURE 6.32: Example for the integration of H5P content in an ‘ExternalPageFrame’.

Note: An integration of data storage and xAPI needs to be added.

6.6.5 Including GeoGebra Applets using `ExternalPageFrame`

Using `ExternalPageFrames` it is also possible to integrate other interactive content such as GeoGebra applets into CBA ItemBuilder tasks (see figure 6.33, for more details, open [ExternalPageFrameWithGeoGebra.zip](#) in the CBA ItemBuilder).

6.6.6 Including QTI Item Content using `ExternalPageFrame`

With the help of JavaScript (client-side) rendering of content in the [IMS Question & Test Interoperability \(QTI\) format](#), using, for instance the MIT licensed [QTI.js](#) library, QTI can be used within CBA ItemBuilder items (see Figure 6.34⁹).

6.6.7 Including SurveyJS Questionnaires using `ExternalPageFrame`

The creation of long surveys with the CBA ItemBuilder is possible but can be more complex than known from survey tools. An easy way to integrate surveys in all delivery modes supported by CBA ItemBuilder (online, offline) is offered by the [SurveyJS](#) library.

⁹The example does not include data storage, that can be added, for instance, using `postMessages` to send data to the CBA ItemBuilder trace data

CBA ItemBuilder Item with Embedded GeoGebra Content

(using `ExternalPageFrame` that includes the GeoGebra applet with additional JavaScript to collect data and uses the GeoGebra runtime from <https://cdn.geogebra.org/apps/deployggb.js>. Note that downloading the GeoGebra applet as "Offline Activity" requires to agree to the terms of GeoGebra's non-commercial license.)

Info

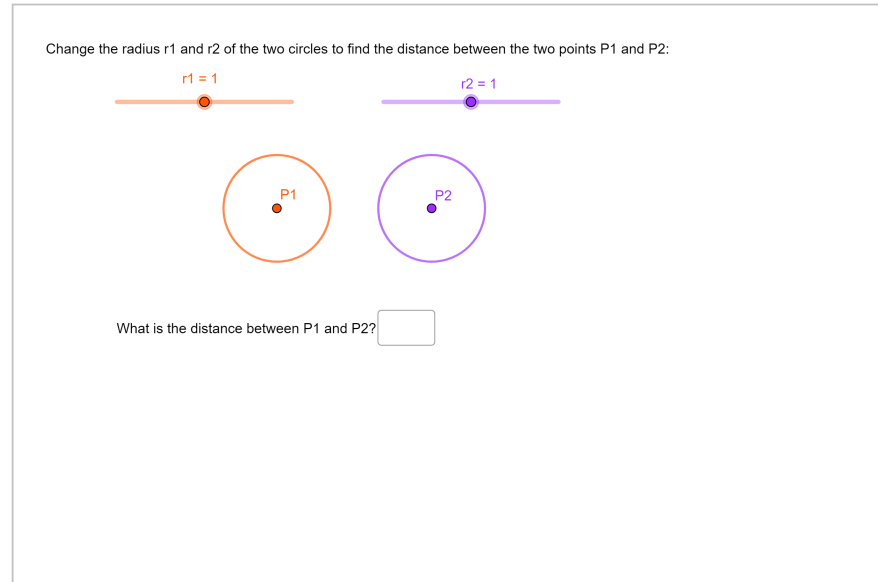


FIGURE 6.33: Integration of GeoGebra content in a `ExternalPageFrame` ([html|ib](#)).

As shown in the following screenshot (see Figure 6.36), the runtime environment of SurveyJS can be inserted into the *Embedded HTML Explorer*. The JSON configuration of the survey can be stored in the file `index.html`, which is used as the *Local* page address in the `ExternalPageFrame`. This file is part of the CBA ItemBuilder *project files* and is available at runtime.

The JSON configuration can be assigned, for instance, to a JavaScript variable `surveyJSON`:

```
var surveyJSON = <!-- COPY JSON OR JSON STRING HERE--> ;

function sendDataToServer(survey) {
  postLogEvent("SurveyJS results changed to: " + JSON.stringify(survey.data));
  postFsmEvent('EV_NextTask');
}

function doOnCurrentPageChanged(survey) {
  postLogEvent("Page changed to: " + survey.currentPageNo);
}
```

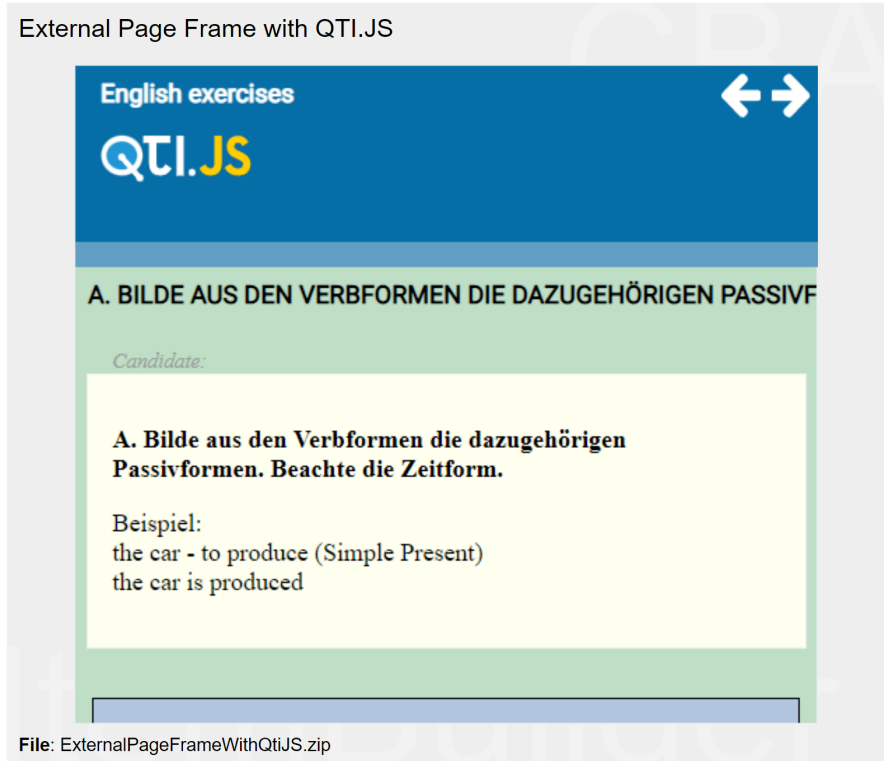


FIGURE 6.34: Integration of QTI content using QTI.js and ExternalPageFrames ([html](#)|[ib](#)).

```
function doOnValueChanged (sender, options) {
    postLogEvent("Answer changed to: " + options.name + " ; " + JSON.stringify(options.value));
};

var survey = new Survey.Model(surveyJSON);
$("#surveyContainer").Survey({model: survey, onComplete: sendDataToServer});
$("#surveyContainer").Survey({model: survey, onCurrentPageChanged: doOnCurrentPageChanged});
$("#surveyContainer").Survey({model: survey, onValueChanged: doOnValueChanged});
```

Two further details of the integration are worth mentioning (see [ExternalPageFrameWithSurveyJS.zip](#)):

- In order that the administration of the CBA ItemBuilder tasks can be continued after the completion of the survey with the integrated SurveyJS, a function is bound to the `onComplete` event of SurveyJS. In this function `sendDataToServer` the answers

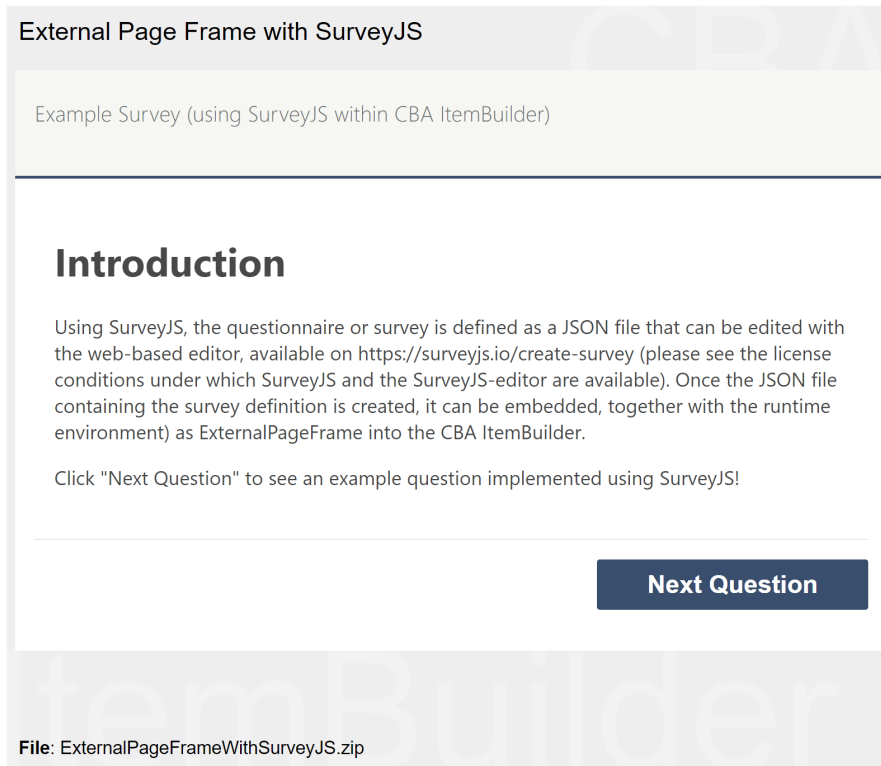


FIGURE 6.35: Integration of Questionnaires using SurveyJS and ExternalPageFrames ([html|ib](#)).

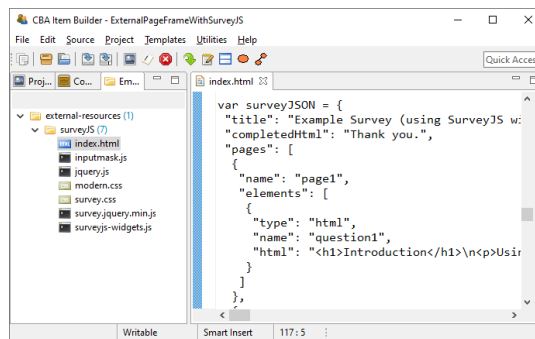


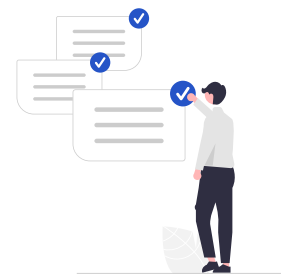
FIGURE 6.36: Screenshot of the Embedded HTML Explorer using SurveyJS content.

are passed as JSON String into the log data of the ItemBuilder delivery. After storing the answers, an FSM-event is triggered in order to call, for instance, the `NEXT_TASK`-command.

- In addition, the `onCurrentPageChanged` and `onValueChanged` events are also used by SurveyJS. With their help, the collection of log events is possible that indicate the loading of a page (`onCurrentPageChanged`), and that identifies a change in response. These two pieces of information are necessary in order to extract response times at item-level for detailed analyses (Kroehne and Goldhammer, 2018).

Note that a deeper integration of the data storage might be developed in future.

6.7 Adaptive Testing with the CBA ItemBuilder



6.7.1 Adaptive Testing *within* CBA ItemBuilder Tasks

Simple forms of adaptivity, e.g. multi-stage tests with routing based on observed scores, can be implemented directly in the CBA ItemBuilder. For this purpose, within a task (i.e. also within a CBA ItemBuilder project) the selection of pages on which items are displayed can be controlled, for example, via the finite state machine.



Adaptivity within CBA ItemBuilder tasks can be implemented using conditional links and finite-state machines, but is limited to fixed branching or adaptivity based on raw-scores.

Inspired by the number sequence test as used in the National Cohort (see Schmiedek et al., 2022), the item in Figure 6.37 illustrates a simple multi-stage test with 8 items.

Number Series Test

In the tasks you are about to work on, you will see rows of numbers, which are interrupted or terminated by blanks. The rows of numbers are based on certain rules. You have to find out these rules and enter the correct number(s) in the blank space. Such a series of numbers could look like this:



The correct answer here would be 4 and you would therefore put a 4 in the blank. Of course, this is a very simple example and the number sequences in the task sheets will not be that simple and may differ for different number sequences.

Nevertheless, please try to solve all the tasks. If you cannot solve a number line, please still enter an answer by simply guessing. The point is not to be as fast as possible, but to complete as many number sequences correctly as possible.

Now please press the "Next" button.



FIGURE 6.37: Example adaptive number series test ([html|ib](#)).

Each person should be shown only a selection of items. For this purpose, two items are administered in a first stage (A). Based on the number of correct answers in stage A (0 items correct → B0, 1 item correct → B1, 2 items correct → B2), a suitable second stage is then administered. The test is implemented with the CBA ItemBuilder in such a way that it can be operated without keyboard exclusively with buttons, which can be operated by mouse or touch screen. The routing is completely implemented in the Finite State Machine, as is the scoring of the input.

The number series used here in the example do not correspond to the instrument used in NAKO but are example number series as summarized in the following table:¹⁰

| Stage | Item | Number Series | Corret Response |
|-------|------|---------------------|---|
| A | 1 | 5, 6, 9, 14, 21, __ | 30 (+1,+3,+5,+7,+9; The series consists of a pattern of addition of consecutive odd numbers.) |

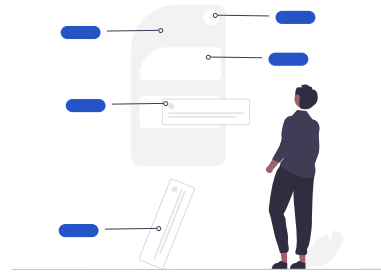
¹⁰Source: <https://learnfrenzy.com/reasoning/logical-reasoning/number-series>

| Stage | Item | Number Series | Corret Response |
|-------|------|--------------------------------------|---|
| B1 | 2 | 1, 3, 9, 27, __, 243 | 81 (*3, *3, *3, *3, *3; Next number of this series is multiplication of 3 with previous one.) |
| | 3 | 1, 3, 6, 11, 18, __ | 29 (+2,+3,+5,+7,+11; The series consists of a pattern of addition of prime numbers.) |
| | 4 | 1, 2, 6, 15, 31, __ | 56 (+1 ² , +2 ² , +3 ² , +4 ² ; The series is based on addition of squares of consecutive natural numbers.) |
| B2 | 5 | 32, 19, 8, __ | 1 (-13, -11, -7; The series consists of a pattern of subtraction of prime numbers.) |
| | 6 | 225, 100, 36, 9, 1, __ | 0 (-5 ³ , -4 ³ , -3 ³ , -2 ³ , -1 ³ ; The series is based on subtraction of cubes of consecutive natural numbers.) |
| B3 | 7 | 1296, 648, 216, 108, __, 18, 6, 3 | 35 (/2, /3, /2, /3, /2, /3, /2; The terms are divided by 2 and 3 alternately.) |
| | 8 | 71, 55, 46, 42, __ | 41 (-4 ² , -3 ² , -2 ² , -1 ² ; The series is based on addition of squares of consecutive natural numbers.) |

6.7.2 Adaptive Testing *across* CBA ItemBuilder Tasks

Adaptivity across CBA ItemBuilder tasks must be supported by the deployment software. The R package `ShinyItemBuilder` provides a simple example to implement an adaptive test using CBA ItemBuilder tasks and R/Shiny (see 7.3.3).

6.8 (More) Efficient use CBA ItemBuilder



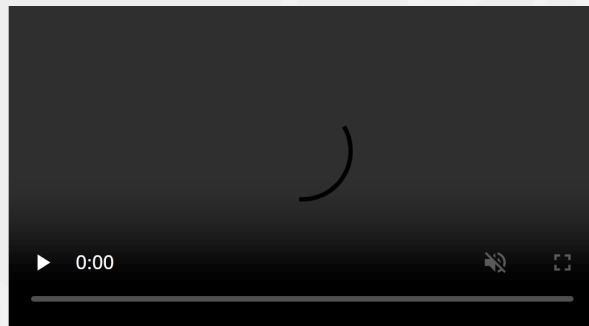
The CBA ItemBuilder is a complex tool for designing interactive assessment components. The features described below and hints on how to use the CBA ItemBuilder can help to work efficiently with this tool.

6.8.1 Window Management

Section 3.1 describes the user interface of the CBA ItemBuilder in the default configuration (i.e., after a fresh installation). The arrangement of views in the three columns of the CBA Item Builder's user interface can be adjusted and configured according to the user's needs (and the available space on the screen). The video included in the item in Figure 6.38 shows how to customize the user interface of the CBA ItemBuilder.

Window Management Video

The arrangement of views in the three columns of the CBA ItemBuilder's user interface can be adjusted and configured according to the user's needs (and the available space on the screen).



File: WindowManagementVideo.zip

FIGURE 6.38: Item illustrating Window Management in CBA ItemBuilder (html|ib).

6.8.2 Available Fonts and Font-List

Assessment content created with CBA ItemBuilder is used in a web browser at run-time. Accordingly, only fonts available for display in the web browser should be used.

Web Safe Fonts: Fonts that should be available in all browsers and result in similar renderings are called *Web Safe Fonts*. The following fonts are typically considered to be web safe:

- Arial (sans-serif)
- Verdana (sans-serif)
- Tahoma (sans-serif)
- Trebuchet MS (sans-serif)
- Times New Roman (serif)
- Georgia (serif)
- Garamond (serif)
- Courier New (monospace)
- Brush Script MT (cursive)



Remark: The behavior of the CBA ItemBuilder regarding fonts (still) described below will change in the next version (10.0). CBA ItemBuilder will support web fonts, that can be added to CBA ItemBuilder project files as ressources.

Font List: The CBA ItemBuilder uses the fonts registered on the system and therefore offers quite many fonts (e.g., in the editors for `TextFields`, `HTMLTextFields` and in the *Appearance* tab of the *Properties* view). This (usually) long font list should be reduced to the fonts that are planned to be used (i.e., *Web Safe Fonts* and similar fonts across several items). Since the item contents created with the CBA ItemBuilder are rendered in the browser, it must also be taken into account when selecting the fonts that these must then be available on the target system.



If unique fonts (not web-safe fonts) are used, which are available on the computer on which the CBA ItemBuilder is executed, then these fonts are also available in the *Preview*. This does not mean, however, that the fonts can be displayed identically in operational assessments (because this requires the fonts to be available on the device on which the items are answered).

To reduce the list of fonts available in the CBA ItemBuilder to design items, the list of *CBA Item Fonts* can be configured (see Figure 6.39) in the section *CBA Item Fonts* in the *Preferences* (menu *Utilities* > *Preference*).

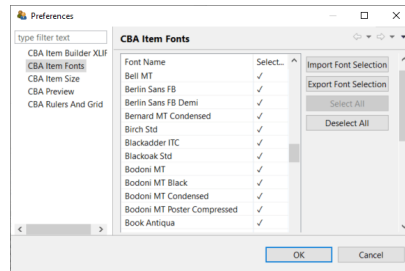


FIGURE 6.39: 'CBA Item Fonts' setting of the CBA ItemBuilder.

To adjust the list first remove the selection in this list using the `Deselect All` button and then select the fonts you want to use by clicking on them. You can also export (`Export Font Selection`) and import (`Import Font Selection`) fonts if you want to use different fonts for different projects or assessments.

6.8.3 Color Codes

The same selection dialog is always used to define color in different editors of the CBA ItemBuilder. Colors can be chosen from a selection of example colors or defined precisely in different formats (HSV, HSL, RGB, and CMYK, see Figure 6.40).

Save Selected Colors: To avoid entering identical color values multiple times, the dialog shown in Figure 6.40 also allows to *Add* a color to the user-defined buttons of *Recent* colors.

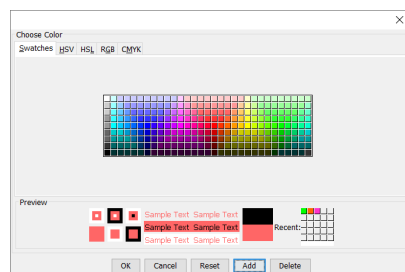


FIGURE 6.40: Color selector (*Choose Color*) of the CBA ItemBuilder allows storing *Recent*-colors.

Colors in the Properties View: The CBA ItemBuilder uses a color representation in the tab *Core* of the *Properties* view as decimal numbers (e.g., white is represented by DEC -1, corresponding to hex #FFFFFF, black is represented by DEC -16777216, corresponding to hex #000000). The item shown in Figure 6.41 contains via an `ExternalPageFrame` an example of JavaScript functions to convert the decimal colors of the CBA ItemBuilder into other formats.

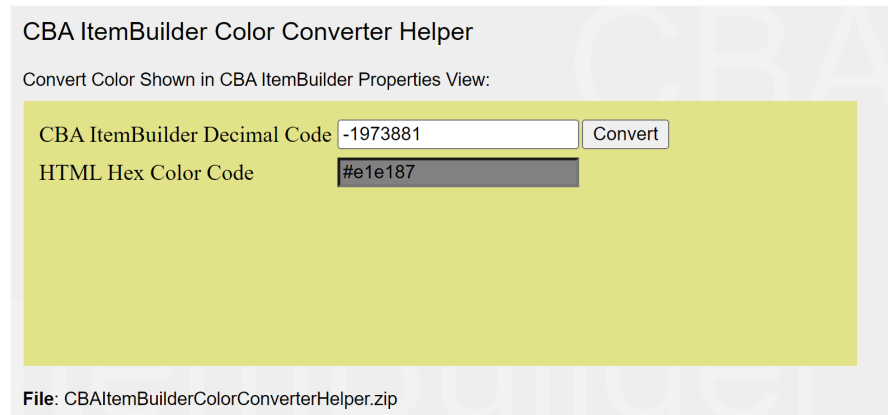


FIGURE 6.41: Item for Color Conversion of CBA ItemBuilder's internal Color Representation ([html|ib](#)).

6.8.4 Error Messages

When using the CBA ItemBuilder, especially when calling the *Preview* and saving *Project Files*, error messages are displayed if the created assessment content is inconsistent. While this might be confusing when using the CBA ItemBuilder for the first time, the error messages typically give precise information on where to find the inconsistency. The item shown in Figure 6.42 allows searching for error messages and provides additional tips on fixing the issues.

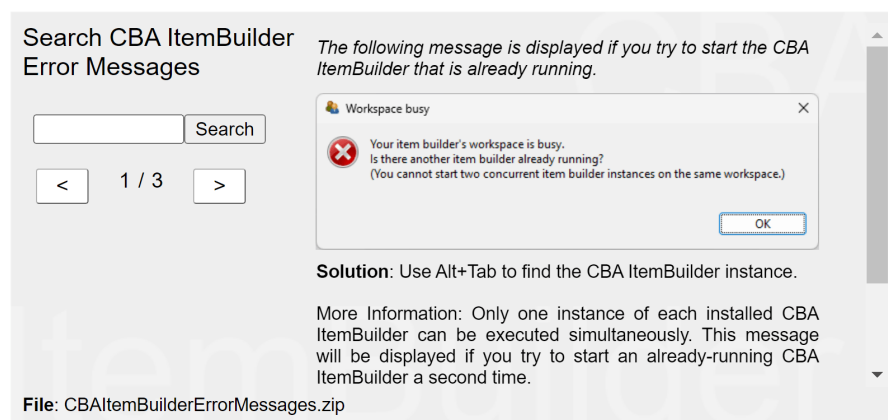


FIGURE 6.42: CBA ItemBuilder Error Messages ([html|ib](#)).

6.8.5 Hints for Designing Interactive Items



Remark: The behavior of the CBA ItemBuilder regarding the implementation of the *z-order* (still) described below will change in the next version (10.0).

The CBA ItemBuilder gives item authors a wide range of freedom in designing assessment components. Some hints are compiled below to ensure that the resulting items work well (and look good, if possible).

Visual Overlapping Components: The CBA ItemBuilder allows to place elements freely using *x* and *y* coordinates within containers (i.e., Frames, Panels etc., see *Page Editor* in section 3.1.3), and each component has a size defined as *Width* and *Height*. The item authors' responsibility is to ensure that the components created and designed in the *Page Editor* are displayed appropriately (i.e., in a meaningful order).



Overlapping Components should be avoided if possible as they can affect the behavior of items when foreground elements prevent the processing of events, links and commands. The CBA ItemBuilder uses a fixed *Z-Order* that takes into account the nesting (i.e., containers with child elements) and the type of components (grouped into layers). Only within these layer the *Z-Order* corresponds to the order components are generated and listed in the *Component Edit* view.

Z-Order: If multiple components are defined on a page, they are displayed in a specific order. This order can be called the *Z-Order* (i.e., besides *X* and *Y* for horizontal and vertical placement, the *Z-Order* defines the order concerning a third dimension, *Z*). In most use cases, item authors do not need to worry about the *Z-Order* any further since it essentially follows from the container principle (see section 2.11.4): The elements within a container (child elements) are always rendered on top of the containers itself (parent elements). The CBA ItemBuilder automatically applies this rule, together with an additional grouping of components within containers into meaningful layers.

The layers group components within containers by component type. Only for components in the same layer the order in which components are listed in the *Component Edit* view (i.e., the order in which components are created).¹¹

Functioning Affected by Overlapping Components: Although the CBA ItemBuilder approach of avoiding user-defined *Z-Orders* by applying heuristics based on layers was proven to be useful over the last years, item authors should be aware of the issue that overlapping components can result in a particular functioning. If, for instance,

¹¹Note that in the current version of the CBA ItemBuilder, changing the order of components is only possible by creating the components in the required order.

an `HTMLTextField` is placed so that it entirely or partially covers a `Button` (see Figure 6.43 for an example), the functioning of the `Button` might result in unexpected behavior. The same is true for various other components. If something overlaps with `SingleLineInputFields`, it can become difficult or even impossible to enter any text into the `SingleLineInputField`. Therefore, when designing assessment components, ensure that the components do not overlap or only overlap sensibly.

The difference between the *Visual Z-Order* and the *Interactions Z-Order* can be seen in Figure 6.43.

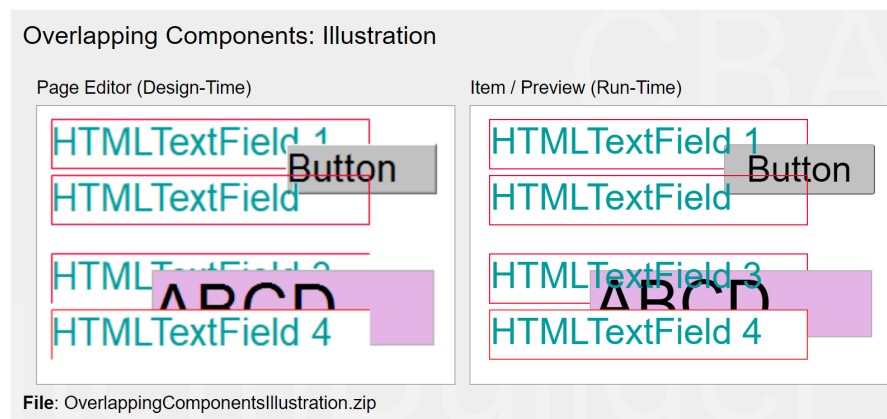


FIGURE 6.43: Item illustrating overlapping components in CBA ItemBuilder ([html|ib](#)).

The left side in Figure 6.43 shows a screenshot of the design time from the Page Editor. `HTMLTextField 1` is shown below the button, overlaid by `HTMLTextField 2`. In the *Preview* and at *Runtime*, as shown on the right, the CBA ItemBuilder follows this structure regarding the filled background (white color) and the border (red color). But note that the text in `HTMLTextField 1` is displayed above the `Button`. An identical rendering is also created by the CBA ItemBuilder for `HTMLTextField 3`, i.e. similarly the text overlays the `SingleLineInputField`, although the filled white background and the red border are rendered behind the `SingleLineInputField`.

Exploring the behavior (i.e. the *Interactions Z-Order*) in the item shown in Figure 6.43, we see that not only the text `HTMLTextField 1` and `HTMLTextField 3` overlay the underlying components `Button` and `SingleLineInputField`. Also, due to the overlapping of the components, the connected events are forwarded to `Button` and `SingleLineInputField` only at the points where no `HTMLTextField` is defined.

Scrollbars: When collecting diagnostic information (i.e., when assessment content is created with the CBA ItemBuilder), the necessity to scroll should be used as controlled as possible. Following this principle, assessment content created with the CBA ItemBuilder is designed for screen sizes (i.e., for an aspect ratio) called *CBA Presentation Size* (see section 3.2.2).

Scrollbars will automatically appear when pages are larger (in width and height) than the available space.¹² The available space for simple pages (i.e., pages at the highest hierarchy level) is identical to the *CBA Presentation Size*, minus the space of an optionally configured *X-Page* (see section 3.4.2). Similarly, pages within *PageAreas* (see section 3.5.4) or *TreeViewArea* (see section 3.9.9) and *Web Child Pages* within a *WebChildArea* (see section 3.13.2) are always displayed with scrollbars if the width or height of the embedded exceeds the width or height of the corresponding area.



If unintended scrollbars appear in the preview, it is suggested to remove them by making sure that no page is larger than the available space. The size of a page is defined as the `Width` and `Height` properties of the `Frame`.

Un-proportional Scaling of Images and Videos: Images and videos added as resources to CBA ItemBuilder projects have a native resolution, i.e., a size defined in pixels (height and width). For optimal display, images and videos should be added to CBA ItemBuilder items in the size they will be displayed by the corresponding component (see section X). The *Page Editor* allows resizing components to display image and video resources.



Avoid scaling images non-proportionally (i.e., changing width and height independently of each other and not maintaining the aspect ratio)!

If images or videos are larger than they will be used during display, the resources will be unnecessarily large and may produce load times that could be avoided by scaling the resources before insertion. If images or videos are smaller than the area used for display, they are scaled up and may only be visibly blurred.

Standardize Page Layout Across Pages: Test takers typically see many individual pages during the processing of assessments created with the CBA ItemBuilder. It is recommended to standardize spacing, font, font size, formatting (font color, bold, italics), etc., as much as possible to ensure a consistent appearance for the test takers.

6.8.6 Export and import Pages

In order to use once-designed pages multiple times when designing multiple assessment components with the CBA ItemBuilder, multiple tasks can be defined within one project file (see *Task Editor* in section 3.1.4 and tips on the division of content to

¹²The size of pages is defined using the `Width` and `Height` properties of the `Frame`

tasks in section 3.6.3). This approach is recommended if feasible, as it eliminates the need to make copies of pages.



Using CBA ItemBuilder project files with multiple tasks is recommended to avoid duplicating pages.

If there are already designed and tested CBA ItemBuilder assessment components that already contain the required functionality, it can also be efficient to use them as templates. The CBA ItemBuilder provides the *Save As* feature, and the project name can be changed using the *Rename Project* function (see section 3.2.1). If this approach is chosen, potential changes that are made to the source file afterward must also be applied to the copy.

Export Page: If instead of an entire project file only a single page is to be moved or copied from one project to another, CBA ItemBuilder provides the option to export pages in the context menu (right-click on the project name) of the *Project Tree* (see Figure 6.44).

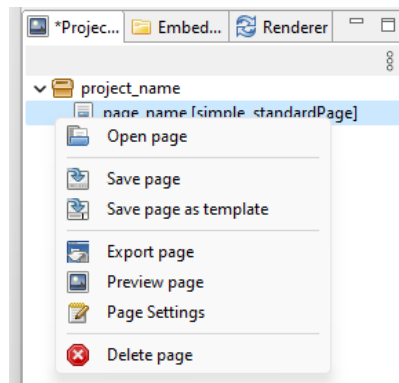


FIGURE 6.44: Context Menu in the *Project Tree*.

One page at a time can be exported and saved as a ZIP archive.

Import Page: Using the similar function *Import Page*, pages can be inserted into another CBA ItemBuilder *Project File*.

6.8.7 Working with Page-Templates

If pages are exported via the *Export Pages* function as in the past section 6.8.6, they are available as ZIP archives in the file system and can be copied, moved, and (for instance) sent as files or uploaded to versioning systems (see section 8.3.2).

If pages are to be used as templates on a computer (i.e., within one instance of the CBA ItemBuilder), the CBA ItemBuilder additionally provides the template functionality.

Save Page as Template: Using the context menu (right-click on the project name) of the *Project Tree* (see Figure 6.44, pages can also be stored as *Template*. *Templates* need a unique name (see Figure 6.45) and are stored inside of the CBA ItemBuilder.

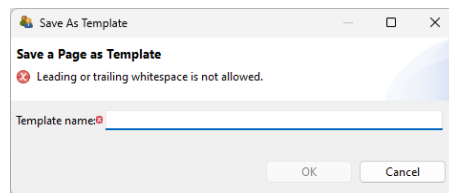



FIGURE 6.45: Dialog to specify the name of a *Template*.

Create Page from Template: Pages saved as templates can then be used to create new pages within a CBA ItemBuilder *Project File*. The toolbar contains the icon  for this purpose, and the function is also accessible via the context menu in the *Project View*.

Template Browser: Selecting and managing page templates is done in the *Template Browser* (see Figure 6.46). After entering a name for the new page to be created, a template can be used with *Create Page*. If *Skip Preview* is not selected, then the view from the *Page Editor* is displayed in the *Template Preview* area. Figure 6.46 shows that templates (analogous to pages) can also be exported or imported. The *Delete Template* button can be used to delete a template from the CBA ItemBuilder instance.

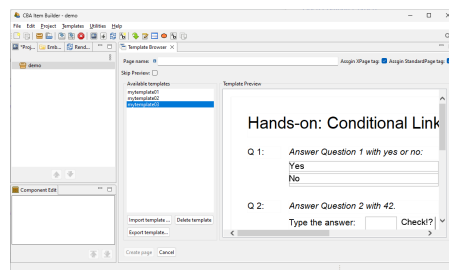


FIGURE 6.46: Dialog to specify the name of a *Template*.

When pages are created from templates, they can be flagged with the *XPage* and *StandardPage* tags.

6.8.8 How to run two CBA ItemBuilder Simultaneously

The CBA ItemBuilder is an application that can be started only once per installation. The underlying reason is that each CBA ItemBuilder installation (also referred to as

instance in the following) has its dedicated workspace in which the contents of a CBA ItemBuilder *Project File* are unpacked during processing.

At the same time, when creating many (as uniform as possible) assessment components with CBA ItemBuilder, it can be helpful to have a *Project File* open to view implementations there while working on another project file in a second CBA ItemBuilder (see Figure 6.47 for an example).



Installing multiple instances of the CBA ItemBuilder is required to run multiple CBA ItemBuilders in parallel.

Multiple Instances: The CBA ItemBuilder can be installed multiple times to run several instances in parallel. Each installation must have its own directory. In addition, a so-called port (i.e., a dedicated connection address on the computer on which CBA ItemBuilder is installed) must be different for each instance. Therefore, after installing a second CBA ItemBuilder or copying the entire program directory, a port configuration must be changed for one of the instances to be operated in parallel.

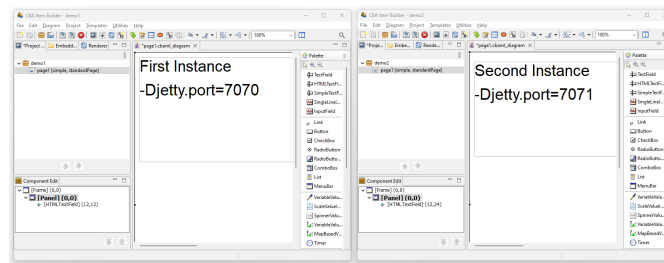


FIGURE 6.47: Example screen shot running two CBA ItemBuilder in parallel.

The following steps are necessary to manually enable the use of two instances of the CBA ItemBuilder:

1. Install the CBA ItemBuilder (if not yet done, as described in section 1.1).
2. Locate the program directory. By default this should be 'C:/users/{UserName}/AppData/Local/CBA-IB-{Version}'
3. Duplicate the sub-directory IB (i.e., create a copy of the directory IB and rename it to IB2).
4. Find the file `cba-itembuilder.ini` in the duplicated folder (IB2) and open it with a text editor (e.g., notepad).
5. Change the number that is assigned to the `Djetty.port` to a higher and different value, e.g.:

```
-Djetty.port=7072
```

6. Save the file `cba-itembuilder.ini` and close the editor.
7. Create a new shortcut in Windows Explorer to the file `cba-itembuilder.exe` in the directory `IB2`. Starting this application from the copied directory should now be possible in parallel to the application from the directory `IB`.

Make sure you never open an identical CBA ItemBuilder *Project File* in two instances in parallel. The clipboard (see section 3.7.2) cannot work between different instances of CBA ItemBuilder, i.e., if something is copied in one instance, it cannot be pasted in another instance. To transfer content from one instance to another, the function to export and import pages is available (see section 6.8.6). Alternatively, to transfer content within an instance, the template function can be used (see section 6.8.7).

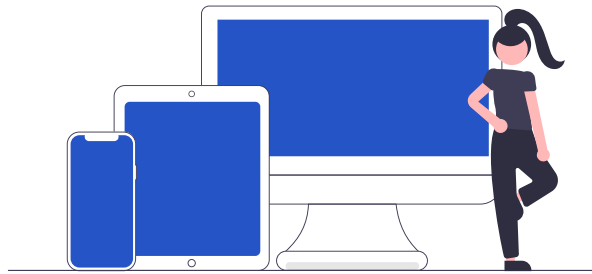
6.9 Creating Assessments in Multiple Languages



The current CBA ItemBuilder supports assessment content in one language at a time. If assessment components are required in multiple languages, they can be translated with the CBA ItemBuilder as an authoring tool and saved in another CBA ItemBuilder *Project File*. In addition to the manual translation of all texts directly within the *Page Editor*, CBA ItemBuilder also supports a translation workflow based on the *XLIFF* format. Several details should be considered when designing items and implementing workflows to use this functionality efficiently. Use the contact options (see section 1.2) to learn more about *XLIFF* support.

7

Test Assembly and Deployment



After the previous chapters focused on the creation of individual item projects (i.e. CBA ItemBuilder project files together with one or multiple tasks as the entry points), this chapter is about the use of these item projects for operational test administration and data collection.



It is not possible to perform or administer an assessment with the CBA ItemBuilder alone. A particular delivery software is always required, and various software tools can be used to integrate CBA ItemBuilder items. Moreover, identical CBA ItemBuilder items can be administered with different tools in different conditions (for instance, offline and online).

7.1 Quick-Start: Assessments using R (and Shiny)



To follow this quick-start tutorial, download and install [R](#) and, for instance, [R-Studio](#).¹ After this installation open R and install the packages: `shiny`, `remotes`, `knitr` and `ShinyItemBuilder`:

```
install.packages("shiny")
install.packages("remotes")
install.packages("knitr")
remotes::install_github("kroehne/ShinyItemBuilder", build_vignettes = TRUE)
```

When both packages are installed, the CBA ItemBuilder projects are required. The project files are expected in the folder provided as argument to the function call `getPool()`. Additional arguments are possible to define the order of tasks within the item pool (by default, all files and tasks are included, ordered by name).

In the following example, we use the function `getDemoPool("demo01")` to illustrate the use of the package with example items:

```
# app.R

library(shiny)
library(ShinyItemBuilder)

# demo items
item_pool <- getDemoPool("demo01")

# your items would be loaded via
# item_pool <- getPool(path="PATH-TO-YOUR-IB-PROJECTS")

assessment_config <- getConfig()

shinyApp(assessmentOutput(pool = item_pool,
                          config = assessment_config,
                          overwrite=T),
         renderAssessment)
```

The assessment is started by executing the complete file `app.R`.

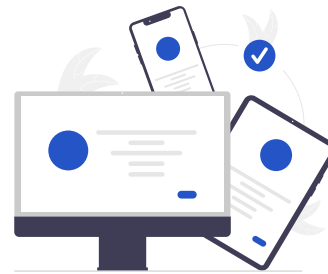
The items can be answered in a browser using the URL either started automatically or displayed by R / Shiny (e.g., <http://127.0.0.1:PORT>)². More configurations are possible (see section 7.3). If you have an account for shinyapps.io, the assessment can

¹Alternatively, [Visual Studio Code](#) could be used.

²PORT is the port assigned by shiny and shown in the R console window as follows: Listening on <http://127.0.0.1:4723>. In this example, 4723 is the PORT.

be published directly from RStudio and used for data collections online (see section 7.3 for details).³

7.2 (Technical) Terminology and Concepts



Finding an appropriate test delivery software requires some technical considerations. Before describing the available tools, this section briefly describes some important technical terms and concepts.

7.2.1 Deployment Mode (Online, Offline, Cached and Mobile)

All assessments created with the CBA ItemBuilder require a web browser or browser component for rendering the generated HTML / JavaScript content. Web browsers are available on various devices, including desktop and mobile computers, cell phones, televisions, and game consoles. CBA ItemBuilder items can be used one way or the other on various platforms with built-in modern web browsers.



Computer-based or digitally-based assessments can be used in various ways for data collection and empirical research. Before selecting the appropriate software, it is therefore helpful to clarify the different *deployment modes*.

Online Deployment: The assessment is conducted in a supported web browser, constantly connected to the internet, and data are stored on a web server. Accordingly, some technology is required for hosting the assessment. The required hosting technology depends on the deployment software, including, for instance, R/Shiny (see sections 7.1 and 7.3) or PHP-based web hosting or cloud technologies (e.g., using

³Note: The number of parallel test takers that can use the online assessment simultaneously depends on the hosting of the Shiny application.

TAO, see section 7.4), or any server-side technology (e.g., dotnet core used either directly or containerized for the software described in section 7.5).

Offline Deployment: The same software components as for the online deployment (i.e., the assessment is conducted in a web browser, together with a server component), but the server- and browser components run on the same device. Accordingly, items and data are stored only locally, and an internet connection is neither required to prepare nor run the assessment. Data are collected by copying the data from the local devices or by collecting the devices (e.g., thumb drives).

Cached Deployment: The assessment is conducted in a web browser connected to the internet while the assessment is prepared, but no (reliable) network connection is required during the assessment. Hence, data are stored locally, and after the assessment is finished, data are collected either online or by copying the data from local devices.

Mobile Deployment: Mobile deployment is a specific offline or cached assessment conducted on a mobile device. Since mobile devices are used, either a native app or a progressive web app is used (instead of a server), and test execution is possible even if there is no reliable internet connection (i.e., it is a cached deployment on mobile devices). When internet connectivity is available, data can be stored simultaneously online or transferred when connectivity is available.

Mobile deployment is specific in different ways. Firstly, no direct file access is typically possible on mobile devices (instead, data are transferred via the internet). Secondly, an app can stay accessible after installation, allowing the implementation of prompts or notifications that are either offline or triggered via so-called push notifications can try to attract the test-taker's attention. Third, mobile devices are often equipped with additional sensors that can contribute to the collected data (and para-data) if incorporated by the app.

7.2.2 Browser Requirements and Availability of (Embedded) Content

Since assessments created with the CBA ItemBuilder are always displayed in a web browser or a browser component⁴, *Cross-Browser Compatibility* and *Browser Requirements* need to be considered.

Browser Requirements: Not all browsers support all features, adhere to the usual standards, show HTML, and similarly interpret JavaScript content. A typical approach is, therefore, to test the prepared assessment content in different browsers and then, if necessary, restrict the assessment to specific browser versions.

For rendering the CBA ItemBuilder generated content, the [React](#) framework is used

⁴This is true even if specific browsers (such as the [Safe Exam Browser](#)) are used or if CBA ItemBuilder content is integrated with the *TaskPlayer API* in mobile applications (so-called hybrid apps) or in desktop applications (e.g., using [Electron](#))

internally in the current versions.⁵ React is supported by modern browsers, and so-called polyfills could be added for older browser versions.⁶ However, the deployment software used to combine CBA ItemBuilder tasks to tests might add additional requirements to be available to run assessments successfully. DIPF/TBA provides tools for online assessment (e.g., IRTlib, see section 7.5) that require a web browser that supports WebAssemblies. Using the *TaskPlayerAPI*, additional deployment software can be implemented with other server technology and for all browsers or browser-components supporting [React](#).

Besides the deployment software, special attention regarding browser support is also required for content that is integrated into CBA ItemBuilder items via `ExternalPageFrames` (see section 3.14). These extensions, created by JavaScript / HTML programmers, are integrated into items using so-called `iframes` and must be checked to ensure that they function correctly in all browsers used for an assessment.



Limitations of the browsers that can be used for an assessment can be due to different languages (e.g., the CBA ItemBuilder runtime, the deployment software used, and the content included as an `ExternalPageFrame`).

Limitations, for example in playing videos, can also be caused by configurations of the web server (e.g. the configuration to support range HTTP requests).

Cross-Browser Compatibility: If the browsers used in online deliveries cannot be controlled and standardized, it is recommended to test the assessments in advance in different browsers. On Windows computers, the display and functioning of CBA ItemBuilder items in locally installed browsers can also be tested by using the regular *Preview* and copying the preview URL (see section 8.4.1). To check cross-browser comparability of CBA ItemBuilder deployments (see section 8.4.1), an online deployment must be prepared first (for instance, using the R/Shiny example described in section 7.1).

A very efficient approach to testing the compatibility of assessments in different browsers is to use third-party vendors that provide web-based access to different browser versions on different devices (cloud web and mobile testing platforms, such as [BrowserStack](#), or similar services).

Availability of (Embedded) Content: Content embedded Using `ExternalPageFrame` must not only be working in the browser that are expected to be used for an assessment. The embedded material must also be available during the assessment. The use of *Local* resources is required for offline deployment. Moreover, network connection must be stable and provide enough *bandwidth* for resources included as *External* resources (see section 3.14 for the configuration of components of type `ExternalPageFrame`).

⁵This is true for all CBA ItemBuilder versions starting with 9.0, see Appendix B.5 for details.

⁶See, for instance, <https://create-react-app.dev/docs/supported-browsers-features/>.

Special attention is required when multiple assessments share the same internet connection. For instance, it must be possible to load large audio and video files fast enough, even if, for example in a class context, all test-takers share a network access. Factors that influence the *bandwidth* are besides the storage of data (*upload* of log data, result data and data for recovery, see section 7.2.9) especially the *download* of media files (images, videos, audio files) that have to be transferred from the server to the client. The underlying *Taskplayer API* (see section 7.7) provides methods to implement the pre-loading of media files.

7.2.3 Fullscreen-Mode / Kiosk-Mode

Test security (see section 2.10) can translate into different requirements, particularly for high-stakes assessments. To standardize assessments, the assessment content should fill the entire screen if possible. Depending on the deployment method, this goal can only be approximated (although not guaranteed in certain browsers or on mobile devices) using *Fullscreen-Mode*.

Fullscreen-Mode: In online assessments, a full-screen display of assessment content can be possible in (regular) browsers on computers with desktop operating systems (Windows, Linux, macOS), and *Fullscreen-Mode* must be initiated typically by a user interactions. However, since test-takers can exit full screen mode at any time, the assessment software might hide at least the item content when full screen mode is exited.

Mandatory Fullscreen: For assessments on desktop computers, a requirement can be to ensure that the assessment is only displayed if the browser is in full-screen mode. Since even browser-based assessment software can, to some extent, diagnose that the current view is part of a full-screen presentation, assessment software can make the full-screen presentation mandatory. If a test-taker exits the full-screen presentation, different actions can be considered: Access-related paradata (i.e., a particular log event) can be created, a test administrator can be informed using, for instance, a dashboard (see section 7.2.4), or the assessment content could be faded-out, advising test-takers to return to the full-screen presentation before an assessment can be continued.

Kiosk-Mode: In offline deployments (or if a pre-defined browser is provided for online deployments), a so-called *Kiosk-Mode* prevents users (i.e., test-takers) from exiting the full-screen presentation of assessment content. Various browsers and add-on's offer *Kiosk-Mode* for different operating systems. For instance, the offline player provided as part of the IRTlib-software (see section 7.5) provides a *Kiosk-Mode* for Windows.

A free software solution for establishing test security on computers with Windows, macOS and iOS operating systems, which is used in a variety of application contexts, is the [Safe Exam Browser \(SEB\)](#). Since SEB is a (standard) HTML browser, which has only been supplemented with additional components and configurations for test

security, this software can be combined in various ways with the CBA ItemBuilder deliveries.

7.2.4 Interviewer-Menu, (Live) Dashboard and (Remote) Proctoring

While a test-taker answers items in a browser, various test administrator observations or interactions (on-site or remotely) may be possible.

Interviewer-Menu: The simplest form, which is also possible for offline deliveries, is an interviewer menu (meaning an interviewer or test administrator can change or configure the course of an assessment on the test-taker's device). Interviewer menus are typically protected by a password or a hidden keyboard shortcut, so only test administrators can access the features prepared for administrative purposes. Typical functions include the possibility to skip a single task or item, to pause or end an assessment or to jump to a particular section of the assessment.

(Live) Dashboards: If the assessment takes place on a computer that is accessible via network (locally or on the internet), control by interviewers or test administrators can also be realized via a *Dashboard*. Unlike an interviewer menu, the functions for monitoring or controlling an assessment using a dashboard are available on a different device than the device on which a test-taker is working on tasks or items. Dashboards can be used for various purposes, for instance, to implement supervised online testing, where an interviewer is informed about the progress of the test-taking and can, for example, give further instructions or answer questions via telephone, text, or video chat.

(Remote) Proctoring: Remote proctoring is a particular form of a dashboard in which additional information from a webcam or microphone can be displayed or automatically evaluated. Dashboard and remote proctoring can be (partially) automated using artificial intelligence techniques, and different types of play (related to privacy and informed consent) are possible.

7.2.5 Input and Pointing Device

Special considerations are required regarding the input devices used for point-and-click and text responses.

Touchscreen vs. Mouse-Click: Assessments on touchscreens (i.e., using touch-sensitive displays) require additional thoughts for at least two other reasons. First, it is inherent in touch operations that the place where the touch gesture is performed (using either a finger or a specific stylus) is not visible when the finger covers parts of the screen. Hence, if a small area needs to be clicked precisely, this property must be considered when designing the interactive item. Second, web browsers might treat touch events differently from click events. In particular, for content embedded as

`ExternalPageFrame`, it is necessary to ensure that touch and click events are treated by the JavaScript / HTML5 content in all browsers as expected.

Touch devices might also, by default, use additional gestures, including pinch zoom gestures and, for Windows operating systems, the so-called charms bar (activated using a touch gesture over the edge of the screen).

Touchpad vs. Mouse-Click: If notebooks with touchpads are used, an external mouse might be considered for standardization purposes (i.e., to make sure that test-taker can answer the assessment regardless of their familiarity with touchpads), and touchpads might be deactivated to avoid distraction when typing and clicking is required.

(Onscreen) Keyboard and Text Entry: Text and numeric entries in assessments are usually recorded using a built-in or connected keyboard. Text input might automatically trigger an on-screen keyboard for devices with a touch screen. The operating system automatically displays the on-screen keyboard and then reduces the space available on the screen for the item content. In addition, the on-screen keyboard might allow the test-taker to exit *Kiosk-Mode*.

Careful usability testing (and if a *Kiosk-Mode* is planned, robustness checks regarding test security) are necessary, particularly on touch devices.

7.2.6 Authentication / Account Management

Regardless of the specific software used for test delivery, the following terms are relevant to web-based assessments.

Session: An essential concept for online assessments is the so-called session. Even if data collection takes place entirely anonymously (i.e., without access restrictions), all data belonging to one person should ideally result in one data record. Accordingly, the session is the unit that bundles the data that can be jointly assigned to one (potential) test-taker. With the help of client-side storage (i.e., session storage, cookies, or local storage), the ID of a session, which is usually created randomly, can be stored so that the data for an assessment, if interrupted and continued from the same test-taker, can be matched. Sessions IDs can be stored within a particular browser on a computer if the client-side storage is activated and the test-taker agrees to store client-side information.

Log-in/Password vs. Token: Authentication for online assessments can use *Log-in* and *Password* (i.e., two separate components) or a one-time *Token* (i.e., only one component). Often, there is no reason to arbitrarily separate log-in and password if the identifier is created as pseudonym only for a particular data collection. A distinction can be made with regard to the question of whether the assessment platform checks the validity of the identifier (access restriction) or if the identifier is only stored.

Protection of Multiple Tabs/Sessions: Finally, when a particular authentication is implemented, it must be considered that the log-in can potentially occur multiple

times and in parallel. The delivery software might ensure that an assessment with one access (e.g., log-in/password or token) cannot be simultaneously answered and accessed more than once.

7.2.7 Task-Flow and Test Assembly

As described in section 2.7 items (i.e., in case of CBA ItemBuilder assessments *Tasks*) can be combined in different ways to assessments. Using the CBA ItemBuilder, the assembly of individual assessment components into tests is part of the test deployment software. An assessment component is any component that can be meaningfully used as component, including a log-in page, cover pages, instruction pages, the actual items or units, and exit or closing pages shown to test-takers.

Linear Sequence: The most common arrangement of assessment components is a *Linear Sequence*. CBA ItemBuilder *Tasks* can create a sequence that might contain items, prompts, feedback pages and dialog pages. As long as the same sequence is used for all test taker, all deployment software can be used that support linear sequences.

Booklets: If different sequences or combinations of CBA ItemBuilder *Tasks* are used (of which each target person works on exactly one), the procedure corresponds to so-called *Booklets* (or test rotations). Depending on the study design, the assignment of a test taker to a particular booklet or rotation is either done in advance (usually assigned to log-ins or tokens, see 7.2.6), or the assignment is done on-the-fly when a session is started.

Skip Rules: The deployment software for CBA ItemBuilder assessments is also responsible for omitting or skipping *Tasks* depending on previous responses. This functionality is only required for the test deployment software, if the skip rule can not be implemented within CBA ItemBuilder *Tasks* (using, for instance, multiple pages and either conditional links or the CBA ItemBuilder task's finite-state machine).

Multi-Stage Tests and Adaptive Testing: The more complex and elaborate the calculations become, which are required for the selection of questions (or pages in a CBA ItemBuilder task), and the more items in total are available, the more complicated the implementation within a (single) CBA ItemBuilder task becomes. Accordingly, special delivery software is required to implement complex multi-stage tests and item- or unit-based, uni- or multidimensional adaptive tests using CBA ItemBuilder items (see section 7.5).

7.2.8 Time Limits across Tasks

Analogous to a sequencing of assessment content across individual parts (such as *Tasks* in the case of CBA ItemBuilder created assessments), the time restriction across tasks can be provided by a deployment software.

Time Measurement: The requirement of a time limit across tasks typically exists only for the actual items of a (cognitive) test, while before and possibly after the time-restricted part further content, for instance, with instruction pages and a farewell page are administered. Moreover, if a time limit across tasks is used, one or multiple additional pages are often shown, when the timeout occurred.

Remaining Time: When the minimal time to solve a particular item is known or can be approximated, the deployment software might apply the time limit already at a *Task-switch* (i.e., when in CBA ItemBuilder based assessments a new task is requested based on a `NEXT_TASK`-command, see section 3.12.2), if the remaining time (i.e., the time before the time limit will occur) is below the expected minimal time that would be required to work on the (requested) next task.

Result-Data (Post-)Processing and Timeouts: Time limits not only affect the presentation of items, meaning that within a particular section of the assessment no more items (i.e., *Tasks* within CBA ItemBuilder project files) are presented, when a timeout occurred. Time limits also affect the (missing) coding of the responses in the current task and in all subsequently tasks, not presented because of a timeout.

First, it is important to note, that the deployment software requests the `ItemScore` (see section 7.7) for the current item (i.e., *Task* within an CBA ItemBuilder project file) when a timeout occurs. Second, within this task, variables (i.e., classes) can be coded as *Not Reached* (NR), if the corresponding questions were not yet presented in a *Task* with multiple pages. Hence, the differentiation between *Not Reached* (NR) and *Omitted Responses* (OR) for *Tasks* with multiple parts must be included in the (regular) CBA ItemBuilder scoring (see section 5.3.11).

For assessment components not presented because of a timeout (or because an interviewer aborted the test, see section 7.2.4), no `ItemScore` will be available, because it is computed only when a *Task* is exited. Hence, the deployment software is required to determine the remaining tasks that would have been presented without the interruption and assign the appropriate missing code, either *Not Reached* (NR) in case of a timeout or *Missing due to Abortion* if an interviewer aborted the assessment.

7.2.9 Date Storage and Test-Resume

Regardless of the deployment software, there are some data types that arise in all CBA ItemBuilder-based data collections. These are briefly described in this section.

ItemScore (Result Data): When exiting CBA ItemBuilder items by a task-related command (see section 3.12.1) or by an external navigation request of the delivery

environment (timeout or test abort), the defined scoring rules (see section 5.3) are evaluated. In the current version of the CBA ItemBuilder (i.e., for the REACT generator), the `ItemScores` are provided by the runtime as JSON data and collected by the delivery platform.

Traces (Log Data): In addition to the `ItemScore` data, trace data are provided automatically by the CBA ItemBuilder runtime. The data are provided as individual log events and can be collected by the deployment software or even be analyzed already instantly during the assessment.

Snapshot: In addition to the two data collected for further empirical analyses, snapshots of all internal states that represent the tasks are provided by the runtime so that even in case of interruptions, the possibly complex CBA ItemBuilder tasks can be continued at the last processing state.

Assessment content that is embedded into CBA ItemBuilder items using `ExternalPageFrames` can use the *Snapshot* of the CBA ItemBuilder Runtime to implement persistence of content across page changes (see section 4.6.5).

7.3 Using CBA ItemBuilder Items with R (Shiny Package)



The R package `ShinyItemBuilder` allows using CBA ItemBuilder items in web-based applications created with `R/Shiny`. This allows local administration of tests (from `RStudio`) and online administration using, for instance, www.shinyapps.io (or hosting shinyproxy).

The use of R/Shiny for assessments is advantageous for two main reasons: It combines the data collection (seemingly) with the (psychometric) use of gathered responses data and collected log events. Moreover, since an easy-to-use infrastructure for Shiny applications exists, it enables a swift approach to run online assessments

without setting up a dedicated hosting environment. Although this hosting might be less performant than hosting using more standard web technologies, item authors can use R-functions to customize the test assembly (e.g., for multi-stage and adaptive testing).

Concerning the different modes of test deployment (see section 7.2.1, R/Shiny can be used for *stand-alone* deployment either locally, or online.

- Local: The assessment is started directly from R locally, and test-taker answer items in a browser on the same computer.⁷
- Online: The assessment is hosted using R / Shiny on a server (e.g., using www.shinyapps.io), and test-takers answer the items in a browser, either on a desktop device or even on a mobile device.

Note that even if an online deployment should be created using the R package `ShinyItemBuilder`, the preparation is done locally in R. After completing the preparation, the Shiny app is deployed to the online server.

7.3.1 Use of CBA ItemBuilder Project Files in `shinyItemBuilder`

The R/Shiny package `ShinyItemBuilder` needs to know which items should be administered. This can be defined by providing an item pool, created from a folder with CBA ItemBuilder project files or a list of CBA ItemBuilder project files and optional tasks.

```
item_pool <- getPool(path="PATH-TO-YOUR-IB-PROJECTS")
```

If no additional function for navigation is defined (see section 7.3.3), the project/-tasks defined in the item pool will be administered as linear sequence. Hence, you can either change the order of project files / tasks in the object `item_pool` after the call of `getPool()`, or provide a specific function `navigation` (see section 7.3.3).

The configuration is done using a list of attributes and functions, created with the function `getConfig()`.

```
assessment_config <- getConfig()
```

⁷Alternatively, a portable R (e.g., using [DesktopDeployR](https://github.com/zarathucorp/shiny-electron-template-m1-2023)) can be used, or R, the Shiny-App and the browser can be bundled together with a browser as electron app (see, for instance, <https://github.com/zarathucorp/shiny-electron-template-m1-2023> for a template).

Various options that can be defined include the visual orientation and zoom of items, as described in the help:

```
?ShinyItemBuilder::getConfig
```

7.3.2 Start and End of Assessments using shinyItemBuilder

The default configuration to start the assessment that the R/Shiny package `ShinyItemBuilder` will create a new identifier, when the assessment is loaded the first time from a particular browser. The identifier is stored in the local session storage, so that the same identifier will be used if the test-taker closes and re-opens the page (or reloads the page). Using this identifier, the test-taker will always return the last visited item. If the last item was ended, an empty page will be presented and the function `end` defined in the configuration will be called.

The function `end` can be defined in the configuration to implement different authentication workflows.

Multiple Runs: A new identifier is created automatically when the URL is visited the first time (or if the application is started in a local deployment). The identifier is stored either in the session storage (default or `sessiontype="sessionstorage"`), in the local storage (`sessiontype="localstorage"`) or using cookies (`sessiontype="cookie"`). As long as the identifier is stored, the started session will be continued.⁸ After the last item the function `end` will be called, that is defined in the object `assessment_config` using the function `getConfig()`:

```
assessment_config$end=function(session){
  showModal(modalDialog(
    title = "You Answered all Items",
    "Please close the browser / tab.",
    footer = tagList(actionButton("endActionButtonOK", "Restart"))))
}
```

This function is called when the last item was shown (i.e., if the function `navigation` returns `-1`). The Shiny `actionButton("endActionButtonOK", "Restart")` allows test-taker to re-start the assessment.

Single Runs: If the function `end` does not include the action button (i.e., if the footer is an empty tag-list: `footer = tagList()`), the test-taker will not be able to start the assessment again, once the last item is reached. Alternatively, the `end` function can also be overwritten to re-direct to another URL:

⁸Currently `ShinyItemBuilder` will not restrict multiple tabs or browser windows accessing the assessment simultaneously.

```
assessment_config$send=function(session){
  session$sendCustomMessage("shinyassess_redirect",
    "https://URL-TO-REDIRECT.SOMEWHERE/?QUERYSTRING")
}
```

Authentication: If the assessment is configured with `sessiontype="provided"`, the object `assessment_config` created with the function `getConfig()` can contain a custom login-function:

```
assessment_config$login=function(session){
  showModal(modalDialog(
    tags$h2('Please Enter a Valid Token and Press "OK".'),
    textInput('queryStringParameter', ''),
    footer=tagList(
      actionButton('submitLoginOK', 'OK')
    )
  ))
}
```

This function is shown, if the assessment is not started with a query-string parameter that includes a parameter with the name defined in the config `assessment_config$queryStringParameterName` (default is `token`). If the parameter is provided, the function `assessment_config$validate` is called to verify that the token is valid.

7.3.3 Define Sequencing / Navigation using `shinyItemBuilder`

Using `shinyItemBuilder` allows to implement different approaches for test assembly and test assembly (see section 7.2.7).

Linear Sequence: If all test-taker should answer the identical CBA ItemBuilder projects/tasks in a similar sequence, ordering the tasks in the object `item_pool` is sufficient.

In the following example, the demo item pool is reordered, so that the items are administered in the reversed order:

```
item_pool <- getDemoPool("demo01")
item_pool <- item_pool [c(6,5,4,3,2,1),]
assessment_config <- getConfig()
shinyApp(assessmentOutput(pool = item_pool,
  config = assessment_config,
```

```

      overwrite=T),
    renderAssessment)

```

Booklets: If multiple sequences are required, a modified function `navigation <- function(pool, session, direction="NEXT")` can be provided. The function obtains the item pool (as defined above) and a session object.

Possible values for `direction` are the possible *Runtime Commands* available in the CBA ItemBuilder (i.e., either `NEXT`, `PREVIOUS` or `CANCEL`, see section 3.12), and the value `START`. The function is expected to return the index of the next item (starting with 1), corresponding to the row in the item pool object. The function can store and retrieve temporary information using the functions `setValueForTestTaker(...)` and `getValueForTestTaker(...)`. The example in `vignette("booklets")` based on `demo01` defines two booklets (using items 2,3,6 in booklet 1 and using items 4,5,6 in booklet 2). A random booklet is assigned (`sample(unique(booklets$Booklet),1)`) and stored for the test-taker.

Adaptive Testing (CAT): A modified function `navigation <- function(pool, session, direction="NEXT")` can also be used to implement various forms of adaptive testing. The package contains an example (see `vignette("cat_with_catR")`) how to use the R-package `catR` (Magis and Raiche, 2012; Magis and Barrada, 2017) for a simple adaptive test (see Figure 7.1).

7.3.4 Score Responses in R

CBA ItemBuilder tasks provide a scoring (see chapter 5) that can be evaluated and reused in R.

Retrieve ItemBuilder Scoring: The R package provides a template for the `score`-function, that can be used to extract information from the CBA ItemBuilder provided `ItemScore` (see section 7.2.9). Adaptation of this function is necessary if selected responses of the already administered tasks are to be used for test sequencing (e.g., if `ShinyItemBuilder` is used for adaptive testing). The `score`-function is called automatically, after the administration of one item is completed.

7.3.5 Feedback in R using Markdown/knitr

As discussed in section 2.9.2, a technical platform for report generation is necessary to provide instant feedback after an assessment is completed. In the ecosystem of R/Shiny, the `knitr` package provides an easy-to-use approach to include dynamic documents to dynamically generated documents.

The example in `vignette("feedback")` based on a selection of items provided as `demo01`

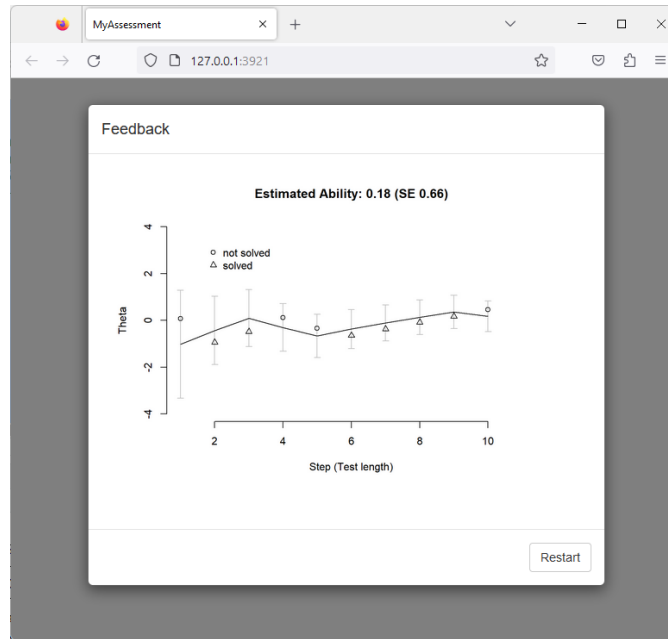


FIGURE 7.1: Output of an Adaptive Test Created with `ShinyItemBuilder` and `catR`.

illustrates how instant feedback can be created using markdown and `kntir`. Figure 7.1 provides another example.

7.3.6 Data Storage and Data Access

The package `ShinyItemBuilder` illustrates how CBA `ItemBuilder` items can be used with R/Shiny, ready to use for small-scale studies that can be hosted, for instance, on shinyapps.io. Data are stored in R in a global variable `runtime.data`, persisted in a folder configured using the argument `Datafolder` (default value is `_mydata`):

```
assessment_config <- getConfig(Datafolder="folderName")
```

Data are stored for each session identifier in a `*.RDS` file that can be loaded in R using the function `readRDS()`. By default, data are only stored in the current instance (i.e., data will be lost if the application will be put into a Sleeping state or the instance is deleted or newly created on shinyapps.io).⁹

To access data online, the package illustrates how a simple *Maintenance* interface

⁹Shiny can be used with more advanced approaches to achieve persistences, see, for instance [here](#).

could look like. If a maintenance password is provided, the keyboard shortcut `Ctrl + X` (configured as argument to the function `getConfig(maintenanceKey=list(key="x", ctrl=T, shift=F, alt=F), ...)`) or the argument `?maintenance` in the query string (configured as argument to the function `getConfig(maintenanceQuery = "maintenance", ...)`) opens a shiny dialog page. (see [vignette\("maintenance"\)](#)).



Warning: Understanding how files persist in the chosen hosting environment is fundamentally necessary. Storage in local files is done only in the running instance and is lost when, for example, the Shiny application is put into a Sleeping state or is updated. Inconsistencies are expected if multiple instances are used. Before running a concrete data collection, be sure to read the [vignette\("datastorage"\)](#).

7.3.7 Side Note: Interactively Inspect Log Events of CBA ItemBuilder Tasks

The R package `ShinyItemBuilder` can also be used to directly log events collected by a CBA ItemBuilder task live in RStudio.

This is illustrated with the following example:

```
item_pool <- getDemoPool("demo02")
assessment_config <- getConfig(Verbose = T)

shinyApp(assessmentOutput(pool = item_pool,
                          config = assessment_config,
                          overwrite=T),
        renderAssessment)
```

Note the argument `Verbose = T` that is provided to the function `getConfig`. In verbose mode, `ShinyItemBuilder` will print detailed information to the R output window, while the items can be interacted with in a web browser using `shiny`.

7.4 Using CBA ItemBuilder Items with TAO (using Portable Custom Interactions)



The repository [fastib2pci](#) can be used as a tool to convert CBA ItemBuilder projects/tasks into PCI components that can then be integrated into QTI. Single projects/tasks and linear sequences of projects/tasks are supported. The generated PCI components can be used, for example, to embedd CBA ItemBuilder items into [TAO](#)-based assessments.

Tests in QTI-compatible platforms such as [TAO](#) are defined as a sequence of QTI items. Each QTI item contains one or more QTI interactions. QTI interactions can be *Common Interactions*, *Inline Interactions*, *Graphical Interactions*, or *(Portable) Custom Interactions* (PCI). If TAO is used as the assessment platform, then QTI items can be created, managed, and edited directly in TAO. For use cases where assessment content cannot be implemented with QTI interactions, *(Portable) Custom Interactions* (PCI) can be integrated.¹⁰

7.4.1 Prepare CBA ItemBuilder Project Files for [fastib2pci-Converter](#)

For item authors without experience in software development or if existing assessment content is already available, the CBA ItemBuilder can be used as authoring tool for *Portable Custom Interactions* (PCI), for instance, to create *Technology-Enhanced Items* (see section 2.3) that can be used in [TAO](#). *Portable Custom Interactions* that can be used in QTI-items can be created using either a single or a linear sequence of multiple CBA ItemBuilder *Project Files*.

¹⁰Note that examples and environments such as the [QTI-PCI development environment](#) can be helpful for software developers when implementing *Portable Custom Interactions*.

To create a *Portable Custom Interactions* with CBA ItemBuilder *Project Files*, the `fastib2pci` can be used. It allows creating *Portable Custom Interactions* with either single or multiple *Project Files*. As a necessary prerequisite, one or multiple *Tasks* must be defined within each *Project Files*. If multiple *Project Files* are used, an alphabetical order is used. The same applies to the order of *Tasks*, if within one CBA ItemBuilder *Project File* multiple *Tasks* are defined.

7.4.2 Generating PCI-Components using `fastib2pci`-Converter

The converter `fastib2pci` is provided as a *Github project template* that contains a so-called *CI/CD worker* (i.e., Github actions). Using the converter requires an account at github.com. After creating an account and log-in to your profile, navigate to the repository `fastib2pci` and push the button ‘Use this template’ as shown in Figure 7.2.

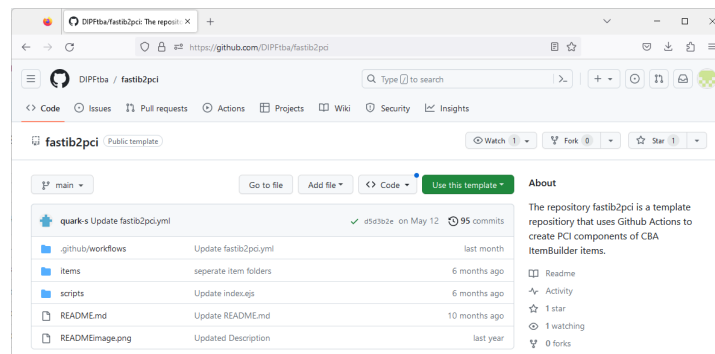


FIGURE 7.2: Repository `fastib2pci` shows the button Use this template after log-in to github.com.

Provide a new project name and make sure to select *Include all branches*.¹¹ As soon as the project is created, the *CI/CD worker* included in the template repository `fastib2pci` will create the PCI components using the example CBA ItemBuilder *Project Files* included in the folder `items`. After the worker completed, the *PCI* components can be downloaded from the repository in the section *Releases* (or using the following link: <https://github.com/{your github account name}/{repository name}/releases>).

To use the converter with your own CBA ItemBuilder *Project Files*, create a new directory (e.g., `component_1`), upload CBA ItemBuilder *Project Files* to the new directory, and delete the example directories ‘test_1’ and ‘test_2’. After committing the files, the *CI/CD worker* will automatically update the *PCI* components and provide them in the section *Releases*. Note that you can use a *git*-client (see section 8.3.2.2) to work with the repository.

¹¹Note that you can select *Public* or *Private* for your new project.



Generating *PCI*-components for using CBA ItemBuilder *Project Files* in *TAO*-based assessments is possible without installing a specific software. The packaging of *PCI*-components is done within the *CI/CD worker* worker that is provided in the template repository [fastib2pci](#).

7.4.3 Flavors of *PCI*-Components Created by *fastib2pci*-Converter

For each sub folder of the directory `items/` one `*.Tar`-file (*tarball* archive)¹² is generated. Accordingly, if you plan to use multiple *PCI*-components for a particular assessment, only one *git*-repository is required.

The `*.Tar`-files generated by the *fastib2pci*-converter contain different versions of the *PCI*-components, using the identical CBA ItemBuilder *Project Files* in the repositories `items/`-folder. There are two reasons to create different flavors:

IMS *PCI* vs. *TAO PCI*: The implementation of *PCI* support in *TAO* changed over the last years from a *TAO*-specific implementation (*TAO PCI*) towards the standard specification (*IMS PCI*). Both versions are generated by the *fastib2pci*-converter.

Generic vs. Specific: CBA ItemBuilder content is embedded using *PCI*-components using `iframes`. All resources used in CBA ItemBuilder *Project Files* are embedded in the generated **Specific** *PCI*-components (required to make the *Portable*). However, since *TAO* might have, depending on the configuration, a size limitation, the *fastib2pci*-converter also creates a **Generic** version that only includes the CBA ItemBuilder runtime in the *PCI*-component (resulting in very small file sizes of about 150 KB), referring to static *GitHub Pages* for hosting the actual item content.



To use the *Generic PCI*-components created by *fastib2pci*-converter, *github pages* must be enabled.¹³

If *GitHub Pages* are activated in the project (see Figure 7.3), the *CI/CD worker* worker will automatically prepare the hosting for the item content as required for the **Generic** *PCI*-components.



Important note: If *GitHub Pages* are activated (i.e., if **Generic** *PCI*-components are used), the item content is freely accessible using the following URL: `https://{your github account name}.github.io/{repository name}/{sub directory name}/`

¹²Note that `*.Tar`-files are archive file that can be extracted, for instance, using [7-Zip](#) on Windows computers.

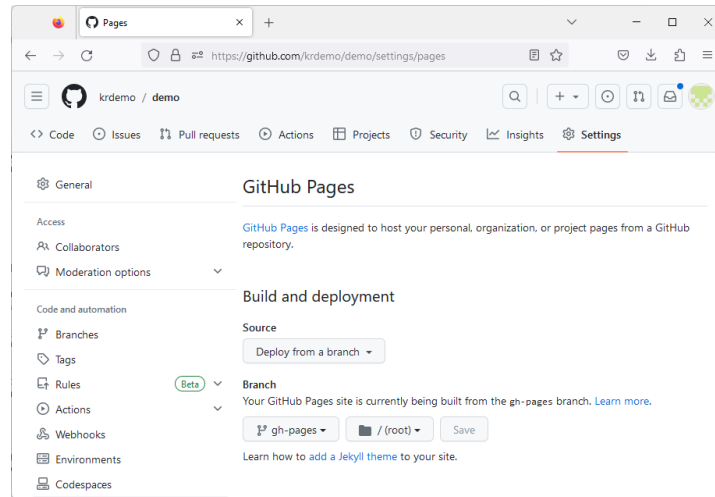


FIGURE 7.3: Example repository settings showing activated *GitHub Pages*.

If, for item protection reasons, the content can not be freely accessible, then using the **Generic PCI** components requires providing a (secured) static hosting of the content in the *Branch* `gh-pages`, generated by the `fastib2pci`-converter. This effort is not required for the **Specific** PCI components.

7.4.4 Side Note: Archive Assessment Content using GitHub Static Pages

For various reasons, it can sometimes be reasonable to provide assessment content statically (i.e. without data storage). In addition to screenshots, CBA ItemBuilder tasks, for example, can also be displayed as interactive web pages. With the converter `fastib2pci` this can be easily realized if you have your *GitHub Pages* activated in your account (either using public repositories or a **GitHub Pro** plan). See the public *GitHub Pages* for the template repository `fastib2pci` as an example: <https://dipftba.github.io/fastib2pci/>

7.5 Using CBA ItemBuilder Items with the IRTlib-Software



The IRTlib software is a dedicated deployment software for complex assessments based on CBA ItemBuilder items. Test compositions can be configured using a visual flow definition and used in offline and online assessments.

7.5.1 IRTlib-Editor vs. IRTlib-Player

The IRTlib Deployment Software consist of two different parts. An IRTlib-*Editor* to configure studies, and a IRTlib-*Player* to run studies. The IRTlib-*Player* requires a valid configuration of a deployment, that can be created, changed and modified with the IRTlib-*Editor*.

The IRTlib-*Player* supports different deployment modes for assessments, either of-line (e.g., from USB stick) or online (e.g., stand-alone with a URL pointing to a dedicated server prepared for hosting the software).

The IRTlib-*Editor* used to configure and define studies can currently be use only of-line.¹⁴ The identical assessment content (i.e., CBA ItemBuilder *Project Files* containing one or multiple tasks) can be used for online and offline delivery.

7.5.2 Runtime Requirements (Offline and Online)

An installation of the IRTlib-*Editor* is not necessary for offline use, because the software (i.e., the IRTlib-*Player* and -*Editor*) is prepared to run without installation (currently only under Windows).

Readiness Tool (offline): A simple command-line [readiness](#)-tool is provided to check

¹⁴The online version of the IRTlib-*Player* and *Editor* as *SaaS*-solution is under preparation.

if a (Windows) computer fulfills the requirements to run the IRTlib-*Player* for offline use. The readiness tool is configured with command-line parameters and, if so configured, also forwards the command-line parameters to the player. This allows the readiness-tool to always be started before the player, or to be called after the player only in the event of an error.

The IRTlib-*Player* for online deployments is provided as `docker` image. For preparation and configuration of online deployments, file management of CBA ItemBuilder project files and configuration files the offline IRTlib-*Editor* can be used.

Requirements (online): Unlike the CBA ItemBuilder runtime itself (see section 7.7) the IRTlib-*Player* for online deployments has additional browser requirements for online data collection. Online collections are possible in browsers that support the execution of WebAssembly (WASM).¹⁵

7.5.3 Study Configuration

Global configurations within the IRTlib-*Editor* concern the available CBA ItemBuilder *Runtimes*. Hence, before using the IRTlib-*Editor* for CBA ItemBuilder deployments it is necessary to add at least one *Runtime*. Moreover, for each study it needs to be configured, if either *Login*¹⁶ and *Password* or a access *Token* will be required.

Deliveries (i.e., *studies*) can be configured in the IRTlib-*Editor*. Studies need a unique *Name* and allow for an additional *Label*. Additional configurations for *Studies* include settings regarding the authentication of test-taker, the requested scaling of content, the configuration of a menu for test-administrators and the expected behavior, when a test-taker finishes a session. Finally, the IRTlib-*Editor* allows to define the *Routing* between *Test Parts*.

Each *Study* needs at least one *Test Part*, and currently two types of *Test Parts* are supported: CBA ItemBuilder and SurveyJS.

7.5.4 Configuration of CBA ItemBuilder Test Parts

To prepare a *Test Part* of type CBA ItemBuilder, the CBA ItemBuilder project files need to be added to the configuration using the IRTlib-*Editor*. CBA ItemBuilder *Project Files* (with *Tasks* and *Scope*) can be added at different sections of a *Test Part* configuration.



The basic building blocks for defining test assemblies are always combinations of CBA ItemBuilder *Project files*, *Tasks* and *Scopes*.

¹⁵ See the website [caniuse](https://caniuse.com/) for browsers supporting the use of WebAssemblies.

¹⁶ These example items for login ([IRTLibLoginExample.zip](#)) and exit ([IRTLibEndItemExample.zip](#)) can be used to design CBA ItemBuilder tasks for the IRTlib software.

After adding a CBA ItemBuilder *Project File* the *Tasks* defined in that *Project File* are available in the IRTlib-Editor. The additional *Scope* allows to use an identical *Task* multiple times. The section `info` allows to add items (i.e., CBA ItemBuilder *Project Files* with *Tasks* and *Scope*) to the following slots:

- *Prefix*: Items administered in the beginning of a *Test Part* (e.g., a cover page, an introduction or tutorial etc.)
- *Epilog*: Items administered at the end of a *Test Part* (e.g., a thank-you page etc.)
- *Timeout*: If the *Test Part* is time restricted, one or multiple items can be added that will be administered, if a *Timeout* occurs in the *Test Part*

The main items that should be administered between *Prefix* and *Epilog* can be added to the section `Items`. If the option *Routing* is not activated, these items are administered in the sequence defined in the view `Items`. If *Routing* is activated, the View *Routing* allows to define the sequence of items (i.e., CBA ItemBuilder *Project Files* with *Tasks* and *Scope*) using the *Visual Designer*.

7.5.5 Configuration of Item Pools

For the definition of sequences, results of concrete items (i.e., *Classes* with scoring conditions and *Result Texts*, see chapter 5) can be accessed directly. For adaptive tests, in which the current item (or the list of current items) is selected using IRT-methods (see section 2.7), the mapping of the scoring defined within the CBA ItemBuilder *Project Files* and the variable *u* (e.g., 0 for *no credit*, 1 for *full credit* and 0.5 for *partial credit*), the IRTlib-Editor provides the additional concept of an *Item Pool*.

7.5.6 Configuration of Codebooks

Each item (i.e., a particular *Task* in a CBA ItemBuilder *Project File* administered in a particular *Scope*) provides a number of result variables (i.e. *Classes*, see section 1.5.1) as part of the `ItemScore`, when the presentation ended for a particular task either because of a navigation-related command (see section 3.12.2) or because of a time limit (see section 7.2.8). The *Codebook* allows to map the values provided by the CBA ItemBuilder scoring into variables with labels and value labels, as required for data sets.

Missing Value Coding: If a response was provided, either the name of the scoring condition (hit/miss) or the *Result Text* represents the value of the variable. Within the tasks, the defined scoring (see section 5.3.11) is expected to differentiate between *Not Reached* (NR) and *Omitted Response* (OR), if no response is provided. Hence, the values provided for classes (i.e., variables) of *Tasks* (i.e., for one or multiple items) are missing value coded based on the CBA ItemBuilder scoring. However, there are a few processing steps required by the IRTlib-Player to apply the missing value coding completely:

- (1) If a timeout or an abort by the test administrator / interviewer occurs, the classes (i.e., variables) within the *current* task (i.e., the task in which the test part was aborted or the task that was ended by a timeout) the *Not Reached* (NR) codes are changed to either *Missing due to Abort* or *Missing due to Timeout*.
- (2) In the next step, the remaining tasks are determined that would have been administered if the timeout or the abort would not have happened. The classes (i.e., variables) of these tasks are then also missing-coded as either *Missing due to Abort* or *Missing due to Timeout*.
- (3) Finally, all remaining tasks are determined (i.e., the tasks that are neither administered before the timeout or the abort and would have also not been administered if no timeout or abort had occurred). The classes (i.e., variables) for these tasks are missing-coded as *Missing by Design*.

Taking all three steps together the resulting data set of the obtained item score can be transformed into a regular rectangular data set, with one column for each class (i.e., variable), where the number of classes (i.e., variables and thereby columns) is known in advance as soon as all pairs of CBA ItemBuilder projects files and task names are defined and the CBA ItemBuilder project files are available.



If the scoring definition *within* the CBA ItemBuilder *Tasks* is complete and error free (i.e., conditions are mutually exclusive per class, see section 5.3.3) and with missing value coding (i.e., the scoring defines missing responses for each class and differentiates, if required, between *Not Reached* and *Omitted Response*), the IRTlib software can provide result data that are ready for raw data archiving.

For test designs with a unit structure (i.e., multiple items within a task), a differentiation between *Not Reached* (NR) and *Omitted Responses* (OR) needs to be included in the scoring definition of the CBA ItemBuilder *Tasks*. The IRTlib software then uses this differentiation to differentiate between normal responses, *Missing by Design*, *Not Reached* (NR) / *Omitted Response* (OR), *Missing due to Abort*, and *Missing due to Timeout* even in the case of complex task-flows or test assemblies that may depend on given responses, conditions and skip rules or even so-called *pre-load* variables (i.e., values assigned in advance to logins or tokens or values provided at the start of an assessment, see section 7.5.3).

7.5.7 Data Collected by the IRTlib-Software

Digitally-based assessments with CBA ItemBuilder *Tasks* create results data (based on the scoring definition, see chapter 5), raw log events (automatically generated log events), and additional user-defined log events (triggered either in the finite-state machine or in content embedded using `ExternalPageFrames`).

Raw Data Archives: The data stored by the IRTlib-Player always have the reference to the so-called *Session*. If a login or token is used as *Person Identifier* to authenticate test-takers, the raw data of a session are stored in a file `{personidentifier.zip}`. If there was no login in a particular session (yet), or if the study is configured to use a random identifier, the raw data archive will be named according to this (random) ID. The *raw data archive* is a zip archive that can contain multiple files. The files `ItemScore.json` and `Trace.json` contain the item scores and log data provided by the CBA ItemBuilder Tasks in JSON format. The file `Log.json` gathers additional log data provided by the IRTlib-Player and the file `Snapshot.json` contains all information required to restore the state of a Task (used, for instance, for crash recovery).

Monitoring Data: After the completion of an individual assessment, summarized status information can also be provided directly by the IRTlib-Player. Using this functionality, it is possible to react to key measures (e.g., particular time measures) already during data collection without the need to process raw data archive containing the log or result data. The status information can either be written to a file (*monitoring file*) or passed in an online assessment as part of a forwarding URL. Computation of monitoring information is defined in the visual editor.

The monitoring data are created as variables in the visual editor and are stored as list of variable-value pairs. In the JSON file the variable-value pairs are stored in the following form:

```
{
  "ExampleDateTime": "2021-08-02T10:25:58.6209884+02:00",
  "ExampleInteger": 42,
  "ExampleString": "Any String",
  "ExampleDecimal": 3.1415926
}
```

Please note that the representation of the values in the JSON format differs slightly depending on the data type.

7.5.8 Integration into Learning Management Systems

If a hosting of the IRTlib software is configured, it can be used as tool in *Learning Management Systems* (LMS) that support the *Learning Tools Interoperability* (LTI) interface.

7.6 Using CBA ItemBuilder Items in SCORM Packages (with xAPI)



The repository [fastib2scorm](#) can be used as a tool to convert CBA ItemBuilder projects/tasks into SCORM packages that can then be integrated into *Learning Management Systems* (LMS). Single projects/tasks and linear sequences of projects/tasks are supported. The generated SCORM components can be used, for example, to embed CBA ItemBuilder items into [moodle](#).

Assessment content embedded into *Learning Management Systems* (LMS) can become *Open Educational Resources*. Open standards, such as the *Sharable Content Object Reference Model* (SCORM) described how content can be packaged into a transferable ZIP-archives, called *Package Interchange Format* to be used in different LMS that support SCORM.

Flavors of SCORM-Packages created by the `fastib2scorm-converter`

- SCORM 1.2
- SCORM 2004

7.6.1 Prepare CBA ItemBuilder Project Files for `fastib2scorm-Converter`

To create a SCORM packages with CBA ItemBuilder *Project Files*, the [fastib2scorm](#) can be used. It allows creating SCORM packages with either single or multiple *Project Files*. As a necessary prerequisite, one or multiple *Tasks* must be defined within each *Project Files*. If multiple *Project Files* are used, an alphabetical order is used. The same applies to the order of *Tasks*, if within one CBA ItemBuilder *Project File* multiple *Tasks* are defined.

7.6.2 Generating SCORM Packages using `fastib2scorm-Converter`

The converter `fastib2scorm` is provided as a *Github project template* that contains a so-called *CI/CD worker* (i.e., Github actions, see also 7.4.2). Using the converter requires an account at github.com. After creating an account and log-in to your profile, navigate to the repository `fastib2scorm` and push the button ‘Use this template’ (see also Figure 7.2 above for the similar approach used for the `fastib2pci-converter`).

7.6.3 General Data Provided to the LMS

SCORM packages that consist of a single CBA ItemBuilder task or a linear sequence of tasks automatically return the information summarized in Table 7.1 to the Learning Management System (LMS) without further configuration:

- **Completion:** If all tasks in a SCORM component are administered, the `cmi.completion_status` is reported as `completed`, otherwise either `incomplete` is reported (if any user interaction with the SCORM content were recorded) or `not attempted` (if the SCORM component was loaded, but no interactions were recorded).
- **Recent Task:** If a SCORM component is created with multiple CBA ItemBuilder *Tasks*, the recent *Task* name is reported as `cmi.core.lesson_location` (1.1 / 1.2) or `cmi.location` (2004 2st, 3nd, 4th) . If a SCORM component is resumed, the component is continued with this *Task*.
- **Progress:** If multiple CBA ItemBuilder *Tasks* are combined as SCORM component, the progress (i.e., the number of already completed *Tasks*) is reported as `cmi.progress_measure` (2004 2nd, 3rd, 4th).
- **Total Time and Session Time:** The total time a SCORM component was used (accumulated across multiple visits) is reported as `cmi.core.total_time` (1.1 / 1.2) or `cmi.total_time` (2004 2nd, 3rd, 4th). The time of the last session is reported as `cmi.session_time` (2004 2nd, 3rd, 4th).
- **Suspend Data:** The snapshot of started CBA ItemBuilder *Tasks* are required to resume the *Tasks*. If possible (i.e., if feasible within the restrictions of the SCORM format definition) the (compressed) *JSON-Snapshot* is provided as `cmi.suspend_data`. Note that the max size varies across SCORM versions (1.1 / 1.2: 4096 characters; 2004 2nd edition: 4000 characters; 2004 3rd / 4th edition: 64000 characters).

TABLE 7.1: Data Reported by all SCORM Components with CBA ItemBuilder Tasks

| Data Model | Description | Versions |
|--------------------------------------|---|--------------------------------|
| cmi.completion_status | Completion status, i.e., completed, incomplete, not attempted, unknown | (all) |
| cmi.core.lesson_location | Recent Task, i.e., the name of the last visited CBA ItemBuilder Project / Task used to resume | (1.1 / 1.2) |
| cmi.location | (see cmi.core.lesson_location) | (2004 2st, 3rd, 4th) |
| cmi.progress_measure | Value between 0 (0% and) and 1 (100%) indicating the progress within the component | (2004 2nd, 3rd, 4th) |
| cmi.core.total_time | Accumulated total time | (1.1 / 1.2) |
| cmi.total_time | (see cmi.total_time) | (2004 2nd, 3rd, 4th) |
| cmi.session_time | Time of the last session | (2004 2nd, 3rd, 4th) |
| cmi.suspend_data | JSON string to restore tasks states | (all, but varying size limits) |
| Not supported yet: cmi.core.exit | Exit status, i.e., time-out, suspend, logout | (1.1 / 1.2) |
| Not supported yet: cmi.exit | (see cmi.core.exit) | (2004 2nd, 3rd, 4th) |
| Not supported yet: cmi.core.entry | First attempt ab-initio OR resume | (1.1 / 1.2) |
| Not supported yet: cmi.entry | (see cmi.core.entry) | (2004 2nd, 3rd, 4th) |

7.6.4 Report Scoring Results Provided by CBA ItemBuilder Tasks to the LMS

The CBA ItemBuilder scoring (see Chapter 5) consists of a list of `Classes`, each providing one active hit (or miss) at a time, and optionally a string or numeric result (called *Result-Text*). Additionally, scoring is provided in the form of variables with values.

- **CBA ItemBuilder Scoring:** Raw results as provided by the CBA ItemBuilder Tasks are converted to the `cmi.interactions-structure`, defined in the SCORM standard, using the fields. For each class and each variable, a `cmi.interactions-entry` is created with a unique key (`id`) and a value (`learner_response`).

Classes:

- `id: {Project-Name}.{TaskName}.{ClassName}`
- `learner_response: Hitname | ResultText`

Variables:¹⁷

- `id: {Project-Name}.{TaskName}.{VariableName}`
- `learner_response: Type | VariableValue`

7.6.5 Mapping of Scoring Result to Indicate Success

The transmission of results-data from *SCORM* packages embedded in learning management systems in the form of `cmi.interactions` is sufficient to make all data available to the LMS for later use. However, it is not sufficient to report the results in a way, that the LMS can understand and feedback to teachers or course administrators. For that purpose, an additional mapping of the raw results to correct responses / incorrect responses is required, so that raw scores, success and credits can be derived.

TODO: We need to define a codebook structure for that purpose.

- **Raw Score:**
 - `cmi.core.score.raw`
 - `cmi.core.score.max`
 - `cmi.core.score.min`
- **Success:**
 - `cmi.success_status`
 - `cmi.core.credit`
 - `cmi.core.lesson_status`

TABLE 7.2: Data Reported by *SCORM* Components With Scoring

| Data Model | Description | Versions |
|---------------------------------|-------------|----------|
| <code>cmi.success_status</code> | ... | (all) |
| ... | ... | ... |

¹⁷Note: Variables are supported starting with CBA ItemBuilder 10.0 and it is expected that variable names and class names are unique.

7.6.6 Trace-Data using xAPI-Statements

Additional behavioral data gathered inside of SCORM package using the CBA ItemBuilder runtime can be stored using xAPI statements. The following statements are provided by default, storing the data provided by the CBA ItemBuilder runtime (see section 7.2.9):

- **Traces (Log-Data)** : JSON messages informing about log events inside the CBA ItemBuilder *Tasks* are provided as single xAPI statements, with the data provided in the *object* part:

```
{
  "actor": { "mbox": "mailto:user@example.com", "name": "User Name" },
  "verb": { "id": "https://example.com/verbs/logged", "display": { "en": "logged" } },
  "object": {
    "id": "http://example.com/system/events/123456",
    "definition": {
      "name": {
        "en": "CBA ItemBuilder Event Log"
      },
      "description": {
        "en": "Logged a CBA ItemBuilder event"
      }
    },
    "data": "... JSON data provided by the runtime ..."
  }
},
"timestamp": "2023-07-25T10:30:00Z"
}
```

- **Scoring Results** : JSON messages containing the scoring results of CBA ItemBuilder *Tasks* are provided as xAPI statements, with the data provided in the *result* part:

```
{
  "actor": { "mbox": "mailto:user@example.com", "name": "User Name" },
  "verb": { "id": "https://example.com/verbs/experienced", "display": { "en": "experienced" } },
  "object": {
    "id": "http://example.com/project-file/task",
    "definition": {
      "name": {
```

```

    "en": "CBA ItemBuilder Project Name"
  },
  "description": {
    "en": "User Name experienced Project Name / TaskName."
  }
  "result": {
    "extensions": {
      "https://xapi.itembuilder.de/extensions/itemscore": {
        ... ItemScore JSON ...
      }
    }
  }
}
},
"timestamp": "2023-07-25T10:30:00Z"
}

```

- **Snapshot** : JSON messages containing the complete restore data of CBA ItemBuilder *Tasks* are provided as xAPI statements, with the data provided in the *result* part:

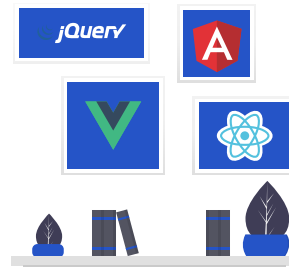
```

{
  "actor": { "mbox": "mailto:user@example.com", "name": "User Name" },
  "verb": { "id": "https://example.com/verbs/experienced", "display": { "en": "experienced" } },
  "object": {
    "id": "http://example.com/project-file/task",
    "definition": {
      "name": {
        "en": "CBA ItemBuilder Project Name"
      },
      "description": {
        "en": "User Name experienced Project Name / TaskName."
      }
    }
    "result": {
      "extensions": {
        "https://xapi.itembuilder.de/extensions/snapshot": {
          ... Snapshot JSON ...
        }
      }
    }
  }
},
}

```

```
"timestamp": "2023-07-25T10:30:00Z"  
}
```

7.7 Using CBA ItemBuilder Items in Custom Web Applications (Taskplayer API)



This section briefly describes how software developers can use CBA ItemBuilder content in web applications.



Important note: This section describes how CBA ItemBuilder projects can be embedded into *new* environments by technically experienced programmers, without using the existing deployment software tools described in this chapter.

The CBA ItemBuilder is the tool for creating individual assessment components. These can be items, instructions, units or entire tests. Typically, several CBA ItemBuilder projects must be used for the application. Each CBA ItemBuilder project file provides one or more entry points called *Tasks*. For a test section you then need a list of ItemBuilder project files and the corresponding task names to administer them, for instance, in a linear sequence.

CBA ItemBuilder *Project Files* are zip archives that contain the following components (see also section 8.3.3):

- A: The information required at design time for creating assessment components with the CBA ItemBuilder (i.e., for editing content). The files are only required for opening and modifying the assessment components with the CBA ItemBuilder and the files are not required at runtime (i.e., when using the assessment components to collect data).

- B: Resource files (i.e., images, videos, and audio files) in web-supported formats that are imported using the CBA ItemBuilder to design pages. The file names of resource files are linked in the CBA ItemBuilder to components (i.e., the resource files are required for item editing and at runtime).
- C: Embedded external resources (i.e., HTML, JavaScript, and CSS files also in web-supported formats) integrated into pages with `ExternalPageFrames / iframes` are stored inside the zip archive. An HTML file is defined for each `ExternalPageFrames / iframes` as entry, but more files might be necessary.
- D: A `config.json` that allows rendering the item content with the CBA ItemBuilder runtime is also stored in the zip archive. Only the `config.json` file and the two folders with the resources (`resources`) and the embedded resources (`external-resources`, that can contain sub-directories) are required for using the assessment components generated with the CBA ItemBuilder.
- E: A file `stimulus.json` is also part of the CBA ItemBuilder project files that contains JSON-serialized, meta information about the tasks, such as the runtime version (`runtimeCompatibilityVersion`), the name (`itemName`) and the preferred size (`itemWidth` and `itemHeight`) as well as a list of all defined *Tasks* (`tasks`). This file also contains a list of required resource files (`resources` and `externalResources`) that allows pre-caching the item before rendering.

Taskplayer API: The required *Runtime* to embed CBA ItemBuilder items into browser-based assessments is provided as a JavaScript file (`main.js`) and a CSS file (`main.css`) for each version of the CBA ItemBuilder. Since version 9.0 the interface of the *Taskplayer API* provided by the JavaScript runtime remained stable, while the internal implementation is changed and updated when new features are implemented in the CBA ItemBuilder. To render an CBA ItemBuilder project of a particular version using the `config.json` file together with the two folders (`resources` and `external-resources`), the same version of the CBA ItemBuilder runtime (i.e., `main.js` and `main.css`) is required.

For individual linear sequences, the runtime provides navigation between the tasks directly. If skip rules or adaptive tests are to be implemented, then several runtimes can be combined for the administration of individual tasks or packages of several tasks. This approach also allows implementing a delivery platform that can handle ItemBuilder tasks of different versions.

For programming a CBA ItemBuilder delivery, the following points must be considered and implemented:

- **Provision of Static Files:** To use CBA ItemBuilder items, the resources (directories `resources` and `external-resources`) must be made available (e.g. via static hosting). This can be done via arbitrary URLs, which are communicated via the configuration of the runtime.

- **Configuration:** Via URL parameters or with a structure `cba_runtime_config` declared in the global JavaScript scope (i.e. as `window.cba_runtime_config`) the runtime of the TaskPlayer API can be configured.
- **Caching of Snapshots:** Browsers can be closed, and assessments should be able to be continued afterward as unchanged as possible. Tasks can also be exited and revisited as part of between-task navigation. For these requirements, the runtime provides the state of a task as a so-called `snapshot`, which the delivery software is expected to store and to provide for restoring the state of tasks. Therefore, for implementing a custom delivery, it is required to enable persistence of the snapshot data because these snapshots have to be made available to the *TaskPlayer API* for resuming and restoring tasks.
- **Storing of Provided Data:** For a data collection with CBA ItemBuilder items using the *TaskPlayer API*, the following two types of data must be stored: At definable intervals, the *TaskPlayer API* transmits the collected log data (referred to as `trace logs`). These data have become the focus of scientific interest for the in-depth investigation of computer-based assessments and should always be stored. The direct results in (i.e., the so-called `Item Scores`) are provided by the *TaskPlayer API* when the *Tasks* are switched and must also be stored. Snapshots, trace data, and item scores are each assigned to a person-identifier and a task so that they can be easily post-processed afterward.

An description of an example implementation of an *Execution Environment* using the *TaskPlayer-API* is provided (see *EE4Basic* in section B.5) together with a technical documentation for developers (see *Reference* in section B.5).



8

Assessment Cycle and Workflows



Starting from a concrete diagnostic question (*Diagnostic Interest*), the development and implementation of computer-based assessments take place in a process that can be illustrated, for example, as an *Assessment Cycle* (see Figure 8.1).

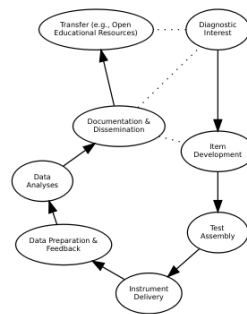


FIGURE 8.1: Illustration of an *Assessment Cycle* that includes *Transfer*.

The first part of this workflow of computer-based assessments (*Diagnostic Interest* and *Item Development*, see also, e.g., [Lane et al., 2015](#)) can be broken down into more detailed parts and steps:

- Overall Planning and Preparation (see section 8.1.1) and
- Item Development (see section 8.1.2).

If the intended use of assessment material, for instance, prepared using the CBA ItemBuilder is defined (i.e., if *Test Design* and *Test Assembly* are known), the distribution of content to *Project Files* can be optimized (see section 8.2).

A well-considered distribution of assessment content to individual project files can reduce effort and the risk of errors for the following steps:

- Testing (see section 8.4),
- Test Administration and Data Collection (section 8.5),
- Data Preparation, Reporting and Feedback (see section 8.6), and
- Documentation and Archiving (see section 8.7).

Finally, developed instruments can be shared and made available, for instance, as *Open Educational Resources* (OER, see section 8.7.4) to be used for further research or in (educational) practice. All different parts of a usual workflow for CBA projects as shown in the *Assessment Cycle* are described in this chapter.

8.1 Planning of CBA Projects



Assessment projects can face time pressure if necessary steps for planning and preparing have either not been considered or if timetables and milestone plans underestimated requirements for necessary steps. However, time pressure at the end might be less likely if a systematic approach is followed.

8.1.1 Overall Planing and Preparation

Table 8.1 lists the initial steps that should be taken before implementing specific items or tasks for a concrete computer-based assessment.

TABLE 8.1: Workflow for Overall Planing and Preparation

| |
|---|
| Overall Planing and Preparation |
| Domain Definition and Claims Statements |
| Content Specification |
| Feature Collection & Requirements |
| Software-Tool Selection |

Domain Definition and Claims Statements: As the first step of planning and preparing an assessment project, the construct domain to be tested needs to be defined and articulated. After naming and defining the domain to be measured, clear statements of the claims to be made about test-takers knowledge, skills and abilities (KSAs) are required.

Content Specification: The fundamental arguments that should be possible based on an assessment require validity-evidence based on the test content to support the specific interpretations and uses of test scores. This requires a precise content specification (i.e., test content and test format), including specifications on how the KSAs should be measured, which cognitive processes are required, and which item formats and response types are intended.

Feature Collection & Requirements: A systematic collection and documentation of all requirements that exist regarding item presentation and test delivery is suggested before selecting a particular assessment software. Planning for technology-based assessments (see, e.g., [International Test Commission and Association of Test Publishers, 2022](#)) also includes considering how the use of technology can directly impact the assessment experience.

What if the required functionalities and features of the assessment software and the requirements for test delivery, analyses, and implementation of the computer-based assessment still need to be precisely described? In that case, creating storyboards and implementing minimal examples (as described in the 8.1.2 section) could help.

Software-Tool Selection: Selecting software components for the different parts is possible based on the collected requirements. Various tools might be appropriate for implementing the actual items, the assessment delivery, and the data processing during or after the assessments. If different tools are used, their interplay poses another requirement.

Some aspects for the selection of software components are:

- Features of the software: Items of which type can be implemented using the assessment software (e.g., specific response formats, support of QTI items, items composed by multiple pages etc.)? Is response times measurement required with an accuracy of milliseconds, or is a web-based implementation appropriate?

- License to use the software: How can the software be used for different parts of the assessment project, including the actual data collection, archiving of the instruments, and the dissemination of the developed CBA instruments?
- Interoperability and vendor lock-in: How can the assessment content be used if key stakeholders or project partners change?
- Support and Service Level Agreement: Is technical support for implementing the items and conducting the data collections available, or can a specific agreement be concluded?
- Runtime requirements for offline delivery: Is test delivery without internet access possible and which devices and operating systems are supported? How is it ensured that testing can be carried out without interruption in the face of incorrect entries and that it can be continued after system failures?
- Requirements for online assessments: Bandwidth for group testing, hosting requirements, number of concurrent supported test-takers, redundancy and backup strategy, supported browser versions?
- Software integration: If developers are involved, are they required to implement the complete assessment or only parts (e.g., specific content embedded using `ExternalPageFrames`, see section 3.14, or the integration of CBA ItemBuilder items using the *TaskPlayer API*, see section 7.7)?

The personal abilities, resources, and skills of those involved in the project also play a not inconsiderable role in the success of CBA projects. Assessment projects often require competencies from different areas, which is an argument for interdisciplinary teams.



How to do this with the CBA ItemBuilder? Using the CBA ItemBuilder as an authoring tool, item authors (see section 2.11.1) should be enabled to create interactive items, tasks, and assessment components. The underlying argument here is that implementing item ideas and assessment concepts by content experts can create implementations that are superior to more traditional waterfall-like processes (complete description of requirements on paper in advance, followed by implementation by programmers).

Programmers and software developers are only needed in this process if specific extensions in the form of 'ExternalPageFrame' content are required or existing HTML5/JavaScript content is to be integrated. Psychometricians (e.g., for scaling and IRT-based item selection), and system administrators (e.g., for hosting online assessments on in-house servers), may be needed to complete a team.

8.1.2 From Idea to Implementation

Once the process model for item creation and software selection has been decided upon, the individual items (in cycles, if necessary) are implemented using the steps shown in Table 8.2.

TABLE 8.2: Workflow for Item Development

| Item Development |
|---|
| Item Story Boards and Item Writing |
| Minimal Examples and Item Computerization |
| Item Review and Modification |
| Scoring Definition and Scoring Testing |
| Item Tryouts (Cog-Labs / Field Testing) |
| Item Banking |

Storyboards: A first step for the creation of more complex computer-based items are *storyboards*, which illustrate in the form of sketches in which sequence information is to be presented and answer options are to be given. For diagnostically meaningful assessment components, particularly the behavior by which test-takers should provide information about their competence or ability is of particular importance, i.e., which behavior or actions should be used for evidence identification. According to the possibilities of computerized assessments to create interactive, authentic, and complex tasks (cf. Parshall, 2002), evidence identification does not have to include the work products exclusively. Still, it can also refer to features of the test-taking process (i.e., process indicators from the log data included in the measurement model).

Minimal Examples: Based on the initial ideas and storyboards, the functionalities and features required for designing the computer-based items can be derived. Typically, developing complex items to the end is not necessary to check whether a specific implementation is possible. Instead, so-called minimal examples, i.e., executable items that exclusively illustrate a particular functionality, can be used.



How to do this with the CBA ItemBuilder? Minimal examples illustrating features of the CBA ItemBuilder are provided via links included in the figure descriptions of this manual (see the links labeled *ib*, which give access to the individual CBA ItemBuilder projects shown in a particular figure).

Feature-Complete Prototype: Based on the division of content into pages, reused page components, and dialogs, designing a prototype is suggested that is as complete as possible and that at least fully maps navigation within assessment components (i.e., *within-unit* navigation). This step is not necessary if items are created based on an already developed and tested template.



How to do this with the CBA ItemBuilder? The CBA ItemBuilder supports the reuse of page templates, and existing projects can be adopted and modified as template project (see section 6.8).

Production of Audio files, Images and Videos: For the production of authentic tasks, simulation-based assessment and the contextualization of assessment content, images, audio, and video files are often required (see section 6.2). These must be created as accurately as possible and across tasks, with consistent font sizes, colors, etc., and saved at the required size.

Item Computerization: Combining and merging the visual material of items with potential possibilities for the test-taker's interactions (i.e., ways to respond to the assessment content) is a creative process that should result in opportunities to collect (valid) evidence about what test-takers know, can do, and understand. In other words, everything should be allowed that helps in making justifiable claims about KSAs.

In order to exploit the potential of computer-based testing for creating tasks that require specific construct-relevant test-taking behavior and that elicit informative evidence, two approaches are possible: A) Collaborative work in interdisciplinary teams (content experts and developers) and an iterative, agile approach for implementing, evaluating, and modifying computer-based tasks. B) Content experts learn and utilize tools to implement computer-based items directly, allowing them to play around with potential implementations and evaluate the impact on task characteristics and the interpretation of work products and test-taking processes.



How to do this with the CBA ItemBuilder? Within the functional scope of the CBA ItemBuilder object model, item authors can design interactive tasks without the help of programmers and optimize them with regard to diagnostic use (approach B). If HTML5/JavaScript content is developed and included as `ExternalPageFrame`, then feedback and review rounds are recommended (approach A).

Item Review and Modification: Tasks and computer-based implementations of items are usually not created in one step. Instead, assessment components are typically reviewed after an initial draft and revised in review loops to improve and optimize them step by step.



How to do this with the CBA ItemBuilder? The preview function can be used for the review process with CBA ItemBuilder items. For this purpose, the CBA ItemBuilder items may have to be shared between different actors, for which version management techniques (see section 8.3.2) can be used, for example.

Item Tryouts (Cog-Labs / Field Testing): After item development (and testing, see section 8.4), initial empirical testing in cognitive labs (so-called *coglabs*) or small-scale testing (e.g., with only one school class) is often helpful. Use cases for tryouts are to learn more about the comprehension and usability of new tasks or to (roughly) estimate the required processing time and task difficulty. The test deployment software described in chapter 7, for instance the R/Shiny package `ShinyItemBuilder` (see section 7.3), can be used to implement a tryout. If necessary, either a screen-recording software can be used to capture the detailed test-taking process, or the tryout can use the CBA ItemBuilder's feature of collecting replay-complete log data (see section 2.8.3).¹

Item Banking: The steps that individual items must go through describe a process from initial design, revisions, and tryouts to scaling and then the operational use of items in an automated or manual test assembly technique. At each stage, persons with different roles, such as item author, item reviewer, psychometrician, test developer, project manager, and others, can change an item's status in a pre-defined workflow. Possible workflows include the dissemination or archiving of operational items and the long-term storage of items required for follow-up assessments, subsequent cycles, or linking or bridge studies. Moreover, the role of persons and the defined workflow also determine which actions, such as commenting on an item, moving it to the next stage, or bringing an item back to a previous stage (or even discarding an item draft), are possible. Hence, instead of managing items in files (or CBA ItemBuilder project files) and metadata about items in spreadsheets, *Item Banking* using, for instance, web-based software is possible.

8.2 Distributing Content to Project Files and Tasks



The CBA ItemBuilder's flexibility for creating assessment components with multiple pages requires planning how the content should be distributed as either one or in multiple *Tasks* (i.e., entry-points) and *Project Files* (i.e., zip archives).

¹Note that content embedded as `ExternalPageFrame` is currently not included in the replay-completeness.

Assessments created with the CBA ItemBuilder are composed of *Tasks*. *Tasks* are stored in *Project Files* that share their resources (e.g., images, videos, audio files). All test deployment tools described in chapter 7 can be used to administer either a single *Task* or a linear sequence of *Tasks*. While this is sufficient for typical data collections using only one booklet or a set of fixed booklets, adaptive testing (including multi-stage tests) can require to analyze responses live during the assessment to select the appropriate subsequent *Task* (or multiple *Tasks* such as stages).



It is suggested to think about the distribution of assessment content into *Tasks* and *Project Files* before creating the CBA ItemBuilder projects.

Assessment material will be used as shown in Table 8.3 to collect data with a particular test design (i.e., implementing a particular test assembly strategy), using either manual or automated test assembly (see section 2.7) or booklets. Typically, in a calibration (or field trial) study, the first version of a newly developed test (i.e., a more extensive selection of items implemented, for instance, in CBA ItemBuilder project files) is administered. After investigating item properties (such as item fit, see section 2.5), items are slightly modified, and a selection of items is used to create the test(s) using the test assembly approach of choice.

TABLE 8.3: Steps for Test Design and Assembly

| |
|-----------------------------|
| Test Design and Assembly |
| Test Assembly Specification |
| Booklet Definition |

For many use cases, the following five rules are helpful in deciding how to distribute content to *Tasks* and how to distribute *Tasks* to *Project Files*:

1. All content that is always administered together should be in one task. For example, if the items that belong to a shared stimulus form a unit, then each unit should be created as a task.
2. Content that might be separated after a revision or item selection should be put into different tasks. This ensures that the CBA ItemBuilder tasks need as little revision as possible after a field test.
3. Tasks that refer to the same pages or resources should be placed in one CBA ItemBuilder project file. This avoids repeated copying of content (e.g., images, videos, etc.).
4. If information from an item is needed, for instance, for a subsequent filter or jump rule, then the items involved can, in the simplest case, be

placed in one task. In this way, the CBA ItemBuilder project files remain as independent as possible from the specific functions of the test delivery software.

5. Generally, the tasks (and the CBA ItemBuilder project files) should be as small as possible. This will save time inspecting and previewing the items and allow different persons to work on different parts of the assessment.

A more detailed, albeit complicated, description of the dependencies is summarized below:

Each CBA ItemBuilder *Task* will always contain at least one page with at least one item. However, multiple pages (and multiple items) within a single *Task* are possible. In order to guide the possibilities to optimize the distribution of items to *Tasks* and *Tasks* to *Project Files*, but also to discuss dependencies and potential limitations, the following section summarizes what needs to be considered when planning the use of *Project Files* with multiple *Tasks*.

Tasks: *Tasks* are the entry-points the test-deployment software can use (see section 3.6). The primary role of a *Task* is to define the first page (or the first page and an additional *X-Page*), shown after an assessment component has been loaded. Only one *Task* can be used at once. If multiple items should be visible simultaneously, the items need to be on the same page and shown within the identical *Task* (see section 2.4 for details about test design, item presentation and navigation).

Runtime Commands: *Runtime Commands* (see section 3.12) can be used to trigger action from the current *Task* to the test deployment software, for instance, to request a navigation to the next or the previous *Task*. Test-taker can trigger a *Runtime Command*, when the *Runtime Command* is attached to components (e.g., *Buttons*). *Runtime Commands* can also be triggered either by timers or by any component that can be linked to *Events* (i.e., *Runtime Commands* can be triggered as operators in rules defined in the *Finite-State Machine*, see section 4.4.6).

Pages: *Tasks* show either single pages (one at a time) or multiple pages simultaneous (either using *X-Page* layout, see section 3.4.2, as *Page Areas*, see section 3.5.4 or using dialog pages, see section 3.15). Each *Page* can be used in a *Task* multiple times and different *Tasks* within one *Project File* can share *Pages* (i.e., different *Tasks* can use the same *Pages*). *Links* (see section 3.11) and *Conditional Links* (see section 4.3) can be used to navigate between *Pages*.

Finite-State Machine: The internal logic layer of the CBA ItemBuilder (i.e., one or multiple *Finite-State Machine(s)*, see section 4.4) is defined for each CBA ItemBuilder *Project File* (i.e., multiple *Tasks*) share the identical *Finite-State Machine* definition(s). However, the *Task Initialization* syntax (see section 4.5) can be used to prepare the general *Finite-State Machine* definition for a particular *Task*.

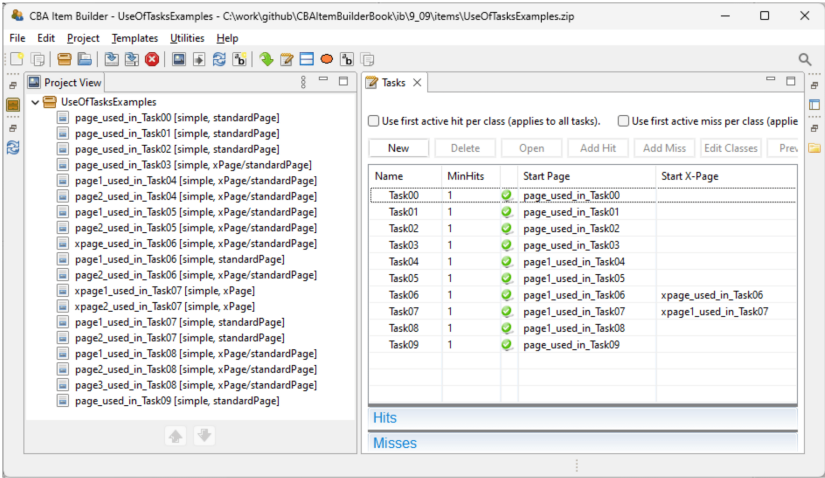
Variables: In addition to the finite-state machine definition, variables in CBA ItemBuilder *Project Files* are also globally defined and valid for all *Tasks*.

Summary: The use of multiple *Tasks* within one *Project Files* requires additional considerations to make sure, the *Tasks* can be used independently. Dependencies can arise based on links (see section 3.11) and conditional links, the *Finite-State Machine(s)* (see section) and *variables* (see section 4.2). If information or results within parts of an assessment needs to be shared, for instance, using *Variables* provided by the *Finite-State Machine*, these assessment components must be implemented with one *Tasks* (see, for instance, the examples provided in section 6.4.2).

Assessment content (i.e., items, units or clusters) distributed in booklet designs, for instance, used in balanced (incomplete) block designs, must be distributed into different *Tasks* to enable an test deployment software to assemble the tests as required. Similarly, for item-level adaptive testing or unit-level adaptive testing, the entities that are selected adaptively from an *Item Pool* must be implemented in separate *Tasks*.

Finally, if identical resources are used in different *Tasks* (for instance, audio and video files), the *Tasks* should be implemented in one *Project File* to reduce redundant files that need to be deployed (and maintained).

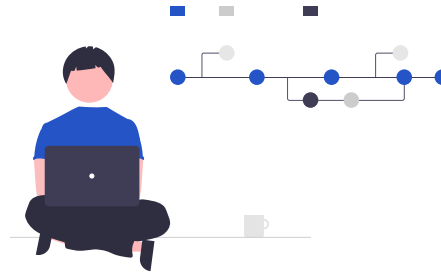
Use of CBA ItemBuilder Tasks (Task00)



Next (Task)

FIGURE 8.2: Example Illustrating the Distribution of Items to Tasks ([html](#)|[ib](#)).

8.3 IT-Management of CBA Projects



Assessment projects are often not created by one person alone. If a complete computer-based instrument is developed, content experts, psychometricians, and computer experts are involved. In research contexts, practical implementation of assessment components is often supported or even delegated to student assistants. Besides, when complex assessment software, such as the CBA ItemBuilder, is utilized, user-support may be involved. Additional content experts may be asked, for instance, to review the developed items or tasks. And when data collection takes place in the context of more extensive empirical studies, professional survey institutes or data collection agencies are sometimes involved and contribute to the overall success based on their diverse experience. This large number of participants and the many individual decisions at the content level and the technical implementation of computer-based assessments quickly become quite complex. For this reason, it is recommended to use project management software whenever possible, as briefly described in the following section.

8.3.1 Use of Project Management Software

First of all, personalized user accounts are needed to use project management software for preparing computer-based assessments in teams or groups. The user accounts might be organized using groups with different roles and permissions. Personal accounts that are not shared between users are prerequisites for efficiently dividing tasks between people and assigning changes to specific users.

To distinguish and structure different phases of creating, testing, piloting, and delivering computer-based assessments, project management tools provide the concept of *versions* or *milestones*. The actual work steps are then divided into parts (issues or tasks) and managed in an issue tracker. Each topic or ticket can then be assigned to a user and processed by them individually or assigned to other users for work sharing. Observers can be registered, informed about the progress of the processing of a task. Typically, issues or tasks can be structured hierarchically and combined into superordinate work packages. Project management software can be used

to process tasks using well-defined workflows (e.g., tickets of a particular category can be assigned to one of the states ‘new’, ‘in progress’, ‘review’, ‘feedback’, ‘solved’ and ‘closed’). Each ticket is typically dedicated to one separate topic, and the individual tickets can be assigned to milestones or versions. In that case, the progress within the issue tracker can be automatically used to create a roadmap that shows which work steps still need to be completed before the next milestone is reached. Finally, project management tools also provide assistance for knowledge management, for example, by providing Wiki pages or additional storage for documents or files.

Open-source tools that can be hosted on your own servers are, for instance, [Redmine](#) ([Lesyuk, 2013](#); [Pavic, 2016](#)), [OpenProject](#) or [Gitlab](#) ([Hethey, 2013](#); [Baarsen, 2014](#)). If the assessment content does not require special protection, public cloud solutions such as [Github](#) can also be used ([Tsitoara, 2020](#)).

8.3.2 Use of Version Control Software

First, the answer to the obvious question: Why is version control helpful for developing computer-based assessments? The explanation refers to the nature of computer-based assessments. Computer-based assessments are created using different software tools. A deployment software (see chapter 7) is used, for instance, together with a Web Browser or a browser component, to show the assessment to test-takers or participants. Moreover, the test content is either created with an authoring software for the test content (such as the CBA ItemBuilder) or is specifically programmed (i.e., implemented with a particular programming language). Altogether, all components together create the computer-based assessment, typically stored in multiple files. Accordingly, a tool that keeps track of all files and identifies differences between files and folders is useful for preparing computer-based assessments. Techniques that originated in software development has proven useful for managing and creating assessments. Version control software allows one or more users to manage and document the status of a set of files (referred to as a *repository*) and make changes traceable at the level of individual files. Moreover, a verbal description for changes in files is documented using so-called *commit*-messages.

Since item authors and users of the CBA ItemBuilder may not be familiar with the idea and concrete implementation of version management software, we describe two popular systems in more detail below. However, version control software is a also a key component in the context of Open Science and Reproducible Research ([Christensen et al., 2019](#)). The development environment for R (RStudio) for example comes with Git integration ([Gandrud, 2020](#)).

8.3.2.1 SVN / Subversion

Often, project management software can be configured already to support the creation of repositories for version management. A project can then be assigned one

or more repositories, each of which supports a specific version management technique. The use of two such methods will be described in some detail: Subversion (SVN) and Git.

Overview: Subversion (SVN) is less complex than GIT, which is why we describe this method first. Version management generally means that all files and documents belonging to a project are stored in a common *repository*. In the context of SVN, this repository is always on a server, i.e., usually accessible via a network. Only selected users have access to the repository. If SVN is used in combination with a project management tool, all registered users with the appropriate permission can access the repository. Every file is stored in the central repository in all created versions. Files can be added to the SVN repository using the commit operation. Check-out or update transfers or updates the files from the central repository to a local working copy (see Figure 8.3).

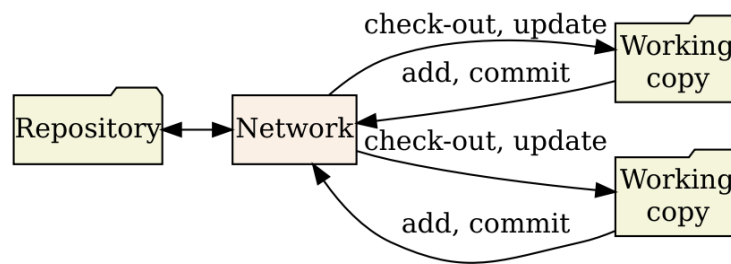


FIGURE 8.3: SVN: Repository and Working Copies (Mason, 2006).

All users of an SVN repository work with local copies and can adjust, modify, delete, and add new files (or folders) in the working copy of the repository. After completing a particular step (e.g., after changing the assessment according to a specific ticket from the issue tracker), the changes are submitted collectively to the repository (i.e., checked-in with the help of a *commit* command). For that purpose, the individual changes are described in a *commit* message so that it is traceable which changes were checked-in. Suppose the changes are related to a specific ticket. In that case, the ticket number can be specified in the *Commit* message, for example, to document the changes in a traceable manner.

Repository URL: A repository URL identifies SVN repositories. If a public domain is used, they are thus globally unique. For example, the repository URL could look like this: `https://{example-domain.org}/svn/{project-title}`. If the version management is integrated into a project management software, then the repository URL can usually also be retrieved there.

Revision: To manage the different versions, SVN uses the concept of *revisions*. A repository is always in a concrete revision (starting with zero). Each commit, i.e. all

changes to one or more files that are submitted at the same time, increases the revision number by one. The SVN repository stores the complete history of changes, i.e. for each file its content can be exactly determined (and if necessary also restored) to a specific revision. If you know the repository URL of an SVN and a specific revision number, then the content is also uniquely referenced.

Requirements: Many project management tools allow you to view a repository online in the browser and, for example, browse through revisions and, if necessary, follow the links to the issue tracker. However, client software is required to work with an SVN repository locally.



To use an SVN repository, client software is required.

Different software tools for the different operating systems can be used to work with SVN repositories. For Windows, for example, [TortoiseSVN](#) is quite widespread, a free edition of an SVN software for MacOS offers, for example, [SmartSVN](#).

Check-out: After installing an SVN client (e.g., [TortoiseSVN](#)), a working copy can be created locally for an existing repository. Even for a repository that is empty until then (i.e., revision 0), working with an SVN starts with a first *check-out*. The first check-out will create the working copy locally and connects the local folder to the SVN repository. If the repository is already filled with files and folders (i.e., in a revision greater than zero), then all files are downloaded and cached locally during the *check-out*. As soon as the check-out is completed (at a particular revision), it will be possible to work, modify, and, if necessary, even execute files in the working copy.



The working copy of an SVN repository is stored in a local directory.

Checking out an SVN repository works the same way if there are already files in the repository.

Commit: The local working copy of an SVN repository can be worked within the same way as any other directory. After an intermediate (completed) state is reached, the files can be committed to the repository via the *Commit* command. For this purpose, the SVN client displays the files that have been changed. The selection to be transferred can be made and described with a *commit-message*. Afterward, the changed files are transferred over the network, and the revision number of the SVN repository increases by one. Newly inserted or files that were not yet under version control must be added to the repository with *Add*.



After committing changes to an (SVN) repository, it is possible to track the history of changes without renaming the files. Since CBA ItemBuilder *Project Files* must not be renamed (see section 3.2.1), a version control system is recommended.

Save changes (for instance, in the CBA ItemBuilder) before committing files.

Update: As soon as more than two working copies are used (e.g., because several people are involved in the preparation of a computer-based assessment), the current status of a working copy may be out of date. For SVN repositories, this means, in the simplest case, that the current revision of the working copy is smaller than the most recent revision on the server (in the repository). If no files have been changed in the outdated working copy, a simple *update* can be used to update the working copy.

Check for Modifications: The question of whether files or directories in a local working copy have changed, been added, or deleted can be easily checked with the help of SVN. For this purpose, the function *Check for Modifications* is available, with the help of which a comparison of the working copy can be displayed with its current revision. This function can also be used to check whether an existing working copy still contains modified files that are not yet under version control.

Conflicts: As long as parallel changes in the SVN repository always affect different files, *Commit* and *Update* allow all users of the SVN repository to edit files in parallel and to share them using the repository. However, if two users make parallel changes to one or multiple identical files, so-called *conflicts* occurs. Conflicts can be related to files (file conflict) or to the directory tree (tree conflicts). Conflicts occur when executing the *update* command. If user A tries to commit a file that was changed and committed already by a different user B, SVN requests to update the working copy for user A if he or she tries to commit changes. Since the SVN repository is agnostic against its content, conflicts need to be resolved by users. With existing conflicts, no *commits* are possible. A graphical user interface (or the explorer integration of [TortoiseSVN](#) using the context menu) is of great value for resolving conflicts.

Advanced Features: A complete introduction to all features, options, and possibilities of version management with SVN would go beyond this book's scope. Therefore, only the keywords for selected advanced features will be mentioned and briefly explained in the following:

- **Merging:** SVN attempts to combine changes in the repository with local changes that have not yet been committed when updating. This process is called *merging*. If this does not succeed, a conflict occurs.
- **Ignored Files or Folders:** Files or directories that should not be part of the repository but are located within the working copy's directory can be excluded from the SVN. For instance, this function is useful if an assessment software that is part of the SVN, result data are written into a subdirectory, and test possible test data should be excluded from the repository.

- **Diff:** For files in text format (i.e., explicit text documents, but also files with program syntax and files in CSV, DAT, INI, XML, JSON, YAML, and similar formats), the difference from a previous version can be easily displayed directly in SVN. The display of differences (diff) is beneficial, especially when the commit-message is not meaningful. SVN client tools (such as [TortoiseSVN](#)) create diffs for other file formats. Unfortunately, the simple visualization of differences of non-text based file formats (i.e., images, but also ZIP archives and CBA ItemBuilder *project files*) is often not possible.
- **Revert:** In order to restore a previous state of the SVN repository, the *revert* function can be used to restore an earlier revision of the files and folders of the repository.

Version management using SVN can also use *tags* and *branches*, and has a concept for *locking* of files (see for more details, for instance, [Mason, 2006](#)).

Summary: Version control allows to prepare and develop computer-based assessment using multiple files in repositories. The critical benefit of using version control compared to other file sharing approaches (e.g., cloud storage) is that modification of files are documented (using commit messages) and that conflicts (i.e., modification of identical part of the repository by multiple users) are detected (and infrastructure to handle conflicts is provided). Moreover, the revision number (of SVN repositories) can be used to exactly define the version of all² files used for deployment of a computer-based assessment (i.e., for a particular data collection).

8.3.2.2 GIT

As a more modern alternative to SVN, the basics of version management with Git is now briefly described.

Overview: Git, unlike SVN, is a distributed version management system. This allow to use Git to manage different versions before pushing changes over a network to a remote repository. This two-step process of commits adds some complexities compared to SVN. However, it allows using Git (i.e., to perform almost all operations) locally.

Repository URL: For GIT, the remote repository is addressed via a URL. How the URL looks exactly depends on the way of communication with the server. Possible protocols are `https` and `ssh`.

Commit Hash: Instead of an incremental revision number (that is used by SVN), each commit in git is identified by an `sha1` hash. To identify a commit, the hash shortened from 40 characters to 6-8 characters is often displayed in the git history. Instead of taking the largest revision number, git uses a `HEAD` as a named pointer to a specific commit, representing the current commit (of a given branch).

Requirements: Various cloud services (e.g., github) and project management tools

²Not that file *hashes* are typically only used for single files.

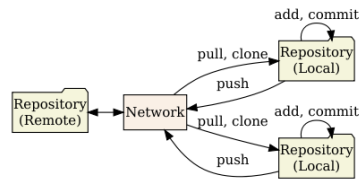


FIGURE 8.4: GIT: Remote Repository and Local Repositories.

offer the possibility of creating git repositories and viewing them in the browser. Files can often also be edited or uploaded in the browser and changed directly in the repository via commit.



To use git, client software is required.

Git clients for all platforms can be downloaded from <https://git-scm.com>. Git clients are directly integrated in a number of tools (e.g., RStudio) and there are graphical tools for git (e.g. GitHub Desktop, SourceTree, and many more) as well as a Windows Explorer integration ([TortoiseGit](#)).

A full introduction to git is not necessary for organizing assessment projects and is beyond the scope of this chapter (see, e.g., [Tsitoara, 2020](#)). In the following, only the basic steps necessary to use git without branches to manage files will be described.

Clone: Before working with files in a local working copy of a git repository, a copy of the repository is required. This can be created for empty and already used repositories via the *Clone* command. Compared to SVN (where *checkout* was used for this step), git uses *clone* to download not only the current commit (HEAD) but also all previous versions of files and all changes in the local repository.

Staged Files: Files within the working copy are, analogous to SVN, not automatically part of the repository. For that they have to be added with *Add*. Git then differentiates between the states for files shown in [Figure 8.5](#).

New files are initially ignored by git, i.e. their contents are *untracked*. When a file is added, git marks it as *staged*. A snapshot of the *staged* files can be created in a git repository via *commit*. After that the state of the compressed files is *unmodified* until they are edited or changed. Then they are marked as *modified*. Before edited files with the status *modified* can become *unmodified* files again via *Commit*, they undergo the status of *staged* files again. *Unmodified* files can be removed from the repository (i.e., marked as *untracked*).

Commit: Tracked files that have been modified or added (i.e. files in the *staged* status,

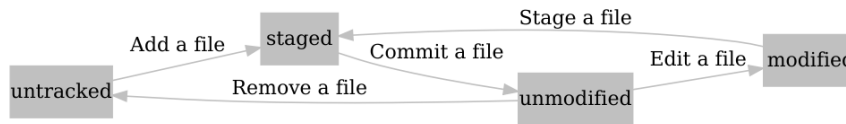


FIGURE 8.5: Lifecycle of file status in git.

see figure 8.5) can be commented. Graphical tools for git often show the files in *staged* state or allow to stage all files that have been modified by a simple selection.



After committing changes to a git repository, it is possible to track the history of changes without renaming the files. Different versions of files stored in git repositories can be used locally.

Analogous to SVN, a commit message is required for each commit, which is then used in the git history to describe the changes. The snapshot of the files from the working copy created with the help of a `commit` is marked with a hash at git and stored in the local repository.

Push: After committing files the additional `push` command is required to transfer the commit, which is made on a local branch of the git repository to a remote repository (see Figure 8.4).

Fetch: When multiple users push to a git repository, commits made by another user can be retrieved with `Fetch`. This makes the repository aware of changes, but `Fetch` does not yet integrate them into the local working copy.

Pull: Only with the command `Pull` will changes pushed to the repository by other users be downloaded and copied to the local working copy. If conflicts occur, these must be resolved and solved via a *merge commit*. Git provides the options `use ours` and `use theirs` for this.

Additional Features: The git tool, popular in software development, goes far beyond the features and functionality needed to manage (binary) assessment components in the form of CBA ItemBuilder *Project Files*.

- **Tagging:** Since each commit is only marked with a hash, this is not well suited for naming a specific version (e.g. the tested final version of an assessment). For this purpose, the option of `tagging` can be used in git, where a concrete state of a repository is named with a (readable) label (e.g. `v1.0`).

- **Branches:** Git has a sophisticated concept for *Branches*, i.e. for the division into several areas, in which files with the same name can have a different status. For example, when changes are made by different users at the same time, git automatically creates *Branches*. *Branches* can also be used to test and develop changes in a protected section, while the main section remains usable. Often, *Branches* are used systematically with git, for instance, whenever a new feature is to be developed and tested. A popular strategy for this approach is *git flow*.
- **Revert and Reset:** Git provides several ways to access previous commits in a repository. Especially when several users work in a common git repository, it is necessary to choose carefully here. In order to track the history of CBA ItemBuilder *Project Files* and to revert to a previous version if necessary, it is often sufficient to display the git history in the browser if the changes were committed to the central repository via 'push'. *Project Files* in earlier versions can then be downloaded and reused if changes are to be discarded.
- Similar to SVN, git can also ignore individual files or files of a certain type or in a certain directory via the `.gitignore` file.

Summary: Git is a complex and powerful version control system whose basic features can also be used to manage assessment projects. It is superior to SVN for this task if the versions are not stored in a central repository and are to be managed locally even without a network connection.

8.3.3 Working with *Project Files* as ZIP Archives



CBA ItemBuilder *Project Files* contain information that can be read manually or automatically from ZIP archives. The ZIP archive is not changed in the process.

Extract CBA ItemBuilder Version: Each CBA ItemBuilder file contains a file `{Project-Name}.json`. In this JSON file, which can be read with a text editor, the supported version of the CBA ItemBuilder (`runtimeCompatibilityVersion`) is directly in the first line.

Extract Scoring Information: Suppose many CBA ItemBuilder files are to be tested automatically and integrated into delivery software. In that case, it is a good idea to automatically check the transfer of result data (i.e., the scoring). For this purpose, the scoring can be read out automatically from the JSON file included in the CBA ItemBuilder project files.³

Reading Metadata: CBA ItemBuilder *Project Files* contain metadata for describing

³See [here](#) for an example.

the content (see section 6.3.4). Metadata can be found in the file `metadata.xml`. This XML-file following the [Dublin Core](#) specification can be extracted from the ZIP archives.

TABLE 8.4: Content of CBA ItemBuilder *Project Files* required at *Runtime*

| Folder/File | Description |
|----------------------------------|--|
| <code>stimulus.json</code> | Metadata and information about the tasks |
| <code>config.json</code> | Runtime definition of the tasks |
| <code>resources/</code> | Resource files required for the task |
| <code>external-resources/</code> | External resources required for the task |

Runtime Code: The CBA ItemBuilder *Project Files* fulfill two functions. They allow modifying, editing, and previewing of items using the CBA ItemBuilder desktop program. In addition, they can be used with existing deployment software (see, for example, section 7.5) or by programmers using the TaskPlayer API (see section 7.7) to execute assessments. Files and directories required at runtime are listed in Table 8.4. Files and directories required for editing with the CBA ItemBuilder are summarized in Table 8.5.

TABLE 8.5: Content of CBA ItemBuilder *Project Files* required at *Design-Time*

| Folder/File | Description |
|--|---|
| <code>metadata.xml</code> | Metadata defined in the <i>Project File</i> (see section 6.3.4) |
| <code>internal.json</code> | Internal project information used by the CBA ItemBuilder |
| <code>global_1_1.xlf</code> | XML file containing texts for translation in XLIIF-format |
| <code>global.cbavaluemap</code> | XML file containing the definition of <i>Value Maps</i> |
| <code>global.cbaitemscore</code> | Scoring definition (with reference to <code>dsl</code> -file in folder <code>scoringResources/</code>) |
| <code>scoringResources/</code> | Folder with <code>dsl</code> -files containing <i>Scoring Conditions</i> |
| <code>global.cbavariabes</code> | XML file containing the definition of <i>Variables</i> |
| <code>global.cbalayoutsettings</code> | XML file containing the layout definition for <i>Tasks</i> |
| <code>conditionFiles/</code> | Folder with <code>dsl</code> -files containing <i>Conditional Link Conditions</i> |
| <code>global.emfstatemachine</code> | XML file containing the <i>State</i> definitions |
| <code>statemachineFiles/</code> | Folder with <code>dsl</code> -files containing <i>Finite-State Machine</i> syntax |
| <code>{page}.cbaml/.cbaml_diagram</code> | Page definition edited with the CBA ItemBuilder for each <code>{page}</code> |
| <code>project.properties</code> | Global properties of the <i>Project File</i> |
| <code>.project</code> | (Can be ignored.) |



Editing *Project Files* outside of the CBA ItemBuilder as ZIP archives can easily destroy the files. Make changes only to files in the directories `resources` and `external-resources` and make backup copies in any case, or use version management (see section 8.3.2).

Replace Resource Files: Images, videos and audio files added to CBA ItemBuilder Project Files via the *Resource Browser* (see section 3.10.1) are included in the ZIP archives in the sub-directory `resources`. If the file names (incl. upper and lower case) and the file formats (incl. the file extension) remain identical, resource files can be exchanged, modified, and updated even without the CBA ItemBuilder in the ZIP archives. The resolution (pixel width times height) for image and video files should remain the same to guarantee that the resources are appropriately rendered during runtime.⁴

Add ExternalPageFrame Resources: If in CBA ItemBuilder *Project Files* content is inserted as `ExternalPageFrame`, then the files are included unchanged in the directory `external-resources` of the ZIP archive. Content can be added to ZIP archives (i.e., CBA ItemBuilder *Project Files*) using the *Embedded HTML Explorer* (see section 3.14.2). As long as the *Page Address* (see Figure 3.152), i.e., the file which is directly included by a component of type `ExternalPageFrame` remains identical, the external resources in the directory `external-resources` can also be updated, added or inserted directly in the ZIP archive.

Edit Value Maps: Defining complex *Value Maps* using the editor provided by the CBA ItemBuilder (see section 4.2.4) can be cumbersome. The definition of *Value Maps* is stored in the file `global.cbavaluemap` inside of the CBA ItemBuilder *Project Files*. The following XML shows the content of the file `global.cbavaluemap` used for the example item shown in Figure 4.14 (see section 4.2.4).

```
<?xml version="1.0" encoding="UTF-8"?>
<valuemap:ValueMapper xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:valuemap="http://valuemap.softcon.de" version="VERSION_01_01">
  <valueMaps xsi:type="valuemap:DiscreteValueMap" name="M_Example">
    <entries guard="1" text="Red" image="1.fw.png"
      audio="red.mp3" video="red.mp4"/>
    <entries guard="2" text="Red-Yellow" image="2.fw.png"
      audio="red-yellow.mp3" video="original.ogv"/>
    <entries guard="3" text="Green" image="3.fw.png"
      audio="green.mp3" video="green.mp4"/>
    <entries guard="4" text="Yellow" image="4.fw.png"/>
```

⁴An example for adjusting the audio volume of embedded audio files can be found [here](#).

```

                                audio="yellow.mp3" video="yellow.mp4"/>
    </valueMaps>
    </valuemap:ValueMapper>

```

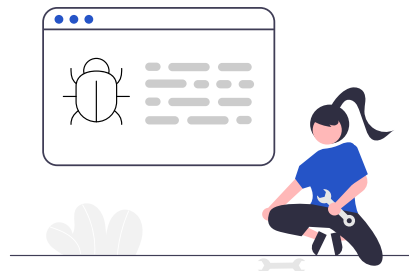
As long as the structure of the XML file remains valid and all resources exist (i.e., images, audio and video files mentioned in the XML attributes are included in the *Project File*), the file `global.cbavaluemap` can be extracted from the ZIP archive, edited in a text editor (or XML editor) and copied in the ZIP archive again. To make sure the CBA ItemBuilder *Project File* remains valid, open the file in the CBA ItemBuilder and preview all *Tasks* after changing the file `global.cbavaluemap`.

8.3.4 Use of Continuous Integration/Continuous Delivery

Continuous Integration (CI) and *Continuous Delivery* (CD) refers to techniques developed in software engineering, for automatically updating software environments enforcing automation in building, testing and the deployment of applications. CI/CD is organized in so-called *Pipelines*, which are code scripts executed based on triggers, such as *Push* commands in git repositories (see section 8.3.2.2).

Content created as *Project Files* with the CBA ItemBuilder can be used together with the TaskPlayer API (see section 7.7) in CI/CD-pipelines. The converter described for the integration of CBA ItemBuilder items as PCI components into TAO (see section 7.4) is based on a [Github](#) workflow. The pipeline that is shared as a [Github](#) template project (see the project [fastib2pci](#)) is configured to automatically build the PCI components using CBA ItemBuilder *Project Files* committed to the corresponding git repository.

8.4 Testing CBA Projects



A downside of the many advantages of computer-based assessments (see section 2.1

is that the more interactive (and innovative) assessment items are, the more sensitive the created content is to errors and potential glitches. Potential issues that require (sometimes) intensive testing can be functional (e.g., a missing `NEXT_TASK` command, see section 3.12.1), or affect only the layout and presentation of tasks (see section 6.8.5 for tips).

Moreover, since different software components and multiple steps are usually involved in generating assessment content, finding errors and testing assessment projects can be complex, requiring a systematic approach. The most crucial part of implementing CBA assessment projects is understanding and developing a notion of what needs to be tested and when tests may need to be repeated if specific changes have been made to a test delivery or the components used.

TABLE 8.6: Steps for Testing

| |
|--|
| Testing |
| Cross-Browser Testing (when required) |
| Component Testing / Scoring Testing |
| Integration Testing / Data Storage Testing |

8.4.1 Testing Cross-Browser Compatibility

The items created with the CBA ItemBuilder are displayed in the browser with the help of a runtime (see TaskPlayer API in section 7.7) and, if necessary, other additional software components of the delivery software. In the current version, the CBA ItemBuilder runtime is implemented in JavaScript based on the React framework and tested with the CBA ItemBuilder in the current browsers at the time of release. However, additional functionality provided by the used browser or the used browser component is always necessary to display the items. Browsers are subject to continuous development and change, and several differences exist between browsers on devices with different operating systems.

The dedicated testing of CBA ItemBuilder content in different browsers is necessary if A) browsers are used which had a low penetration at the time of the release of the CBA ItemBuilder version (e.g., browsers on specific devices like SmartTVs) or were still unknown (e.g., newer browser versions after the release of a specific CBA ItemBuilder version/runtime). Cross-browser testing is B) also necessary if any content is included via the `ExternalPageFrame` interface that was either implemented specifically for an assessment or has not yet been tested in the intended browsers.

Running Preview in Specific Browser: The assessment figures created with the CBA ItemBuilder can be viewed in the browser directly from the authoring tool (see section 1.4). By default, the system browser is used, i.e., the browser registered *Web browser* on the local computer used to run the CBA ItemBuilder. After a preview has been started, the URL opened in the default browser can be opened in other browsers

if they are installed locally on the computer. In this way, assessment content can be viewed and checked in different browsers.⁵

Using External Tools for Cross-Browser Testing: Since CBA ItemBuilder created assessment components are generated as HTML output for integration into test deliveries, tools for testing websites in different browsers can also be used to verify cross-browser compatibility.

Key technologies for automating website testing such as [Selenium](#) or [PlayWright](#), and the various solutions for automated website testing on various devices can be used to test cross-browser comparability. Cross-browser testing is suggested in particular when external components are embedded into CBA ItemBuilder project files using the `ExternalPageFrame`-component (see sections 3.14 and 4.6.2), or if CSS adaptations are used (see section 6.3.3).

8.4.2 Testing Assessments Using Synthetic Data

For the further test steps shown in table 8.6, it has proven helpful to consider the structure of assessment projects. Assessments generally consist of many individual components (items and units, as well as instructional components like tutorials). The Components are administered either in a fixed sequence (see *Fixed Form Testing* in section 2.7.1), in different booklets or rotations (see section 2.7.2) or in individualized sequences (see *Multi-Stage Testing* in section 2.7.3 and *Computerized-Adaptive Testing* in section 2.7.4).

An essential first step is *Component Testing* to ensure that test cases systematically cover the specific conditions of all components. This means that the individual components (i.e., items or units, instruction pages, tutorials, etc.) are tested separately.

Component Testing / Scoring Testing: Component testing regarding the behavior of items can be combined with scoring testing of individual items. Depending on the complexity of the evidence identification rules used to define the scoring, scoring testing might be trivial (for instance, to make sure that the selection of radio buttons is appropriately captured). However, it can get more complex if, for instance, multiple pages are used and the missing-value coding is included in the scoring definition.



How to do this with the CBA ItemBuilder? The item authors can best test CBA ItemBuilder tasks in scoring and functioning if they have also defined the rules for evidence identification and evaluating responses and concrete test-taker behavior. For this reason, the CBA ItemBuilder provides a possibility with the *Scoring-Debug Window* (see, for instance, section 5.1), how component tests of single *Tasks* can be

⁵The regular URL is `http://127.0.0.1:7070/app/`. In this URL 127.0.0.1 is the local IP address, 7070 is the *Port* used by the CBA ItemBuilder for any preview (see appendix B.4).

executed directly from the preview and repeated several times by *Reload* of the preview. If necessary, the scoring can also be checked by persons who have not dealt with the CBA ItemBuilder in depth (see the *Quick Start: Explore Scoring* in section 1.5).

To test the scoring, the individual assessment components must first be identified. For each item, all the different correct and incorrect solutions should be systematically entered, produced, and checked. If several items or ratings are contained within a component, it is recommended to check to what extent they are independent. If the defined scoring rules reveal dependencies, then the scoring check of the individual components should also consider all combinations, as far as possible, with reasonable total effort.



How to do this with the CBA ItemBuilder? *Component Testing* of CBA ItemBuilder content becomes more straightforward if multiple small tasks are created and stored in (independent) *Project Files* (see section 8.2).

For organizing the testing of CBA projects, it is recommended to use a version control system and to organize the process using an issue tracker (see section 8.3).

Integration Testing / Data Storage Testing: Data storage (typically for result data) uses the single assessment components integrated into the test deployment software. A systematic approach is based on *Click Pattern* (also called *Mock Cases*), meaning pre-defined responses to all items in a test or booklet. To verify the entered responses (synthetic data) with the results collected by the assessment software, the data post-processing (see section 8.6) should be in place (end-to-end testing).

For pragmatic reasons, the use of screen recording software (e.g., *OBS Studio*) to check the scoring may also prove helpful. If the screen is continuously recorded during the scoring check, then possible input errors can be identified more quickly in the event of inconsistencies.

Missing or skipped responses, time constraints, timeouts, and special test events, such as test leader interventions or test terminations at defined points, should be included in the mock cases so that missing coding can be checked later.



How to do this with the CBA ItemBuilder? In the current version of the CBA ItemBuilder, *Classes* are used within the *Project Files* to define result variables whose values are either categorical (hit or miss *Scoring Conditions*) or non-categorical (when the `result-text()`-operator is used). In addition, the delivery software can use a

codebook to translate them into variables (with customized variable names and variable labels) and, for categorical variables, with (newly defined) variable values (e.g., of type integer) and with additional value labels.

Please note that if item selection is also dependent on random processes in the context of adaptive testing, for instance, as part of exposure control, then the algorithms for item selection must be tested as an additional step. Testing adaptive algorithms is done, for instance, in pre-operational simulation studies, if the algorithms used for operational testing are accessible for simulation studies as well.

Verification of Log Events: As described in the section 2.8, the theory-driven collection of log data is increasingly important. Various possible process indicators can be extracted from log data, providing information about emotional and cognitive processes during test and item processing. Log data thus provide the basis for a possible improvement in the interpretation of test scores and, if use is planned, should be reviewed before an assessment is conducted. When verifying and checking log data, special attention should be paid to the fact that log events depend on their context (*Contextual Dependency of Log Events*). Therefore, verifying and checking log events may require a reconstruction of the context from previous log events.



How to do this with the CBA ItemBuilder? Inspection and verification of log events is possible in the state of *Component Tests* within the CBA ItemBuilder *Preview* using the build-in *Trace-Debug Window* (see section 1.6). Moreover, the data post-processing workflows (see section 8.6 available for the different deployment approaches also allow inspecting the log data as part of the testing procedures of CBA projects.

8.5 Running Assessments

After the preparation and testing of assessment projects, the actual data collection (fieldwork) takes place. The data collection can be released with a specific revision or tagged version status if a software tool is used to version the assessment content (see section 8.3.2). Suppose this revision number or version information is also stored in the survey data. In that case, it can be traced which exact version was used for a test-taker, even in the case of longer data collection phases and possible adjustments during the field time.

TABLE 8.7: Steps for Test Administration / Data Collection

| |
|---------------------------------------|
| Test Administration / Data Collection |
| User Management / Authentication |
| Test Deployment |

The steps summarized in table 8.7 start with *User Management / Authentication*. If assessments are embedded into other (digital) environments or (longitudinal) designs, information might be linked to the identifiers used for test-takers authentication (e.g., as log-in or token). These so-called *pre-load* variables need to be handled by the test deployment software (see, for instance, section 7.5). Special focus is also on identifier used for persons (user management / authentication), as these identifiers might need to be replaced during data post-processing.

While the assessment is running typical using one or multiple (mixed-mode) delivery modes described in section 7.2.1, *intermediate data* might be available to start data processing while the data collection is running (for instance, by providing the *Raw Data Archives* of already completed assessments incrementally). If CBA is integrated into a computerized survey with even more components (e.g., questionnaires or interviews), then selected data from the assessment can also be taken over directly for field monitoring (e.g., in the form of a monitoring file, see section 7.5.7).

8.6 Data Processing after Assessments

The data preparation process should be already tested as part of the *Integration Testing* (see section 8.4.2). For this purpose, the required routines (e.g., R scripts) should already have been created prior to data collection and tested with the help of synthetic data. Testing is complete if it is verified that the central information can be derived from completing tasks required for identifying evidence about the test-taker's knowledge, skills, and abilities.

TABLE 8.8: Steps for Data Preparation / Reporting / Feedback

| |
|---|
| Data Preparation / Reporting / Feedback |
| Data Preparation |
| Coding of Open-Ended Responses |
| Final Scoring / Cut Scores / Test Reports |
| Data Set Generation / Data Dissemination |



How to do this with the CBA ItemBuilder? The simplest case for *Data Preparation* of assessment projects using the CBA ItemBuilder are studies in which the *Codebook* and *Missing Value Coding* are already included in the deployment software. If all instruments are implemented using CBA ItemBuilder content and if no `External-PageFrame` content is used, the required data preparation for result data boils down to combining data stored in case-wise individual *raw data archives* into a data set in the desired file format.

Data Preparation: Data preparation can begin during data collection if intermediate data are provided or made available. Typically, the data is generated in smaller units (i.e., sessions) in which a test-taker processes a set of tasks compiled for him or assigned to him via pre-load information. The data on a test taker, as provided by the assessment software, can be understood as a *Raw Data Archive*. Analogous to the scans of paper test booklets, these *Raw Data Archives* (for example, combined as a ZIP archive) are the starting point for data preparation. If raw data from computer-based assessments must be archived in terms of good scientific practice, then this can be understood as the requirement for long-term storage of the *Raw Data Archives*.

A first step often required to describe the collected data as pseudonymized or anonymized is the exchange of the person identifiers (ID change) that were used during data collection. Person identifiers might be used as the file name of the *Raw Data Archives* and might be included in several places. Since the *Raw Data Archives* should not be changed after data collection, the data processing means extracting the relevant information from the *Raw Data Archives* and changing the person identifier in the extracted result data and the extracted log data.

Approaches known for *Open Science* and *Reproducible research* (Gandrud, 2020) should be used (i.e., using scripts that are maintained under version control), to allow re-running the complete data preparation starting from the *Raw Data Archives* to the final data sets. If the data preparation is carried out entirely with the help of scripts (e.g., using R), later adjustments are more straightforward. Possible adjustments include deletion requests for the data of individual test-takers, which might otherwise be cumbersome if, for example, a large number of data sets is created due to the collected log data (see section 2.8).



How to do this with the CBA ItemBuilder? For automatic and script-based processing of data collected with the CBA ItemBuilder and selected delivery software, the readout of raw data archives can be automated with the R package `LogFSM` (see section 2.8.5).

Coding of Open-Ended Responses: Operators described in chapter 5 for the CBA ItemBuilder for evaluating so-called *Open-Ended Answers* are currently limited. Open-ended answers (such as text answers) can only be scored automatically to a

minimal extent (in the CBA ItemBuilder, only with the help of regular expressions). More modern methods of evaluating open-text responses using natural language processing methods [NLP; see, for instance, [Zehner et al. \(2016\)](#)] might require a two-step procedure. Training data are collected in the first step and not evaluated live during test-taking. Afterward, classifiers are trained based on NLP language models or adapted in the form of fine-tuning. Once such classifiers are available, the answers can be automatically evaluated by test-takers and transferred to the data set.

A similar procedure applies to graphical answer formats (e.g., when an `ExternalPageFrame` allows test takers to create a drawing for their answer). For the creation of training data as preparation of an automatic coding or if answers are to be evaluated exclusively humanly, the open answer must be extracted from the *raw data archives* for an evaluation process (*Human Scoring*).



How to do this with the CBA ItemBuilder? Using the data collected during runtime (in particular using the so-called *Snapshot*) the `TaskPlayer` API can be used to restore the item in exactly the state in which the item was left by the test-taker. This allows to build solutions for human coding of open responses. Note that if content is embedded using `ExternalPageFrames`, the JavaScript/HTML5 content embedded into CBA ItemBuilder items must implement the `getState()/setState()`-interface to collect the state of the `ExternalPageFrames` on exit and to allow to restore the content for scoring purposes (rating).

Final Scoring: The decision of whether items already score the responses (scoring) or whether only the raw responses (i.e., the selected items, entered texts, etc.) are collected at runtime is made differently for different assessments. As long as the responses are not needed for filtering or ability estimation (see section 2.7), there is no critical reason why scoring should not be performed as part of post-processing. Only if created assessment content is shared (see section 8.7.3) is it helpful to define the scoring, for instance, directly within the CBA ItemBuilder *Project Files* (i.e., the files to be shared), because this way, items are automatically shared with the appropriate scoring.

Cut Scores and Item Parameters: Even if the scoring, i.e., for example, the mapping of a selection to a scoring (correct, wrong, partially correct), can be part of the item (i.e., is implemented, for instance, using the scoring operators described in Chapter 5), the *Item Parameters* and potential *Cut Scores* (i.e., threshold values for estimated latent abilities) are not considered to be part of the assessment content, because these parameters might either not be known when a newly developed instrument is used for the first time or the values might depend on the intended target population.



How to do this with the CBA ItemBuilder? To implement adaptive testing (i.e., a dynamic selection of *Tasks* during testing, see section 6.7.2), *Item Parameters* are needed. Depending on the deployment software used, the *Item Parameters* can be stored, for example, as an *Item Pool* (see section 7.5.5) or used in an R function (see section 7.3.3).

Test Reports: Different parts of an assessment software might be responsible for *feedback* either during the assessment (see section 2.9.1), or after data processing and scoring of open-ended responses (see section 2.9.2). Hence, reports can be generated either online (as part of the assessment software) or offline (as part of the data processing).



How to do this with the CBA ItemBuilder? You can implement basic examples of using CBA ItemBuilder items with automatic, out-of-the-box feedback using the R package `ShinyItemBuilder` (see section 7.3.5).

Data Dissemination: The provision and distribution (i.e., dissemination) of data from computer-based assessments, for example in research data centers, can be done for *Result Data* and *Process Indicators* in the typical form of data sets (one row per person, one column per variable). Since the different assessment platforms and software tools provide log data in different ways, log data can be transformed into one of the data formats described in section 2.8.4 as part of the data processing after an assessment.

8.7 Documentation and Archiving of Computer-Based Assessments



The assessment cycle introduced in the beginning of this chapter (see Figure 8.1) con-

tains the *Documentation & Dissemination* as the last component. In general, documentation can be understood with respect to the items (i.e., the instrument) and the data (see Table 8.9).

TABLE 8.9: Steps for Documentation

| |
|-----------------------------------|
| Documentation |
| Item and Instrument Documentation |
| Data and Log-Data Documentation |

Archiving and documentation of computer-based assessments can have different objectives. The first central question is whether there is a link to research data that has already been collected. Hence, the software's archiving often takes place in the context of the data archiving so that questions regarding the interpretation or understanding of the existing data can be answered concerning the used software. In this case, the software used should be provided along with the assessment content (i.e., tasks, instruction pages, etc.) as closely as possible to how they were used to collect the research data. However, since the software might come with specific requirements archiving the computer-based assessment must take into account these requirements so that the software can (hopefully) also be executed in the future.

Archiving of computer-based assessments can also serve the purpose that other researchers or stakeholders can use the developed assessment instruments (sharing). The two goals need not be mutually exclusive, but it should be made clear what the goal of archiving computer-based assessments is.

- Goal to archive assessment content to document an existing data set
- Goal to allow the use of developed content in future data collections

A second key issue concerns the separation of assessment software and assessment content. Such a separation exists, for example, if the software allows the export of the assessment content created with it, as it is the case, for instance, with TAO that allows exporting the items in QTI format (*Question and Test Interoperability*⁶). In the case of QTI, different software components could be used to administer assessments that use the QTI content. A similar separation also applies to the CBA ItemBuilder, which allows the assessment components created with it to be archived independently of the software (i.e., the specific version used to author the CBA ItemBuilder project files and the software used for test-deployment). Since the CBA ItemBuilder project files contain the runtime configuration (see section 8.3.3), that is sufficient to use deployment software (including TAO, see section 7.4) or the *TaskPlayer* API (see section 7.7).

- Requirements to run / use the software (operating system / frameworks / browsers)

⁶Que (2022)

- Requirements to run / use the content (compatibility of content, e.g., QTI version)

A third question concerns the anticipated, expected, and allowed use and possible modifications to the archived computer-based assessment, for instance, for future data collections on new samples. This third question includes licensing issue regarding the content (i.e., the items and possible embedded resources such as images), licensing of the software, and the technical aspects required for using (i.e., executing and running) the software securely.

- Right to use the software / the content for specific purposes (e.g., new data collection)
- Right to store the software / the content (for instance, for achieving)
- Right to distribute the software / the content for further use (e.g., for other projects)
- Right to change the software / the content (for instance, to adjust for further needs)

8.7.1 Archiving CBA Software to Document Datasets

If the goal is to archive a digitally-based assessment to interpret existing data, a first idea could be to archive the complete software *as used for the data collection*.⁷ The underlying rationale is similar to paper-based assessments and the practice of archiving the assessment materials (i.e., booklets), for instance, as PDF files. However, acknowledging that the assessment was digitally based, more than static representations for items or screens (e.g., screenshots) might be required, and archiving the assessment as *an interactive* system might be considered the natural choice.

Documentation of Requirements: Whether the archiving of the software used in data collection is useful depends, first of all, on how the requirements needed to run the software can be fulfilled. Accordingly, a prerequisite for investigating the viability of this approach is a documentation of all runtime requirements from a technical perspective. Assessments used in *offline deployments* (see section 7.2.1) might require a particular operating system, require a minimum screen resolution, and might be tested only for particular pointing devices (e.g., not tested for touch input). Beyond these apparent requirements, dependencies (i.e., specific browser versions, installed frameworks or components, such as Java or .NET), user privileges (i.e., is admin access required), and network requirements (e.g., free ports) need to be documented and considered. If assessments were performed with dedicated hardware (i.e., computers that were deployed to the assessment sites), additional settings and configurations (e.g., at the operating system level) might also be necessary in order

⁷Sometimes deciding which version of a computer-based assessment should be archived might also be relevant. Suppose changes in the assessment software and content are tracked, for instance, using version control tools (see section 8.3.2). In that case, the datasets might reference that particular version, and archiving should contain all versions used during data collection.

to be able to reproduce the data collection with the archived software. In particular for *mobile deployments* using apps, the distribution of the assessment software to the mobile devices needs special attention. For *online deployments*, both perspective need to be distinguished: For *online deployments*, both perspectives need to be distinguished: At the client side, supported (i.e., tested, see section 8.4.1) browsers need to be documented, while at the server side, documentation of runtime requirements and the server configuration might be relevant to run the assessment software.

Software Virtualization: Techniques such as *Virtual Machines* (used, for instance, for desktop virtualization, such as VMWare, Virtual Box or Parallels) and *Containers* (used, for instance, for server virtualization, such as Docker or LXC) might help to make software (in specific environments) available for a more extended period. However, in particular for desktop virtualization, licensing of the operating system need to be considered.

Intended Use of Archived Assessment Software: The critical question regarding the usefulness of this type of archiving is what researchers can do with assessment software archived in this way. If no further precautions have been taken in the assessment software itself, then items can be replayed and answered in the combinations used in the field (e.g., within a booklet design). This option can be helpful, for example, to learn about the items (i.e., the assessment content) in context, to inspect the behavior of items and the assessment platform, and to investigate how prompts or feedback were displayed. If the archived assessment software also provides the generated (raw) data access, this approach also allows checking how a particular test taker or response behavior is stored or represented in the data set.

8.7.2 Dedicated Approaches for Documenting CBA Data

As described in the past section, archiving the assessment software itself, while an obvious idea, is of limited benefit for documenting data from computer-based assessments without special provisions within the assessment software.

Documentation of Result Data and Process Indicators: In terms of documentation of outcome data (i.e., raw responses and input as well as scored responses), data sets with result data of computer-based surveys are standard. Hence, codebook documents can be used to describe the result data (in terms of metadata). Result Data, available in variable values per person, can be supplemented by additional *Process Indicators* (i.e., information describing the test-taking process), for which a value (including NA) is also expected for each person.

If knowledge of the specific item content is necessary for interpreting the result data or the process indicators, insight into tasks provided by an archived assessment software may be sufficient. However, some information about the log data generated when interacting with the assessment content can be necessary to document *Raw Log Events* and *Contextualized Log Events* (see section 2.8.1 for the terminology).

Documentation of Raw Log Events and Contextualized Log Events: Which interac-

tions are generally stored by a digitally-based assessment can often be documented and described even without the specific assessment content. In case of the CBA ItemBuilder, the log events provided by the items are described for the different components used to implement the content (see appendix B.7 for a documentation of log events), and additional log events might be defined by the item author (described as *Contextualized Log Events*). Moreover, the deployment software is expected to add additional log events at the platform-level.

The more challenging part of the documentation is to relate the assessment content and the collected log data so that the data can be meaningfully interpreted in the context of test-takers interactions and assessment content.

Real Items and Live Access to Log Events: The obvious option to allow researchers to inspect interactive assessments is to give them the computerized items in a form where the events stored in the log data are visible after one has demonstrated a particular behavior or interacted with the item. This can be achieved by different approaches, either by modifying the deployment software (see, for instance, section 7.3.7) or by using the authoring software (see *Trace Debug Window* in section 1.6.2).

Documenting Instruments Using Mock-Items: Given that assessments are often translated (e.g., in the context of international large-scale assessments), there is another way of documenting interactive items to facilitate the interpretation of log data. For that purpose, we define *Mock-Items* as items in which the sensitive item content (i.e., everything that should not become public to keep the items secure) is replaced by placeholders. Such a replacement is required for all texts, images, video, and audio files that could provide hints about the item's content. However, it is assumed that replacing the content is possible without altering or destroying the interactive items' structure and functioning.



How to do this with the CBA ItemBuilder? The recommended strategy to document the log data of assessment content created with CBA ItemBuilder is to provide access directly to the CBA ItemBuilder *Project Files*. If this is not possible for item content protection, mock items might help.

Screen Casts or Annotated Screenshots: Documenting log events can also be done using screen casts (i.e., screen recordings showing a particular behavior and the generated log events), for instance using released items. Or annotated screenshots of computer-based instruments can be used. And even specifically created webpages that show how specific interaction are logged can be used (e.g., [PIAAC RI Log Data Documentation](#)).

8.7.3 Approaches to Archive or Share Assessments for Re-Use

Beyond documenting existing data, an important goal can be sharing developed assessment content to use in future data collections.

Sharing Software *as is*: Although similar to the idea described above (see section 8.7.1), sharing assessment content bundled with an assessment software *as-is* for re-use adds additional challenges. The following aspects require special attention: First, it must be considered that the redistribution of the software is different from the use of the software, so it may be a question of licensing whether the right to redistribute exists for the software and for the included content. A second aspect concerns the issue of IT security. For archiving accompanying a data set, the assessment software is used under controlled conditions. However, if sharing assessment for re-use aims to facilitate future data collections with a digitally-based assessment using existing software *as is*, it must also be possible to do so safely. For online deliveries, in particular, this requires patching and applying security updates sooner or later, meaning the possibility of maintaining the software.

Sharing of Software with Sources: Many assessment software maintenance and customization issues can be solved if the runtime components (i.e., compiled or built code) and the source code are archived. In particular, if assessment content and assessment software are not separate, making them available, for example, via a public source code repository (e.g., GitHub.com) may allow other researchers to reuse the resources developed. While the open source provision of assessment software naturally presupposes the right to disseminate the sources, it also presupposes the human resources (i.e., appropriate IT know-how) to be able to use them.

Sharing of Content (Only): An obvious alternative to sharing created assessment content for further use arises when the *Content* can be separated from the *Software*. The option to share created items as *Content* is at first analogous to paper-based assessment. As soon as a PDF or Word document of a test booklet is shared, it can be used to prepare future assessments.



Question & Test Interoperability (QTI) is a standard to share assessment content that is supported, for instance, by [TAO](#). Using the converter [fastib2pci](#), CBA Item-Builder generated content can be packaged as PCI components, that can be embedded and used in QTI items (see section 7.4).

Two examples will be examined in more detail here. If a standard exists (as is the case, for example, with Open Office XML⁸ for text documents), then different programs can use documents that follow that standard. The *Question & Test Interoperability* (QTI) specification can be understood as a similar standard for computer-based assessments. If, for example, items created in [TAO](#) are exported in QTI format, then these can be stored and used in later assessments if an assessment software can read and process the QTI format. The apparent prerequisite for this model to be applicable is that the assessment content can be implemented as QTI items. As the field of computer-based assessment continues to evolve, the QTI standard is also being ex-

⁸ECMA-376, ISO/IEC 29500

panded and adapted⁹. Hence, it might be necessary to document the exact version of the QTI standard, and only the particular version of the software used to author the QTI items (e.g., a specific [TAO](#) version) might interpret the assessment content precisely (i.e., the rendering and behavior of the interactive content might be different across different QTI players). Moreover, if the software used for QTI editing adopt to a new version, a migration process might be required.



How to do this with the CBA ItemBuilder? Assessment content created with CBA ItemBuilder in one version can be archived as CBA ItemBuilder *Project Files* and shared for later use. To use the assessment content, delivery software containing the runtime of this CBA ItemBuilder version is required.

If the standard is not sufficient, sharing the content independently from the software used to create the content can also be possible. This is illustrated with the CBA ItemBuilder, which does not follow the QTI standard. However, as long as a deployment software is available that supports this version of the CBA ItemBuilder, the generated content can be used for future data collections.

Migration Strategy: *Project Files* of recent CBA ItemBuilder versions can be used for assessment projects, as long as sufficient browser support is provided and no technical or security-related issues prohibit the use of old versions. If archived *Project Files* of an older version can no longer be used in current delivery software, older *Project Files* can be migrated using the CBA ItemBuilder. Migrating an outdated *Project File* means opening the *Project File* in a newer CBA ItemBuilder and then saved as a *Project File* in this new version. Doing so will update the generated code or the runtime configuration required to use the *Project File* with a particular deployment software.

The update of *Project Files* is possible because the implementation of the CBA ItemBuilder ensures that a newer version can read the content of the previous version and convert it if necessary. Accordingly, it may be necessary to perform the migration in multiple steps (using intermediate versions of the CBA ItemBuilder). The release notes of the CBA ItemBuilder (see Table [B.5](#)) provide information on points to be considered regarding *backward compatibility*.



How to do this with the CBA ItemBuilder? The recommended strategy for sharing and archiving assessment content created with CBA ItemBuilder is to provide *Project Files*. As long as suitable deployment software supports the version, the *Project Files* can be used directly. The *Project Files* can be migrated with the CBA ItemBuilder if no deployment software supports the (outdated) version.

⁹See, for instance, [IEdTech Question & Test Interoperability \(QTI\)](#)

8.7.4 Assessment Content as Open Educational Resources (OER)

The archiving of created and digitally based implemented assessment content in educational science applications can be understood as a particular form of *Open Educational Resources (OER)*. This is particularly true if the goal is to enable content sharing, where the developed items constitute the shared resource.

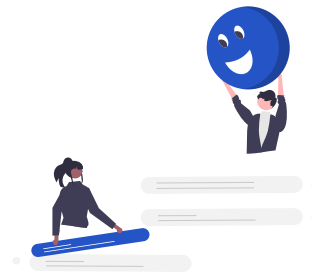


How to do this with the CBA ItemBuilder? To support the sharing and provision of assessment content created with CBA ItemBuilder, a suitable license should be defined and metadata stored within the CBA ItemBuilder *Project Files* (see section [6.3.4](#))

Before making extensive adjustments to items, it must be thought about whether this will change psychometric properties and item parameters that have been empirically determined or verified, for example, with the help of a scaling study (see section [2.5.4](#)).



Closing Chapter

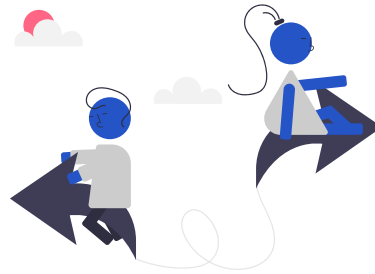


Without a doubt, the possibilities and options of computer-based assessment have not yet been exhausted. Further research and the development of future software tools are necessary to collect diagnostic information efficiently that allows deriving inference with valid interpretations. Even though there are still many ideas and reasons to further develop in particular the CBA ItemBuilder, we provide this tool as it is to support *Open Science* (Stodden et al., 2014) and to support *Reproducible Research* (Gandrud, 2020; Christensen et al., 2019) in the area of computer-based assessment.

- Why Open Science? The development of assessment instruments, psychometric tests, and questionnaires in a research context is often publicly funded. Making these instruments available for later use contributes to an *Open Science* understanding.
- Why Reproducible Research? The comparability of assessments (i.e., mode effects) is influenced by a large number of small *Properties of Measurement* (see section 2.2.1). To increase the reproducibility of findings, insight and exchange about the computer-based measurement instruments are necessary, which might be fostered by sharing or providing restricted access to CBA ItemBuilder items.

Interactive learning content and related assessment items created in CBA ItemBuilder can also be shared in terms of *Open Educational Resources* (OER, see section 8.7.4).

How to Share?



Sharing CBA ItemBuilder items is easy. Define the metadata (see section 6.3.4) within the CBA ItemBuilder project files to inform about the author, the license, and the possibilities to use your content and upload the CBA ItemBuilder item to a repository (e.g., [osf](#) or [GitHub](#)). If you choose GitHub, you might create a new repository using the template provided as [fastib2pci](#) (see section 7.4). Using this template will not only automatically provide a PCI component but also a simple preview of the item content as a static GitHub page. You can also archive your computer-based items as assessment material to research data centers.

How to Contribute?



Requirements for the development of the CBA ItemBuilder are collected and prioritized by the [Centre for Technology-Based Assessment \(TBA\)](#). If features are needed or need to be expanded for future projects, contact the ib-support@dipf.de.

Open Source development is more than welcomed for HTML5/JavaScript packages that can be used as `ExternalPageFrame` and for embedding of CBA ItemBuilder tasks

using the *TaskPlayer-API* (see section 7.7) into open source software that can be used for test deployment.

Suggestions for example items, references or feedback about errors or improvement of this book can be provided to ib-support@dipf.de.



A

Glossary of Terms

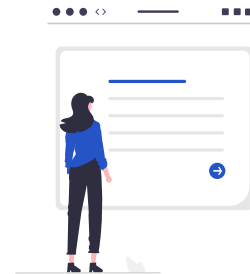


TABLE A.1: Glossary of (Technical) Terms used in this Book

| Term | Description |
|--|--|
| <i>Adaptive Testing Assessment Component</i> | (see <i>Computer-Adaptive Testing</i>) Depending on the intended use, the <i>Project Files</i> created with the CBA ItemBuilder can contain either instruction pages or single items, questions, so-called units, stages or complete tests. For simplification, this book will therefore refer to <i>Assessment Components</i> , meaning <i>Project Files</i> with a particular <i>Task</i> used as entry point. |
| <i>CBA Presentation Size</i> | CBA ItemBuilder projects are always created for a specific resolution. The <i>CBA Presentation Size</i> for new projects is defined as a property of the started CBA ItemBuilder instance and can be changed in the menu via the entry 'Utilities > Open preferences' (see section 3.2.2). The <i>CBA Presentation Size</i> for existing projects is defined in the <i>Global Properties</i> (see section 3.6.2). |
| <i>Class</i> | In the context of the CBA ItemBuilder, the term <i>Class</i> is used for the assignment of hit or miss conditions of the <i>Scoring</i> to groups or variables (see section 5.1). |
| <i>Command</i> | (see <i>Runtime Command</i>) |

| Term | Description |
|---------------------------------------|---|
| <i>Computer-Adaptive Testing</i> | Computer-based adaptive testing (CAT) is a psychometric technique in which the selection of administered items from an item bank is designed to increase measurement efficiency by taking into account previously observed answers (see sections 2.7.4 and 6.7.2). |
| <i>Component</i> | For creating items with the CBA ItemBuilder, all objects that can be added to an item from the <i>Palette</i> are called <i>Components</i> (see section 3.7). <i>Components</i> that can be added to a currently selected <i>Component</i> in the <i>Drawing Area</i> of the <i>Page Editor</i> , are listed in the <i>Palette</i> . Only <i>Containers</i> can have nested <i>Components</i> . |
| <i>Component Edit</i> | The list of all <i>Components</i> that are nested in the the current selected <i>Component</i> are listed in the so-called <i>Component Edit</i> view (see section 3.1.2). |
| <i>Conditional Link</i> | Link which can refer to different pages (including the current pages) via conditions and can execute optional operators when called (see section 4.3). |
| <i>Container</i> | <i>Containers</i> are <i>Components</i> that can contain other <i>Components</i> (see section 2.11.4). <i>Components</i> that belong to a common container might share properties. For instance, the radio buttons (components of type <code>RadioButton</code> , see section 3.9.2) that belong together are nested in a specific container, called <code>RadioButtonGroup</code> . |
| <i>Context Menu</i> | Menu (see section 3.1) that can be called to configure components in the <i>Page Editor</i> and to call commands in the <i>Project View</i> , <i>Component Edit</i> and in the <i>Embedded HTML Explorer</i> using the right mouse button (secondary click). |
| <i>Dialog</i> | Pages whose <code>Frame</code> component is configured to be displayed as a popup or dialog (see section 3.15). Dialogs can either be displayed as an additional page while continuing to interact with the main page, or dialogs are displayed as <i>Modular Dialogs</i> which are displayed exclusively and do not allow interaction with the underlying page. |
| <i>Domain Specific Language (DSL)</i> | The syntax components used for designing the dynamic parts of assessment components and for scoring tasks are not defined in any specific programming language. Instead, the CBA ItemBuilder uses its own syntax for the domain of computer-based assessment, which can be referred to as a domain-specific language (DSL). Based on this syntax, a configuration is then generated, which can be processed at runtime in a programming language (e.g. JavaScript). |

| Term | Description |
|-----------------------------------|---|
| <i>Entry Point</i> | Each <i>Project File</i> that is edited with the CBA ItemBuilder and then used to configure test deliveries must define entry points. These entry points are defined as <i>Tasks</i> and also include the scoring definition (see <i>Task</i>). |
| <i>Event</i> | (see either <i>Log-Event</i> or <i>FSM-Event</i>) |
| <i>Execution Environment</i> | The CBA ItemBuilder creates <i>Assessment Components</i> that can be used in assessments as test assemblies in so-called <i>Test Deliveries</i> (see chapter 7). |
| <i>Finite-State Machine (FSM)</i> | <i>Finite-state machines</i> are used as <i>logic layer</i> inside the CBA ItemBuilder to allow complex item behavior and advanced interactivity (see section 4.4). This layer can be used to modify the visual presentation or behavior of items in a very flexible way, using <i>FSM Events</i> , <i>FSM Variables</i> and <i>FSM Operators</i> . Many of the advanced functions of the IB can be constructed by using one or multiple (nested) finite-state machines. |
| <i>Frame</i> | Assessment components are designed with the CBA ItemBuilder using pages of different types. Each page needs a root component, which is used to define page properties (such as size). Components of type <code>Frame</code> serve as root element for pages (see section 3.5.1). |
| <i>FSM Event</i> | Finite-state machines (FSM) process so-called <i>FSM events</i> (see section 4.4.3) using deterministic rules (see section 4.4.4). The <i>FSM events</i> can be triggered automatically by time intervals, by timers or by user interactions. Many components therefore allow one or more <i>FSM events</i> to be assigned to them (see section 4.4.3). <i>FSM events</i> can be used during the administration of assessment components. After administration, user interactions and internal state changes are stored in <i>Log events</i> in the <i>Log data</i> , and can be analyzed (see section 2.8), for instance, to extract <i>Process Indicators</i> . |
| <i>FSM Operators</i> | If a rule is defined for the current state of a <i>Finite-State Machine (FSM)</i> that matches a <i>FSM event</i> that has just been triggered, then the <i>FSM</i> will process that rule and change to the new state if so defined as <i>FSM Transition</i> . As part of the rule, operators can be defined that are executed to make changes to variables or components. The CBA ItemBuilder provides a set of <i>FSM Operators</i> that can be used to perform actions in transitions (see section 4.4.6). |
| <i>FSM Rules</i> | The possible transitions between states of <i>Finite-State Machines (FSM)</i> are defined using rules (see section 4.4.4). Rules are triggered by <i>FSM events</i> and can contain conditions (see section 4.4.5). |

| Term | Description |
|---------------------------|---|
| <i>FSM Variables</i> | The CBA ItemBuilder adds variables to the capabilities of (nested) <i>Finite-State Machines (FSM)</i> . Variable values can be used with the help of <i>Value Maps</i> and <i>Map-Based Value Displays</i> (see section 4.2.5) to change the appearance of pages. FSM variables can also be used in scoring conditions (see section 5.3.5) and in conditions of <i>FSM Rules</i> (see section 4.4.5). |
| <i>GIF</i> | Graphics Interchange Format is an 8-bit-per-pixel bitmap image format that was introduced by CompuServe in 1987 and has since come into widespread usage on the World Wide Web due to its wide support and portability (see section 3.10.1 for supported file formats of the CBA ItemBuilder). |
| <i>Global Properties</i> | Settings that apply to a particular project are defined in the <i>Global Properties</i> (see section 3.6.2). |
| <i>HTML5/JavaScript</i> | Hypertext Markup Language (HTML) in version 5, as defined by the World Wide Web Consortium (W3C) and the scripting languages <i>JavaScript</i> that is used browsers to support interactive content. |
| <i>Hit-Condition Item</i> | (see <i>Scoring Condition</i>) A part of a test assembly corresponding to a question or task in an assessment. <i>Items</i> can be presented each on a single page or multiple items are placed on one page, depending on the task design and the implemented navigation (see section 2.4). |
| <i>Item Size</i> | (see <i>CBA Presentation Size</i>) |
| <i>JPEG</i> | JPEG (Joint Photographic Experts Group) is a commonly used method of compression for photographic images (see section 3.10.1 for supported file formats of the CBA ItemBuilder). |
| <i>Link</i> | The connection of pages in the CBA ItemBuilder is done via <i>links</i> . A <i>link</i> can be triggered by clicking on different components. <i>Links</i> can also be integrated into components of type <code>HTMLTextField</code> or <code>TextFiel</code> . <i>Links</i> that always lead to the same page (see section 3.11) are distinguished from <i>Conditional Links</i> (see 4.3). |
| <i>Main Menu</i> | Main menu of the CBA ItemBuilder at the top of the program window (see section 3.1.1). This menu must be distinguished from the context menus of the CBA ItemBuilder (see <i>Context Menu</i>). |
| <i>Miss-Condition</i> | (see <i>Scoring Condition</i>) |
| <i>Mock Cases</i> | Click patterns used to check scoring and data storage (see section 8.4.2). |

| Term | Description |
|------------------------|---|
| <i>Mock Items</i> | For the interpretation of log data from computerized assessment, access to tools is necessary so that the meaning of log events can be clearly interpreted and contextualized. If the release of the original items is not possible, it is possible to create so-called <i>Mock Items</i> , which do not contain any protected item content, but are similar enough in structure to the original items that they can be used to interpret the log data (see section 1.6). |
| <i>Modal Dialog</i> | (see <i>Dialog</i>) |
| <i>Log-Events</i> | Interactions between test-takers and an assessment platform can be stored in the form of log data, which consists of a variety of events of different types (<i>Log-Events</i> , see section 1.6). Together with the task content, process indicators can be derived from the analysis of log data (see section 2.8). |
| <i>Minimal Example</i> | Small, fully executable example illustrating a particular functionality (see section 8.1). |
| <i>Navigation</i> | The term navigation is used to describe how test-takers can move between different <i>elements</i> (between-item navigation), or within an element (such as an item or a units (within-item navigation). Different response elements (for example buttons) can be used to trigger navigation between pages within-items. Likewise, <i>FSM operators</i> or <i>Commands</i> can be used to trigger between-item navigation. |
| <i>Operators</i> | (see <i>FSM Operators</i> or <i>Scoring Operators</i>) |
| <i>Panel</i> | Components of type 'Panel' are used to group components and to display background images or frames, for example. Typical pages of type <i>Simple Page</i> need at least one additional component of type <i>Panel</i> within the root element (<i>Frame</i> , see section 3.5). |
| <i>Page</i> | Each CBA ItemBuilder project can consist of one or multiple pages. Pages are of a particular <i>page type</i> . Pages typically contain a <i>Frame</i> and a <i>Panel</i> , as well as additional <i>Components</i> (either visual elements such as texts, images or response elements, such as <i>CheckBoxes</i> or <i>Buttons</i>) or <i>Containers</i> (for instance, additional <i>Panels</i> or <i>RadioButtonGroups</i>). |

| Term | Description |
|-------------------------------------|--|
| <i>Page Type</i> | Each page is of a particular type, and several <i>page types</i> are available in the CBA ItemBuilder. The available components and containers which can be used to add content to a page depend on the page type. Pages of different type have to be used to implement specific parts of a complex computer-based item. The type of a page, for example, <i>Simple Page</i> , <i>WebBrowserPages</i> OR <i>TaskBarPages</i> is defined when the page added to the project and it is impossible to change the page type. |
| <i>Palette</i> | The Palette is the part of the CBA ItemBuilder user interface that gives access to components that can be added to the currently selected part in the visual designer. |
| <i>PNG</i> | Portable Network Graphics, a bitmapped image format that employs lossless data compression (see section 3.10.1 for supported file formats). |
| <i>Preview</i> | The CBA ItemBuilder is the authoring tool to create elements for computer-based assessments. These elements can be used in browser-based test deliveries. In order to see how the items will be rendered in the browser environment, the CBA ItemBuilder allows to <i>preview</i> the elements (<i>Tasks</i> , <i>Projects</i> or <i>Pages</i>) by automatically generating the required HTML pages and opening the pages a web-browser. |
| <i>Preferences</i> | Default settings for the CBA ItemBuilder are defined in the CBA ItemBuilder <i>Preferences</i> (see section 3.2.2). |
| <i>Project File</i> | Item authors can build and edit <i>Project Files</i> within the CBA ItemBuilder. A <i>Project File</i> can contain <i>assessment components</i> (i.e., either a single item, several units, or even a complete test). The decision about how many separate project files are used to computerize a particular test material is up to the item author. Project files created with the CBA ItemBuilder are ZIP archives. However, very large files with many items in single IB project are not suggested, because the size of the ZIP files might become too large. |
| <i>Properties / Properties view</i> | The components added to an item are configured using additional <i>properties</i> . Properties of the currently selected component can be edited in the so-called <i>Properties</i> view. |
| <i>Process Indicator</i> | Specific information that, extracted from individual log events (raw log data), provides knowledge about a particular behavior as a person variable (see section 2.8). |
| <i>RAP (depraved)</i> | The technical platform used to generate HTML content in older CBA ItemBuilder versions (RAP = Remote Application Platform). |

| Term | Description |
|---------------------------|--|
| <i>React</i> | A JavaScript library used by the current CBA Item Builder as part of the generated HTML content. |
| <i>Resources</i> | Resources used within CBA ItemBuilder projects are included in the <i>project files</i> , for example, pictures, audio files and video files. |
| <i>Regular Expression</i> | A sequence of characters that define a search or validation pattern (<i>RegExpr</i> , see section 6.1). The CBA ItemBuilder uses “Unicode Technical Standard#18 Unicode Regular Expressions” (http://www.unicode.org/reports/tr18/) to score text responses (see section 6.1.2) and as <i>Input Validation Pattern</i> to restrict the characters that can be entered in text entry items (see section 6.1.3). |
| <i>Raw Variables</i> | For the definition of result variables, a distinction can be made between variables that contain the raw response (as entered by the test-taker) and result variables that store the scored response (see <i>Scoring Variables</i>). In the CBA ItemBuilder <i>Raw Variables</i> and <i>Scoring Variables</i> can be defined. For storing input text (and numeric variables) the <code>result_text()</code> operator can be used (see section 5.3.10). |
| <i>Rules</i> | (see <i>FSM Rules</i>) |
| <i>Runtime Command</i> | Command that can be assigned as action to components such as Buttons, and that is processed by the runtime environment (e.g. to switch to the next task, see 3.12). |
| <i>Scoring</i> | Scoring refers to the automatic analysis of test-taker's response with respect to pre-defined (scoring) rules. Item authors can precisely define scoring patterns for each possible response. Scoring patterns are organized within <i>Task</i> (see section 5.1 for details). |
| <i>Scoring Condition</i> | The scoring of CBA ItemBuilder <i>Tasks</i> is structured using conditions, which can be defined in CBA ItemBuilder as <i>Hit-</i> (or <i>Miss-</i>) Conditions. For categorical variables, a <i>Scoring Condition</i> assigned to a <i>Class</i> (i.e. variable) is a possible categorical value. <i>Scoring operators</i> can be used to define when exactly a <i>Class</i> should be assigned this value (see section 5.3). This procedure also allows the definition of missing values (see section 2.5.2). |
| <i>Scoring Operators</i> | For the definition of scoring conditions, operators can be used to form logical expressions or to evaluate the state of components, the visited pages or occurred states, etc. (see section 5.3). |

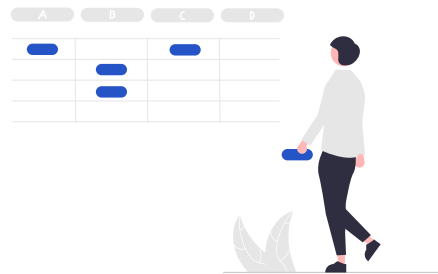
| Term | Description |
|--------------------------|---|
| <i>Scoring Variables</i> | In the scoring definition, the information about which selection or input should be counted as correct (<i>Full Credit</i>) or partial (<i>Partial Credit</i>) can already be taken into account. In this case it is possible to speak of <i>Scoring Variables</i> . |
| <i>String</i> | A string is a data type used in programming that is comprised of a set of characters that can contain text, spaces and numbers. |
| TAO | The TAO Framework is an open-source project which provides a very general and open architecture for computer-assisted test development and delivery. Assessment components created with the CBA ItemBuilder can be used with TAO using the PCI interface (see section 7.4). |
| <i>Task</i> | Within CBA ItemBuilder projects, <i>Tasks</i> are self-contained item packages defined by the item author. Tasks are used to provide entry points into CBA ItemBuilder <i>project files</i> that can be used for test assemblies. |
| <i>Task-Editor</i> | Part of the CBA ItemBuilder that allows to define <i>Tasks</i> (see section 3.6 and to define <i>Hit-/Miss-Conditions</i> for <i>Scoring</i>). |
| <i>Template</i> | Pages can be exported as templates for later use. Using this functionality, page duplication and efficient workflows for creating assessment components can be realized with the CBA ItemBuilder (see section 6.8.7). |
| <i>Test Assembly</i> | The assembly of items into tests or test parts is called <i>test assembly</i> . This can be done to create booklets, rotations or multi-stage tests before test delivery (once), or during testing <i>on the fly</i> or depending on previously observed responses (see <i>Computer-Adaptive Testing</i>). |
| <i>Test Delivery</i> | The collection of data in computer-based assessment is referred to as test-delivery. Assessment components created with the CBA ItemBuilder can be used for different forms of delivery, e.g. stand-alone offline or online (see chapter 7). |
| <i>Toolbar</i> | Important functions of the CBA ItemBuiler are permanently accessible via small icons directly below the Main Menu via the <i>Toolbar</i> (see section 3.1.1). |
| <i>Trace-Event</i> | (see either <i>Log-Event</i>) |
| <i>Transitions</i> | (see <i>FSM Rules</i>) |
| <i>Variables</i> | (see <i>FSM Variables</i> , <i>Raw Variables</i> , or <i>Scoring Variables</i>) |
| <i>Weight</i> | Decimal number that can be defined when scoring (usually 1). If a <i>weight</i> is used, then one result can be scored per <i>task</i> (the <i>hit</i> or <i>miss</i> condition with the highest <i>weight</i>). The use of <i>Classes</i> is then not possible (included to maintain compatibility with previous CBA ItemBuilder projects). |

| Term | Description |
|----------------------|--|
| WYSIWYG | “What you see is what you get” – content (text and graphics) displayed onscreen during editing appears in a form closely corresponding to its appearance when printed or displayed as a finished product. |
| <i>X-Page</i> | Expression used for pages that can be displayed in addition to other pages. The <i>X-Page</i> layout distinguishes between <i>X-Pages</i> and regular pages (such as simple pages). The definition of a page as <i>X-Page</i> is done when the page is created (see 3.4). |
| <i>X-Page-Layout</i> | Arrangement of pages using a regular page and an <i>X-Page</i> simultaneously and side by side. Can be defined for a <i>Task</i> (see 3.6.2). |



B

Useful Tables



B.1 Main Menus

The tables in this annex provide information on the different menus of the CBA ItemBuilder. The description contains references to the sections where further information can be found.

The CBA ItemBuilder contains the following main menus that can be accessed directly in the CBA ItemBuilder window:

- File menu (see table [B.1](#))
- Edit menu (see table [B.2](#))
- Diagram menu (see table [B.3](#))
- Project menu (see table [B.4](#))
- Templates menu (see table [B.5](#))
- Utilities menu (see table [B.6](#))
- Help menu (see table [B.7](#))

TABLE B.1: Overview of the File menu.







| File > | Toolbar | Description |
|---------------|---|---|
| New Project | yes  | Create a new project (see 3.2.1). If a project is currently open that contains unsaved changes, you must decide whether to save or discard the changes before creating the new project. |
| Open Project | yes  | Open an existing CBA ItemBuilder project for editing and previewing (see section 1.4.1). |
| Save | yes  | Save changes in the current CBA ItemBuilder project. |
| Save As... | yes  | Save current CBA ItemBuilder project with new name (see section 3.2.1). |
| Close Project | yes  | Closes the current project. If the current <i>Project File</i> contains unsaved veining, a prompt is displayed. |
| Exit | no  | Exits the CBA ItemBuilder. If the current <i>Project File</i> contains unsaved veining, a prompt is displayed. |

TABLE B.2: Overview of the Edit menu


















| Edit > | Toolbar | Description |
|------------|--|---|
| Undo | no  | Undo last action. Several actions can be undone (see 3.7). |
| Redo | no  | Redo the undone action and perform it again. If several actions have been undone, several actions can also be redone (see 3.7). |
| Cut | no  | Not implemented for <i>Page Editor</i> (see section 3.7.2 on how to alternatively use the <i>Duplicate</i> feature). |
| Copy | no  | Not implemented for <i>Page Editor</i> (see section 3.7.2 on how to alternatively use the <i>Duplicate</i> feature). |
| Paste | no  | Not implemented for <i>Page Editor</i> (see section 3.7.2 on how to alternatively use the <i>Duplicate</i> feature). |
| Delete | no  | Not implemented for <i>Page Editor</i> . |
| Select All | no | Select all components in the current <i>Page Editor</i> |

TABLE B.3: Overview of the Diagram menu

| Diagram > | Toolbar | Description |
|----------------|--|---|
| Font | no  | Font setting for a component selected in the <i>Page Editor</i> , if the component provides a <code>text</code> property. Similar to the settings in the tab <i>Appearance</i> of the <i>Properties</i> view (see section 3.1.4). |
| Fill Color | no  | Fill color for a component selected in the <i>Page Editor</i> , if the component provides a <code>Background Color</code> property (and <code>Is Transparent: false</code>). Identical to the settings in the tab <i>Appearance</i> of the <i>Properties</i> view (see section 3.1.4). |
| Line Color | no  | Line color for a component selected in the <i>Page Editor</i> , if the component provides a <code>Border Color</code> property (and a non-zero <code>Border Width</code> is configured). Identical to the settings in the tab <i>Appearance</i> of the <i>Properties</i> view (see section 3.1.4). |
| Line Type | no  | (not supported) |
| Line Width | no  | (not supported) |
| Arrow Type | no  | (not supported) |
| Line Style | no  | (not supported) |
| Select | no  | Sub-menu with the entries <code>All</code> , <code>All Shapes</code> and <code>All Connectors</code> to select elements nested in the currently selected component of the <i>Page Editor</i> . Note that <i>Connectors</i> refers to paths defined for <i>ImageMaps</i> (see section 3.9.10). |
| Arrange | no  | (Function not supported for current use.) |
| Align | no  | Sub-menu with the entries <code>Align Left</code> <code>Align Center</code> , <code>Align Right</code> and <code>Align Top</code> <code>Align Middle</code> , <code>Align Bottom</code> . |
| Text Alignment | no | Text alignment settings for components that support the property <code>Alignment</code> . |
| Order | no | Sub-menu with the entries <code>Bring to Front</code> , <code>Send to Back</code> , <code>Bring Forward</code> and <code>Bring Backward</code> that can be used to change the <i>Z-Order</i> of components in the <i>Drawing Area</i> of the <i>Page Editor</i> . Note that this order is only for editing purposes and does not affect the order in the <i>Preview</i> . |
| Auto Size | no  | (Function not supported for current use.) |
| Make Same Size | no | Helper to make <code>Width</code> , <code>Height</code> or <code>Both</code> (<code>Width</code> and <code>Height</code>) of two or more selected components identical. |














| Diagram > | Toolbar | Description |
|-----------|--|--|
| Filters | no | (not supported) |
| View | no | Sub-menu to show or hide <code>Grid</code> or <code>Rulers</code> in the <i>Drawing Area</i> of the <i>Page Editor</i> . Also allows to activate or de-activate the <code>Snap to Grid</code> feature (see section 3.7.1). |
| Zoom | no  | Increases zoom in the <i>Page Editor</i> (see section 3.1.1). |

TABLE B.4: Overview of the Project menu

| Project > | Toolbar | Description |
|---------------------------|---|--|
| Preview project | yes  | Launch the dialog to start a <i>Preview</i> of a selected <i>Task</i> , the whole project or a selected <i>Page</i> (see section 1.4.2). |
| New page from template | no  | Create a new page based on an existing page template (see section 6.8.7). |
| Import page | no  | Import an exported page to the current CBA ItemBuilder <i>Project File</i> (see section 6.8.7). |
| Import external content | no  | (Function not supported for current use.) |
| Import XLIFF | no  | Import XLIFF file from external translation (see section 6.9). |
| Verify text translation | no  | Verify imported XLIFF translation (see section 6.9). |
| Finalize translation | no  | Finalize external translation (see section 6.9). |
| Edit all text blocks | no  | Edit all defined text blocks for <code>TextFields</code> used in the current CBA ItemBuilder <i>Project File</i> (see section 5.3.8). |
| Edit all text fields | no  | Edit all text fields used in the current CBA ItemBuilder <i>Project File</i> (see section 1.7.2). |
| Edit all user defined IDs | no  | Dialog to edit all <code>UserDefinedIds</code> in the current CBA ItemBuilder <i>Project File</i> (see section 3.7.4). |
| Update rich text displays | no  | Update the images of all <code>TextFields</code> used in the <i>Page Editor</i> . |
| Browse resources | yes  | Opens the <i>Resource Browser</i> used to import and manage media files resources (see section 3.10.1). |





| Project > | Toolbar | Description |
|----------------------------|---|---|
| Browse Task and Item Score | yes  | Opens the <i>Task Editor</i> used to define <i>Tasks</i> and <i>Scoring</i> for the current CBA ItemBuilder <i>Project File</i> (see section 3.6). |
| Browse Value Maps | yes  | Opens the editor for <i>Value Maps</i> in the current CBA ItemBuilder <i>Project File</i> (see section 4.2.4). |
| Edit State Machine | yes  | Opens the <i>State Machine Tree View</i> and the <i>State Machine Syntax</i> for the current CBA ItemBuilder <i>Project File</i> (see section 4.4.1). |
| Edit State Chart | yes  | (Function not supported for current use.) |

TABLE B.5: Overview of the Template menu


| Templates > | Toolbar | Description |
|------------------|--|---|
| Browse templates | no  | Opens the <i>Template Browser</i> of the CBA ItemBuilder that allows to show, import, delete and export pages saved in the CBA ItemBuilder <i>Instance</i> as <i>Templates</i> (see section 6.8.7). |

TABLE B.6: Overview of the Utilities menu






| Utilities > | Toolbar | Description |
|------------------|---|--|
| XLiff Editor | yes  | (Currently not maintained, see section 6.9) |
| Open preferences | no | Open the CBA ItemBuilder preferences dialog (see section 6.9 for CBA Item Builder XLIFF settings, section 6.8.2 for CBA Item Fonts, section 3.6.2 for CBA Presentation Size, section 1.4.2 for CBA Preview and section 3.1.4 for CBA Ruler And Grid settings). |

TABLE B.7: Overview of the Help menu

| Help > | Toolbar | Description |
|---------------|--|--|
| Help Contents | no  | Shows the <i>Quick Reference</i> embedded in the CBA ItemBuilder. |
| Help Online | no  | Shows online help. |
| About | no  | About dialog showing the version of the CBA ItemBuilder (see section 1.3). |

| Help > | Toolbar | Description |
|-----------------|--|--|
| Update Software | no  | Not implemented. See section 1.2 for information about new versions and updates. |

B.2 Operators

The power of CBA ItemBuilder's *Conditional Links* (see section 4.3) and *Finite-State Machines* (see section 4.4) is achieved by the fact that actions can be executed when *Conditional Links* are activated or transition between states are triggered by active *Rules*. Moreover, scoring of *Tasks* can incorporate component states and additional information provided by the CBA ItemBuilder at runtime (see section 5.3.2).

For many purposes, so-called *Operators* are available (see section 4.4.6), which are listed in a table in this part of the appendix:

- Table B.8 lists operators for *State Machines*
- Table B.9 lists operators for *Component States*
- Table B.10 list operators for *Trees*
- Table B.11 lists operators for *Evaluations*
- Table B.12 lists finite-state machine operators for *Task Management*
- Table B.2 lists finite-state machine operators for *Calculation Engine*
- Table B.13 lists *Logical Expressions*
- Table B.15 lists operators for *Comparisons*
- Table B.16 lists *Arithmetic* operators
- Table B.17 lists *Miscellaneous* operators

In addition to the operator and a description, each table also indicates whether the operator has a return value. If this is the case, an operator can also be used in a condition (see section 4.4.5 for conditions in *FSM Rules*, section 4.3 for conditions in *Conditional Links*, and section 5.3.2 for *Scoring Conditions*).



Tables in this appendix last verified and updated for version 9.1.

TABLE B.8: Operators for *State Machine Variables*

| Operator | Description |
|---|--|
| <code>initFSM(Events)</code> | Execute state machine events. |
| <code>raise(Event)</code> | Raise the given Event after the current event has been processed completely. Multiple operator calls in one rule raise the events in the order of the operator calls. |
| <code>reset(Variable, Variable, ...)</code> | Set the values of the given Variables to 0. |
| <code>set(Variable, Value)</code> | Set the given Variable to the given Value. |
| <code>setFSMEvent(Event, Timeout)</code> | Set the triggering interval for the given timer Event to the given Timeout (in milliseconds). |
| <code>setFSMState(State, Page)</code> | Set the page attribute for the given State in the state machine to the given Page. |
| <code>is_last_state(State, State, ...)</code> | Return true if the last state the state machine was in is one of the given States. |
| <code>raised_all_events(Event, Event, ...)</code> | Return true if all given Events have been raised during the execution of the current task. We consider an event raised even if it did not trigger a transition. We also include events raised by the <code>raise()</code> operator. |
| <code>raised_events()</code> | Return the number of events that have been raised during the execution of the current task. We consider an event raised even if it did not trigger a transition. We also include events raised by the <code>raise()</code> operator. |
| <code>raised_all_events_in_state(State, Event, Event, ...)</code> | Return true if all given Events have been raised in the given State during the execution of the current task. We consider an event raised even if it did not trigger a transition. We also include events raised by the <code>raise()</code> operator. |
| <code>raised_nb_events(Event, Event, ...)</code> | Return the number of events among the given Events that have been raised during the execution of the current task. We consider an event raised even if it did not trigger a transition. We also include events raised by the <code>raise()</code> operator. |
| <code>raised_nb_events_in_state(State, Event, Event, ...)</code> | Return the number of events among the given Events that have been raised in the given State during the execution of the current task. We consider an event raised even if it did not trigger a transition. We also include events raised by the <code>raise()</code> operator. |

| Operator | Description |
|---|---|
| variable_in (Variable, Value, Value, ...) | Return true if the value of the given Variable equals one of the given Values. |
| visited_all_states (State, State, ...) | Return true if all given States have been visited during the execution of the current task. |
| visited_all_values_of_variable (Variable, Value, ...) | Return true if the value of the given Variable has covered all given values during the execution of the current task. Only initial values and values before and after an event has been processed are considered. Intermediate value changes due to modifications by the user (e.g. through a SpinnerValueInput) or during event processing are left out. O is used as variable value at the start of the FSM unless the variable has been initialized explicitly to a different value when processing the initial transition from the start state. |
| visited_nb_states (State, State, ...) | Return the number of the given States that have been visited during the execution of the current task. |
| visited_nb_values_of_variable (Variable, Value, ...) | Return the number of given Values that the given Variable has covered during the execution of the current task. Only initial values and values before and after an event has been processed are considered. Intermediate value changes due to modifications by the user (e.g. through a SpinnerValueInput) or during event processing are left out. O is used as variable value at the start of the FSM unless the variable has been initialized explicitly to a different value when processing the initial transition from the start state. |

TABLE B.9: Operators for Component States

| Operator | Description |
|---|---|
| setActive(Component), unsetActive(Component) | Set the Component 'active' or 'inactive'. This switches between the two 'toggling' states of the Component like checked/unchecked for a CheckBox or a RadioButton, pushed/released for a Button in toggling mode or selects an item in a list like the ComboBoxItem in a ComboBox. Set the 'Is Frozen' attribute of the given Component. |
| setFrozen(Component), unsetFrozen(Component) | Set the 'Is Hidden' attribute of the given Component. |
| setHidden(Component), unsetHidden(Component) | Set the 'Highlightable' attribute of the given RichText component. |
| setHighlightable(RichText), unsetHighlightable(RichText) | Switch the selection mode of the given select group Container between multi-select and single-select. |
| setMultiSelect(Container), unsetMultiSelect(Container) | Set the 'Selectable' attribute for the given Component. |
| setSelectable(Component), unsetSelectable(Component) | Set the given Page as embedded page for the given PageArea. |
| setEmbeddedPage(PageArea, Page) | Set the focus to a component. Note that the object should be on the current page, otherwise the focus operation might have no effect. If the object is not visible on the page it is scrolled to the object to make it visible. |
| focus(Component) | Initialize the given media player Component. The following Properties will be set (if not explicitly given the default values are used): - automaticStart (default false): Starts the media player automatically when its parent page is displayed, stops when the page is left. - hideControls (default false): Hide all controls. |
| initMediaPlayer(Component, Property, Property, ...) | |

| Operator | Description |
|---|--|
| <code>setMaxPlay(Component, Operation)</code> | <ul style="list-style-type: none">- <code>maxPlay</code> (default 0, means unlimited) : Number of times the media player resource can be played. Note that a resource is counted as 'played', if it reaches the end, or if it is stopped explicitly (i.e. not when leaving the page). A sequence of pause and start operations does not count. Control the given media player Component. The following Operations are available:- <code>mp_start</code>: starts the media player execution at the beginning or continues when paused.- <code>mp_stop</code>: stops the media player execution and rewinds to the beginning.- <code>mp_pause</code>: pauses the media player execution (without rewinding). Control the volume of the given media player Component. The Volume ranges from 0 (mute) to 10 (maximum volume). Set the drag&drop mode of the given Component. The following Modes are available: |
| <code>setMediaPlayerVolume(Component, Volume)</code> <code>setValueDisplayMode(Component, Mode)</code> | <ul style="list-style-type: none">- <code>dd_none</code>: the component does not allow dragging or dropping.- <code>dd_drag</code>: the component only allows dragging.- <code>dd_drop</code>: the component only allows dropping.- <code>dd_dragdrop</code>: the component allows both dragging and dropping. Set the text value of the InputTarget component to the current value of the InputSource component and initialize InputSource with the value of the NewSourceValue if a NewSourceValue is given. Notes: |
| <code>setInputValue(InputSource, InputTarget, NewSourceValue)</code> | <ul style="list-style-type: none">- InputSource and Input Target can be identical, in that case only the NewSourceValue is relevant.- If values from (multi-line) InputFields are assigned to SingleLineInputFields or SearchFields, then only the first line of the InputField (up the the 'first end of line' character) is taken into account.- In a NewSourceValue String any 'end of line' character is represented as <code>\n</code> (e.g. <code>line1\nline2\nline3</code>).- The operator may overwrite SingleLineInputFields or InputFields, even if they are 'read-only'. |

TABLE B.10: Operators for *Trees*

| Operator | Description |
|--|--|
| <code>current_node(Tree, RegularExpression)</code> | Return true if the node ID of the current node in the given Tree matches the given RegularExpression. |
| <code>exists_nodes(Tree, RegularExpression, ...)</code> | Return the number of nodes in the given Tree whose node ID matches at least one of the given RegularExpressions. Each node counts once only. |
| <code>visited_nodes(Tree, RegularExpression, ...)</code> | Return the number of visited nodes in the given Tree whose node ID matches at least one of the given RegularExpressions. Each node counts once only. |
| <code>visited_nodes(Tree, NodeIdPattern, ColumnPattern, ColumnPattern, ...)</code> | Return the number of nodes in the given Tree whose node ID matches the NodeIdPattern and whose column values match the specified ColumnPatterns: The first ColumnPattern corresponds to the node name, the second ColumnPattern to the first additional column, etc. Each node counts once only. |
| <code>tree_move(Tree, Node)</code> | Move the currently selected node in the given Tree into the given Node. The operator ignores the 'read-only' flag of the Tree. |
| <code>tree_copy(Tree, Node)</code> | Copy the currently selected node in the given Tree into the given Node. The operator ignores the 'read-only' flag of the Tree. |

TABLE B.II: Operators for Evaluations

| Operator | Description |
|---|--|
| <code>matches(Component, RegularExpression),</code> <code>matches(Component, RegularExpression,</code> <code>Selector)</code> | <p>Return true if the text content of the Component matches the RegularExpression. For spreadsheet table cells containing a formula the formula result counts as text content. The Selector options are:</p> <ul style="list-style-type: none">- <code>formula</code>: The formula text of a spreadsheet table cell counts as text content (instead of the formula value). |
| <code>user_interactions()</code> <code>getItemScore(Task, Calculation)</code> | <p>Return the number of user interactions within the current task execution.</p> <p>Trigger all scoring calculations for the requested Task and return the requested calculation result. Valid Calculation values are:</p> <ul style="list-style-type: none">- <code>result</code>: overall result ('1' if there are no misses and the minimum required number of hits has been reached, otherwise '0')- <code>nb_hits</code>: number of hits- <code>hit_weight</code>: total weight of hits- <code>nb_misses</code>: number of misses- <code>miss_weight</code>: total weight of misses- <code>credit_class</code>: name of the class with the highest class weight- <code>credit_weight</code>: weight of the class with the highest class weight- <code>nbInteractions</code>: number of user interactions since start of the current task execution- <code>nbInteractionsTotal</code>: accumulated number of user interactions in previous executions of the task- <code>reactionTime</code>: time (in milliseconds) between the start of this task execution and the first user interaction- <code>reactionTimeTotal</code>: accumulated time (in milliseconds) between the start of the task execution and the first user interactions in previous executions of the task- <code>execTime</code>: time in (milliseconds) since the start of this task execution |

| Operator | Description |
|--|---|
| highlighted(RichText, RichText, ...) | - execTimeTotal: accumulated time (in milliseconds) in previous executions of the task Return true if in at least one of the given TextFields all non-blank characters are highlighted. Empty lines correspond to a blank character. |
| complete(Selection, Selection, ...) | Return true if all Selections are selected or completely highlighted. A text block is completely selected if all non-blank characters of the text block are highlighted. Empty lines correspond to a blank character. A Link counts as selected if it was visited already. |
| partial(Selection, Selection, ...) | Return true if at least one of the given Selections is selected or partly highlighted. A text block is partly selected if at least one non-blank character of the text block is highlighted. Empty lines correspond to a blank character. A Link counts as selected if it was visited already. |
| current_page(Page), current_page(Page, PageArea) | Return true if the given Page is currently displayed: |
| bookmarked(Page) | - One parameter variant: Is the page displayed at the top level? |
| integer_value(Component, RoundingMode, Default) | - Two parameter variant: Is the page displayed in the given PageArea? Return true if the given Page is currently bookmarked. Evaluate the text content (or formula value for spreadsheet table cells containing a formula) of the given Component as integer: - If the text content is empty or does not represent a number, return the Default value. - If the text content represents a number and RoundingMode is 'up', then round up always: 1.0 => 1; 1.1 => 2; -1.0 => -1; -1.1 => -2 - If the text content represents a number and RoundingMode is 'down', then round down always: 1.9 => 1; 2.0 => 2, -1.9 => -1; -2.0 => -2 - If the text content represents a number and RoundingMode is 'half_up', then round up at x.5: 1.4 => 1; 1.5 => 2, -1.4 => -1; -1.5 => -2 - If the text content represents a number and RoundingMode is 'half_down', then round down at x.5: 1.5 => 1; 1.6 => 2, -1.5 => -1; -1.6 => -2 |

| Operator | Description |
|--|--|
| <code>panel_distance_range (Container, MinDistance, MaxDistance, Center, Component, Component, ...)</code> | <p>Return true if</p> <ul style="list-style-type: none">- the mutual distance of all given Components in the given Container is within the given range MinDistance..MaxDistance and- for all other components in the given Container the distance is outside the given range for at least one of the given Components. <p>The distance is calculated between</p> <ul style="list-style-type: none">- the centers of the Components if the Center flag it true- the upper left corners of the Components if the Center flag is false. <p>Return true if</p> |
| <code>panel_position_range (Container, XStart, XEnd, YStart, YEnd, Center, Component, Component, ...)</code> | <ul style="list-style-type: none">- the (X,Y) positions of all given Components in the given Container are within the range given by XStart, XEnd, YStart and YEnd relative to the Container's (X,Y) position and- the (X,Y) positions of all other components in the given Container are outside the given range. <p>As (X,Y) position of a Component and the Container counts</p> <ul style="list-style-type: none">- the center of the component if the Center flag it true- the upper left corner of the component if the Center flag is false. |

| Operator | Description |
|--|---|
| <code>result_text(TextSource), result_text (TemplateSource, ValueSource, ValueSource, ...)</code> | The operator always returns true. The single parameter version copies the text from the TextSource to the result text of the Hit/Miss condition. The multiple parameter version uses the text from the TemplateSource as template and replaces each occurrence of <code>%<index>\$\$</code> (where <code><index></code> is an integer between 1 and the total number of given ValueSources) by the text snippet obtained from the ValueSource corresponding to the given index. Example: <code>result_text("Sorted text: %1\$s %3\$s %2\$s", "one", "two", "three")</code> writes Sorted text: one three two |
| <code>trace_snapshot (TextSource), trace_snapshot (TemplateSource, ValueSource, ValueSource, ...)</code> | The operator always returns true. The single parameter version traces the text from the TextSource in the trace log and dumps a snapshot of the current task to trace log. The multiple parameter version uses the text from the TemplateSource as template and replaces each occurrence of <code>%<index>\$\$</code> (where <code><index></code> is an integer between 1 and the total number of given ValueSources) by the text snippet obtained from the ValueSource corresponding to the given index. Example: <code>trace_snapshot("Sorted text: %1\$s %3\$s %2\$s", "one", "two", "three")</code> writes Sorted text: one three two |
| <code>trace_text(TextSource), trace_text (TemplateSource, ValueSource, ValueSource, ...)</code> | The operator always returns true. The single parameter version traces the text from the TextSource in the trace log. The multiple parameter version uses the text from the TemplateSource as template and replaces each occurrence of <code>%<index>\$\$</code> (where <code><index></code> is an integer between 1 and the total number of given ValueSources) by the text snippet obtained from the ValueSource corresponding to the given index. Example: <code>trace_text("Sorted text: %1\$s %3\$s %2\$s", "one", "two", "three")</code> writes Sorted text: one three two |

The operator `matches()` is important for scoring string responses (see section 5.3.4), while the `result_text()` can be used to copy responses to result variables (see section 5.3.10). The operators `trace_text()` and `trace_snapshot()` are important for adding user-defined log events to the trace log (see section 4.4.6).

TABLE B.12: Finite-state machine operators for *Task Management*

| Operator | Description |
|--|--|
| isCurrentTask(Task) next_task(Task, Test) | Return true if Task is currently executing. Switch to the next task. Both parameters are optional, but Task is mandatory if Test is specified. - If no parameter is given, switch to the next task. If there is no next task do nothing. (The execution environment selects the 'next' task.) - If a Task is given, go to this task (in the specified Test if a Test is given). If no such task exists do nothing. |
| back_task() | If the task switch targets to a task that was interrupted before, the execution environment decides whether to initiate a new task instance now or to resume the task interrupted earlier. Switch back to the previous task. If there is no previous task do nothing. (The execution environment selects the 'next' task.) If the task switch targets to a task that was interrupted before, the execution environment decides whether to initiate a new task instance now or to resume the task interrupted earlier. |
| cancel_task() | Terminate the current task without switching to another task. |

Note about execution sequence when used in a state machine transition rule: The system postpones task switches until all other operators in a transition are executed. For multiple task switch operators in a single transition it will only execute the last task switch encountered during the transition. For a transition that leads to a state with a page assigned, the system switches to this page after all actions assigned to the transition are executed. If a task switch appears in the action list of the transition, the page switch is done in the old task since the task switch is postponed as described above.

Table: Finite-state machine operators for *Calculation Engine*

TABLE B.13: Logical expressions in the domain specific language (DSL)

| Operator | Description |
|---------------------------------|--|
| calcGetMem(MemIdx) | Return the rounded integer value of the calculator memory identified by MemoryIndex. Return 0 if the memory is not initialized or not set. |
| calcOp(Operation, IntegerParam) | Performs an operation on the stack of the calculator. The following operations are available: <ul style="list-style-type: none">- clear: Clears the last operand on top of the stack (if any).- clearall: Clears the full stack.- equals: computes the result by executing all operations on the operands on the stack.- add: pushes the add operation to the stack.- subtract: pushes the subtract operation to the stack.- multiply: pushes the multiply operation to the stack.- divide: pushes the divide operation to the stack.- fact: calculates the factorial of the current operand and replaces the current operand by the result.- power: calculates the power of the current operand based on the exponent 'exp' given as IntegerParam and replaces the current operand on the stack by the result: x^{exp}- npower: pushes the power function on the stack (and expects a n-th power integer operand): x^n- root: calculates the power of the current operand based on the inverse exponent given as IntegerParam and replaces the current operand on the stack by the result: $x^{1/exp}$- nroot: pushes the root function on the stack (and expects a n-th root integer operand): $x^{1/y}$- sin: calculates the sine function of the current operand and replaces the current operand on the stack by the result.- cos: calculates the cosine function of the current operand and replaces the current operand on the stack by the result.- tan: calculates the tangent function of the current operand and replaces the current operand on the stack by the result. |

| Operator | Description |
|----------|--|
| | <ul style="list-style-type: none">- <code>cot</code>: calculates the cotangent function of the current operand and replaces the current operand on the stack by the result.- <code>sec</code>: calculates the secant function of the current operand and replaces the current operand on the stack by the result.- <code>csc</code>: calculates the cosecant function of the current operand and replaces the current operand on the stack by the result.- <code>e</code>: calculates the natural exponentiation (base 'e') of the current operand on the stack and replaces it by the result: e^x- <code>exp</code>: calculates the exponentiation of the current operand on the stack with the base given as IntegerParam and replaces it by the result: $base^x$- <code>ln</code>: calculates the natural logarithm (base 'e') of the current operand on the stack and replaces it by the result: \ln- <code>log</code>: calculates the logarithm of the current operand on the stack with the base given as IntegerParam and replaces it by the result: \log_{base}- <code>invmult</code>: calculates the inverse of the current operand, i.e. divides 1 by the current operand: $1/x$- <code>invpower</code>: pushes the inverse power function, i.e. n-th function on the stack (and expects a n-th power integer operand > 0): $\log_y x$- <code>leftbr</code>: pushes a left (opening) bracket on the stack as a marker for intermediate calculation.- <code>rightbr</code>: pushes a right (closing) bracket on the stack and calculates the result of the expression between the corresponding left bracket and replaces it by pushing the result on the stack.- <code>msave</code>: saves the current operand on top of the stack into the memory referred to by the memory index given as IntegerParam.- <code>mread</code>: reads the memory value referred to by the memory index given as IntegerParam and pushes it on top of the stack (or replaces the current operand).- <code>mclear</code>: clears the memory referred to by the memory index given as IntegerParam (i.e. sets it to 0). |

| Operator | Description |
|---|---|
| | <ul style="list-style-type: none">- <code>madd</code>: adds the current operand on top of the stack to the referred memory value and saves the result in the referred memory.- <code>msubtract</code>: subtracts the current operand on top of the stack from the referred memory value and saves the result in the referred memory. <p>Modifies the current operand on the stack of the calculator. The following Operations are available (the parameter Digits – a non negative integer – may be optional):</p> <ul style="list-style-type: none">- <code>add</code>: adds Digits at the end of the current operand.- <code>decimal</code>: sets the decimal place and adds Digits at the end of the current operand (if specified).- <code>back</code>: removes the last digit (or decimal place) from the current operand (if any).- <code>invadd</code>: multiplies the current operand by -1 (additive inverse). Note that more digits can be added to the current operator afterwards. <p>Initializes the calculator and sets Parameters. This must be done at the beginning, before any calculation is started, otherwise it has no effect. The following Parameters are available:</p> <ul style="list-style-type: none">- <code>angle</code>: the unit representation of angles: degree (default) or radian.- <code>displayWidth</code>: the number of digits (> 0) which can be used for external representation (default 10).- <code>includeOperandInHistory</code>: displays current operand in history, if set to true (default true).- <code>withThousandSeparator</code>: thousands separators are not displayed, if set to false (default true).- <code>scale</code>: the number of digits (> 0) after the decimal place, corresponds to the precision used for internal calculations, rounded half away from zero when required (default 0). |
| <code>calcOpnd(Operation, Digits)</code> | |
| <code>calcSettings (Parameter, Parameter, ...)</code> | |

| Operator | Description |
|--|--|
| <code><opnd1> and <opnd2></code> | Binary operator for logical 'and', i.e. returns true if both operands evaluate to true. |
| <code><opnd1> or <opnd2></code> | Binary operator for logical 'or', i.e. returns true if at least one operand evaluates to true. |

| Operator | Description |
|------------------------------------|--|
| not <opnd> | Unary operator for logical 'not', i.e. returns true if the operand evaluates to false. |
| ifthenelse (Condition, Then, Else) | Return the value of then if Condition evaluates to true, otherwise return the value of Else. |

Important: The combination of logical operators requires bracketing (see section 4.1.3).

TABLE B.15: Comparisons in the domain specific language (DSL)

| Operator | Description |
|--------------------|---|
| <opnd1> == <opnd2> | Compares two text or numerical values. If both operands evaluate to an integer, a numerical comparison is done. If both operands evaluate to a text, a text comparison is done. Otherwise the operator returns false. |
| <opnd1> <> <opnd2> | Compares two text or numerical values. If both operands evaluate to an integer, a numerical comparison is done. If both operands evaluate to a text, a text comparison is done. Otherwise the operator returns false. |
| <opnd1> < <opnd2> | Compares numerical values. If both operands evaluate to an integer (or a text that represents a numerical value), a numerical comparison is done. Otherwise the operator returns false. |
| <opnd1> <= <opnd2> | Compares numerical values. If both operands evaluate to an integer (or a text that represents a numerical value), a numerical comparison is done. Otherwise the operator returns false. |
| <opnd1> > <opnd2> | Compares numerical values. If both operands evaluate to an integer (or a text that represents a numerical value), a numerical comparison is done. Otherwise the operator returns false. |
| <opnd1> >= <opnd2> | Compares numerical values. If both operands evaluate to an integer (or a text that represents a numerical value), a numerical comparison is done. Otherwise the operator returns false. |

TABLE B.16: *Arithmetics* in the domain specific language (DSL)

| Operator | Description |
|-------------------|--|
| <opnd1> + <opnd2> | Arithmetic add operation. |
| <opnd1> - <opnd2> | Arithmetic subtract operation. |
| <opnd1> * <opnd2> | Arithmetic multiply operation. |
| <opnd1> / <opnd2> | Arithmetic divide operation. |
| <opnd1> % <opnd2> | Arithmetic remainder operation: Calculate the remainder when dividing the first operand by the second. |

TABLE B.17: Miscellaneous operators in the domain specific language (DSL)

| Operator | Description |
|-------------------------------------|---|
| openDialog(Page, X, Y) | Open the given Page as dialog at the position (X,Y). |
| setGlobalProperty (Property, Value) | Set the given global Property to the given Value. The following properties can be used: - highlight_color: Modifies the current highlighting color to Value. The Value is interpreted as RGB color integer value specified as plain positive or negative integer or as hexadecimal value (prepended with 0x). Invalid values are ignored silently. |

| Operator | Description |
|---|---|
| <code>recommend(Parameter, Parameter, ...)</code> | <p>Highlight recommended Links, Tasks and Tests. Each parameter specifies a test name, a task name and the User Defined Id Path of a Link component separated by dots:</p> <p><code><TestName>.<TaskName>.<UserDefinedIdPath></code> Example: <code>recommend(myTest.someTask.myLink, myTest.anotherTask.someLink)</code> To recommend a task without a specific Link component use an asterisk (*) as value for the User Defined Id Path like this: <code>myTest.someTask.*</code> To recommend a test without a specific task use asterisks for task and User Defined Id Path like this: <code>myTest.*.*</code> For User Defined Ids that are not defined in the current item, the ItemBuilder assumes that their components are part of an XPage.</p> |

B.3 Regular Expression Symbols

TABLE B.18: Regular Expression Symbols

| Symbols | Description of Regular Expression |
|-----------|---|
| . | Matches any character except line breaks. Equivalent to <code>[^\n\r]</code> . |
| ^ | Beginning. Matches the beginning of a string ³ or of a line. |
| \$ | End. Matches the end of the string or of the line. |
| () | Capturing group. Grouping multiple expressions. |
| {} | Quantifier. Defines the number of times that the character in front of the opening bracket occurs |
| {x,y} | x = minimum, y = maximum number of repetitions. |
| [] | Character set. Matches any character in the set. |
| [^] | Negated set. Match any character that is not in the set. |
| \ | Alternation. Matches the expression before or after the <code>\ </code> . |
| ? | Optional. Match between 0 and 1 of the preceding token. |
| + | Match 1 or more of the preceding token. |
| * | Match 0 or more of the preceding token. |
| .* | Set the preliminary or later string part free. |
| \\ | Matches the character after the <code>\\</code> . |
| ?: | Non-capturing Group. Groups multiple tokens together without creating a capture group. |
| \\ | Matches a backslash. |
| [\\s\\S] | Match any character. |
| [a-z] | Range. Matches a character in the range a to z. |
| [A-Z 0-9] | Matches only upper case letters, blanks and digits. |
| [a-zA-Z?] | Matches single characters (only one character or empty string). |
| [^a-z] | Matches all characters except lower case letters. |
| [0-9] | Matches any character in the range 0 to 9. |
| \\d | Digit. Matches any digit character, equals <code>[0-9]</code> . |
| \\D | Not digit. Matches any character that is not a digit character, equals <code>[^0-9]</code> . |
| \\s | Whitespace. Matches any whitespace character (spaces, tabs, line breakes). |
| \\S | Not whitespace. Matches any character that is not a whitespace character. |
| \\w | Word. Matches any word character (alphanumeric), equals <code>[a-zA-Z 0-9]</code> . |
| \\W | Not word. Matches any character that is not a word character (alphanumeric), equals <code>[^a-zA-Z0-9]</code> . |
| \\n | Line break. |

B.4 Technical Configuration

By default, the configuration of the CBA ItemBuilder provided in the `cba-itembuilder.ini` activates the following features:

- State-Machine Debugger (`-DAllowFSMDebugging=true`)
- Trace Debugger (`-DAllowTraceDebugging=true`)
- Scoring Viewer (`-DAllowScoreDebugging=true`)

| Parameter | Description |
|---|--|
| <code>-DAllowFSMDebugging=true</code> | Allows opening the <i>Finite-State Machine (FSM) Viewer</i> in the preview (using Ctrl+M) for debugging the FSM state. Only intended for item development. |
| <code>-DAllowTraceDebugging=true</code> | Allows opening the <i>Trace Event Viewer</i> in the preview (using Ctrl+T) for testing and debugging the logging. Only intended for item development. |
| <code>-DAllowScoreDebugging=true</code> | Allows opening the <i>Scoring Viewer</i> in the preview (using Ctrl+S) for debugging the implemented scoring. Only intended for item development. |

Additional features can be de-activated by modifying the `cba-itembuilder.ini` file:

| Parameter | Description |
|--|---|
| <code>-Dfile.encoding=UTF-8</code> | Support for non-Western European languages. |
| <code>-DsimpleTextEditorWrap=yes</code> | Automatic word wrap in the Simple Text Editor (for SimpleTextFields and InputFields). |
| <code>-DTraceHighlightedText=true</code> | Traces the highlighted texts (start, end position and text). |
| <code>-DTraceVerbose=true</code> | Shows Trace Events on console and log files. |
| <code>-DalwaysGenerateAtSave=yes</code> | Forces automatic generation of the project before saving (should be set to yes). |
| <code>-DdefaultItemHeight=NNN</code> | Default CBA Item height in pixel (NNN), 715 pixel if none is defined. |

| Parameter | Description |
|---|---|
| -DdefaultItemWidth=NNN | Default CBA Item width in pixel (NNN). 722 pixel if none is defined. |
| -DDefinedOrdering=false | Does not apply the defined ordering for overlapping graphical components. Only used for backward compatibility for the old style undefined ordering (applied before rel. 3.4.0). |
| -DisTreeViewShowsAll=no | Shows all components in the Component View (to be used only for debugging purposes). |
| -Donline.help.update.site= | Default path where the bundle to update the CBA Item Builder help feature is located. |
| -DPreloadImages=true | Preloads images of all image/IconValueDisplays at startup of an item, so that value changes are displayed without delay at runtime (default= false). |
| -DPreloadTextFields | TextFields are pre-loaded when the item is initialized the first time. The preloaded TextFields will render much faster at first display time. This is only supported for the browsers Firefox and Internet Explorer due to technical limitations in other browsers (for example Chrome). |
| - | Required MergeSort algorithm. |
| Djava.util.Arrays.useLegacyMergeSort=true | |
| -Declipse.consoleLog=true | TODO |
| -Djetty.port=7070 | Port used by the internal servlet engine Jetty for previewing items. |
| -Donline.help.update.site= | TODO |
| - | TODO |
| Dosgi.Framework.extensions=org.eclipse.equinox.weaving.hook | |
| - | TODO |
| Dsun.rmi.dgc.client.gcInterval=3600000 | |
| -D defaultItemHeigh= | Default presentation size (height) |
| -D defaultItemWidth= | Default presentation size (width) |

B.5 CBA ItemBuilder Versions

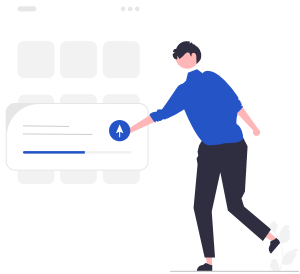
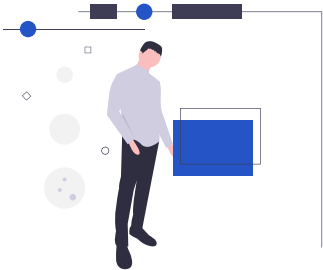


TABLE B.21: CBA ItemBuilder Versions (with [React](#)-based runtime)

| Version | Date | Links |
|---------|------------|--|
| 10.2 | 2024-12-09 | Runtime , JSON Schema , EE4Basic , Reference |
| 10.1 | 2024-04-18 | Runtime , JSON Schema , EE4Basic , Reference |
| 10.0 | 2023-11-06 | Runtime , JSON Schema , EE4Basic , Reference |
| 9.9 | 2023-03-23 | Runtime , JSON Schema , EE4Basic , Reference |
| 9.8 | 2022-09-22 | Runtime , JSON Schema |
| 9.7 | 2022-04-20 | Runtime , JSON Schema |
| 9.6 | 2021-12-16 | Runtime , JSON Schema |
| 9.5 | 2021-08-19 | Runtime , JSON Schema |
| 9.4 | 2021-05-28 | Runtime , JSON Schema |
| 9.3 | 2021-04-08 | Runtime , JSON Schema |
| 9.2 | 2020-12-18 | Runtime , JSON Schema |
| 9.1 | 2020-10-01 | Runtime , JSON Schema |
| 9.0 | 2020-06-17 | Runtime , JSON Schema |
| ... | ... | Previous versions not listed here |

B.6 Component Register



The following Table [B.22](#) contains references for all components of the CBA Item-Builder, indicating where in this manual the components are described (column *Reference*), which page type is required to use the component (column *Page*) and which containers allow using the component (column *Container*).

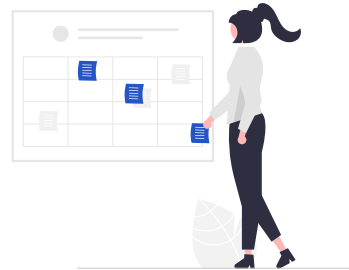
TABLE B.22: Register of all Components

| Component | Page | Container | Reference |
|-------------------|---------------------|---------------------------------|------------------------|
| Audio | Simple | Panel | 3.10.3 |
| BackButton | Web Browser | WebBrowserToolBar | 3.13.2 |
| Bookmark | Web Browser | WebBrowserToolBar | 3.13.2 |
| Button | Simple, Web Browser | Panel, WebBrowserToolBar, | 3.11.2 |
| | | ImageMap | |
| ChildArea | Tabfolder, Taskbar | TabfolderFrame, TaskbarFrame | 3.13.1 |
| Checkbox | Simple | Panel | 3.9.3 |
| ComboBox | Simple | Panel | 3.9.5 |
| ExternalPageFrame | Simple | Panel | 3.14.1 |
| ForwardedButton | Web Browser | WebBrowserToolBar | 3.13.2 |
| Frame | Simple | Page | 3.5.1 |
| HomeButton | Web Browser | WebBrowserToolBar | 3.13.2 |
| HTMLTextField | Simple, Web Browser | Panel, WebBrowserToolBar, | 3.8.2 |
| | | ImageMap | |
| InputField | Simple | Panel | 3.9.1 |
| ImageArea | Simple | ImageMap | 3.10.2 |
| ImageField | Simple, Web Browser | Panel, WebBrowserToolBar | 3.10.2 |
| ImageMap | Simple | Panel | 3.9.10 |
| ImageTextField | Simple | ImageMap | 3.9.10 |

| Component | Page | Container | Reference |
|-------------------------|---------------------|-----------------------------|------------------------|
| Line Horizontal | Simple, Web Browser | Panel, WebBrowserToolBar | 3.7.5 |
| Line Vertical | Simple, Web Browser | Panel, WebBrowserToolBar | 3.7.5 |
| Link | Simple | Panel | 3.11.1 |
| List | Simple | Panel | 3.9.5 |
| MapBasedVariableDisplay | Simple | Panel | 4.2.5 |
| MenuBar | Simple, Web Browser | Panel, WebBrowserToolBar | 3.9.7 |
| Menue | Simple, Web Browser | MenuBar | 3.9.7 |
| RadioButton | Simple | RadioButtonGroup | 3.9.2 |
| RadioButtonGroup | Simple | Panel | 3.9.2 |
| Rectangle | Simple | Panel | 3.7.5 |
| SimpleTextField | Simple | Panel | 3.8.1 |
| SingleLineInputField | Simple, Web Browser | Panel, WebBrowserToolBar | 3.9.1 |
| TabButton | Tabfolder | WebBrowserToolBar | |
| TabfolderFrame | Tabfolder | TabfolderGroup | 3.13.1 |
| TabfolderGroup | Tabfolder | Page | 3.13.1 |
| Table | Simple | TabfolderFrame | 3.13.1 |
| TableCellEditor | Simple | Panel | 3.9.8 |
| TaskbarButton | Taskbar | Panel | 3.9.8 |
| TaskbarFrame | Taskbar | TaskbarGroup | 3.13.1 |
| TaskbarGroup | Taskbar | Page | 3.13.1 |
| TaskbarStartButton | Taskbar | TaskbarFrame | 3.13.1 |
| | Taskbar | TaskbarGroup | 3.13.1 |

| Component | Page | Container | Reference |
|----------------------|---------------------|---------------------------------------|------------------------|
| TextField | Simple, Web Browser | Panel, WebBrowserToolBar | 3.8.3 |
| Tree | Simple | Panel | 3.9.9 |
| TreeView | Simple | Panel | 3.9.9 |
| TreeChildArea | Simple | Panel | 3.9.9 |
| Timer | Simple | Panel | 4.4.10 |
| PageArea | Simple | Frame | 3.5.4 |
| Panel | Simple | Frame, Panel | 3.5.2 |
| Rectangle | Simple, Web Browser | Panel, WebBrowserToolBar | 3.7.5 |
| ScaleValueInput | Simple | Panel | 4.2.2 |
| SpinnerValueInput | Simple | Panel | 4.2.2 |
| ValueInput | Simple | Panel | 4.2.2 |
| VariableValueDisplay | Simple | Panel | 4.2.5 |
| Video | Simple | Panel | 3.10.3 |
| VideoTextArea | Simple | Video | 3.10.3 |
| WebBrowserFrame | Web Browser | Page | 3.13.2 |
| WebBrowserToolBar | Web Browser | WebBrowserFrame, WebBrowserToolBar | 3.13.2 |
| WebChildArea | Web Browser | WebBrowserFrame | 3.13.2 |
| WebChildFrame | Web Child | Page | 3.13.2 |

B.7 Documentenation Log Events



During the delivery of assessment components created with the CBA ItemBuilder, log events are generated, which can be collected and stored by the software used for test deployment. The log events are delivered using the *TaskPlayer API* (see section 7.7) as JSON objects. The deployment software is responsible for storing and probably transforming the format of the data provided in log events. *Raw Log Events* are generated by default (see section 2.8.1). The documentation of the different *Raw Log Events* provided by CBA ItemBuilder is done separately for events of different components-categories:

- *Raw Log Events* for basic components (table B.23): Click events for buttons and links, page change and scroll events
- *Raw Log Events* for components that support selection (table B.24): Log events that occur when components in the test are clicked or selected
- *Raw Log Events* for answer-changes (table B.25): Log events, which can indicate the change of a response
- *Raw Log Events* for text entry components (table B.26): Log events for selection and modification of components for text entry
- *Raw Log Events* for value inputs (table B.27): Log events for value input components
- *Raw Log Events* for audio an video components (table B.28): Log events for player components used to embed media
- *Raw Log Events* for tree components (table B.29): Log events for the CBA ItemBuilder tree component
- *Raw Log Events* for advanced components (table B.30): Log events for advanced components
- *Raw Log Events* for text highlighting (table B.31): Log events for the text highlighting in `TextFields`
- *Raw Log Events* for test deployment (table B.32): Log events for the test deployment. Note that the deployment software can provide additional events.

As described in section 2.8.4 log data can be stored in different formats, such as the

Universal Log Format, Flat and Sparse Log Data Tables or as eXtensible Event Stream (XES). All formats use the The event name is either contained as a column (flat and sparse log data table) or corresponds to the table in which the log data of this event type is stored (universal log format).

The different tables list the events with different names (event type), give a short description, and name the central event-specific data transmitted by the events. Optional attributes are specially marked. Beyond the documented attributes, log events often provide the following additional attributes:

- indexPath: The index path of the component instance.
- userDefIdPath: The user defined ID path of the component instance.
- clientX/clientY/pageX/pageY/screenX/screenY: The position of the element in different metrics.

These attributes' values reproduce properties of the affected components and can be used to map the affected parts of the instrument (beyond the UserDefinedID).¹

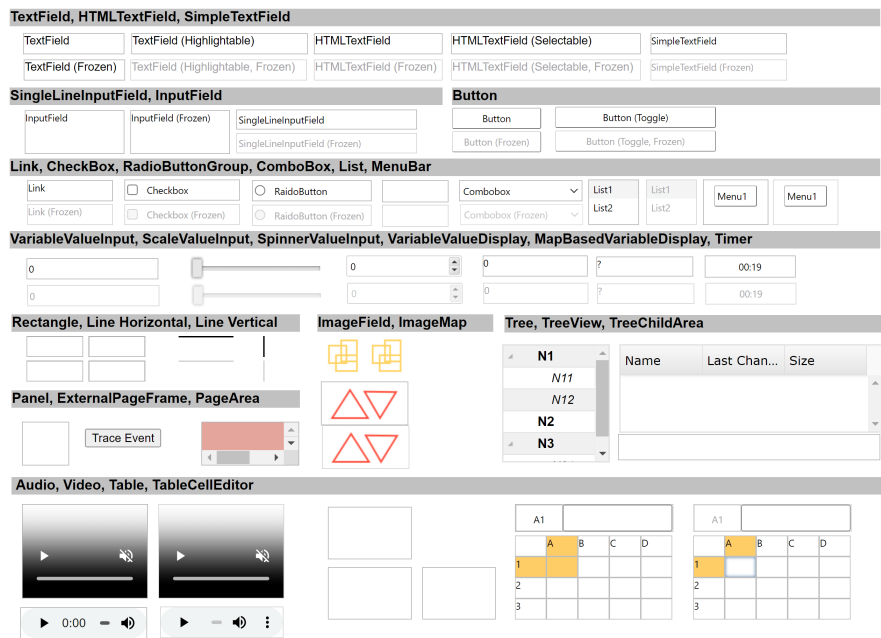


FIGURE B.1: Example item illustrating Raw Log-Events ([html|ib](#)).

¹Two additional events are not included in the tables: The event snapshot provides a snapshot (i.e., a restoring point for the current task) and the vent Recommend is currently marked as obsolete.

TABLE B.23: Log events for basic elements

| Event Name | Description | Event-Specific Data |
|--------------------|--|--|
| Button | Button is pressed. | UserDefId of the button, oldSelected (was the button selected before), subtype ¹ (type of the button) |
| Link | Link is hit. | UserDefId of the Link, oldSelected (was the Link selected before) |
| ValueDisplay | Value display is selected. | UserDefId of ValueDisplay, displayType (type of the display, either number, text, image icon, audio or video) |
| EmbeddedLink | Embedded link is selected. | indexPath (Index path of the embedded link), no UserDefId |
| ScrollbarMove | Scrollbar slider moved. | userDefId of the component, orientation (either horizontal or vertical), direction (either down, up, left, or right), horizontalScroll (position of the horizontal slider at the end of the move in percent of total size, 0% is at the left end, 100% is at the right end), verticalScroll (position of the vertical slider at the end of the move in percent of total size, 0% is at the top, 100% is at the bottom) |
| PageSwitchTopLevel | A page switch in a top level page area occurs. | pageAreaType (type of the page area, either main, dialog or modal), pageAreaName (name of the page area), newPageName (name of the page displayed in the page area), position ¹ (X/Y) |
| PageSwitchEmbedded | A page switch in an embedding component instance occurs. | userDefId of the component, newPageName (name of the new page embedded in the component instance), tab ¹ (name of the target tab in the embedding component instance), historyMove ¹ (history move operation that triggered the page switch, either home, back, or forward) |

¹ Optional event-specific data.

TABLE B.24: Log events for component selection

| Event Name | Description | Event-Specific Data |
|-------------------|-------------------------------------|------------------------------------|
| SimpleTextField | Simple text field is selected. | UserDefId of the SimpleTextField |
| ImageField | Image field is selected. | UserDefId of the ImageField |
| ExternalPageFrame | An external page frame is selected. | UserDefId of the ExternalPageFrame |
| Panel | Panel is selected. | UserDefId of the Panel |
| Container | Container is selected. | UserDefId of the Container |
| RegionMap | Region map is selected. | UserDefId of RegionMap |
| PageArea | Page area is selected. | UserDefId of PageArea |

TABLE B.25: Log events for answer-changes

| Event Name | Description | Event-Specific Data |
|-------------|--|---|
| Checkbox | Checkbox is pressed. | UserDefId of the Checkbox, oldSelected (was the Checkbox selected before) |
| RadioButton | RadioButton is pressed. | UserDefId of the RadioButton, oldSelected (was the RadioButton selected before) |
| Combobox | Combo box selected entry changes. The combo box will not trace an event if the user selects the currently selected item once more. | UserDefId of the Combobox, oldSelected (index of the previously selected combo box item), oldSelectedUserDefId (user defined ID of the previously selected item), newSelected (index of the now selected combo box item), newSelectedUserDefId (user defined ID of the now selected item) |

| Event Name | Description | Event-Specific Data |
|----------------|-------------------------------|---|
| ImageArea | Image area is selected. | UserDefld of ImageArea, oldSelected (was the ImageArea selected before) |
| ImageTextField | Image text field is selected. | UserDefld of ImageTextField, oldSelected (was the ImageTextField selected before) |
| RichText | Rich text field is selected. | UserDefld of RichText, oldSelected (was the RichText selected before) |

¹ Optional event-specific data.

TABLE B.26: Log events for text entry components

| Event Name | Description | Event-Specific Data |
|------------------------------|--|---|
| SingleLineInputField | Single line input field is selected. | UserDefld of SingleLineInputField, currentTextView (text value of the input field) |
| SingleLineInputFieldModified | Single line input field receives modification request. | UserDefld of SingleLineInputFieldModified, oldTextView (old text value of the input field), newTextView (new text value of the input field), origin (origin of the modification request, either keyboard or cutAndPaste), validationPattern ¹ (validation pattern configured for the field), invalidTextView ¹ (text value that was rejected by the pattern validation) |
| InputField | Input field is selected. | UserDefld of InputField, currentTextView (text value of the input field) |

| Event Name | Description | Event-Specific Data |
|----------------------|--|--|
| InputFieldModified | Input field receives modification request. | UserDefId of InputFieldModified, oldTextValue (old text value of the input field), newTextValue (new text value of the input field), origin (origin of the modification request, either keyboard or cutAndPaste), validationPattern ¹ (validation pattern configured for the field), invalidTextValue ¹ (text value that was rejected by the pattern validation) |
| OperatorSetTextValue | Operator sets text value of input element. | UserDefId of component, oldTextValue (old text value of the input field), newTextValue (new text value of the input field) |

¹ Optional event-specific data.

TABLE B.27: Log events for value inputs

| Event Name | Description | Event-Specific Data |
|--------------------|---|--|
| ValueInput | Variable value input field is selected. | UserDefId of ValueInput |
| ValueInputModified | Variable value Input field is modified. | UserDefId of ValueInput, newValue (new value for the variable) |
| ScaleValueInput | Scale value input field is selected. | UserDefId of ScaleValueInput |
| SpinnerValueInput | Spinner value input field is selected. | UserDefId of SpinnerValueInput |

TABLE B.28: Log events for audio/video components

| Event Name | Description | Event-Specific Data |
|--------------------|---|--|
| AudioPlayer | Audio player receives click with hidden controls. | UserDefId of AudioPlayer |
| AudioPlayerControl | Audio player starts/pauses/ends playing. | UserDefId of AudioPlayerControl, operation (either play, pause, stop, or ended), maxPlay (maximum number of plays allowed), currentPlayNo (number of the current play), automaticStart (true if automatic start), hideControls (true if controls are hidden), volumeLevel (current volume level in percent), isStateMachineTriggerred (true if triggered by FSM) |
| VideoPlayer | Video player receives click with hidden controls. | UserDefId of VideoPlayer |
| VideoPlayerControl | Video player starts/pauses/ends playing. | UserDefId of AudioPlayerControl, operation (either play, pause, stop, or ended), maxPlay (maximum number of plays allowed), currentPlayNo (number of the current play), automaticStart (true if automatic start), hideControls (true if controls are hidden), volumeLevel (current volume level in percent), isStateMachineTriggerred (true if triggered by FSM) |

TABLE B.29: Log events for tree components

| Event Name | Description | Event-Specific Data |
|---------------|--|--|
| TreeNode | Operation on node in tree. | UserDefId of the tree, operation (type of operation, either emptySelection, selection, doubleClick, expandNode, collapseNode, new, delete, rename, cut, copy, paste, drag, or drop), nodePathId ¹ (node path id of selected node), nodeType ¹ (type of selected node), nodeName (name of selected node), oldValue ¹ (for operation=rename only: old column value), newValue ¹ (for operation=rename only: new column value), columnName ¹ (for operation=rename only: name of column changed by the operation), triggeredEvent ¹ (for operation=delete only: name of the FSM event triggered instead of executing the operation) |
| TreeNode | Operation on node in tree view. | UserDefId of the tree, operation (type of operation, either emptySelection, selection, doubleClick, expandNode, collapseNode, new, delete, rename, cut, copy, paste, drag, or drop), nodePathId ¹ (node path id of selected node), nodeType ¹ (type of selected node), nodeName (name of selected node), oldValue ¹ (for operation=rename only: old column value), newValue ¹ (for operation=rename only: new column value), columnName ¹ (for operation=rename only: name of column changed by the operation), triggeredEvent ¹ (for operation=delete only: name of the FSM event triggered instead of executing the operation) |
| TreeViewSort | Sort by column operation in tree view. | UserDefId of the tree, sortDirection (direction of the sorting applied to the rows, either ascending, descending, or none), columnName ¹ (name of the column to sort rows by), columnIndex ¹ (index of the column to sort rows by) |
| TreeChildArea | Tree child area receives click | UserDefId of the tree |

¹ Optional event-specific data.

TABLE B.30: Log events for advanced components

| Event Name | Description | Event-Specific Data |
|--------------------|--|--|
| TableCell | Table cell is selected. | UserDefId of the TableCell, tableUserDefId (userDefId of the table containing the cell), row (index of the row of the cell), column (index of the column of the cell), oldSelected (was the TableCell selected before) |
| TableCellModified | Table cell is modified. | UserDefId of the TableCell, tableUserDefId (userDefId of the table containing the cell), row (index of the row of the cell), column (index of the column of the cell), cellType (text combo or formula), oldValue (old value of the cell), newValue (new value of the cell), oldEvaluatedValue (evaluated value of the old formula), newEvaluatedValue (evaluated value of the new formula), errorInFormula (error value returned by the formula evaluation for the new formula) |
| BrowserTab | Switch the current Browser tab. Event is raised by selecting the tab. | UserDefId of the web child area component, tab (name of the selected tab), page (name of the page displayed in the selected tab) |
| DragAndDropReceive | Drag & drop operation completed. | senderUserDefId ¹ (user defined ID of the sending component), receiverUserDefId ¹ (user defined ID of the receiving component), startPosition (X/Y), endPosition (X/Y), sendingType, receivingType, operation |
| JavaScriptInjected | Some external JavaScript code triggered this entry in the trace log. | userDefId ¹ (as specified by the JavaScript code), origin (origin URL of the event triggered by the JavaScript code), message ¹ (as specified by the JavaScript code) |

| Event Name | Description | Event-Specific Data |
|-----------------------|---|---|
| CutCopyPaste | A cut/copy/paste command was triggered. | contentUserDefId ¹ (ser defined ID of the component providing/receiving the clipboard content), triggerUserDefId ¹ (user defined ID of the triggering component), triggerType (type of the triggering action, either button, contextMenu, or keyboard), operation (operation triggered, either cut, copy, or paste), content ¹ (xchanged clipboard content, not given for denied requests), isPerformed ¹ (indicates whether the requested operation was performed or denied) |
| Bookmark | A bookmark related command triggered. | ownerUserDefId ¹ (user defined ID of the component owning the bookmarks), triggerUserDefId ¹ (user defined ID of the triggering component), triggerType (type of the triggering action, either button or contextMenu), operation (operation triggered, either add, drop, select, or manage), pageName ¹ (name of the target page of the bookmark), pageUrl ¹ (URL of the target page of the bookmark), tab ¹ (tab in the browser targeted by the bookmark, for tabbed browsers only) |
| OperatorTraceText | Operator trace_text includes text in trace log. | text (evaluated text from the operator arguments) |
| OperatorTraceSnapshot | Operator trace_snapshot includes text in trace log. | text (evaluated text from the operator arguments) |

¹ Optional event-specific data.

TABLE B.31: Log events for text highlighting

| Event Name | Description | Event-Specific Data |
|-------------------|--|--|
| RichTextHighlight | Highlighted area in rich text field changes. | UserDefId of the RichTextHighlight, oldSelections (list), newSelections (list) |

Event-specific data provided in the list of `oldSelections` and `newSelections` have the following attributes:

- `startKey`: The ID of the block (i.e. the row) where the area starts.
- `startOffset`: The position in the block (i.e. in the row) where the area starts.
- `endKey`: The ID of the block (i.e. the row) where the area ends.
- `endOffset`: The position in the block (i.e. in the row) where the area ends.

TABLE B.32: Log events for test deployment

| Event Name | Description | Event-Specific Data |
|------------------|---|---|
| TasksViewVisible | The tasks view becomes visible. The view showing the task(s) becomes visible for the test user. | Settings for running the task |
| UserLogin | A test-taker logged in. | User name, information about the browser used to display the test |
| ItemSwitch | An item switch occurs. | Configuration of the new item |

| Event Name | Description | Event-Specific Data |
|-----------------------|---|---|
| TaskSwitch | A task switch occurs. | oldTask (name of the interrupted tas), oldItem (name of the item containing the interrupted task), oldTest (name of the test of the interrupted task), newTask (name of the installed task), newItem (name of the item containing the installed task), newTest (name of the test of the installed task), taskResult (results for the named evaluations for the interrupted task) index (index of the header buttons in the header buttons list) navigationType (type of the navigation button, either test or task), navigationTarget (name of the test or task to navigate to) |
| HeaderButton | One of the header buttons provided by the viewer was pressed. | |
| NavigationButton | One of the task/test navigation buttons provided by the viewer was pressed. | |
| RuntimeController | The controlling environment triggered an action. | actionType (triggered action), details (parameters specifying the details of the triggered action) |
| ApplicationFullScreen | A fullscreen related command triggers. | type (type of operation triggered, either enterFullScreen or exitFullScreen), alternateStateDuration (duration in seconds since the last visibility change) |
| ApplicationVisibility | Application visibility changed. | type (type of change, either pageHidden or pageShown), alternateStateDuration (duration in seconds since the last visibility change) |

| Event Name | Description | Event-Specific Data |
|-------------|---|--|
| PauseResume | Test run was paused or resumed after pause. | type (type of action, either <code>pause</code> or <code>resume</code>) |

¹ Optional event-specific data.



Bibliography

- (2022). Question and Test Interoperability (QTI): Implementation Guide. http://www.imsglobal.org/question/qtiv2p2/imsqti_v2p2_impl.html.
- Alagar, V. S. and Periyasamy, K. (2011). *Specification of Software Systems*. Texts in Computer Science. Springer, New York, 2nd ed edition.
- Attali, Y. (2011). Immediate Feedback and Opportunity to Revise Answers: Application of a Graded Response IRT Model. *Applied Psychological Measurement*, 35(6):472–479.
- Attali, Y., Runge, A., LaFlair, G. T., Yancey, K., Goodwin, S., Park, Y., and von Davier, A. A. (2022). The interactive reading task: Transformer-based automatic item generation. *Frontiers in Artificial Intelligence*, 5:903077.
- Baarsen, J. V. (2014). *GitLab Cookbook*.
- Baghaei, P. and Tabatabaee, M. (2015). The C-Test: An Integrative Measure of Crystallized Intelligence. *Journal of Intelligence*, 3(2):46–58.
- Bartram, D. (2005). Testing on the Internet: Issues, Challenges and Opportunities in the Field of Occupational Assessment. In Bartram, D. and Hambleton, R. K., editors, *Computer-Based Testing and the Internet*, pages 13–37. John Wiley & Sons, Ltd, West Sussex, England.
- Bartram, D. and Hambleton, R. K., editors (2006). *Computer-Based Testing and the Internet: Issues and Advances*. Wiley, Chichester.
- Becker, B., Debeer, D., Sachse, K. A., and Weirich, S. (2021). Automated Test Assembly in R: The eatATA Package. *Psych*, 3(2):96–112.
- Bengs, D., Kroehne, U., and Brefeld, U. (2021). Simultaneous Constrained Adaptive Item Selection for Group-Based Testing. *Journal of Educational Measurement*, 58(2):236–261.
- Bennett, R. E., Braswell, J., Oranje, A., Sandene, B., Kaplan, B., and Yan, F. (2008). Does it Matter if I Take My Mathematics Test on Computer? A Second Empirical Study of Mode Effects in NAEP. page 39.
- Böckenholt, U. and Meiser, T. (2017). Response style analysis with threshold and multi-process IRT models: A review and tutorial. *British Journal of Mathematical and Statistical Psychology*, 70(1):159–181.

- Bolt, D. (2016). Item response models for CBT. In Drasgow, F., editor, *Technology and Testing: Improving Educational and Psychological Measurement*, page 305. Routledge.
- Born, S. and Frey, A. (2017). Heuristic Constraint Management Methods in Multidimensional Adaptive Testing. *Educational and Psychological Measurement*, 77(2):241–262.
- Bridgeman, B. (2009). Experiences from large-scale computer-based testing in the USA. *The transition to computer-based assessment*, 39.
- Bryant, W. (2017). Developing a Strategy for Using Technology-Enhanced Items in Large-Scale Standardized Tests.
- Buchanan, T. (2002). Online assessment: Desirable or dangerous? *Professional Psychology: Research and Practice*, 33(2):148–154.
- Buerger, S., Kroehne, U., and Goldhammer, F. (2016). The transition to computer-based testing in large-scale assessments: Investigating (partial) measurement invariance between modes.
- Buerger, S., Kroehne, U., Koehler, C., and Goldhammer, F. (2019). What makes the difference? The impact of item properties on mode effects in reading assessments. *Studies in Educational Evaluation*, 62:1–9.
- Bugbee, A. C. (1996). The equivalence of paper-and-pencil and computer-based testing. *Journal of Research on Computing in Education*, 28(3):282.
- Christensen, G. S., Freese, J., and Miguel, E. (2019). *Transparent and Reproducible Social Science Research: How to Do Open Science*. University of California Press, Oakland, California.
- Clariana, R. and Wallace, P. (2002). Paper-based versus computer-based assessment: Key factors associated with the test mode effect. *British Journal of Educational Technology*, 33(5):593–602.
- Cochran, G. L., Foster, J. A., Klepser, D. G., Dobesh, P. P., and Dering-Anderson, A. M. (2020). The Impact of Eliminating Backward Navigation on Computerized Examination Scores and Completion Time. *American Journal of Pharmaceutical Education*, 84(12):ajpe8034.
- Dann, P. L., Irvine, S. H., and Collis, J. M., editors (1991). *Advances in Computer-Based Human Assessment*. Springer Science+Business Media, Dordrecht.
- Das, B., Majumder, M., Phadikar, S., and Sekh, A. A. (2021). Automatic question generation and answer assessment: A survey. *Research and Practice in Technology Enhanced Learning*, 16(1):5.
- Deribo, T., Kroehne, U., and Goldhammer, F. (2021). Model-Based Treatment of Rapid Guessing. *Journal of Educational Measurement*, 58(2):281–303.

- Diao, Q. and van der Linden, W. J. (2011). Automated Test Assembly Using Ip_Solve Version 5.5 in R. *Applied Psychological Measurement*, 35(5):398–409.
- DiBattista, D. (2013). The Immediate Feedback Assessment Technique: A Learner-centered Multiple-choice Response Form. *Canadian Journal of Higher Education*, 35(4):111–131.
- DiCerbo, K., Lai, E., and Matthew, V. (2020). Assessment design with automated scoring in mind. In *Handbook of Automated Scoring*, pages 29–48. Chapman and Hall/CRC.
- Dirk, J., Kratzsch, G. K., Prindle, J. P., Kroehne, U., Goldhammer, F., and Schmiedek, F. (2017). Paper-Based Assessment of the Effects of Aging on Response Time: A Diffusion Model Analysis. *Journal of Intelligence*, 5(2):12.
- Downing, S. M. and Haladyna, T. M., editors (2006). *Handbook of Test Development*. L. Erlbaum, Mahwah, N.J.
- Ebel, R. L. (1953). The Use of Item Response Time Measurements in the Construction of Educational Achievement Tests. *Educational and Psychological Measurement*, 13(3):391–401.
- Embretson, S. and Reise, S. P. (2013). *Item Response Theory*. Psychology Press.
- Embretson, S. and Yang, X. (2006). Automatic Item Generation and Cognitive Psychology. In *Handbook of Statistics*, volume 26, pages 747–768. Elsevier.
- Feskens, R., Fox, J.-P., and Zwitser, R. (2019). Differential item functioning in PISA due to mode effects. In Veldkamp, B. P. and Sluijter, C., editors, *Theoretical and Practical Advances in Computer-Based Educational Measurement*, pages 231–247. Springer International Publishing, Cham.
- Fink, A., Born, S., Spoden, C., and Frey, A. (2018). A continuous calibration strategy for computerized adaptive testing. *Psychological Test and Assessment Modeling*, 3(60):327–346.
- Finn, B. and Metcalfe, J. (2010). Scaffolding feedback to maximize long-term error correction. *Memory & Cognition*, 38(7):951–961.
- Fishbein, B., Martin, M. O., Mullis, I. V. S., and Foy, P. (2018). The TIMSS 2019 Item Equivalence Study: Examining mode effects for computer-based assessment and implications for measuring trends. *Large-scale Assessments in Education*, 6(1):11.
- Frey, A., Hartig, J., and Rupp, A. A. (2009). An NCME Instructional Module on Booklet Designs in Large-Scale Assessments of Student Achievement: Theory and Practice. *Educational Measurement: Issues and Practice*, 28(3):39–53.
- Frey, A., Spoden, C., Goldhammer, F., and Wenzel, S. F. C. (2018). Response time-based treatment of omitted responses in computer-based testing. *Behaviormetrika*, 45(2):505–526.

- Frey, B. B., Schmitt, V. L., and Allen, J. P. (2012). Defining Authentic Classroom Assessment.
- Gabadinho, A., Ritschard, G., Müller, N. S., and Studer, M. (2011). Analyzing and Visualizing State Sequences in R with **TraMineR**. *Journal of Statistical Software*, 40(4).
- Gandrud, C. (2020). *Reproducible Research with R and RStudio*. The R Series. CRC Press, Boca Raton, FL, third edition edition.
- George, A. C. and Robitzsch, A. (2015). Cognitive Diagnosis Models in R: A didactic. *The Quantitative Methods for Psychology*, 11(3):189–205.
- Gierl, M. J. and Lai, H. (2013). Instructional Topics in Educational Measurement (ITEMS) Module: Using Automated Processes to Generate Test Items. *Educational Measurement: Issues and Practice*, 32(3):36–50.
- Gierl, M. J., Lai, H., and Turner, S. R. (2012). Using automatic item generation to create multiple-choice test items: Automatic generation of test items. *Medical Education*, 46(8):757–765.
- Gobert, J. D., Sao Pedro, M., Raziuddin, J., and Baker, R. S. (2013). From Log Files to Assessment Metrics: Measuring Students' Science Inquiry Skills Using Educational Data Mining. *Journal of the Learning Sciences*, 22(4):521–563.
- Goldhammer, F. (2015). Measuring Ability, Speed, or Both? Challenges, Psychometric Solutions, and What Can Be Gained From Experimental Control. *Measurement: Interdisciplinary Research and Perspectives*, 13(3-4):133–164.
- Goldhammer, F., Hahnel, C., and Kroehne, U. (2020). Analysing Log File Data from PIAAC. In Maehler, D. B. and Rammstedt, B., editors, *Large-Scale Cognitive Assessment: Analyzing PIAAC Data*. Springer, Cham.
- Goldhammer, F., Hahnel, C., Kroehne, U., and Zehner, F. (2021). From byproduct to design factor: On validating the interpretation of process indicators based on log data. *Large-scale Assessments in Education*, 9(1):20.
- Goldhammer, F. and Kroehne, U. (2020). Computerbasiertes Assessment. In Moosbrugger, H. and Kelava, A., editors, *Testtheorie und Fragebogenkonstruktion*, pages 119–141. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Goldhammer, F., Martens, T., and Lüdtke, O. (2017). Conditioning factors of test-taking engagement in PIAAC: An exploratory IRT modelling approach considering person and item characteristics. *Large-scale Assessments in Education*, 5(1):18.
- Goldhammer, F., Naumann, J., Stelter, A., Tóth, K., Rölke, H., and Klieme, E. (2014). The time on task effect in reading and problem solving is moderated by task difficulty and skill: Insights from a computer-based large-scale assessment. *Journal of Educational Psychology*, 106(3):608–626.

- Goldhammer, F. and Zehner, F. (2017). What to Make Of and How to Interpret Process Data. *Measurement: Interdisciplinary Research and Perspectives*, 15(3-4):128–132.
- Gong, T., Jiang, Y., Saldivia, L. E., and Agard, C. (2022). Using Sankey diagrams to visualize drag and drop action sequences in technology-enhanced items. *Behavior Research Methods*, 54(1):117–132.
- Gorgun, G. and Bulut, O. (2021). A Polytomous Scoring Approach to Handle Not-Reached Items in Low-Stakes Assessments. *Educational and Psychological Measurement*, 81(5):847–871.
- Greiff, S., Niepel, C., Scherer, R., and Martin, R. (2016). Understanding students' performance in a computer-based assessment of complex problem solving: An analysis of behavioral data from computer-generated log files. *Computers in Human Behavior*, 61:36–46.
- Hahnel, C., Eichmann, B., and Goldhammer, F. (2020). Evaluation of Online Information in University Students: Development and Scaling of the Screening Instrument EVON. *Frontiers in Psychology*, 11:562128.
- Hahnel, C., Jung, A. J., and Goldhammer, F. (2023). Theory Matters: An Example of Deriving Process Indicators From Log Data to Assess Decision-Making Processes in Web Search Tasks. *European Journal of Psychological Assessment*, 39(4):271–279.
- Hahnel, C., Kroehne, U., Goldhammer, F., Schoor, C., Mahlow, N., and Artelt, C. (2019). Validating process variables of sourcing in an assessment of multiple document comprehension. *British Journal of Educational Psychology*, page bjep.12278.
- Hahnel, C., Ramalingam, D., Kroehne, U., and Goldhammer, F. (2022). Patterns of reading behaviour in digital hypertext environments. *Journal of Computer Assisted Learning*, page jcal.12709.
- Haigh, M. (2010). Why use computer-based assessment in education? A literature review. (10):8.
- Hao, J., Shu, Z., and von Davier, A. (2015). Analyzing process data from game/scenario-based tasks: An edit distance approach. *JEDM-Journal of Educational Data Mining*, 7(1):33–50.
- Harsch, C. and Hartig, J. (2016). Comparing C-tests and Yes/No vocabulary size tests as predictors of receptive language skills. *Language Testing*, 33(4):555–575.
- Haverkamp, Y. E., Bråten, I., Latini, N., and Salmerón, L. (2022). Is it the size, the movement, or both? Investigating effects of screen size and text movement on processing, understanding, and motivation when students read informational text. page 20.
- Hethley, J. M. (2013). *GitLab Repository Management: Delve into Managing Your Projects with GitLab, While Tailoring It to Fit Your Environment*. Community Experience Distilled. Packt Publ, Birmingham.

- Hetter, R. D. and Sympson, J. B. (1997). Item exposure control in CAT-ASVAB. In Sands, W. A., Waters, B. K., and McBride, J. R., editors, *Computerized Adaptive Testing: From Inquiry to Operation.*, pages 141–144. American Psychological Association, Washington.
- Heyne, N., Artelt, C., Gnabbs, T., Gehrler, K., and Schoor, C. (2020). Instructed highlighting of text passages – Indicator of reading or strategic performance? *Lingua*, 236:102803.
- Hornke, L. F. (2005). Response time in computer-aided testing: A “Verbal Memory” test for routes and maps. page 14.
- Ihme, J. M., Senkbeil, M., Goldhammer, F., and Gerick, J. (2017). Assessment of computer and information literacy in ICILS 2013: Do different item types measure the same construct? *European Educational Research Journal*, 16(6):716–732.
- International Test Commission and Association of Test Publishers (2022). *Guidelines for Technology-Based Assessment*.
- Jaeger, J. (2018). Digit Symbol Substitution Test: The Case for Sensitivity Over Specificity in Neuropsychological Testing. *Journal of Clinical Psychopharmacology*, 38(5):513–519.
- Jiang, Y., Gong, T., Saldivia, L. E., Cayton-Hodges, G., and Agard, C. (2021). Using process data to understand problem-solving strategies and processes for drag-and-drop items in a large-scale mathematics assessment. *Large-scale Assessments in Education*, 9(1):2.
- Jiao, H., Liao, D., and Zhan, P. (2019). Utilizing Process Data for Cognitive Diagnosis. In von Davier, M. and Lee, Y.-S., editors, *Handbook of Diagnostic Classification Models*, pages 421–436. Springer International Publishing, Cham.
- Jung, S., Heller, J., Moeller, K., and Klein, E. (2019). Mode effect: An issue of perspective? Writing mode differences in a spelling assessment in German children with and without developmental dyslexia. page 38.
- Jurecka, A. (2008). Introduction to the computer-based assessment of competencies. *Assessment of competencies in educational contexts*, pages 193–214.
- Kirsch, I. and Lennon, M. L. (2017). PIAAC: A new design for a new era. *Large-scale Assessments in Education*, 5(1):11.
- Koehler, C., Pohl, S., and Carstensen, C. H. (2014). Taking the Missing Propensity Into Account When Estimating Competence Scores: Evaluation of Item Response Theory Models for Nonignorable Omissions. *Educational and Psychological Measurement*.
- Kreuter, F., editor (2013). *Improving Surveys with Paradata: Analytic Uses of Process Information*. Wiley Series in Survey Methodology. Wiley & Sons, Hoboken, New Jersey.

- Kroehne, U. (In Preparation). Standardization of Log Data from Computer-Based Assessments.
- Kroehne, U., Buerger, S., Hahnel, C., and Goldhammer, F. (2019a). Construct Equivalence of PISA Reading Comprehension Measured With Paper-Based and Computer-Based Assessments. *Educational Measurement: Issues and Practice*, page emip.12280.
- Kroehne, U., Deribo, T., and Goldhammer, F. (2020). Rapid Guessing Rates Across Administration Mode and Test Setting. *Psychological Test and Assessment Modeling*, 62(2):147–177.
- Kroehne, U., Gnambs, T., and Goldhammer, F. (2019b). *Disentangling Setting and Mode Effects for Online Competence Assessment*, pages 171–193. Edition ZfE. Springer VS, Wiesbaden.
- Kroehne, U. and Goldhammer, F. (2018). How to conceptualize, represent, and analyze log data from technology-based assessments? A generic framework and an application to questionnaire items. *Behaviormetrika*.
- Kroehne, U. and Goldhammer, F. (in Press). Tools for Analyzing Log-File Data.
- Kroehne, U., Goldhammer, F., and Partchev, I. (2014). Constrained Multidimensional Adaptive Testing without intermixing items from different dimensions. *Psychological Test and Assessment Modeling*, 56(4):348.
- Kroehne, U., Hahnel, C., and Goldhammer, F. (2019c). Invariance of the Response Processes Between Gender and Modes in an Assessment of Reading. *Frontiers in Applied Mathematics and Statistics*, 5:2.
- Kroehne, U. and Martens, T. (2011). Computer-based competence tests in the national educational panel study: The challenge of mode effects. *Zeitschrift für Erziehungswissenschaft*, 14(S2):169–186.
- Kyllonen, P. and Zu, J. (2016). Use of Response Time for Measuring Cognitive Ability. *Journal of Intelligence*, 4(4):14.
- Lahza, H., Smith, T. G., and Khosravi, H. (2022). Beyond item analysis: Connecting student behaviour and performance using e-assessment logs. *British Journal of Educational Technology*, page bjet.13270.
- Lane, S., Raymond, M. R., and Haladyna, T. M., editors (2015). *Handbook of Test Development*. Routledge, New York, second edition edition.
- Lesyuk, A. (2013). *Mastering Redmine*. Packt Publishing, Birmingham, UK.
- Linden, W. J. and Diao, Q. (2011). Automated Test-Form Generation. *Journal of Educational Measurement*, 48(2):206–222.

- Lu, J., Wang, C., Zhang, J., and Tao, J. (2019). A mixture model for responses and response times with a higher-order ability structure to detect rapid guessing behaviour. *British Journal of Mathematical and Statistical Psychology*, page bmsp.12175.
- Magis, D. and Barrada, J. R. (2017). Computerized Adaptive Testing with R : Recent Updates of the Package **catR**. *Journal of Statistical Software*, 76(Code Snippet 1).
- Magis, D. and Raîche, G. (2012). Random generation of response patterns under computerized adaptive testing with the R package catR. *Journal of Statistical Software*, 48(8):1–31.
- Mason, M. (2006). *Pragmatic Version Control Using Subversion*. Number v. 1 in Pragmatic Starter Kit. Pragmatic Bookshelf, Raleigh, N.C, 2nd ed edition.
- Mayerl, J. (2013). Response latency measurement in surveys. Detecting strong attitudes and response effects. *Survey Methods: Insights from the Field (SMIF)*.
- Mills, C. N., editor (2002). *Computer-Based Testing: Building the Foundation for Future Assessments*. L. Erlbaum Associates, Mahwah, N.J.
- Naglieri, J. A., Drasgow, F., Schmit, M., Handler, L., Prifitera, A., Margolis, A., and Velasquez, R. (2004). Psychological testing on the Internet: New problems, old issues. *American Psychologist*, 59(3):150.
- Naumann, J. (2015). A model of online reading engagement: Linking engagement, navigation, and performance in digital reading. *Computers in Human Behavior*, 53:263–277.
- Naumann, J. and Goldhammer, F. (2017). Time-on-task effects in digital reading are non-linear and moderated by persons' skills and tasks' demands. *Learning and Individual Differences*, 53:1–16.
- Naumann, J. and Sälzer, C. (2017). Digital reading proficiency in german 15-year olds: Evidence from PISA 2012. *Zeitschrift für Erziehungswissenschaft*, 20(4):585–603.
- Neubert, J. C., Kretschmar, A., Wüstenberg, S., and Greiff, S. (2015). Extending the Assessment of Complex Problem Solving to Finite State Automata: Embracing Heterogeneity. *European Journal of Psychological Assessment*, 31(3):181–194.
- OECD (2013). *The Survey of Adult Skills: Reader's Companion*. OECD.
- OECD (2019). *Beyond Proficiency: Using Log Files to Understand Respondent Behaviour in the Survey of Adult Skills*. OECD Skills Studies. OECD.
- Parshall, C. G., editor (2002). *Practical Considerations in Computer-Based Testing*. Springer, New York.
- Partchev, I. (2004). A visual guide to item response theory. Retrieved November, 9:2004.

- Pavic, A. (2016). *Redmine Cookbook: Over 80 Hands-on Recipes to Improve Your Skills in Project Management, Team Management, Process Improvement, and Redmine Administration*.
- Persic-Beck, L., Goldhammer, F., and Kroehne, U. (2022). Disengaged response behavior when the response button is blocked: Evaluation of a micro-intervention. *Frontiers in Psychology*, 13:954532.
- Phillips, A. and Davis, M. (2009). Tags for identifying languages. Technical report.
- Pohl, S. (2013). Longitudinal Multistage Testing. *Journal of Educational Measurement*, 50(4):447–468.
- Reips, U.-D. (2010). Design and formatting in Internet-based research. In Gosling, S. and Johnson, J., editors, *Advanced Methods for Conducting Online Behavioral Research*, pages 29–43. Washington, DC: American Psychological Association.
- Reis Costa, D., Bolsinova, M., Tijmstra, J., and Andersson, B. (2021). Improving the Precision of Ability Estimates Using Time-On-Task Variables: Insights From the PISA 2012 Computer-Based Assessment of Mathematics. *Frontiers in Psychology*, 12:579128.
- Reis Costa, D. and Leoncio, W. (2019). *LOGAN: Log File Analysis in International Large-Scale Assessments*.
- Rios, J. A. (2022). Assessing the Accuracy of Parameter Estimates in the Presence of Rapid Guessing Misclassifications. *Educational and Psychological Measurement*, 82(1):122–150.
- Rios, J. A., Guo, H., Mao, L., and Liu, O. L. (2017). Evaluating the Impact of Careless Responding on Aggregated-Scores: To Filter Unmotivated Examinees or Not? *International Journal of Testing*, 17(1):74–104.
- Rios, J. A. and Soland, J. (2021). Parameter Estimation Accuracy of the Effort-Moderated Item Response Theory Model Under Multiple Assumption Violations. *Educational and Psychological Measurement*, 81(3):569–594.
- Rios, J. A. and Soland, J. (2022). An investigation of item, examinee, and country correlates of rapid guessing in PISA. *International Journal of Testing*, pages 1–31.
- Robitzsch, A., Kiefer, T., and Wu, M. (2022). *TAM: Test Analysis Modules*.
- Robitzsch, A. and Lüdtke, O. (2022). Some thoughts on analytical choices in the scaling model for test scores in international large-scale assessment studies. *Measurement Instruments for the Social Sciences*, 4(1):9.
- Robitzsch, A., Lüdtke, O., Goldhammer, F., Kroehne, U., and Köller, O. (2020). Re-analysis of the German PISA data: A comparison of different approaches for trend estimation with a particular emphasis on mode effects. *Frontiers in Psychology*, 11(884).

- Rölke, H. (2012). The ItemBuilder: A Graphical Authoring System for Complex Item Development. In *World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, volume 2012, pages 344–353.
- Rose, N., von Davier, M., and Nagengast, B. (2017). Modeling Omitted and Not-Reached Items in IRT Models. *Psychometrika*, 82(3):795–819.
- Sahin, F. and Colvin, K. F. (2020). Enhancing response time thresholds with response behaviors for detecting disengaged examinees. *Large-scale Assessments in Education*, 8(1):5.
- Scalise, K. and Allen, D. D. (2015). Use of open-source software for adaptive measurement: Concerto as an R-based computer adaptive development and delivery platform. *British Journal of Mathematical and Statistical Psychology*, 68(3):478–496.
- Scalise, K. and Gifford, B. (2006). Computer-based assessment in E-learning: A framework for constructing “intermediate constraint” questions and tasks for technology platforms. *The Journal of Technology, Learning and Assessment*, 4(6).
- Schmiedek, F., Kroehne, U., Goldhammer, F., Prindle, J. J., Lindenberger, U., Klinger-König, J., Grabe, H. J., Riedel-Heller, S. G., Pabst, A., Streit, F., Zillich, L., Kleineidam, L., Wagner, M., Rietschel, M., Rujescu, D., Schmidt, B., Investigators, N., and Berger, K. (2022). General cognitive ability assessment in the German National Cohort (NAKO) – The block-adaptive number series task. *The World Journal of Biological Psychiatry*, pages 1–12.
- Schnipke, D. L. and Scrams, D. J. (1997). Modeling Item Response Times With a Two-State Mixture Model: A New Method of Measuring Speededness. *Journal of Educational Measurement*, 34(3):213–232.
- Schnitzler, M., Baumann, R., Barkow, I., and Rölke, H. (2013). Chapter 5: Development of the cognitive items.
- Shin, H. J., Jewsbury, P. A., and van Rijn, P. W. (2022). Generating group-level scores under response accuracy-time conditional dependence. *Large-scale Assessments in Education*, 10(1):4.
- Shute, V. J. (2008). Focus on Formative Feedback. *Review of Educational Research*, 78(1):153–189.
- Shute, V. J., Wang, L., Greiff, S., Zhao, W., and Moore, G. (2016). Measuring problem solving skills via stealth assessment in an engaging video game. *Computers in Human Behavior*, 63:106–117.
- Sideridis, G. and Alahmadi, M. (2022). Estimation of Person Ability under Rapid and Effortful Responding. *Journal of Intelligence*, 10(3):67.
- Sireci, S. G. and Zenisky, A. L. (2015). Innovative item formats in computer-based testing: In pursuit of improved construct representation. In *Handbook of Test Development*, pages 313–334. Routledge.

- Slepkov, A. D. and Godfrey, A. T. K. (2019). Partial Credit in Answer-Until-Correct Multiple-Choice Tests Deployed in a Classroom Setting. *Applied Measurement in Education*, 32(2):138–150.
- Soland, J., Kuhfeld, M., and Rios, J. (2021). Comparing different response time threshold setting methods to detect low effort on a large-scale assessment. *Large-scale Assessments in Education*, 9(1):8.
- Stemmann, J. (2016). *Technische Problemlösekompetenz Im Alltag - Theoretische Entwicklung Und Empirische Prüfung Des Kompetenzkonstruktes Problemlösen Im Umgang Mit Technischen Geräten*. PhD thesis.
- Stodden, V., Leisch, F., and Peng, R. D., editors (2014). *Implementing Reproducible Research*. The R Series. CRC Press, Taylor & Francis Group, Boca Raton.
- Striewe, M. and Kramer, M. (2018). Empirische untersuchungen von lückentext-items zur beherrschung der syntax einer programmiersprache. *Commentarii informaticae didacticae*, (12):101–115.
- Tang, X., Zhang, S., Wang, Z., Liu, J., and Ying, Z. (2021). ProcData: An R Package for Process Data Analysis. *Psychometrika*, 86(4):1058–1083.
- Tomasik, M. J., Berger, S., and Moser, U. (2018). On the Development of a Computer-Based Tool for Formative Student Assessment: Epistemological, Methodological, and Practical Issues. *Frontiers in Psychology*, 9:2245.
- Toplak, M. E., West, R. F., and Stanovich, K. E. (2014). Assessing miserly information processing: An expansion of the Cognitive Reflection Test. *Thinking & Reasoning*, 20(2):147–168.
- Tóth, K., Rölke, H., Greiff, S., and Wüstenberg, S. (2014). Discovering Students' Complex Problem Solving Strategies in Educational Assessment. In *Proceedings of the 7th International Conference on Educational Data Mining*. (Pp. 225-228). International Educational Data Mining Society.
- Tsitoara, M. (2020). *Beginning Git and GitHub: A Comprehensive Guide to Version Control, Project Management, and Teamwork for the New Developer*. Apress, Berkeley, CA.
- Ulitzs, E., He, Q., and Pohl, S. (2022). Using Sequence Mining Techniques for Understanding Incorrect Behavioral Patterns on Interactive Tasks. *Journal of Educational and Behavioral Statistics*, 47(1):3–35.
- Ulitzs, E., Penk, C., von Davier, M., and Pohl, S. (2021a). Model meets reality: Validating a new behavioral measure for test-taking effort. *Educational Assessment*, 26(2):104–124.
- Ulitzs, E., Pohl, S., Khorramdel, L., Kroehne, U., and von Davier, M. (2021b). A Response-Time-Based Latent Response Mixture Model for Identifying and Modeling Careless and Insufficient Effort Responding in Survey Data. *Psychometrika*.

- Ulitzsch, E., von Davier, M., and Pohl, S. (2020). A Multiprocess Item Response Model for Not-Reached Items due to Time Limits and Quitting. *Educational and Psychological Measurement*, 80(3):522–547.
- van der Kleij, F. M., Eggen, T. J., Timmers, C. F., and Veldkamp, B. P. (2012). Effects of feedback in a computer-based assessment for learning. *Computers & Education*, 58(1):263–272.
- van der Linden, W. J. (2007). A Hierarchical Framework for Modeling Speed and Accuracy on Test Items. *Psychometrika*, 72(3):287–308.
- van der Linden, W. J. and Glas, C. A. (2000). *Computerized Adaptive Testing: Theory and Practice*. Springer.
- van der Linden, W. J., Klein Entink, R. H., and Fox, J.-P. (2010). IRT Parameter Estimation With Response Times as Collateral Information. *Applied Psychological Measurement*, 34(5):327–347.
- van der Linen, W. J. (2006). Model-Based Innovations in Computer-Based Testing. In Bartram, D. and Hambleton, R. K., editors, *Computer-Based Testing and the Internet: Issues and Advances*, pages 39–58. Wiley, Chichester.
- Veldkamp, B. P. and Sluijter, C., editors (2019). *Theoretical and Practical Advances in Computer-based Educational Measurement*. Methodology of Educational Measurement and Assessment. Springer International Publishing, Cham.
- von Davier, M. (2018). Automated Item Generation with Recurrent Neural Networks. *Psychometrika*, 83(4):847–857.
- Weiss, D. J. (1982). Improving Measurement Quality and Efficiency with Adaptive Testing. *Applied Psychological Measurement*, 6(4):473–492.
- William, D. (2011). What is assessment for learning? *Studies in Educational Evaluation*, 37(1):3–14.
- Williamson, D. M., , Robert J, M., and Bejar, I. I. (2006). *Automated Scoring of Complex Tasks in Computer-Based Testing*. Lawrence Erlbaum Associates, Mahwah, N.J.
- Wise, S. L. (2017). Rapid-Guessing Behavior: Its Identification, Interpretation, and Implications. *Educational Measurement: Issues and Practice*, 36(4):52–61.
- Wise, S. L. (2019). An Information-Based Approach to Identifying Rapid-Guessing Thresholds. *Applied Measurement in Education*, 32(4):325–336.
- Wise, S. L. and DeMars, C. E. (2006). An Application of Item Response Time: The Effort-Moderated IRT Model. *Journal of Educational Measurement*, 43(1):19–38.
- Wise, S. L., Kuhfeld, M. R., and Soland, J. (2019). The Effects of Effort Monitoring With Proctor Notification on Test-Taking Engagement, Test Performance, and Validity. *Applied Measurement in Education*, 32(2):183–192.

- Wools, S., Molenaar, M., and Hopster-den Otter, D. (2019). The Validity of Technology Enhanced Assessments—Threats and Opportunities. In Veldkamp, B. P. and Sluijter, C., editors, *Theoretical and Practical Advances in Computer-based Educational Measurement*, pages 3–19. Springer International Publishing, Cham.
- Xie, Y. (2015). *Dynamic Documents with R and Knitr*. CRC Press/Taylor & Francis, Boca Raton, second edition edition.
- Yan, D., Rupp, A. A., and Foltz, P. W., editors (2020). *Handbook of Automated Scoring; Theory into Practice*. CRC Press/Taylor & Francis Group.
- Yousfi, S. and Böhme, H. F. (2012). Principles and procedures of considering item sequence effects in the development of calibrated item pools: Conceptual analysis and empirical illustration. *Psychol. Test Assess. Model*, 54:366–396.
- Zehner, F., Goldhammer, F., Lubaway, E., and Sälzer, C. (2018). Unattended consequences: How text responses alter alongside PISA's mode change from 2012 to 2015. *Education Inquiry*, pages 1–22.
- Zehner, F., Sälzer, C., and Goldhammer, F. (2016). Automatic coding of short text responses via clustering in educational assessment. *Educational and Psychological Measurement*, 76(2):280–303.
- Zheng, Y. and Chang, H.-H. (2015). On-the-Fly Assembled Multistage Adaptive Testing. *Applied Psychological Measurement*, 39(2):104–118.
- Zumbo, B. D. and Hubley, A. M., editors (2017). *Understanding and Investigating Response Processes in Validation Research*, volume 69 of *Social Indicators Research Series*. Springer International Publishing, Cham.



Index

- Aspect Ratio, [50](#)
- Audio
 - Alternate Audio, [211](#)
 - Automatic Start, [12](#), [210](#)
 - Component, [207](#)
 - Controls, [209](#)
 - Internal vs. External Media, [208](#)
 - Link Audio to Components, [208](#)
 - Max Play, [209](#)
 - Media Raised Events, [210](#)
 - Volume, [210](#)
- Auto-Layout Panel, [143](#)
 - Grid Area, [143](#)
 - Layout Type=ABSOLUTE, [140](#)
 - Layout Type=GRID, [143](#)
- BACK_TASK (*see* Runtime Command)
- Button, [216](#)
 - Attach ScaleValueInput, [253](#)
 - Close Dialog
 - (*see also* Dialog Pages)
 - Component, [216](#)
 - Deselect Event, [220](#)
 - Image Button, [217](#)
 - Select Event, [220](#)
 - Set Command, [224](#)
 - Standard Button, [217](#)
 - Text Color, [220](#)
 - Toggle Buttons, [217](#)
- Calculator
 - Embedded Calculator using ExternalPageFrame, [393](#)
 - Internal Calculator using Finite-State Machine, [391](#)
- CANCEL_FULLSCREEN (*see* Runtime Command)
- CANCEL_TASK (*see* Runtime Command)
- CBA ItemBuilder
 - Context Menu, [94](#), [96](#)
 - Installation, [3](#)
 - Main Menu, [94](#)
 - Main Window, [6](#)
 - Manual Installation, [4](#)
 - Multiple Instances, [4](#), [415](#)
 - Support, [5](#)
 - System Requirements, [4](#)
 - Toolbar, [94](#)
 - Version, [6](#)
- CBA Presentation Size, [108](#)
 - Existing Projects, [109](#)
 - Global Properties, [375](#)
 - New Projects, [108](#)
 - Proportional Scaling, [109](#), [327](#)
 - xPage Size, [153](#)
- Checkbox, [185](#), [186](#)
 - Frame Select Group, [140](#), [186](#)
- Clipboard
 - Clipboard Viewer, [161](#)
 - Copy Nested Components, [161](#)
 - Delete Components, [162](#)
 - Duplicate Components, [161](#)
 - Duplicate Multiple Components, [161](#)
- Clipboard (*see* Runtime Command)
- CLOSE (*see* Runtime Command)
- Close Dialog (*see* Runtime Commands)
- CLOSE_AND_NEXT_TASK (*see* Runtime Command)
- ComboBox, [188](#)
 - ComboBox Item, [188](#)
 - Component, [188](#)
- ComboBox Item (*see* ComboBox)

- Command (*see* Runtime Command)
- Component Edit View, [96](#), [99](#), [188](#)
 - Edit ComboBox and List Items, [188](#)
- Components
 - Components to Collect Responses, [178](#)
 - Positioning, [158](#)
- Conditional Links, [264](#)
 - closeDialog()-Operator, [307](#)
 - Conditional Links and Dialog Pages, [266](#)
 - initFSM()-Operator, [313](#)
 - Links versus Conditional Links, [212](#)
 - matches()-Operator, [268](#)
 - openDialog()-Operator, [307](#)
 - Raise Finite-State Machine Events, [313](#)
 - reset()-Operator, [297](#)
 - set()-Operator, [297](#)
 - setEmbeddedPage()-Operator, [278](#), [312](#)
- Configure Select Group (*see* Frame Select Group)
- COPY (*see* Runtime Command)
- CUT (*see* Runtime Command)
- Dialog Pages, [238](#)
 - Closable, [226](#)
 - Closable versus Not-Closable Dialogs, [240](#)
- Dialogs
 - Closable, [241](#)
 - Modal vs. Non-Modal, [241](#)
- Domain Specific Language (DSL)
 - Scoring, [346](#)
- Drag and Drop, [258](#), [306](#)
- Drawing Area, [98](#)
- Embedded HTML Explorer, [96](#), [235](#)
- Embedded Preview (*see* Rendering View)
- External Page Frame, [234](#)
 - Caution Note, [236](#)
 - Component, [234](#)
 - External, [234](#)
 - Local, [234](#)
 - Scoring using Variables, [102](#)
 - Set Page Address, [234](#)
- ExternalPageFrame
 - postMessage, [331](#)
- Finite-State Machine
 - Basic Concept, [279](#)
 - Change Events, [313](#)
 - Combination of Conditions, [294](#)
 - Current Page in Conditions, [296](#)
 - Current State, [244](#), [253](#), [282](#)
 - Current Task in Conditions, [295](#)
 - Definition of Events, [284](#)
 - Elapsed Time in Conditions, [296](#)
 - Guards (Conditions in Rules), [292](#)
 - Internal Processing of Events, [317](#)
 - Mathematical Expressions in Conditions, [296](#)
 - Multiple Pages Linked to States, [322](#)
 - Raise Finite-State Machine Events, [313](#)
 - Regions, [318](#), [321](#)
 - Self-Transitions, [317](#)
 - State Entry and State Exit, [316](#)
 - State Machine Debug Window, [244](#)
 - State Type, [282](#)
 - Text Input in Conditions, [296](#)
 - Trigger Runtime Commands, [224](#)
- Frame, [136](#)
 - Initialization for New Pages, [132](#)
- Frame Select Group, [140](#), [186](#)
- Fullscreen, [49](#), [227](#)
 - (*see also* Runtime Command)
- Global Properties
 - CBA Presentation Size, [109](#), [375](#)
 - Link Color, [375](#)
- Guards (*see* Finite-State Machine)
- HTMLTextField
 - Clipboard, [175](#)
 - Component, [174](#)
 - HTML Text Editor, [174](#)
 - Links, [175](#)

Images

- Add to Project File, [202](#)
- Link Images to Components, [207](#)
- Supported File Formats, [202](#)

Input Validation, [368](#)

- Decimal Numbers, [370](#)
- Example, [368](#)
- Feedback, [370](#)
- Integer Numbers, [368](#)
- Letters, [369](#)

InputField

- Input Validation Pattern, [368](#)

Item Size, [50](#)

- (see also CBA Presentation Size)

Links, [212](#)

- Embedded Links, [220](#)
- Link Color, [375](#)
- Link Pages, [212](#)
- Link-Component, [214](#)
- Links vs. Commands, [214](#)
- Pages and xPages, [221](#)
- Pages of Different Type, [222](#)
- Visited Link Color, [216](#), [375](#)

List, [188](#)

- Component, [188](#)
- List Item, [188](#)

ListItem (see List)

MapBasedVariableDisplay, [257](#)

MenuItems

- Set Command, [224](#)

Navigation, [49](#)Nesting, [92](#)

NEXT_TASK (see Runtime Command)

Operators

- back_task()-Operator, [306](#)
- calcGetMem()-Operator, [310](#), [520](#)
- calcOp()-Operator, [310](#), [520](#)
- calcOpnd()-Operator, [310](#), [520](#)
- calcSettings()-Operator, [310](#), [520](#)
- callExternalPageFrame()-Operator, [334](#)
- cancel_task()-Operator, [306](#)

closeDialog()-Operator, [242](#), [307](#)elapsedTime()-Operator, [315](#)focus()-Operator, [300](#)ifthenelse()-Operator, [296](#)initMediaPlayer()-Operator (Task Initialization), [310](#)matches()-Operator, [296](#), [365](#)next_task()-Operator, [306](#)openDialog()-Operator, [241](#), [307](#)raise()-Operator, [313](#)raised_all_events()-Operator, [353](#)raised_all_events_in_state()-Operator, [353](#)raised_events()-Operator, [352](#)raised_nb_events()-Operator, [353](#)raised_nb_events_in_state()-Operator, [353](#)reset()-Operator, [253](#), [297](#)scrollEmbeddedPage()-Operator, [307](#)scrollTopLevelPage()-Operator, [307](#)set()-Operator, [253](#), [297](#)setActive()-Operator, [302](#)setEmbeddedPage()-Operator, [278](#), [312](#)setFrozen()-Operator, [298](#)setFSMEvent()-Operator, [313](#)setFSMState()-Operator, [314](#)setGlobalProperty()-Operator, [303](#)setHidden()-Operator, [300](#)setHighlightable()-Operator, [303](#)setInputValue()-Operator, [304](#)setMediaPlayer()-Operator, [309](#)setValueDisplayMode()-Operator, [306](#)trace_snapshot()/trace_typed_snapshot()-Operator, [315](#)trace_text()/trace_typed_text()-Operator, [314](#)tree_copy()-Operator, [315](#)tree_move()-Operator, [315](#)unsetActive()-Operator, [302](#)unsetFrozen()-Operator, [298](#)unsetHidden()-Operator, [300](#)unsetHighlightable()-Operator, [303](#)

- visited_all_states()-Operator, [354](#)
- visited_nb_states()-Operator, [353](#)
- Page
 - New Page, [131](#)
 - Page Name, [135](#)
 - Page Settings, [135](#)
 - Page Type, [131](#)
 - Simple Page, [131](#)
 - Tabfoler Page, [230](#)
 - Taskbar Page, [231](#)
 - Preview, [10](#)
 - Simple Page, [133](#)
 - standardPage Tag, [135](#)
 - Tagging, [133](#)
 - xPage, [133](#)
 - xPage Tag, [135](#)
- Page Editor, [97](#)
 - Clipboard, [161](#)
 - Page Editor
 - Drawing Area (*see* Drawing Area)
 - Duplicate Components, [161](#)
 - Page Editor
 - Palette (*see* Palette)
 - Rulers & Grid, [101](#)
 - Visual Editing, [158](#)
 - Zoom, [99](#)
- PageAreas, [146](#)
 - Embedded Page, [146](#)
 - Link Embedded Page, [147](#)
 - Scrollbars, [146](#)
 - User Defined IDs, [148](#)
- Palette, [97](#)
- Panel, [140](#)
 - Container, [140](#)
 - Initialization for New Pages, [132](#)
 - Nesting, [142](#)
- PASTE (*see* Runtime Command)
- Popups Window (*see* Dialog Pages)
- Preferences
 - CBA Presentation Size, [108](#)
 - CBA Preview Browser, [12](#)
- Preview, [7](#)
 - Automatic Start, [12](#)
 - Preview Dialog, [10](#)
 - Terminology, [7](#)
- Project File, [8](#)
 - Downloaded Files, [105](#)
 - Inconsistencies, [106](#)
 - Migration, [6](#), [108](#)
 - Preview, [10](#)
 - Project Name, [9](#), [105](#)
 - Rename (Save as...), [105](#)
 - Save, [106](#)
 - Terminology, [9](#)
- Project Names
 - Project Name vs. File Name, [9](#)
 - Valid Names, [105](#)
- Project View, [95](#)
- Proportional Scaling, [50](#)
- RadioButton, [181](#), [186](#)
 - Frame Select Group, [140](#)
 - RadioButtonGroup, [181](#)
- Regular Expression, [364](#)
 - Decimal Number, [367](#)
 - Empty Response, [366](#)
 - Input Validation, [368](#)
 - matches()-Operator, [365](#)
 - Range, [367](#)
 - User Defined Id, [364](#)
- Rendering View, [97](#)
- Resolution, [50](#)
- Resource Browser, [99](#)
- Response Mode
 - Forced Choice, [51](#)
- Runtime Command
 - BACK_TASK, [225](#)
 - CANCEL_FULLSCREEN, [227](#)
 - CANCEL_TASK, [225](#)
 - Clipboard, [227](#)
 - CLOSE, [226](#), [241](#)
 - CLOSE_AND_NEXT_TASK, [226](#), [241](#)
 - COPY, [227](#)
 - CUT, [227](#)
 - NEXT_TASK, [225](#)
 - PASTE, [227](#)
 - TRIGGER_FULLSCREEN, [227](#)
- Scaling Options, [11](#), [110](#)

- Scoring
 - Use of Regular Expressions, [365](#)
 - variable_in()-Operator, [349](#)
 - visited_all_values_of_variable()-Operator, [350](#)
- Screen Orientation, [50](#)
- Scrolling, [50](#), [146](#)
- Select Group (*see* Frame Select Group)
- Set Page Address (*see* External Page Frame)
- SimpleTextField
 - Component, [171](#)
 - Input Source, [171](#)
- SingleLineInput Field
 - Input Validation Pattern, [368](#)
- State
 - Scoring, [354](#)
- States
 - Assign Page, [241](#), [314](#), [320](#)
 - Current State, [282](#)
 - State Types (NORMAL, START and END), [282](#)
- Syntax
 - Auto-Completion, [246](#)
 - Bracketing, [248](#), [294](#)
 - Case Sensitive, [247](#)
 - Comments, [247](#)
 - Conditional Links Syntax, [245](#)
 - Errors, [247](#)
 - Finite-State Machine Syntax, [245](#)
 - Operators, [246](#)
 - Scoring Syntax, [245](#)
 - Task Initialization, [151](#)
 - Task Initialization Syntax, [245](#)
 - User Defined Id, [246](#)
- Tab-Order, [170](#)
- Task, [149](#)
 - Definition (Browse Task and ItemScore), [150](#)
 - Entry Point, [149](#)
 - First Page, [150](#)
 - Layout Settings, [151](#)
 - MinHits, [150](#)
 - Preview, [10](#), [151](#)
 - Task Initialization, [151](#)
 - Task Name, [150](#)
 - Tasks View, [150](#)
 - xPage Layout, [152](#)
- Task Editor, [102](#)
- Task Management
 - back_task()-Operator, [306](#)
 - cancel_task()-Operator, [306](#)
 - isCurrentTask()-Operator, [287](#)
 - next_task()-Operator, [306](#)
- Text
 - Components to Collect Text Responses, [178](#)
 - Components to Display Text, [176](#)
- TextField
 - Component, [175](#)
 - Highlighting, [175](#)
 - Mathematical Formulas, [176](#)
- Timer
 - Link Timer Event, [324](#)
 - Timer Component, [324](#)
- Toggle Buttons (*see* Button)
- Transparent Background, [166](#)
- TRIGGER_FULLSCREEN (*see* Runtime Command)
- Undo/Redo, [26](#), [157](#)
- User Defined Id, [101](#)
 - Edit all User defined IDs, [165](#)
 - Use in Syntax, [246](#)
 - Valid User Defined Ids, [364](#)
- Value Maps, [103](#), [255](#)
 - Editor (Browse Value Maps), [255](#)
 - Guards, [103](#)
 - Link Value Map to Components, [257](#)
 - Value Maps Editor, [103](#)
- Variables, [102](#), [251](#)
 - Editor (Browse Variables), [251](#)
 - Link Variable to Components, [253](#)
 - Named Variable Values, [251](#)
 - Set Variable attributes, [251](#)
- Types
 - BOOLEAN, [251](#)

- INTEGER, [251](#)
- NUMBER, [251](#)
- STRING, [251](#)
- Variables Editor, [102](#)
- VariableValueDisplay, [255](#)
- Video
 - Alternate Video, [211](#)
 - Automatic Start, [12](#), [210](#)
 - Component, [207](#)
 - Controls, [209](#)
 - Internal vs. External Media, [208](#)
 - Link Video to Components, [208](#)
 - Max Play, [209](#)
 - Media Raised Events, [210](#)
 - Volume, [210](#)
- xPage
 - (*see also* Pages)
- xPage Layout (*see* Task)
- Z-Order, [92](#), [410](#)
- Zoom
 - See also* Page Editor