Axiell Go Form Design Guide

Axiell Go version 1.0.x

Document Version 1.1

June 2019

Form design guide

Forms are used throughout Axiell Go to list records following a search, to display data and to enable users to create new records and update existing records in the collection management system. Building forms is a fairly technical undertaking at the moment. A user-friendly Form creation utility utilising drag and drop will be available in a future release of Axiell Go, and in the meantime you should discuss your Form design requirements with Axiell Support.

This design guide describes the building blocks of Axiell Go forms. It is intended for Axiell staff.

Definition of a form

A form is a JSON object with the following top-level properties:

Property	Details
`"format"`	A numeric value specifying a JSON object with a particular structure. Currently, the only `"format"` supported is `1`. This guide describes `"format" 1` JSON forms.
`"id"`	A unique identifier for the form. It is referenced when manipulating an existing form and creating form instances. A form's identifier typically combines the module name with its form type: » A form is built for a particular module (called an entity). -AND- » As we see below, there are two types of form, `"list"`and `"detail"`, one for listing records, the other for viewing record details. For example, a form designed for viewing record details in the Parties module will have an `"id"` of: `"id": "epartiesDetail",`
`"entityId"`	The entity for which the form has been designed. This is the name of a module, e.g.: `"entityId": "eparties",`
`"entityType"`	The type of entity. For the purposes of this document this will always be `table` (a module is a database table): `"entityType": "table",`

`"formType"`	The type of form. For module entities, this is either `"list"` or `"detail"`:
	» A `"list"` form displays a set of results (typically following a search) in a table:
	`"formType": "list",`
	» A `"detail"` form displays the details of a single record:
	`"formType": "detail",`
	Creating new records and updating existing records is done with `"detail"` forms.
	`"list"` and `"detail"` forms are described below (page 6).

Property	Details
`"formLabel"`	A descriptive label for the form. This property is generated automatically and therefore does not need to be specified. It is the same as `"id"` unless there is a translation found in the `"translate"` property (described below): `"formLabel": "epartiesDetail",`
`"form"`	A JSON object containing an ordered list of elements (fields, containers, panels, sections, etc.) which together determine what displays on the form (its layout). This is described in **General structure of the `"form"` object** (page 4).
`"translate"`	Contains multilingual translations of text used in a form (including more user friendly / descriptive labels in the current language). Each element to be translated is listed and translations in multiple language can be provided. For example, the elements `"firstName"` and `"lastName"`, translated into English and Spanish would be listed as: `"translate": { "firstName": { "en": "First Name", "es": "El Nombre de Pila" }, "lastName": { "en": "Last Name", "es": "El Apellido" } }` Where the current language is English, the label for the first field will show as `First Name`; where the language is Spanish, the field label will be `El Nombre de Pila`. Values in `translate` are only used for a form element when that element has the property `"label": null`. If a `"label"` property has been specified for an element or the `"label"` property has not been specified (it is absent), `"translate"` values are not used for that element. Elements do not need to be ordered in any particular way in `"translate"` as every object is a direct child of `"translate"`. **Note:** See `"hint"` and `"placeholder"` in **List of properties** (page 43) for details about translating these two properties.

The top-level properties of an `epartiesDetail` form would appear similar to:

```
{
   "format": 1,
   "id": "epartiesDetail",
   "formType": "detail",
   "formLabel": "epartiesDetail",
   "entityType": "table",
   "entityId": "eparties",
   "form": { … },
   "translate": { … }
}
```

The `"form"` object

A form is experienced by users as a number of structured elements, such as fields, containers, panels, sections, etc. These are specified as an ordered list of JSON objects within the top-level `"form"` property. Each element must have a unique reference.

There are two categories of element:

- Simple, such as a text field or radio buttons.
- Container, which contains a structured list of child elements (simple or other containers) within an object called `"elements"`.

As JSON objects in their own right, each element has a number of top-level properties. All elements contain these top-level properties:

Property	Details
`"elem"`	Defines a type of element: » Simple elements include text fields, radio buttons, etc. Each simple element has a specific `"elem"` type. For example, a text field is specified as `"elem": "text"`. Simple elements are described below (page 25). » Container elements contain a structured list of child elements (simple or other containers) within an object called `"elements"`. For example, a section container is specified as `"elem": "section"`. Container elements are described below (page 19).

`"properties"`	Lists attributes of the element. A list of properties and their definitions is given below (page 43).
	Which attributes are specified depends on the element's type:
	» Container elements may only include the `"label"` property.
	» Simple elements typically have more attributes than container elements. These specify the way the element appears, as well as the back-end column the element references. Examples include:
	» `"name"`
	» `"label"`
	» `"type"`
	» `"display"`
	For example, a text field `"firstName"` in the module eparties that references the `"NamFirst"` column is represented as:
	`"firstName": {`
	`    "elem": "text",`
	`    "properties": {`
	`        "name": "NamFirst",`
	`        "label": null`

Property	Details

	`        }`
	 `}`
	 Note: Each element must have a unique reference. In this example the text field element is referenced as `"firstName"`; this reference cannot be used again in this form.

Container elements also include these top-level properties:

Property	Details
`"elements"`	An object containing a list of the container's child elements. These may be simple elements or more container elements, and are objects with their own properties.
`"traversal"`	A list of the container's child elements in the order in which they will display. If there is no `"traversal"` property, the elements will display but not automatically in the order in which they are listed within the `"elements"` object.

As we see below, these elements are arranged differently for `"list"` and `"detail"` forms (page 6).

`"list"` and `"detail"` forms

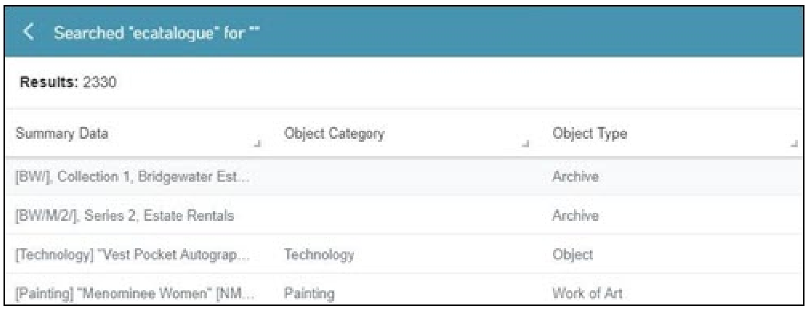
As we saw above (page 1), forms with an `"entityType"` of `"table"`, have a `"formType"` of `"list"` or `"detail"`. In order for users to be able to search, display, edit and insert records in a module, a module requires both types of form:

`"list"`

When a search has been run, a `"list"` form is used to display the search results. Two different views are possible with `"list"` forms, and users can switch between the two views:

- Grid view

  By default, search results are presented in Grid View:



Searched "ecatalogue" for ""

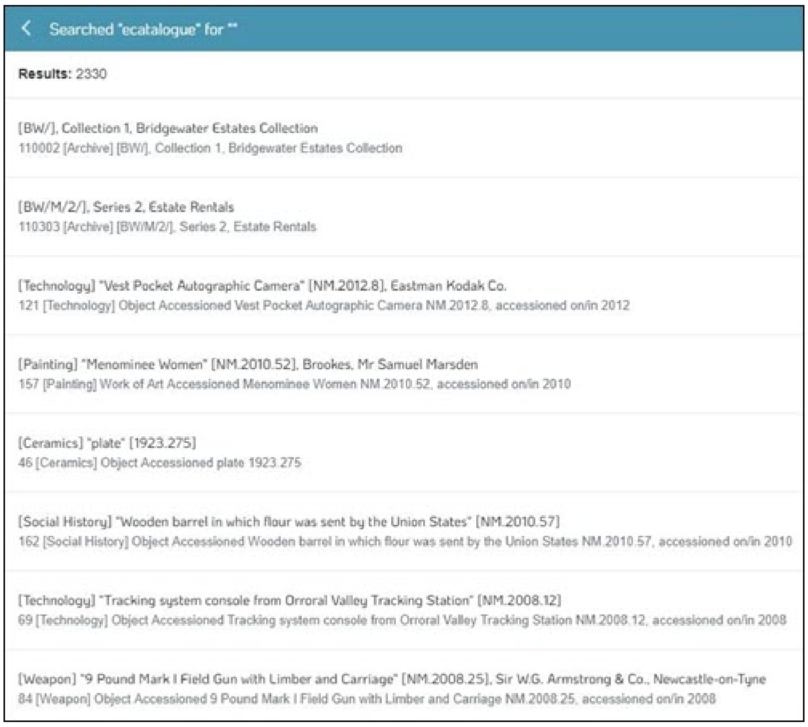
Results: 2330

Summary Data	Object Category	Object Type
[BW/], Collection 1, Bridgewater Est...		Archive
[BW/M/2/], Series 2, Estate Rentals		Archive
[Technology] "Vest Pocket Autograp...	Technology	Object
[Painting] "Menominee Women" [NM...	Painting	Work of Art

  In Grid View, record data is listed in columns. Results can be sorted (lowest to highest; A to Z and vice versa) by selecting a column header.

- List View

  In List View, records are listed in rows with a subset of data displaying from each record:



Searched "ecatalogue" for ""

Results: 2330

[BW/], Collection 1, Bridgewater Estates Collection
110002 [Archive] [BW/], Collection 1, Bridgewater Estates Collection

[BW/M/2/], Series 2, Estate Rentals
110303 [Archive] [BW/M/2/], Series 2, Estate Rentals

[Technology] "Vest Pocket Autographic Camera" [NM.2012.8], Eastman Kodak Co.
121 [Technology] Object Accessioned Vest Pocket Autographic Camera NM.2012.8, accessioned on/in 2012

[Painting] "Menominee Women" [NM.2010.52], Brookes, Mr Samuel Marsden
157 [Painting] Work of Art Accessioned Menominee Women NM.2010.52, accessioned on/in 2010

[Ceramics] "plate" [1923.275]
46 [Ceramics] Object Accessioned plate 1923.275

[Social History] "Wooden barrel in which flour was sent by the Union States" [NM.2010.57]
162 [Social History] Object Accessioned Wooden barrel in which flour was sent by the Union States NM.2010.57, accessioned on/in 2010

[Technology] "Tracking system console from Orroral Valley Tracking Station" [NM.2008.12]
69 [Technology] Object Accessioned Tracking system console from Orroral Valley Tracking Station NM.2008.12, accessioned on/in 2008

[Weapon] "9 Pound Mark I Field Gun with Limber and Carriage" [NM.2008.25], Sir W.G. Armstrong & Co., Newcastle-on-Tyne
84 [Weapon] Object Accessioned 9 Pound Mark I Field Gun with Limber and Carriage NM.2008.25, accessioned on/in 2008

  The data displayed for each record is specified with the `"previewElements"` attribute (described below).

The arrangement of a `"list"` form is typically less complicated than a `"detail"` form:

- In the top-level `"form"` object is a single property with the same value as the form's `"id"`.

  For example, if the `"id"` of the form is `"epartiesList"`, we'd see:

  ```
  {
      "format": 1,
      "entityId": "eparties",
      "entityType": "table",
      "formLabel": "epartiesList",
      "formType": "list",
      "id": "epartiesList",
      "form": {
          "epartiesList": {
  ...
  ```

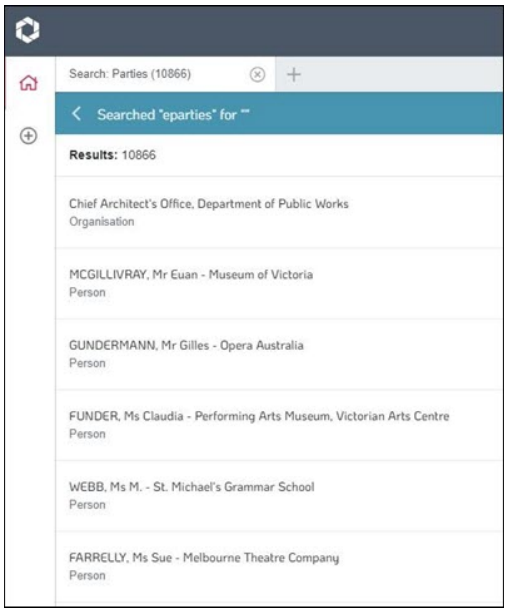
  The top-level of this property is a container element with an `"elem"` of `"rootForm"`:

  ```
  "form": {
          "epartiesList": {
              "elem": "rootForm",
  ...
  ```

- Within the `"rootForm"` container is a list of one or more simple elements (page 25) that will display as column headings in Grid View.

  > **Note:** See **Container elements** below (page 19) for details about `"rootForm"` properties. Within `"rootForm"`, simple elements are specified within the `"elements"` property.

- As we've seen, it is possible for users to switch from Grid View (in which record data is listed in columns) to List View, in which records are listed in rows with a subset of data. In this List View, each row is a record and data is pulled from Summary Data and Party Type:


Search: Parties (10866)

Searched "eparties" for ""

Results: 10866

Chief Architect's Office, Department of Public Works
Organisation

MCGILLIVRAY, Mr Euan - Museum of Victoria
Person

GUNDERMANN, Mr Gilles - Opera Australia
Person

FUNDER, Ms Claudia - Performing Arts Museum, Victorian Arts Centre
Person

WEBB, Ms M. - St. Michael's Grammar School
Person

FARRELLY, Ms Sue - Melbourne Theatre Company
Person

This is specified with the `"previewElements"` attribute within the `"properties"` object. The example above is specified as:

```
    "form": {
        "epartiesList": {
            "elem": "rootForm",
            "properties": {
                "previewElements": [
                    "summaryData",
                    "partyType"
                ]
        }
...
```

» As well as the usual `"elem"`, `"properties"`, `"traversal"` and `"elements"`, a `"list"` form has the additional property of `"relatedForms"`. This links the `"list"` form to a `"detail"` form counterpart in the same module.

> Note: If the module has more than one `"detail"` form, described in **Alternative `"detail"` forms** (page 13), this will be the name of the form to be displayed by default upon selection of a record in the `"list"` form.

For example, a `"list"` form for eparties would have the property:

```
"relatedForms": {
    "detail": "epartiesDetail"
}
```

Example structure

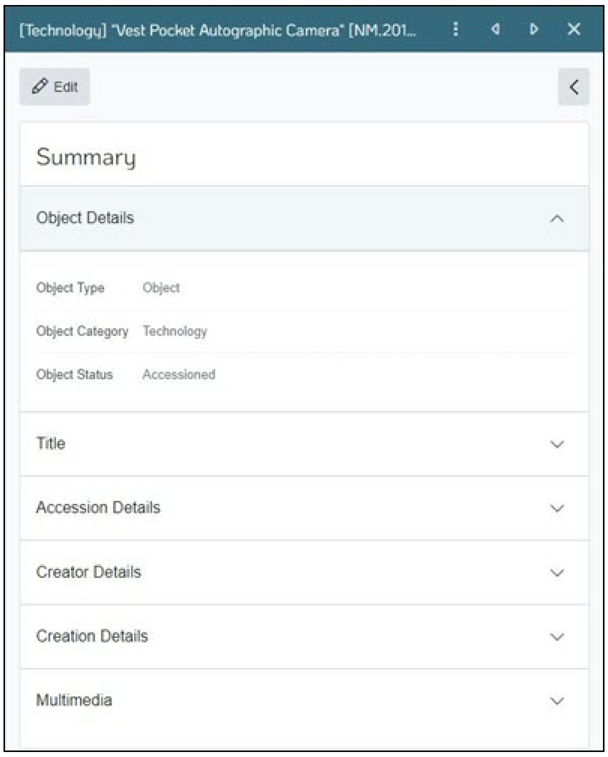
Example of the structure within the "`form`" object of a "`list`" form:

```
{
    "format": 1,
    "entityId": "eparties",
    "entityType": "table",
    "formLabel": "epartiesList",
    "formType": "list",
    "id": "epartiesList",
    "form": {
        "epartiesList": {
            "elem": "rootForm",
            "properties": {
                "label": null,
                "previewElements": [
                    "summaryData",
                    "partyType"
                ]
            },
            "traversal": [
                "summaryData",
                "partyType",
                "extendedData"
            ],
            "relatedForms": {
                "detail": "epartiesDetail"
            },
            "elements": {
                "summaryData": {
                    "elem": "text",
                    "properties": {
                        "label": null,
                        "name": "SummaryData"
                     }
                },
                "partyType": {
                    "elem": "text",
                    "properties": {
                        "label": null,
                        "name": "NamPartyType"
                    }
                },
                "extendedData": {
                    "elem": "text",
                    "properties": {
                        "label": null,
                        "name": "ExtendedData"
                        }
                }
            }
        }
    },
    "translate": {
        "summaryData": {
            "en": "Summary Data"
        },
        "extendedData": {
            "en": "Extended Data"
        },
        "partyType": {
            "en": "Party Type"
        }
    }
}
```

"detail"

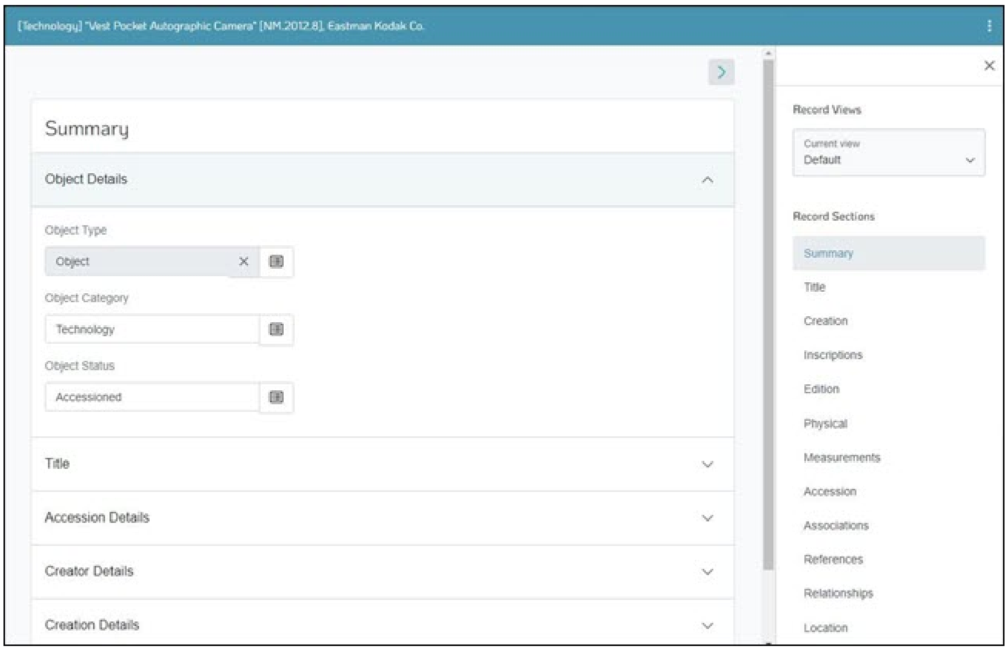
A "detail" form displays details of a single record and is accessed by either inserting a new record (using a Task for instance) or selecting a record in a "list" form:

» When a record is selected in a "list" form, a "detail" form displays to the right of the workspace with record data in read-only format:



Object Type	Object
Object Category	Technology
Object Status	Accessioned

» When a record is edited or a new record has been added, a "detail" form presents the record in an editable state:



Object Type: Object
Object Category: Technology
Object Status: Accessioned
Current view: Default

The arrangement of elements in a standard "detail" form is as follows:

- In the top-level "`form`" object is a single property with the same value as the form's "`id`".

  For example, if the "`id`" of "`form`" is "`epartiesDetails`", we'd see:

  ```
  {
      "format": 1,
      "id": "epartiesDetail",
      "formType": "detail",
      "formLabel": "epartiesDetail",
      "entityType": "table",
      "entityId": "eparties",
      "form": {
          "epartiesDetail": {
  ...
  ```

  This top-level element of this property is a container element with an "`elem`" of "`rootForm`":

  ```
  "form": {
          "epartiesDetail": {
              "elem": "rootForm",
  ...
  ```

- "`rootForm`" contains one or more "`section`"s.
- Each `section` contains one or more "`panel`"s.
- Each `panel` contains at least one element. This may be a simple element such as a text field, or a container element such as "`listGrid`", which itself contains simple elements.

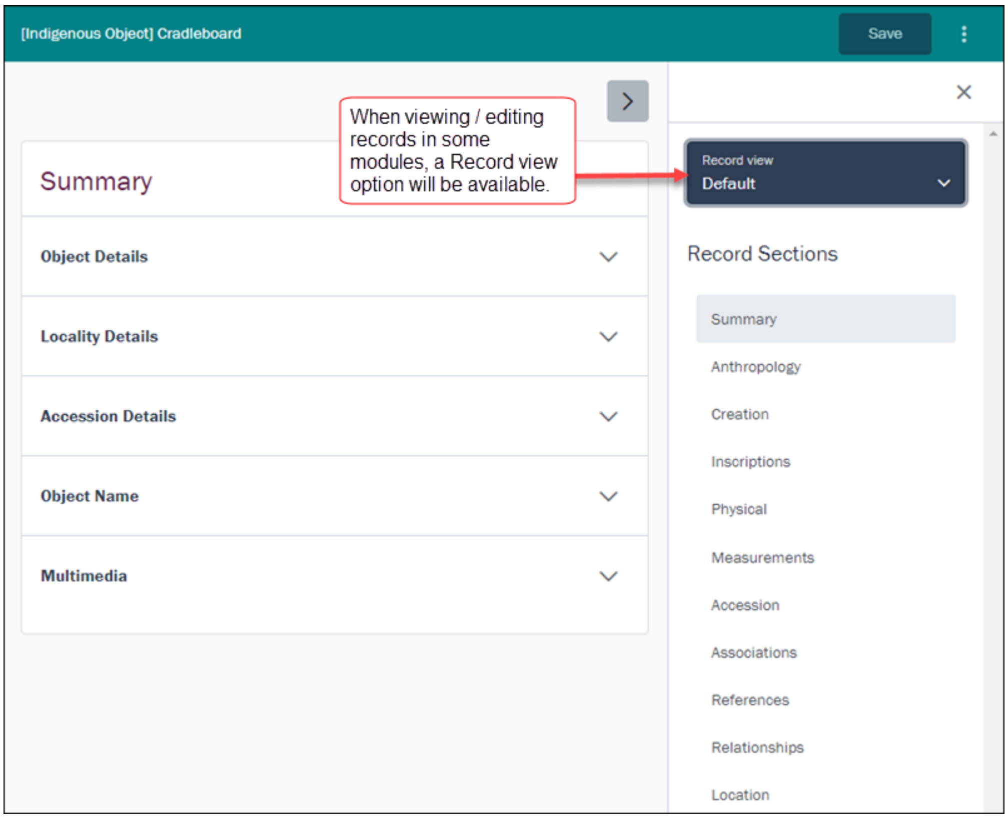
> Note: See **Container elements** below (page 19) for details about "`rootForm`", "`section`" and `panel` properties.

Record Views: alternative "`detail`" forms

With Record Views it is possible to have more than one "`detail`" form in a module, each view presenting a different set of fields in a record.

About Record Views

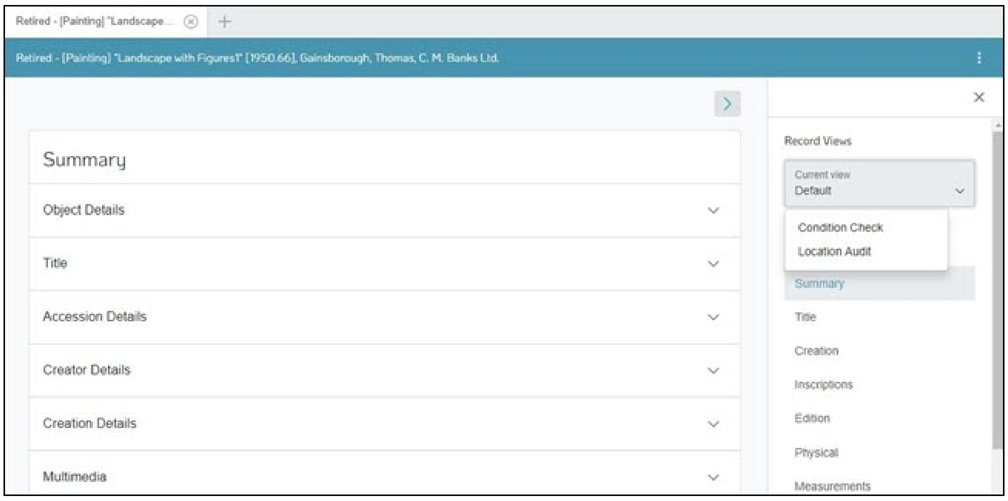
When viewing or editing records in some modules, the Record Sections pane will include a Record View drop list:



A Record View is a view of a record tailored for a particular purpose (or group of users). Typically, tailoring involves simplifying the information presented to a user by reducing the number of Record Sections and fields displayed.

By default, when viewing or editing a record's details, all Record Sections and fields that you are authorised to access are displayed. Depending on the module, there might be twenty or more Record Sections and many hundreds of fields. For many everyday tasks you do not need to see every field, only a subset, and a Record View can be configured to present only those fields required to perform the task.

An example of an alternative "`detail`" form is the Condition Check Record View available when viewing records in the Catalogue module. Here we see the Catalogue module's default "`detail`" form (`ecatalogueDetail`):



As we see, two other "`detail`" forms are available from the Catalogue module Record View drop list:

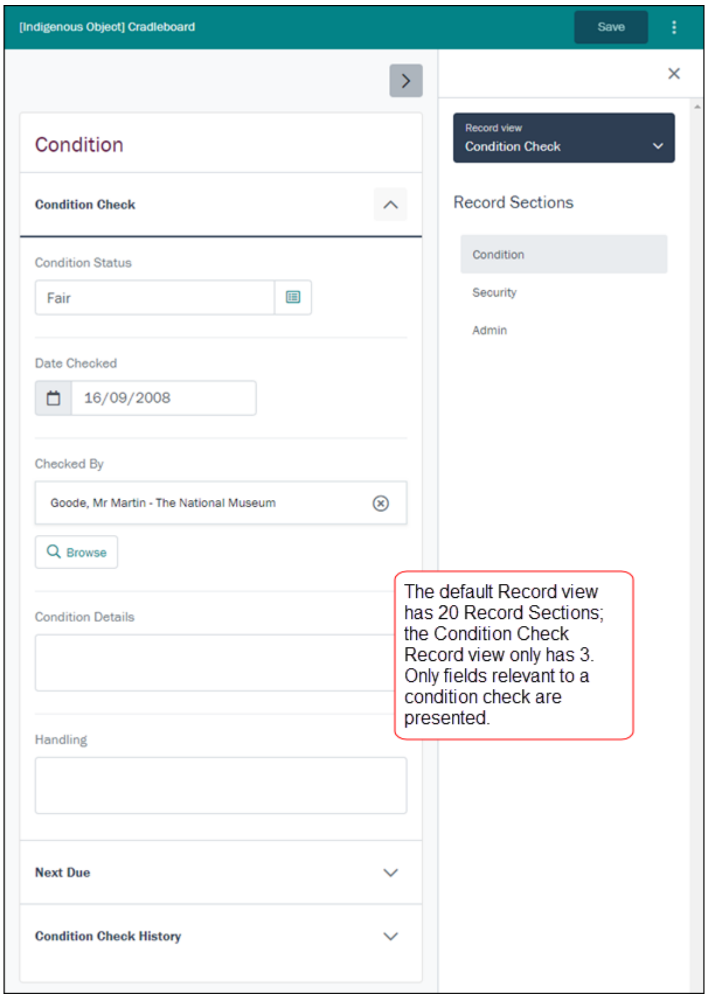
- Condition Check (`ecatalogueConditionCheck`)
- Location Audit (`ecatalogueLocationAudit`)

To be included in the Record View drop list for a module, the value for "`entityId`" in a "`detail`" form must match the back-end name for the module. In the example above, both forms state:

```
"entityId": "ecatalogue",
```

Note: See **The "`modes`" property** (page 16) for details about other conditions that determine the inclusion of a "`detail`" form in the Record View drop list.

Selecting the **Condition Check** Record View will present only those details (fields) relevant for condition checks:



The default Record view has 20 Record Sections; the Condition Check Record view only has 3. Only fields relevant to a condition check are presented.

When there is more than one "`detail`" form available for a module, it is necessary to specify which form should display by default when a record or task is selected:

- When using a Task to insert a record, the default form is specified in the `processes.pl` configuration file using the "`formDefinitionId`" property.
- When selecting a record in a "`list`" form, the default "`detail`" form is specified in the "`list`" form with the "`relatedForms`" property. Other form's will then be available from the Record View drop list (subject to the "`modes`" property - see below).

The "modes" property

The "modes" property is an additional top-level property for "detail" forms which sets the conditions in which a form is included in the Record View drop list.

The "modes" property has three possible values:

Property	Details
"read"	The form is used to display data. When selecting a record in a "list" form, for instance, a "detail" form with this value will be available from the Record View drop list. If a form only has "modes": ["read"], it will not display in the Record View drop list when inserting or editing a record.
"edit"	The form is used for editing records. When editing a record, a "detail" form with this value will be available from the Record View drop list and an Edit button will display when viewing the record.
"insert"	The form is used when inserting a new record.

The Condition Check form "ecatalogueConditionCheck" described above has the following structure:

```
{
    "format": 1,
    "formType": "detail",
    "id": "ecatalogueConditionCheck",
    "entityType": "table",
    "entityId": "ecatalogue",
    "modes": [
        "read",
        "edit"
    ],
    "form": { … },
    "translate": { … }
}
```

This "detail" form is available in both display and edit modes, but is not available when inserting a new Cataloguerecord.

Example structure

Example of the structure within the `"form"` object of a `detail` form, containing a section (`"section1"`), a panel (`"panel1"`), and two simple elements (`"first"` and `"last"`):

Note: Each element must have a unique reference (e.g. the text field *Party Type* might be referenced as `"partyType"`).

```
{
    "format": 1,
    "id": "epartiesDetail",
    "formType": "detail",
    "formLabel": "epartiesDetail",
    "entityType": "table",
    "entityId": "eparties",
    "form": {
        "eparties": {
            "elem": "rootForm",
            "properties": {
                "label": null
            },
            "traversal": [
                "section1"
            ],
            "elements": {
                "section1": {
                    "elem": "section",
                    "properties": {
                    "label": null
                },
                "traversal": [
                    "panel1"
                ],
                "elements": {
                    "panel1": {
                        "elem": "panel",
                        "properties": {
                            "label": null
                        },
                        "traversal": [
                            "first",
                            "last"
                        ],
                        "elements": {
                            "first": {
                                "elem": "text",
                                "properties": {
                                    "name": "NamFirst",
                                    "label": null
                                }
                            },
                            "last": {
                                "elem": "text",
                                "properties": {
                                    "name": "NamLast",
                                    "label": null
                                 }
                            }
                        }
                    }
                }
            }
        }
...
```

Description of element types

Container elements

"rootform"

The top-level element in "form". It groups all child elements and is not used anywhere else within the "form" definition.

A "rootForm" contains:

Property	Details
`"elem": "rootForm",`	
`"properties"`	Which properties are listed depends on the type of form: » A "detail" form typically just contains "label". » A "list" form will also include "previewElements". A list of "properties" and their definitions is available below (page 43).

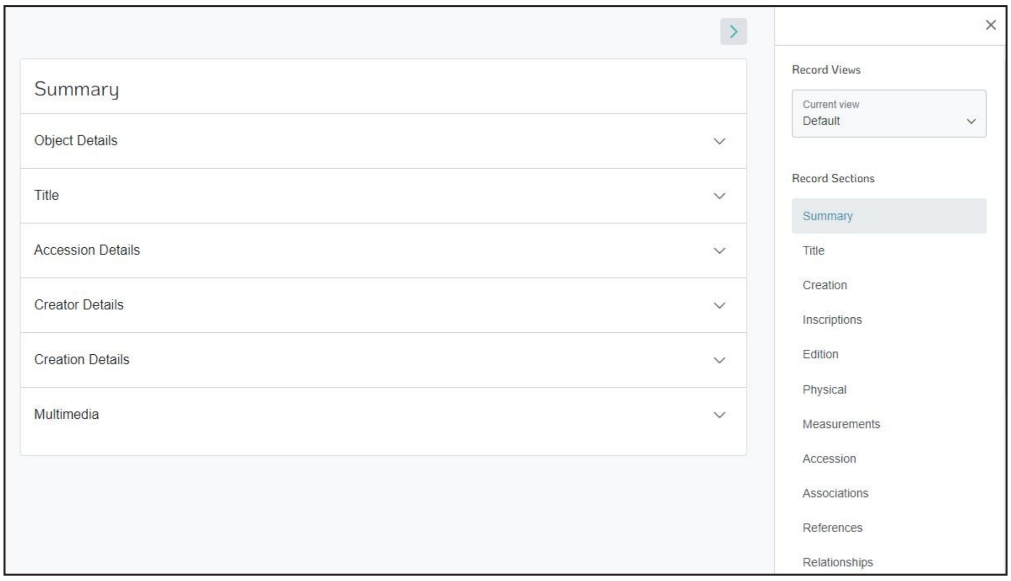
`"relatedForms"`	Used by "list" forms to link to its "detail" form counterpart in the same module.
	Note: If a module has more than one "detail" form, described in **Alternative "detail" forms** (page 13), this will be the name of the form to be displayed by default when selecting a record in the "list" form.
	For example, a "list" form for eparties would have the property:
	`"relatedForms": {`
	`    "detail": "epartiesDetail"`
	`}`
	See **"list" form** (page 6)for details.

`"elements"`	Lists child elements. In a "detail" form these are generally "sections". Each child element is an object with its own "elem", "properties", and possibly "traversal" and "elements".
`"traversal"`	A list of child elements in the order in which they should display in the form.

`"section"`

A `"section"` is similar to a Tab in a module in the EMu desktop client. Typically comprises one or more `"panel"`s (which are similar to field groups in the EMu desktop client).

`"section"`s are accessed from a `"detail"` form's sidebar. In this example, `"section"`s include Summary, Title, Creation, Inscriptions, etc., and the Summary section is selected:



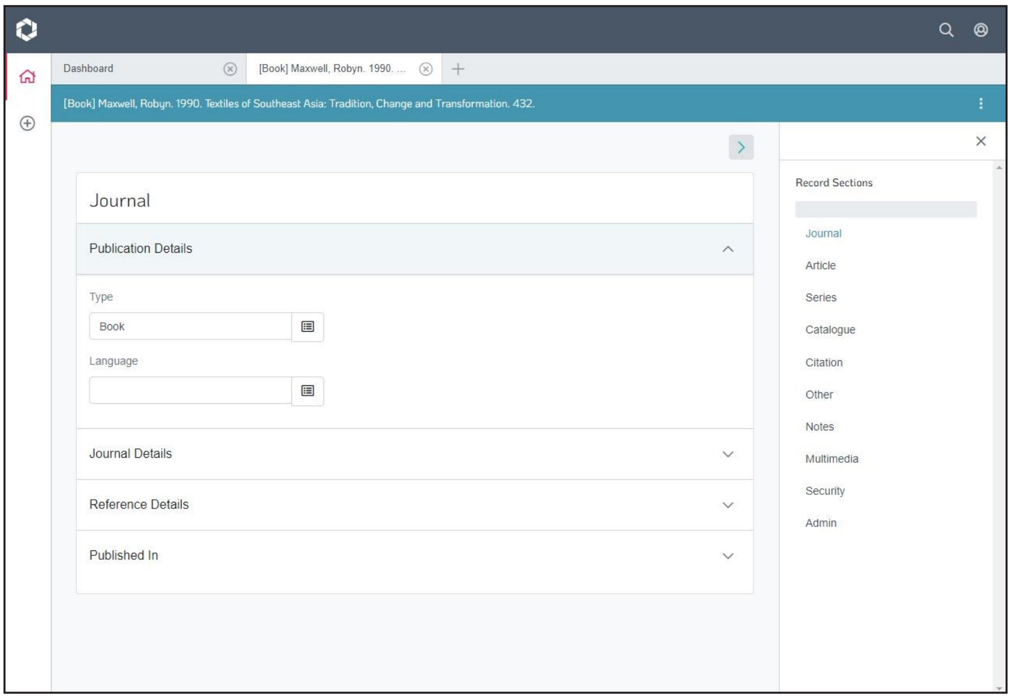
A `"section"` element contains:

Property	Details
`"elem": "section",`	
`"properties"`	This is where tab switching (page 46) properties are generally found, as well as `"label"`. A list of `"properties"` and their definitions is available below (page 43).
`"elements"`	Lists child elements. Each child element is an object with its own `"elem"`, `"properties"`, and possibly `"traversal"` and `"elements"`.
`"traversal"`	A list of child elements in the order in which they should display in the form.

"panel"

A "panel" is similar to a field groups in the EMu desktop client and is used to group fields within a "section".

Comprises one or more elements which may be simple elements, such as "text" fields, or the container element "listGrid". These elements will be grouped in a "panel" that can be expanded and collapsed independently of other "panel"s:

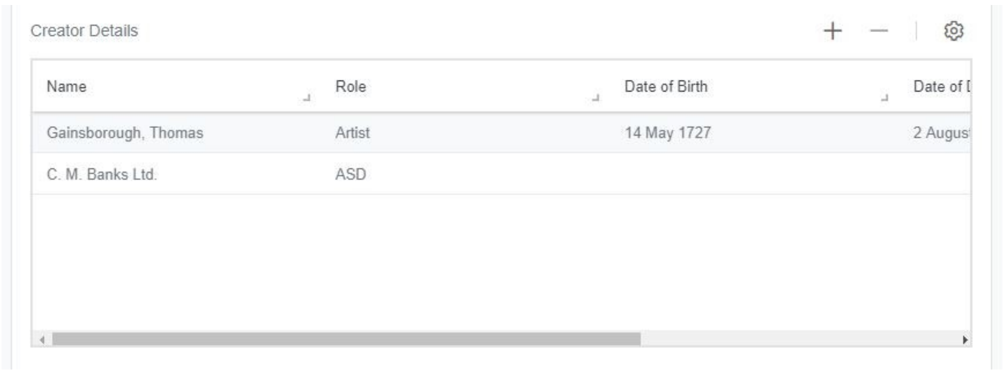


A "panel" contains:

Property	Details
"elem": "panel",	
"properties"	Typically just contains "label". A list of "properties" and their definitions is available below (page 43).
"elements"	Lists child elements. Each child element is an object with its own "elem", "properties", and possibly "traversal" and "elements".
"traversal"	A list of child elements in the order in which they should display in the form.

"listGrid"

Used when a column can contain more than one value. For example an object can have more than one creator, and creators are listed in a grid:



Creator Details

Name	Role	Date of Birth	Date of I
Gainsborough, Thomas	Artist	14 May 1727	2 Augus
C. M. Banks Ltd.	ASD		

As a container element, the grid object contains child elements, which are simple elements bound to columns in the back-end. In the example above, the child elements are:

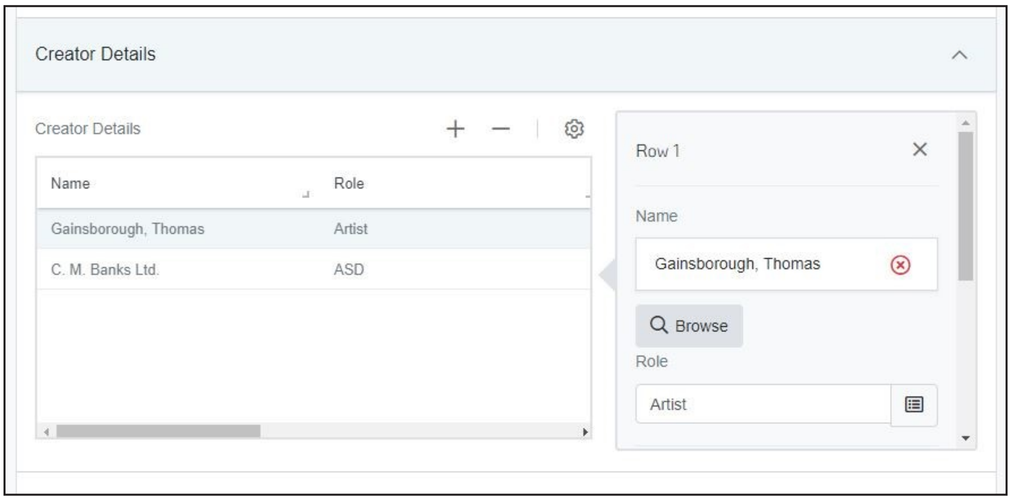
- *Name*
- *Role*
- *Date of Birth*
- *Date of Death*

These elements can be any type of simple element. As it is these elements that are bound to the back-end, and not the "`listGrid`" element itself, the "`listGrid`"element has no "`name`" property.

Currently unsupported features of EMu

At the moment, a "`listGrid`" cannot contain other container elements and therefore elements such as nested tables in the EMu client are not currently supported in Go.

Go also does not support Link Grids (in the EMu client a Link Grid presents a summary of records as rows in a table, and selecting a row displays more record details in fields above the table; for example, the Tasks tab contains a Link Grid). The best way to represent such fields in Go is to include every field in the one "`listGrid`". If a user wants to edit a "`listGrid`", clicking on a row will bring up a side panel with the elements listed in their "`traversal`" order:



Creator Details

Creator Details

Name	Role
Gainsborough, Thomas	Artist
C. M. Banks Ltd.	ASD

Row 1

Name: Gainsborough, Thomas

Browse

Role: Artist

A "`listGrid`" contains:

Property	Details
`"elem": "listGrid",`	
`"properties"`	Typically just contains "`label`". A list of "`properties`" and their definitions is available below (page 43).
`"elements"`	Lists child elements. Each child element is an object with its own "`elem`" and "`properties`". The "`name`" of children elements will generally end with a "`_tab`" or an integer, indicating that more than one value is permitted to be entered in that column.
`"traversal"`	A list of child elements in the order in which they should display in the form.

The "listGrid" above is represented as:

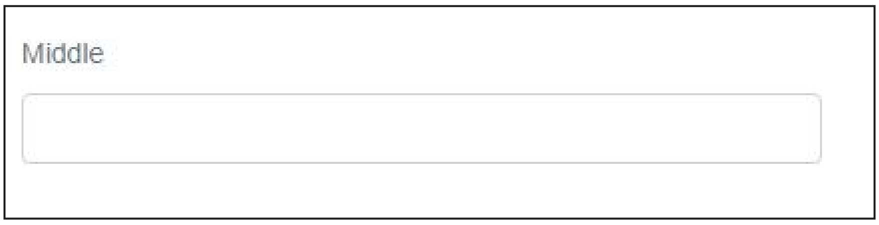
```
{
    "elem": "listGrid",
    "properties": {
        "label": null
    },
    "traversal": [
        "creatorName",
        "creatorRole",
    ],
    "elements": {
        "creatorName": {
            "elem": "reference",
            "properties": {
                "label": null,
                "name": "CreCreatorRef_tab",
                "display": "SummaryData"
            }
        },
        "creatorRole": {
            "elem": "text",
            "properties": {
                "label": null,
                "name": "CreRole_tab"
            }
        }
    }
}
```

where "creatorName" and "creatorRole" are two simple elements, reference and text respectively.

Simple elements

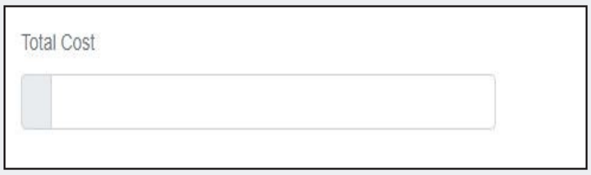
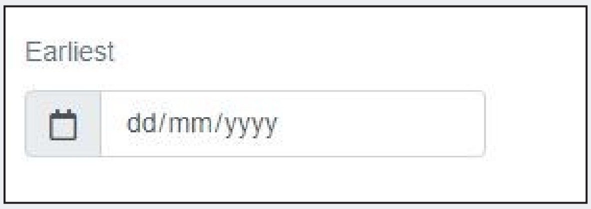
"text"

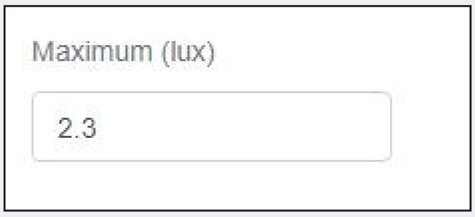
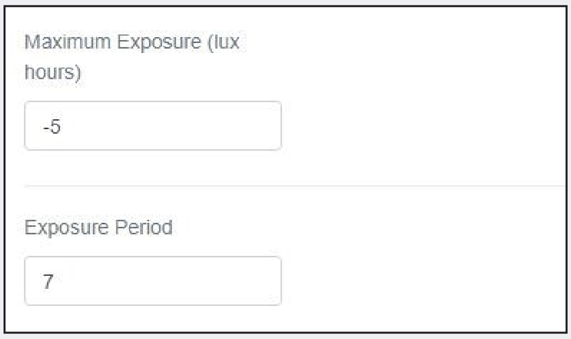
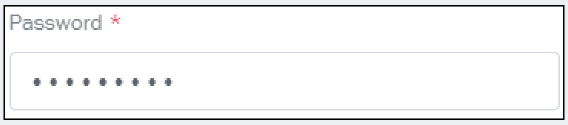
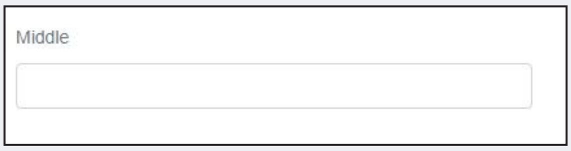
A text field used for inserting and editing text. A standard "text" element appears as a text field with a label:



Other types of "text" field can be specified with the "type" property.

A "text" element contains:

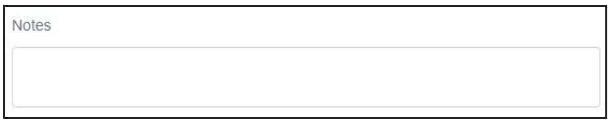
Top-level properties	Details
"elem": "text",	
"properties"	Properties include: <table><tr><th>Property</th><th>Details</th></tr><tr><td>"type"</td><td>Specifies different types of text input field: » "currency" Specified as "type": "currency":  » "date" Specified as "type": "date":  » "email" » "float" A text field with "type": "float" will accept all real numbers that can be expressed in decimal form (up to the length of the text field):</td></tr></table>

Top-level properties	Details	
	Property	**Details**
		 Maximum (lux) 2.3 » `"integer"` A text field with `"type": "integer"` will only accept integers:  Maximum Exposure (lux hours) -5 Exposure Period 7 » `"latitude"` » `"longitude"` » `"password"` Used to ensure text entered into a field is unreadable:  Password * ••••••••• » `"phone"` » `"text"` A standard text field is specified as `"type": "text"`. If no `"type"` property is specified, this type of text field will display by default:  Middle » `"time"` » `"url"` **Note:** If a `"type"` is not specified, `"type": "text"` is assumed.

Top-level properties	Details	
	Property	**Details**
		A list of `"properties"` and their definitions is available below (page 43).
	`"length"`	The size of the input box (integer).
	`"name"`	Binds the element to the named back-end column. Specified, for example, as: `"name": "NamMiddle"` **Note:** This property must be provided.
	`"required"`	Specifies that the user must supply a value (the value is mandatory).
	`"label"`	A label to display with the control. When the value is `null`, the element's `"id"` or relevant text from `"translate"` is used. If this property is absent, no label will appear.
	`"disabled"`	The field will be greyed out and uneditable.

"textarea"

Displays as a text box able to accept multiple lines of text:



A "textarea" element contains:

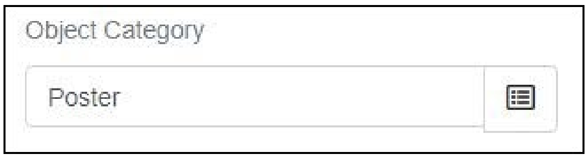
Top-level properties	Details
"elem": "textarea",	
"properties"	Properties include: <table><tr><th>Property</th><th>Details</th></tr><tr><td>"type"</td><td>Must be specified as "type": "textarea". **Note:** This property must be provided.</td></tr><tr><td>"label"</td><td>A label to display with the control. When the value is null, the element's "id" or relevant text from "translate" is used. If this property is absent, no label will appear.</td></tr><tr><td>"name"</td><td>Binds the element to the named back-end column. Specified, for example, as: "name": "NotNotes" **Note:** This property must be provided.</td></tr></table>A list of "properties" and their definitions is available below (page 43).

The "textarea" element pictured above is represented as:

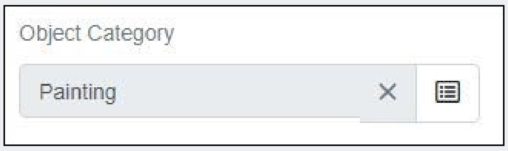
```
{
    "elem": "textarea",
    "properties": {
        "label": null,
        "type": "textarea",
        "name": "NotNotes"
    }
}
```

"lookup"

A "lookup" element contains a text field and Lookup button that brings up a list of possible values for the field:

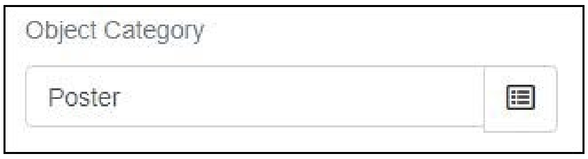


A "lookup" element contains:

Top-level properties	Details
`"elem": "lookup",`	
`"properties"`	Properties include: **Property** / **Details** `"label"`: A label to display with the control. When the value is `null`, the element's `"id"` or relevant text from `"translate"` is used. If this property is absent, no label will appear. `"name"`: Binds the element to the named back-end column. Specified, for example, as: `"name": "TitObjectCategory"` **Note:** This property must be provided. `"mode"`: Specifies how the user can interact with the Lookup field: » `"readonly"`: the user can only select values from the Lookup List. A Lookup with `"mode": "readonly"` will appear with the text field greyed out and disabled:  Object Category Painting » `"readwrite"`: the user can select values from the Lookup List and add a new value to the list. If no `"mode"` is specified, the default for Lookups is `"readwrite"`.

Top-level properties	Details
	A list of "properties" and their definitions is available below (page 43).

For example:



This "lookup" element is represented as:

```
{
    "elem": "lookup",
    "properties": {
        "label": null,
        "name": "TitObjectCategory"
    }
}
```

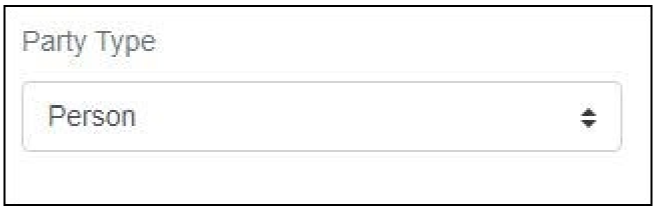
"select"

A "select" element displays as a drop list of values. A user can pick a value but not insert a new value.

A "select" element contains:

Top-level properties	Details
"elem": "select",	
"properties"	Properties include: **Property:** "label" — **Details:** A label to display with the control. When the value is null, the element's "id" or relevant text from "translate" is used. If this property is absent, no label will appear. **Property:** "name" — **Details:** Binds the element to the named back-end column. Specified, for example, as: `"name": "NamPartyType"` **Note:** This property must be provided. A list of "properties" and their definitions is available below (page 43).

For example:



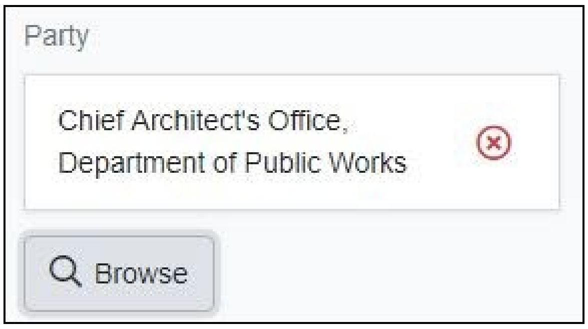
This "select" element is represented as:

```
"partyType": {
    "elem": "select",
    "properties": {
        "label": null,
        "name": "NamPartyType"
    }
}
```

"reference"

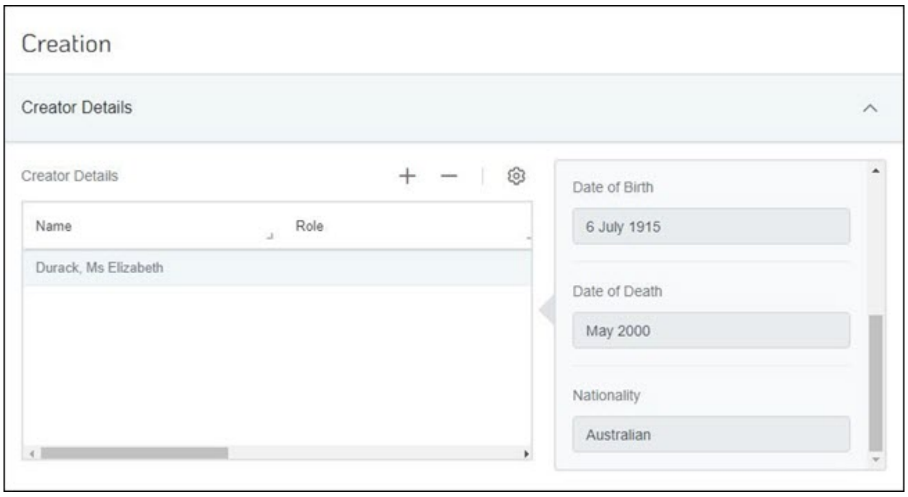
Tip: See the EMu Help (emu.help.axiell.com) for full details about attachments, and for definitions of primary and target records and reverse attachments.

A "reference" element attaches a **target**[1] record to the currently open record (the **primary**[2] record) and shows some of its data in the reference field. A "reference" element will typically display with a browse button, which opens a list of records from a specified module (the example below has referenced a Parties record):



An alternative use of a "reference" is through "text" elements with the same column name as an existing "reference" element. These are used to display more than one column of data from the attached record. These "text" elements will always be disabled; they can only be altered by selecting the Browse button and selecting a different attachment record. They are structured in the same way as standard "reference" elements but have an "elem" of "text" rather than "reference".

The following example shows a "listGrid" where a party record can be attached using a "reference" element. The text fields of *Date of Birth*, *Date of Death* and *Nationality* are "text" elements with the same name as the "reference" element, and differing "display" properties. They display further information from the attached Parties record:



Durack, Ms Elizabeth

Date of Birth: 6 July 1915

Date of Death: May 2000

Nationality: Australian

[1]An attachment is made from one record to another. The record from which the attachment is initiated is the Primary record; the record that is attached is called the Target.

[2]An attachment is made from one record to another. The record from which the attachment is initiated is the Primary record; the record that is attached is called the Target.

A "`reference`" element contains:

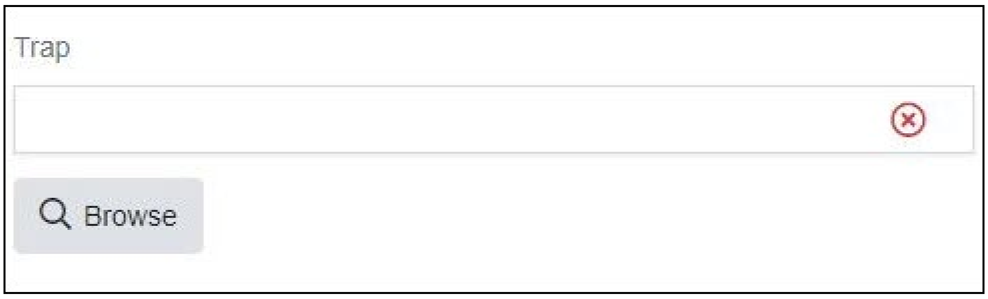
Top-level properties	Details
`"elem": "reference",` -OR- `"elem": "text",`	`"elem": "text",` is used when a "`reference`" element of the same name already exists.
`"properties"`	Properties include: <table><tr><th>Property</th><th>Details</th></tr><tr><td>"label"</td><td>A label to display with the control. When the value is <code>null</code>, the element's "<code>id</code>" or relevant text from "<code>translate</code>" is used. If this property is absent, no label will appear.</td></tr><tr><td>"name"</td><td>Binds the element to the named back-end column. Typically, this ends with <code>Ref</code>, or an integer. Specified, for example, as: <code>"name": "AssAssociationRef_tab"</code> **Note:** This property must be provided.</td></tr><tr><td>"display"</td><td>Specifies which data from the target record should be shown by specifying the back-end column name of the field in the target record's module. **Note:** This property must be provided.</td></tr></table> A list of "`properties`" and their definitions is available below (page 43).

The "`reference`" element pictured above is represented as:

```
{
    "elem": "reference",
    "properties": {
        "label": null,
        "name": "AssAssociationRef_tab",
        "display": "SummaryData"
    }
}
```

where "`AssAssociationRef_tab`" is the back-end column, and "`SummaryData`" is the data from the target record that will be displayed in the form.

If the user does not have permission to access the module of the attached reference, the record data will not fully display. Lookups used to attach new records from that module will be empty, while an existing attached record will exist but be blank:



Reverse references

Reverse references also have an "`elem`" of "`reference`". While a standard "`reference`" shows target records that are attached to the primary record, reverse references are present in target records and show the primary records. For example, a primary record in Parties (eparties) attaches a target record in the Catalogue (ecatalogue). The record in eparties has a standard reference element that shows the target record. The target record in ecatalogue has a reverse reference element that shows the primary record.

These elements will always be read-only as the user is only viewing information from another record. Because the number of primary records attaching to the target record could be more than one, these elements are also always found in a "`listGrid`".

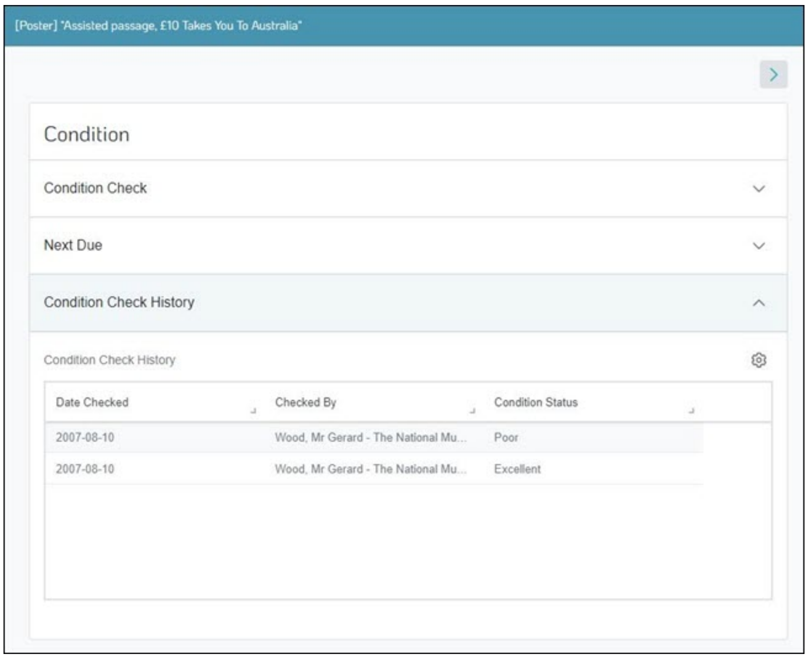
Reverse reference elements have the same structure as "`reference`" elements, with a "`name`" and "`display`" property. However, the "`name`" property in will be different. Because the element is showing records in other modules, with particular values in particular columns, the "`name`" of a reverse reference must specify both a module and a column name from that module. The column will be the column that attaches the target record. An example "`name`" property is:

```
"name": "econdition:ConCatalogueRef"
```

This specifies that primary records in the Condition Checks module (`econdition`) are shown if they attach to the target record using the "`ConCatalogueRef`" column.

A "`display`" property indicates what data from the primary record will be displayed in the target record.

For example, the Condition Check History "`listGrid`" in a Catalogue record is a reverse reference element with the "`name`" property "`econdition:ConCatalogueRef`":



[Poster] "Assisted passage, £10 Takes You To Australia"

Condition

Condition Check

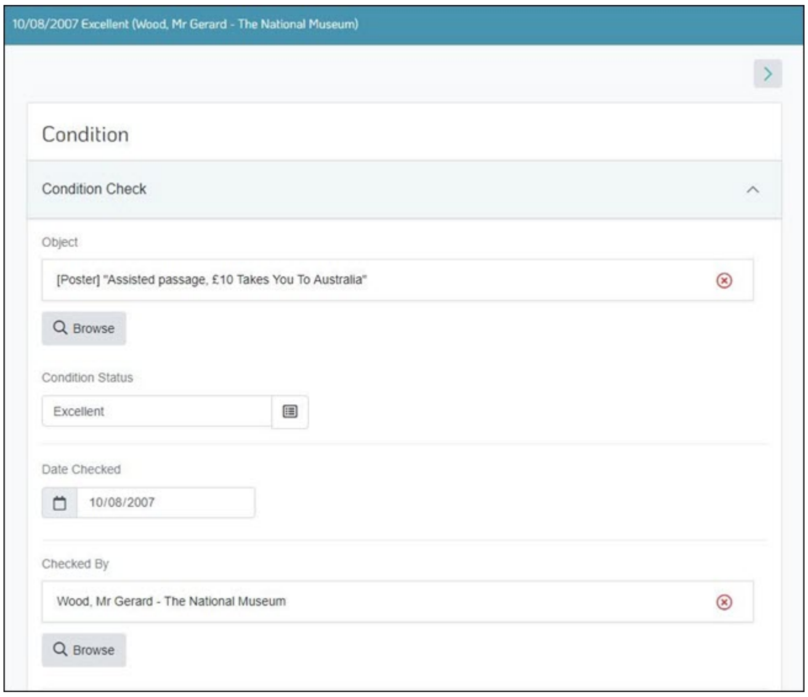
Next Due

Condition Check History

Condition Check History

Date Checked	Checked By	Condition Status
2007-08-10	Wood, Mr Gerard - The National Mu...	Poor
2007-08-10	Wood, Mr Gerard - The National Mu...	Excellent

The data in the "`listGrid`" indicates that there are two primary records that have attached to this target record using the column `ConCatalogueRef` (labelled in Go as the *Object* reference field). These records are in the econdition module:



10/08/2007 Excellent (Wood, Mr Gerard - The National Museum)

Condition

Condition Check

Object

[Poster] "Assisted passage, £10 Takes You To Australia"

Condition Status

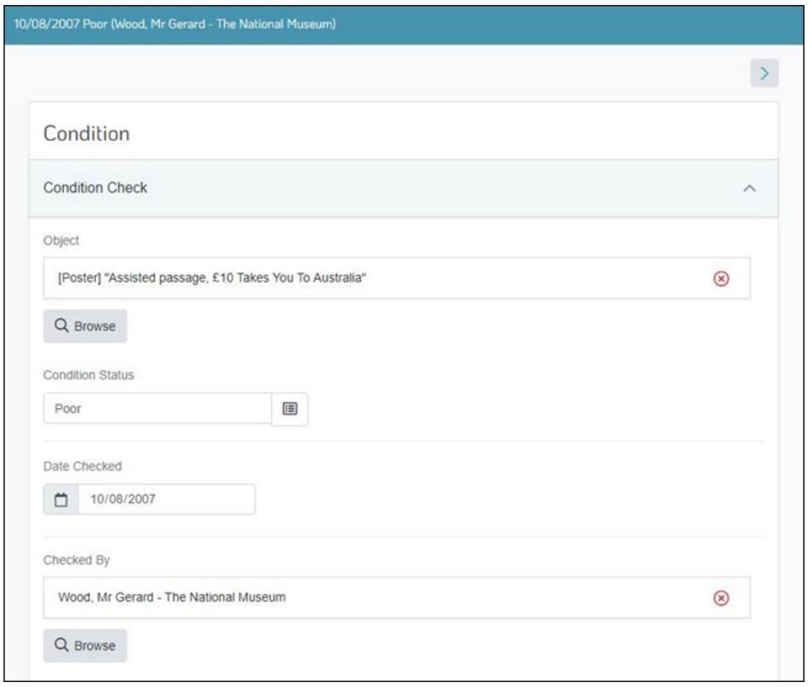
Excellent

Date Checked

10/08/2007

Checked By

Wood, Mr Gerard - The National Museum



10/08/2007 Poor (Wood, Mr Gerard - The National Museum)

Condition

Condition Check

Object

[Poster] "Assisted passage, £10 Takes You To Australia"

Condition Status

Poor

Date Checked

10/08/2007

Checked By

Wood, Mr Gerard - The National Museum

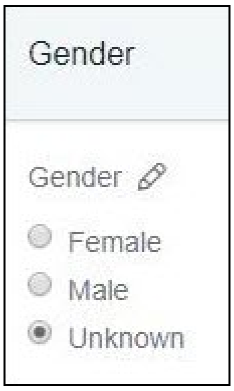
This "`reverse`" reference "`listGrid`" is represented as:

```
"conditionCheckHistoryGrid": {
    "elem": "listGrid",
    "properties": {
        "label": null
    }
    "traversal": [
        "dateChecked",
        "checkedBy",
        "conditionStatus"
    ],
    "elements": {
        "dateChecked": {
            "elem": "reference",
            "properties": {
                "label": null,
                "name": "econdition:ConCatalogueRef",
                "display": "ConDateChecked"
            }
        },
        "checkedBy": {
            "elem": "reference",
            "properties": {
                "label": null,
                "name": "econdition:ConCatalogueRef",
                "display": "ConCheckedByLocal"
            }
        },
        "conditionStatus": {
            "elem": "reference",
            "properties": {
                "label": null,
                "name": "econdition:ConCatalogueRef",
                "display": "ConConditionStatus"
            }
        }
    }
}
```

where `"ConDateChecked"`, `"ConCheckedByLocal"` and `"ConConditionStatus"` are column names from the two primary records in the Condition Checks module.

"radio"

A "radio" element displays as a group of radio buttons, where each radio button represents a value which the user can select:



A "radio" element contains:

Top-level properties	Details
"elem": "radio",	
"properties"	Properties include:

Property	Details
"label"	A label to display with the control. When the value is null, the element's "id" or relevant text from "translate" is used. If this property is absent, no label will appear.
"name"	Binds the element to the named back-end column. Specified, for example, as: "name": "NamSex" **Note:** This property must be provided.
"options"	A list of objects representing the values of each radio button. Each object will have: » "id": a string value used to identify the radio button. » "value": the value the field will have when the button is selected, e.g. "Yes" or "Person". » "label" is typically also provided **Note:** This property must be provided.

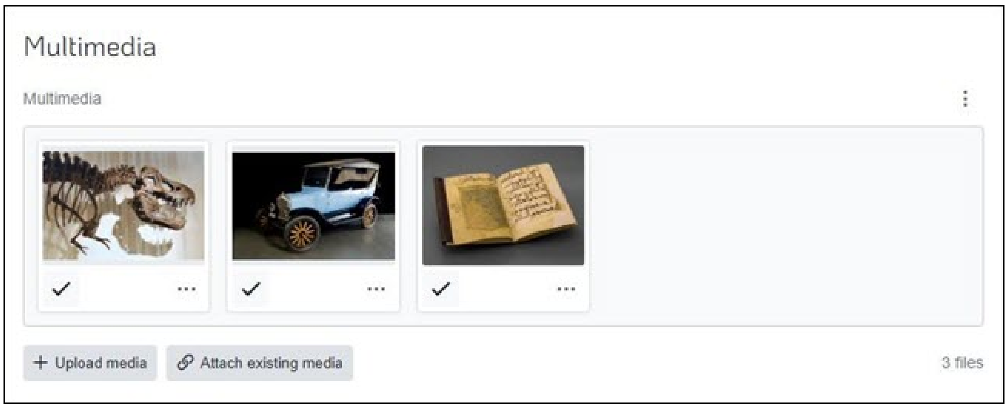
Top-level properties	Details
	A list of "properties" and their definitions is available below (page 43).

The "radio" element pictured above is represented as:

```
"genderRadio": {
    "elem": "radio",
    "properties": {
        "name": "NamSex",
        "label": null,
        "options": [
            {
                "id": "femaleRadio",
                "label": null,
                "value": "Female"
            },
            {
                "id": "maleRadio",
                "label: "null,
                "value": "Male"
            },
            {
                "id": "unknownRadio",
                "label": null,
                "value": "Unknown"
            }
        ]
    }
}
```

"mediaList"

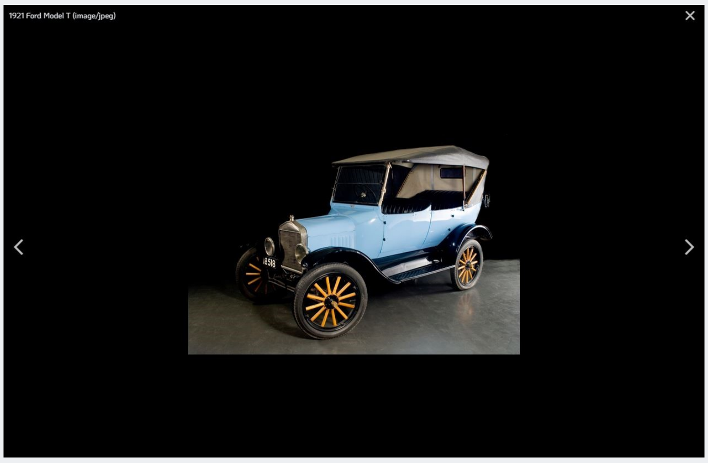
Displays multimedia and enables adding, removing and sorting of multimedia. Options include **Attach existing media** (media saved in the Multimedia module) and **Upload media** (upload new media files):



A "mediaList" element contains:

Top-level properties	Details
"elem": "mediaList",	
"properties"	Properties include: **Property** | **Details** "label" | A label to display with the control. When the value is null, the element's "id" or relevant text from "translate" is used. If this property is absent, no label will appear. "name" | Binds the element to the named back-end column. This will be a reference to the Multimedia module, often "MulMultiMediaRef_tab". "_tab" indicates that multiple values can be entered. **Note:** This property must be provided. "display" | Specifies a text field in emultimedia to be used to label media when it is opened in full-screen mode. For example, the text in the top left corner of the screen is Summary Data for the image:

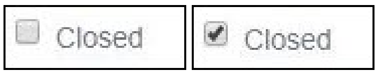
Top-level properties	Details

Property	Details
	 If this property is not present, the text will default to an image record's *SummaryData*.

A list of "`properties`" and their definitions is available below (page 43).

"checkbox"

A checkbox element allows for selection / deselection of a value and will display as unchecked or checked:



A "checkbox" element contains:

Top-level properties	Details
"elem": "checkbox",	
"properties"	Properties include: (see the property table below) A list of "properties" and definitions is available below (page 43).

Property	Details
"label"	A label to display with the control. When the value is null, the element's "id" or relevant text from "translate" is used. If this property is absent, no label will appear.
"name"	Binds the element to the named back-end column. Specified, for example, as: "name": "InfLoanStatus" **Note:** This property must be provided.
"checkedValue"	Specifies what value the field will have when the checkbox is checked. **Note:** This property must be provided.

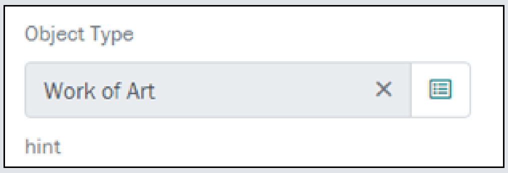
The checkbox shown above is represented as:

```
"inf1LoanClosed: {
    "elem": "checkbox",
    "properties": {
        "label": null,
        "name": "InfLoanStatus",
        "checkedValue": "Closed"
    }
}
```

List of properties

Simple and container elements can have the following properties:

Property	Data type	Details
`"checkedValue"`	string	Specifies what value the field will have when the checkbox is checked, e.g. `"Closed"`.
`"disabled"`	boolean	Indicates whether a field can be interacted with. If `"true"`, the field is greyed out and uneditable.
`"display"`	string	Specifies which data from the target record should be shown in a `"reference"` field by naming a back-end column in the target record's original module.

`"hint"`	string	Text hint describing the kind of data / information expected in the field. Users access the hint by clicking the **hint** link displaying alongside or below the field:  Hints are translated in `"translate"` using the element `"id"` of the `"hint"`:
		`"id_hint"`
		For example, if a `"hint"` has been specified for an element with an `"id"` of `"password"`, the hint is referenced in `"translate"` as:
		`"password_hint"`

`"label"`	string	A label to display with the control. When the value is `null`, the element's `"id"` or relevant text from `"translate"` is used. If this property is absent, no label will appear.
`"length"`	numeric	The size of an input text box (integer).
`"mode"`	string	Describes additional attributes of a field, such as whether a `"lookup"` is read-only. If no `"mode"` property is present for a `"lookup"` element, the default mode is `"readwrite"`.
`"name"`	string	Binds an element to the named back-end column.
`"options"`	list of objects	A list of objects representing the values of each radio button in a `"radio"` element:

Property	Data type	Details
"id"	string	A string value used to identify a specific radio button.
"value"	string	The value the field will have when the button is selected, e.g. "Yes" or "Person".

Property	Data type	Details
"placeholder"	text	A (faded) hint in an otherwise empty text field about what the field requires. Placeholder text disappears as soon as the user begins to enter a value. Placeholder text is translated in "translate" using the element "id" of the "placeholder": "id_placeholder" For example, if a "placeholder" has been specified for an element with an "id" of "password", the placeholder is referenced in "translate" as: "password_placeholder"
"previewElements"	list of string	The columns that will be displayed when viewing a "list" form in List View.
"required"	boolean	Specifies that the user must supply a value (the value is mandatory).
"type"	string	For text elements, this is a string indicating the type of text field that will be displayed, such as "date" or "integer" field.
"visible"	object or boolean	Controls the visibility of an element. Can be a boolean or an object. If boolean, the element is either permanently visible or permanently invisible. If an object, the following properties are available:

Property	Data type	Details
"column"	string	Back-end column controlling visibility of an element based on values entered in the column.
"value"	list of string	The value that must be present in "column" for the element to be visible.

Property	Data type	Details

Property	Data type	Details
`"tab"`	string	Name of the tab in the EMu client with visibility settings that this field emulates.

See Visibility / Tab Switching for details (page 46).

Visibility / tab switching

The "visible" property can be used to control the visibility of an element when viewing a record. The value of "visible" can be an object or boolean. If boolean, the element is either permanently visible or permanently invisible (if there is no "visible" property, the default is "visible": true).

If the value of "visible" is an object, it is possible to emulate **Tab switching**[1] in the EMu desktop client. As such, it is most commonly used on "section" elements, but it can be applied to simple elements or other container elements too. If applied to a container element, hiding that element will also hide its children.

There are two ways of emulating tab switching with "visible":

With "column" and "value"

To determine visibility based on the value in a column, the "visible" property can contain:

Property	Details
"column"	Specifies the name of the column that controls visibility; must be one of the elements contained in the form (though it does not need to be a direct child of the "section").
"value"	Specifies the value that must be present in "column" for the section to be visible.

In this example, "personSection" will only be visible when the Party Type is "Person":

```
"personSection": {
    "elem": "section",
    "properties": {
        "visible": {
            "column": "NamPartyType",
            "value": [ "Person" ]
        }
     },
     "traversal": [ … ],
    "elements": { … }
}
```

An element with the name "NamPartyType" must be present somewhere in the form in order to be able to show and hide these elements. If "NamPartyType" is itself then hidden, the user will be stuck with these settings.

[1]Dynamic adjustment of the tabs displayed in a module based on a user's choices (e.g. different sets of tabs display when "Person" or "Organisation" are selected from the Party Type drop list in the Parties module).

With "tab"

Tab Switching can also be emulated with the "tab" property within the "visible" object. With this approach it is possible to emulate visibility settings for a particular tab exactly as it occurs in the EMu client.

Instead of using the "column" and "value" properties within the "visible" object, a single property, "tab", is used. The value of "tab" is a tab name from the EMu client, e.g. "AllPerTab". In this example, whatever Registry settings have been specified to control visibility of "AllPerTab" in the EMu desktop client will also apply in Go.

For example, "personSection" will display in Go in the same circumstance that the "AllPerTab" would display in the EMu client:

```
"personSection": {
    "elem": "section",
    "properties": {
        "visible": {
            "tab": "AllPerTab"
        }
    },
    "traversal": [ … ],
    "elements": { … }
}
```