# Tasks Developer Manual

***Release 15.4.1.1***

**CONTACT Software**

**Oct 29, 2018**

# Contents

CHAPTER 1

# Configuration

*cs-taskmanager-web* consists of two major elements: A table and a details panel. In order to show heterogenous task objects in a single table, the system has to know which objects are tasks and how to map them to the table's schema. This is done in the application's backend in order to be able to access Powerscript References.

**Warning:** Configuration data and *Frontend Plugins* (page 4) are cached and not updated at runtime. For testing and development purposes, you may call `cs.taskmanager.conf.Cache.refresh()` to completely rebuild the cache.

## 1.1 Task Class Definition

**class** `cs.taskmanager.conf.`**TaskClass**(***values*)

Defines a class of task-like objects and how to display them.

Tasks matching this class definition are identified using an Object Rule, which has to be queryable, since it is compiled to a view `cs_tasks_headers_v` to support fast access.

- **name**: Unique name.
- **classname**: Name of an Elements class.
- **rule_id**: Name of an Object Rule identifying tasks to be displayed for a user. Will usually involve a user-specific variable, such as `$(role)` or `$(persno)`.
- **activities_context**: (Optional) Name of a Powerscript Reference_1 to be used as the channel for the task's activity stream. If empty, the task itself will be used.

The mapping to the tasks table is derived from the *Attribute Mapping* (page 2) relationship containing the class's attribute definitions.

TaskClasses may also specify a number of different *Frontend Plugins* (page 4), a *Context* (page 2), and *Relationships* (page 2).

## 1.2 Attribute Mapping

**class** cs.taskmanager.conf.**Attribute**(***values*)

Attribute mapping of a *Task Class Definition* (page 1). For each *Column Definition* (page 3) and TaskClass, up to one Attribute may exist, telling the system how to map the Column to tasks of this TaskClass.

- **tclass_object_id**: cdb_object_id of a task class definition.
- **column_object_id**: cdb_object_id of a column definition.
- **propname**: An attribute, property, or method of any object of this task class. Will be evaluated at runtime using cs.taskmanager.eval.

## 1.3 Context

**class** cs.taskmanager.conf.**TaskClassContext**(***values*)

Defines a *Task Class Definition* (page 1)'s context path, which is an ordered list of Powerscript Reference names of cardinality 1, into a list of objects, resolving the references depth-first:

- Try to resolve each reference name (once; we don't go down the whole rabbit hole) using the object last resolved as the referer.
- If an object could be resolved, add it to a list of resolved objects.
- Return the reversed list of the resolved objects.

For a task class with context `10: Parent`, `20: Sibling`, `30: Project`, the context of a task `T` would resolve like this (actual results are reversed; first path to not include any `None` values is returned):

1. `T`, `T.Parent`, `T.Parent.Sibling`, `T.Parent.Task.Project`
2. `T`, `T.Parent`, `T.Parent.Sibling`
3. `T`, `T.Parent`, `T.Parent.Project`
4. `T`, `T.Parent`
5. `T`, `T.Sibling`, `T.Sibling.Project`
6. `T`, `T.Project`
7. `T`, `T.Sibling`
8. `T`

- **tclass_object_id**: cdb_object_id of a task class definition.
- **position**: Integer, must be unique for this task class.
- **reference_name**: Name of a Powerscript Reference of cardinality 1.

## 1.4 Relationships

**class** cs.taskmanager.conf.**TaskClassRelship**(***values*)

A *Task Class Definition* (page 1)'s relationship entries are resolved when rendering a task's details. If any related objects could be resolved, a ContentBlock showing the related objects is rendered.

- **tclass_object_id**: cdb_object_id of a TaskClass.
- **position**: Integer, must be unique for this TaskClass.
- **reference_name**: Name of a Powerscript Reference.
- **label**: ID of an Elements label to be used as the ContentBlock's title.

## 1.5 Column Definition

**class** cs.taskmanager.conf.**Column**(**values*)

Columns define the schema of the tasks table. They may also define a custom frontend component to render its cells. If you want to render complex values (e.g. no simple strings or numbers), you will want to specify a plugin.

- **name**: ID of a label naming this column.
- **tooltip**: ID of a label with a more detailed description of this column.
- **plugin_component**: Name of a React component registered in the frontend registry. Note that you can only use components of libraries already included in the application's header, usually only `cs-taskmanager-web` itself.

CHAPTER 2

# Frontend Plugins

A *Task Class Definition* (page 1) may specify a number of plugins to dynamically embed custom detail content. The following plugin IDs are used:

- **cs-tasks-desc**: Component rendered inside task's "Description" ContentBlock.
- **cs-tasks-custom-relship**: Replaces default component used for rendering relationships. May be used to render heterogenous relationships, such as workflow briefcases. The component class should have a `getLabel` string or function.
- **cs-tasks-custom**: Component rendered inside task's custom ContentBlock. The component class should have a `getLabel` string or function.
- **cs-tasks-custom-status-cell**: Replaces default component used for rendering the task's status cells. Used whenever a task class uses a mechanism other than object lifecycles.

For an example of how to replace a default plugin with a custom one, see the administration manual of *cs.taskmanager*.

For more information on plugins, see the documentation of *cs.web*.

CHAPTER 3

# Data

Persistent (non-configuration) data:

- *Settings* (page 5) (for members of a common roles)
- *Personal Settings* (page 6) (for a single user)
- *Read Status* (page 6) (for a single user and a single task)
- *Task Tags* (page 6) (for a single user and a single task)

## 3.1 Settings

Please use the Powerscript module `cs.taskmanager.settings` to read and write user settings.

Default settings are provided as user settings for user `cs.taskmanager.dflt` for technical reasons.

### 3.1.1 Application Settings

Application-level settings for `cs.taskmanager.web` are stored as serialized JSON in `cdb_setting` using the keys `setting_id = "cs.taskmanager"` and `setting_id2 = "settings"`. The JSON data is expected to have the following keys and values:

- **size**: String, determines the width of the details area. Value has to be valid for the CSS property "width". Defaults to "40rem".
- **notificationInterval**: Integer, amount of milliseconds between checks for updated task data.

### 3.1.2 Column Settings

Table-level settings are stored in the same way using `setting_id2 = "cs-taskmanager-web-tasks-table"`. They are maintained by the table component.

## 3.2 Personal Settings

Personal settings have the same JSON schema as *Settings* (page 5). They are stored in `cdb_usr_setting/cdb_usr_setting_long_txt`.

## 3.3 Read Status

**class** cs.taskmanager.userdata.**ReadStatus**(**values*)

To keep track of new or already seen tasks, tasks read by a user have a corresponding entry in this class.

While read_status can be modified to be 0, and thus not "count" as read, the system itself only creates entries with read_status 1 and does not modify it. To set a task back to "unread", the entry is deleted instead.

## 3.4 Task Tags

**class** cs.taskmanager.userdata.**Tags**(**values*)

Tags are personal labels to help users organize their tasks.

CHAPTER 4

# Public Components

**Warning:** Only components documented in this chapter are considered public components. Some components exported/registered for external access only for configuring *CONTACT Tasks*.

The components documented in this chapter are reusable in your own components. Their properties are desribed below. Additionally, you can test them interactively using their storybooks:

- Start a web server with environment variable :envvar:`CADDOK_WEBUI_STORYBOOK` set to `True`,
- Open a web browser and navigate to `/storybook`,
- Select a story of one of the components below to see it in action.

## 4.1 Table Cell Plugins

All table cells get the following React Properties, although they are not used in all cell components:

### 4.1.1 Standard Table Cell React Properties

| Property | Type | Default | Use |
|---|---|---|---|
| column | Immutable.Map | - | The table column this cell represents. |
| row | Immutable.Map | - | The table row this cell represents. |
| isGroup | Boolean | - | `true` if this is a column group header. |

The following components can be used to render columns:

- cs-taskmanager-web-table.cell_components.DateCell.__default__
- cs-taskmanager-web-table.cell_components.IconCell.__module__
- cs-taskmanager-web-table.cell_components.OverdueCell.__module__
- cs-taskmanager-web-table.cell_components.PrioCell.__module__
- cs-taskmanager-web-table.cell_components.ProceedCell.ProceedCell

- cs-taskmanager-web-table.cell_components.ReadStatusCell.ReadStatusCell
- cs-taskmanager-web-table.cell_components.StatusCell.__module__
- cs-taskmanager-web-table.cell_components.TagsCell.TagsCell

## 4.2 Task Details Plugins

To quickly set up a simple description plugin for a new *Task Class Definition* (page 1), you can leverage the supplied template:

```
webmake create custom.module.appname --templates cs.taskmanager:cs-tasks-desc
```

You may use these components in your task details plugins:

- cs-taskmanager-web-details.components.AttributeList.__module__
- cs-taskmanager-web-details.components.LongText.__module__
- cs-taskmanager-web-details.components.TaskContextTree.__module__
- cs-taskmanager-web-details.components.TaskReferenceList._TaskReferenceList

CHAPTER 5

# Redux Store

`cs.taskmanager` uses a Redux store to keep its application state. When running, it will reside under the `cs-taskmanager-web-reducer` key of the global store.

To help you debug your configuration and/or custom plugins, the `DebugInfo` component shows you the store `cs-taskmanager-web-reducer` when hovering over its area. This component is also used in automated frontend tests to determine outstanding asynchronous actions and the number of currently loaded tasks (since they are not always represented by DOM elements due to performance optimization).

**Note:** The `DebugInfo` component will only be visible if activated for the logged in user.

The debug mode is controlled by the class `cs.taskmanager.settings.DebugMode`:

**class** `cs.taskmanager.settings.`**DebugMode**
Enables or disables frontend debug mode for a single user.

```
from cs.taskmanager.settings import DebugMode

DebugMode.activate("caddok")
DebugMode.deactivate("caddok")
```

The following sections describe some of the store's more important contents in detail.

## 5.1 settings

| Key | Description |
|---|---|
| *-pane | User settings for details and file info panes |
| userSelection | If `true`, the current user may display tasks for other users |
| contextClassnames | Filterable context classes. Represent all entries of class `cs_tasks_context` |
| taskClasses | All classes that have a corresponding `cs_tasks_class` entry |
| userViews | The current state of user views, filters and table settings |
| notificationInterval | Interval in milliseconds, after which a check for new/removed tasks is made |

The backend persists most of this data, see *Personal Settings* (page 6) and *Settings* (page 5).

## 5.2 table

Contains standard cs.web table data. Some additional attributes are added to each entry in the `rows` List:

| Key | Description |
|---|---|
| className | Additional CSS class names (most notably the task's read status) |
| @plugin_discriminators | List of identifiers for custom plugins |
| read_status | Number representing the read status of the task. A `0` stands for "unread", a `1` for "read" |
| activities_context | @id of the activity channel to use |

## 5.3 delta

Keeps delta between tasks in backend and frontend. This data is updated whenever the frontend checks for updates, querying the backend for all task ids currently relevant for the selected user.

| Key | Description |
|---|---|
| newTasks | List of `cdb_object_ids` of tasks included in last check that are currently not shown in the tasks table |
| missingTasks | List of `cdb_object_ids` of tasks currently shown in the tasks table that were not part of the last check |
| temporaryTasks | List of `cdb_object_ids` of tasks currently shown striked-through. When updating task data, tasks that were externally removed from the task pool (for example by changing their status), these tasks will be shown striked-through until the next update |
| proceeding | List of `cdb_object_ids` the user just changed the status of. If one of these are not part of the next update, they will be removed instead of shown striked-through |

## 5.4 currentSelection

| Key | Description |
|---|---|
| task_object_id | `cdb_object_id` of the currently selected task. Should always equal the task selection URL part |
| byObjectID | Map of cs.taskmanger-specific task data indexed by `cdb_object_id`. Reflects the resolved configuration for the task's context and relationships and any custom plugin discriminators |

### 5.4.1 Object Maps

Any objects contained in JSON responses from the backend are added to the cs.web object store. Some objects are represented by minified REST objects containing only these attributes:

- `@id`
- `system:ui_link`
- `system:classname`
- `system:description`
- `system:icon_link`
- `system:status`

- `relship:files`

## 5.5 fileContainerID

Contains null, is undefined or contains the `cdb_object_id` of the object selected as the file list pane's context.

## 5.6 inFlight

Contains a list of strings identifying a currently running asynchronous action. Used for displaying activity in the `DebugInfo` component.

## 5.7 userTags

Contains all unique tags of currently displayed tasks as strings. Used for typeahead suggestions when adding new tags.

# List of Figures

# List of Tables

# Index