Tasks

Release 15.4.1.1

CONTACT Software

Oct 29, 2018

Contents

1 Introduction **1**

2 Task classes **2**
2.1 Assigning operations . . . 2

3 Column Definitions **4**

4 Attribute Mapping **6**

5 Context and Relationships **7**
5.1 Context . . . 7
5.2 Relationships . . . 8

6 Plugins **9**
6.1 cs-tasks-desc . . . 9
6.2 cs-tasks-custom-status-cell . . . 9
6.3 cs-tasks-custom . . . 9
6.4 Custom relationship blocks . . . 10
6.5 Example: Adding your own task description plugin . . . 10

7 Data for standard components **12**

8 Range of Values **13**

9 Filterable context classes **14**

10 Settings **15**
10.1 *CONTACT Tasks* settings . . . 15
10.2 Table settings . . . 15
10.3 Updating predefined table settings . . . 15

CHAPTER 1

Introduction

CONTACT Tasks provides a complete personal cockpit of current tasks, along with the most important contextual information and sign-off functionality.

CHAPTER 2

Task classes

CONTACT Tasks can display any object that resembles a task (just called "Tasks" in this document). To be able to do so, object classes have to be configured at *Administration/Configuration → Configuration → Tasks → Task Classes*.

Relevant attributes when creating a task class

Name Unique name, also the primary key. It should be an English name without spaces or special characters. Treat this like a technical system constant.

Class Name Name of a Data Dictionary class. Please make sure that the class is activated for use with the REST API.

Object Rule References an Object Rule, which determines the objects to be displayed, e.g. the personal, currently active tasks of the logged in user. For example, a project task would fit this description, if the user is responsible for it and the task status is "Ready" or "Execution".

No object should match more than one task class. Please make sure all combinations of *Classname* and *Object Rule* lead to mutually exclusive sets of objects at any time.

Activity Context Name of a Powerscript Reference of cardinality 1 defined for all objects of this task class (but not necessarily resolvable). If a valid name is given, the resolved referenced object is used as the context for the task's activities. If not, the task itself is the context.

Deadline Name of a native date attribute of the object class that contains the deadline. Specifying the deadline attribute is optional. If no attribute is specified, no task of this class has a deadline.

Tip: How a task is displayed and which additional functionality (such as completing it) is shown depends on the object class. Please ask your project coordinator at CONTACT Software for assistance with configuring object classes.

2.1 Assigning operations

The table header provides access to operations available for the currently selected tasks. To enable an operation, activate it for use with Web UI and add it to the operation context *CsTasksModal*.

Objects displayed in the detail area's relationship blocks evaluate the operation context *CsTasksDetailReference* (independent of their class).

CHAPTER 3

Column Definitions

CONTACT Tasks displays tasks in a table with configurable columns. Out of the box, *CONTACT Tasks* provides these columns:

Visible?	System?	Name
Yes	Yes	(Selection Column)
Yes	Yes	Unread?
Yes	No	Type
Yes	No	Name
Yes	No	Status
Yes	Yes	Deadline
Yes	No	Tags
No	No	Responsible
No	No	Priority
No	No	Effort
No	No	Progress [%]
No	Yes	Overdue?

Columns that are not visible initially may be shown by changing the table settings.

"System columns" are configured, but contain values calculated by *CONTACT Tasks*. You can create *attribute images* (page 6) for these columns, but they are always overwritten and therefore have no effect.

Application modules can add further columns, for example "Project". You can define custom columns at *Administration/Configuration → Configuration → Tasks → Columns*.

Relevant attributes when creating a column

Label Unique name of the column, also ID of an existing label. The name should be in english and not contain any spaces or special characters.

Title A column's title is derived from both its label and the language used in the current session.

Tooltip Optional label containing descriptive text to be shown when hovering the cursor above the column header.

Component Name of a frontend component as registered in the registry of cs-web-components-base. It must be registered at runtime of *CONTACT Tasks*, so usually only components from `cs-web-components-base` and `cs-taskmanager-web` can be used.

Tip: Details about meaning and content of some of the provided columns can be found in the user manual.

CHAPTER 4

Attribute Mapping

CONTACT Tasks displays tasks of different classes in a single table. This requires mapping each class's attributes to the table's columns. Attribute mappings can be defined in the *Attributes* tab of a task class.

Relevant attributes when creating an attribute mapping

Task ID of a task class.

Column The *Column* (page 4) the attribute mapping refers to.

Attribute Name of an attribute, a Powerscript Reference or a method that exists for objects of this task type. When using Powerscript references of cardinality 1, attributes of the referenced object can be accessed similar to object rules by separating them with dots, for example `Task.Project.cdb_project_id`.

Some of the standard columns require complex values. The class `cs.taskmanager.mixin.WithTasksIntegration` defines the methods "getCsTasks..." to provide these. The chapter *Data for standard components* (page 12) contains information about the data format the columns expect.

Methods for complex values may be overwritten or replaced by custom implementations. Method signatures have to accept the Parameter `request` (using `None` as a default value).

Note: Status columns should always map to the Powerscript method `getCsTasksProceedData`. This method may be called by *CONTACT Tasks* with an additional parameter `targets` to asynchronously load possible target statuses.

CHAPTER 5

Context and Relationships

5.1 Context

The detail block “Context” is always shown. If no context is explicitely defined, it consists of the task itself only. Otherwise, the context is the first path of parent objects as resolved from the configuration.

A context configuration consists of Powerscript References of cardinality 1. They are evaluated in ascending order of their position, but displayed in reverse (which reflects the actual object hierarchy):

This is an example using a checklist:

- Position 10: “ParentChecklistItem”
- Position 20: “Checklist”
- Position 30: “Task”
- Position 40: “Project”

First, the checklist’s “ParentChecklistItem” reference is resolved. If it resolves to an object, the following references will be resolved using this object instead of the checklist itself. If each reference is resolved on the result of the prior reference, the checklist’s complete context “C” is thus:

1. C.ParentChecklistItem.Checklist.Task.Project
2. C.ParentChecklistItem.Checklist.Task
3. C.ParentChecklistItem.Checklist
4. C.ParentChecklistItem
5. C

Is the checklist neither child of a task, nor of another checklist, the context would look like this:

1. C.Project
2. C

Relevant attributes when creating a context entry

Task Class ID of a task class.

Position Integer which has to be unique for this task class's context entries. It determines the resolution order.

PowerScript Reference 1 Name of a PowerScript Reference of cardinality 1 defined for the task class.

5.2 Relationships

Tasks can show any number of relationship blocks in their details area. Every relationship configured (in ascending order of their position) will be resolved and, if not empty, be displayed as a separate block.

Relevant attributes when creating a relationship entry

Task Class ID of a task class.

Position Integer which has to be unique for this task class's relationship entries. It determines the display order.

PowerScript Reference Name of a PowerScript Reference defined for the task class.

Label ID of a label to be used for the block's title.

Objects in relationship blocks also provide all operations of operation context *CsTasksDetailReference* (that are activated for the Web UI).

CHAPTER 6

Plugins

You can control the contents of some elements using plugins, which are frontend components registered in the system.

Task classes may use multiple plugins. They are identified using a discriminator matching the class's name. Only certain plugin IDs are used.

6.1 cs-tasks-desc

This component will be rendered inside the block "Description". Usually, it contains a textual description of the task as a long text attribute. It may also contain interactions that are useful while working on the task, such as updating its completion.

6.2 cs-tasks-custom-status-cell

Replaces the default component to render status values. Should be used for task classes without an object lifecycle or using a different display status for *CONTACT Tasks*.

6.3 cs-tasks-custom

Optional additional block. An example use case would be a dedicated block for recording time spent on a task.

This component should have a function `getLabel` to be used as the block's title:

```
function MyComponent(props) {
    return (
        <div>
            This is an example
        </div>
    );
}

MyComponent.getLabel = function(relship) {
    return (
```

```
        relship && (
            <span>
                {relship.get('mappedName')}
            </span>
        )
    );
}
```

6.4 Custom relationship blocks

If a this plugin's discriminator matches a relationship's name, the relationship will be rendered using this plugin instead of the default one. This plugin is not assigned to the task class directly, but rather to a task class's relationship configuration.

This plugin is used for workflow tasks to "flatten" nested briefcase relationships.

Just like custom blocks, components should define the function `getLabel`.

6.5 Example: Adding your own task description plugin

In the following example, we will replace the task description of project tasks with a custom component.

Note: In the example the customer module "customer.plm" is used. Replace it with your respective customer module.

6.5.1 Step 1: Creating a new web application

Create a new web application in your customer module to contain your customized plugins for *CONTACT Tasks*: `webmake.exe create customer.plm --templates cs.taskmanager:cs-tasks-desc`

The new application is then located in your customer module in the folder `customer/plm/js/src`. The component `customer-plm-TaskDescription` is now available in the cs.web registry.

6.5.2 Step 2: Changing the plugin configuration

1. In the Windows client, navigate to *Administration/Configuration* → *Configuration* → *GUI* → *Web UI* → *Plugins* and search for the plugin with ID `cs-tasks-desc`.
2. Click the "Configurations" tab. You will see the existing plugins for each task class.
3. Change the plugin for the discriminator `cs_tasks_cls_pcs_task`:
   (a) Change "React Component" from `cs-tasks-pcs-task-plugin-PCSTaskDetails` to `customer-plm-TaskDescription`.
   (b) Delete the existing entry in the "Libraries" tab and create a new entry with library `customer-plm`, version `15.3.0` and module assignment `customer.plm`.

6.5.3 Step 3: Testing and iterative customization

Your new task description should now be displayed in the details pane when you select a project task in *CONTACT Tasks*.

If you want to make further adjustments, edit `TaskDescription.jsx` and then call `webmake webpack customer.plm` to prepare the changes for runtime in the web browser.

CHAPTER 7

Data for standard components

The following table shows the data formats to be respected by the attribute configuration in the backend. Some values will be automatically converted by *CONTACT Tasks* before actually being processed by the frontend component. The column `None?` indicates whether an empty value is allowed.

Note: A leading + denotes mandatory keys.

Component	None?	Format
CalculatedPrioCell	No	{prio: int, tooltip: str}
DateCell	Yes	datetime.datetime
IconCell	Yes	{+icon: str, tooltip: str}
ObjectCell	Yes	{@id: str, +system:icon_link: str, system:ui_link: str, +system:description: str}
OverdueCell	Yes	bool
PrioCell	Yes	str
ProceedCell	Yes	{current: StatusCell value, targets: [StatusCell values]}
ReadStatusCell	Yes	int
StatusCell	No	{custom_renderer: str, data: {priority: int, +status: int, +label: str, +color: str, mandatory: [{key: value}]}}
TagsCell	Yes	[str]

CHAPTER 8

Range of Values

The following table shows valid value ranges of components not accepting any values.

Component	Key	Range of Values
CalculatedPrioCell	prio	0 - 100
IconCell	icon	Static URL of an image
ObjectCell	@id	REST URL of an object
ObjectCell	@system:icon_link	Static URL of an image
ObjectCell	@system:ui_link	Frontend URL of an object
PrioCell		cs_tasks_prio_high, cs_tasks_prio_medium, cs_tasks_prio_low
ReadStatusCell		0 - 1
StatusCell	custom_renderer	Identifier of a frontend component
StatusCell	color	Hexadecimal RGB, ex. red: #FF0000

CHAPTER 9

Filterable context classes

To add your own classes to the filter for context objects, you must enable classes individually.

You can create an entry in the class `cs_tasks_context`. You can find the class in the navigation tree under *Administration/Configuration → Configuration → Tasks → Task Contexts*

CHAPTER 10

Settings

Settings for *CONTACT Tasks* and the task table are saved as *User settings* (see Administration Manual *CONTACT Elements: Administration and Configuration*). The default settings are the settings for user `cs.taskmanager.dflt`.

10.1 *CONTACT Tasks* settings

CONTACT Tasks itself keeps global settings with keys `setting_id='cs.taskmanager'` and `setting_id2='settings'`.

Values must be valid JSON and contain these keys:

notificationInterval Interval in milliseconds, after which *CONTACT Tasks* will check for new or already completed tasks. Values below `5000` will be interpreted as `5000`.

Also, these optional keys are recognized:

size Detail pane's width as a value legal for the CSS property "width".

10.2 Table settings

Look for the settings with keys `setting_id='cs.webcomponents.table'` and `setting_id2='cs-taskmanager-web-tasks-table'`.

Values must be valid JSON. Detailed information can be found in the table component documentation (part of `cs.web`).

10.3 Updating predefined table settings

Application modules with *CONTACT Tasks* plugins may also contain table settings. These will be added to the default settings and (if not already present) user settings when updating.

You can also use this mechanism manually:

```
from cs.taskmanager import settings
settings.update_all_settings()
```

List of Figures

List of Tables