ConSol*CM Process Designer Manual (CM up to version 6.9.3)

Table Of Contents

1 Introduction to the ConSol*CM Process Designer

1.1 ConSol*CM for Business Process Management

ConSol*CM is a **customer centric business process management system**. Using ConSol*CM you can control and steer business processes with a strong focus on human communication and interaction, e.g. user help desk, customer service processes, marketing and sales, or ordering processes. Basically, every process that is in operation in a company can be modeled and brought to life with ConSol*CM6.

Using ConSol*CM you can handle all components which are relevant in business processes to represent and control your company's processes in an optimal way. ConSol*CM is used in various different industries and branches ranging from insurances and banks over fashion designing companies to producers of ticket vending machines or car washes. The flexible process designing mechanism and workflow engine provide a perfect basis for the modeling and controlling of business processes of different kinds.

1.2 This Manual

1.2.1 Before You Read this Book ...

When you read this manual, your company is presumably using ConSol*CM6 as a business process management tool and it is your job to administer the system and to implement your company's processes in the application. The book will help you to understand the principles of ConSol*CM workflows and to learn the work with the Process Designer. Numerous *tips and tricks* provided by our experienced consultants will help you to find the best way to improve your processes.

Before you start work with the Process Designer you should have a profound knowledge of ConSol*CM administration, because programming CM workflows requires the usage of several CM components which are configured before (or while) the workflow development takes place. So please read the *ConSol*CM Administrator Manual* first.

1.2.2 The Book's Structure

1. First, some basic components of business processes in general are explained (see this section).
2. Then, an overview of the implementation of the processes in ConSol*CM is given (see section Basic Components of ConSol*CM Processes).
3. Following this, the Process Designer is explained in detail (see sections Work with the Process Designer Application and Components of ConSol*CM Workflows).
4. The sections Process Logic, Workflow Programming, and Best Practices provide expert knowledge about workflow development.
5. Since every workflow has to be deployed to become active, the section Deploying Workflows treats this topic.
6. In the appendices, you find lists of all important terms that are used in the book (glossary), of all annotations (important for the GUI design), and properties (important for the CM system management). Please see also the trademarks page.

1.2.3 Layout Explanations

In order to emphasize and/or mark a section, icons are used.

Information:

This is an additional information.

Attention:

This is an important note. Be careful here!

Warning:

This is a warning!

Tip:

This is a recommendation from our every-day consulting life.

1.3 Business Processes

In a business process, a certain number of tasks have to be performed in a defined order to achieve a specific goal.

The following components are (usually) relevant in business processes. Please see section Basic Components of ConSol*CM Processes to gain an overview of the ConSol*CM objects which represent those components.

- **Process**
  This is a collection of tasks which have to be performed in a certain order. Tasks might be serialized or performed in a parallel way. In ConSol*CM, the process is modeled by one or more workflows. ConSol*CM can model single processes and can also manage complex process chains.
  Each process has to have a defined input and a defined output. The object which represents a case and which runs through the process is a *ticket*. For the end user, it can be named *Ticket* or *Case* or any other required term.
- **Roles and responsibilities**
  Usually, the persons who work in a process represent different roles, i.e. different responsibilities. In ConSol*CM each engineer, i.e. each person who works with the system, can have one or more roles.
- **Access permissions**
  A business process management system can control various processes in a company. Therefore the assignment and control of access permissions is a core functionality. In ConSol*CM, the access permissions are assigned to roles.
- **Customer**
  This is the person who has an interest in the outcome of the process. In ConSol*CM, there is always one main customer for a ticket. This can be a person, i.e. a contact, or this can be a company. More customers can be added.
- **Tasks**
  In a business process, there might be several kinds of tasks:
  - manual tasks
  - system-aided tasks
  - fully automatic tasks

  ConSol*CM can manage all types of tasks. For manual tasks, there are to-do lists for the engineer and several mechanisms which guarantee that no task will be forgotten or ignored.

1.4 Introduction to Workflows in ConSol*CM

One of the core components of ConSol*CM is a powerful workflow engine. Hence, a process is represented in ConSol*CM by a **workflow**. This is the technical representation of the consecutive steps which are required to fulfill all steps which should be performed during the business process.

Examples:

In an IT helpdesk environment, a workflow could consist of the steps:
New Ticket - Accept Ticket - Work on Solution - Inform Customer - Close Ticket.

In a sales process these steps could be:
First Contact: Lead - Second Contact: Opportunity - Contract Candidate - Contract.

The workflow containing all required steps runs in a workflow engine. In this manual you will get to know the details about all components of a workflow and how to use them to build the workflow which represents your business process.

A workflow ...

- represents a specific process, e.g. the steps that have to be performed to handle a customer request.
- puts activities and decisions in a defined order.
- defines the possible paths a ticket can take.

The case or request which has to be dealt with is represented by a **ticket**, i.e this is the object which passes through the workflow.

The following picture shows the graphical representation of a simple help desk process.

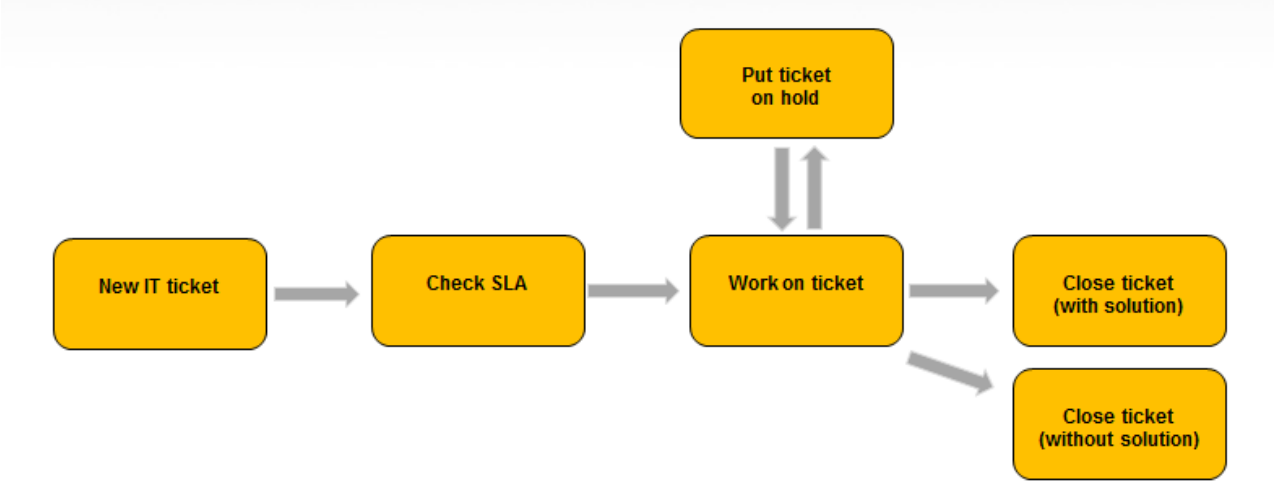


Fig. 1: ConSol*CM Process Designer - Process: Simple Representation

1.5 The ConSol*CM Process Designer at a Glance

1.5.1 Modeling Workflows

A business process is modeled in ConSol*CM using the *Process Designer*, an application which is an integral element of a standard ConSol*CM installation. A process can be represented by one or more workflows, i.e. you use the *Process Designer* to develop workflows.

In ConSol*CM terminology, a *workflow* always represents the technical entity, whereas a *process* represents the business process from the logical or management point of view.

One of the Process Designer's advantages is that there is no procedural gap between process design and workflow implementation. You can design a workflow for a process using the graphical interface of the Process Designer and as soon as you have assigned the workflow to a queue and have defined roles and users, the process comes alive and engineers can work with it. That means you can use the Process Designer for both steps which are of importance when you want to create IT-supported business processes:

- Model and design the process from a logical point of view
- Implement the process in a technical instance

Due to this flexibility, you can start with a simple version of a workflow, usually in a test environment, and develop the desired functionalities of the process using an iterative approach. In each step of the development and optimization process the team of engineers can test if the use cases are represented as desired.

The graphical representation of a workflow in the Process Designer is very similar to the *Business Process Model and Notation* (BPMN) and can be handled in a very intuitive way.

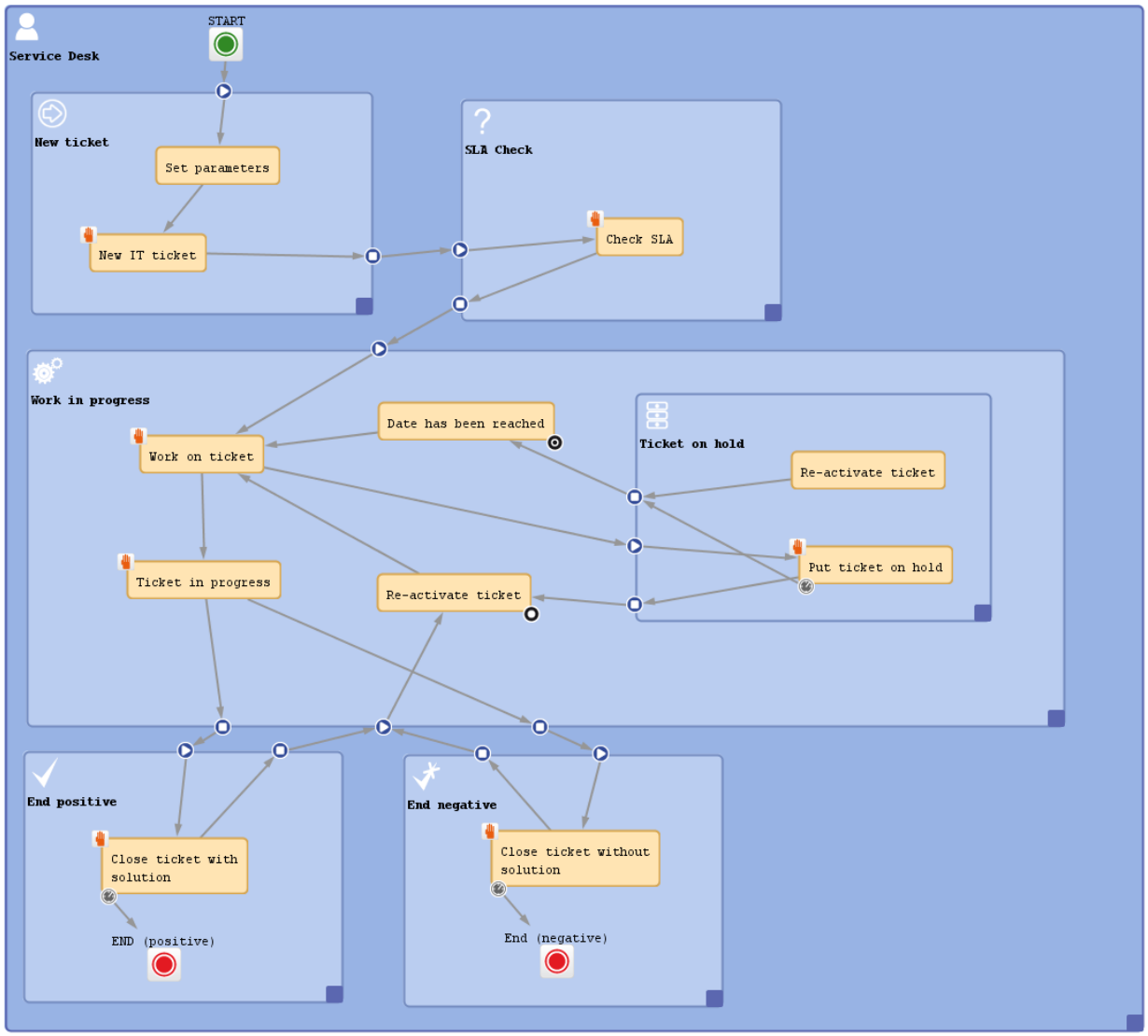


Fig. 2: ConSol*CM Process Designer - Workflow Modeling the Process in the Previous Figure

Read the following sections to get a first impression of the Process Designer's features and functionalities. All topics will be explained in detail in the respective chapters of the manual.

1.5.2 Tickets and Activities

Each case, which has to be treated, will be represented by a *ticket*. Thus a ticket is a concrete run through a workflow. This can be a request, an order, or any other task which has to be processed in a business process.

When a new ticket is created within ConSol*CM, it is associated with a workflow (via the queue it belongs to). At first the new ticket is in the START node. During its further life cycle the ticket runs through the various activities of the workflow. Its life cycle ends when it has reached an END node.

You model a process in a workflow by connecting activities in a specific order. The result is a directed flow graph. It shows which activities have to be carried out for a ticket in order to run through the workflow (and thus the business process) successfully. Workflows can have branches so that different flow paths are possible. In this way, you can make sure that, for example, a ticket first has to be accepted, then the problem has to be solved, then the solution has to be documented. Only then the ticket can be closed.

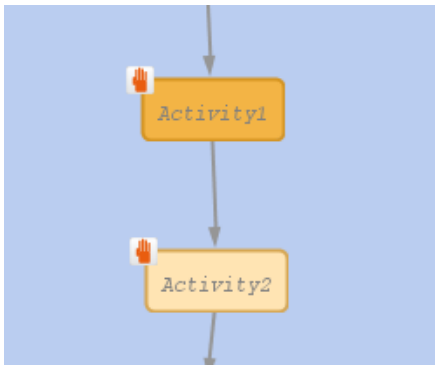


Fig. 3: ConSol*CM Process Designer - Two Sequential Manual Activities

There are manual and automatic activities. Manual activities require engineer interaction and are offered as *Workflow activities* in the Web Client. In contrast, automatic activities are performed without any human input and are kept away from the engineer. This enables ConSol*CM to save time for the engineer and to process data from various sources behind the scenes. Only when user interaction is required, the process will come to a halt and wait for engineer input.

Fig. 4: ConSol*CM/Web Client - Workflow Activities

1.5.3 Drag & Drop Modeling of Workflow Components

You can develop your workflow easily and intuitively using drag-and-drop. Drag the required workflow elements, e.g. an activity or a decision node, from the palette to the work space and link them. Then adjust the properties of the elements within the Properties Editor.

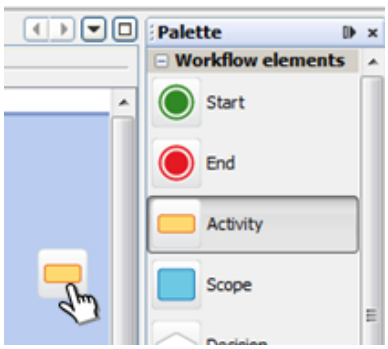


Fig. 5: ConSol*CM Process Designer - Drag & Drop Activities

Using basic elements you build complex workflows step by step. In this way you can model even the most sophisticated business processes.

1.5.4 Scopes and Nesting of Scopes

During a process, a ticket passes through different status, e.g. new ticket, pre-qualification, active work, and documentation. It might even have to be set on hold for a certain period of time. All those status are represented by scopes. In each scope, there can be one or more activities. In this way, it is easy to develop workflows with a clear structure. Scopes can even be organized in a hierarchical way, e.g. during documentation the ticket has to be set on hold. So, using hierarchical scopes you can even keep track of complicated processes. Choose the level of detail you need any time you want.

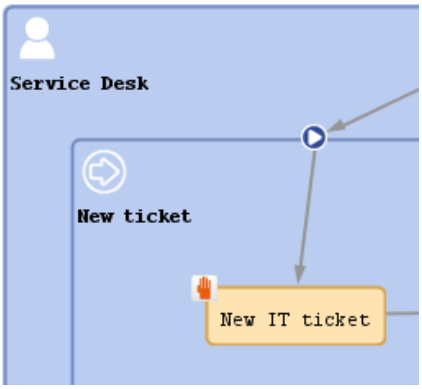


Fig. 6: ConSol*CM Process Designer - Nesting Scopes

1.5.5 Modeling Escalation Mechanisms (Triggers and Wait States)

In most business processes, adherence to schedules and deadlines is indispensable. ConSol*CM helps stick to deadlines and prevents delays by providing automatic timer triggers. These triggers measure for example the reaction time or they initialize reminders.

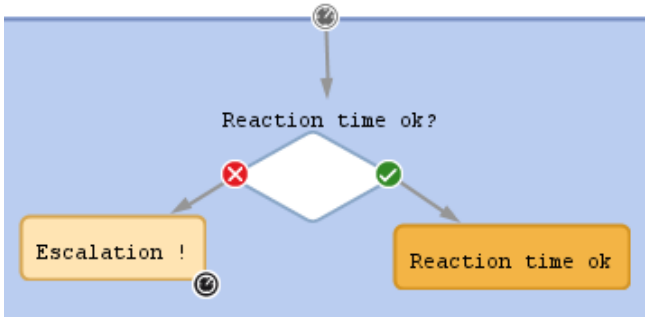


Fig. 7: ConSol*CM Process Designer - Triggering Processes

1.5.6 Modeling Interrupts and Exceptions

In the real world, tasks of a process are not always performed step by step, but may be interrupted by exceptional events. These can be various external incidents. To model such interrupts sequentially is often very complex or even impossible. The Process Designer provides extensive tools to do this.

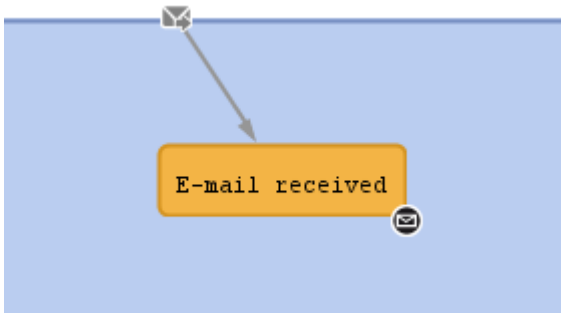


Fig. 8: ConSol*CM Process Designer - Modeling Interrupts

1.5.7 Scripting Capabilities

The process which has been modeled as a ConSol*CM workflow cannot only consist of basic elements like activities or decision nodes. In every node of the workflow a script can be added to provide the *intelligence* of the process. For example, e-mails can be sent to customers or to engineers, interactions with other systems can be implemented, tickets can be handed-over. Basically, all operations which can be implemented in Groovy scripts can be performed.

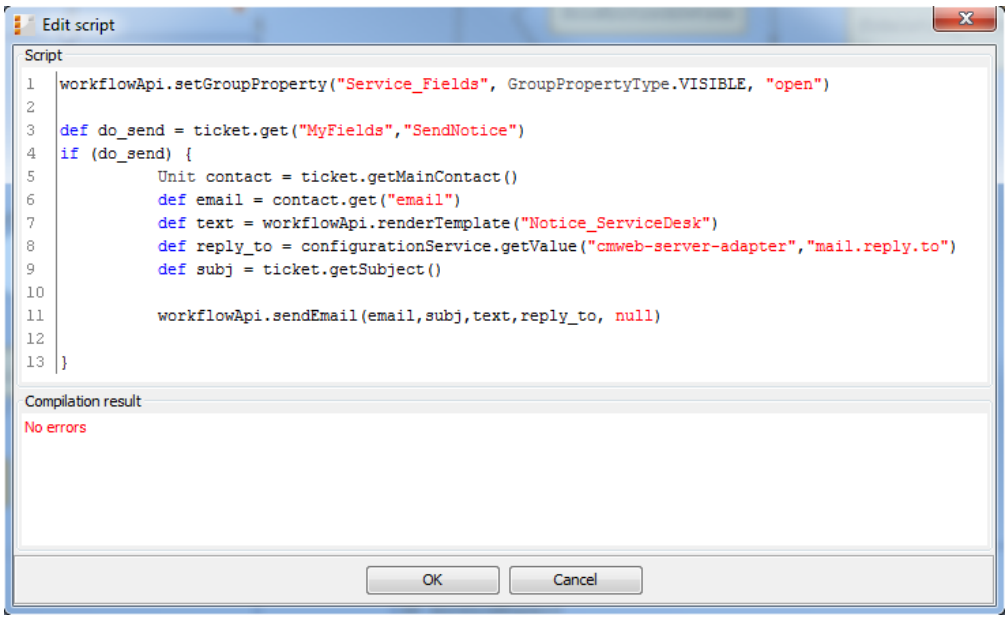


Fig. 9: ConSol*CM Process Designer - Script of an Activity

1.5.8 Versioning of Workflows

Business processes are changing constantly, following the changing requirements of the economic and technical environment. The Process Designer provides continuous versioning of installed workflows. In this way, you can easily discard a new workflow (e.g. when you have tested a new implementation during system development) and go back to one of the previous versions.

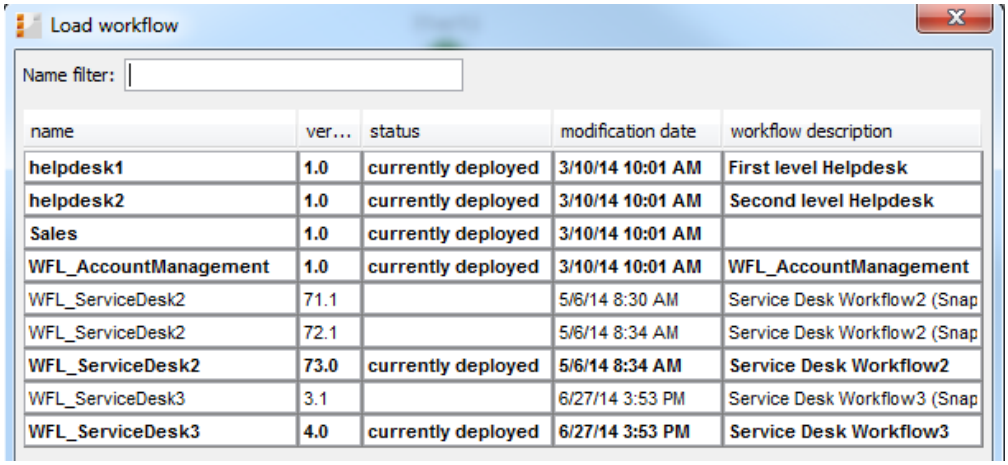


Fig. 10: ConSol*CM Process Designer - Workflow Versions

2 Basic Components of ConSol*CM Processes

2.1 General Objects

During process design and workflow development you will have to deal mainly with the following objects:

Mandatory objects:

- **Ticket**
  This represents the case. Depending on the use case this can be, for example, a help desk case, a sales opportunity, a direct order, or a service request.
- **Primary customer**
  This is the person, i.e. the contact, who is the client, the initiator of the ticket. In ConSol*CM version 6.9 and higher, this can also be a company. The customer represents the external side of the ticket.
- **Queue**
  This is the organizing unit within the ConSol*CM system which groups tickets of one realm and which is access point for the assignment of access permissions and of the workflow. One queue has exactly one workflow which cannot be changed. For example, in a company, there could be one queue for the sales department, one for the customer service, and one for the internal IT.
- **Engineer**
  This is the person who is responsible for completing the tasks in the ticket. A ConSol*CM engineer has a login and password for the Web Client. The main engineer can also be called the ticket owner. It can change during the process.
- **Workflow**
  This is the design or model for the process. A workflow is assigned to a queue (and can be assigned to more than one queues). Hence, all tickets which are in this queue run through the process defined by this workflow. The workflow elements, e.g. activities, conditions, or decisions, represent the most important means in ConSol*CM to configure and control the process flow. One workflow can be assigned to one or to several queues, e.g. the IT service desk team as well as the customer service team, both could work with the workflow *serviceWorkflow*.
- **Custom fields** (CM versions 6.8 and 6.9) and **data object group fields** (CM version 6.9)
  These are the data fields which are used to define the data model for the ticket and customer data. They also determine the GUI design of the Web Client. Custom fields are never defined on a single-field basis, but always in *custom field groups*.
  In ConSol*CM version 6.9 and higher, we call the data fields for ticket data *custom fields* and the data fields within the customer data model *data object group fields*.

Optional objects:

- **One or more additional customer(s**)
  In addition to the main customer, i.e. main contact or (version 6.9 and higher:) main company, more contacts (or companies) can be added to a ticket. For each additional customer a customer role might be assigned. For example, there might be a representative for someone who has opened the ticket or the team manager should also be a contact for a support case. An additional customer can become the main customer during the process and vice versa.

- **One or more additional engineer(s)**
  Additional engineers can be added to a ticket in specific roles which are defined as required. For example, a supervisor might be set as additional engineer to give an approval (role *approver*) or a QA team member can be added to the ticket in the role *QA* to check the result before the ticket is closed.

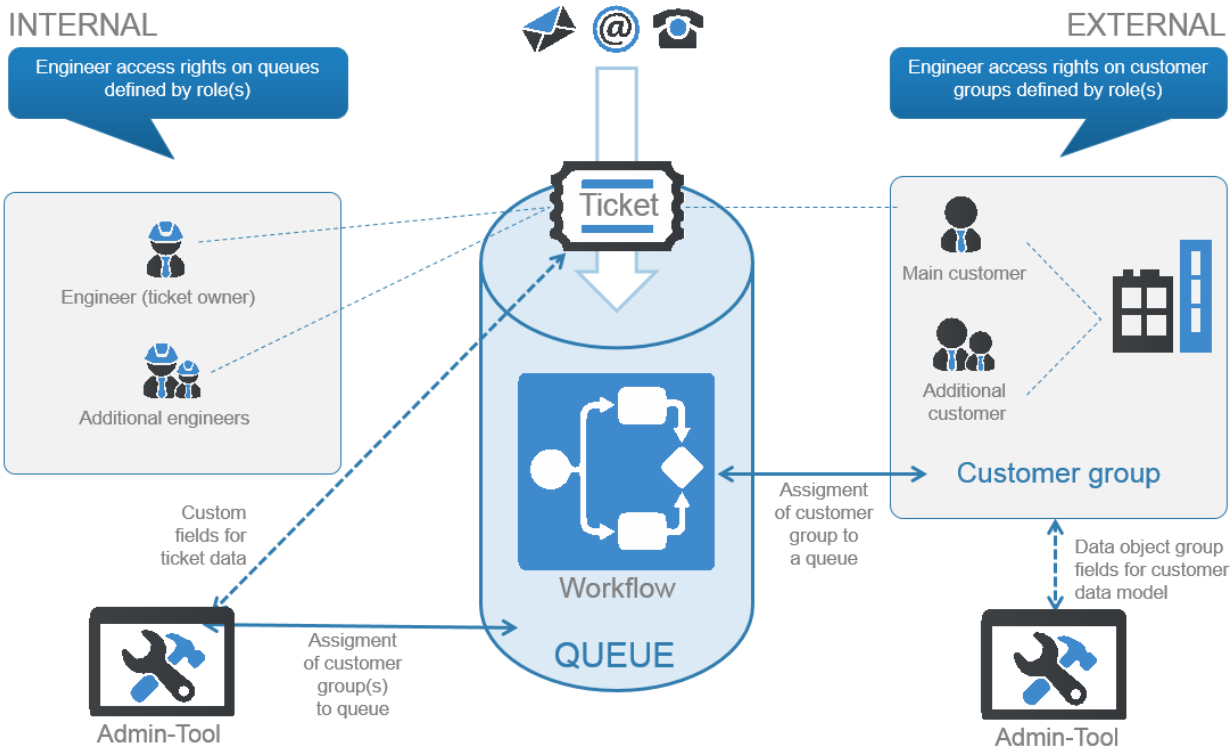


Fig. 1: ConSol*CM - Basic Principle

2.2 Data Fields

2.2.1 Data Fields in ConSol*CM Versions 6.8 and Earlier

Custom fields are data fields of a specific data type which can contain ticket or customer data. The custom fields in their entirety define the data model of the ConSol*CM system. All custom fields can be configured as required, i.e. you as a system administrator can create as many custom fields as you think suitable and can place them in the Web Client GUI where you like or where the best usability will be given.

Custom fields are always managed in *custom field groups* never on a single-field basis. Of course, you can read or set the value of a single field when you write workflow scripts, but in the Admin-Tool as well as in the Process Designer, a great number of operations can only be performed for custom field groups, e.g. fading-in the group, placing the group data in a tab, or assigning the custom field group to a queue. In scripts, in general, you access a field using the following notion:

Access to content of custom field, CM versions 6.8 and earlier

```
ticket:
ticket.get("<group name>.<field name>")

unit:
unit.get("<field name>")
```

The initial definition of custom field groups and custom fields is done using the Admin-Tool. Ticket data are defined in the *Custom Field Administration* section for *Ticket data* and *Customer data* in the respective tabs.

2.2.2 Data Fields in ConSol*CM Versions 6.9 and Higher

Starting with CM version 6.9.0, there are two types of data fields:

- **Custom fields**
  Used to define ticket data, managed in custom fields groups, as known from previous CM versions.
- **Data object group fields**
  Used to define customer data as part of the FlexCDM, the new customer data model. Managed in data object groups.

You can access the content of a custom field or a data object group field using the following notation:

Access to content of data object group field, CM versions 6.9 and higher

```
ticket:
ticket.get("<group name>.<field name>")

unit, for one field:
unit.get("<group name>:<field name>")
```

2.3 Standard Ticket Data Fields

Some fields do not have to be defined as custom fields in the Admin-Tool, because they are always present. These are the following fields of a ticket:

- **Ticket ID**
  Invisible for the user, only internal use in the database.
- **Ticket name**
  Visible in the Web Client, usually called ticket number.
- **Ticket subject**
  Must be set.
- **Create date**
  Is set automatically by the system.
- **Engineer/ticket owner**
  Can be null or one of the engineers.
- **Queue**
  The current queue of the ticket.

3 ConSol*CM Process Designer Manual - Work with the Process Designer Application

3.1 Work with the Process Designer Application

3.1.1 Steps to Perform for a New Process

The work with the Process Designer is one of the first steps in the pipeline of steps which you have to perform when you want to create a new process with users, roles etc. Before we start explaining how to work with the Process Designer, we will therefore provide a short list of tasks you have to do:

1. Design and deploy the workflow using the Process Designer.
2. Create a new queue with this workflow. Here, you will also need the definition of all required custom fields and customer groups.
3. Create the views for the new users/engineers using the scopes of the new workflow.
4. Create one or more role(s) that have access to the new queue. Keep in mind that the access to the customer group(s) must match that of the queue.
5. Create one or more engineers/users and assign the new role(s) to them.
6. Check the login in the Web Client. Can you create a ticket in the new role?

3.1.2 Start of the Process Designer

You can start the Process Designer on every PC or laptop where a standard web browser is installed (please see *System Requirements*) and which has network access to the ConSol*CM server and database.

To start the Process Designer, open the ConSol*CM start page and click on the Process Designer hyperlink. Java Web Start (JWS) is required to start the Process Designer application which runs on the local machine. However, JWS is an integral part of all Java distributions nowadays so that should not be a problem.

Information:

In case the Process Designer cannot be started, the network connection might be the problem. On Windows systems, check the Java parameters for network connections. *Use direct connection* might be required. On Linux systems, check the proxy settings.

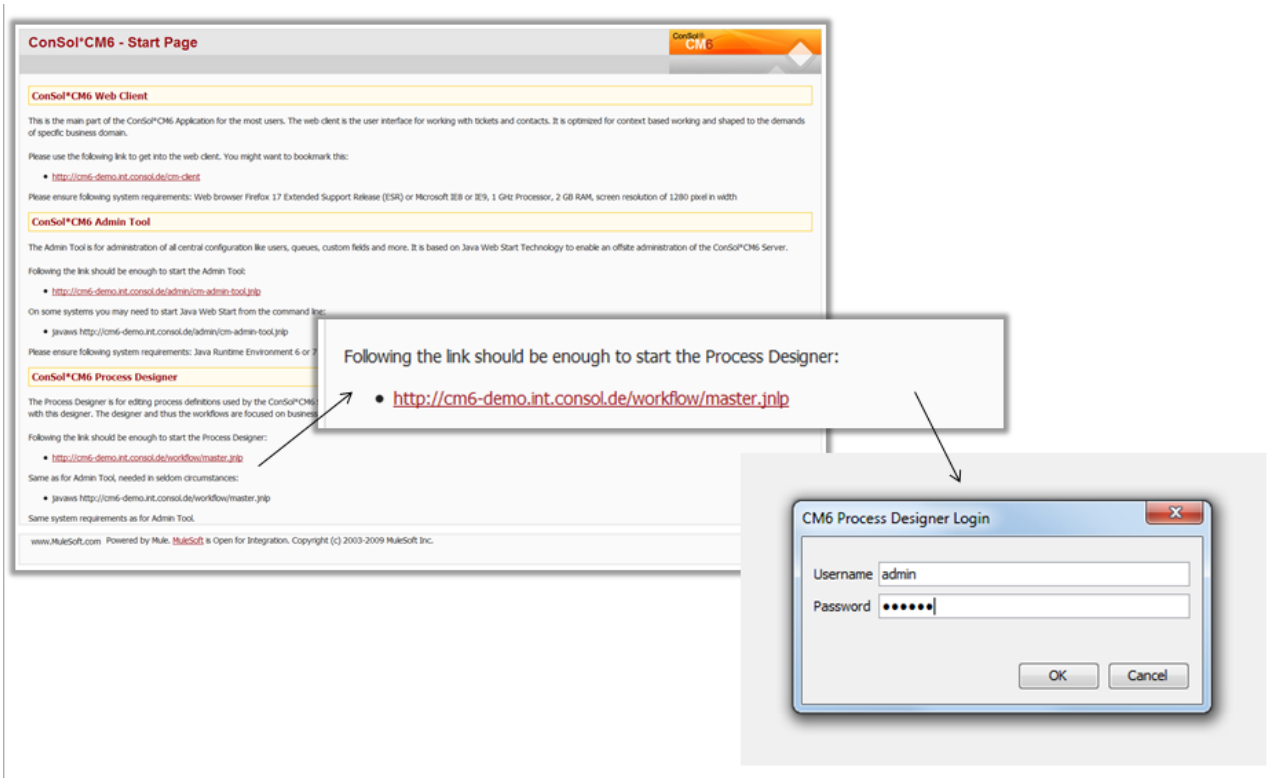


Fig. 1: ConSol*CM - Start of the Process Designer

Log in with an administrator account or with an account which has the workflow management permissions. Please refer to the *ConSol*CM Administrator Manual*, section *Role Administration*, for details.

3.2 Process Designer GUI

3.2.1 Introduction to the Process Designer GUI Elements

Overview: GUI Sections

The Process Designer GUI contains the following elements, please see the next figure and the list below.

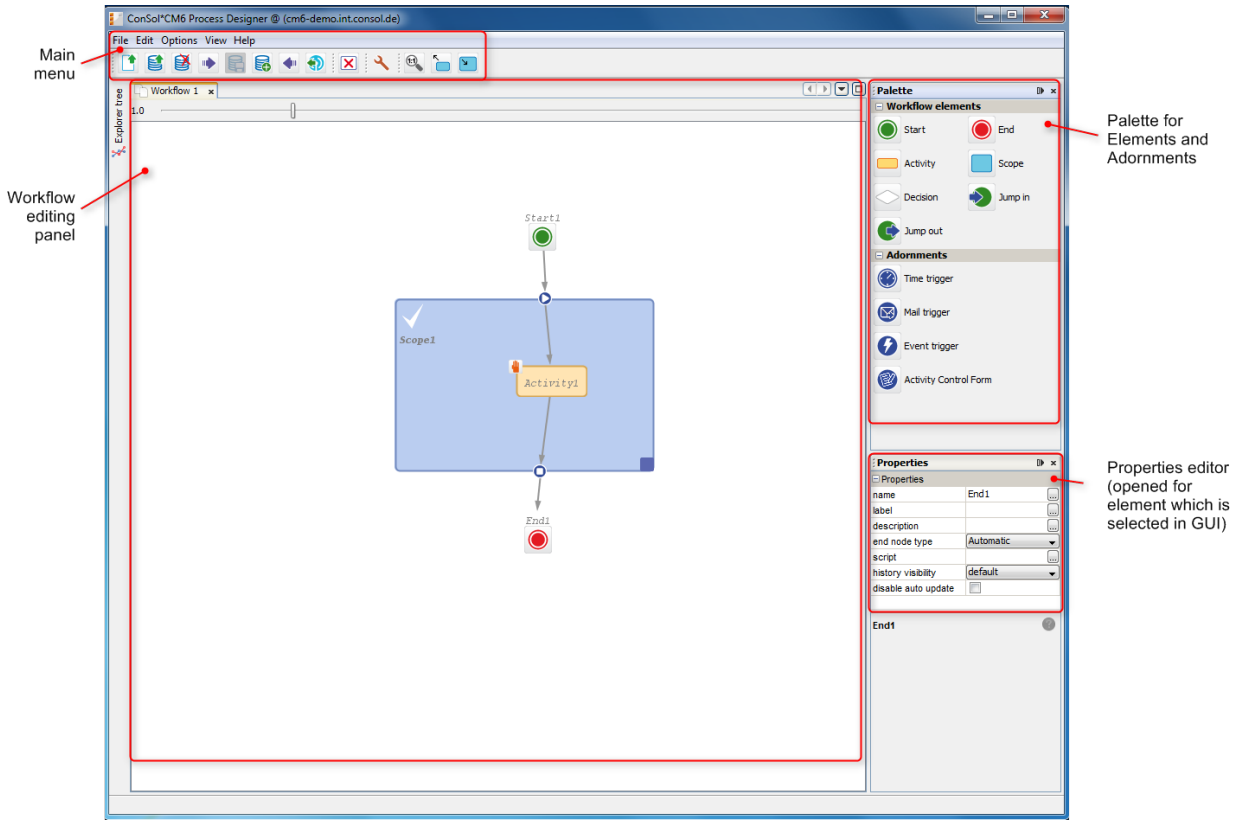


Fig. 1: ConSol*CM Process Designer - GUI Elements

Main Menu

The main menu contains the menu items as text entries and a menu icon list.

Menu main entry	Menu sub entry	Icon	Note
File			
	New ...		Start a new workflow.
	Load ...		Load a new workflow. Opens table with existing workflows, see section Loading a Workflow.
	Delete ...		Delete a workflow. Opens table with existing workflows, see section Deleting a Workflow.
	Import ...		Import a workflow from a (proprietary workflow format) file.
	Save ...		Save workflow (existing version).
	Save as new version		Save the workflow as a new version.
	Export		Export the workflow to a file. Opens file browser of the operation system. The workflow is saved in a proprietary workflow format (*.par*).

Menu main entry	Menu sub entry	Icon	Note
	Deploy		(Save as new version and) deploy the workflow, i.e. install the workflow in the system. The system might prompt you for a decision: • Keep position of the tickets in the process (see section Actions During Workflow Deployment). • Start at START node again.
	Log in		Log in to the Process Designer. Usually the login window is displayed directly after the start of the Process Designer. As login an account with administrator permissions or with the permissions to manage workflows (see *ConSol* CM Administrator Manual*, section *Role Administration*) is required.
	Log out		Log out. Does not exit the Process Designer.
	Exit		Exit/stop the Process Designer application.
Edit			
	Clear current tab		Delete the entire workflow, all elements in the main editing panel.
Options			

Menu main entry	Menu sub entry	Icon	Note
	Local configuration		Display pop-up window where you can select the display language of the Process Designer. All languages which have been configured for the system (see section *Configuration* in the *ConSol*CM Administrator Manual*) are available. The labels in the workflow in the main editing panel will be displayed in the selected language.
View			
	Normal zoom		Display workflow in default zoom (like at start of Process Designer).
	Expand all scopes		Display all scopes in the expanded version.
	Collapse all scopes		Display all scopes in the collapsed version.
	Hide/Show palette		Do (not) display palette in GUI.
	Hide/Show properties		Do (not) display Properties Editor in GUI.
	Hide/Show explorer		Do (not) display explorer (tree).
	Show ticket transfer history		Opens a pop-up window where the parameters for the ticket transfer during the deployment of a new workflow are displayed:

Menu main entry	Menu sub entry	Icon	Note
			• **Workflow name** Name of the workflow. • **Version** Version of the old workflow. • **Start time** Start of the transfer, will be the start time of the *Deploy* operation. • **End time** End of the transfer, after this time the new workflow will be in full operation. • **Transferred tickets** Number of tickets which have been transferred, i.e. which had to be touched by the system during workflow deployment. Should be identical to the sum of open tickets in all queues which use the workflow. • **Details** Additional information concerning the deployment with ticket transfer.

Menu main entry	Menu sub entry	Icon	Note
	IDE log		Opens the Log File Editor in the lower half of the screen and displays the user-specific log file of the Process Designer: *<USER_HOME_DIR>\.cmas\wfeditorR1\var\log*
Help			
	About		Display version information about the Process Designer and about the Java virtual machine it uses in the current configuration (this is the JVM of the browser plug-in).

Workflow Editing Panel

To design a workflow define the workflow elements using the graphical layout mode of the Process Designer and add the scripts to the elements where required.

A new element can be added to the workflow using drag-and-drop of the element from the palette.

A new element as successor of an existing element can also be created by using the context menu (right mouse click) of an existing element, e.g. for an activity (see the following figure). The new element and the connection to this element will be created.

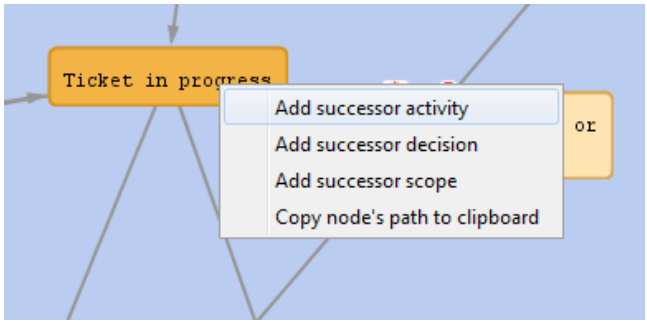


Fig. 2: ConSol*CM Process Designer - Context Menu for a Workflow Activity

A new connection between elements is established using the left mouse button while pressing the *CTRL* key and just drawing the line. If the connection goes from one scope to another, the scope entry and exit points are added automatically.

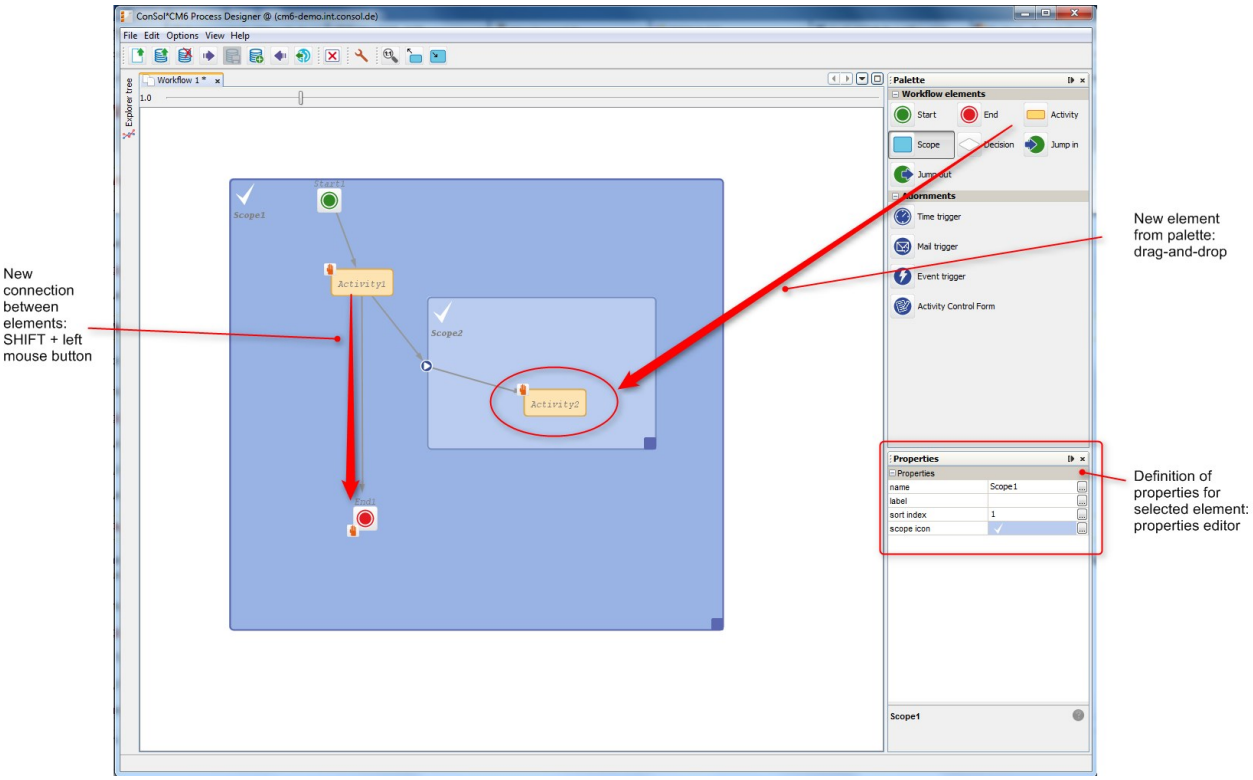


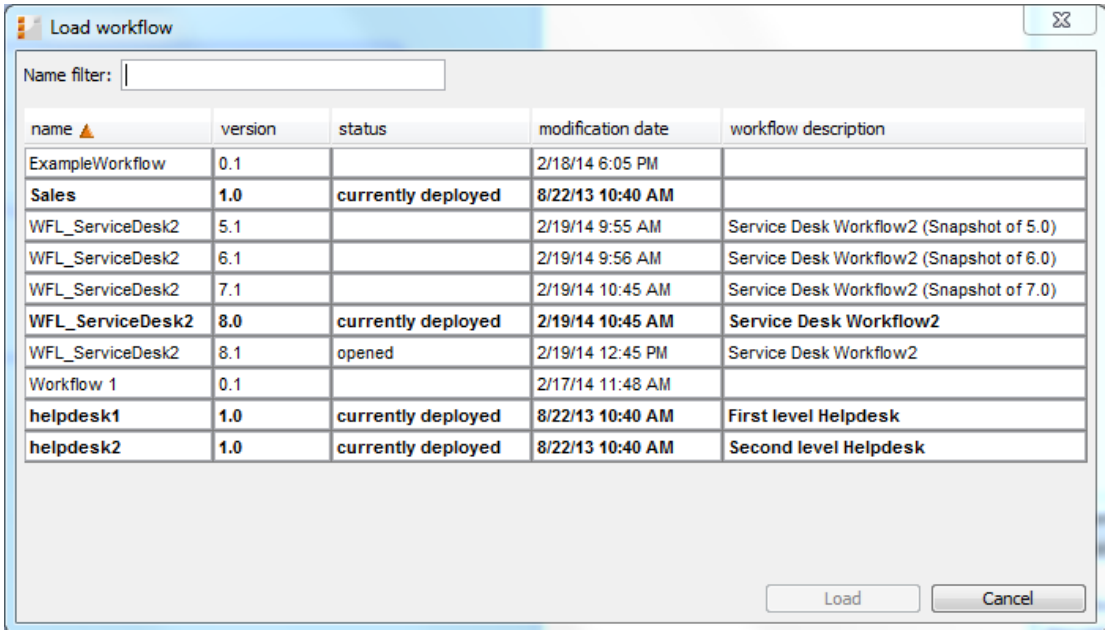
Fig. 3: ConSol*CM Process Designer - Adding New Elements and Connections

You might consider using a global scope for each workflow. Please refer to the Best Practices section for more information about how to design good workflows.

Loading and Deleting Workflows

Loading a Workflow

When you have selected the icon or menu item *Load*, a table with all available workflows is displayed.



name	version	status	modification date	workflow description
ExampleWorkflow	0.1		2/18/14 6:05 PM	
Sales	**1.0**	**currently deployed**	**8/22/13 10:40 AM**	
WFL_ServiceDesk2	5.1		2/19/14 9:55 AM	Service Desk Workflow2 (Snapshot of 5.0)
WFL_ServiceDesk2	6.1		2/19/14 9:56 AM	Service Desk Workflow2 (Snapshot of 6.0)
WFL_ServiceDesk2	7.1		2/19/14 10:45 AM	Service Desk Workflow2 (Snapshot of 7.0)
WFL_ServiceDesk2	**8.0**	**currently deployed**	**2/19/14 10:45 AM**	**Service Desk Workflow2**
WFL_ServiceDesk2	8.1	opened	2/19/14 12:45 PM	Service Desk Workflow2
Workflow 1	0.1		2/17/14 11:48 AM	
helpdesk1	**1.0**	**currently deployed**	**8/22/13 10:40 AM**	**First level Helpdesk**
helpdesk2	**1.0**	**currently deployed**	**8/22/13 10:40 AM**	**Second level Helpdesk**

Fig. 4: ConSol*CM Process Designer - Load a Workflow

The table can be sorted based on a column by clicking on the little triangle icon next to the column header.

The table contains the following columns:

- **name**
  The name of the workflow as set in the *name* property of the workflow (click into the white space around the global scope to see it for a workflow).
- **version**
  The version of the workflow. This is assigned automatically by the ConSol*CM system. When a scenario has been exported and is imported again, the numbering will start with 1.0 anew.
- **status**
  For older workflows this field is empty. The workflows which are deployed are described by *currently deployed*.
- **modification date**
  The date of the last modification (date when the workflow was saved) is indicated.
- **workflow description**
  The description which has been entered into the field *workflow description* (**not** *description*!).

To load a workflow, select it in the list and click *Load*. Only single selection is possible.

Deleting a Workflow

When you have selected the icon or menu item *Delete*, a table with all available workflows is displayed.

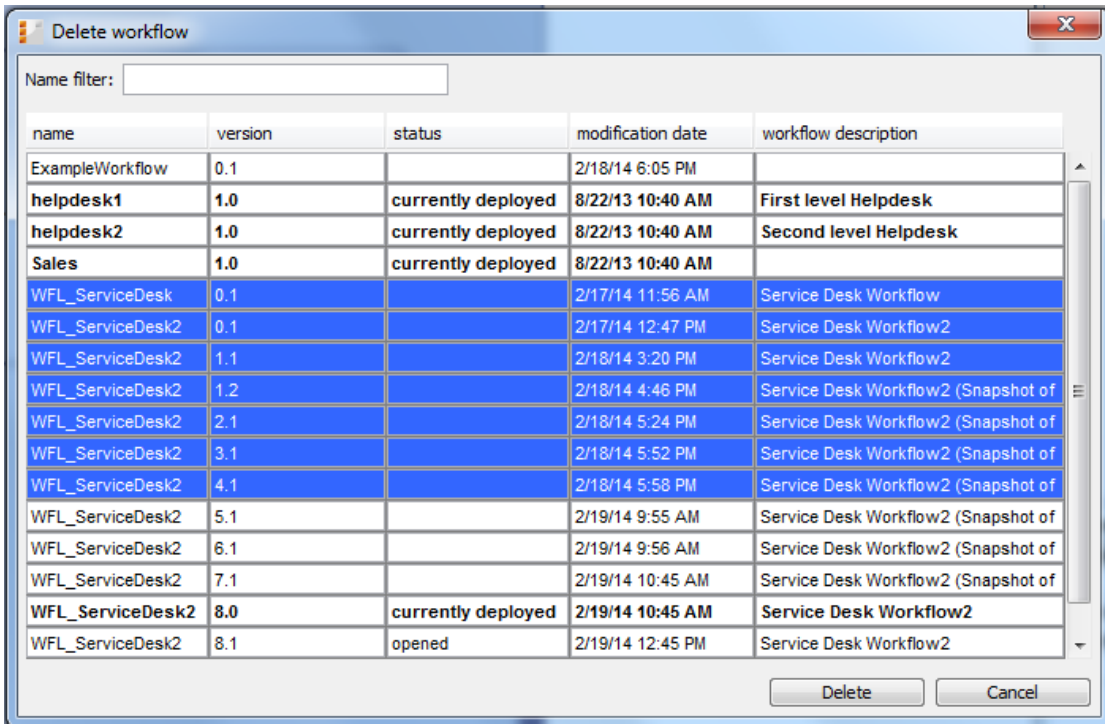


Fig. 5: ConSol*CM Process Designer - Delete a Workflow

The table can be sorted based on a column by clicking on the little triangle icon next to the column header.

The table contains the following columns:

- **name**
  The name of the workflow as set in the *name* property of the workflow (click into the white space around the global scope to see it for a workflow).

- **version**
  The version of the workflow. This is assigned automatically by the ConSol*CM system. When a scenario has been exported and is imported again, the numbering will start with 1.0 anew.
- **status**
  For older workflows this field is empty. The workflows which are deployed are described by *currently deployed*.
- **modification date**
  The date of the last modification (date when the workflow was saved) is indicated.
- **workflow description**
  The description which has been entered into the field *workflow description* (**not** *description*!).

To delete one or more workflow(s), select it/them in the list and click *Delete*. For every workflow you are prompted to confirm the deletion, so when you have marked a great number of workflows to delete and then you realize that you would like to keep one of them this is possible without canceling the entire operation.

Information:

You might want to delete all or almost all older workflows before exporting a scenario, because a great number of workflows increases the size of the scenario considerably. For export and import of scenarios, please refer to the respective section in the *ConSol*CM Administrator Manual*.

Palette for Elements and Adornments

As a default setting the palette is displayed in the top right corner. You can hide (and re-display) the palette using the main menu entry *Hide/Show palette* under *View*.

The palette contains two types of workflow components:

- elements
- adornments

Elements

Elements are basic components which form the workflow and represent the process logic.

Icon	Element	Note	Section
	Start node	Is set automatically, no other start node than the default start node can be added.	START Node
	End node	A workflow can contain one or more end nodes.	END Nodes
	Activity	The actions in the workflow, manual or automatic.	Activities

Icon	Element	Note	Section
	Scope	The highest hierarchy level in workflows	Scopes
	Decision	Decision node which has a *true* and a *false* ex it point.	Decision Nodes
	Jump-in	Entry point for tickets from other workflows/queues.	Jump-out and Jump-in Nodes
	Jump-out	Exit point for tickets. A target queue has to be defined. A target node can be defined but is optional.	Jump-out and Jump-in Nodes

Adornments

Adornments are objects which are assigned to a workflow activity or to a scope. Please see indicated sections for detailed explanations.

Icon	Adornment	Note	Section
	Time trigger	Can measure time intervals. Fires when the end of the interval has been reached. Can optionally use a business calendar.	Time Triggers
	Mail trigger	Fires when an e-mail for the ticket has come in.	Mail Triggers
	Business event trigger	Fires when an event has occurred. The type of event can be specified (e.g. change of engineer, change of priority).	Business Event Triggers
	ACF (Activity Control Form)	Defines the ACF which should be displayed when the activity is executed. ACFs are defined in the Admin-Tool.	Activity Control Forms (ACFs)

The Properties Editor (Example: Activity)

The Properties Editor is opened for the element which has been selected in the main editing panel and contains component-specific parameters. Some general parameters are present for all components, some are present only for a certain type of component.

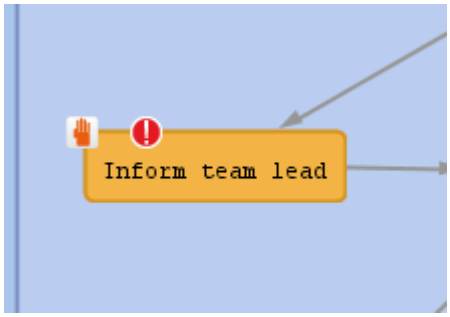


Fig. 6: ConSol*CM Process Designer - Selected Activity in Workflow

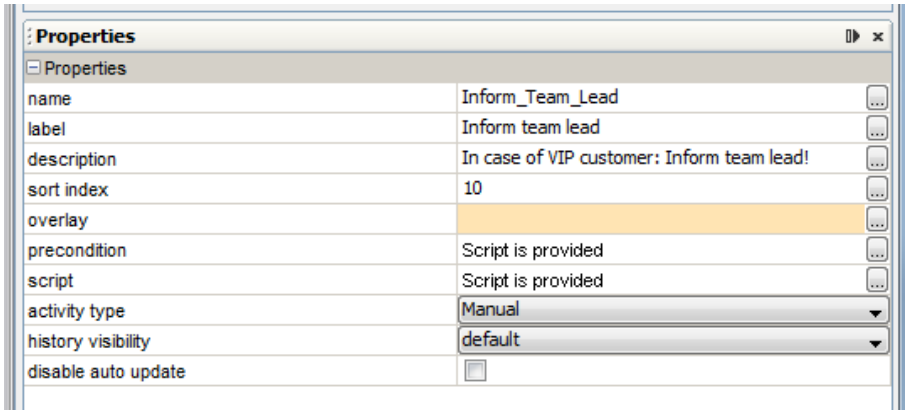


Fig. 7: ConSol*CM Process Designer - Properties Editor

Properties:

- **name**
  Mandatory. This is the technical object name. When an object is newly created, you can edit the label and the object name will be generated automatically from the label (umlauts are omitted). Afterwards, the object name is never changed automatically but can be edited manually. Allowed characters for names are:
  - letters (small or capital), but no umlauts
  - underline
  - numbers
- **label**
  The localized name of the element. All languages which have been configured for the system are available and can be filled. In the web browser of the engineer the description will be displayed according to the browser locale. If it is not available, the label will be displayed using the default locale.

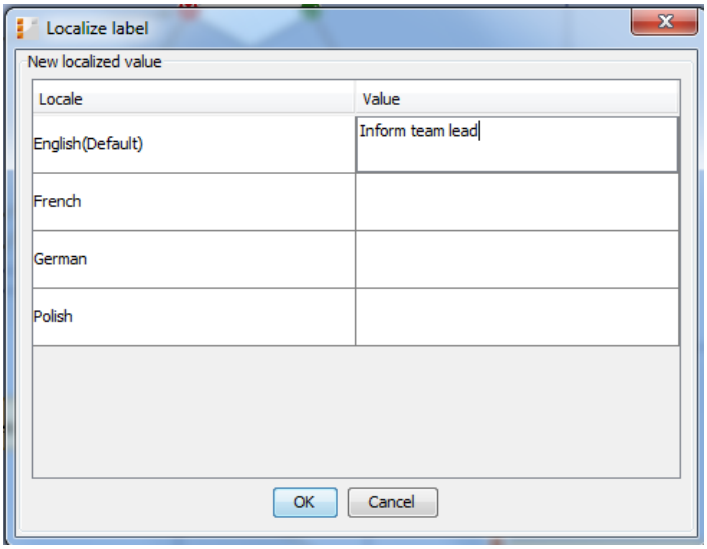


Fig. 8: ConSol*CM Process Designer - Localization for Labels

- **description**
  Optional. A localized text can be entered which will be displayed as mouse-over in the Web Client. This might help the engineer to understand what will happen when the respective workflow activity is executed.

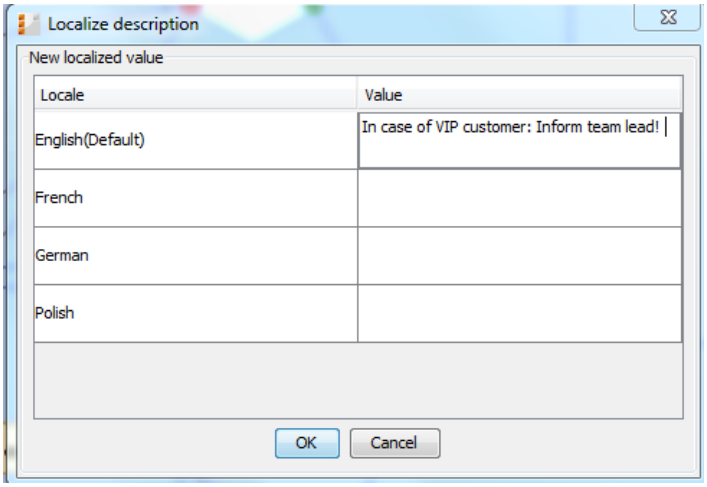


Fig. 9: ConSol*CM Process Designer - Localized Description of an Activity

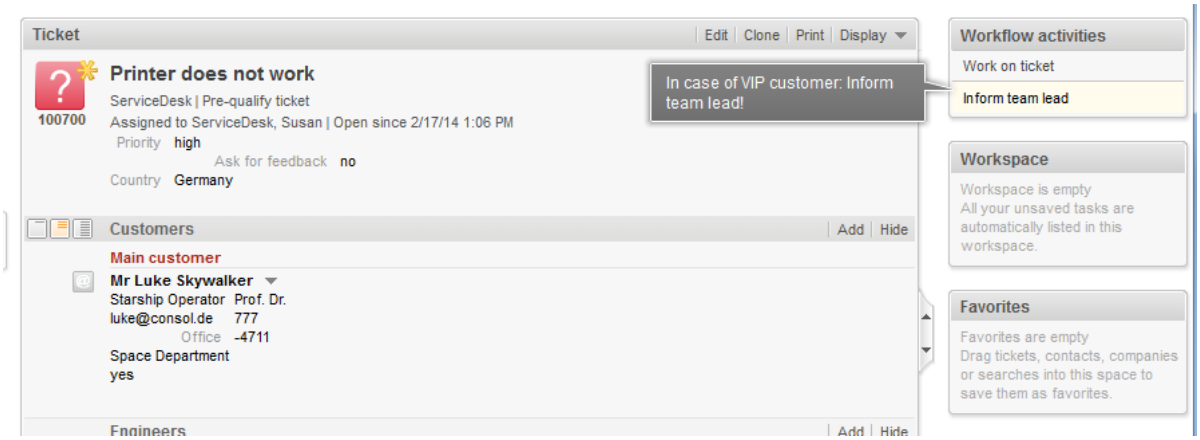


Fig. 10: ConSol*CM/Web Client - Localized Description of an Activity as Mouse-over

- **sort index**
  Defines:
  - **For activities:**
    The order of the activities in the list of *Workflow activities* in the Web Client. The higher the number the more at the bottom of the list the activity is offered in the Web Client.
  - **For scopes:**
    The order of the tickets in the ticket list (Web Client) in views. The higher the scope index the more at the top of the list the tickets are displayed.

- **overlay**
  Optional, for activities. Click into the orange space to define a standard ConSol*CM overlay or one that has already been uploaded. Click on the file explorer (...) button to open the file explorer of the operation system for an upload of a new icon. When the ticket passes through an activity the overlay is added to the ticket icon in the Web Client. As a maximum, three overlays can be attached to a ticket icon. This mechanism can be used for several purposes, some examples are:
  - **An escalation:**
    The ticket has been opened without any engineer taking care of it.
  - **An e-mail:**
    The ticket has received an e-mail.
  - **A note for the engineer:**
    E.g. another engineer has added a comment to *my* ticket.

Fig. 11: ConSol*CM Process Designer - Properties Editor: Standard Overlays and One Customer-Defined Overlay

Fig. 12: ConSol*CM/Web Client - Icons with Overlays

- **overlay range**
  Only displayed when an overlay has been set.
  - **activity**
    The overlay is attached only as long as the ticket stands behind the activity. As soon as the next activity is executed, the overlay is deleted from the ticket icon.
  - **scope**
    The overlay is deleted when the ticket leaves the scope.
  - **process**
    Once the overlay has been attached to the ticket icon, it stays there for the rest of the process.
  - **next overlay**
    The overlay is attached to the ticket icon as long as no new overlay appears. In that case, only the new one is attached, the old one is deleted.
- **precondition**
  Optional, for activities. A script can be entered using the Script Editor (see section The Script Editor) which has to return *true* or *false*. The script is executed when the previous activity has been performed, i.e. when it becomes possible to display the activity with the precondition. In case *true* is returned, the activity is displayed, in case *false* is returned, the activity is not displayed. An activity which has a precondition is marked by the icon *exclamation mark/precondition* .
- **script**
  Optional, for activities. A script can be entered using the Script Editor (see section The Script Editor) which is executed when the ticket enters the activity.

- **activity type**
  Mandatory, for activities. *Manual* or *Automatic* has to be selected. A manual activity is displayed in the Web Client and has to be explicitly selected/executed by an engineer. In the Process Designer it is marked by the icon *hand/manual* .
  An automatic activity is executed without any engineer interaction. For a detailed explanation of the ConSol*CM process logic, please see section Process Logic.
- **history visibility**
  Mandatory, but default value has been set (default). The value defines the display levels of the Web Client GUI where the action (that the activity has been performed) should be displayed:
  - 2nd level and 3rd level
  - only 3rd level
  - on every level
  - default
    This refers to the value defined in the Admin-Tool under *Ticket History* for the activity configuration. Depending on the type of activity, one of the following parameters is used:
    - Manual activity or activity with overlay executed
    - Activity executed after escalation
    - Automatic activity executed

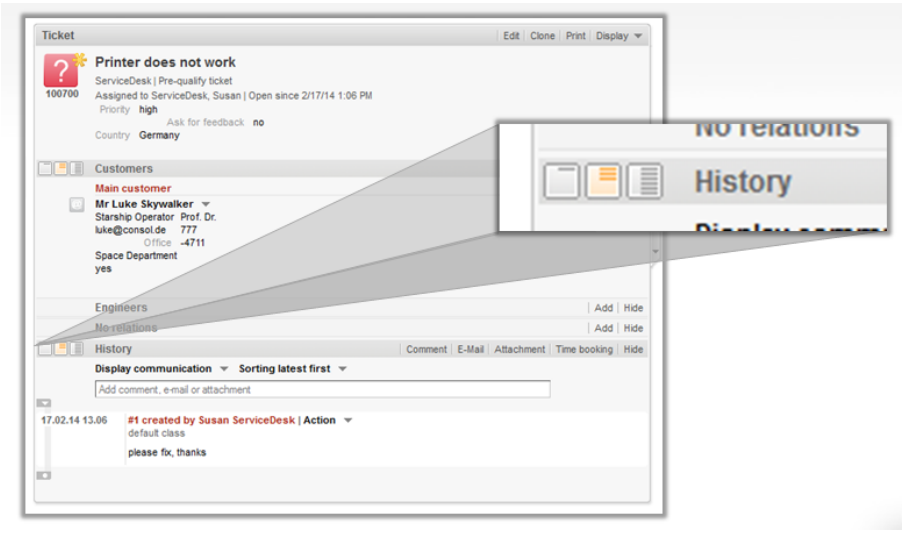


Fig. 13: ConSol*CM/Web Client - Display Levels in Ticket History

- **disable auto update**
  Defines ticket behavior of the ticket when an event has been fired or executed. Usually, after an event, a ticket update operation is performed automatically. In case a chain of events is used you should avoid triggering a ticket update operation after every single event. To avoid this, set *disable auto update* to *true* in all events except for the last one. Then, only after the last event, the ticket is updated.

3.2.2 The Script Editor

You use the Script Editor in the Process Designer to write Groovy scripts (i.e. pure Groovy and Java code is accepted). For explanations, recommendations, and examples concerning workflow programming using scripts, please see section Workflow Programming.

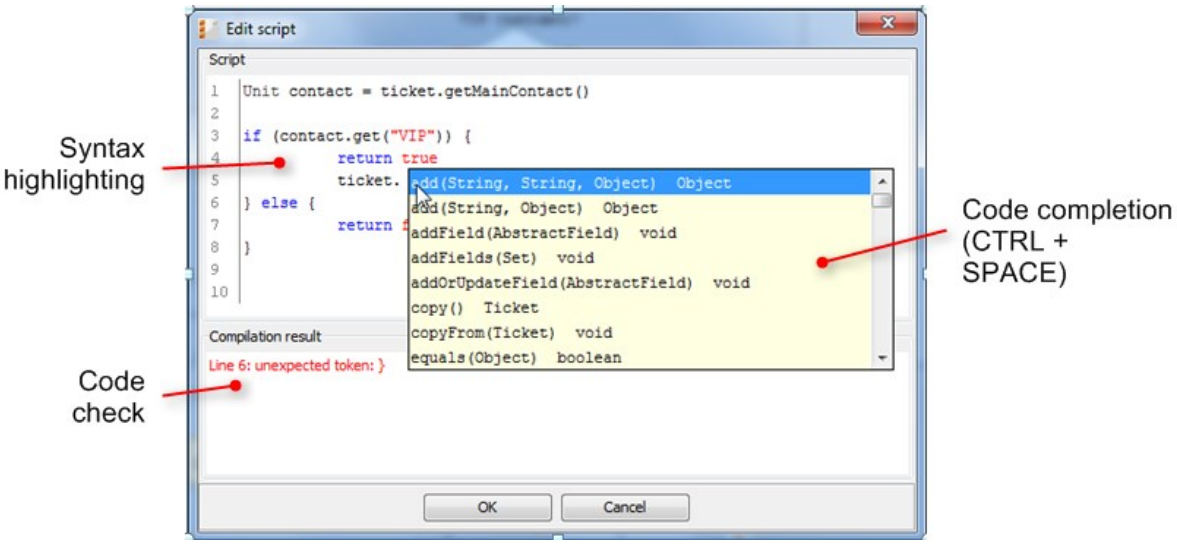


Fig. 14: ConSol*CM Process Designer - Script Editor

The Script Editor provides the following features:

- **Syntax highlighting**
  Groovy code is highlighted according to key words.
- **Code completion**
  When you have entered the name of an object and the dot, the possible methods are suggested. Press *CTRL + SPACE* to activate code completion.
- **Code check**
  The entered code is controlled according to the correct use of general syntax and methods. The error code is displayed in the *Compilation result* panel.

4 ConSol*CM Process Designer Manual - Components of ConSol*CM Workflows

4.1 Components of ConSol*CM Workflows

4.1.1 Introduction

You can work with various types of workflow components to build the workflows for your ConSol*CM system. The palette in the Process Designer offers all elements and adornments, see section Palette for Elements and Adornments for an overview.

In the following chapters, all workflow elements and adornments will be explained in detail.

Workflow Element	Explanation
START Node	The first node in a workflow, see section START Node.
END Node(s)	One or more end nodes of the process. The ticket is closed. See section END Nodes.
Scopes	Realms of a process, see section Scopes.
Activities	The steps of a process. Can be automatic or manual, see section Activities.
Decision Nodes	Workflow element which represents a *true/false* de cision, see section Decision Nodes.
Adornments	Elements to control the process flow: triggers and activity control forms. See section Adornments (Triggers and ACFs).
Jump-out and Jump-in Nodes	Elements which connect workflows, see section Ju mp-out and Jump-in Nodes.

4.2 Workflow Components: START Node

Every workflow contains exactly one START node. When you create a new workflow the start node is added automatically, you do not have to add it yourself.

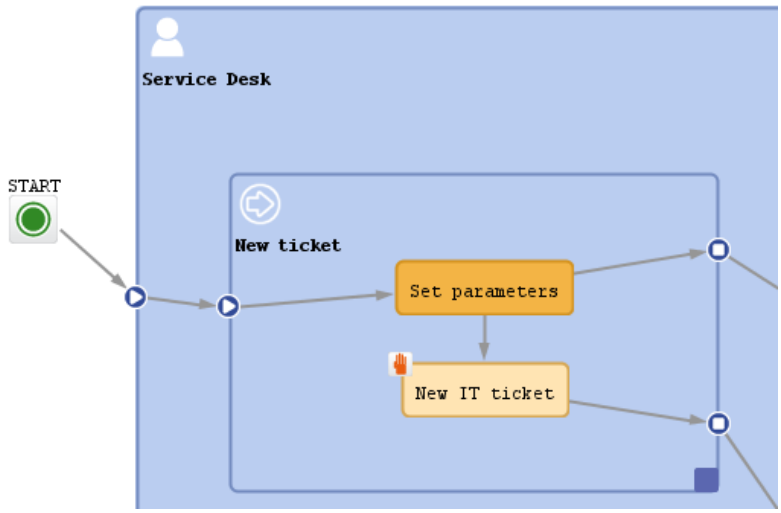


Fig. 1: ConSol*CM Process Designer - Start Node

The start node does not have any scripts and cannot be configured in any way.

When a ticket enters the workflow and no specific entry point has been defined, the ticket passes through the start node.

Best Practices:

The start node should not be positioned within the global scope. See also section Best Practices.

4.2.1 Properties of a Start Node

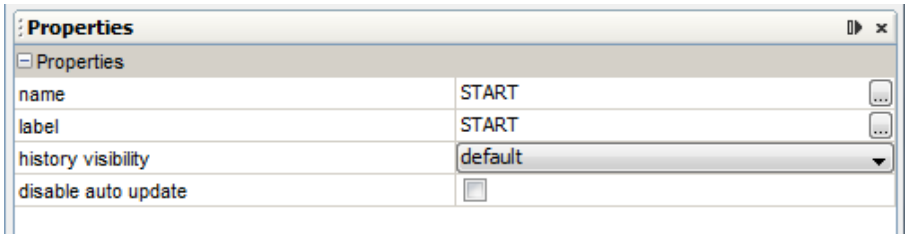


Fig. 2: ConSol*CM Process Designer - Start Node Properties

Properties:

- **name**
  Technical object name.
- **label**
  Localized name which will be displayed on the GUI.
- **history visibility**
  See section history visibility.
- **disable auto update**
  See section disable auto update.

4.3 Workflow Components: END Nodes

A workflow in ConSol*CM can have one or more END nodes.

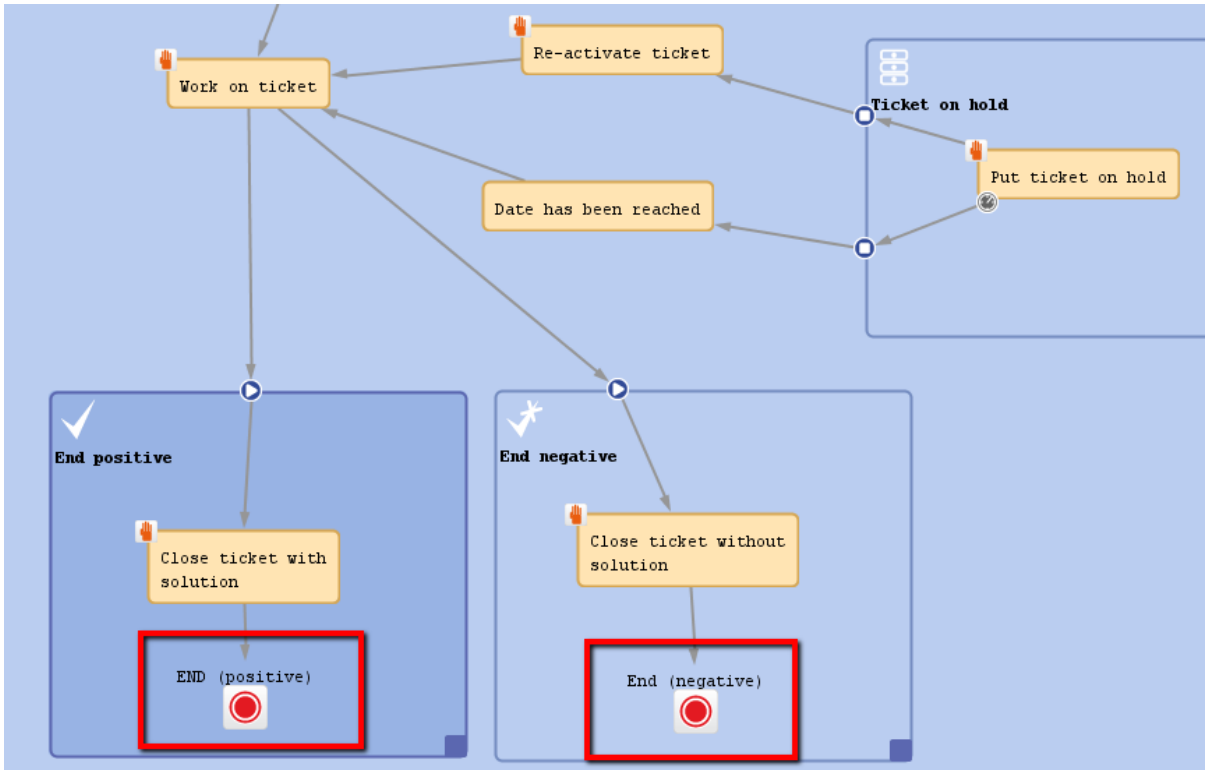


Fig. 1: ConSol*CM Process Designer - End Nodes

An end node represents the closing of the ticket, i.e. when a ticket is passed to an end node it is closed in a technical sense. No engineer can edit the ticket anymore. The ticket can be re-opened by an administrator using the *Ticket Administration* in the Admin-Tool, please see the respective section in the *ConSol*CM Administrator Manual* for detailed information.

However, assuming engineers have the required access permissions, they can still read the ticket. This is an important basis for the use of all ConSol*CM tickets of a system as knowledge base.

The passing of the ticket to the end node can be a manual or an automatic action. In the figure above, the end nodes are automatic nodes, i.e. the ticket passes to this node when the previous activity has been performed.

As a minimum a workflow has to contain one end node, because there has to be a way to close the ticket.

You might want to create more than one end node. This can be helpful when you create reports, e.g. to distinguish between positive and negative endings.

An end node might have a script, i.e. before the ticket is closed, a script can be executed.

Best Practices:

Sometimes, it might be required to set a ticket to *closed*, *completed*, or *done* from an engineer's point of view, i.e. to set a ticket to a *preliminary END*. After a while, if there are no more questions or remarks from the customer, the ticket should be closed automatically. You can achieve this by setting a time trigger to an end activity and letting the ticket go to the end node automatically after the defined time (see following figure).

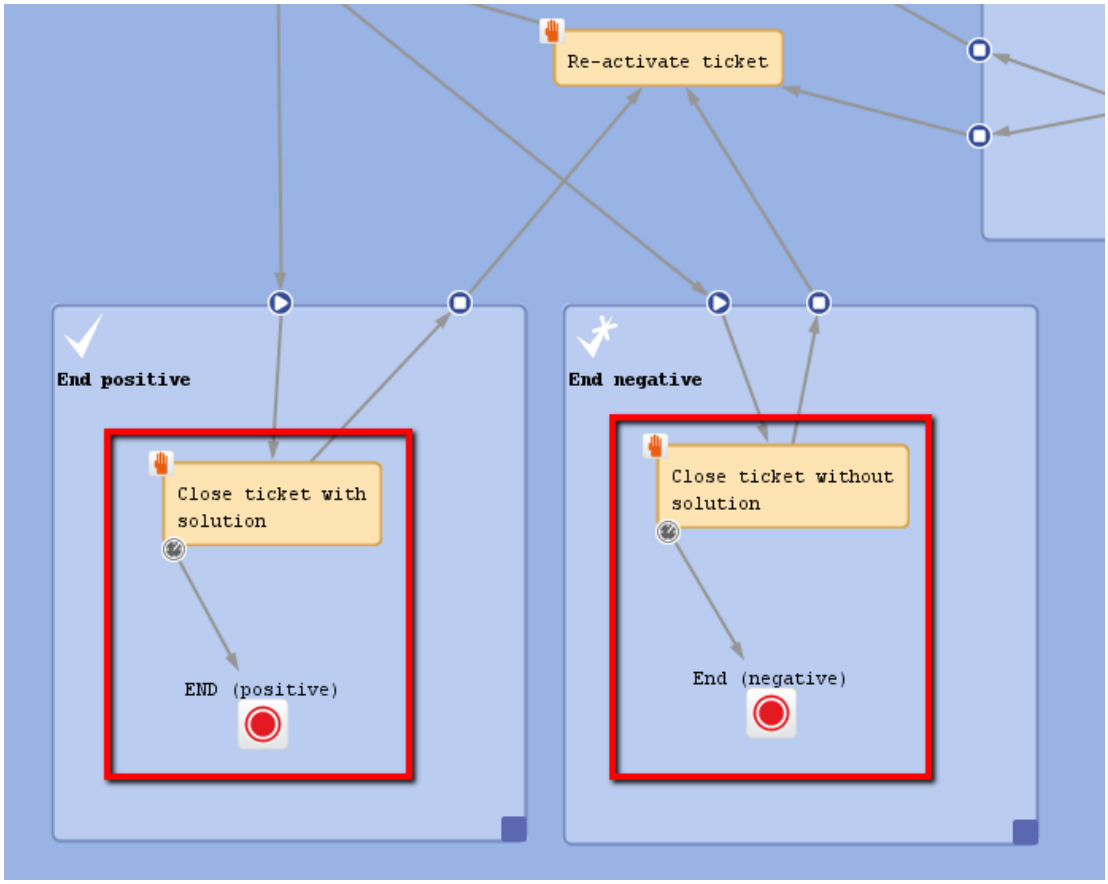


Fig. 2: ConSol*CM Process Designer - End Nodes Reached via Time Trigger

4.3.1 Properties of an End Node

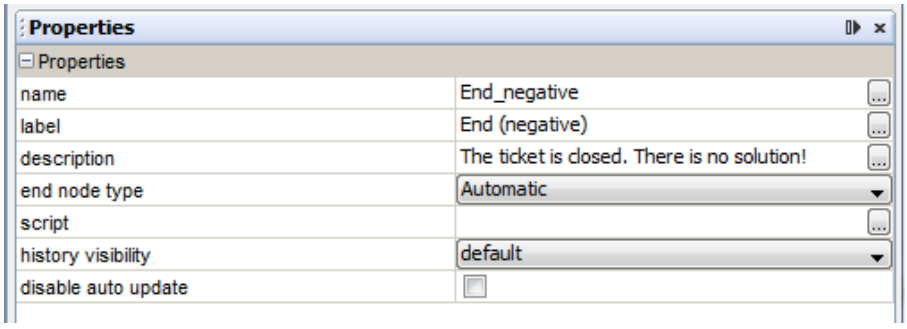


Fig. 3: ConSol*CM Process Designer - End Node Properties

Properties:

- **name**
  Technical object name.
- **label**
  Localized name which will be displayed on the GUI.
- **description**
  Description which is displayed as mouse-over text.
- **end node type**
  Automatic/Manual.
- **script**
  Here, a script which should be executed when the ticket enters the end node, i.e. before the ticket is closed, can be edited.
- **history visibility**
  See section history visibility.
- **disable auto update**
  See section disable auto update.

4.4 Workflow Components: Scopes

4.4.1 Introduction to Scopes

When a ticket passes through a process there are several positions it has to pass, all in a pre-defined order. For example, in a service desk environment, the ticket comes in as *new ticket*, then it has to be pre-qualified (in our example: are there any SLAs which have to be taken into consideration, is it a VIP customer?). Subsequently, the engineer can work on the ticket and might put it on hold for a while. Then the ticket should be closed, either as *positive, with solution* or *negative, without solution*. Those major steps of the process are represented as scopes in ConSol*CM workflows. See the following figure for an example workflow.

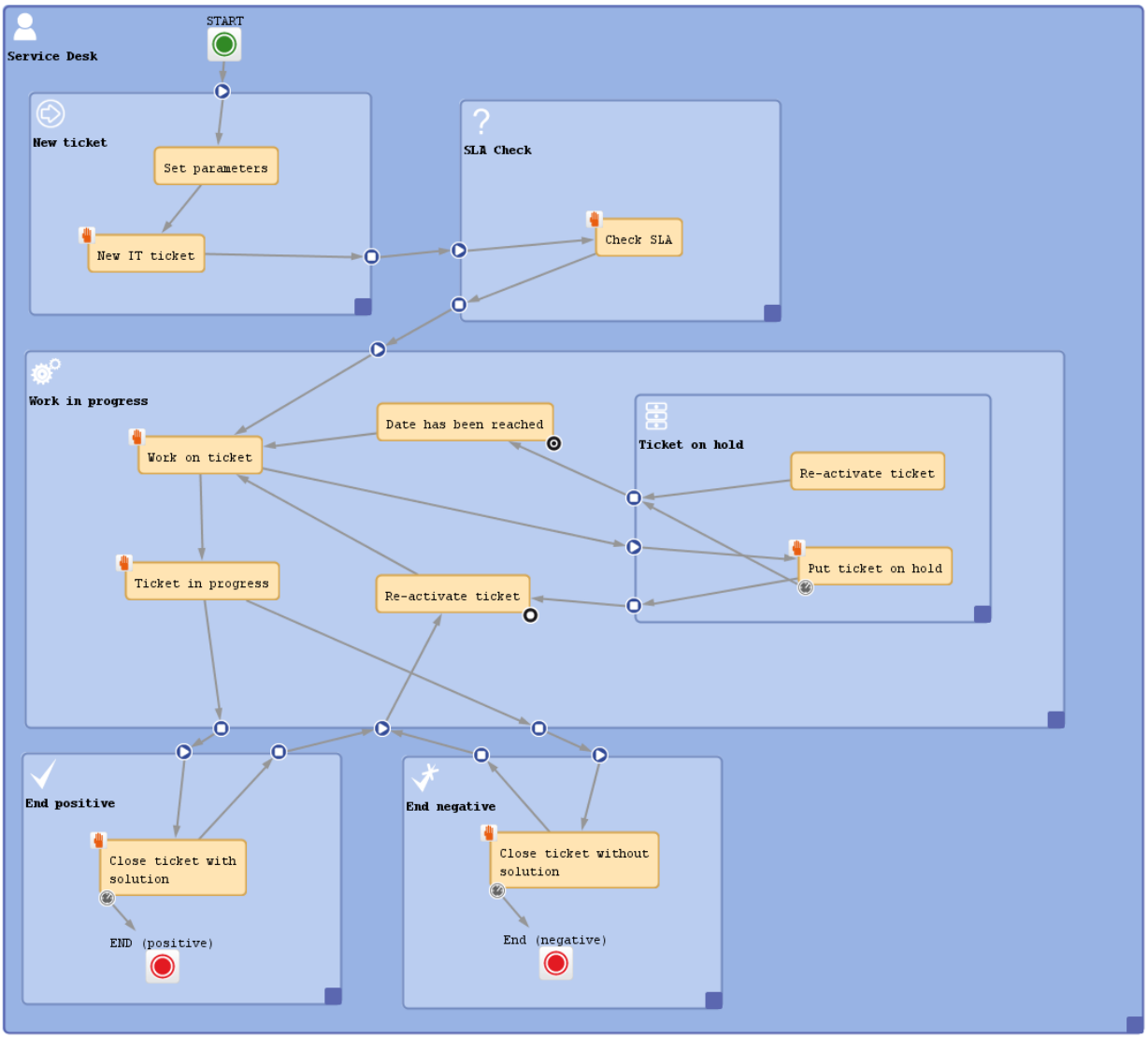


Fig. 1: ConSol*CM Process Designer - Workflow with Scopes

Within each process step, there can be one or more activities, e.g. during pre-qualification, first the VIP customer check is performed, then the SLA is checked. Those activities are described in detail in the section Activities. Here, only scopes are explained.

A scope can be part of another scope or - seen from the opposite point of view - a scope can contain sub-scopes.

A scope can have various types of triggers, e.g. a mail trigger fires whenever an e-mail to a ticket, which is currently in the scope, has been received. Please see sections Mail Triggers, Time Triggers, and Business Event Triggers for details.

4.4.2 Defining a New Scope

In order to define a new scope, i.e. to add a new scope to the workflow, grab the scope icon in the palette and drag-and-drop it to the workflow at the position where you would like to locate it. Activate it with a double-click. Then you can add new activities or other elements or drag existing activities/elements into the scope. When you connect elements by drawing arrows, the entry and exit points of a scope are defined automatically.

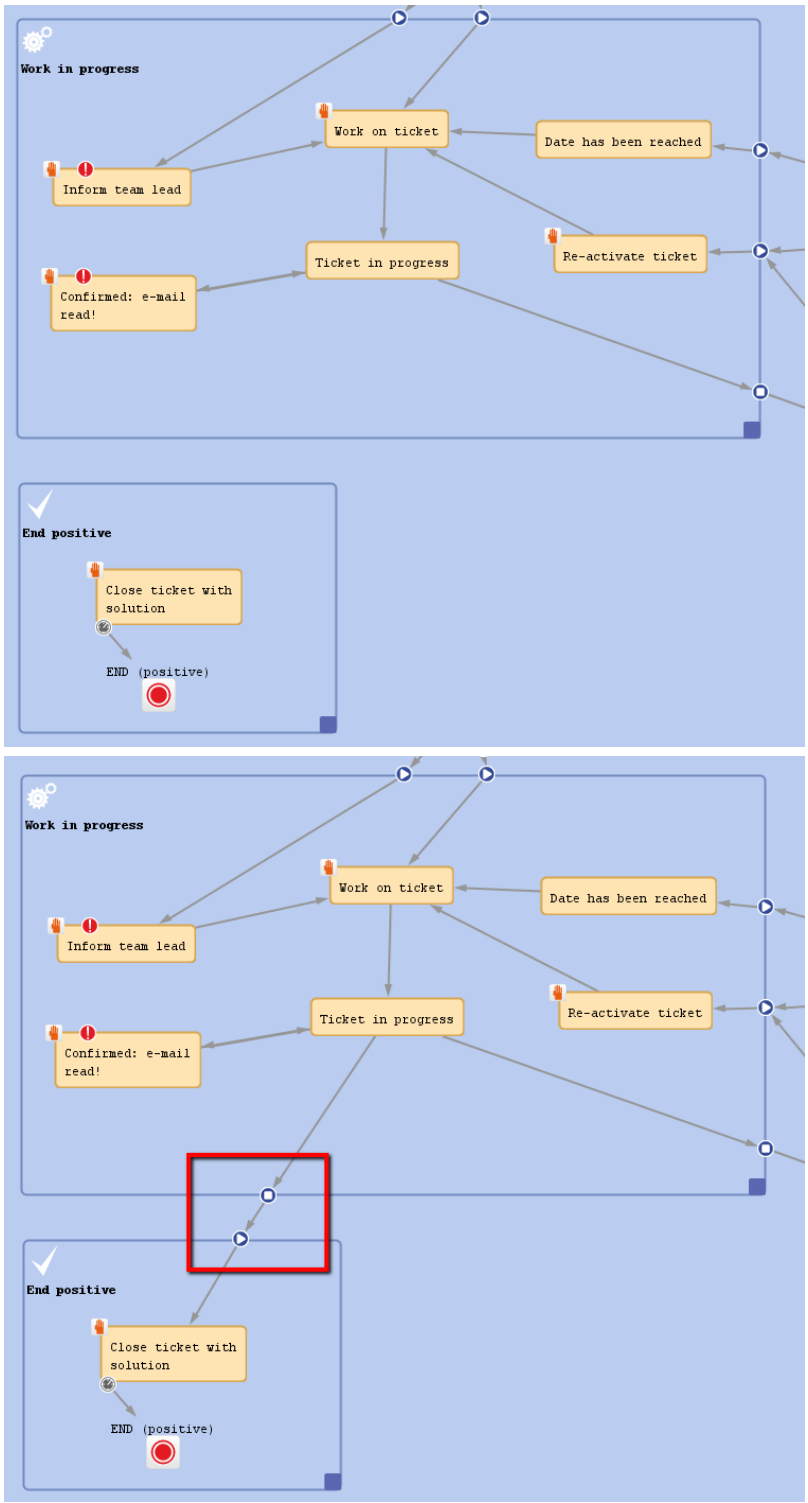


Fig. 2: ConSol*CM Process Designer - Automatically Generated Exit and Entry Points in Scopes

When you have defined/added the new scope you can define the scope's properties, see next section.

4.4.3 Properties of a Scope

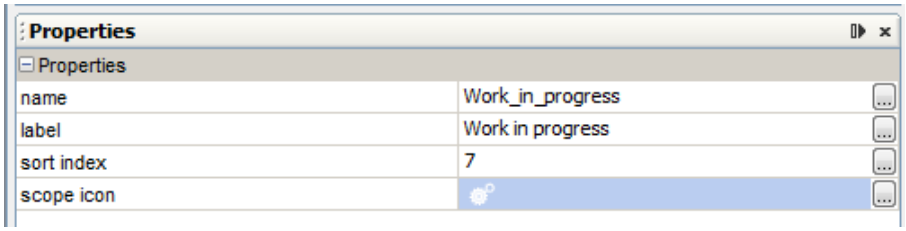


Fig. 3: ConSol*CM Process Designer - Scope Properties

The following properties can be defined for a scope:

- **name**
  The technical object name.
- **label**
  The localized name which will be displayed in the Web Client GUI.
- **sort index**
  Defines the position of tickets of this scope in a view (in case the view comprises more than one scope)**.**
- **scope icon**
  The icon which is displayed as scope icon in the Web Client GUI (see following figure). Click into the blue area to pick one of the ConSol*CM standard icons or use the file browser (...) to load an icon from the file system.

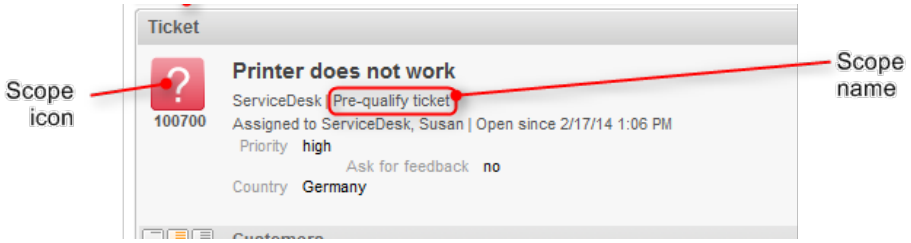


Fig. 4: ConSol*CM/Web Client - Scope Icon

> **Attention:**
>
> Please keep in mind that the icon is merged with the ticket color. So (in case you would like to upload your own icons) transparent images should be used for ticket icons. Otherwise, the background color might be lost or only be seen in a small border around the icon.

4.4.4 Scopes and Views

Views, i.e. the selection criteria for the ticket list(s), are defined based on scopes. For a detailed explanation of views and view definition, please refer to the respective section in the *ConSol*CM Administrator Manual*.

In the present context, i.e. when you define scopes in the workflow, it is important to keep in mind which views might be required later on. For example, the mechanism of *new, active*, and *pending* tickets is based entirely on the scope and view definition:

- **View: New**
  All new tickets in the scope *new*.
- **View: Active**
  All active tickets, i.e. tickets which **are not** in a scope *on hold*, *resubmission*, or the like.
- **View: Pending**
  All tickets which **are** in a scope *on hold*, *resubmission*, or the like.

That means, whenever a view is required to display only a certain sort of tickets, a scope has to be defined.

Attention:

We strongly recommend **not** to define views which contain closed tickets!

The number of closed tickets will grow considerably during work with the application. Therefore, the view of closed tickets would always reach the maximum number of tickets allowed for a view (which can be defined using a system property). This can have negative influence on the GUI performance and in most cases the desired tickets will not even be among the first 50 or 100 tickets.

Conclusion: A view of closed tickets does not help and might decrease the speed of the system for the engineers. Only in test environments, a view for closed tickets might be an option.

4.5 Workflow Components: Activities

4.5.1 Introduction to Activities

An activity represents an action in a workflow. An activity is located within a scope and is of one of the following types:

- manual
- automatic

A **manual** activity has to be performed by a manual action of the engineer using the Web Client GUI. The activity is displayed as *Workflow activity* in the Web Client (provided at least one of the roles of the engineer has the *Execute* permission (please refer to the *ConSol*CM Administrator Manual*, section *Role Administration*, for a detailed explanation). In the Process Designer, the activity is marked by the *hand/manual* icon  .

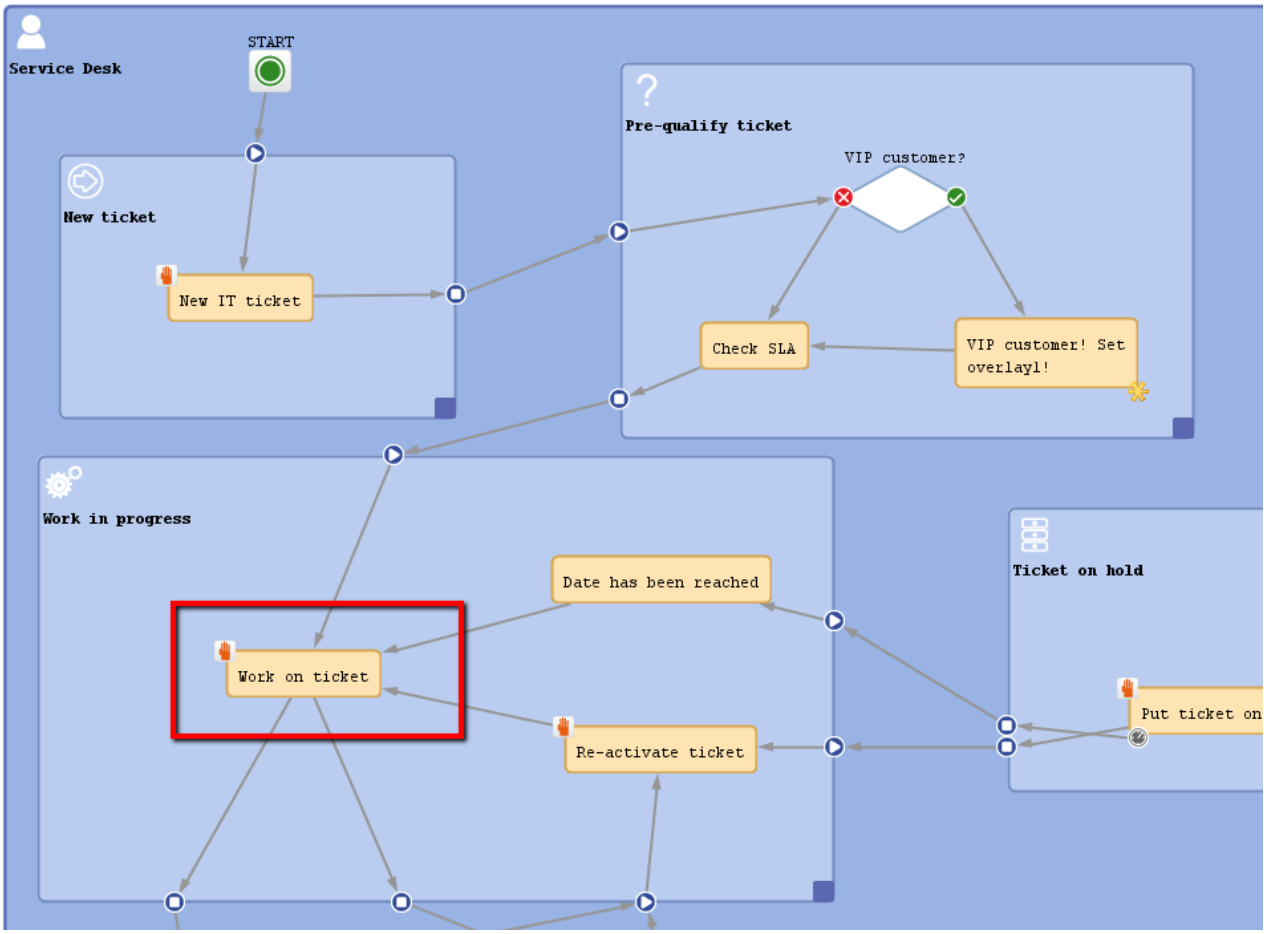


Fig. 1: ConSol*CM Process Designer - Manual Activity in Workflow

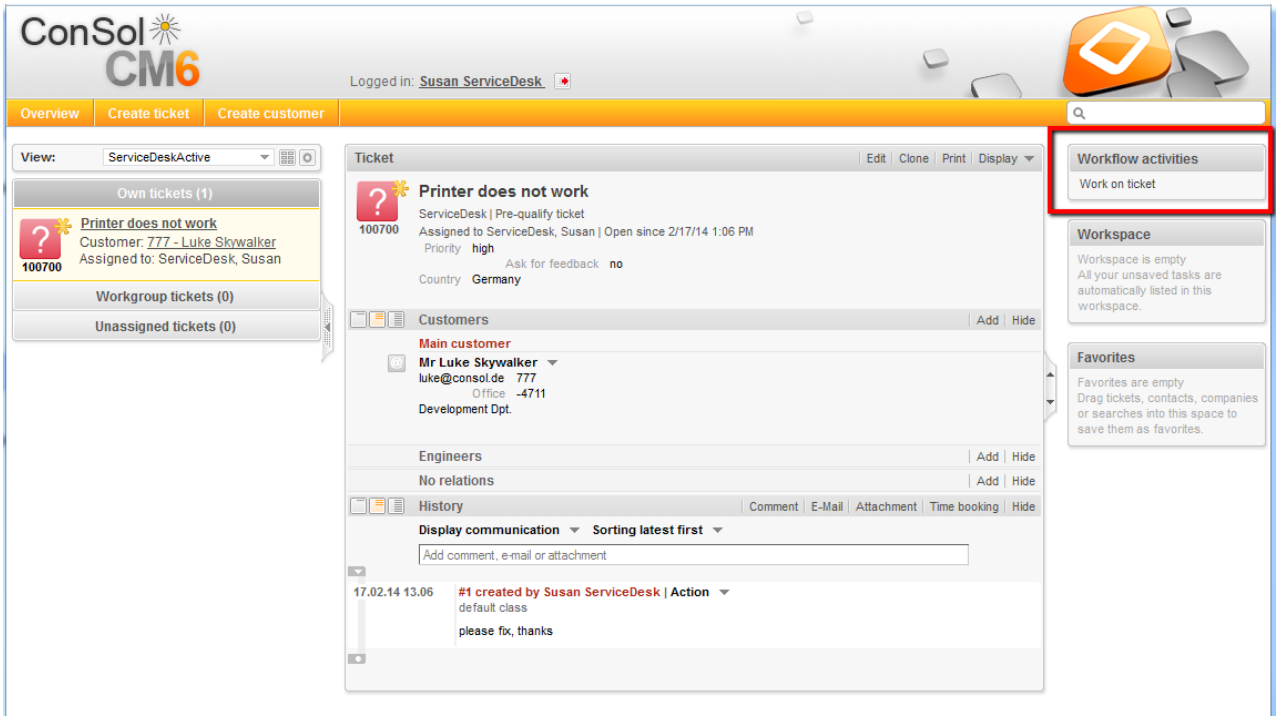


Fig. 2: ConSol*CM/Web Client - Manual Activity

An **automatic** activity is performed automatically by the system and is not displayed in the Web Client. In the Process Designer, an automatic activity is not marked by any special icon.

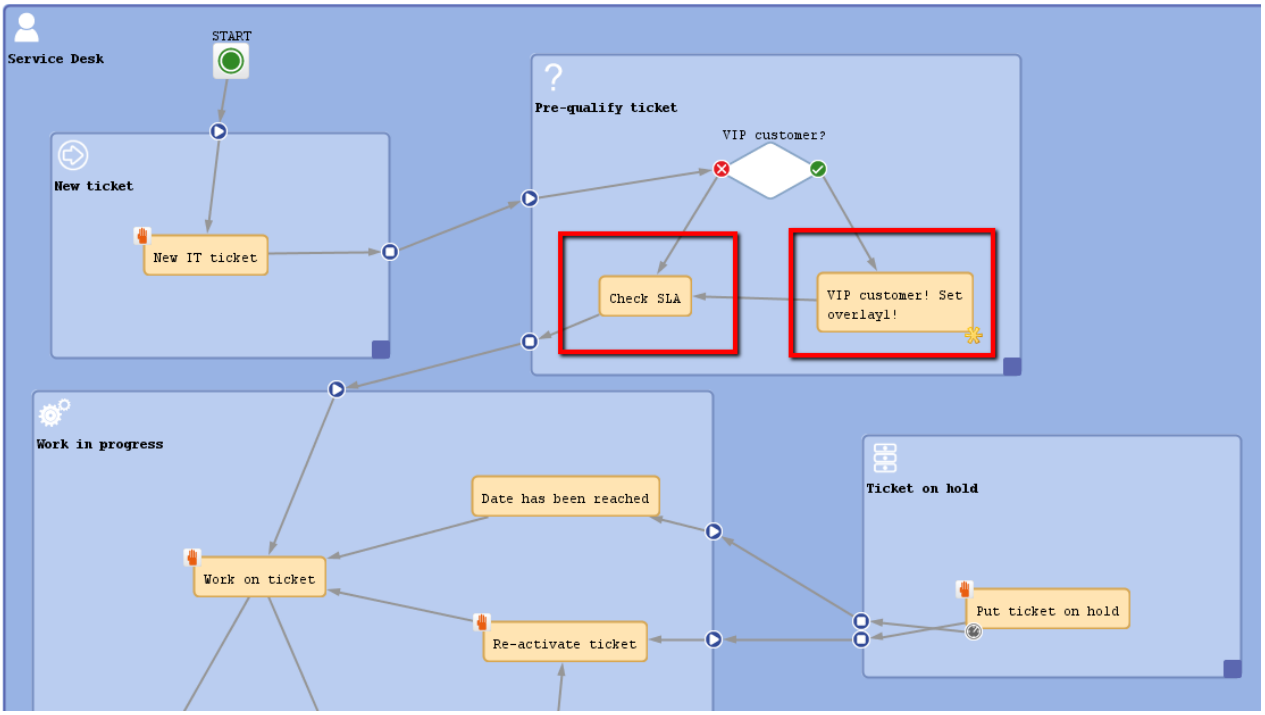


Fig. 3: ConSol*CM Process Designer - Automatic Activities

4.5.2 Properties of an Activity

In order to display and edit the properties of an activity, mark the activity in the Process Designer.

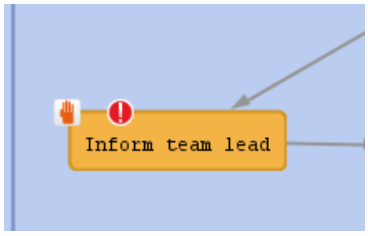


Fig. 4: ConSol*CM Process Designer - Activity

The Properties Editor will be opened for this activity.

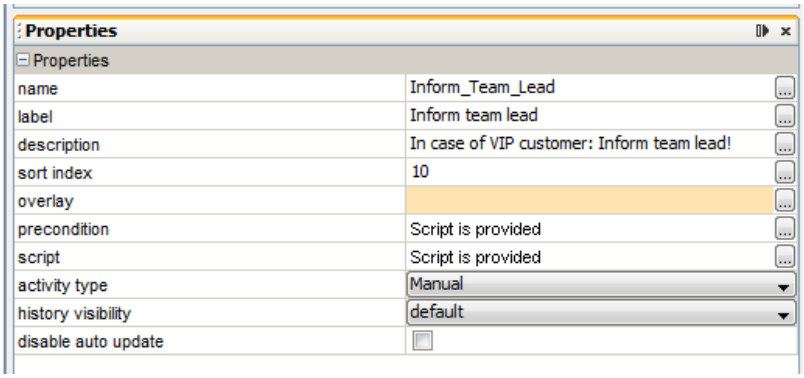


Fig. 5: ConSol*CM Process Designer - Properties of an Activity

An activity can have the following properties:

- **name**
  Mandatory, technical object name.
- **label**
  Optional (if not set, the technical name is used). Localized name which will be displayed in the Web Client. The language which is configured in the web browser is used.
- **description**
  Optional. Will be displayed as mouse-over in the Web Client.
- **sort index**
  Defines the order of the activities in the Web Client.
- **overlay**
  Optional. Click into the orange space to load standard ConSol*CM overlay icons or use the file browser (...) to upload another icon from the file system.
- **overlay range**
  Only displayed when an overlay has been set:
  - **activity**
    The overlay is attached only as long as the ticket stands behind the activity. As soon as the next activity is executed, the overlay is deleted from the ticket icon.
  - **scope**
    The overlay is deleted when the ticket leaves the scope.
  - **process**
    Once the overlay has been attached to the ticket icon, it stays there for the rest of the process.
  - **next overlay**
    The overlay is attached to the ticket icon as long as no new overlay appears. In that case, only the new one is attached, the old one is deleted.

- **precondition**
  Optional. A script can be entered which is executed when the activity should be offered in the Web Client GUI. The script has to return *true* or *false*. If a precondition has been defined for an activity, the activity is marked by the *exclamation mark/precondition* icon (see figure above).
  - Return value is **true**.
    The activity is displayed. If it is a manual activity it can be selected/performed by the engineer in the Web Client GUI.
  - Return value is **false**.
    The activity is not displayed in the Web Client GUI

> **Attention:**
>
> CM version 6.9 and higher:
> When you work with data object group fields, i.e. with data fields that contain customer data, please keep in mind that it might be required to consider the data models of different customer groups in case a workflow is used for queues which have been assigned to more than one customer group!

- **script**
  Optional. A script can be defined which is executed when the ticket passes through the activity.
- **activity type**
  Mandatory. Either automatic or manual has to be selected. In case it is a manual activity, the activity is marked with the *hand/manual* icon in the Process Designer GUI.
- **history visibility**
  See section history visibility.
- **disable auto update**
  See section disable auto update.

4.5.3 Process Logic of Activities

This is the process logic of activities:

1. When a ticket has passed through an activity it always waits behind this activity (and not before the next one!).
2. When a ticket has passed through an activity it checks if there is an automatic activity. If yes, the ticket passes through this automatic activity as well.
3. The ticket passes automatically through (automatic) activities as long as there are new automatic activities. It comes to a halt as soon as there is/are one or more manual activities where engineer interaction is required.
4. If one or more of the following manual activities have a precondition script, this script is executed in order to decide if the activity has to be displayed in the Web Client GUI or not.
5. If the engineer selects the activity in the Web Client GUI, the script of the activity is executed.
6. If there is a *postActivityScript*, this script is executed immediately after the execution of the activity script.

7. The ticket waits behind the manual activity. If the following activity is located in a new scope, the ticket will not enter the new scope. It always waits behind the old activity and not before the new one!

> **Attention:**
>
> A ticket always waits behind the last activity which has been executed and not before the new one!!

4.5.4 Examples for Activities

Example 1: Precondition for Displaying Activity "Inform team lead"

In case the ticket has been opened by a *VIP* contact, i.e. a contact where the boolean field *vip* is *true*, the team lead should be informed. If it is no *VIP*, the activity should not be offered. The custom field *vip* which is part of the customer data model is checked for this purpose.

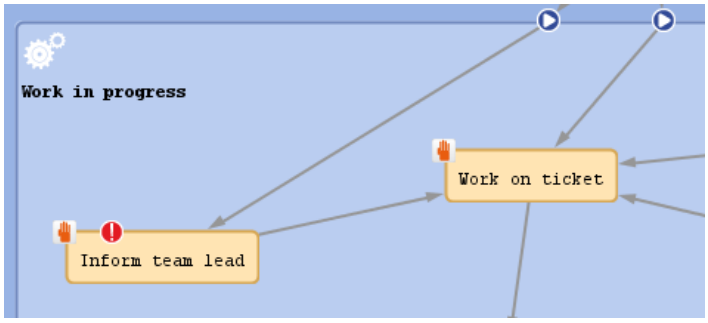


Fig. 6: ConSol*CM Process Designer - Workflow Activities (One with Precondition Script)

Precondition script: Workflow used only for queues of one customer group

```
// Get the main contact of the ticket. The unit object (can be a customer or a company) is
provided;
// here it has to be a customer, i.e. a contact:

Unit contact = ticket.getMainContact()

// Check the custom field "vip" of the main contact. (see next image)
// If it is set to true, return true, i.e. the condition is TRUE.
// Else return false, i.e. the condition is FALSE:

if (contact.get("vip")) {
    return true
} else {
    return false
}
```

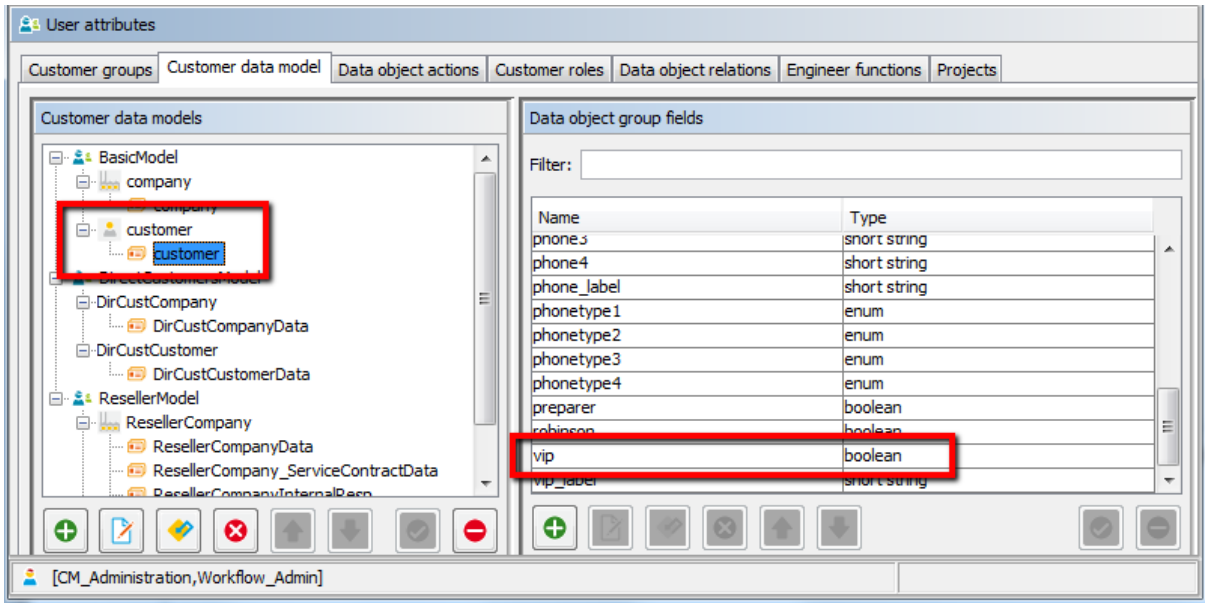


Fig. 7: ConSol*CM Admin-Tool - Data Object Group Field "vip" (CM Version 6.9)

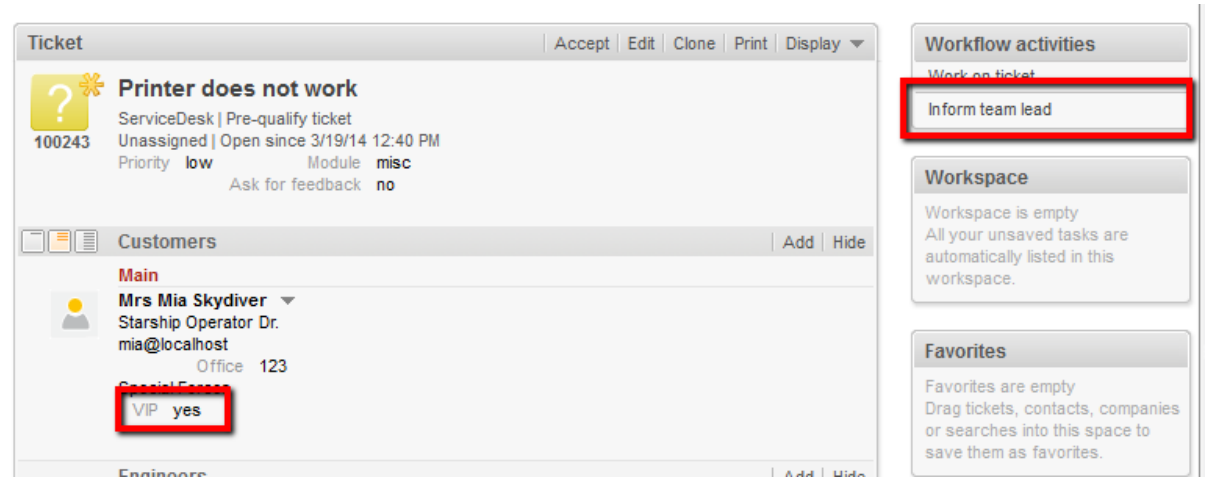


Fig. 8: ConSol*CM/Web Client - Precondition: Return Value TRUE

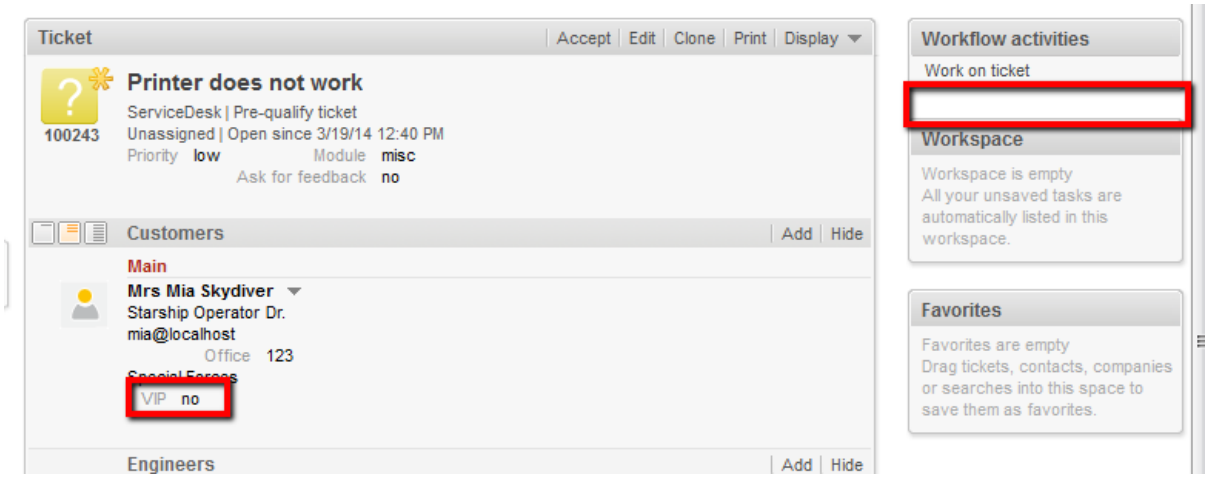


Fig. 9: ConSol*CM/Web Client - Precondition: Return Value FALSE

Example 2: Send an E-Mail to the Main Contact When a Ticket Has Been Opened

When a ticket has been opened, an e-mail should be sent to the main contact of the ticket.

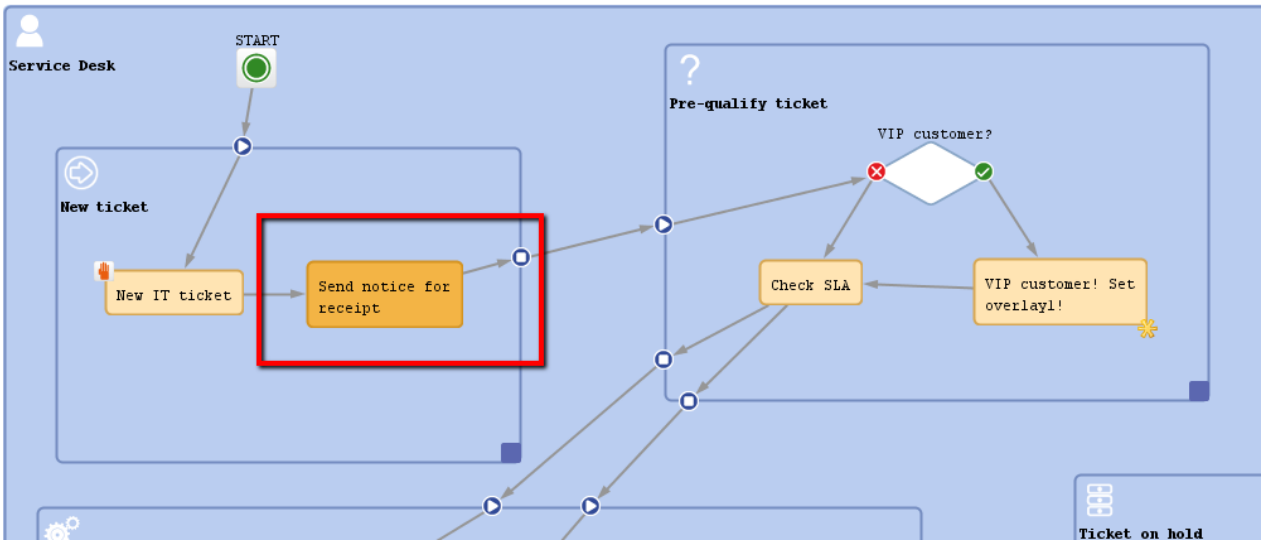


Fig. 10: ConSol*CM Process Designer - Automatic Activity Where Receipt Note Is Sent

Script for automatic activity where receipt note is sent, variant 1

```
// Get the main contact of the ticket:
def contact = ticket.getMainContact()

// Get the value of the custom field "email" of the main contact:
def contact_e = contact.get("email")

// Use as text the e-mail template with name "receipt_notice_ServiceDesk".
// Can be located in the Template Designer or in the Admin-Toool.
// Usually e-mail templates are stored in the Template Designer:
def text = workflowApi.renderTemplate("receipt_notice_ServiceDesk")

// Get the reply-to address for the e-mail.
// This is stored in the system property "cmweb-server-adapter","mail.reply.to":
def replyto = configurationService.getValue("cmweb-server-adapter","mail.reply.to")

// Build the string for the ticket subject.
// Keep in mind that the regular expression which defines the ticket identifier has to be in
this subject.
// Otherwise, an e-mail cannot be assigned to the correct ticket.
def subj = "Your request has been received: ticket (" + ticket.getId() + ")"

//Send out the e-mail
workflowApi.sendEmail(contact_e,subj,text,replyto,null)
```

Script for automatic activity where receipt note is sent, variant 2

```
// all lines of code identical to variant 1 except for the last line:

new Mail().setSubject( subj ).setTo( contact_e ).setReplyTo( replyto ).setText( text
).setTicketAttachments( null ).send()
```

Example 3: Assign the Ticket to the Current Engineer

The ticket should be assigned to the engineer who executes the activity *New IT ticket*.

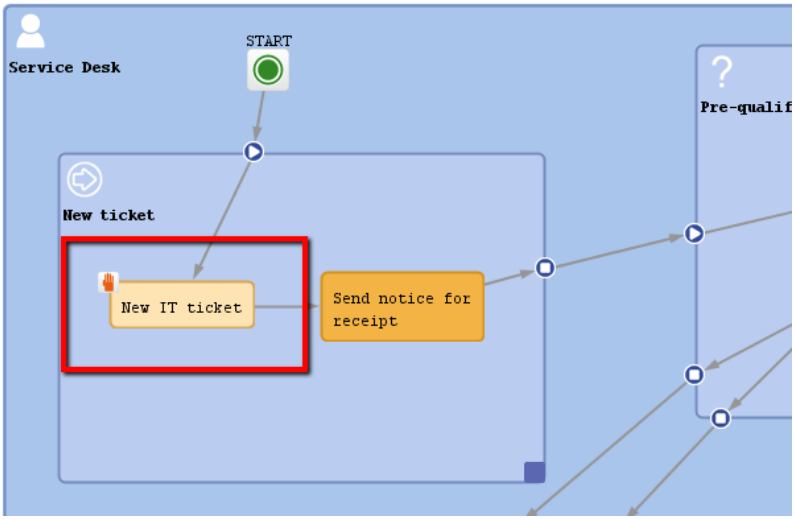


Fig. 11: ConSol*CM Process Designer - Workflow Activity Where Engineer Should Be Assigned

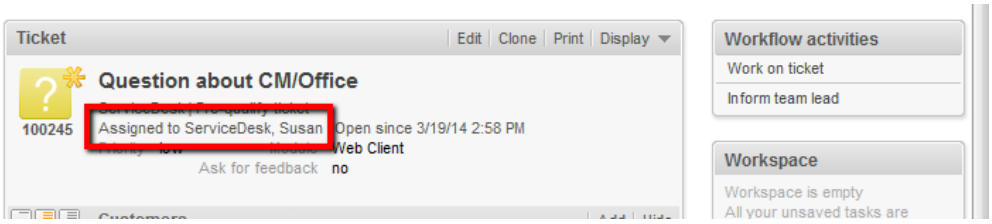


Fig. 12: ConSol*CM/Web Client - Ticket Passed through Activity Where Engineer Was Assigned

Script for assigning ticket to current engineer

```
// Get the engineer who is executing the activity:
def curr_eng = workflowApi.getCurrentEngineer()

// Assign the ticket to the current engineer
ticket.setEngineer(curr_eng)
```

Attention:

Make sure that you always use the correct engineer object!

The current engineer is the engineer who is logged in, who is executing the current activity. You can get the object by using the following method:

```
def curr_eng = workflowApi.getCurrentEngineer()
```

The ticket engineer is the person who is (at this point of time) the ticket owner and responsible for the ticket. You can get the object by using the following method:

```
def tic_eng = ticket.getEngineer()
```

4.6 Workflow Components: Decision Nodes

4.6.1 Introduction to Decision Nodes

A decision node is a node which has one or more entry points and exactly two exit points: *true* and *false*. A decision node always has to have a script which has to return either *true* or *false*.

The ticket enters the decision node, then the script is executed and - depending on the result (*true* or *false*) - the ticket leaves the node via the respective exit point.

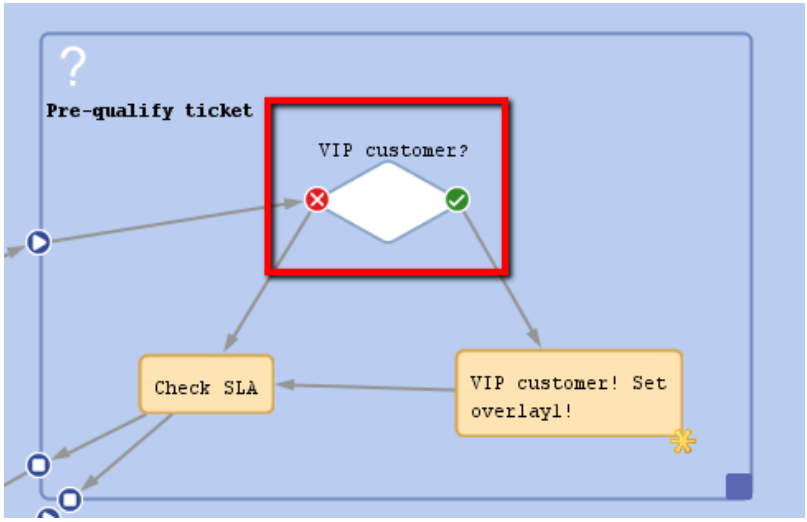


Fig. 1: ConSol*CM Process Designer - Decision Node

4.6.2 Properties of a Decision Node

A decision node has the following properties:

- **name**
  Mandatory, the technical object name.
- **label**
  Optional, the localized name which is displayed in the Web Client GUI.
- **condition**
  Mandatory, a script which returns *true* or *false* has to be provided.
- **history visibility**
  See section history visibility.

- **disable auto update**
  See section disable auto update.

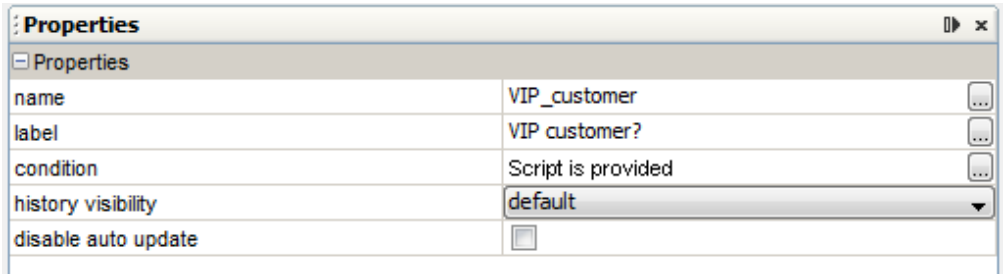


Fig. 2: ConSol*CM Process Designer - Decision Node: Properties

4.6.3 Example for a Decision Node

In the following example, the system should automatically check if the customer (main contact of the ticket) is a *VIP* customer. If yes, the ticket should be marked with the *VIP* overlay (in the example a yellow star).

1. A custom field of type *boolean* has to be defined in the customer data model to mark a customer as *VIP* (yes/no). Please refer to the *ConSol*CM Administrator Manual 6.8*, section *Custom Field Administration*.

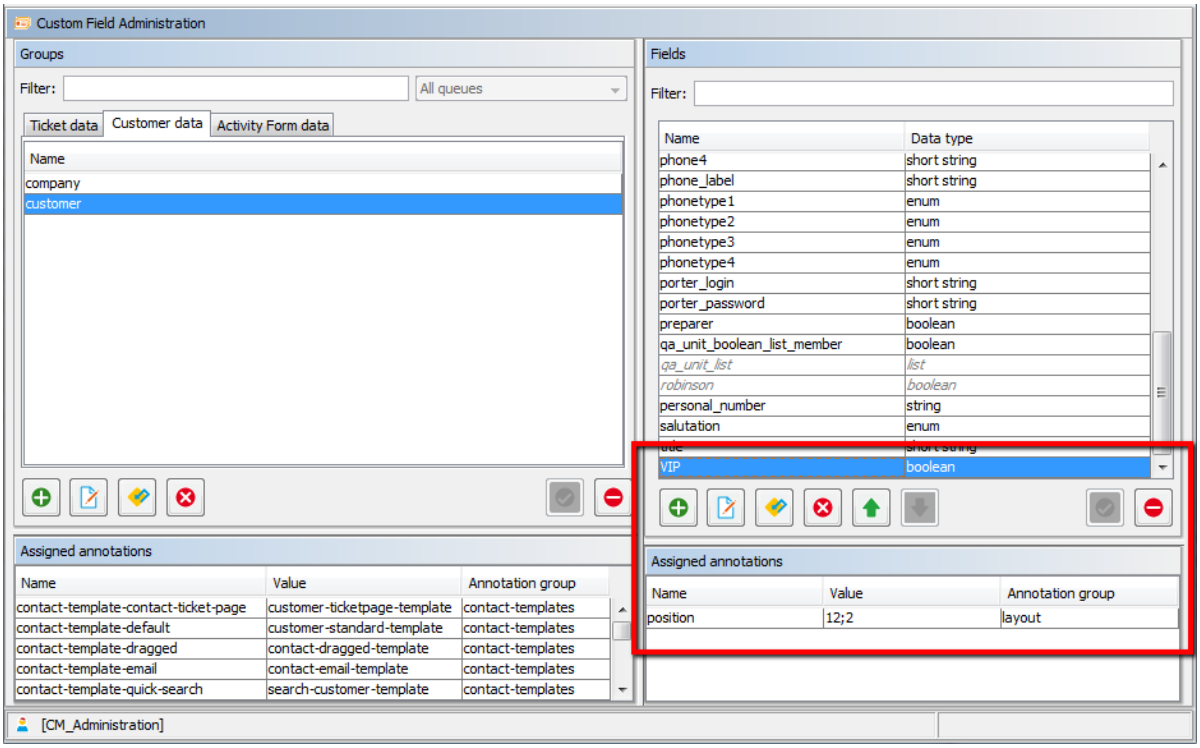


Fig. 3: ConSol*CM Admin-Tool - Custom Field "VIP" in Customer/Contact Data (CM Version 6.8)

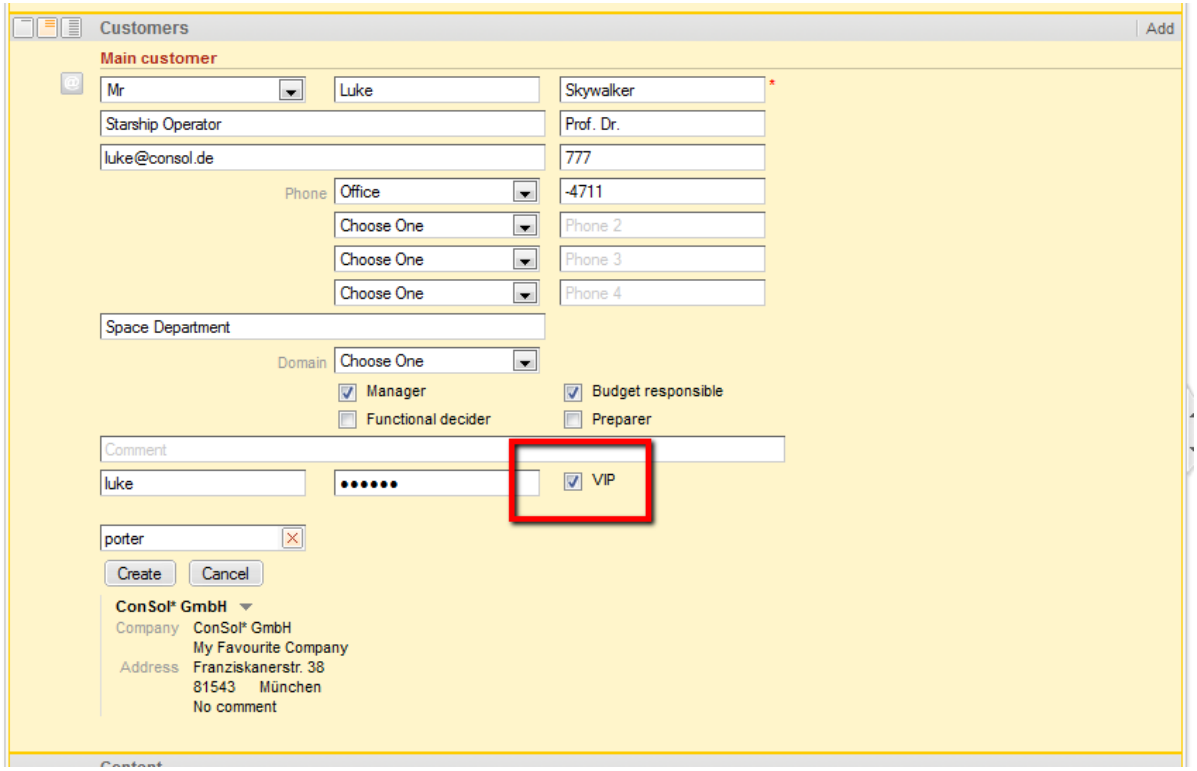


Fig. 4: ConSol*CM/Web Client - Custom Field "VIP" for Customer/Contact Data

2. In the script of the decision node, it has to be checked if the customer is a *VIP* (return value: *true*) or not (return value: *false*).

Example from CM version 6.8

```
// Get the main contact of the ticket. The unit object (can be a customer or a company) is
provided;
// here it has to be a customer, i.e. a contact:

Unit contact = ticket.getMainContact()

// Check the custom field "VIP" of the main contact. (see next image)
// If it is set to true, return true, i.e. the condition is TRUE.
// Else return false, i.e. the condition is FALSE:

if (contact.get("VIP")) {
    return true
} else {
    return false
}
```

3. When a ticket has passed automatically through the decision node and the following automatic activity where the *VIP* overlay is added, the ticket icon in the Web Client is marked with the overlay, see following figure.

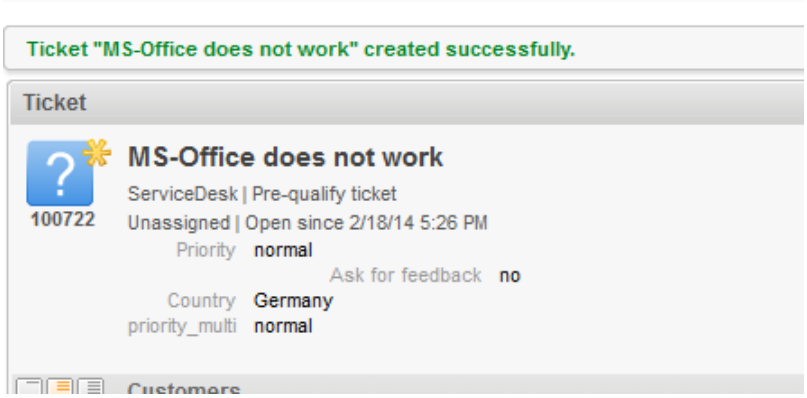


Fig. 5: ConSol*CM/Web Client - Ticket Icon with VIP Overlay

4.7 ConSol*CM Process Designer Manual - Adornments (Triggers and ACFs)

4.7.1 Adornments (Triggers and ACFs)

The ConSol*CM workflow engine can react to several kinds of events. This is controlled by triggers. ACFs offer dynamic forms.

Adornment type	Explanation
Time Triggers	Control the time which has elapsed since the ticket has entered a scope or an activity, see section Time Triggers.
Mail Triggers	Control if an e-mail has been received by a ticket in the scope, see section Mail Triggers.
Business Event Triggers	Control events like the change of the engineer or adding of a comment. See section Business Event Triggers.
ACF	Using Activity Control Forms (ACFs) you can control the data that have to be entered by the user in a certain step of the process, see section Activity Control Forms (ACFs).

4.7.2 Time Triggers

Introduction to Time Triggers

A workflow can contain several time triggers.

Fig. 1: ConSol*CM Process Designer - Time Trigger

A time trigger is a mechanism which reacts when a certain period of time has elapsed. This can be required, for example, in the following situations:

- **Use case 1:**
  An engineer wants to put a ticket on hold for a defined time, because he/she knows that the customer will not be available until then.
- **Use case 2:**
  The system should automatically control the escalation time, i.e. when a ticket has come in and has not been taken care of, there should be an alert (this can be an overlay at the ticket icon, an e-mail to the team lead, or other actions).
- **Use case 3:**
  A ticket has been solved and the engineer closes it. However, this should be a preliminary end and the ticket should be closed technically after a defined period of time.

Those use cases can be implemented using time triggers.

A time trigger can be configured to use a business calendar, i.e. to take only those times into consideration which are defined as working hours.

A time trigger can be attached to ...

- **a scope**
  Then it controls all tickets which are currently in the scope.
- **an activity**
  Then it controls only the tickets which have just entered this activity.

A time trigger has to be of one of two types:

- manual
- with a defined period of time

> **Information:**
>
> You as a workflow developer have to implement everything that should happen as a consequence when a time trigger has fired! There are no automatic actions. All the time trigger does, is to give a signal *time elapsed* - just like an alarm clock.

Adding a Time Trigger to a Workflow

Adding a Time Trigger to a Scope

Grab the time trigger icon in the palette and drop it into the desired scope. It is automatically attached to the top of the scope. You can modify the position afterwards (move it to the left or right to change the order of triggers or just to improve the layout).

A time trigger, which has been attached to a scope, cannot be moved to another scope or activity. In case you would like to attach a time trigger to another scope/activity, remove the one you have defined and create a new one for the correct scope/activity.

To configure the properties of the trigger, select it in the editing panel and set the correct values in the Properties Editor. See section Properties of a Time Trigger.

You can draw connections from the trigger to put activities or decision nodes behind it. The first step which is executed after a time trigger always has to be an automatic activity!

Adding a Time Trigger to an Activity

Grab the time trigger icon in the palette and drop it into the desired activity. It will be attached to the corner of the activity.

A time trigger which has been attached to an activity cannot be moved to another scope or activity. In case you would like to attach a time to another scope/activity, remove the one you have defined and create a new one for the correct scope/activity.

To configure the properties of the trigger, select it in the editing panel and set the correct values in the Properties Editor. See section Properties of a Time Trigger.

You can draw connections from the trigger to put activities or decision nodes behind it. The first step which is executed after a time trigger always has to be an automatic activity!

Properties of a Time Trigger

A time trigger has the following properties:

- **name**
  Mandatory. The technical name of the trigger. It is set automatically but can be changed manually.
- **minutes/hours/days**
  Here you can enter the time interval after which the trigger should fire. The display mode always refers to a 24-hours-day, i.e. when you have entered 30 hours as reaction time and you re-open the workflow, there will be 1 day, 6 hours.
- **use calendar**
  Optional. Mark this check box when the business calendar should be taken into consideration when the time interval is calculated.

> **Attention:**
>
> Please keep in mind that there are three steps which are necessary to make sure time intervals are calculated using a business calendar:
>
> 1. Define a business calendar (see *ConSol*CM Administrator Manual*, section *Business Calendars*).
> 2. Assign the correct business calendar to a queue (see *ConSol*CM Administrator Manual*, section *Queue Administration*).
> 3. Mark the check box *use calendar* for each trigger which should work with the calendar.

> **Principle of the use of a business calendar:**
>
> 1 day means 24 hrs of absolute time, it has nothing to do with the use of a calendar. The calendar only plays a role when the time trigger is activated, then the 24 hrs, i.e. 86400000 milliseconds, will be taken as business calendar input (if the calendar is enabled).
> **Example:**
> When we have as trigger time 1 day = 24 hrs without calendar, the 24 hrs are calculated like regular time, so the escalation will fire one day later at the same time.
> In contrast: When we use a calendar (with, for example, 7 work hrs per work day), the 24 hrs will be split-up according to the calendar, resulting in the firing event more than 3 days later (24 hrs = 3 x 7 hrs + 3 hrs).

See also section Working with Calendars and Times.

- **repeatable**
  Optional. Mark this check box to make sure the trigger can fire more than once for one ticket. If a trigger is *repeatable*, it is reset immediately after it has fired, i.e. the time count starts again.

  > **Info for experts:**
  >
  > The script on timer start is executed again. The first firing event is initialized by the (technical) user *admin*, all following firing events are initiated by the *Job Executor*.

- **script after timer**
  Optional. A script can be defined which is executed when the time interval which is controlled by the trigger has elapsed, i.e. when the time trigger fires.
- **script on timer start**
  Optional. A script can be defined when the time trigger starts to measure time, i.e. when the ticket has entered the scope/activity to which the trigger is attached.
- **activate manually**
  Optional, only for time triggers at activities. Mark this check box when the user (the engineer) should select the time when the trigger should fire. For the user, a date-picker (web calendar) is displayed.
- **retry interval**
  The time in seconds after which the trigger execution should be executed again in case a script has run with an error. The time can be configured in the Admin-Tool (property *jobExecutor.timerRetryInterval.seconds*).

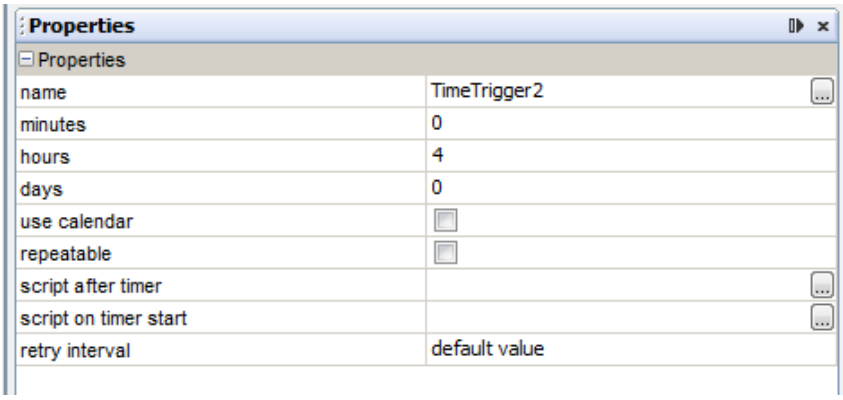


Fig. 2: ConSol*CM Process Designer - Properties of a Time Trigger

Business Logic and Initialization of a Time Trigger

The time measuring of a trigger is started (i.e. the trigger is initialized) when the ticket enters the scope/activity. It stops (i.e. the trigger fires) when the defined period of time which has been set as fixed value (minutes/hours/days) or the manually defined time has elapsed.

When you as a workflow developer would like to initialize a trigger using other values, this has to be done using scripts. Here, short examples will be provided, please see section Working with Calendars and Times for a detailed explanation of programming workflow trigger times. In those chapters, the code examples are provided, too.

- **Example 1:**
  The reaction time for a ticket should be calculated based on the priority. In the *script on timer start*, the different reaction times are used (a good way to implement this, would be to use customer-specific system properties) and the reaction time is calculated. Then the trigger is initialized, i.e. the time interval is set.
- **Example 2:**
  When an e-mail to a ticket has come in and after three hours, no engineer has read the e-mail and has taken care of the ticket, an alert should be triggered. To implement this, an incoming e-mail (see section Mail Triggers) has an adjacent automatic activity which re-initializes a time trigger with 3 hours.

A time trigger can also be deactivated. In *Example 2*, this would be required to prevent the time trigger from firing initially, because it should not be initialized before any e-mail comes in.

Examples for Time Triggers

The implementations for the use cases mentioned above (see Introduction to Time Triggers) would be:

- **Use case 1:**
  Put a manual time trigger to the activity *Put ticket on hold*. The engineer can select the desired end date by using the date picker in the Web Client. Usually then the ticket is led back to the active tickets.

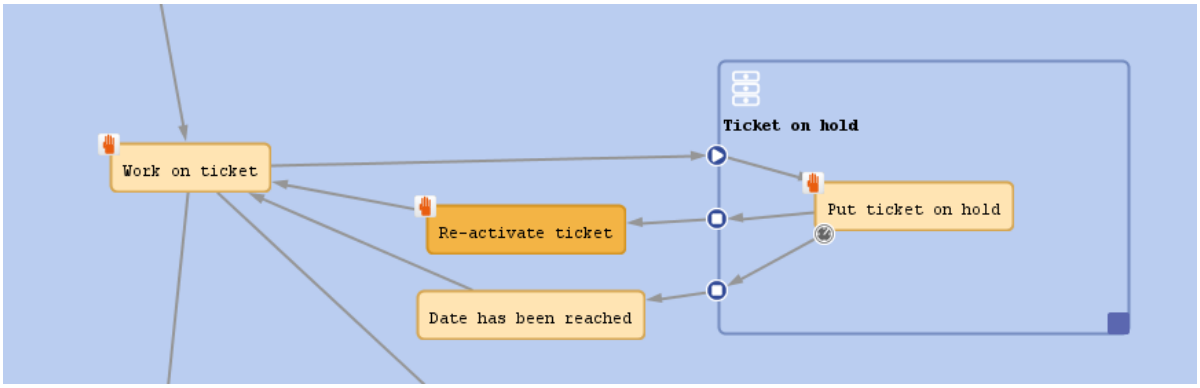


Fig. 3: ConSol*CM Process Designer - Use Case 1: Workflow

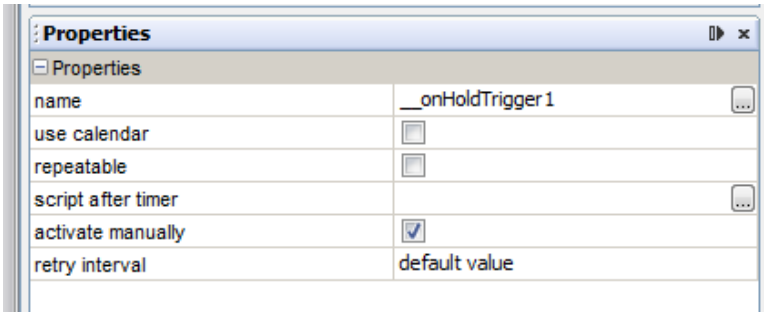


Fig. 4: ConSol*CM Process Designer - Use Case 1: Properties Editor for Time Trigger

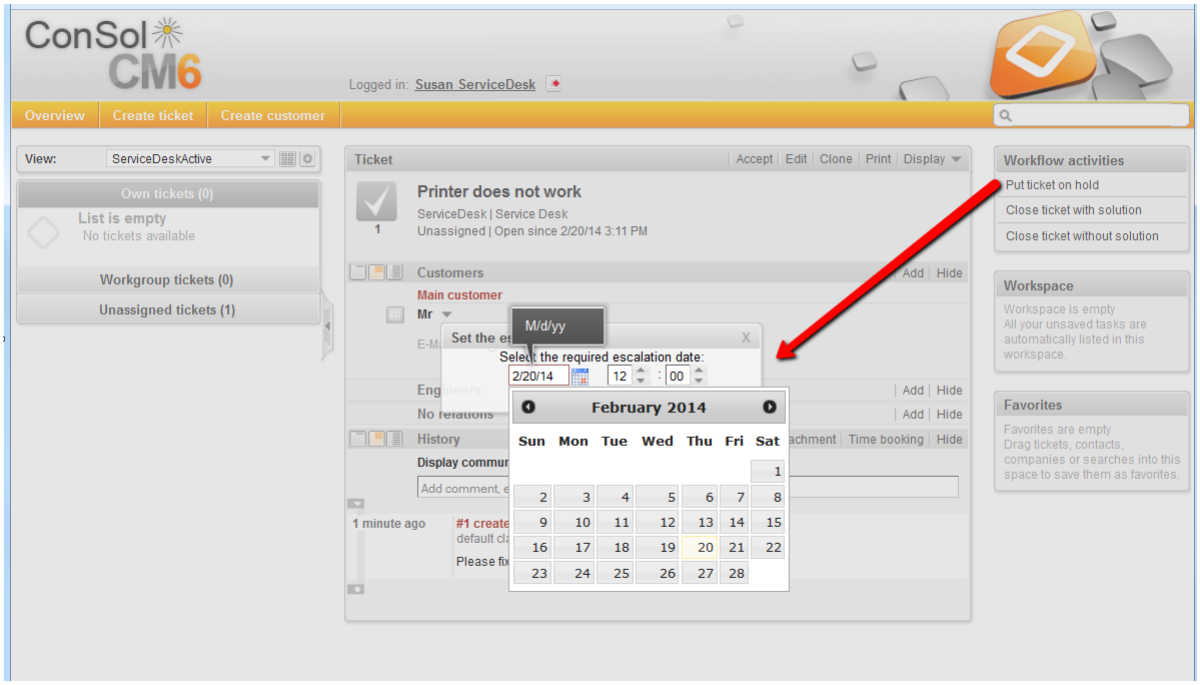


Fig. 5: ConSol*CM/Web Client - Use Case 1: Date Picker

- **Use case 2:**
  Put a time trigger on the scope where the new tickets come in. Define the time for the trigger (this might depend on SLAs), e.g. four hours. Put a control behind the trigger if an engineer has taken care of the ticket or not. If not, an e-mail is sent to the team lead.

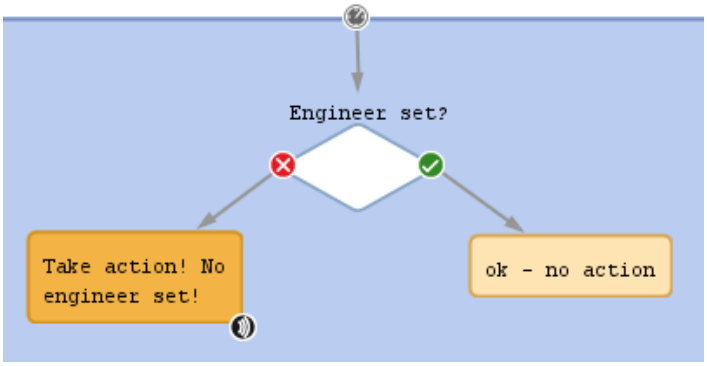


Fig. 6: CM Process Designer - Use Case 2: Workflow

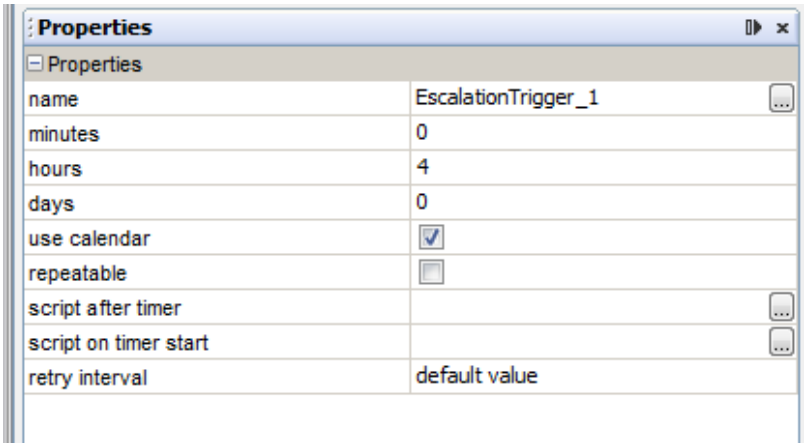


Fig. 7: ConSol*CM Process Designer - Use Case 2: Properties Editor for Time Trigger

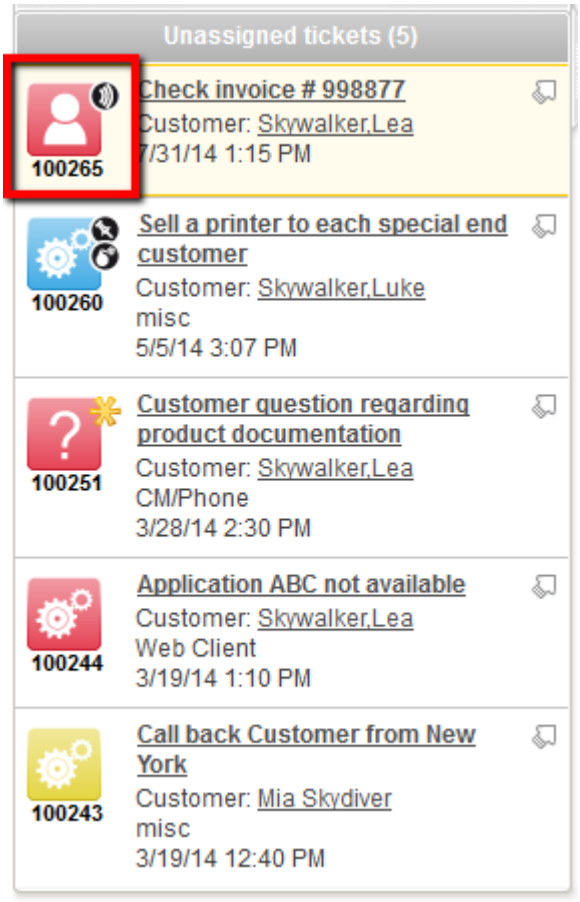


Fig. 8: ConSol*CM/Web Client - Use Case 2: Ticket List

- **Use case 3:**
  Put a time trigger to the activity *Close ticket with solution* and set a defined period of time for the trigger, e.g. five days. Behind the trigger there is the end node of the process. For five days, the ticket can still be edited, after this time, it is closed automatically.

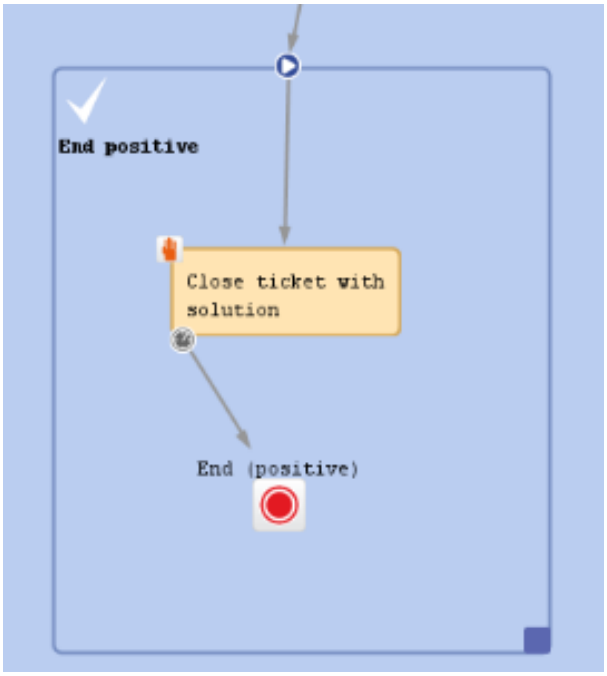


Fig. 9: CM Process Designer - Use Case 3: Workflow

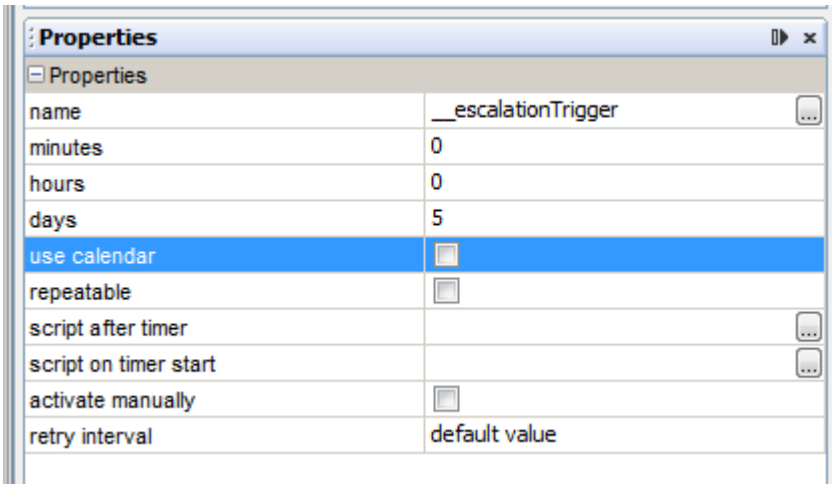


Fig. 10: ConSol*CM Process Designer - Use Case 3: Properties Editor for Time Trigger

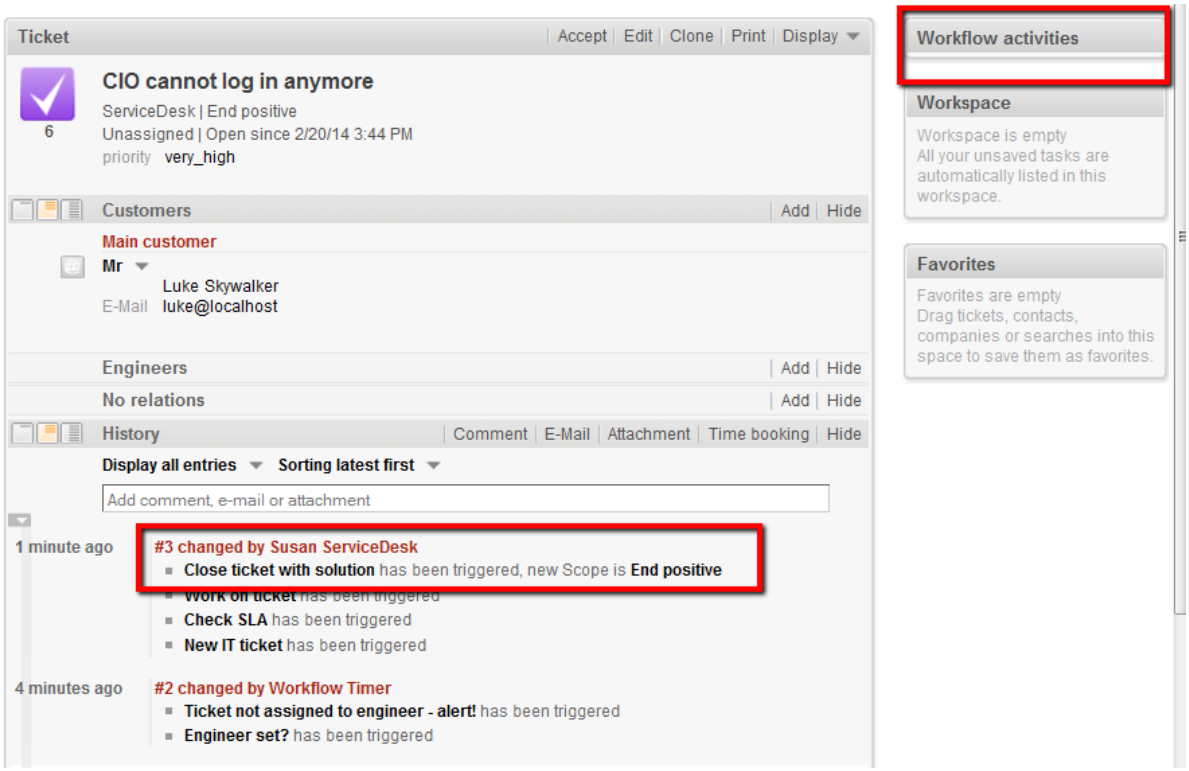


Fig. 11: ConSol*CM/Web Client - Closed Ticket

Scripting with Time Triggers

The following methods are of major importance when you work with time triggers:

- **TimerTrigger.setDueTime(long pDueTime in millisecs)**
  Sets the time when the trigger should fire. The time recording starts when the trigger enters the scope or activity where the trigger is attached. So *setDueTime()* defines the time period in milliseconds from the entry time to the desired firing event.
- **workflowApi.reinitializeTrigger()**
  (different method signatures)
  Starts the time recording for the given trigger again, i.e. re-sets its start time.

- **workflowApi.deactivateTimer()**
  (different method signatures)
  Deactivates the given time trigger, i.e. the trigger will never fire until re-initialized.
  (There is **no** method *activateTimer().* Use *workflowApi.reinitializeTrigger()* to re-activate the trigger).

Please see also section Working with Calendars and Times.

Example 1: Set the Due Time of a Time Trigger Depending on the Queue

This script could be used as a script on timer start for a time trigger at a scope. It will initialize the trigger for an escalation depending on the queue, i.e. if the ticket is in the *HelpDesk_1st_Level* queue there is less time until the escalation than in the *HelpDesk_2nd_Level* queue.

Within the scripts *scripts on timer start* and s*cript after timer*, the object timer exists as an implicit initialization of *TimerTrigger*. So you can work using triggers without any steps before. However, in an Admin-Tool script you will have to import the *TimerTrigger* class or the respective Java package.

The following script could be used in a service desk and help desk environment and placed in the following *TimerTrigger*.

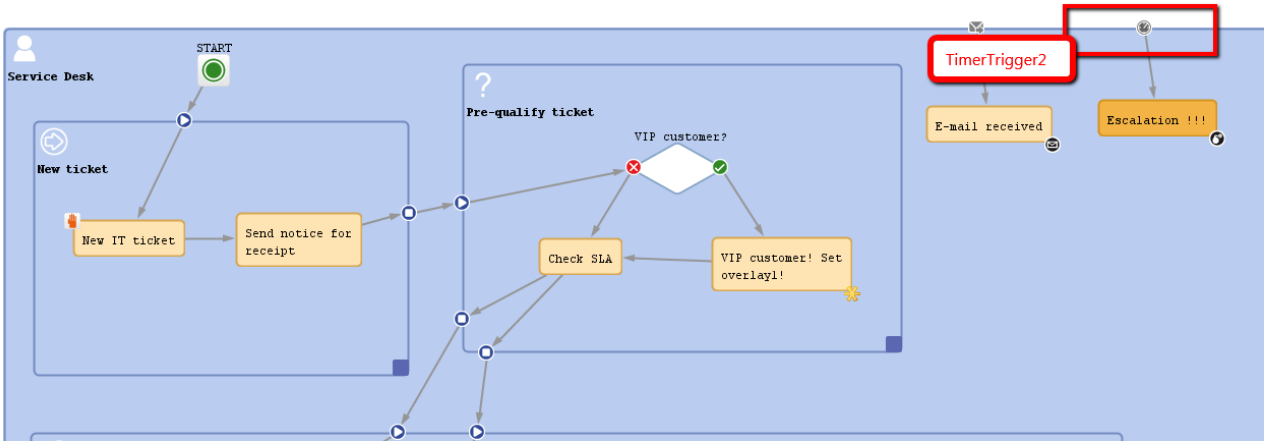


Fig. 12: ConSol*CM Process Designer - TimerTrigger in ServiceDesk Workflow

Example for a script on timer start

```
def addedEscalMillis = 0
switch (ticket.queue.name) {
  case "HelpDesk_1st_Level":
    addedEscalMillis = 12*60*60*1000L;
    break;
  case "HelpDesk_2nd_Level":
    addedEscalMillis = 24*60*60*1000L;
    break;
  case "ServiceDesk":
    addedEscalMillis = 4*60*60*1000L;
 }
trigger.setDueTime(addedEscalMillis)
```

> **Attention:**
>
> For this example, it makes sense to use fixed values for the times directly in the script code. In real life environments you might want to store escalation times and the like in system properties and retrieve them using the *configurationService*. That way, an administrator can easily access and edit the escalation times without any manipulation of the workflow implementation.
>
> In real life, a business calendar might also be used - please see *Example 2*.

In the *server.log* file, you can see the time when the trigger is supposed to fire.

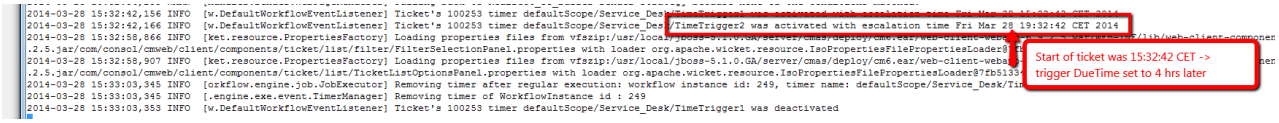


Fig. 13: File server.log with Calculated Timer DueTime

The same principle could be applied to calculate the escalation time depending on the ticket priority, the *VIP* status of a customer, or any other parameter.

Example 2: Calculate an Escalation as Warning 2 Days before Desired End Date

Calculate and set time for TimerTrigger using BusinessCalendar

```
def now = new Date()
def wunschTermin = ticket.get("helpdesk_standard", "date_test")
def twoWorkDays = -2*8*60*60*1000L

// calculate escalation date
def escalDate = BusinessCalendarUtil.getBusinessTime(wunschTermin, twoWorkDays,
ticket.queue.calendar)
// calculate and set due time
trigger.setDueTime(escalDate.time - now.time)
```

4.7.3 Mail Triggers

Introduction to Mail Triggers

One of the core functionalities of ConSol*CM is its interaction with an e-mail infrastructure. This makes it possible for the engineer to send manual e-mails and for the system to send automatic e-mails to customers and to engineers, as required in the respective process step. Obviously, ConSol*CM has also to receive e-mails. This is done by retrieving e-mails from one or more mailboxes with ConSol*CM-owned addresses. For a detailed explanation of all interactions between the mail server and ConSol*CM, please refer to the *ConSol*CM Administrator Manual* and the *ConSol*CM Operations Manual*. Here, only the workflow interactions are explained.

Fig. 1: ConSol*CM Process Designer - Mail Trigger

Mail Trigger at a Scope

When an e-mail is received which belongs to an existing and active (open) ticket, it might be required to register this action and to perform specific actions subsequently. This can be achieved using one or more mail triggers within a workflow.

Attention:

Please keep in mind that (in the default configuration, i.e. without modification of the Admin-Tool script *AppendToTicket.groovy*) the **only automatic action**, which is performed by ConSol*CM after having received an e-mail in a specific mailbox, is to attach this e-mail to the ticket with the matching ticket tag in the subject, e.g. *Ticket (<TicketNumber>)*. See also *ConSol*CM Administrator Manual* section *Scripts of Type E-Mail*.

All other actions, which should be executed when an e-mail has been received, have to be programmed **manually** in the workflow (and/or in Admin-Tool scripts)!

Examples for the use of mail triggers are:

When an e-mail has been received ...

- the engineer of the ticket (the ticket owner) should also get an e-mail as notification.
- the ticket icon (in the Web Client) should be marked by an overlay.
- the ticket should be transferred to an activity where the engineer has to confirm that he/she has read the e-mail.
- the sender and the subject of the e-mail are checked and parsed. If the e-mail is a confirmation or a denial in an approver process, the ticket is managed according to the defined rules and activities in the workflow. That way, the approval can be performed using the e-mail only, no login of the approver in the Web Client is required.

Mail Trigger at an Activity

When a mail trigger is attached to an activity, this activity is only executed when an e-mail is received.

Fig. 2: ConSol*CM Process Designer - Mail Trigger at Activity

Adding a Mail Trigger to a Workflow

Adding a Mail Trigger to a Scope

Grab the mail trigger icon in the palette and drop it into the desired scope. It is automatically attached to the top of the scope. You can modify the position afterwards (move it to the left or right in order to improve the layout). Only one mail trigger can be used per scope.

A mail trigger which has been attached to a scope cannot be moved to another scope. In case you would like to attach a mail trigger to another scope, remove the one you have defined and create a new one for the correct scope.

You can draw connections from the trigger to put activities or decision nodes behind it. The first step which is executed after a mail trigger always has to be an automatic activity!

Adding a Mail Trigger to an Activity

In the very rare case that you have to attach a mail trigger to an activity (we do not recommend this!), grab the mail trigger icon in the palette and drop it into the desired activity. It will be attached to the corner of the activity.

A mail trigger which has been attached to an activity cannot be moved to another scope or activity. In case you would like to attach a mail trigger to another scope/activity, remove the one you have defined and create a new one for the correct scope/activity.

Properties of a Mail Trigger

A mail trigger does not have any properties.

Examples for Mail Triggers

Use Case 1: Overlay for Ticket Icon

When an e-mail has been received for a ticket which is currently in the scope, the ticket icon in the Web Client GUI should be marked with the overlay *mail*.

The mail trigger is attached to the scope and the overlay is attached to the adjacent automatic activity. The overlay range is *activity*.

That way, the ticket is marked with the overlay when the e-mail has come in. As soon as an engineer has moved the ticket to another activity, the overlay disappears.

Please note that the ticket does not leave its context. All that happens is the attachment of the overlay to the ticket icon. Then the ticket returns to its original position in the workflow. We call this an interrupt. Please read the section Process Logic for a detailed explanation.

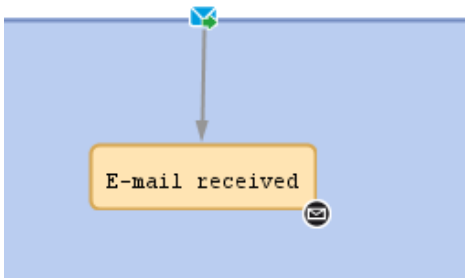


Fig. 3: ConSol*CM Process Designer - Use Case 1: Scope with Mail Trigger

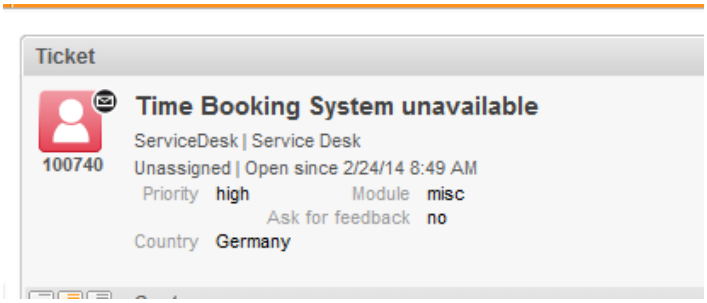


Fig. 4: ConSol*CM/Web Client - Use Case 1: Ticket with Overlay Icon

Use Case 2: Overlay for Ticket Icon and E-Mail Confirmation by Engineer

When an e-mail has been received for a ticket which is currently in the scope, the ticket icon in the Web Client GUI should be marked with the overlay *mail*. Additionally, the ticket should be transferred to a position where it waits until the engineer has confirmed that he/she has read the e-mail.

The mail trigger is attached to the scope and the overlay is attached to the adjacent automatic activity. The overlay range is *activity*. That way, the ticket is marked with the overlay when the e-mail has come in.

Within the script which follows the mail trigger, a *boolean* field *mail_to_read* is set to *true*. In the workflow, an activity *Confirmed: e-mail read!* is offered wherever required. It is only displayed in case the value of the *boolean* field *mail_to_read* is *true*. This is a stronger mechanism to remind the engineer of an incoming e-mail than to use only the overlay. The engineer has to confirm the e-mail by executing the workflow activity *Confirmed: e-mail read!* explicitly. Within this workflow activity the value of the *boolean* field *mail_to_read* is set back to *false*. Now the ticket is ready to receive another e-mail and to notify the engineer.

Please note that also in this case the ticket does not leave its context as a consequence of the action which is executed after the e-mail has come in. All that happens is the attachment of the overlay to the ticket icon and the modification of a *boolean* variable. The ticket returns to its original position in the workflow. So this is also an interrupt. Please read the section Process Logic for a detailed explanation.

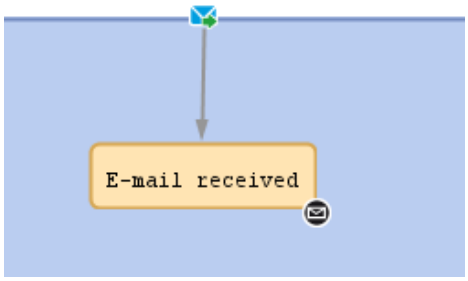


Fig. 5: ConSol*CM Process Designer - Use Case 2: Scope with Mail Trigger

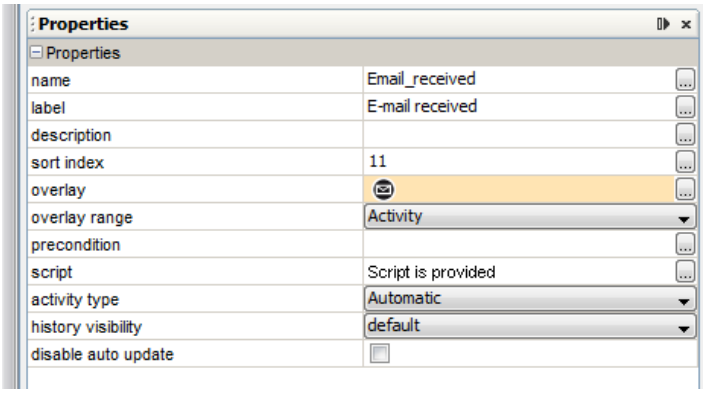


Fig. 6: ConSol*CM Process Designer - Use Case 2: Properties of Activity "E-mail received"

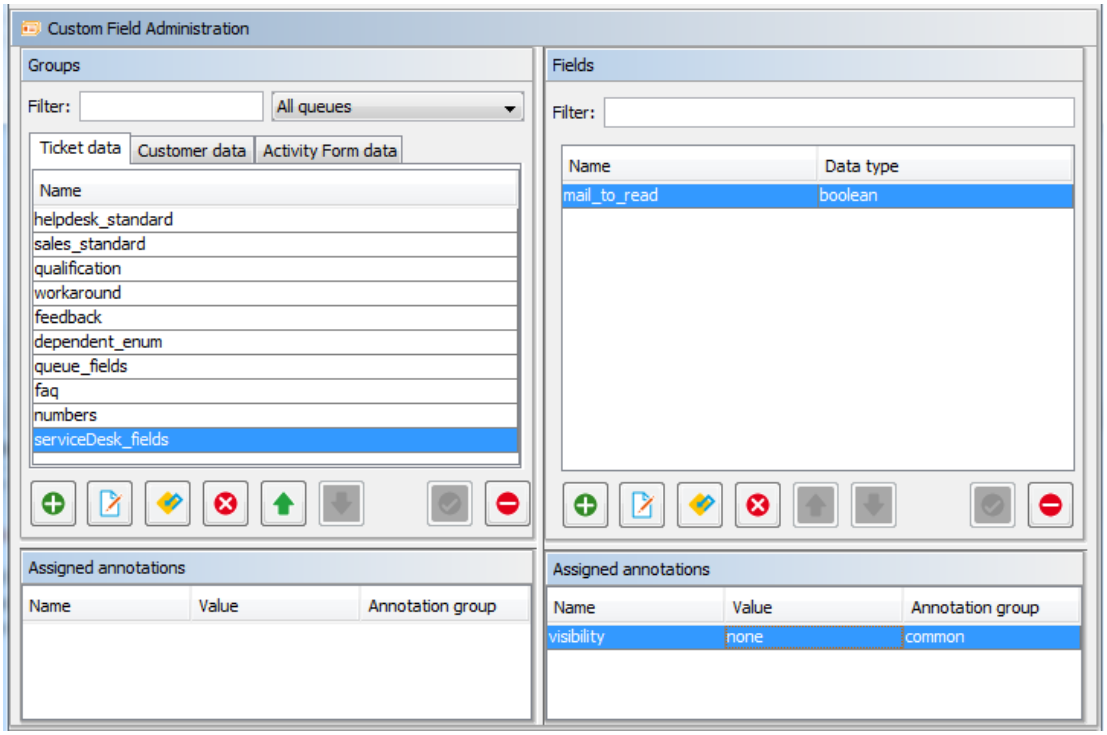


Fig. 7: ConSol*CM Admin-Tool - Use Case 2: New Boolean Field to Register E-Mail

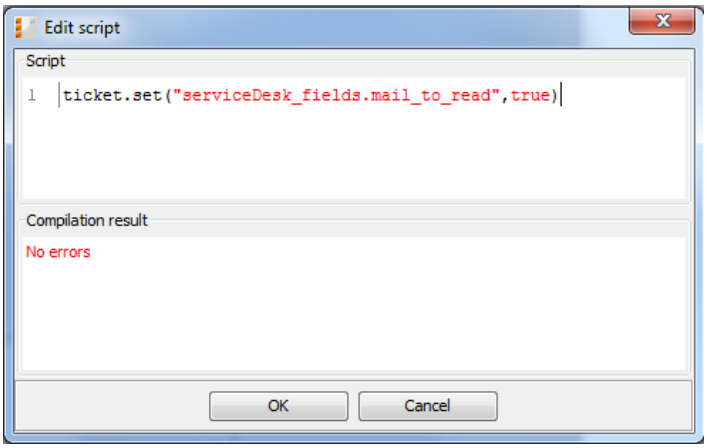


Fig. 8: ConSol*CM Process Designer - Use Case 2: Script for Activity "E-mail received"

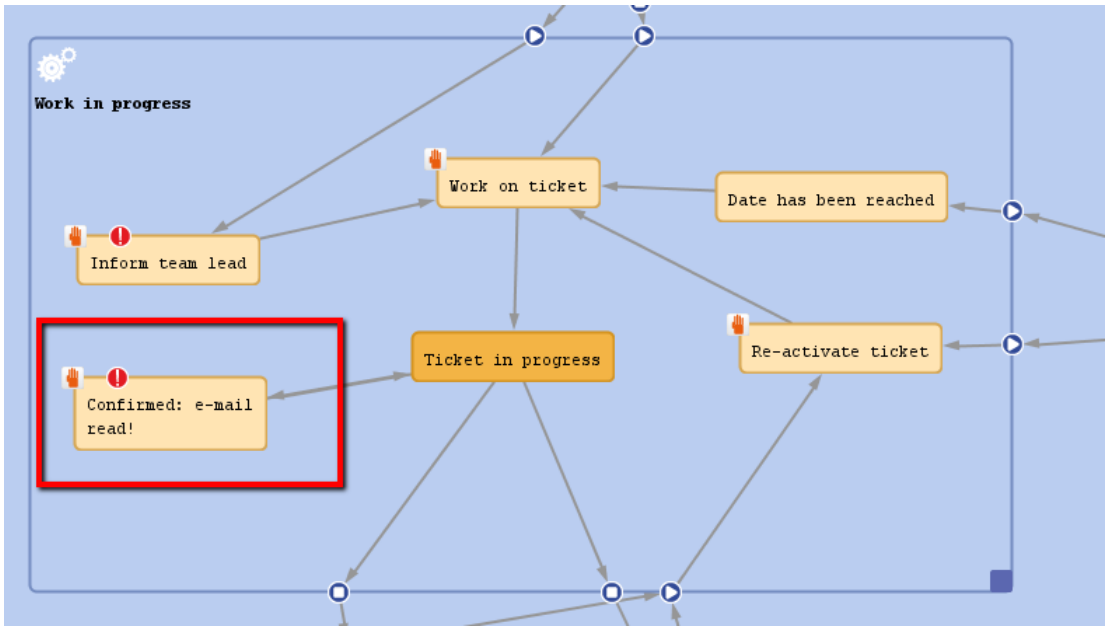


Fig. 9: ConSol*CM Process Designer - Use Case 2: Activity for E-Mail Confirmation

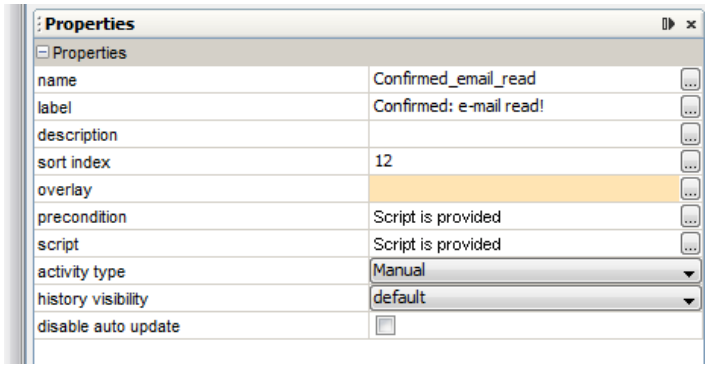


Fig. 10: ConSol*CM Process Designer - Use Case 2: Properties of Activity "Confirmed: e-mail read!"

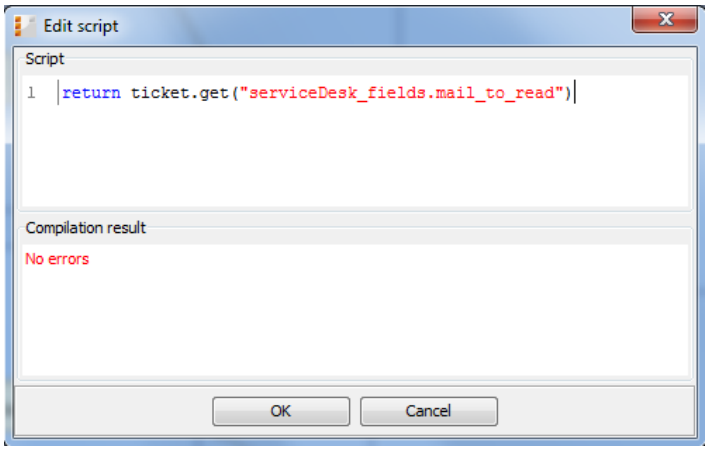


Fig. 11: ConSol*CM Process Designer - Use Case 2: Precondition Script for Activity "Confirmed: e-mail read!"

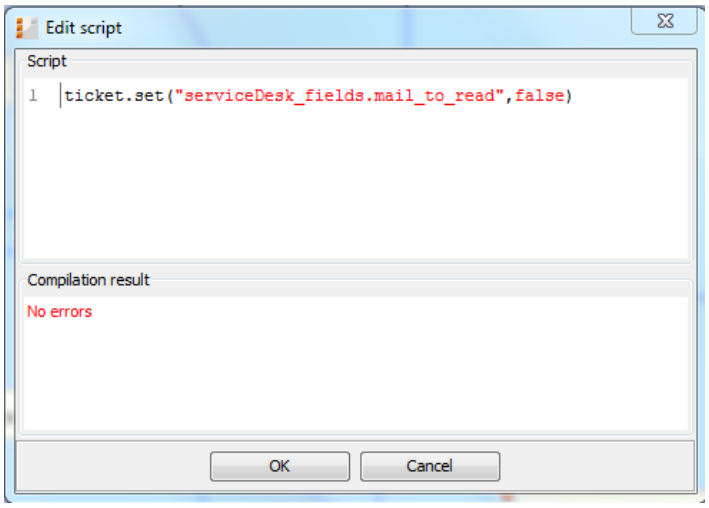


Fig. 12: ConSol*CM Process Designer - Use Case 2: Script for Activity "Confirmed: e-mail read!"

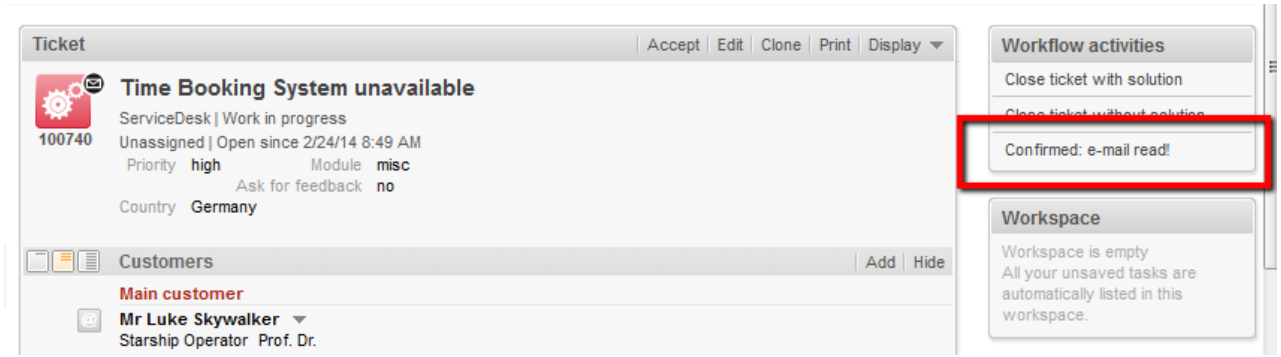


Fig. 13: ConSol*CM/Web Client - Use Case 2: Workflow Activity "Confirmed: e-mail read!"

Process Logic with Mail Triggers

When an e-mail is received, the mail trigger of the innermost possible scope fires.

Example 1:

The ticket is at position **(1)** in the *Ticket on hold* scope. When an e-mail comes in, the mail trigger for this scope fires **(2)** and, as a consequence, the ticket is moved to another scope **(3)**.

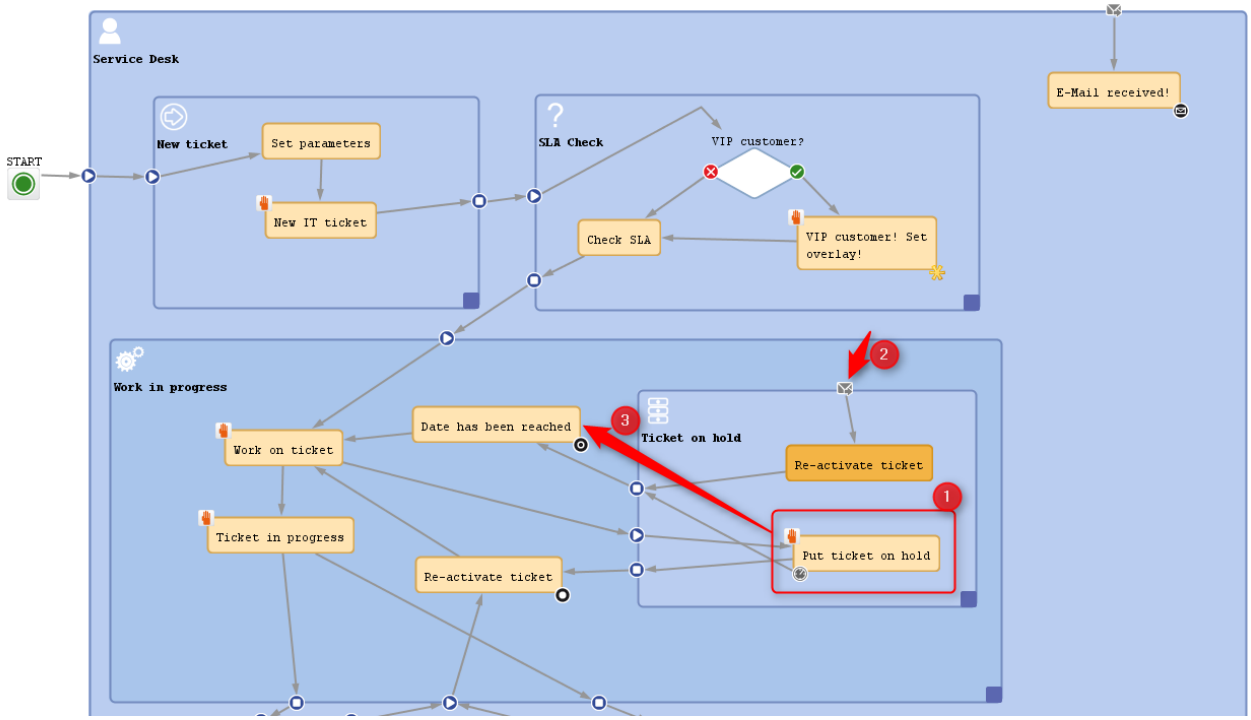


Fig. 14: ConSol*CM Process Designer - Example 1: Mail Trigger of Sub-Scope Active

Example 2:

The ticket is at position **(1)** in the *Work in progress* scope. When an e-mail comes in, the mail trigger of the main scope **(2)** fires (because the *Work in progress* scope does not have a mail trigger). So the ticket position is not changed **(3)**.

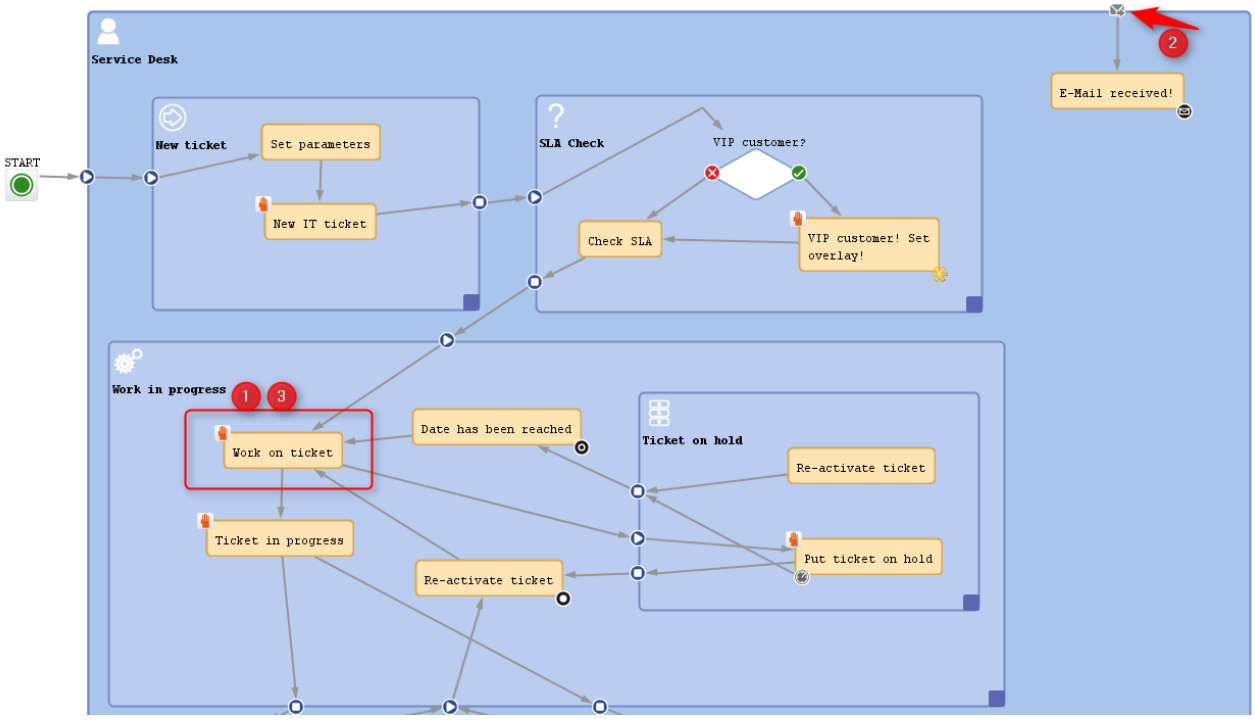


Fig. 15: ConSol*CM Process Designer - Example 2: Mail Trigger of Main Scope Active

4.7.4 Business Event Triggers

Introduction to Business Event Triggers

In business processes, there are often events during a regular process which have to be taken care of. For example, it might be required to inform the team lead if someone sets a ticket priority to *Extra High*. Or, after a change of the engineer of a ticket, it might be required to see if the engineer is logged in (if he/she is not in, the ticket has to be transferred to another engineer). There are numerous examples in business life for such events.

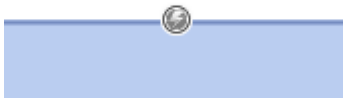


Fig. 1: ConSol*CM Process Designer - Business Event Trigger

ConSol*CM can notice events using business event triggers and can react to the following types of events:

- change of engineer
- change of queue
- change of the subject
- change of the referenced engineer(s)
- change of a comment
  (usually adding a new comment, i.e. a text comment or an e-mail)
- change of any custom field which has been defined by the system developer
  (this can be e.g. the priority, a category, the content of a certain text box)

When the event occurs, the business event trigger fires.

> **Information:**
>
> You as a workflow developer have to implement everything that should happen as a consequence when a business event trigger has fired! There are no automatic actions. All the business event trigger does, is to give a signal *event has occurred*.

A workflow can contain as many business event triggers as required. However, you have to make sure that in the process it is possible that all business event triggers can fire potentially (and that one does not depend on an action which cannot ever happen, because another business event (or time) trigger has fired before). Please see section Process Logic for more information.

Adding a Business Event Trigger to a Workflow

Business event triggers can only be attached to a scope, never to activities.

Adding a Business Event Trigger to a Scope

Grab the business event trigger icon in the palette and drop it into the desired scope. It is automatically attached to the top of the scope. You can modify the position afterwards (move it to the left or right to change the order of triggers or just to improve the layout).

A business event trigger which has been attached to a scope cannot be moved to another scope. In case you would like to attach a business event trigger to another scope, remove the one you have defined and create a new one for the correct scope.

To configure the properties of the trigger, select it in the editing panel and set the correct values in the Properties Editor. See the following section Properties of a Business Event Trigger .

You can draw connections from the trigger to put activities or decision nodes behind it. The first step which is executed after a business event trigger always has to be an automatic activity!

Properties of a Business Event Trigger

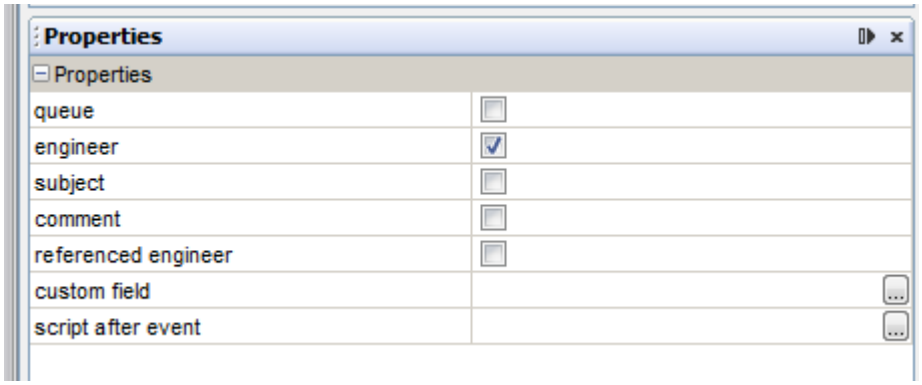


Fig. 2: ConSol*CM Process Designer - Properties of a Business Event Trigger

A business event trigger has the following properties:

- **queue**
  Mark this check box if the business event trigger should react to a change of the queue, i.e. the trigger fires when the ticket is transferred to another queue. It is not relevant if this has been a manual action or has been performed automatically by the system.
- **engineer**
  Mark this check box when the trigger should react to a change of the engineer (owner) of the ticket. This can be a manual or an automatic action. There are three possible constellations:
  - The ticket did not have an engineer and an engineer is set.
  - The ticket has an engineer and the ticket is given to another engineer.
  - The ticket has an engineer and the engineer is set to *null* (no engineer).
- **subject**
  Mark this check box when the trigger should react to a change of the ticket subject.
- **comment**
  Mark this check box when the trigger should react to the change of a comment, i.e.:
  - An engineer has added a new (text) comment.
  - A customer has added a new (text) comment using ConSol*CM/Track access.
  - An e-mail has been received for the ticket.
  - An e-mail has been sent out from the ticket.
  - One or more attachment(s) has/have been added to the ticket.
- **referenced engineer**
  Mark this check box when the trigger should react to a change of additional engineers in certain engineer roles of the ticket (ticket section *Engineers*). This can be one of the following situations (manually set or automatically by the system):
  - The ticket did not have any additional engineers and one or more additional engineer(s) is/are set.
  - The ticket has one or more additional engineer(s) and one or more of them is/are set to *null* or changed to another name.
  - The ticket has one or more additional engineer(s) and all those engineers are set to *null* (no engineer).

- **custom field**
  Use the (...) button to open the pop-up window *Event trigger* (see next figure) where you can select the custom field(s) which should be monitored. Use the *plus* and *minus* buttons to add more fields or to reduce the number of monitored fields. As in the custom field definition (see *ConSol*CM Administrator Manual*, section *Custom Field Administration*), you first have to select the custom field group in the left pull-down menu and then you can choose one of the custom fields of this group in the right pull-down menu. You can select as many custom fields as you like.

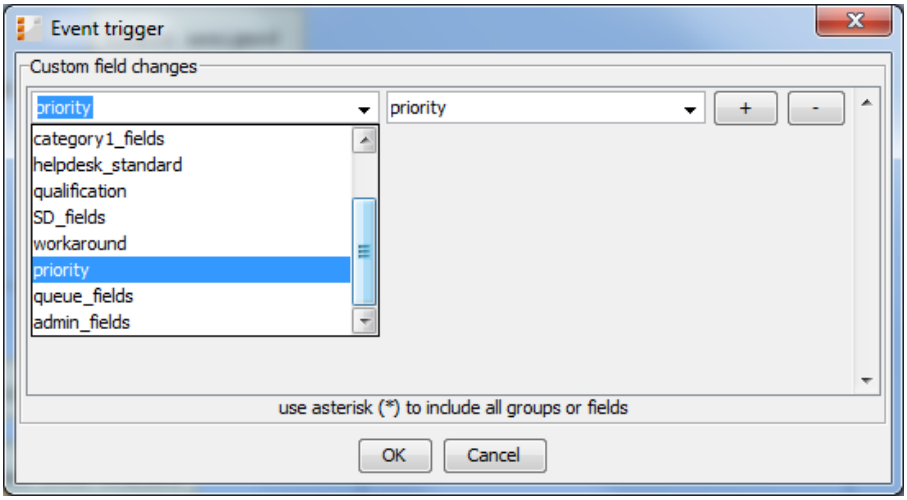


Fig. 3: ConSol*CM Process Designer - Property "custom field" of a Business Event Trigger

- **script after event**
  Here you can define a script (using the ConSol*CM Script Editor) which should be executed when the business event trigger has fired. It has to return *true* or *false*. When it returns *true*, the event is really fired, i.e. the automatic activity behind the business event trigger is executed. In case the script returns *false*, the event is blocked and the automatic activity is not executed. That way you can exactly control when the action (activity) should be performed, e.g. the trigger reacts to a change of the priority but should only really fire when the new priority is *Extra High*. Then the script checks the new priority and only when the new value is *Extra High* the script returns *true*, for all other values it returns *false*.

> ⚠ **Attention:**
>
> The *script after event* is only used to control and fine-tune the firing of the business event trigger! Every action which should be performed when the trigger has fired has to be located in an automatic activity behind the trigger! This guarantees a good process logic and helps visualize the process in the Process Designer.

Business Logic of Business Event Triggers

Firing Order of Serialized Business Event Triggers

When an event has occurred which is relevant for a business event trigger, this trigger fires. Then the *script after event* is executed. If it returns *true*, the following automatic activity or decision node with two following automatic activities is executed.

If the engineer changes more than one ticket parameter and different business event triggers have been defined for those parameters at the scope, the business event triggers fire according to their order at the scope.

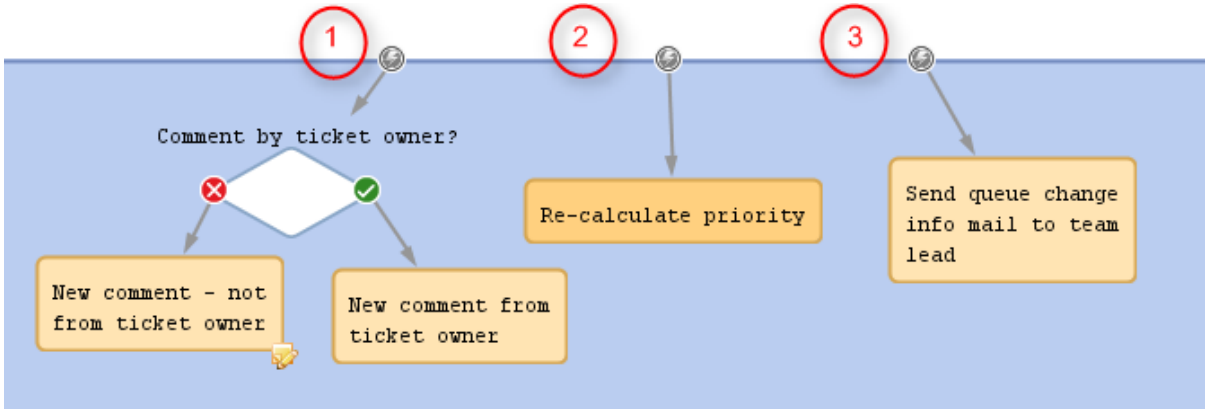


Fig. 4: ConSol*CM Process Designer - Firing Order of Business Event Triggers (1)

If one of the business event trigger actions leads the ticket to a new destination (i.e. it is no longer in the scope where the next business event trigger would be located), the following business event trigger is not fired. In the example in the next figure, business event trigger **(3)** will not be fired, if the *Re-calculate priority* trigger **(2)** has been fired (see *Use Case 2* in section Examples for Business Event Triggers), because the subsequent actions lead the ticket to another queue.

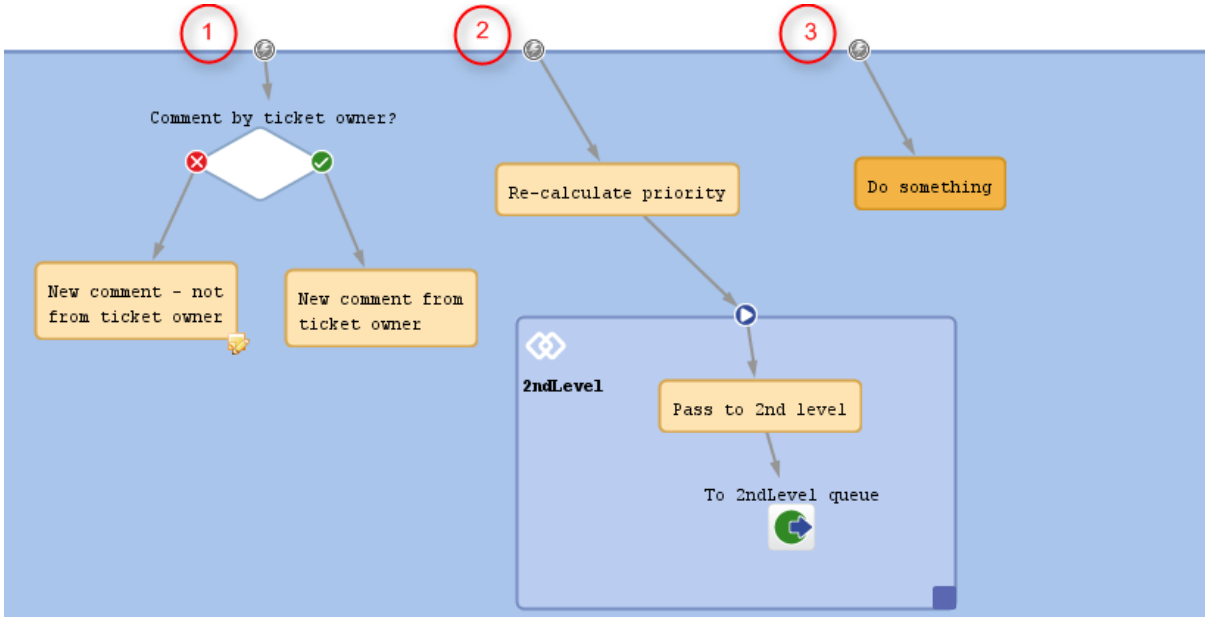


Fig. 5: ConSol*CM Process Designer - Firing Order of Business Event Triggers (2)

Firing Order of Business Event Triggers in Hierarchical Scopes

In case there are business event triggers in hierarchical scopes, the event is *consumed* by the innermost business event trigger, i.e. by the business event trigger of the innermost scope. All events which have not been consumed there, are further processed by the next outer scope, then the next and so on.

Case 1

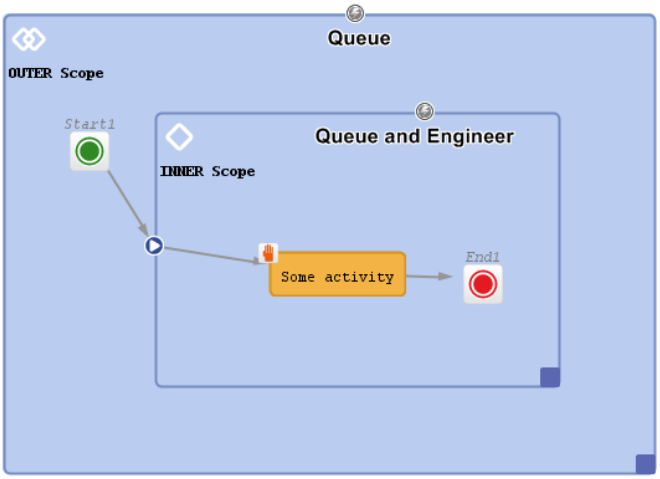


Fig. 6: ConSol*CM Process Designer - Hierarchical Business Event Triggers (1)

Fired events:

Events	Triggers fired
Queue	Inner
Queue and Engineer	Inner for both
Engineer	Inner

Case 2

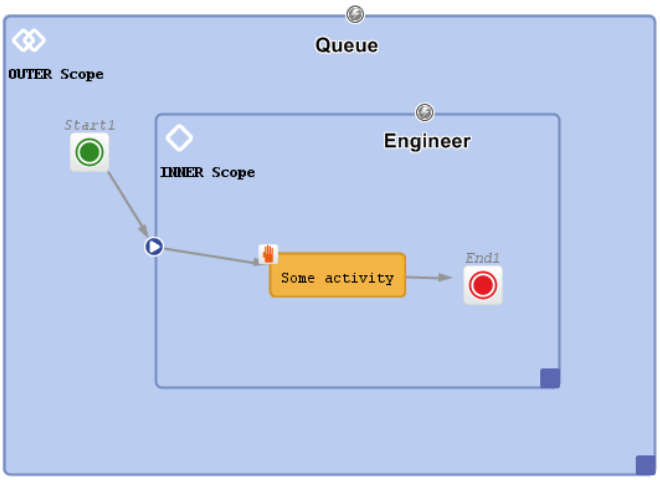


Fig. 7: ConSol*CM Process Designer - Hierarchical Business Event Triggers (2)

Fired events:

Events	Triggers fired
Queue	Outer
Engineer	Inner
Queue and Engineer	Inner and Outer

Case 3

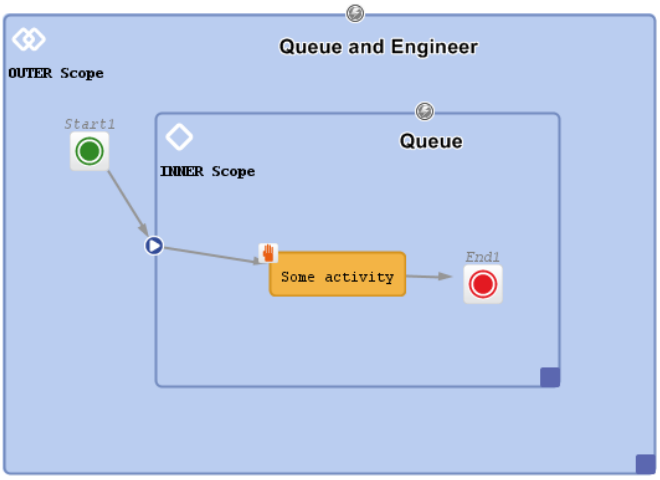


Fig. 8: ConSol*CM Process Designer - Hierarchical Business Event Triggers (3)

Fired events:

Events	Triggers fired
Queue	Inner
Engineer	Outer
Queue and Engineer	Inner (queue) and Outer (engineer only)

Examples for Business Event Triggers

Use Case 1: Check Engineer Comment

If a new comment has been added to the ticket by someone else, not by the current engineer (the ticket owner), then an overlay should be attached to the ticket icon. That way the ticket is marked and the engineer can see in the ticket list that there is a new comment in one of his/her tickets. The comment can be made by another engineer who has writing access to the queue or by a customer who can add comments using ConSol*CM/Track access. Or an e-mail might have been received.

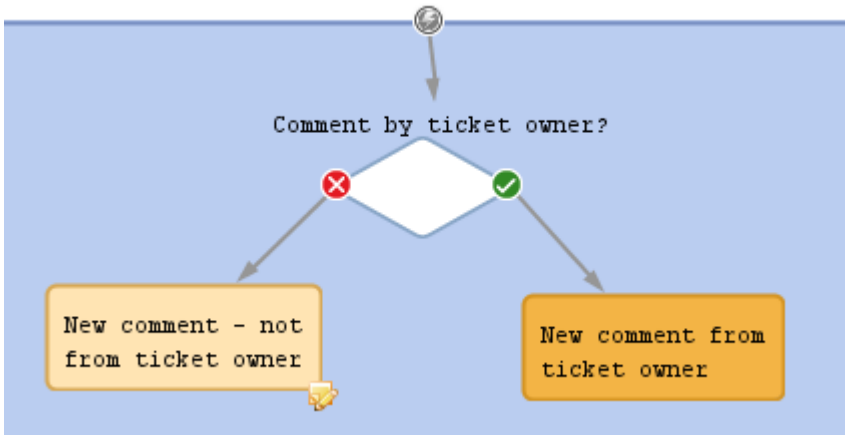


Fig. 9: ConSol*CM Process Designer - Business Event Trigger with Following Activities

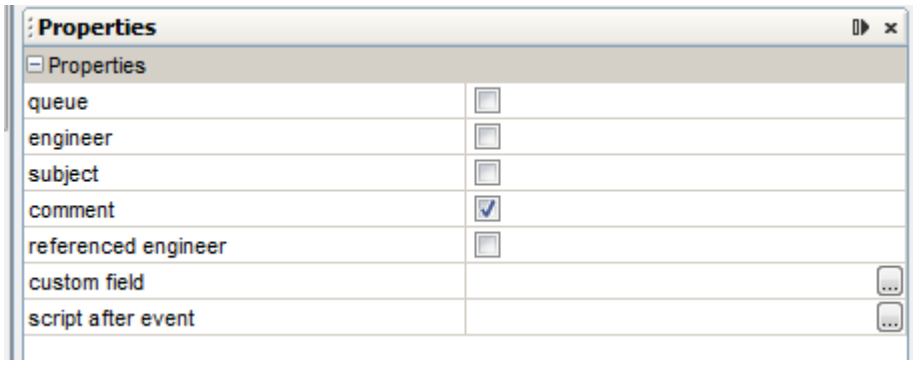


Fig. 10: ConSol*CM Process Designer - Properties of a Business Event Trigger (1)

Code of decision node script

```
return (workflowApi.getCurrentEngineer() == ticket.getEngineer())
```

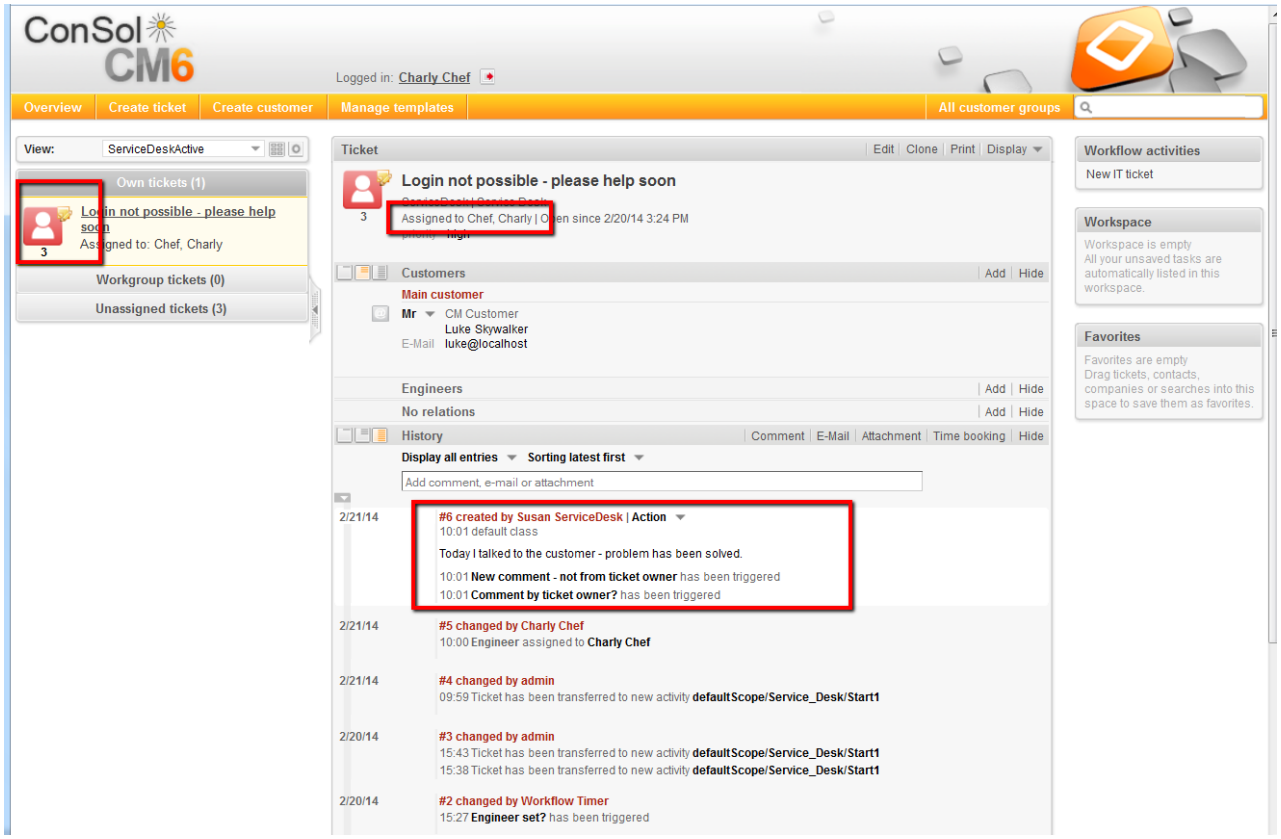


Fig. 11: ConSol*CM/Web Client- Ticket Marked with New Overlay

Use Case 2: Re-Calculate the Ticket Priority if Impact and/or Urgency Have Been Changed

This is an example from an *ITIL Service Desk* environment. According to the *ITIL* standards, the ticket priority is calculated from two values: impact and urgency. That means, in the ticket there are two fields which can be modified by the engineer and the priority is calculated automatically from the two values. The priority might then be displayed as ticket color or as read-only list (or both).

This principle requires a re-calculation of the priority in case at least one of the two fields (impact/urgency) has been changed. This is achieved using a business event trigger with an adjacent activity where the re-calculation is performed.

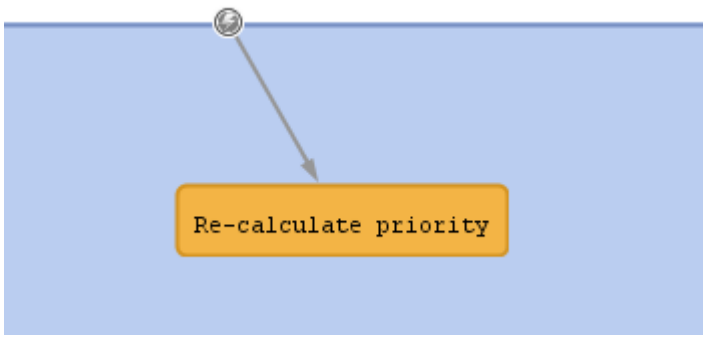


Fig. 12: ConSol*CM Process Designer - Business Event Trigger with Following Automatic Activity

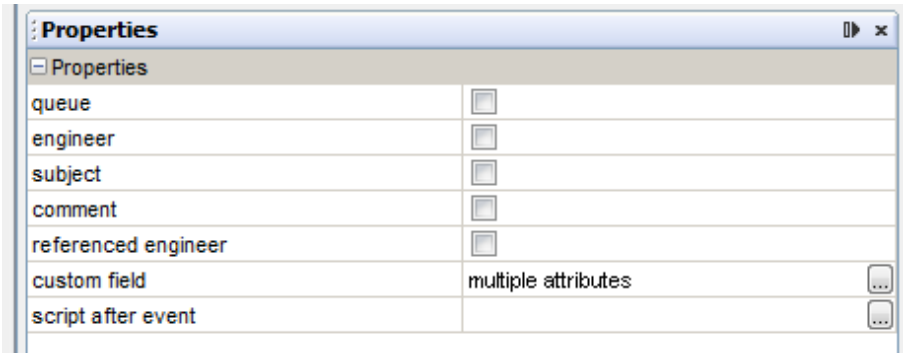


Fig. 13: ConSol*CM Process Designer - Properties of a Business Event Trigger (2)

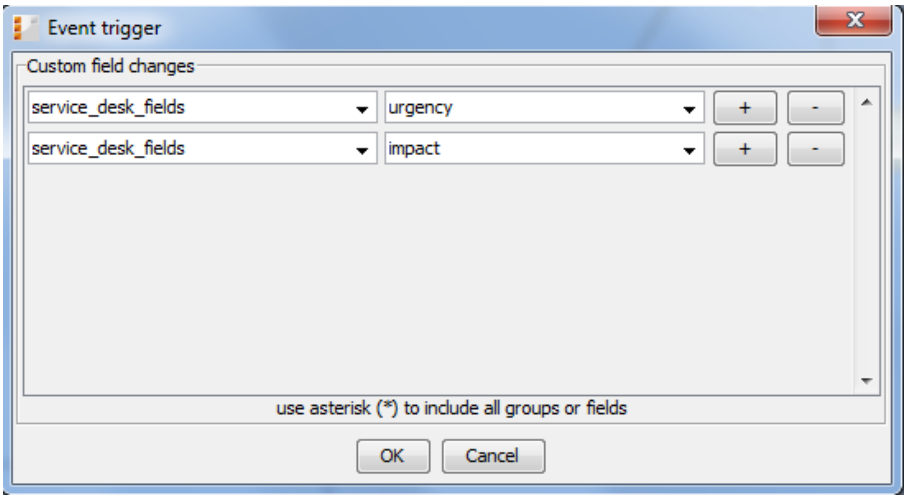


Fig. 14: ConSol*CM Process Designer - Property "custom field" of a Business Event Trigger (2)

Code of automatic activity script Re-calculate priority

```
// Re-calculate priority:
String imp_value = ticket.get("service_desk_fields.impact").getName()
String urg_value = ticket.get("service_desk_fields.urgency").getName();

ScriptProvider scriptProvider =
scriptProviderService.createDatabaseProvider("calculatePriority.groovy")
//content of calculatePriority.groovy is omitted here, because it is not relevant for the
current context
```

Use Case 3: Continue Delivery Process When Shipment for the Order Has Arrived

This is an example taken from a shipment and delivery process: new components (e.g. hardware) are ordered. The ticket waits in the scope *Order: Waiting for shipment*. When the shipment has arrived, an engineer of another team registers this shipment and sets the *Shipment received* tag. This change of ticket data (*Shipment received* from *false* to *true*) is registered by the business event trigger which listens to the respective *boolean* value (the check box). After the business event trigger has fired, the check box is checked (in the decision node), and when the value is set to *true*, the ticket is forwarded to the next scope *Deliver components*. The engineers who are responsible for the delivery now see the ticket in their view *Components ready for delivery* and can acknowledge the delivery when they are done with *All components delivered*.

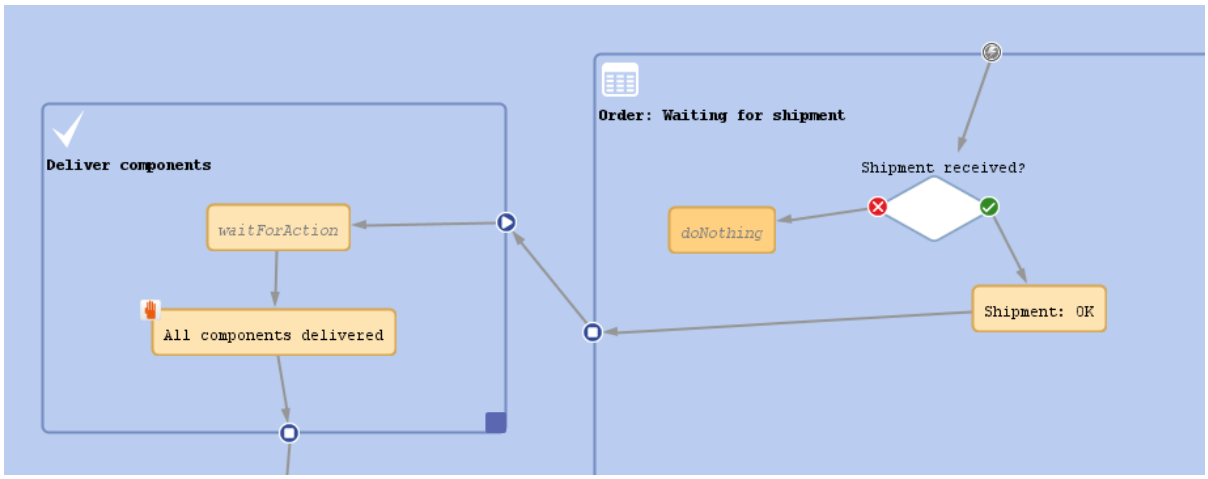


Fig. 15: ConSol*CM Process Designer - Workflow for Use Case 3

Best Practices: Using Business Event Triggers

See section Best Practices: Avoid Self-Triggering Business Event Triggers.

4.7.5 Activity Control Forms (ACFs)

Introduction to ACFs

An *Activity Control Form (ACF)* is a web form which is offered to the engineer at one or more process steps. In this way, the data input can be controlled in a very strict way.

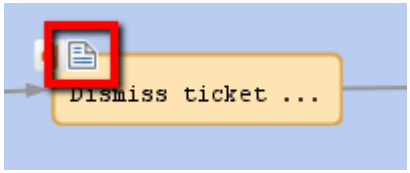


Fig. 1: ConSol*CM Process Designer - Activity Control Form (ACF)

For example, when a help desk agent wants to dismiss a complaint, this cannot be performed without giving a reason. In the process this is implemented using an ACF which is displayed when the engineer has clicked on the workflow activity *Dismiss complaint*. A form is opened where the engineer has to select a category for the dismissal and a text box where he/she can enter a note. Or, using the example of a sales process, when an engineer (a sales agent in this case) clicks on *Make appointment with potential customer*, a form is displayed, where the budget, the size of the customer's company, and the products of interest have to be entered.

An ACF can offer optional and mandatory fields.

Information:

We recommend to set a "..." behind the name of every activity which will automatically open an ACF. This helps the user to distinguish between ACF-loaded activities and simple activities.

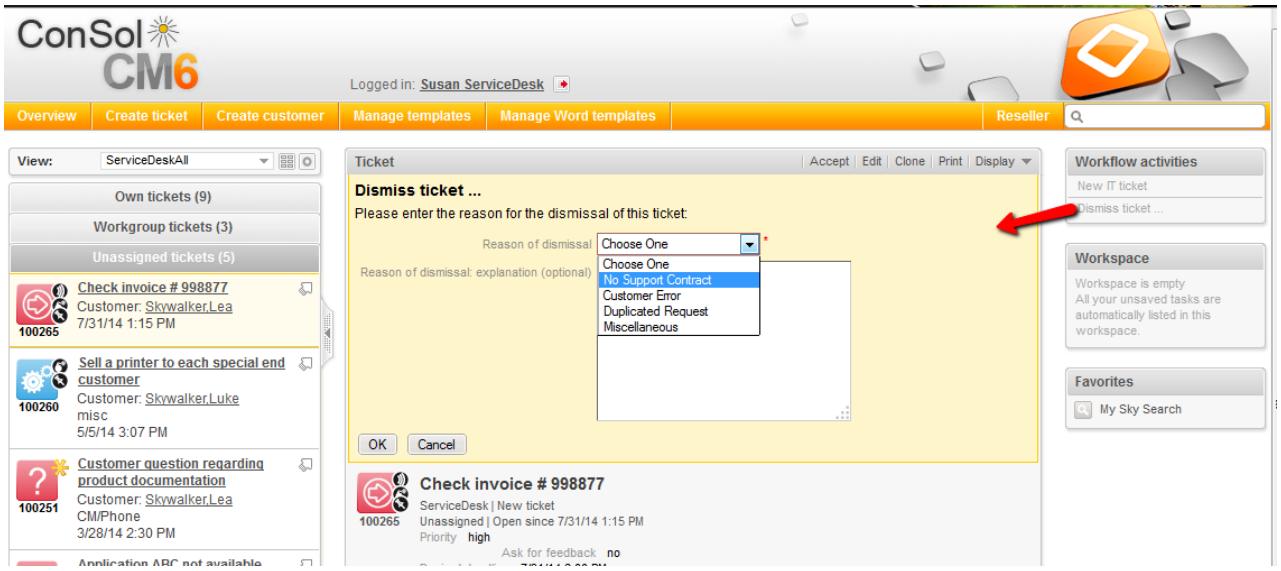


Fig. 2: ConSol*CM/Web Client - Opened ACF

Adding an ACF to a Workflow

Variant A: Starting the ACF Definition Using the Admin-Tool

Before you can add an ACF to the workflow, it has to be defined using the Admin-Tool. Please refer to the *ConSol*CM Administrator Manual*, chapter *Custom Field Administration* for a detailed explanation. Here, we assume you have already defined an ACF and want to add it to the workflow.

An ACF is always added to a manual activity. To add an ACF to the target activity, grab the ACF icon in the palette and attach it to the activity using drag-and-drop. Then you can configure the ACF properties. In case you add an ACF to an automatic activity, this activity is changed to type *Manual*.

In the Web Client, the ACF will be opened when the user clicks on the workflow activity to which the ACF is attached in the workflow. See figure above.

Variant B: Starting the ACF Definition Using the Process Designer

You can also add an empty ACF to a workflow activity and define the name during this operation. Then an empty ACF will be created in the Admin-Tool and you have to assign the custom fields to this ACF in a later step.

> **Attention:**
>
> Do not forget to reload the Admin-Tool data! When you have defined the ACF in the Process Designer, there is no automatic data transfer to the Admin-Tool.

Properties of an ACF

These are the properties of an ACF:

- **name**
  The name of the ACF. Select the name from the drop-down menu. All ACFs which have been defined in the Admin-Tool are available.
- **required fields**
  This opens a pop-up window (see figure below) where you can define mandatory fields. As a default, all ACF fields are optional, i.e. when the form is opened in the Web Client, the user can enter data but can also continue the process without doing so. For mandatory fields, the process can only be continued when the field has been filled.
- **Script**
  Here, you can define a script which will be executed before the ACF is loaded. Usually, this kind of script is used to set default values in ACF custom fields.

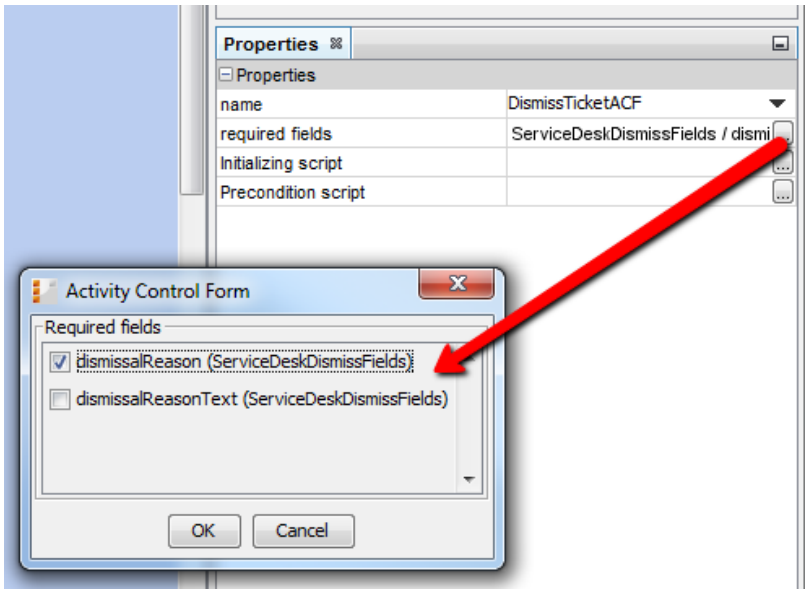


Fig. 3: ConSol*CM Process Designer - Properties of an ACF

Attention:

All custom fields which are part of an ACF have to be available in the target queue, i.e. the respective custom field group (CF group) has to be assigned to the queue where the workflow is used! There are two possibilities to achieve that:

1. You assign the CF group to a queue manually.
2. You just create the ACF and use it in a worflow. When you deploy the workflow, ConSol*CM will automatically assign the required CF groups to the queues where the workflow is used.

For a detailed explanation of queue management, please see the *ConSol*CM Administrator Manual*.

Business Logic of ACFs

ACF at Manual Activity

ACFs are only possible for manual activities. When a user selects a workflow activity in the Web Client, the ACF script is executed (if there is a script). Then the ACF is opened in the Web Client (with optional and mandatory fields). If fields, which are part of the ACF, are also available in the regular ticket data fields, those fields might have been edited/filled-in by an engineer before the ACF is used. Thus those fields might be already filled-in in the ACF. The engineer can leave them as-is (and use the ACF as control only) or can modify the content of the fields.

If the data of the ACF should not be shown before a certain step in the process has been reached, the data can be put into one (or more) separate custom field group(s) which are *invisible* at the start of the process. In the step after the activity with the ACF, the custom field groups are faded in using the script of a workflow activity. Please refer also to the Best Practices section in this manual for more recommendations concerning the use of ACFs.

When an ACF is canceled, it returns to the scope of the last activity, because the ticket always waits **behind** the last activity (and **not** before the next).

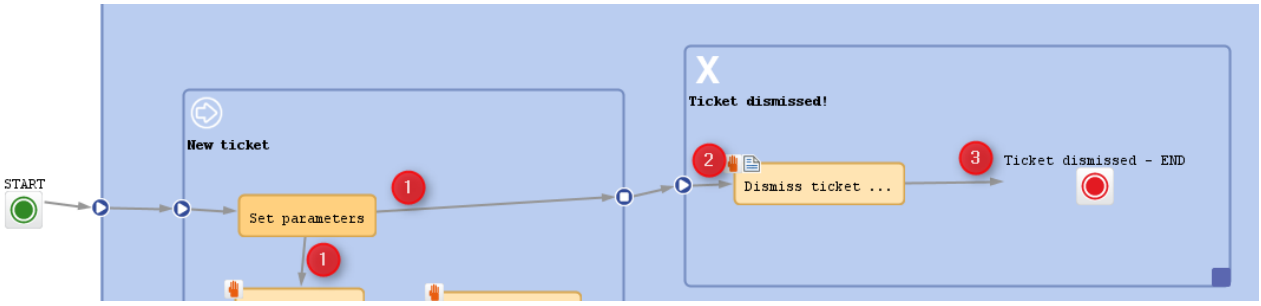


Fig. 4: ConSol*CM Process Designer - ACF Process Logic

Example:

- A ticket is created and runs through the automatic activity *Set parameters*.
- It waits behind this activity, at position **(1)** in the scope *New ticket*. The next activities *Dismiss ticket ...* and *New IT ticket* (not shown here) are displayed in the Web Client.
- The engineer selects *Dismiss ticket ... .*
- The script for the ACF at *Dismiss ticket ...* is executed **(2)**.
- The ACF is shown in the GUI.
    1. **Variant 1:**
        a. The ACF is canceled.
        b. The ticket goes back to **(1)**.
    2. **Variant 2:**
        a. The ACF is filled-in and confirmed.
        b. The activity *Dismiss ticket ...* is executed (in case there is a script in this activity, the script is executed), the ticket passes through the node and continues on its way **(3)**. In the example above, it is closed.

ACF at Manual Activity with Condition

In case a manual activity has a condition, the activity is only displayed if the condition script returns *true*, i.e. also the ACF is only displayed if the condition script returns *true*.

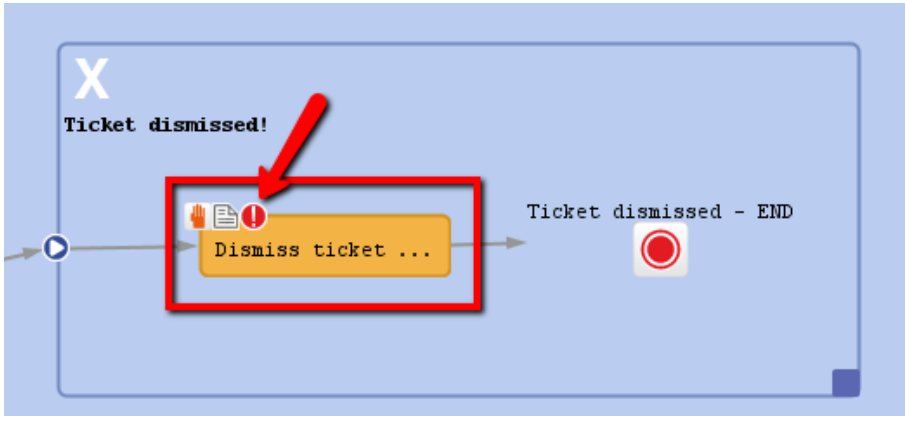


Fig. 5: ConSol*CM Process Designer - Manual Activity with ACF and Condition

Examples for the Use of ACFs

Use Case 1: ACF for the Dismissal of a Customer Request

This example was used in the previous sections. The engineer can only dismiss a customer request when a reason has been given. This is selected from a drop-down menu. Additionally, the engineer can add a note in a text field.

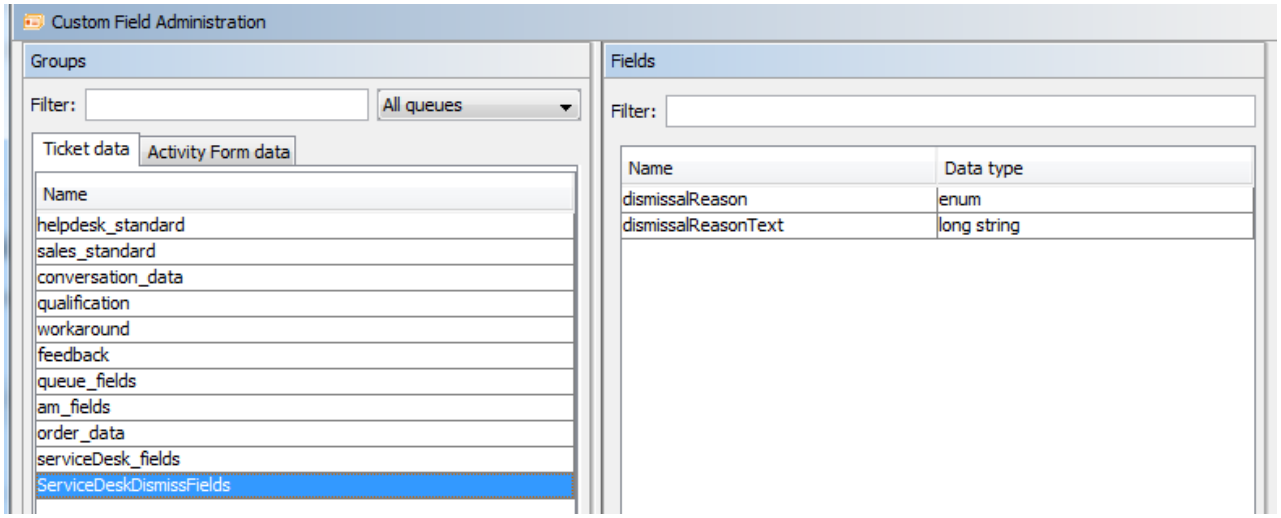


Fig. 6: ConSol*CM Admin-Tool - ACF Definition

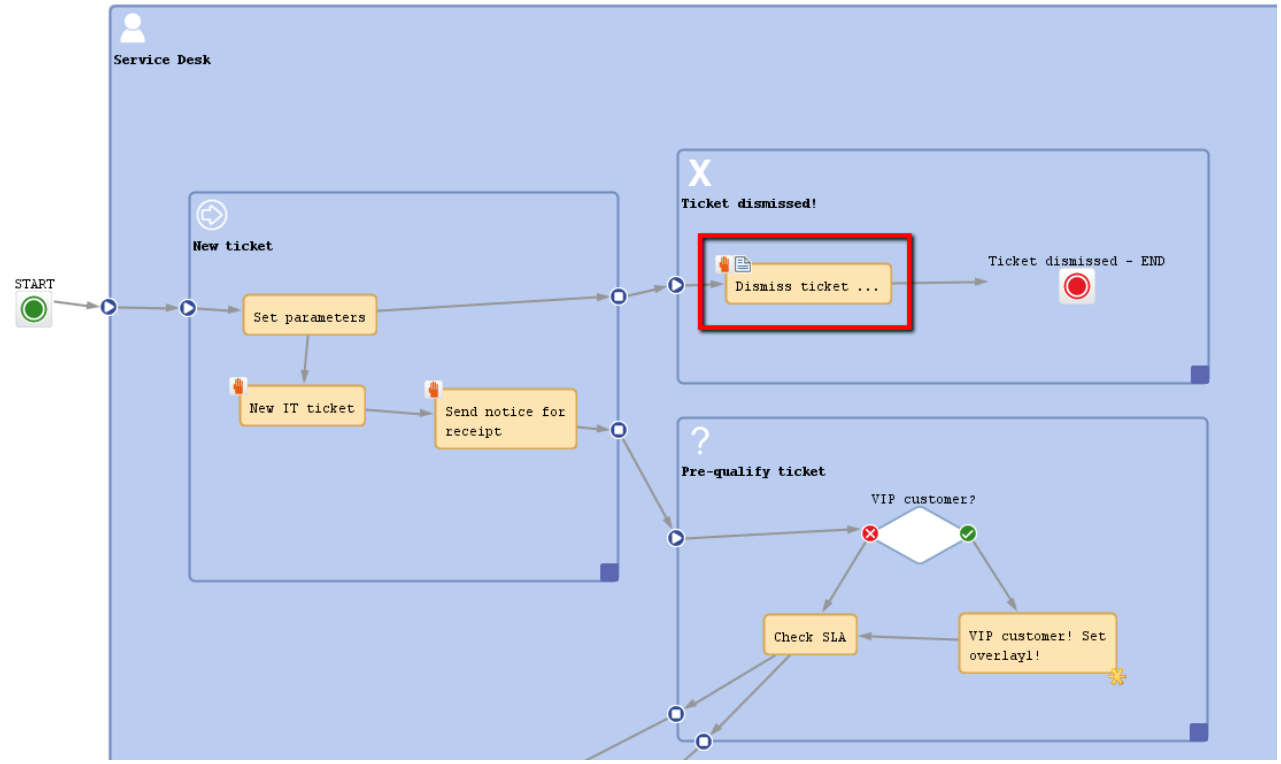


Fig. 7: ConSol*CM Process Designer - ACF in Workflow

The Web Client GUI and the ACF properties are shown in the figures of the previous paragraphs.

Use Case 2: Fill-in Sales Information when Bid is Created

When a sales representative selects the workflow activity *Create bid* in the Web Client GUI, an ACF is opened where several fields are offered. One field is a drop-down menu and a default value is set via script. The other fields are optional. The field *Product* has been filled-in for the ticket in previous process steps, so this field is offered with the selected value. It can either be left unchanged or it can be modified.

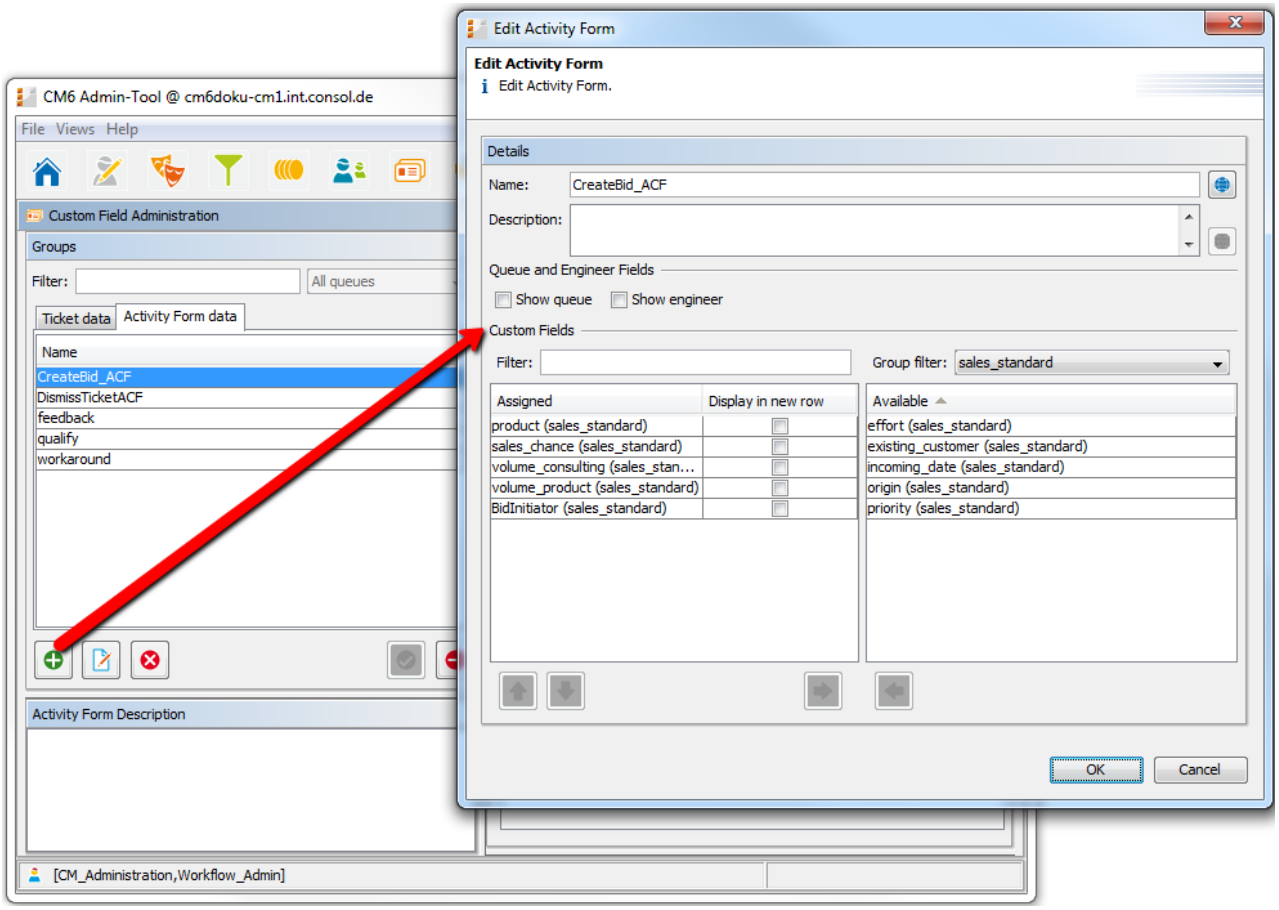


Fig. 8: ConSol*CM Admin-Tool - ACF for Sales Workflow

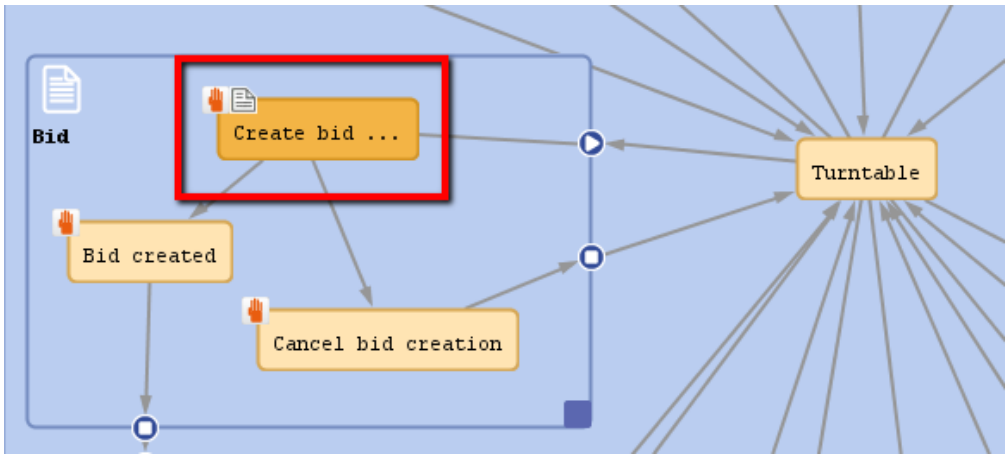


Fig. 9: ConSol*CM Process Designer - ACF in Sales Workflow

Process Designer: Initializing Script for Create Bid ACF

```
ticket.set("sales_standard.BidInitiator","Mr. Miller")
```

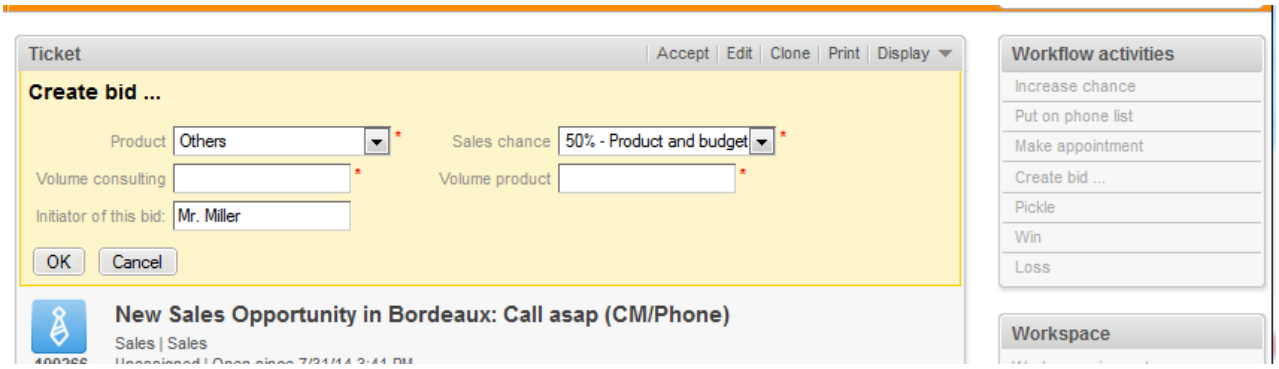


Fig. 10: ConSol*CM/Web Client - Sales Process ACF

4.8 Jump-out and Jump-in Nodes

4.8.1 Introduction

A process often consists of one or more sub-processes, e.g. in an IT help desk, there might be a first level team who accepts and qualifies the tickets, a second level team who can solve several problems, and some third level team with specialists. When you want to represent this process, you have to build a workflow for each special sub-process (1st level, 2nd level, 3rd level). Then the sub-processes have to be linked to make sure the handover of the ticket from one team to the next uses the correct way in the process.

A ticket might pass from the first level to the second level, on to a third level team, back to the second level team with another question, back to another third level team, and then back to the first level team who contacts the customer. So we need connections from one sub-process to the next one, i.e. nodes where a ticket leaves the present workflow, a **jump-out node**, and the counterpart in the following workflow, the **jump-in node**. If the ticket should start at the *START* node of the new process, no jump-in node is required.

In the Process Designer, jump-out and jump-in nodes are inserted into the workflow by drag-and-drop from the palette and are linked to other workflow elements depending on the desired process.

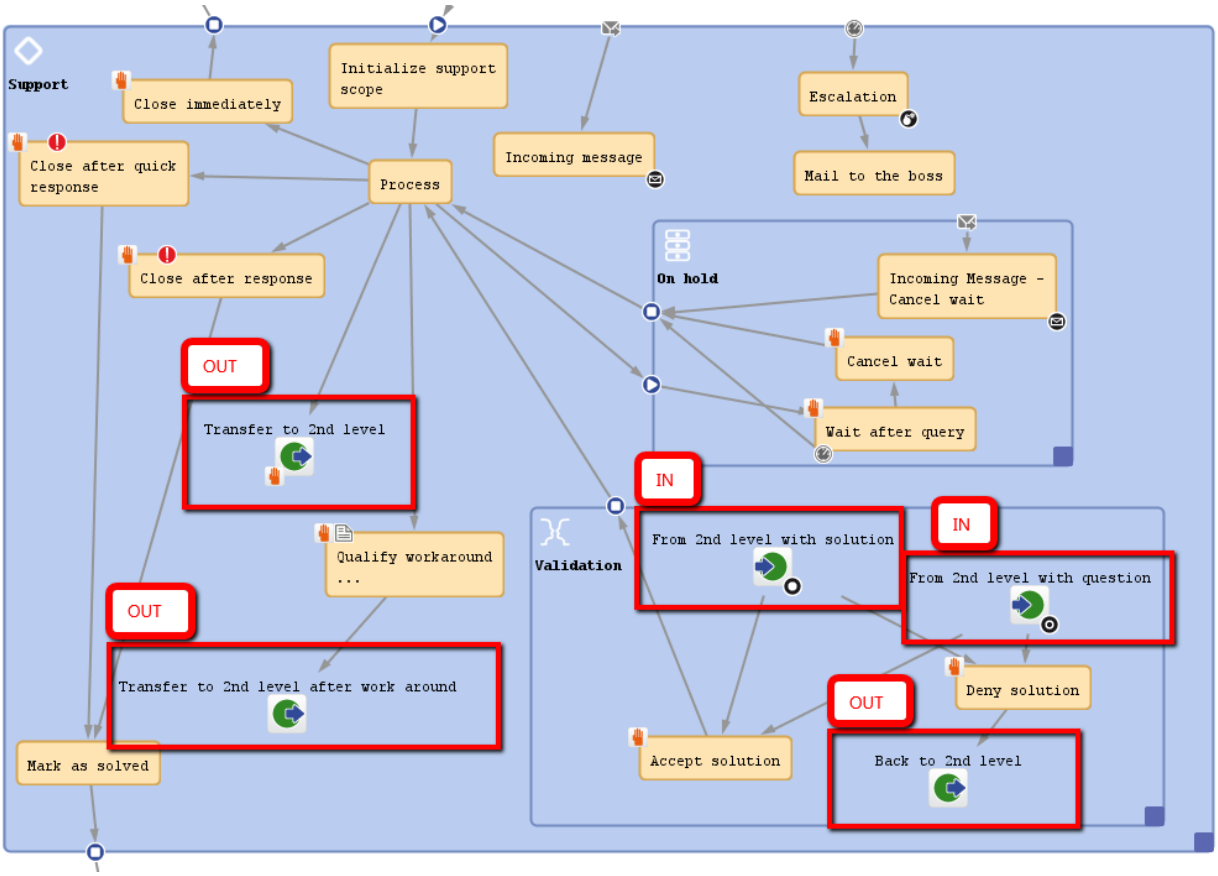


Fig. 1: ConSol*CM Process Designer - Example for Jump-out and Jump-in Nodes

4.8.2 Jump-out Nodes

A jump-out node defines a position where the ticket is to leave the (sub-)process and to enter the next (sub-)process.

Fig. 2: ConSol*CMProcess Designer - Jump-out Node

Properties of a Jump-out Node

For a jump-out node the following properties can be defined:

- **name**
  Mandatory. Technical object name.

- **label**
  Optional. Localized name (if not set, the technical name is used) that will be displayed in the Web Client GUI.
- **description**
  Optional. It will be displayed as mouse-over in the Web Client GUI.
- **sort index**
  Defines the order of the activities in the Web Client GUI.
- **jump out node type**
  Mandatory. Either *Automatic* or *Manual* has to be selected. In case it is a manual node, the node is marked with the *hand/manual* icon in the Process Designer GUI.
- **script**
  Optional. A script can be defined which is executed when the ticket enters the node.
- **target queue name**
  Select the queue name to which the ticket should be passed.
- **target jump in node**
  Select the jump-in node from the drop-down menu. All jump-in nodes from the workflow of the selected queue are offered. If no jump-in node is selected, the ticket will enter the other process, i.e. the target queue, at the *START* node.

> **Information:**
>
> When you start designing workflows you might have a *chicken-and-egg* problem when you start to define jump-out and jump-in nodes, because obviously you will have to start with one workflow when the other workflow is not yet present. We recommend to work with dummy queues without specific jump-in node. Then add the correct target queue name and the name of the jump-in node later.

- **history visibility**
  See section history visibility
- **disable auto update**
  See section disable auto update

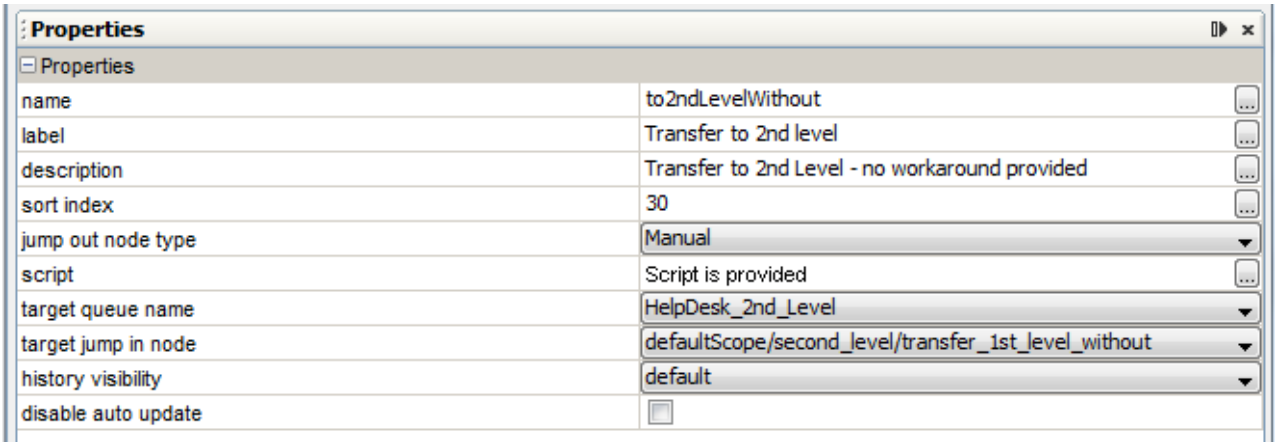


Fig. 3: ConSol*CM Process Designer - Jump-out Node: Properties Editor

4.8.3 Jump-in Nodes

A jump-in node is a node which defines the position where a ticket from another process (queue) can enter a queue with the current workflow. All jump-in nodes of a workflow are offered as target jump-in nodes when the queue with the respective workflow has been selected as target queue for a jump-out node.

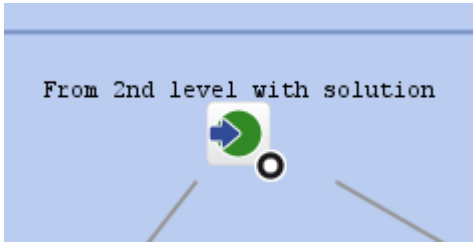


Fig. 4: ConSol*CM Process Designer - Jump-in Node

Properties of a Jump-in Node

For a jump-in node the following properties can be defined:

- **name**
  Mandatory. Technical object name.
- **label**
  Optional. Localized name (if not set, the technical name is used) that will be displayed in the Web Client GUI.
- **description**
  Optional. It will be displayed as mouse-over in the Web Client GUI.
- **script**
  Optional. A script can be defined which is executed when the ticket enters the node.
- **overlay**
  Optional. Click into the orange space to load a standard ConSol*CM overlay or use the file explorer (...) for an upload of another icon from the file system.
- **overlay range**
  Only displayed when overlay has been set.
  - **Activity**
    The overlay is attached only as long as the ticket stands behind the activity. As soon as the next activity is executed, the overlay is deleted from the ticket icon.
  - **Scope**
    The overlay is deleted when the ticket leaves the scope.
  - **Process**
    Once the overlay has been attached to the ticket icon, it stays there for the rest of the process.
  - **Next overlay**
    The overlay is attached to the ticket icon as long as no new overlay appears. In that case, only the new one is attached, the old one is deleted.
- **history visibility**
  See section history visibility.
- **disable auto update**
  See section disable auto update.

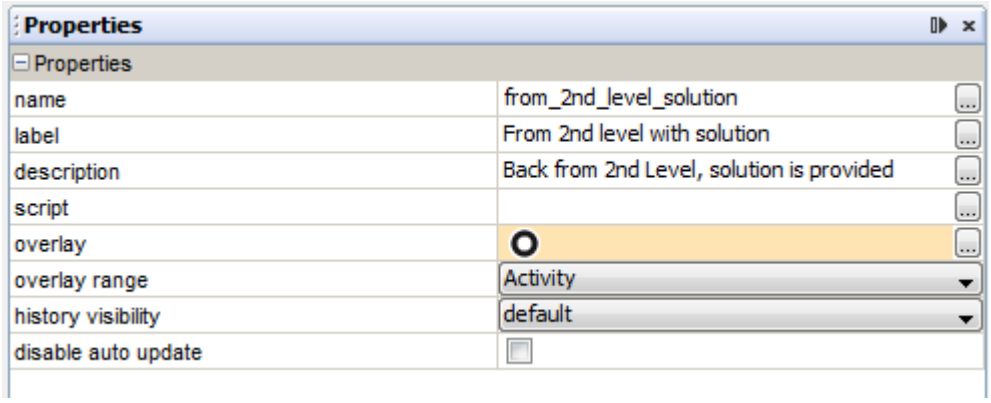


Fig. 5: ConSol*CM Process Designer - Jump-in Node: Properties Editor

5 Process Logic

When you create and modify workflows it is important to know the basic principles of the workflow engine which result in the behavior of the ticket during the process. Therefore, we will give you a short overview of the basic rules of ConSol*CM ticket processing.

5.1 Activities

Basic rules:

- Passing through a workflow, a ticket always waits **behind** the last activity, **not** before the next!
- Then it looks for the next activity which can be executed/passed.
- If the next possible activity is a manual activity, the ticket stays at the position behind the previous activity (number **(1)** and **(2)** in the following figure).
- If the next possible activity is an automatic activity, the activity is executed, i.e. the ticket passes through this activity (number **(3)** in the following figure).
- An activity can have **one or more manual** activities as successor activities **or** an activity can have (only) **one automatic** activity as successor activity.
- When you save a workfow, the Process Designer automatically executes a consistency check. If there are any inconsistencies (e.g. two automatic activities), an error message is displayed and the workflow cannot be saved.

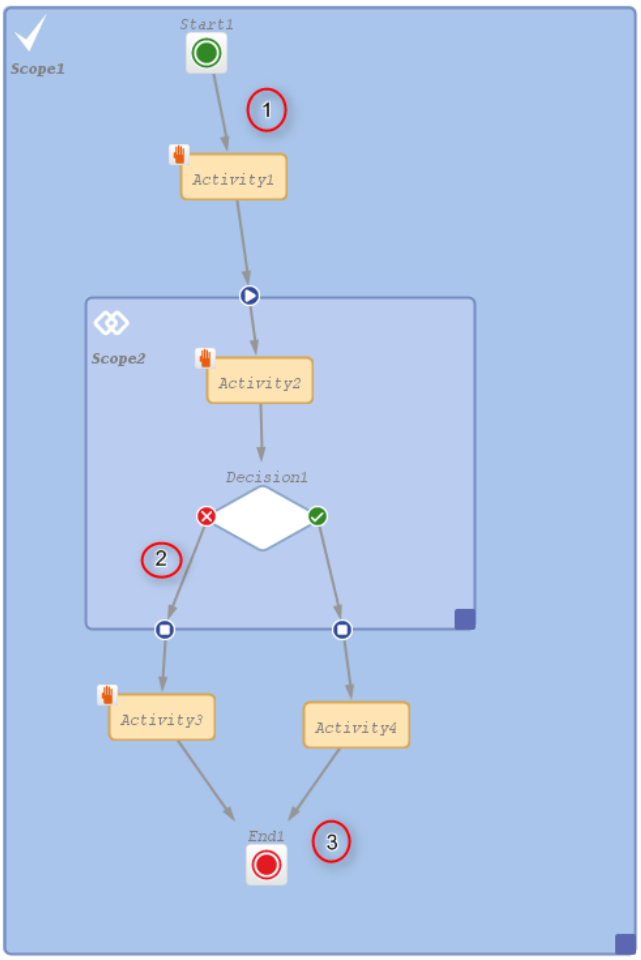


Fig. 1: ConSol*CM Process Designer - Process Logic 1

5.2 Interrupts and Exceptions

In the course of a process, i.e during the time when the ticket is open and engineers work on it, there might be events which have to be taken care of. For example, when an e-mail is received by the ticket or when a time range for an SLA has run out, it is important to register the event and to react accordingly.

There are two ways to define the reaction and behavior of the tickets. You can implement an ...

- **interrupt**
  This is a workflow architecture where the event is registered, one or more automatic activities are executed, and the ticket returns to its previous position in the workflow.
- **exception**
  This is a workflow architecture where the event is registered and, due to the following manual or automatic activities, the ticket leaves its previous position and is taken to a new position within the workflow or in another workflow.

5.2.1 Interrupts

Interrupts ...

- are activated by triggers.
- cause a short interruption of the process to react to the trigger event.
- use automatic activities (one or more subsequent automatic actions).
- put the ticket back to its previous position in the workflow, i.e. back to the position where it was when the interrupt event has fired.
- are often used to mark the ticket icon with an overlay, e.g. when an e-mail has been received (see figure below) or when an escalation time has been reached.

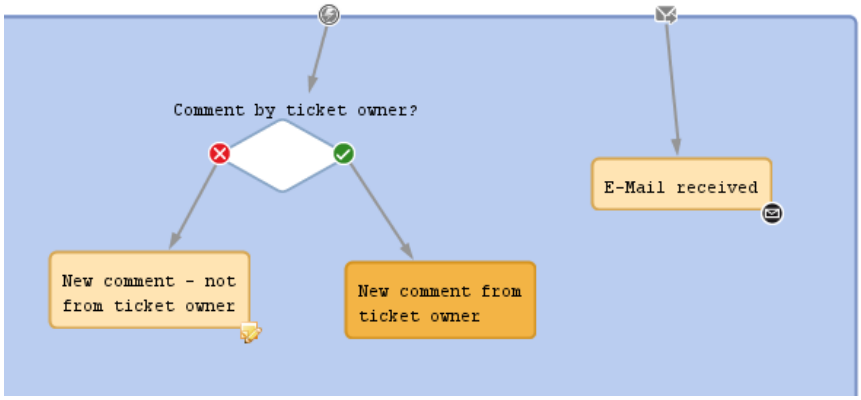


Fig. 2: ConSol*CM Process Designer - Two Interrupts

5.2.2 Exceptions

Exceptions ...

- are activated by triggers.
- move the ticket from its old position in the workflow to a new position. The latter can be in the same or in another workflow.
- cause the process to continue at the new position.

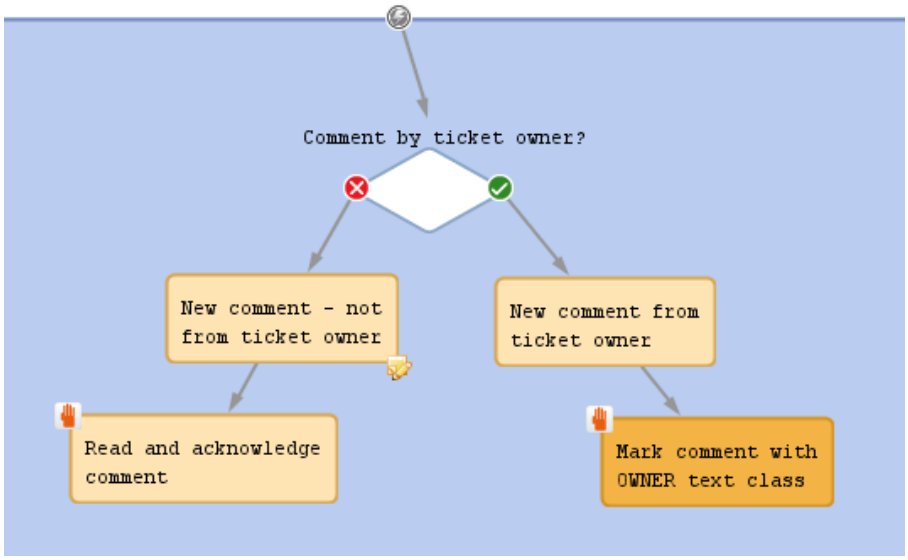


Fig. 3: ConSol*CM Process Designer - Exception

5.3 Loops (Errors in Workflows)

(Infinite) Loops will cause errors in a process. They cannot be detected by the Process Designer, so you could deploy a workflow which contains a loop as shown in the figure below.

However, the process engine detects such loops at run-time and throws an *InfiniteWorkflowLoopException* to prevent the complete system failure. You can of course see the exception in the *server.log* file. In the Web Client, an error message is displayed.

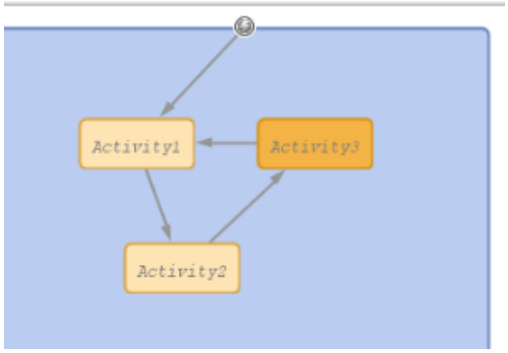


Fig. 4: ConSol*CM Process Designer - Loop in Workflow

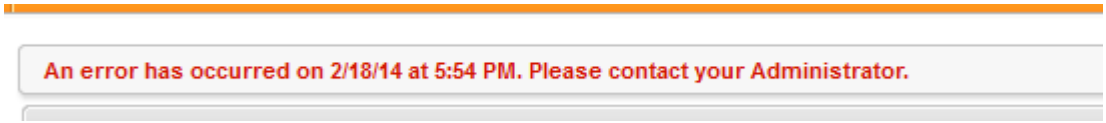


Fig. 5: ConSol*CM/Web Client - Error Message when Loop Was Detected

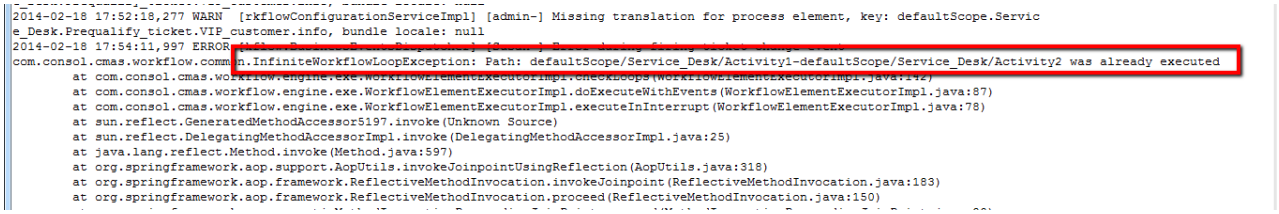


Fig. 6: Console - File server.log: Error Message Caused by Workflow Loop

Business event triggers can also cause loops when the automatic activity which is attached to the trigger changes the parameter to which the trigger reacts. See section Best Practices - Avoid Self-Triggering Business Event Triggers.

5.4 Process Logic of Time Triggers

See section Time Trigger Business Logic.

5.5 Process Logic of Business Event Triggers

See section Business Event Trigger Business Logic.

6 ConSol*CM Process Designer Manual - Workflow Programming

6.1 Workflow Programming

6.1.1 Introduction

The process logic of ConSol*CM workflows is implemented using the two basic pillars of ConSol*CM process intelligence:

1. The logic which results from the order of scopes, activities, and other **workflow elements**.
2. The **workflow scripts** (which contain the *real intelligence*).

So far in this manual, we have concentrated on explaining the workflow elements which can be implemented using the graphic-driven functionalities of the Process Designer. In the following chapter, we will provide a deeper insight in workflow construction and will explain workflow programming.

You should have a basic knowledge of *Java and Groovy* programming, because ConSol*CM scripts are written in Groovy. We will not provide an introduction to programming in general.

In ConSol*CM workflows, scripts are used in the following contexts:

- As activity script for an activity.
- As precondition script for an activity which has to return *true* or *false*.
- As script for a decision node which has to return *true* or *false*.
- As script for a business event trigger which is executed when the trigger has fired.
- As script for a time trigger
  - which is executed when the time trigger is initialized, i.e. when the ticket enters the scope where the time trigger is attached.
  - which is executed when the time trigger fires, i.e. when the defined time has elapsed.
- As script for end nodes.
- As script for jump-in or jump-out nodes.
- As scripts for ACFs.

Please refer to the respective sections in this manual for an explanation how to insert the scripts.

6.1.2 Additional Tools for Workflow Programming

To write scripts for workflow elements, you use the Workflow Script Editor which has been explained in section The Script Editor.

As an important tool you will also use the *ConSol*CM Java API documentation*. Please ask your ConSol* sales representative or CM consultant to receive the respective *JAR* file. It is a standard Java API Doc, so as an experienced Java programmer you will get along quickly.

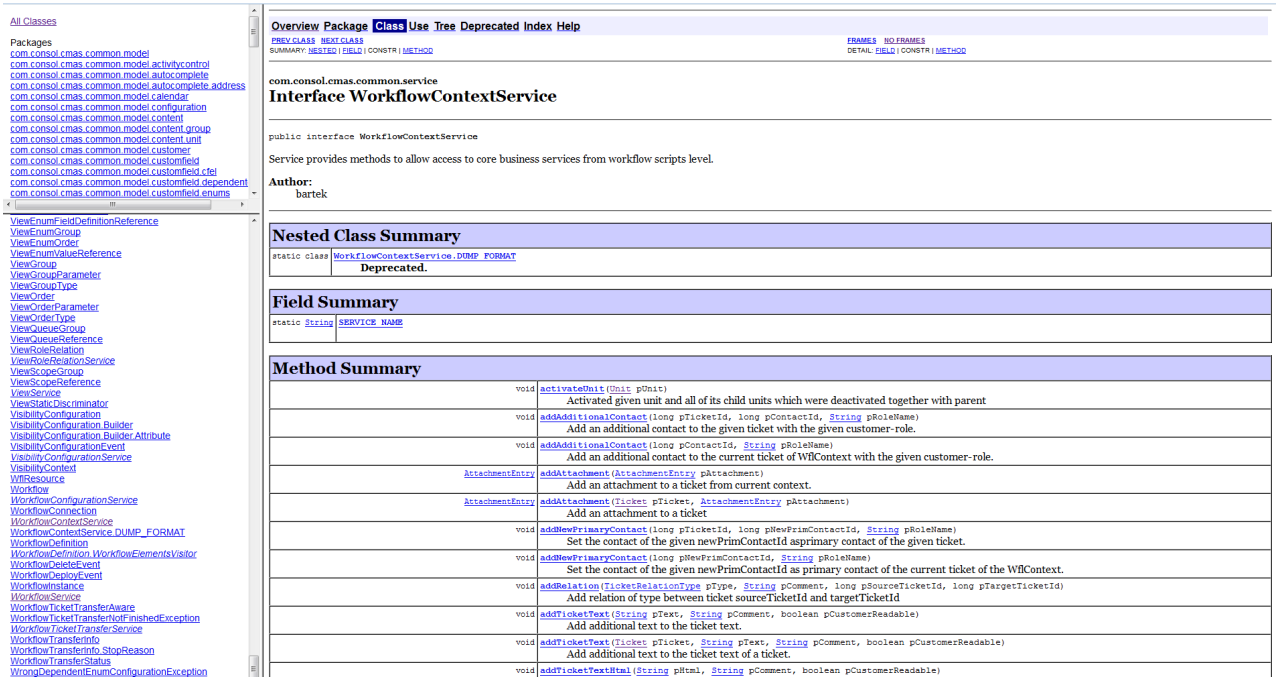


Fig. 1: ConSol*CM Java API Doc

6.1.3 Notes About Method Syntax

As mentioned above, you have to use Groovy syntax for ConSol*CM scripts. There might be different possibilities to express or code the same content. In the following paragraphs we will give you some hints and provide some examples how to work with the Groovy API.

Getter Methods Can Often Be Omitted

Most Java objects possess numerous *getter* methods to retrieve values from object attributes. In ConSol*CM you can either use the complete *getter* methods, or you can use the short (convenience) form. Please see the following examples for workflow scripts.

Use case	Java-like syntax (extended version)	Groovy syntax (short version)
Get the subject of a ticket.	String mysubject = ticket.getSubject()	def mysubject = ticket.subject
Get the engineer of a ticket.	Engineer myeng = ticket.getEngineer()	def myeng = ticket.engineer
Get the main contact of a ticket.	Unit mymaincontact = ticket.getMainContact()	def mymaincontact = ticket.mainContact

Use case	Java-like syntax (extended version)	Groovy syntax (short version)
Get the value of a certain custom field from a ticket.	String myprio = ticket.get("helpdesk_fields", "prio")	def myprio = ticket.get("helpdesk_fields.prio")
Get the unit type for the primary contact.	Unit mycustomer = workflowApi.getPrimaryContact() UnitDefinition myunitdef = customer.getDefinition() UnitDefinitionType mydeftype = myunitdef.getType()	def mycustomer = workflowApi.primaryContact def myunitdef = customer.definition def mydeftype = customer.definition.type

Access to custom fields cannot be shortened, because there are no getter methods for those fields. Please read the section Working With Data Fields for details about working with data from custom fields.

Setter Methods Can Often Be Omitted

Most Java objects possess numerous *setter* methods to set values for object attributes. In ConSol*CM you can either use the complete *setter* methods, or you can use the short (convenience) form. Please see the following examples for workflow scripts.

Use case	Java-like syntax (extended version)	Groovy syntax (short version)
Set the subject of a ticket.	ticket.setSubject("asd")	ticket.subject = "asd"

6.2 Important Classes and Objects

6.2.1 Introduction

To make ConSol*CM script programming easier, the CM Workflow API provides easy access to objects which are frequently used. Furthermore, convenience classes and methods provide a short way to various objects and methods.

6.2.2 Important Objects

Some objects are implicitly present in workflow scripts.

> ⚠ **Attention:**
>
> The same objects are **not** present in Admin-Tool scripts, i.e. within Admin-Tool scripts you will have to use import statements!

Ticket

In every workflow script, the current ticket can be easily accessed by the object *ticket*. It is derived from the class *Ticket* and is implicitly present. No import and no instantiation is required.

Example:

Using the ticket object

```
def myId = ticket.getId()
```

workflowAPI

The object *workflowApi* is also implicitly present. It provides easy access to the interface *WorkflowContextService* which is used for numerous operations.

Examples:

Using workflowApi to send an e-mail

```
workflowApi.sendEmail(contact_e,subj,text,replyto,null)
```

Using workflowApi to assign a ticket to current engineer

```
def curr_eng = workflowApi.getCurrentEngineer()
ticket.setEngineer(curr_eng)
```

Using workflowApi to deactivate a trigger

```
workflowApi.deactivateTimer("defaultScope/Service_Desk/TimeTrigger1")
```

Using workflowAppi to display a GUI message for the engineer/user

```
workflowApi.addValidationError("1", "The ticket cannot be closed before a solution is provided.
Please fill-in solution and mark it with text class SOLUTION first.") }
```

6.2.3 Convenience Classes and Methods

The ConSol*CM API provides various convenience interfaces and methods which make access to most objects of every-day CM programming a lot easier. Most of those convenience interfaces are part of the package *com.consol.cmas.common.service* and its sub-packages. Please refer to the *ConSol*CM Java API documentation* for details. Here, we will show you some examples which might prove useful for most CM programmers.

The implementing instance of the interface is always available by replacing the first letter, which is a capital letter, in the class name by a lower case one, e.g. the object (singleton) with the interface *EngineerService* is available with the object *engineerService*, see *Example 2*.

Example 1: Using ConfigurationService to Retrieve System Properties

Using ConfigurationService to retrieve number of engineer management ticket

```
def tic_nr =
configurationService.getValue("custom-mycompany-properties","engineer_management.ticket.nr")

// then: ... do something with the engineer management ticket, e.g. find out the name of the
next engineer a service ticket should be assigned
```

Using ConfigurationService to retrieve base URL of the system

```
def baseUrl = configurationService.getValue("custom-mycompany-properties","base.url.mycompany")
def url = baseUrl + "/cm-client/ticket/ticket_name/" + ticket.getName()
def itComplete = url + " " + ticket.getName()

// ... do something with the ticket url, e.g. place a link to a child ticket in a table of the
parent ticket
```

Example 2: Using EngineerService to Assign the Ticket to an Approver

Example with use of EngineerService

```
// Script does the following:
// Hand-over ticket to approver only when approver has been set in ticket as additional engineer

// Import package, because classes are not available in workflow otherwise:
import com.consol.cmas.common.model.ticket.user.function.*

// Get the name of the approver which has been written/stored in a custom field,
// namely the field with the name „CF_ApproverName“ in the custom field group
// „CF_GroupApproverData“. The value could be for example „Mr. Miller“:
def gen = ticket.get("CF_GroupApproverData.CF_ApproverName").getName()

// Get the engineer object where the name „Mr. Miller“ is set, i.e. the engineer
// object of the desired approver:
def gen_eng = engineerService.getByName(gen)

// Get the ticketFunction object which represents the ticketFunction (engineer role) „Approver“:
TicketFunction tf = ticketFunctionService.getByName("Approver")

// Add the engineer object of Mr. Miller as Approver. i.e. in the ticketFunction
// (engineer role) „Approver“ to the ticket. One of the paramaters is ticket. This
// does not have to be instantiated, because it is implicitly present in workflow scripts:
def tu = ticketUserService.addTicketUser(ticket, gen_eng, tf, "Approver")

// Assign the ticket to the engineer, i.e. set the engineer Mr. Miller also as ticket owner.
def tic2 = workflowApi.assignEngineer(ticket, gen_eng)
```

We have two assignments here:

1. Mr. Miller is set as additional engineer in the engineer role *Approver*.
2. Mr. Miller is set as ticket owner.

Example 3: Using EnumService to Retrieve an Enum Value by Name

Using EnumService to retrieve an enum value by name

```
def enumValueMLA = enumService.getValueByName( "priority", "REGULAR" )
ticket.set( "helpdesk_fields.prio", enumValueMLA )
```

Example 4: Using TicketService to Retrieve all Tickets of a Certain View

Using TicketService to find ticket of a view

```
List<Ticket> mylist = ticketService.getByView(new ViewCriteria(
    viewService.getByName("helpdesk_active_tickets"),
    ViewAssignmentParameter.allAssignedTickets(),
    ViewGroupParameter.allTickets(),
    viewOrderParameter.addByName(true)))
```

Example 5: Using EngineerRoleRelationService to Send an E-Mail to All Engineers of a Role

Using EngineerRoleRelationService to send an e-mail to all engineers of a role

```
// Send e-mail to all engineers of a regular role

def mail = new Mail()

mail.setTo(engineerRoleRelationService.getEngineersWithRoles(roleService.getByName("Supervisor"
))*.email.join(","))

mail.setSubject("Ticket (${ticket.name}) -- Escalation!")

mail.setText(workflowApi.renderTemplate("Ticket escalation note to supervisor"))

mail.send()
```

6.3 Working With Data Fields

6.3.1 Introduction to Data Fields

The access to data fields is an essential part of ConSol*CM programming. It is potentially required in all scripts of the system, workflow as well as Admin-Tool scripts, no matter of which type. Here, we will set the focus on workflow programming, but the access to data fields is basically the same in all scripts.

ConSol*CM Version 6.8 and Older

In ConSol*CM versions 6.8 and older, all data fields are called **custom fields** (CFs). CFs are used to define the CM data model which consists of **ticket data** and of **customer data**. The layout of the Web Client is also defined by the help of CFs using special annotations (e.g. *position*).

Examples for custom fields for tickets are:

- priority of the ticket
- escalation date due to an SLA
- printer model
- contract number

Examples for custom fields for customer data are:

- customer name
- zip code
- phone number
- e-mail address

For a detailed introduction to the work with custom fields for ticket data, please refer to the *ConSol*CM Administrator Manual 6.8*, section *Custom Field Administration*.

Rules for work with custom fields CM 6.8 and older:

When you work with custom fields, there are two main rules you have to keep in mind:

1. Custom fields are always managed and referenced in custom field groups, e.g. when you want to retrieve the value of a CF, you use <CF GroupName>.<CF FieldName>
2. You always use the technical unique name to reference a CF or a CF group, not the localized value.

ConSol*CM Version 6.9 and Higher

Starting with ConSol*CM version 6.9.0, there are two types of data fields:

- **custom fields**
  Used to define ticket data, managed in custom field groups, as known from previous CM versions.
- **data object group fields**
  Used to define customer data as part of the *FlexCDM*, the new customer data model. Managed in data object groups.

The work with custom fields of the new (version 6.9 and higher) customer data model (FlexCDM) is explained in detail in the *ConSol*CM Administrator Manual - Customer Data Model 6.9: FlexCDM* and in the *ConSol*CM Administrator Manual (Version 6.9)*, section *The CM Customer Data Model: FlexCDM*.

Rules for work with custom fields CM 6.9 and higher:

When you work with custom fields and data object group fields, there are three main rules you have to keep in mind:

1. Custom fields are always managed and referenced in *custom field groups*, e.g. when you want to retrieve the value of a CF, you use <CF GroupName>.<CF FieldName>
2. Data object group fields are always managed and referenced in *data object groups*, e.g. when you want to retrieve the value of a data object group field, you use <Data Object GroupName>:<Data Object Group FieldName>
3. You always use the technical unique name to reference a data object group field or a data object group, not the localized value.

6.3.2 Data Types for Data Fields

A data field is always of a certain data type. As for any variable in programming, it depends on the data type how you have to handle the value of the field, e.g. a *string* field cannot be used for calculating numbers, an *enum* field needs a specific access method.

The following data types are available in ConSol*CM:

- **boolean**
  Values: *true* or *false*. Depending on the annotation *boolean-type*, the value is displayed as a check box, radio buttons, or a drop-down list.
- **date**
  Format and accuracy can be set by annotations.
- **enum**
  For sorted lists. The engineer can choose one of the enum values on the Web Client. Enums and values have to be created previously within the *Enum Administration* in the Admin-Tool (see *ConSol*CM Administrator Manual*).
- **list**
  A data field of this type is the basis for a *list* (one column) or a *table* (multiple columns) of input fields in the Web Client. A table contains lines, each of data type *struct* (see below). Each line (struct) contains individual data fields. A simple list consists of a *list* field which contains the custom fields.
- **struct**
  A data field of this type defines a data structure (line of a table) which groups one or multiple field(s).
- **number**
  For integer values.
- **fixed point number**
  For numbers with a fractional part, e.g. currencies. You have to enter the total number of digits (*Precision*) and the number of digits that fall to the right of the decimal point (*Scale*).

- **string**
  For up to 4000 alphanumeric characters.
- **long string**
  For large objects, unrestricted length.
- **short string**
  For up to 255 alphanumeric characters.
- **contact data reference** (up to version 6.8)
  Special data type used internally for referencing the contacts associated with a ticket. Additionally the *contact data type* (customer or company) has to be selected in the field below.
- **MLA field**
  This data type is used for custom fields that contain hierarchical lists with a tree structure called *MLA* (Multi Level Attributes). The name of the custom field will be the name of the new MLA that has to be defined within the MLA Administration. The group of the custom field has to be referenced when the MLA is created.

6.3.3 Custom Fields for Ticket Data

In the Admin-Tool, the custom fields for ticket data are defined in the *Custom Field Administration* section, file card *Ticket data*.

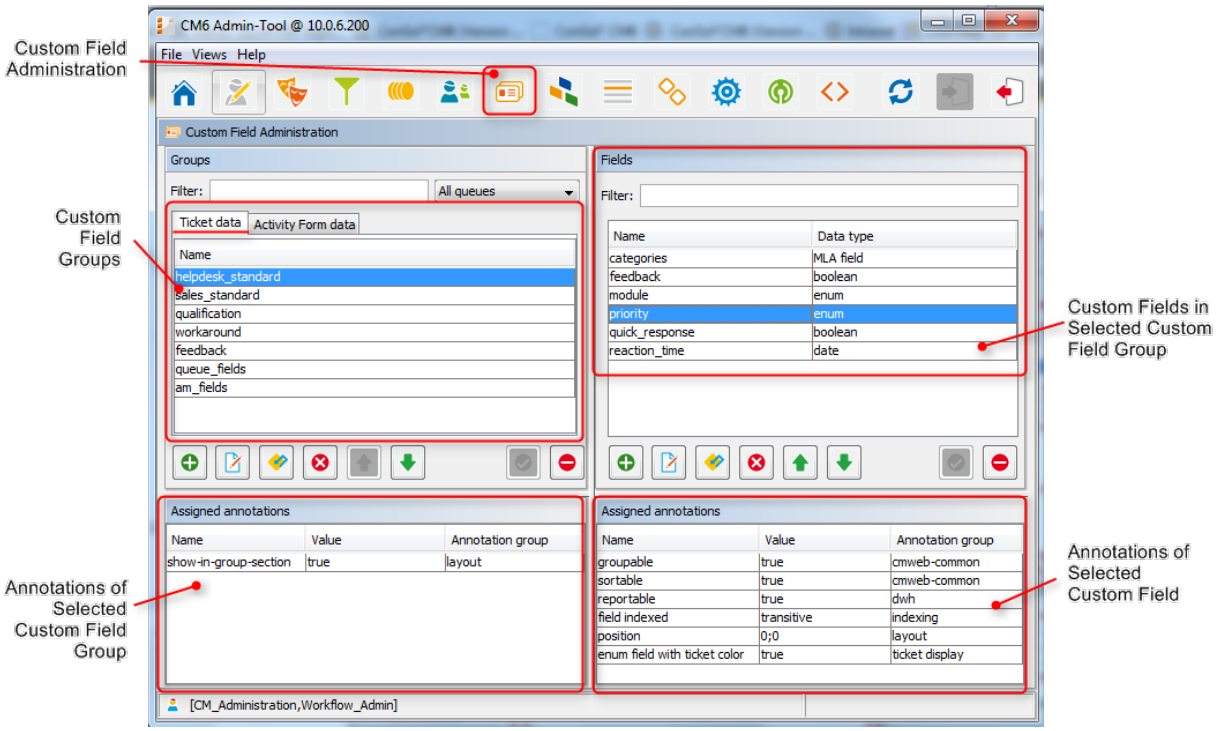


Fig. 1: ConSol*CM Admin-Tool: Custom Field Administration for Ticket Data

Most Important Methods for Access to Ticket Custom Fields

Three methods are of major importance for programming CF access in CM scripts. They all are methods of the class *Ticket*:

- **Ticket.get()**
  - For retrieving data from a CF.
- **Ticket.set()**
  - For setting data in an already existing CF.
- **Ticket.add()**
  - For calculating with a value within a CF, i.e. to add a certain time range to a *date* field.
  - For adding a new line in list fields (simple lists and tables).

Another method might be used when a field should be emptied, i.e. when its value should be set to *null*:

- **Ticket.remove()**
  - Sets the value of the field to *null*.

Retrieve Custom Field Values for Ticket Data

To retrieve data from a custom field in a script, you have to reference it by using the technical names of the custom field group and of the custom field.The method which has to be used can vary depending on the data type of the CF.

Simple Data Types

The following examples refer to the custom fields in the figure above.The method which should to be used (because it is the most convenient way) is:

```
ticket.get("<Group_name>.<CF_name>")
```

Attention:

Please keep in mind that the *getter* method for attributes will return the attribute (an object) and not the value of the object!

For example:

ticket.getField("helpdesk_standard","reaction_time") will return an AbstractField.

When you want to work with the value of the field use:

def myvalue = ticket.get("helpdesk_standard", "reaction_time")

Or:

def myfield = ticket.getField("helpdesk_standard", "reaction_time");

def myvalue = myfield.getValue();

Best:
(the version we recommend for standard use)

def myvalue = ticket.get("helpdesk_standard.reaction_time")

Retrieve value of boolean CF

```
def fedb = ticket.get("helpdesk_standard.feedback")
// will return TRUE or FALSE or NULL because it is a BOOLEAN field
```

A precondition script of a workflow activity could look like the following code:

Precondition script where boolean value is checked

```
boolean vip_info = ticket.get("am_fields","vip");

if(vip_info == true){
  return true;
}
else {
  return false;
}
```

Or shorter:

Precondition script where boolean value is checked, short version

```
return ticket.get("am_fields.vip")
```

Enum Values

An enum (ordered list) field is a field where the value is one of various list values. For example, a list with priorities is the basis for an enum field. To retrieve the value of an enum field, you can use the same syntax as for simple data types. The *get* method provides the enum list value, the *getName()* method provides the *string* attribute with the name of the value.

Retrieving an enum value for a CF

```
def prio = ticket.get("helpdesk_standard.priority")
println "Priority is now " + prio.getName()
```

Lists

Lists of Simple Data Types

A list of simple data types consists of a list (= array) which has a value of a simple data type in each line, a *date* in our example.The CF of type *date* has to have the parameter *Belongs to* which points to the list.

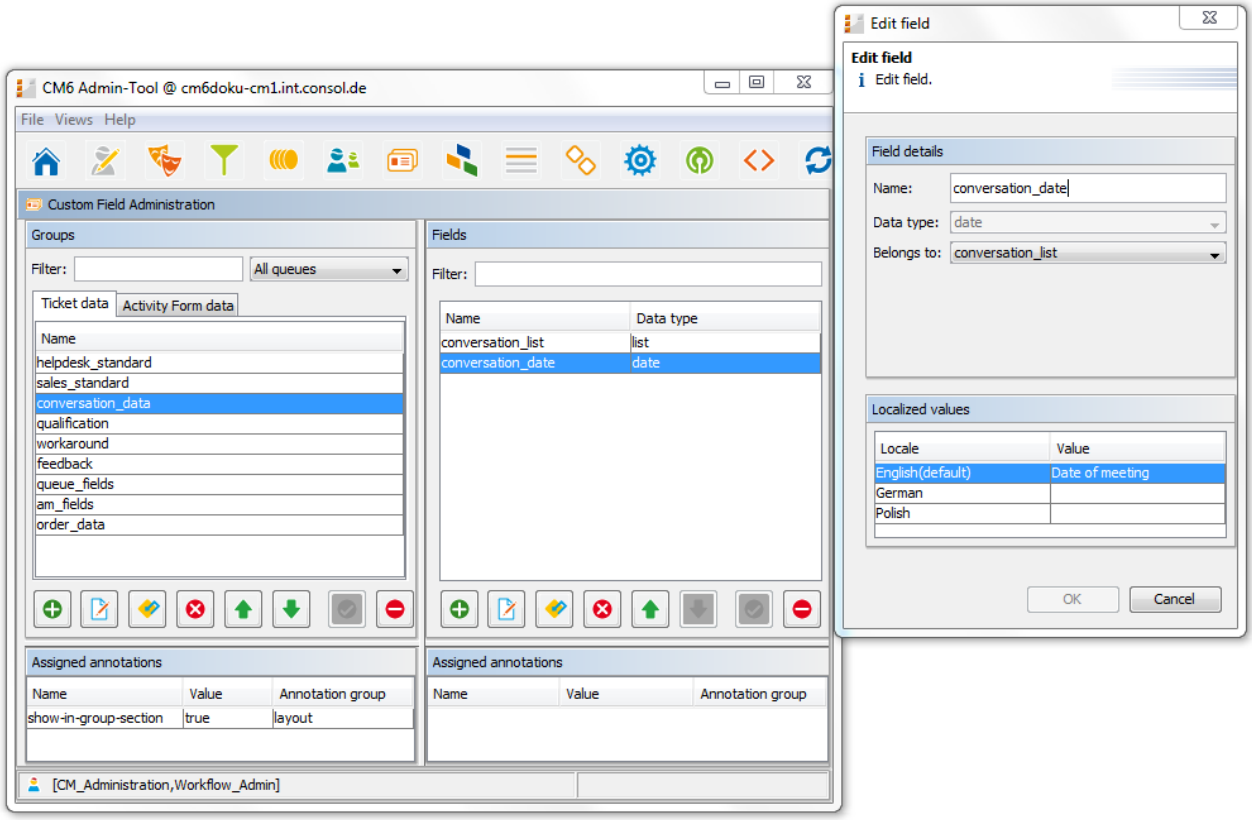


Fig. 2: ConSol*CM Admin-Tool - CFs for a List of Date Fields

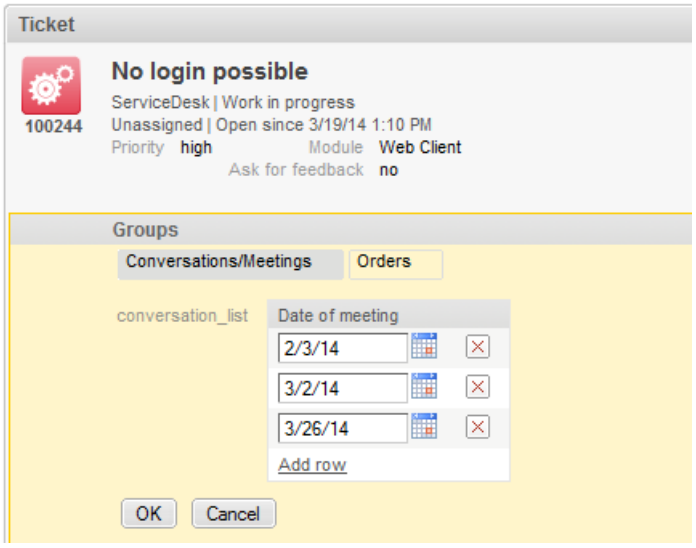


Fig. 3: ConSol*CM/Web Client - List of Date Fields in a Ticket (Edit Mode)

For access to each *date* CF within a list use the following lines of code:

Displaying the content of a list of date objects

```
def convs = ticket.get("conversation_data.conversation_list").each() { conv ->
    println "NEXT DATE is :" + conv
    println "CLASS of NEXT DATE is " + conv.getClass()
}
```

```
2014-03-27 12:21:45,274 INFO  [STDOUT                    ] NEXT DATE is :2014-02-03 00:00:00.0
2014-03-27 12:21:45,279 INFO  [STDOUT                    ] CLASS of MEXT DATE is class java.sql.Timestamp
2014-03-27 12:21:45,280 INFO  [STDOUT                    ] NEXT DATE is :2014-03-02 00:00:00.0
2014-03-27 12:21:45,280 INFO  [STDOUT                    ] CLASS of MEXT DATE is class java.sql.Timestamp
2014-03-27 12:21:45,281 INFO  [STDOUT                    ] NEXT DATE is :2014-03-26 00:00:00.0
2014-03-27 12:21:45,281 INFO  [STDOUT                    ] CLASS of MEXT DATE is class java.sql.Timestamp
```

Fig. 4: Log File - Output for Script

To access a certain line, you can use the following syntax:

Retrieve a certain value from a list of simple data types

```
def mydate = ticket.get("conversation_data.conversation_list[1]")
```

Lists of Structs (Tables)

The data construct *list of structs* is the technical basis for a table structure in the Web Client. The list is the parent object which contains lines. Each line is an instance of a struct. Each line (struct) contains as many custom fields (table columns) as required.

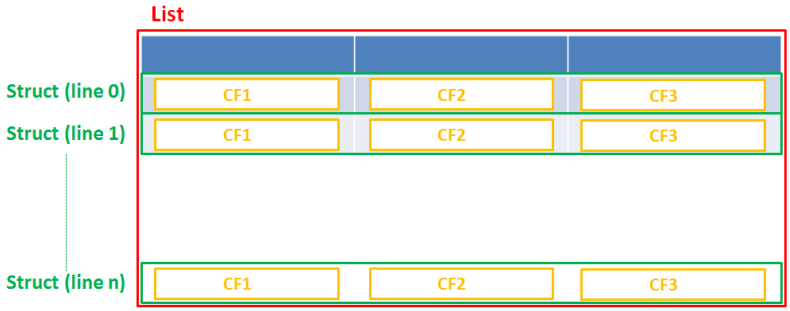


Fig. 5: List of Structs - Logical Principle

Technically spoken, the list is an array which contains a map (= key:value pairs) in each field.

Fig. 6: List of Structs - Technical Principle

To retrieve the data from a list of structs you can work with an iteration over the lines (= structs). In the following example (from an order system, not displayed in the figure above) we work with a table where ...

- the CF *orders_list* represents the list.
- the CF *orders_list* is located within the CF group *order_data.*
- the iterator *str* represents the struct.

- the struct has three fields:
  - *orders_hardware*
    which represents the article that should be ordered (*enum*).
  - *orders_contact*
    which represents the contact person (*string*).
  - *orders_number*
    which represents the number of articles that should be ordered (*integer*).

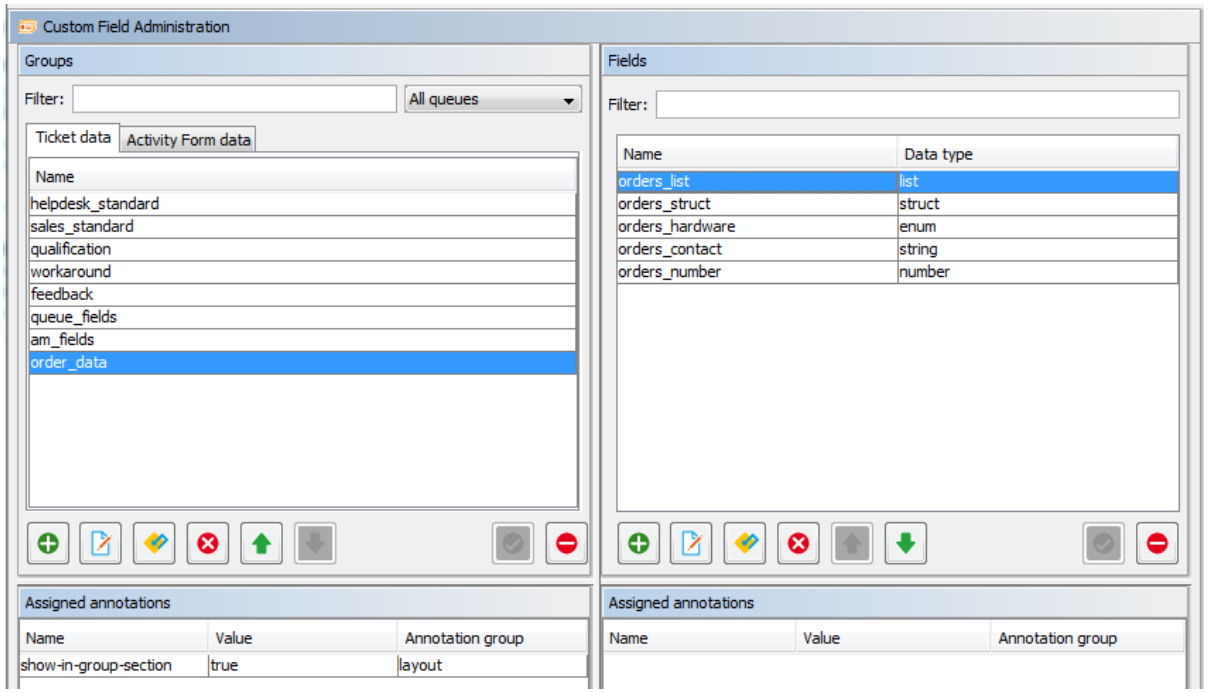


Fig. 7: ConSol*CM Admin-Tool - Custom Fields for List

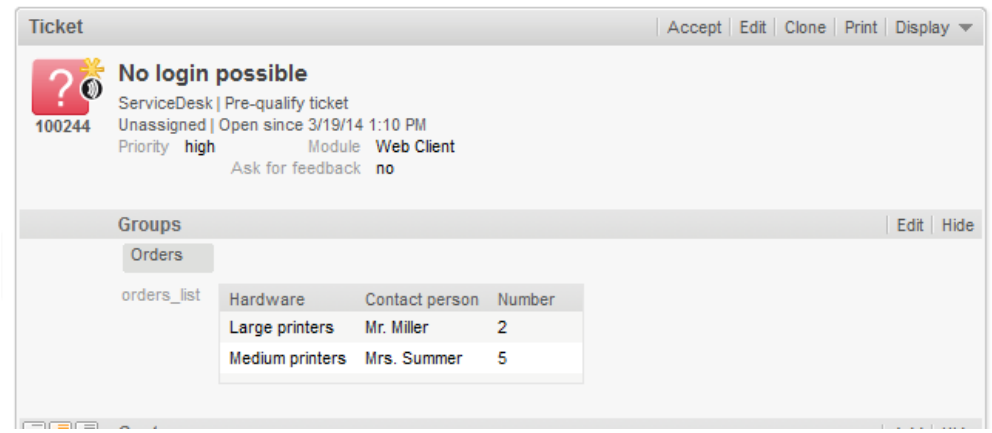


Fig. 8: ConSol*CM/Web Client - Ticket with Filled-in Table

Retrieve data from a list of structs

```
def structs = ticket.get("order_data.orders_list").each() { str ->
    println("CLASS of LINE is " + str.getClass())

    println("FIELD VALUE HARDWARE is " + str.orders_hardware.getName())
    println("CLASS of FIELD VALUE HARDWARE is " + str.orders_hardware.getName().getClass())
    println("FIELD VALUE CONTACTis " + str.orders_contact)
    println("CLASS of FIELD VALUE CONTACT is " + str.orders_contact.getClass())
    println("FIELD VALUE NUMBER is " + str.orders_number)
    println("CLASS of FIELD VALUE NUMBER is " + str.orders_number.getClass())
}
```

```
2014-03-27 11:39:44,425 INFO  [STDOUT                    ] CLASS of LINE is class com.consol.cmas.common.model.customfield.cfel.Struct
2014-03-27 11:39:44,429 INFO  [STDOUT                    ] FIELD VALUE HARDWARE is large_printers
2014-03-27 11:39:44,429 INFO  [STDOUT                    ] CLASS of FIELD VALUE HARDWARE is class java.lang.String
2014-03-27 11:39:44,430 INFO  [STDOUT                    ] FIELD VALUE CONTACTis Mr. Miller
2014-03-27 11:39:44,431 INFO  [STDOUT                    ] CLASS of FIELD VALUE CONTACT is class java.lang.String
2014-03-27 11:39:44,431 INFO  [STDOUT                    ] FIELD VALUE NUMBER is 2
2014-03-27 11:39:44,454 INFO  [STDOUT                    ] CLASS of FIELD VALUE NUMBER is class java.lang.Long
2014-03-27 11:39:44,455 INFO  [STDOUT                    ] CLASS of LINE is class com.consol.cmas.common.model.customfield.cfel.Struct
2014-03-27 11:39:44,456 INFO  [STDOUT                    ] FIELD VALUE HARDWARE is medium_printers
2014-03-27 11:39:44,456 INFO  [STDOUT                    ] CLASS of FIELD VALUE HARDWARE is class java.lang.String
2014-03-27 11:39:44,456 INFO  [STDOUT                    ] FIELD VALUE CONTACTis Mrs. Summer
2014-03-27 11:39:44,457 INFO  [STDOUT                    ] CLASS of FIELD VALUE CONTACT is class java.lang.String
2014-03-27 11:39:44,458 INFO  [STDOUT                    ] FIELD VALUE NUMBER is 5
2014-03-27 11:39:44,458 INFO  [STDOUT                    ] CLASS of FIELD VALUE NUMBER is class java.lang.Long
```

Fig. 9: Log File - Script Output

Setting Custom Field Values for Ticket Data

To set values for ticket CFs, you follow the same principle as for getting data: use the CF group name and the technical name of the CF as a reference. Of course, additionally, the new value is required. And of course it has to be of the correct data type.

```
ticket.set("<Group_name>.<CF_name>", <value>)
```

Setting Values for Custom Fields with Simple Data Types

Set a CF value for a date CF

```
ticket.set("fields.reaction_time", new Date());
```

When you work with *number* or *date* fields, you can even calculate with the CF values in a very comfortable way, see following example.

Calculate with value of date CF

```
//add 24 hours (in millis) to current field value
 ticket.add("fields.deadline", 24*60*60*1000);
```

Setting a value to *null* (i.e. emptying the field) is the same as removing the value:

Setting a CF value to null

```
ticket.set("fields.numberOfEmployees", null)
```

Or shorter:

Setting a CF value to null via removing the value

```
ticket.remove("fields.numberOfEmployees" )
```

Setting Enum Values

To set an *enum* value use the following syntax. Of course, the new value has to be present in the ordered list (enum) which is referenced by the CF.

```
ticket.set("Group_name.CF_name",<technical name of value>)
```

Setting an enum value

```
ticket.set("fields.priority", "URGENT");
```

Setting List Values

Setting Values in Lists of Simple Data Types

When you want to add a line, you can simply use the *add* method:

Adding a new line in a list of strings

```
ticket.add("fields.tags", "my new String")
```

When you want to refer to a certain value to set a new value for it, you have to use the syntax for an array:

Setting a value in a list of strings

```
ticket.set("fields.tags[last]", "consol cm6")
```

Setting Values in Lists of Structs

Working with *structs*, you always have to work with the key of the value you would like to add or set. When you want to *add* a new line, you have to build a new struct as new line. The *set* method can be used one after another for each new field.

Adding a new line in a list of structs

```
ticket.add("order_data.orders_list", new Struct().set("tA_Id",
id).set("orders_hardware",mynewhardware_model).set("orders_contact",
thenewcontactname).set("orders_number",thenewnumber)
```

Fading-in and -out of Custom Field Groups

A custom field group (CF group) can be faded-in (made visible) and faded-out (made invisible) using a *workflowApi* method. This works for CF groups which are displayed in the main ticket data section as well as for CF groups which are displayed in the tabbed section.

A typical use case is a CF group which is invisible at first (CF group annotation *group-visibility* = *false*) and is faded-in when the engineer needs to work with the data in the process. For example, a CF group which contains reasons for the dismissal of a request is only displayed (faded-in) when the engineer has used the workflow activity *Dismiss ticket ...* . This prevents an information overload of the ticket.

Fade-in a CF group

```
workflowApi.setGroupProperty(„CF_Group_Dismissal",GroupPropertyType.VISIBLE,"true"
```

To fade-out some CF groups, e.g. when the ticket has been qualified and some of the CF groups will no longer be required in the process, use code according to the following example:

Fade-out a CF group

```
workflowApi.setGroupProperty("CF_Group_HardwareInfo",GroupPropertyType.VISIBLE,"false")
workflowApi.setGroupProperty("CF_Group_SoftwareInfo",GroupPropertyType.VISIBLE,"false")
```

6.3.4 Data Fields for Customer Data

Custom Fields for Customer Data (CM Version 6.8 and Older)

In CM version 6.8 and older, customer data are defined in the Admin-Tool, section *Custom Field Administration*, tab *Customer data*.

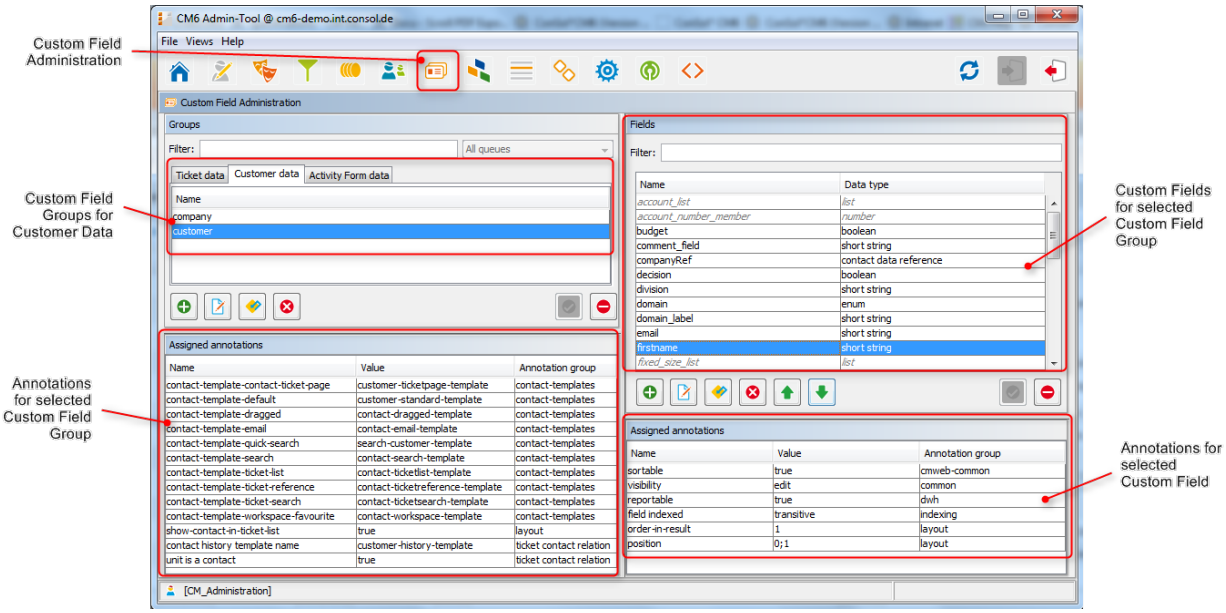


Fig. 10: ConSol*CM Admin-Tool - Custom Field Administration for Customer Data (CM Version 6.8 and Older)

The customer data can comprise one level (only a contact level) or two levels (contact = customer level and company level). I.e. you have to deal with two objects maximum. The names of the objects depend on the names which have been assigned to them in the Admin-Tool. In the example (see figure above), the contact (= customer) object is named *customer* and the company object is named *company*.

Retrieving Values

Each object within the customer data represents a *unit* (i.e. an instance of the *Unit* class). In scripts, the unit (customer or company) has to be retrieved, before you can work with it. If the customer data model contains two levels (contact = customer and company), you will see a CF in the contact object which has the data type *contact data reference*. This is the link between the contact and company object.

```
Unit contact = ticket.getMainContact()
```

```
Unit company = contact.get('<contact data reference_field>')
```

For all other CFs, the access to data is based on the same principle as for ticket data.

```
Type t = contact.get('<CF_name>')
```

For example:

Retrieving customer data from a CF

```
def fn = customer.get("firstname")
```

Setting Values for Customer Data in CM Version 6.8 and Older

```
company.set('<CF_name>', <new value>)
```

Setting values for a company in a list of structs

```
ticket.set("person_data.responsibleConsultants", new Struct[]{
 new Struct().set("lastName", "Miller").set("email", "miller@consol.com"),
 new Struct().set("lastName", "Smith").set("email", "smith@consol.com"),
 new Struct().set("lastName", "Burger").set("email", "burger@consol.com")
 });
```

Data Object Group Fields for Customer Data (CM Version 6.9 and Higher)

In CM version 6.9 and higher, the customer data object groups are part of the new customer data model (*FlexCDM*) and are defined in the Admin-Tool, section *User attributes*, file card *Customer data model*.

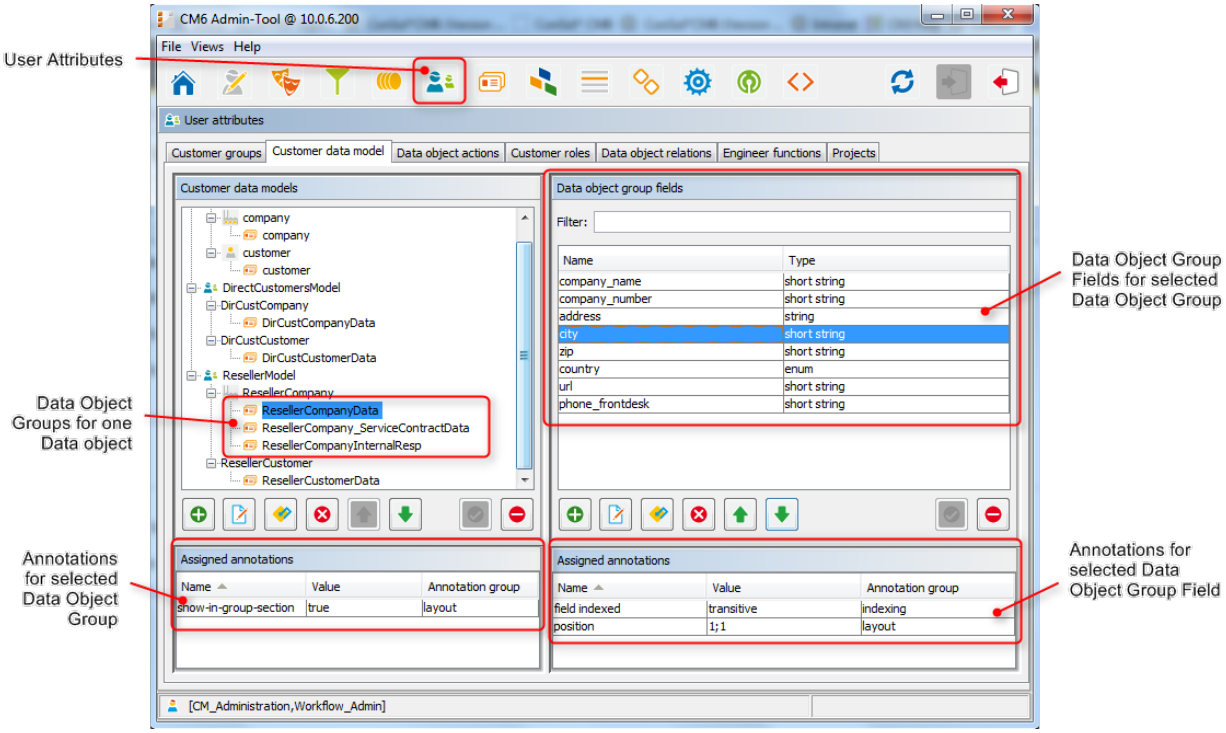


Fig. 11: ConSol*CM Admin-Tool - Custom Field Administration for Customer Data (CM Version 6.9 and Higher)

The fields, which were called custom fields in the customer data model of previous versions, are now called **data object group fields**. However, the principle you use to retrieve and set values for the data fields is principally the same as in CM version 6.8 and older.

Most Important Methods for Access to Customer Data Data Object Group Fields

Three methods are of major importance for programming access to data object group fields (DOGF) in CM scripts. They all are methods of the class *Unit*:

- **Unit.get()**
  - For retrieving data from a DOGF.

- **Unit.set()**
  - For setting data in an already existing DOGF.
- **Unit.add()**
  - For calculating with a value within a DOGF, i.e. to add a certain time range to a *date* field.
  - For adding a new line in list fields (simple lists and tables).

Another method might be used when a field should be emptied, i.e. when its value should be set to *null*:

- **Unit.remove()**
  - Sets the value of the field to *null*.

Retrieving Values for Customer Data in CM Version 6.9 and Higher

Because the name of a *data object group field* might appear in more than one *data object group*, the name of the data object group has to be provided when accessing the customer data. For example, in the customer data model shown in the figure above, the data object groups *ResellerCompanyData* and *DirCustCompanyData* could have a data object group field named *city*. Therefore, it is important to mention group name and field name.

Please use the following syntax:

```
unit.get("group1:name")
```

For example:

Retrieving a field value for a company

```
def mycity = company.get("ResellerCompanyData:city")
```

There are various objects and methods to work with data on different levels of the FlexCDM. Please see the following example where several common objects and methods have been applied. It is an Admin-Tool script which is accessed from a workflow activity. The only purpose is to display some data of the ticket's main customer. The following figure shows the Java objects used in the script and the ConSol*CM objects in the Admin-Tool which are referenced.

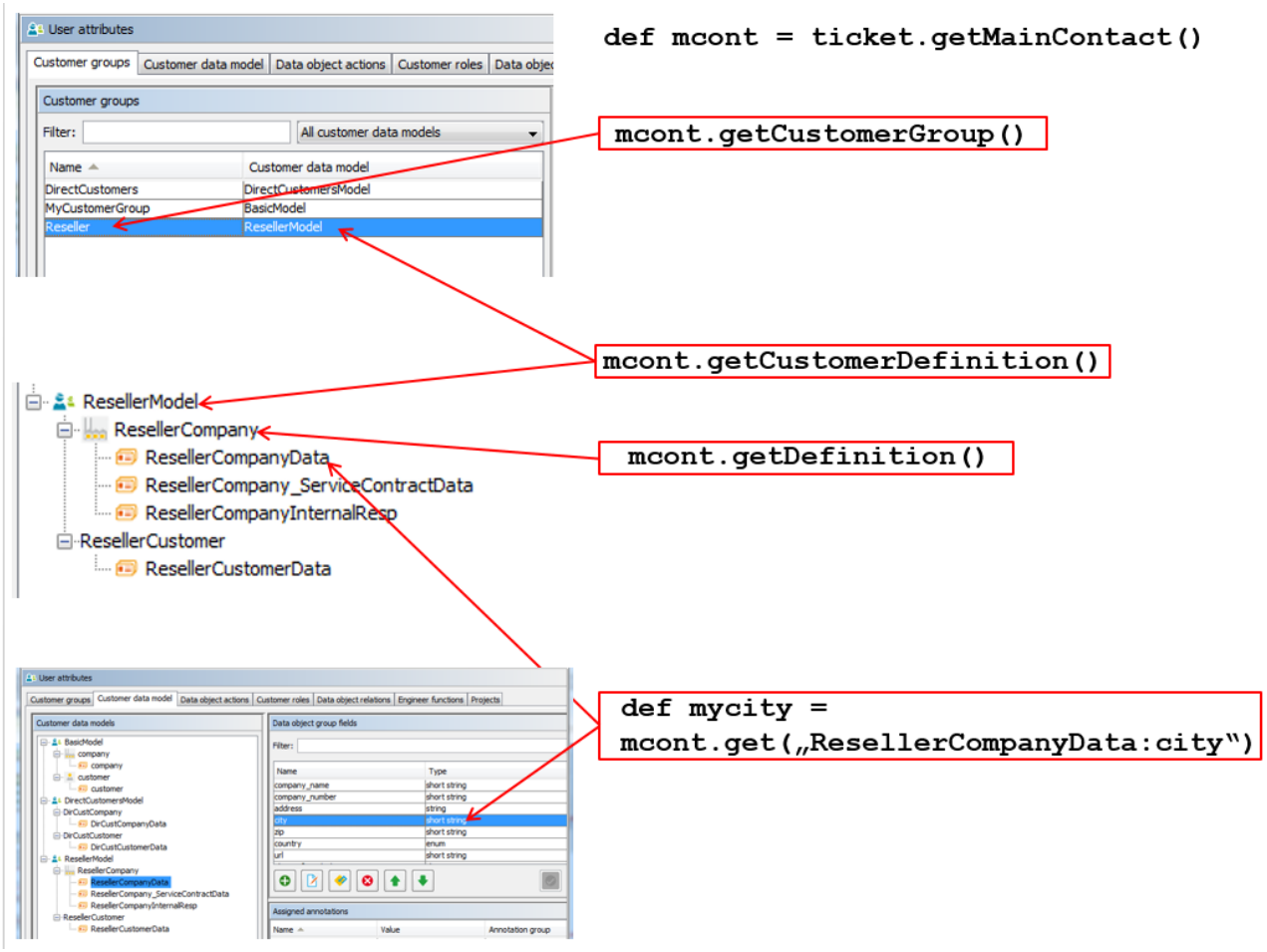


Fig. 12: ConSol*CM Customer Objects in Script and Admin-Tool

Information:

Please keep in mind that you might also use the short notation like *unit.definition.type* for getter methods like *unit.getDefinition().getType()*.

Admin-Tool script for displaying customer data

```
import com.consol.cmas.common.model.ticket.Ticket
import com.consol.cmas.common.model.customfield.meta.UnitDefinitionType

def ticket = workflowApi.getTicket()

def mcont = ticket.getMainContact()
println "CustomerGroup of main contact is now " + mcont.getCustomerGroup().getName()
println "Customer definition of main contact is now " + mcont.getCustomerDefinition().getName()
println "UnitDefinition of main contact is now " + mcont.getDefinition().getName()

def custmod = mcont.getCustomerDefinition().getName()
// println "CUSTMOD is now " + custmod

def cityfield

switch (custmod) {

    case "BasicModel" : cityfield = "company:city";
    break;
    case "DirectCustomerModel" : cityfield = "DirCustCompanyData:dir_cust_company_city";
    break;
    case "ResellerModel": cityfield = "ResellerCompanyData:city";
    break;
}


println "CITYFIELD is now " + cityfield

def utype1 = mcont.getDefinition().getType()
def utype2 = mcont.definition.type

println "UTYPE1 is now " + utype1
println "UTYPE2 is now " + utype2

def company = mcont

if (utype2 == UnitDefinitionType.CONTACT) {
    company = mcont.get("company()")
}

def mycity = company.get(cityfield)
println " CITY is now " + mycity
```

For the following data set the log file output is shown below. The *Reseller* model of the figure above is used.

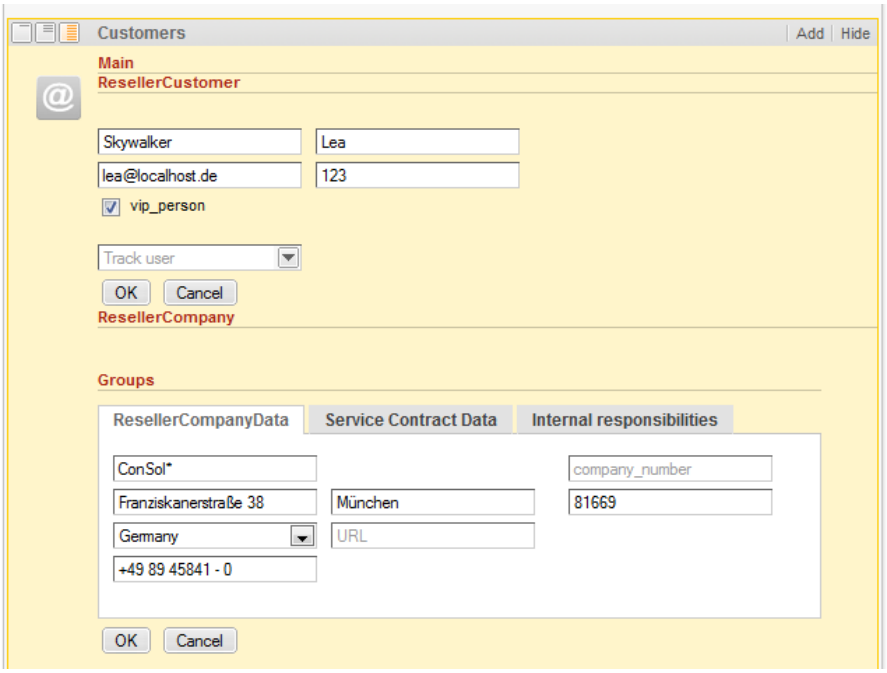


Fig. 13: ConSol*CM/Web Client - Customer Data Set

```
2014-03-27 16:09:21,739 INFO  [STDOUT                                ] CustomerGroup of main contact is now Reseller
2014-03-27 16:09:21,739 INFO  [STDOUT                                ] Customer definition of main contact is now ResellerModel
2014-03-27 16:09:21,740 INFO  [STDOUT                                ] UnitDefinition of main contact is now ResellerCustomer
2014-03-27 16:09:21,743 INFO  [STDOUT                                ] CUSTMOD is now ResellerModel
2014-03-27 16:09:21,744 INFO  [STDOUT                                ] CITYFIELD is now ResellerCompanyData:city
2014-03-27 16:09:21,750 INFO  [STDOUT                                ] UTYPE1 is now CONTACT
2014-03-27 16:09:21,751 INFO  [STDOUT                                ] UTYPE2 is now CONTACT
2014-03-27 16:09:21,759 INFO  [STDOUT                                ]  CITY is now München
```

Fig. 14: Log File - Script Output

Retrieving a value from a list of structs using index notation

```
String firstName = company.get("responsibleConsultants[0].firstName");
```

Setting Values for Customer Data in CM Version 6.9 and Higher

Setting Values for Data Object Group Fields with Simple Data Types

The *set* and *add* methods work as described for ticket custom fields. For example:

Set and add values for a data object group field of type integer

```
//set number field
company.set("numberOfEmployees", 1);
//add 1 to field value, afterwards the value of the field is 2
company.add("numberOfEmployees", 1);
```

Lists

Setting Values in a List of Structs for Customer Data

Creating a new list of structs, version 2

```
company.set("responsibleConsultants", [
 new Struct().set("lastName", "Miller").set("email", "miller@consol.com"),
 new Struct().set("lastName", "Smith").set("email", "smith@consol.com"),
 new Struct().set("lastName", "Burger").set("email", "burger@consol.com")
]);
```

Adding a new line in a list of structs for company data

```
company.add("responsibleConsultants", new Struct().set("lastName", " Nowitzki ").set("email",
"dnowitzki@consol.us"));
```

Setting a value in a list of structs using index notation

```
company.set("responsibleConsultants[0].firstName", "John");
```

Removing a struct (= line) from a list of structs (= table)

```
company.set("responsibleConsultants[last]", null);
```

Convenience Methods for Access to Customer Data in CM Version 6.9 and Higher

Convenience methods for access to customer data

```
Unit mainContact = ticket.getMainContact();

// "company" extension returns company for contact
Unit company = mainContact.get("company()");

// it is also possible to set company using "company" extension
mainContact.set("company()", company);

// "contacts" extension returns list of contacts for company
List contacts = company.get("contacts()");

// "tickets" extension returns list of tickets for contact or company
List tickets = company.get("tickets()");
tickets = mainContact.get("tickets()");

// extensions can be chained
Integer count = contact.get("company().contacts()[0].tickets()[count]");

// parenthesescan be omitted, but it is not recommended (possible collision with name of group
or field)
count = contact.get("company.contacts[0].tickets[count]"); // here "company" is not extension
but name of field
```

6.3.5 Using Data Fields for (Invisible) Variables

Sometimes it is necessary to work with variables which are not used as values for GUI-visible custom fields or data object group fields, but which are only used as containers for internal programming variables.

Those of you who know how to program ConSol*CM5 workflows know those containers as *global variables*. In ConSol*CM6, you can achieve the same goal by creating regular custom fields (for ticket data) or data object group fields (for customer data) with the required data type and setting the field to *invisible*. This has to be done by using the annotation *visibility = none*. You can even let the variable be visible during the development of the process and control the field's value. Then you can set it to invisible when the system is handed-over to QA and users.

6.4 Sending E-Mails

6.4.1 Introduction to Sending E-Mails

The capability of receiving and sending e-mails is a core feature of ConSol*CM. Please read the detailed introduction in the *ConSol*CM Administrator Manual* for information.

In this section we will describe how you can write scripts to send e-mails from the workflow. This is very useful for use cases like the following:

- You want to send an automatic acknowledgment of receipt to the customer when he/she has opened a ticket.
- You want to inform the engineer and his supervisor when the highest escalation level has been reached.
- You want to inform the customer that a problem has been solved (and how).

Usually, you do not write the text of the e-mail into the script but you work with e-mail templates. So please read the detailed introduction to the *ConSol*CM Template Designer* in the *ConSol*CM Administrator Manual* first.

6.4.2 Important Methods

ConSol*CM Version 6.8 and Older

Use *workflowApi.sendEmail().*

ConSol*CM Version 6.9 and Higher

Use an object of class *Mail*.

Here you can define all required parameters for an e-mail and you can configure the *Mail* object to use the queue-specific e-mail default script. This is a script which processes the e-mail before it leaves the CM system. This kind of script can be assigned to a queue (*E-Mail script*, see section *Queue Administration* in the *ConSol*CM Administrator Manual*). To use such a script can prove helpful, for example when you want to set a *REPLY TO* address which is not the standard *REPLY TO* address (stored in a system property).

6.4.3 Examples

Sending an Automatic Acknowledgment of Receipt to the Customer When He/She Has Opened a Ticket

ConSol*CM Version 6.8 and Older

This script might be placed in one of the first activities of the workflow.

```
// fetch main contact of the ticket
def contact = ticket.getMainContact()

// fetch e-mail address = Custom Field of contact
def contact_e = contact.get("email")

// build e-mail text using a template which is stored in the Template Designer
def text = workflowApi.renderTemplate("Acknowledgement_of_receipt")

// fetch the REPLY TO address which is stored in a system property
def replyto = configurationService.getValue("cmweb-server-adapter","mail.reply.to")

// set the subject of the e-mail, the ticket number with the correct Regular Expression
// has to be set for correct recognition of incoming e-mails for the ticket
def subj = "Your case has been registered as Ticket (" + ticket.getId() + ")"

// send out the e-mail
workflowApi.sendEmail(contact_e,subj,text,replyto,null)
```

ConSol*CM Version 6.9 and Higher

This script might be placed in one of the first activities of the workflow.

```
// create new mail object
def mail = new Mail()

// fetch main contact of the ticket
def maincontact = ticket.getMainContact()

// fetch e-mail address of the main contact. The data object group field has to be addressed
using data object group name:data object group field name
def toaddress = maincontact.get("MyCustomerDataObjectGroup:email")

// put the e-mail TO address into the Mail object
mail.setTo(toaddress)

// fetch the REPLY TO address, this is stored in a system property
def replyaddress = configurationService.getValue("cmweb-server-adapter","mail.reply.to")

// put the e-mail REPLY TO address into the Mail object
mail.setReplyTo(replyaddress)

// build e-mail text using a template which is stored in the Template Designer
def text = workflowApi.renderTemplate("Acknowledgement_of_receipt")

// put the e-mail text into the Mail object
mail.setText(text)

// create the subject of the e-mail, the ticket number with the correct Regular Expression has
to be set for correct recognition of incoming e-mails for the ticket
def ticketname = ticket.getName()
def subject = "Your case has been registered as Ticket (" + ticketname + ")"

// put the subject into the Mail object
mail.setSubject(subject)

// send out the e-mail
mail.send()
```

Sending an E-Mail to the Engineer When a Certain Escalation Level Has Been Reached

This script might be placed in an automatic activity which is connected to a time trigger. The time trigger measures the escalation interval. When the deadline has been reached, the trigger fires and the ticket enters the automatic activity.

ConSol*CM Version 6.8 and Older

```
// fetch current engineer of the ticket
def eng = ticket.getEngineer()

// fetch e-mail address = Standard Data Field of engineer, check if there is a current engineer
to avoid NullPointerException
def  eng_email = eng?.getEmail()

// build e-mail text using a template which is stored in the Template Designer
def text = workflowApi.renderTemplate("ESCALATION_Mail")

// fetch the REPLY TO address which is stored in a system property
def replyto = configurationService.getValue("cmweb-server-adapter","mail.reply.to")

// set the subject of the e-mail, the ticket number with the correct Regular Expression has to
be set for correct recognition of incoming e-mails for the ticket
def subj = "ESCALATION  Level 3 REACHED! Ticket (" + ticket.getId() + ")"

// send out the e-mail
workflowApi.sendEmail(eng_email,subj,text,replyto,null)
```

ConSol*CM Version 6.9 and Higher

```
// create new mail object
def mail = new Mail()

// fetch current engineer of the ticket and set it as e-mail receiver
if (ticket.engineer){
    mail.setTargetEngineer(ticket.engineer)

    // fetch the REPLY TO address, this is stored in a system property
    def replyaddress = configurationService.getValue("cmweb-server-adapter","mail.reply.to")

    // put the e-mail REPLY TO address into the Mail object
    mail.setReplyTo(replyaddress)

    // build e-mail text using a template which is stored in the Template Designer
    def text = workflowApi.renderTemplate("ESCALATION_Mail")

    // put the e-mail text into the Mail object
    mail.setText(text)

    // create the subject of the e-mail, the ticket number with the correct Regular Expression
has to be set for correct recognition of incoming e-mails for the ticket
    def ticketname = ticket.getName()
    def subject = "ESCALATION  Level 3 REACHED! Ticket (" + ticket.getId() + ")"

    // put the subject into the Mail object
    mail.setSubject(subject)

    // send out the e-mail
    mail.send()
}
```

Sending an E-Mail to a Customer Integrating the Queue-Specific Mail Script

This is the same script as shown in the example above, but the queue-specific mail script will be used. For a detailed explanation of this type of script, refer to the *ConSol*CM Administrator Manual*, section *Admin-Tool Scripts*.

As an effect, the outgoing e-mail will pass through the script before it leaves the CM system. E-mail parameters, like *CC*, *BCC*, or *REPLY TO* can be changed.

```
// create new mail object
def mail = new Mail()

// fetch main contact of the ticket
def maincontact = ticket.getMainContact()

// fetch e-mail address of the main contact. The data object group field has to be addressed
using data object group name:data object group field name
def toaddress = maincontact.get("MyCustomerDataObjectGroup:email")

// put the e-mail TO address into the Mail object
mail.setTo(toaddress)

// fetch the REPLY TO address, this is stored in a system property
def replyaddress = configurationService.getValue("cmweb-server-adapter","mail.reply.to")

// put the e-mail REPLY TO address into the Mail object
mail.setReplyTo(replyaddress)

// build e-mail text using a template which is stored in the Template Designer
def text = workflowApi.renderTemplate("Acknowledgement_of_receipt")

// put the e-mail text into the Mail object
mail.setText(text)

// create the subject of the e-mail, the ticket number with the correct Regular Expression has
to be set for correct recognition of incoming e-mails for the ticket
def ticketname = ticket.getName()
def subject = "Your case has been registered as Ticket (" + ticketname + ")"

// put the subject into the Mail object
mail.setSubject(subject)

// Mail should use the e-mail script which is configured for the queue
mail.useDefaultScript()

// send out the e-mail
mail.send()
```

Sending an E-Mail to All Contacts of the Ticket

This will send one e-mail with all customers (that have an e-mail address) as receiver. Please note that this is a simple example which demonstrates the use of a list. The *REPLY TO* address is not set, so answers to the e-mail would not be appended to the ticket.

```
def custEmails = workflowApi.getContactList()*.get("email").findAll{it != null}.join(",")
workflowApi.sendEmail(custEmails, "Confirmation", "Good afternoon, we received your request!",
null, null)
```

Sending an E-Mail to Each Contact in a List of All Contacts of the Ticket

This will send one e-mail to every single customer (that has an e-mail address). Please note that this is a simple example which demonstrates the use of a list. The *REPLY TO* address is not set, so answers to the e-mail would not be appended to the ticket.

```
workflowApi.getContactList().each {
  def custEmail = it.get("email")
  if (custEmail) workflowApi.sendEmail(custEmail, "Confirmation", "Good afternoon, we received
your request!", null, null)
}
```

6.5 Working with Path Information

6.5.1 Introduction

Like a file in a file system on a computer, every element of a workflow can be addressed using the path of this element. This might be required when you want to work with the element within a workflow script. A path represents the hierarchical structure of the workflow.

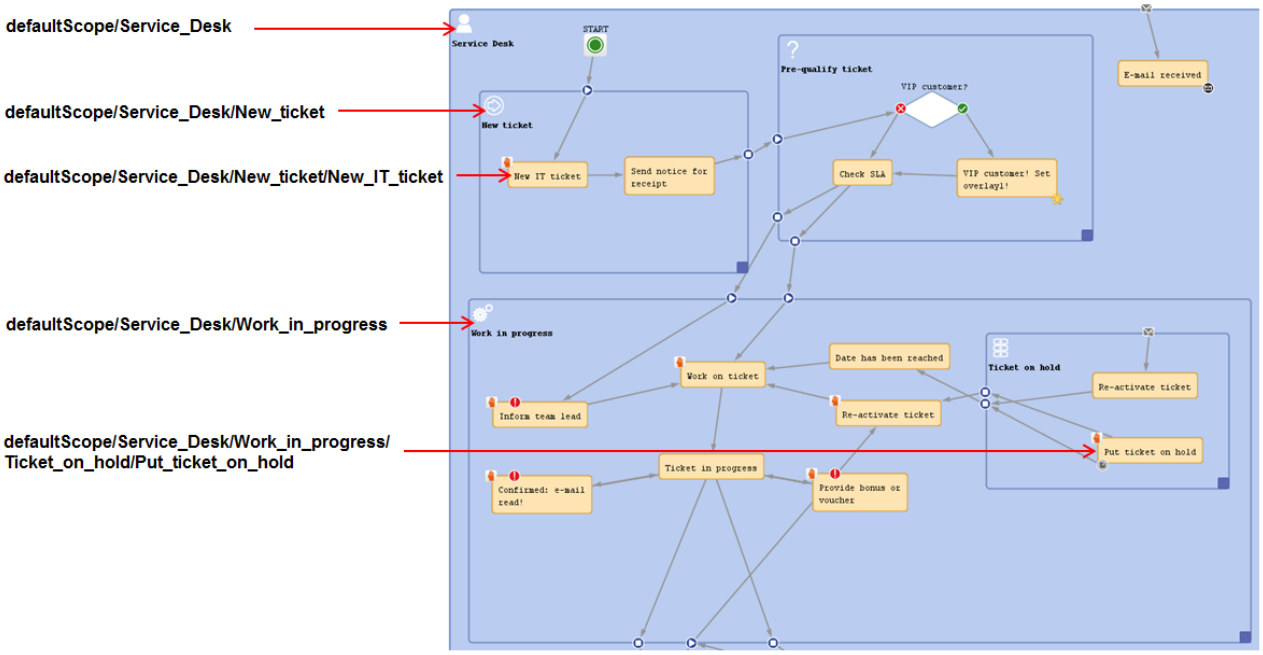


Fig. 1: ConSol*CM Process Designer - Path Information (Example: Activities and Scopes)

6.5.2 Retrieve Path Information for a Workflow Element

You can copy the path of an element by clicking on the element with the right mouse tab and selecting *Copy adornment's path to clipboard*.

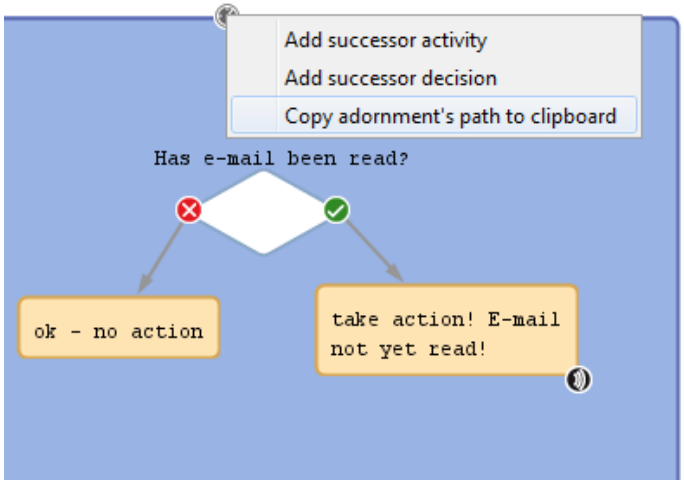


Fig. 2: ConSol*CM Process Designer - Copying the Path of a Workflow Element

6.5.3 Examples for the Use of Path Information

Example 1: Deactivate and/or Re-Initialize a Time Trigger

A typical case for the use of path information is the re-initialization of a time trigger, e.g. if you want to measure the time after an e-mail has been received and make sure that the e-mail is taken care of within a period of 10 minutes maximum. That means you have to use a time trigger over and over again and re-initialize it after each e-mail which has been received by the ticket.

When the ticket is created, the time trigger has to be deactivated. The following code would be used:

Deactivate a time trigger

```
workflowApi.deactivateTimer("defaultScope/Service_Desk/TimeTrigger1")
```

When an e-mail has been received, the trigger has to be re-initialized. The following code would be used:

Re-initialize time trigger

```
workflowApi.reinitializeTrigger("defaultScope/Service_Desk/TimeTrigger1")
```

6.6 Working with Calendars and Times

6.6.1 Introduction

Calculating dates and times plays an important role in ConSol*CM workflow programming. For a time trigger (see section Time Triggers), the exact point in time when it is supposed to fire can be set via script. This adds various possibilities in controlling escalation times, reminders for engineers, and other *active* components of a ConSol*CM process. Examples for potential calculations with dates and/or times are:

- escalation dates with time triggers
- *date* fields, like a desired (or required) deadline

When you calculate a date and/or time, you have to decide if a business calendar should be used or not. A business calendar defines working hours for a process. It is defined using the Admin-Tool and assigned to one or more queues.

For example, the service desk team might have working hours from 8 to 6 for 6 days a week, whereas the administration team works on a 9-to-5 basis, 5 days a week. Using a CM business calendar makes sure that an escalation will not be set during spare time and that non-working hours are not included into the calculation of the elapsed escalation time. Please refer to the *ConSol*CM Administrator Manual* for a detailed introduction about business calendars.

On the other hand, there are examples, when a business calendar is not required but the *pure* time based on the regular calendar should be used. For example, when it is required to get back to a customer three weeks after the initial contact. The following paragraphs will show you examples for both use cases.

How the time of a time trigger with calendar is calculated:

1 day means 24 hrs of absolute time, it has nothing to do with the use of a calendar. The calendar only plays a role when the time trigger is activated, then the 24 hrs, i.e 86400000 milliseconds, will be taken as business calendar input (if the calendar is enabled).

Example:
When we have as trigger time 1 day = 24 hrs without calendar, the 24 hrs are calculated like regular time, so the escalation will fire one day later at the same time.
In contrast: When we use a calendar (with, for example, 7 work hrs per work day), the 24 hrs will be split-up according to the calendar, resulting in the firing event more than 3 days later (24 hrs = 3 x 7 hrs + 3 hrs).

6.6.2 Calculating with Dates and Times without a CM Business Calendar

Example: Setting a Time Trigger Time with Dynamic Time Range

Depending on the priority, the time trigger for an escalation is configured:

Setting time for a time trigger

```
// prio is 'medium'
def escalationTime =
configurationService.getValue("custom-mycompany-properties","escalation.time.medium")
def escalationTimeMillisecs = escalationTime * 60 * 1000L
trigger.setDueTime( escalationTimeMillisecs )
```

6.6.3 Calculating with Dates and Times Using a CM Business Calendar

Example: Using a Time Trigger with a Business Calendar to Calculate Escalation Time (CM 6.9)

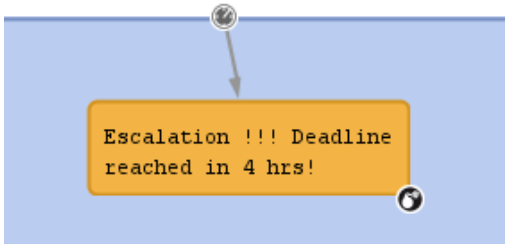


Fig. 1: ConSol*CM Process Designer - Time Trigger for Escalation 4 Hours before Deadline

Script for time trigger for escalation 4 hours before deadline

```
def deadl = ticket.get("serviceDesk_fields.desiredDeadline")

// 4hrs before deadline the escalation should be set
// business calendar should be used
// ServiceDeskCalendar is assigned to queue ServiceDesk, this is transparent here
def now = new Date()

// time required in millisecds
def four_hours = -4*60*60*1000L

// calculate escalation date
def escalDate = BusinessCalendarUtil.getBusinessTime(deadl, four_hours, ticket.queue.calendar)

// calculate and set due time

def dueTime = escalDate.time - now.time
trigger.setDueTime(dueTime)
```

6.7 ConSol*CM Process Designer Manual - Working with Object Relations

6.7.1 Working with Object Relations

In ConSol*CM, you can work with two types of relations:

Relation type	Explanation
Ticket Relations	Hierarchical or one-level relations between two tickets, see section Working with Ticket Relations
Customer Relations	Relations between customer data objects, i.e. contacts and companies, see section Working with Customer Relations (Data Object Relations)

6.7.2 Working with Ticket Relations

Introduction

Relations between tickets can help to model your business processes in a very efficient way.

ConSol*CM offers three types of relations:

- **Simple ticket relations**
  Non-hierarchical, simple reference. Each ticket can have any number of references.
  A simple ticket relation can be built by an engineer using the Web Client or by a programmer using the ConSol*CM programming interface.
  In both cases, a reference can only be established between two existing tickets.
- **Master-Slave relations**
  Hierarchical. A master ticket can have several slave tickets. A slave ticket always has exactly one master ticket.
  This construct can be built by an engineer using the Web Client or by a programmer using the ConSol*CM programming interface.
  A Master-Slave relation can only be established between two existing tickets, i.e. the tickets both have to exist first, then a Master-Slave relation can be built to connect them.
- **Parent-Child relations**
  Hierarchical. A parent ticket can have several child tickets. A child ticket always has exactly one parent ticket.
  This construct can only be built and manipulated using the ConSol*CM programming interface.
  A Parent-Child relation can be built between existing tickets. Also a new child ticket can be created during the process.

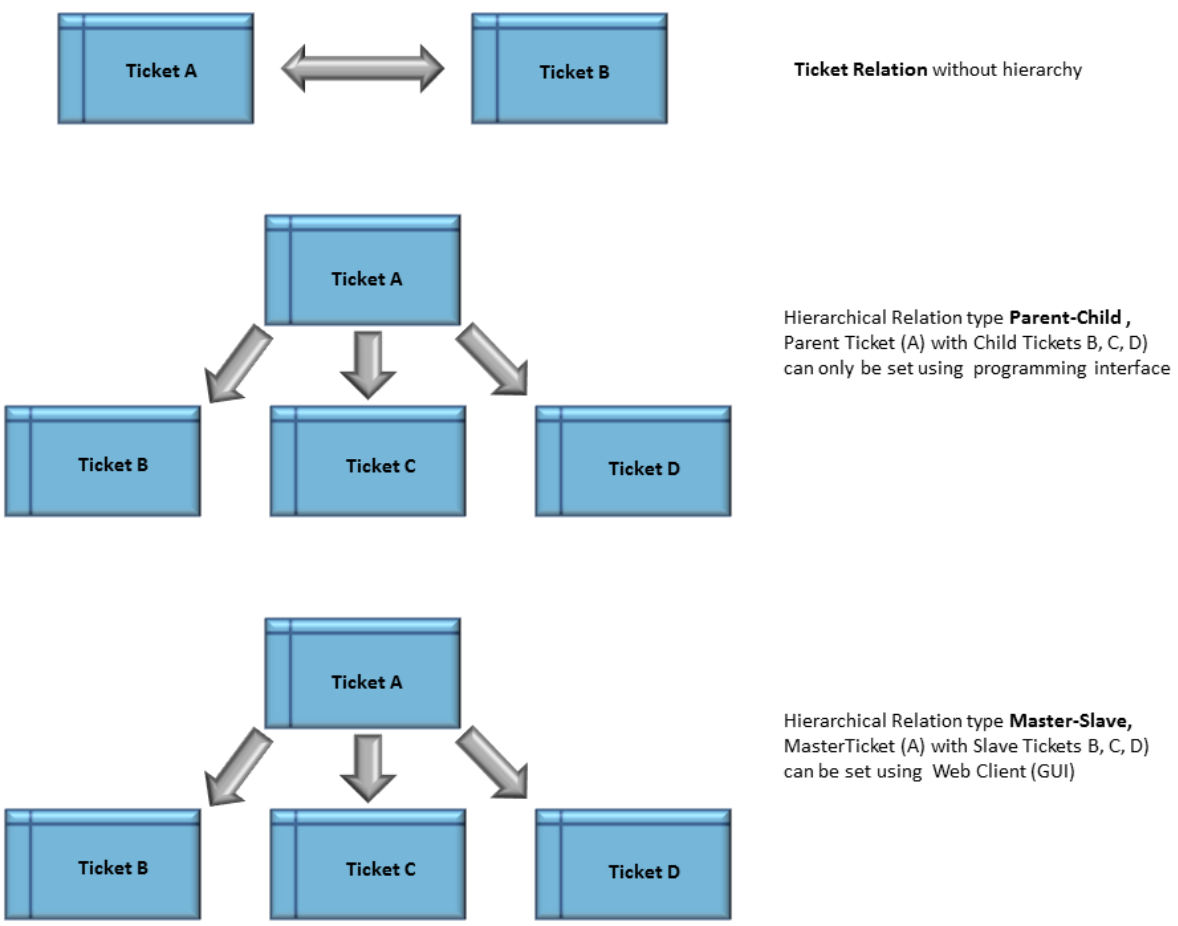


Fig. 1: ConSol*CM Relation Types

In this section, we will not explain how to set-up ticket relations using the Web Client, but we will show you how to establish relations using the programming interface, namely workflow scripts.

In the ConSol*CM Workflow API, the reference type is represented by the class (enum) *com.consol.cmas.common.model.ticket.TicketRelationType*. This offers three values:

- REFERENCE
- MASTER_SLAVE
- PARENT_CHILD

Simple Ticket Relation without a Hierarchy

This relation type can be helpful when you want to create references which help to find the tickets related to one ticket easier than using the search function.

Example use cases are:

- When a new ticket is created you want to see if there are any other open tickets from the same customer. If yes, you create a relation between the tickets. In this way, an engineer can easily jump from one open ticket of the customer to the next.

- When a new ticket is created for a certain hardware category, you want to establish references to all other tickets with the same hardware type.

This relation type can be built and manipulated using either the Web Client or the programming interface. Thus, a relation of type *REFERENCE* can be built within a workflow script and can then be manipulated by an engineer using the Web Client, provided he/she has the required access rights.

Example: Creating a Simple Relation between Two Tickets

Creating a ticket relation of type REFERENCE using workflowAPI

```
workflowApi.addRelation(TicketRelationType.REFERENCE, "This is a very important relation",
pSourceTicketId, pTargetTicketId)
```

Master-Slave Relations

This relation type can be helpful when you want to create a hierarchy between a certain number of existing tickets. Remember that this relation type can be established using either the Web Client or using the programming interface. However, here, only the programming approach will be explained.

Example use cases are:

- In a company, there are several projects, each represented by a ticket. When the decision has been made to integrate one of the projects in an overall program (also represented by a ticket), the project manager uses the workflow activity *Integrate into Program*. There, the correct program has to be selected (e.g. using an ACF). In the script of the workflow activity *Integrate into Program*, the program ticket is set as *Master* ticket of the current project ticket.
- In a service team, tickets for several different products are managed. For each product, there is one product ticket. When a new service ticket has been opened, the engineer uses the activity *Set product* where he can select the related product from a drop-down menu. In the workflow script of the activity *Set product*, the service ticket is automatically set as *Slave* of the product ticket.

Attention:

A Master-Slave relation can be built and manipulated using either the Web Client or the programming interface. Thus, a relation of type *MASTER_SLAVE* can be built within a workflow script and can then be manipulated by an engineer using the Web Client, provided he/she has the required access rights. Use the *Parent-Child* construct when you want to make sure that no engineer can manipulate the ticket hierarchy.

Example: Creating a Master-Slave Relation between Two Tickets

Creating a ticket relation of type MASTER_SLAVE using workflowAPI

```
//in this script the project ticket (= current ticket) is set as slave ticket to
// the program ticket which becomes the master

// fetch the program ticket ID. The ID of the program ticket is already stored
// in a CF in the project (=current) ticket
def progTicketId = ticket.get("ReferencesFields.ProgramTicketId")

// fetch ID of current ticket (which will become the slave)
def mySlaveProjectId = ticket.id

workflowApi.addRelation(TicketRelationType.MASTER_SLAVE, "Slave Ticket: This project is part of
the program indicated in the master ticket", progTicketId, mySlaveProjectId)
```

Syntax: Finding All Slave Tickets

Version A: Finding all target tickets (here: all slave tickets)

```
// the ticket can be set, might be current ticket or another ticket
List<Ticket> mytickets = workflowApi.getTargetTickets(myTicket.getId(),
TicketRelationType.MASTER_SLAVE)
```

Version B: Finding all target tickets (here: all slave tickets)

```
// used for current ticket
List<Ticket> mytickets = workflowApi.getTargetTickets(TicketRelationType.MASTER_SLAVE)
```

Parent-Child Relations

This relation type can be helpful when you want to create a hierarchy between a certain number of tickets which should not be manipulated manually.

Example use cases are:

- A project should be managed by the project management ticket which becomes the parent. All tasks within the project are represented as child tickets. This structure is automatically created by a workflow script during set-up of the project ticket.
- A system migration is planned using one parent ticket. For each single component which has to be migrated a child ticket is built. This structure is automatically created by a workflow script during set-up of the project ticket.

The relation type *PARENT_CHILD* can only be built and manipulated using the programming interface. Thus, a relation of this type can be built within a workflow script and can then only be manipulated by other scripts.

Example 1: Creating a New Child Ticket as Child of Current Ticket

Creating a child ticket

```
// this script creates a ticket for a task which will be child ticket
// of a project ticket (which will be the parent)

// create a new ticket, which will become the task (=child) ticket
Ticket newTask = new Ticket()

// fetch the subject of the parent-to-be ticket, i.e. of the current ticket
def subj = ticket.subject
// or longer:  def subj = ticket.getSubject()

// set the subject of the new task (= child) ticket
newTask.setSubject("New Task for project " + subj)

// put the task (= child) ticket into the tasks queue
def tasksQueue = queueService.getByName("Tasks")
newTask.setQueue(tasksQueue)

// Initially, the new task ticket will not have an engineer
newTask.setEngineer(null)

// define the ticket text, i.e. the first comment in the new task ticket
def taskTicketText = "Please work on this task asap"

// the contact for the new task ticket should be the same as the one for the project ticket:
def taskContact = workflowApi.getPrimaryContact()

//create PARENT_CHILD relation between project (parent) and task (child)
workflowApi.createChildTicket(newTask, taskTicketText, taskContact)
```

Example 2: Finding the Parent Ticket of a Ticket

Finding the parent ticket of a ticket

```
def my_parent = workflowAPI.getParentTicket()
```

Example 3: Finding All Child Tickets of a Ticket

Finding all child tickets of a ticket

```
// only works for current ticket:
List<Ticket> my_childtickets = workflowApi.getChildTickets()
```

Example 4: Finding All Brother Tickets (Other Child Tickets) of the Same Parent Ticket

Finding all brother tickets of a (child) ticket

```
// only works for current ticket:
List<Ticket> my_brothers = workflowApi.getBrotherTickets()
```

Important Methods for the Work with Ticket Relations

Note the following rules for the work with ticket relations:

- In MASTER_SLAVE relations, the master is always the source.
- In PARENT_CHILD relations, the parent is always the source.
- In simple REFERENCE relations the source is the ticket from which the relation has been created.

The following methods are methods of the class **WorkflowContextService** which is implicitly available as **workflowApi** object in workflow scripts.

Method	Explanation
Ticket createChildTicket(Ticket pTicket, String pTicketText, Unit pCustomer)	Creates a new child ticket. Queue, priority, and category have to be set correctly.
List getChildTickets()	IntSet containing the ticket objects of the child tickets of the current ticket.
List getBrotherTickets()	IntSet containing the ticket objects of the brother tickets of the current ticket.
Ticket getParentTicket()	Ticket object of the parent ticket or *null* if the current ticket does not have a parent ticket.
List getTargetTickets(TicketRelationType pType)	Get list of ticket objects that current ticket has relations of certain type to. For those relations, the current ticket is the source ticket.
List getTargetTickets(long pTicketId, TicketRelationType pType)	Get list of ticket objects that current ticket has relations of certain type t*o*. For those relations, the ticket given with *pTicketId* is the source ticket.
List getSourceTickets(TicketRelationType pType)	Get list of ticket objects that current ticket has relations of certain type from. For those relations, the current ticket is the destination ticket.
List getSourceTickets(long pTicketId, TicketRelationType pType)	Get list of ticket objects that current ticket has relations of certain type from. For those relations, the given ticket is the destination ticket.

Method	Explanation
boolean hasTargetTickets(TicketRelationType pType)	Check if ticket has target tickets. Check if relations exist that have this ticket as source ticket.
boolean hasTargetTickets(long pTicketId, TicketRelationType pType)	Check if given ticket has target tickets. Check if relations exist that have this ticket as source ticket.
boolean hasSourceTickets(TicketRelationType pType)	Check if ticket has source tickets. Check if relations exist that have this ticket as target ticket.
boolean hasSourceTickets(long pTicketId, TicketRelationType pType)	Check if given ticket has source tickets. Check if relations exist that have this ticket as target ticket**.**
void changeSourceTickets(TicketRelationType pType, long pTargetTicketId, List<Long> pSourceTicketIds)	For the target ticket (e.g. a child ticket) the relations of a given type (e.g. PARENT_CHILD) are removed. For the same relation type a new relation is created with the provided source tickets.
void changeTargetTickets(TicketRelationType pType, long pSourceTicketId, List<Long> pTargetTicketsIds)	For the given source ticket all relations of the given type are removed. For the list of provided target tickets new relations of the given type are created.
void removeRelation(TicketRelationType pType, long pSourceTicketId, long pTargetTicketId)	Remove ticket relation between two tickets with specified type.
void addRelation(TicketRelationType pType, String pComment, long pSourceTicketId, long pTargetTicketId)	Add relation of the specified type between ticket *sourceTicketId* and *targetTicketId*.

6.7.3 Working with Customer Relations (Data Object Relations)

Introduction

Since version 6.9.0, ConSol*CM offers *customer relations*. In older versions, this feature is not available!

To be able to work with customer relations, you have to have a profound knowledge of the *FlexCDM*, the ConSol*CM Flexible Customer Model. Please refer to the *ConSol*CM Administrator Manual (Version 6.9)* for a detailed introduction.

Three objects are essential:

Object	Java class	Admin-Tool description	Explanation
Customer	Unit	<none>	The general description or the general object which represents a customer, i.e. some person or company who is registered in the CM database
Company	Unit	Data object of type *company*	An object on company level (i.e. the highest level in the customer model). This can be a real company or this can be a machine or another object which represents the level. An object on the *company level* can be the *parent level* for an object on the *contact level*. From a logical point of view, a company can have several contacts.

Object	Java class	Admin-Tool description	Explanation
Contact	Unit	Data object of type *contact*	An object on contact level (i.e. the lowest level in the customer model). This can be a real person or another object which represents the level. An object on the *contact level* can be a stand-alone object (in a one-level customer model) or can belong to a *company level* object. From a logical point of view, a contact can belong to none or exactly one company.

Attention:

Keep in mind that, starting with CM version 6.9, the main customer of a ticket can be a contact or a company! The method used is *ticket.getMainContact()*. This returns an object of class *Unit*. The object can be a contact or a company!

Customer relations represent relations between customers, i.e. companies and contacts.

They can be:

- **directional**
  different levels in a hierarchy
- **reference**
  same level, no hierarchy

A relation is of one of the following types:

- **company - company**
  e.g. ... *has a cooperation with* ... (company X cooperates with company Y)
  - The companies can belong to the same or to different customer groups.
  - The involved customer groups can have the same or different customer data models.

- **company - contact**
  e.g. ... *is customer of* ... (contact X is customer of company Y)
  - The company and the contact can belong to the same or to different customer groups.
  - The involved customer groups can have the same or different customer data models.
- **contact - contact**
  e.g. ... *is serviced by* ... (contact X from company X is serviced by contact Y from company Y)
  - The companies and contacts can belong to the same or to different customer groups.
  - The involved customer groups can have the same or different customer data models.

In the programming interface, a customer object (i.e. a contact or a company) is represented by an object of the class *Unit*.

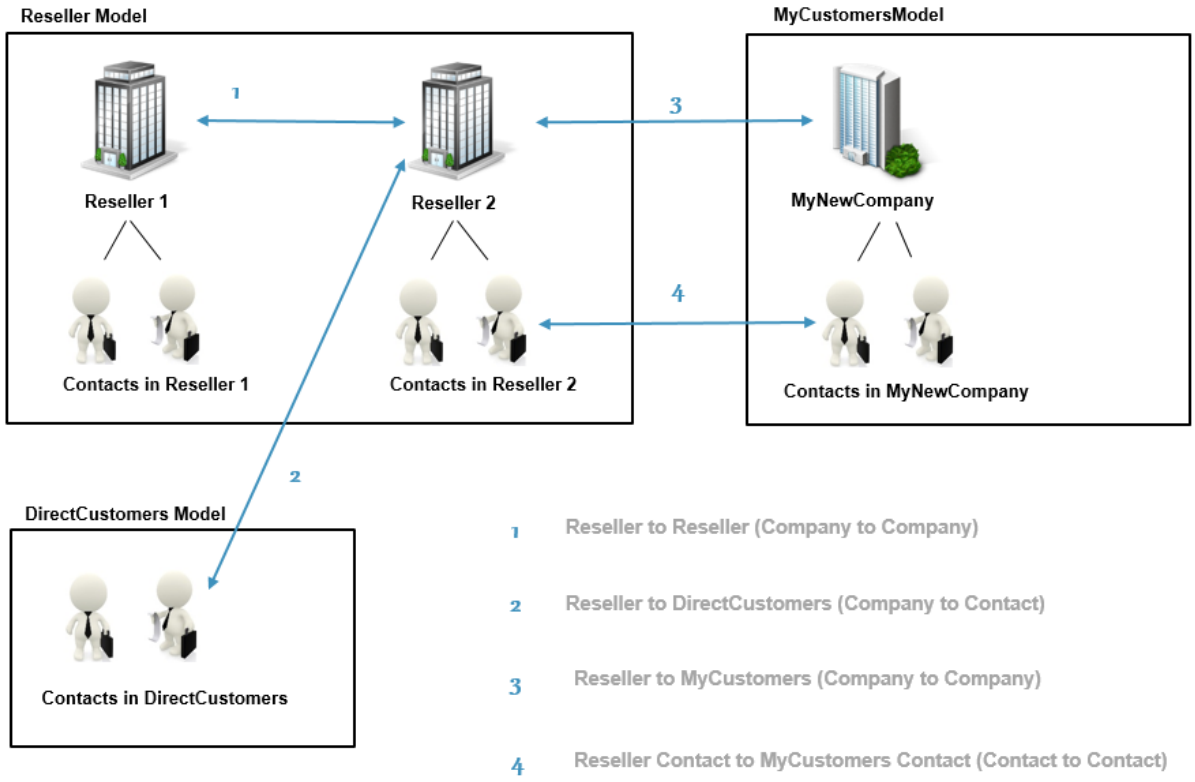


Fig. 1: ConSol*CM Customer Relations

Attention:

To work with *unit* relations in workflow scripts, make sure you have established and configured all required relations using the Admin-Tool before you start programming.

Creating Unit Relations Using the Programming Interface

Attention:

In this book we sometimes use the new terms *data object* and *data object definition* which are part of the new customer model of ConSol*CM version 6.9 and higher (*FlexCDM*). However, the names of the corresponding Java classes are still *Unit* and *UnitDefinition*. All other Java classes which deal with customer data objects are also still named *Unit... .* Please keep that in mind when you work on the administrator level as well as on the programmer's level with a 6.9.x version. Please refer to the *ConSol*CM Java API* documentation for details.

Example: Add a Reseller - End Customer Relation

In the following example, a relation has been defined in the Admin-Tool to reflect a *reseller - end customer* relation. A company of the customer group *Reseller* sells products to a customer (a person, a contact) of the customer group *DirectCustomers*.

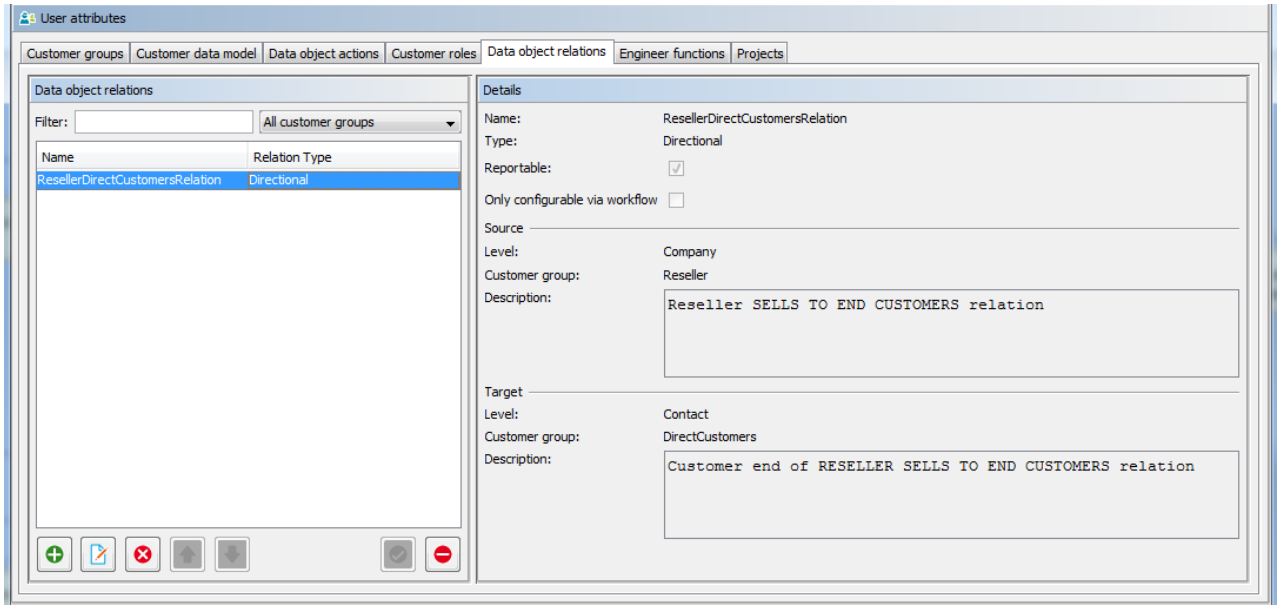


Fig. 2: ConSol*CM Admin-Tool - Definition of Reseller - End Customer Relation

A ticket is created with a main customer. This customer is an employee of a reseller company. The end customer to whom the reseller company sells products is added as additional customer in the role *end customer* to the ticket. The engineer who works on the ticket should be able to create a relation between the reseller company (source) and the end customer person (target) using a workflow activity.

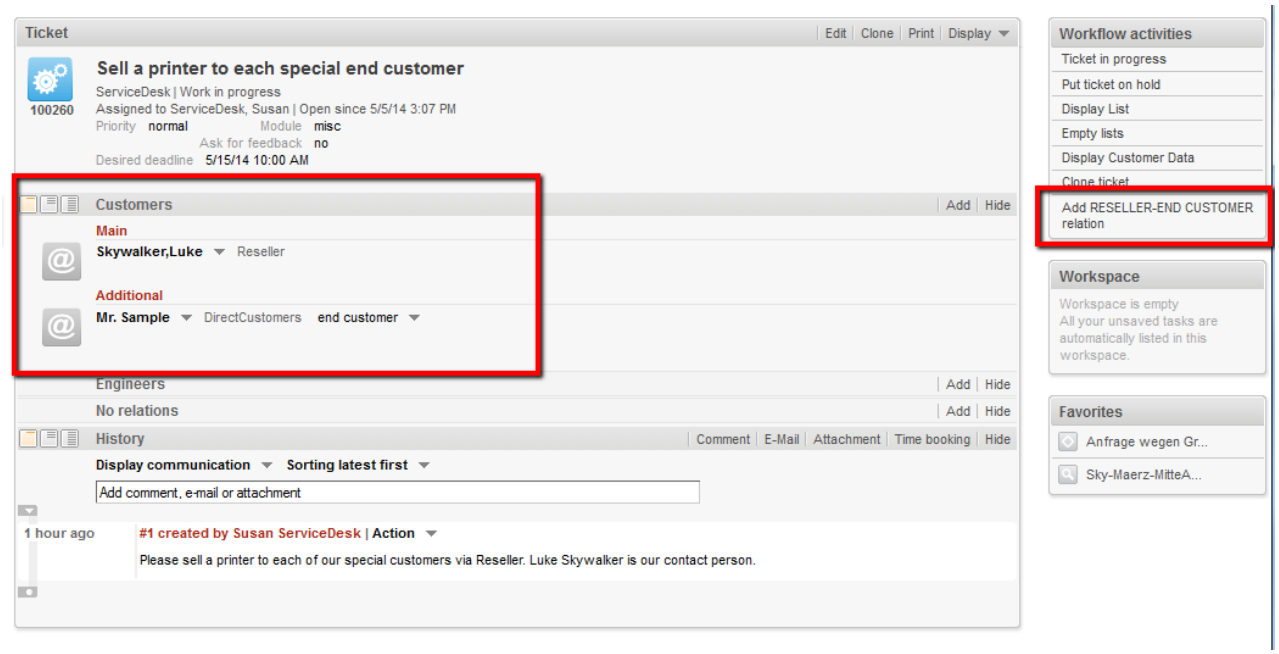


Fig. 3: ConSol*CM/Web Client - Example Ticket with Main Customer and One Additional Customer

In the *Service Desk* workflow, there is a workflow activity *Add RESELLER-END CUSTOMER relation* (see next figure).

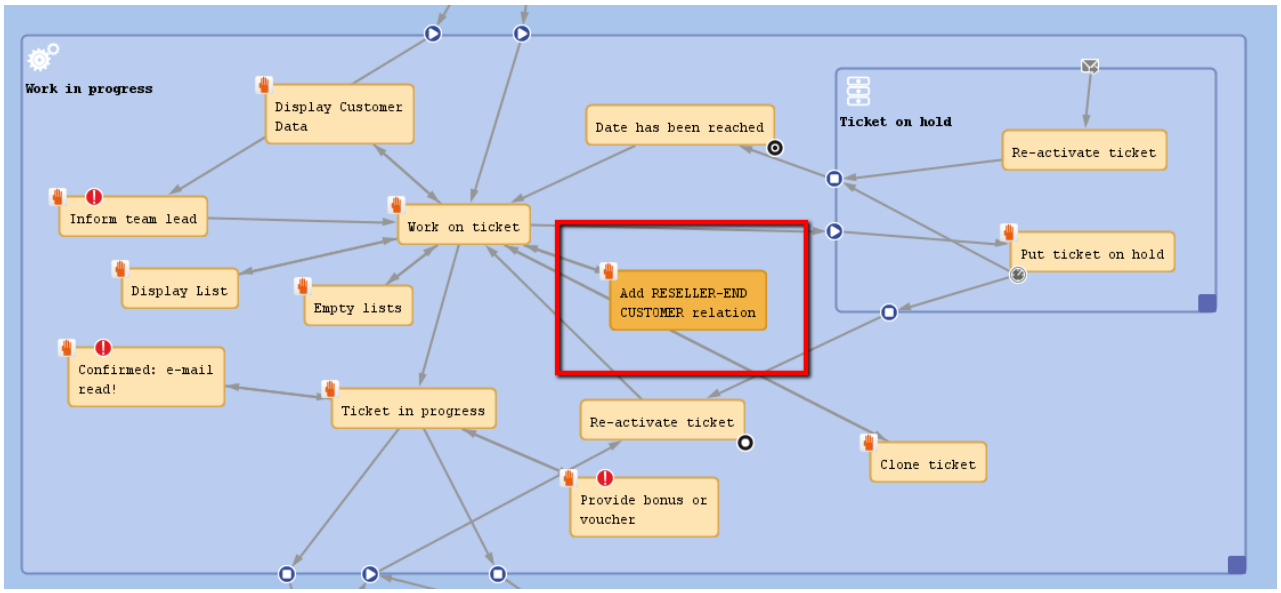


Fig. 4: ConSol*CM Process Designer - Workflow Activity for Adding a Unit Relation

The following script is used in the workflow activity *Add RESELLER-END CUSTOMER relation*:

Adding a data object relation using a workflow script

```
// get Company of the main customer of the ticket, this is the RESELLER company:
// 1. get the main contact of the ticket. Here, this is a person = contact:
def cont = ticket.getMainContact()
// 2. get the company of the contact, this is the reseller company
def comp = cont.getCompany()

// get all additional contacts of the ticket in the customer role „end customer“
//and start the loop for all those additional customers:
def end_custs = ticket.getContacts("end customer").each() { e_cust  ->

    //build all components for new unit relation:
    // 1.get the UnitDefinition by name (this is the name used in the Admin-Tool):
    def unitrel_def = unitRelationDefinitionService.getByName("ResellerDirectCustomersRelation")
    // create a new unit relation object with the unit definition and source
    // (the reseller company) and target (the end customer person)
    def new_rel = new UnitRelation(unitrel_def, comp, e_cust, "This Reseller sells to the end
customer")

    // create the new unit relation in the system
    def new_rel2 = unitRelationService.create(new_rel)
}
```

When the engineer has executed the workflow activity, the relation from the *reseller* company to the *end user* has been established.

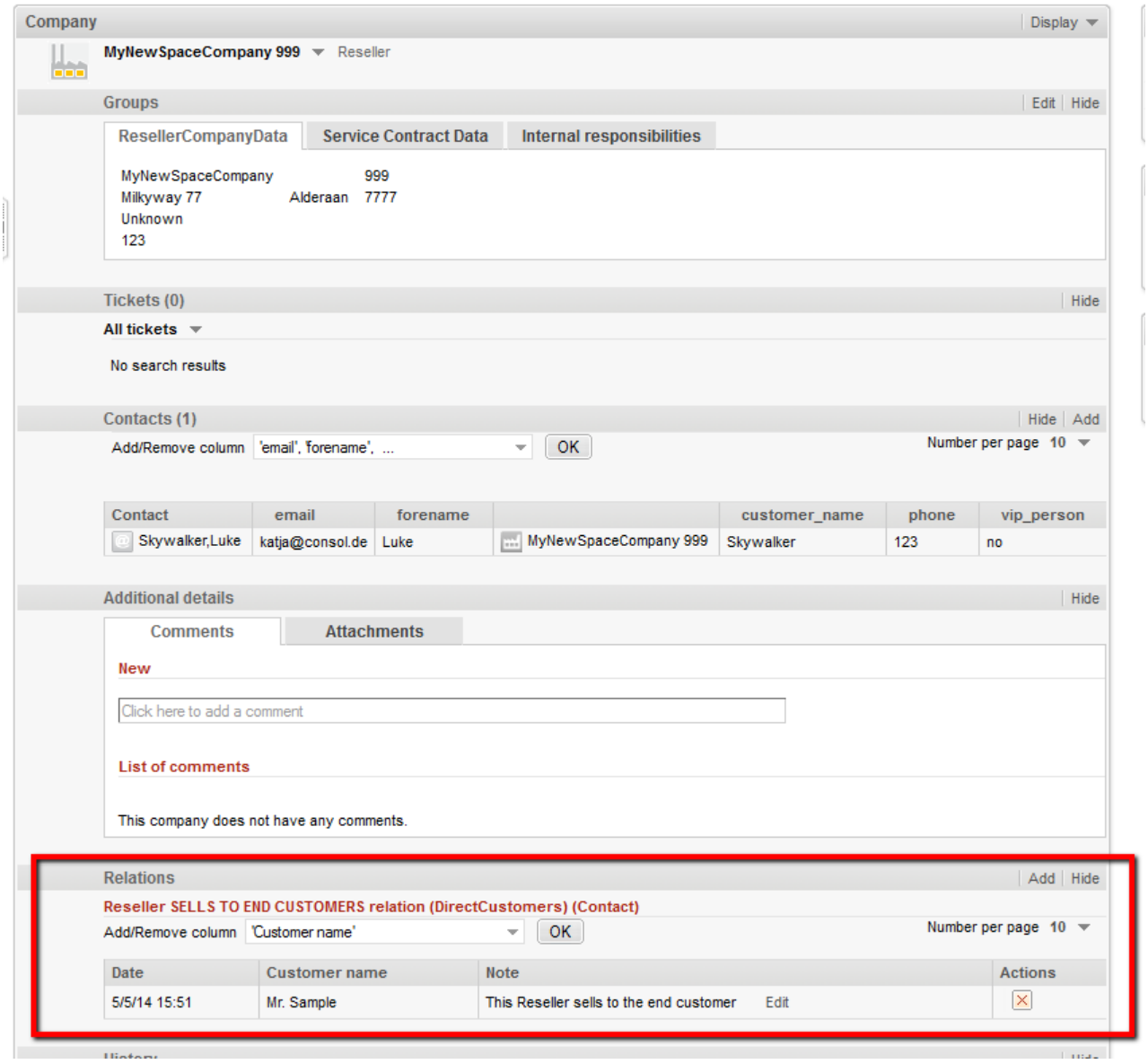


Fig. 5: ConSol*CM/Web Client - New Unit Relation (Created by Workflow Script)

Important Java Classes for the Work with Unit Relations

Java class	Explanation
Unit	A data object (unit): a contact or a company.
UnitRelation	A relation between two data objects (units). Visible in the Web Client on the contact or company page under *Relations*.

Java class	Explanation
UnitRelationDefinition	The definition of a unit relation as configured in the Admin-Tool under *User attributes* - *Data object relations*. A *UnitRelation* always has a certain *UnitRelationDefinition*.
UnitRelationDefinitionService	Singleton. Available as object *unitRelationDefinitionService*. Service which provides helpful methods for the work with data object (unit) relations. See the *ConSol*CM Java API* documentation for details.
UnitRelationService	Singleton. Available as object *unitRelationService*. Service which provides helpful methods for the work with data object (unit) relations. See the *ConSol*CM Java API* documentation for details.

6.8 Searching for Tickets and Customers Using the ConSol*CM Workflow API

6.8.1 Introduction

In ConSol*CM you can search the database for tickets or for units (contacts and companies). Both search modes are based on the same principle:

1. A *criteria* object is created where all parameters for the target objects are stored.
   a. **TicketCriteria** for tickets
   b. **UnitCriteria** for contacts and companies
2. This criteria object is handed over to a service which then returns a list with the result objects.
   a. **workflowApi** (WorkflowContextService) for tickets
   b. **UnitService** for units

The fields which are set as parameters for the criteria objects have to be indexed, i.e. the annotation *field-indexed* has to be set.

6.8.2 Searching for Tickets

To search for tickets you have to create the *TicketCriteria* object. The following fields can be set (see also the respective *setter* methods in the following picture):

- Date of ticket creation
- Engineer
- System-specific custom fields
- Ticket history criteria
- Ticket IDs
- Modification date
- Ticket name
- Pattern for the ticket subject
- Queue IDs

- IDs for current workflow scopes
- Current status (closed/open)
- Additional engineers

```
setCreationDateRange(DateRange pCreationDateRange)
setEngineerCriteria(TicketCriteria.EngineerCriteria pEngineerCriteria)
setFields(Set<AbstractField> pFields)
setHistoryCriteria(TicketCriteria.HistoryCriteria pHistoryCriteria)
setIdRange(org.apache.commons.lang.math.LongRange pIdRange)
setIds(Set<Long> pIds)
setModificationDateRange(DateRange modificationDateRange)
setName(String pName)
setPattern(String pPattern)
setQueueIds(Set<Long> pQueueIds)
setScopeIds(Set<Long> pScopeIds)
setStatus(TicketCriteria.Status pStatus)
setSubject(String pSubject)
setUserCriteria(Set<TicketUserCriteria> pUserCriteria)
```

Fig. 1: Setter Methods of Class TicketCriteria, CM Version 6.9.3

The *TicketCriteria* object has to be handed over to the *WorkflowContextService* which is implicitly available as singleton *workflowAPI* in each script. Please see the following examples and refer to the *ConSol*CM Workflow API Java* documentation for details about classes and methods.

Example 1: General Example to Search for Tickets

Search for tickets

```
def ticketCrit = new TicketCriteria()
ticketCrit.subject = "TICKET_SUBJECT"
ticketCrit.setQueueIds([new Long(workflowApi.getQueueByName("QUEUE_NAME").id)] as Set)
ticketCrit.setFields([new StringField(new FieldKey("FIELD_GROUP", "FIELD_NAME"),
"SEARCH_VALUE")] as Set)
def foundTickets = workflowApi.getTicketsByCriteria(ticketCrit)
def firstTicket = foundTickets?.first()
```

Example 2: Find All Tickets with the Same Service as the Current Ticket

The following example is taken from a workflow of a help desk environment. When the ticket has been created and the service has been set from a list, the workflow should check automatically if there are other open tickets with the same service. A dependent *enum* is used for the services:

- **1st level**
  Several categories, one of them is *HARDWARE*.
- **2nd level**
  Exists only when *HARDWARE* was selected in the 1st level. In the 2nd level, hardware categories are listed.

Find tickets with the same service as the current ticket

```
def crit = new TicketCriteria()
crit.setStatus(TicketCriteria.Status.OPEN)
Set<AbstractField> cfs = new HashSet<AbstractField>();
if (serv1.getName().equals("HARDWARE")){
    def serv2 = ticket.get("Service_Fields.Hardware")
    cfs.add(new EnumField(new FieldKey("Service_Fields", "Hardware"), serv2));
} else {
    cfs.add(new EnumField(new FieldKey("Service_Fields", "Service"), serv1));
}
crit.setFields(cfs)
List<Ticket> foundTickets = workflowApi.getTicketsByCriteria(crit);
```

Example 3: Search for Tickets by Unit

In this example, we look for the *Account Management* ticket for a certain company.

Search for tickets by unit

```
import com.consol.cmas.common.model.scripting.unit.PostActionType
import com.consol.cmas.common.model.scripting.unit.PostActionParameter
import com.consol.cmas.common.model.customfield.Unit
import com.consol.cmas.common.model.ticket.TicketCriteria
import com.consol.cmas.common.model.customfield.ListField
import com.consol.cmas.common.model.customfield.ContactReferenceField
import com.consol.cmas.common.model.customfield.UnitReferenceSearchField
import com.consol.cmas.common.model.customfield.ContactReferenceSearchField
import com.consol.cmas.common.model.customfield.meta.FieldKey
import com.consol.cmas.common.model.ticket.Ticket
import com.consol.cmas.common.model.ContactTicketRole
import com.consol.cmas.common.model.customfield.StringField
import com.consol.cmas.common.model.scripting.unit.UnitActionScriptResult
//get AM queue for search
def q_id = (workflowApi.getQueueByName("AccountManagement")).id
def q_ids = new HashSet()
q_ids.add(q_id)
//find AM ticket for the company
def crit = new TicketCriteria()
crit.setQueueIds(q_ids)
// Create List Field Key
def contactSearchListFieldKey  = new FieldKey("queue_fields","contacts")
// Prepare List Field
def contactsListField = new ListField(contactSearchListFieldKey )
// Create Memberfield Key
def contactSearchFieldKey = new FieldKey("queue_fields","contacts_member")
// Create Unit Memberfield with Unit and Ticket-Main Role
def contactsMember = new ContactReferenceSearchField(contactSearchFieldKey, unit,
ContactTicketRole.MAIN_ROLE)
// Put Member Field in Unit List Field
contactsListField.addChild(contactsMember)
// Put prepared fields into TicketCriteria
crit.setFields([contactsListField] as Set)
// Search ... and Result
def foundTickets = ticketService.getByCriteria(crit)
println "Found tickets: ${foundTickets}"
if ( foundTickets ) {
  def AM_tic = foundTickets.first()
  def AM_tic_id = AM_tic.id
}
```

6.8.3 Searching for Units (Contacts and Companies)

To search for units (i.e. for contacts and/or companies) you have to create the *UnitCriteria* object. The following fields can be set (see also the respective *setter* methods in the following picture):

- Customer group
- System-specific data object group fields
- Unit IDs
- Patterns for units
- Phone number (new in CM version 6.9.3, used for CM/Phone)
- TicketCriteria
- UnitDefinition name
- Boolean UseInCriterion

Then you use the *unitService* to get the search result.

```
setCustomerGroupIds(Set<Long> pCustomerGroupIds)
setFields(Set<AbstractField> pFields)
setGroupNames(Set<String> pGroupNames)
Deprecated.
setIdRange(org.apache.commons.lang.math.LongRange pIdRange)
setIds(Set<Long> pIds)
setPattern(String pPattern)
setPhoneNumber(String pPhoneNumber)
setTicketCriteria(Set<AbstractField> pCallUnitReferences, TicketCriteria pTicketCriteria)
setUnitDefinitionNames(Set<String> pUnitDefinitionNames)
setUseInCriterion(boolean pUseInCriterion)
```

Fig. 2: Setter Methods of Class UnitCriteria, CM Version 6.9.3

Example 1: Search for Contacts by First Name and Last Name

Search for contacts by first name and last name

```
def unitCrit = new UnitCriteria()
unitCrit.setFields([new StringField(new FieldKey("UNIT_GROUP_NAME", "firstname"), "Max"),
                    new StringField(new FieldKey("UNIT_GROUP_NAME", "lastname"), "Mustermann")]
as Set)
def foundContacts = unitService.getByCriteria(unitCrit)
def firstContact = foundContacts?.first()
```

General Syntax for Unit Search by Enum Value

Search for units by enum value (general syntax)

```
import com.consol.cmas.common.model.customfield.UnitCriteria
import com.consol.cmas.common.model.customfield.EnumSearchField
import com.consol.cmas.common.model.customfield.meta.FieldKey

def unitCrit = new UnitCriteria()
def companyEnumField = new EnumSearchField(new FieldKey("customer", "company"),
[enumService.getValueByName("ENUM_GROUP_NAME",ENUM_VALUE_NAME)] as Set)
unitCrit.setFields([companyEnumField] as Set)
unitService.getByCriteria(unitCrit).each { foundContact ->
  println "Processing found contact: "+foundContact.get("name")

}
```

Example 2: Search for Units by Enum Value

Search for units by enum value (example)

```
def unitCrit = new UnitCriteria()

//all other UnitCriteria init operations skipped

// this is the requested value inside the list:
def secLvl = ticket.get("transportEntryData.securityLevel")

//ShipperData/securityLevel is the path of the EnumField inside the list
def secLvlEnumFieldKey = new FieldKey("ShipperData","securityLevel")

//create the template field with FieldKey and our value to search for
def secLvlTemplateField = new EnumField(secLvlEnumFieldKey, secLvl)

//ShipperData/securityLevels is the path of the list itself
def secLvlListTemplateFieldKey = new FieldKey("ShipperData","securityLevels")

//init the template list with the value to be searched for
def secLvlListTemplateField = new ListField(secLvlListTemplateFieldKey,[secLvlTemplateField])

// put the template list into the UnitCriteria object
def unitCrit.setFields([secLvlListTemplateField] as Set)

// Search ... and Result
def shippers = unitService.getByCriteria(unitCrit)
```

6.9 Debug Information

6.9.1 Introduction

Sometimes you might want to check the output of a workflow or Admin-Tool script by using debug output into log files. In ConSol*CM, the debug output usually is written to *server.log* which is located in the following path:

- **In JBoss:**
  `<SERVER_HOME>\log\server.log`
- **In Oracle WebLogic:**
  `<DOMAIN_HOME>\cm-logs and <DOMAIN_HOME>\cmrf-logs\server.log`

The logging configuration can be changed by editing the *log4j* configuration file. If you have defined a non-standard log path, you will know where to find the *server.log* file.

As an alternative, you can write information into the ticket as text.

6.9.2 Using Statements for Debug Output

Debug Output to server.log File

The following statements can be used to write log information to the *server.log* file. This works in workflow scripts as well as in Admin-Tool scripts.

- `println 'This is my debug message.'`
- `println("This is my debug message.")`
- `log.info("This is my debug message.")`
- `log.info "This is my debug message."`

Attention:

In a WebLogic system, usually the *log.info* statement has to be used. The *println* might not work.

Debug Output as Text Entry in Ticket

If you would like to display the information to the ticket (e.g. because you do not have access to the file system where the log files are stored) you can write the text into the ticket as regular comment:

- `workflowApi.addTicketText('This is my debug message', 'This is the subject of my debug message', false)`

Debugging ConSol*CM Standard Scripts

In ConSol*CM standard scripts, e.g. *createTicket.groovy*, you will find statements similar to the following:

Debug entry in ConSol*CM standard e-mail script

```
if (log.isDebugEnabled()) {
        log.debug("Extracted email from from-field is $email")
}
```

To activate the debug output, i.e. to have CM write the debug information into the log file, you have to set the log level of the respective module (here: e-mail) to *DEBUG*. This is done in the file *jboss-log4j.xml*.

We will not elaborate on this topic here. If you would like to learn more about CM logging, please refer to the *ConSol*CM Operations Manual*.

7 Best Practices

7.1 The Basic Organization of a Workflow: Using Scopes

One of the first things you have to consider, when you start making a concept for a workflow, is the number and organization of scopes.

> **Information:**
>
> Of course you can always modify the workflow in later steps, but this might have implications for existing tickets, views, and reports. This is particularly significant if the workfow is used in a production environment.

Consider the following points when setting up the basic structure of a workflow:

- Which trigger should be active for the ticket in which states of the process?
  For example, should a time trigger, which monitors the new tickets, also be active for tickets which are already in progress? Or, should a mail trigger be active when the ticket has been finished by the engineer?
- Which views are required?
  Views are based on the position of tickets in scopes, see *ConSol*CM Administrator Manual* section *View Administration* for details.

7.1.1 Variant A: Use of a Global Scope

A global scope is a scope which contains all other scopes of the workflow. You might want to use such a global scope because some processes require reactions to events during the entire process. Those events are implemented using triggers which are attached to the global scope. For example, if you want to supervise for the entire process, if an e-mail has been received, you attach a mail trigger (see section Mail Triggers) to the global scope. All sub-scopes of the global scope inherit the sensitivity to this trigger. If the e-mail should only be supervised for a sub-scope, you can attach the mail trigger to this sub-scope.

The same applies to all kinds of triggers, i.e. business event triggers (see section Business Event Triggers) and time triggers (see section Time Triggers).

The START node always has to be positioned outside the Global Scope!

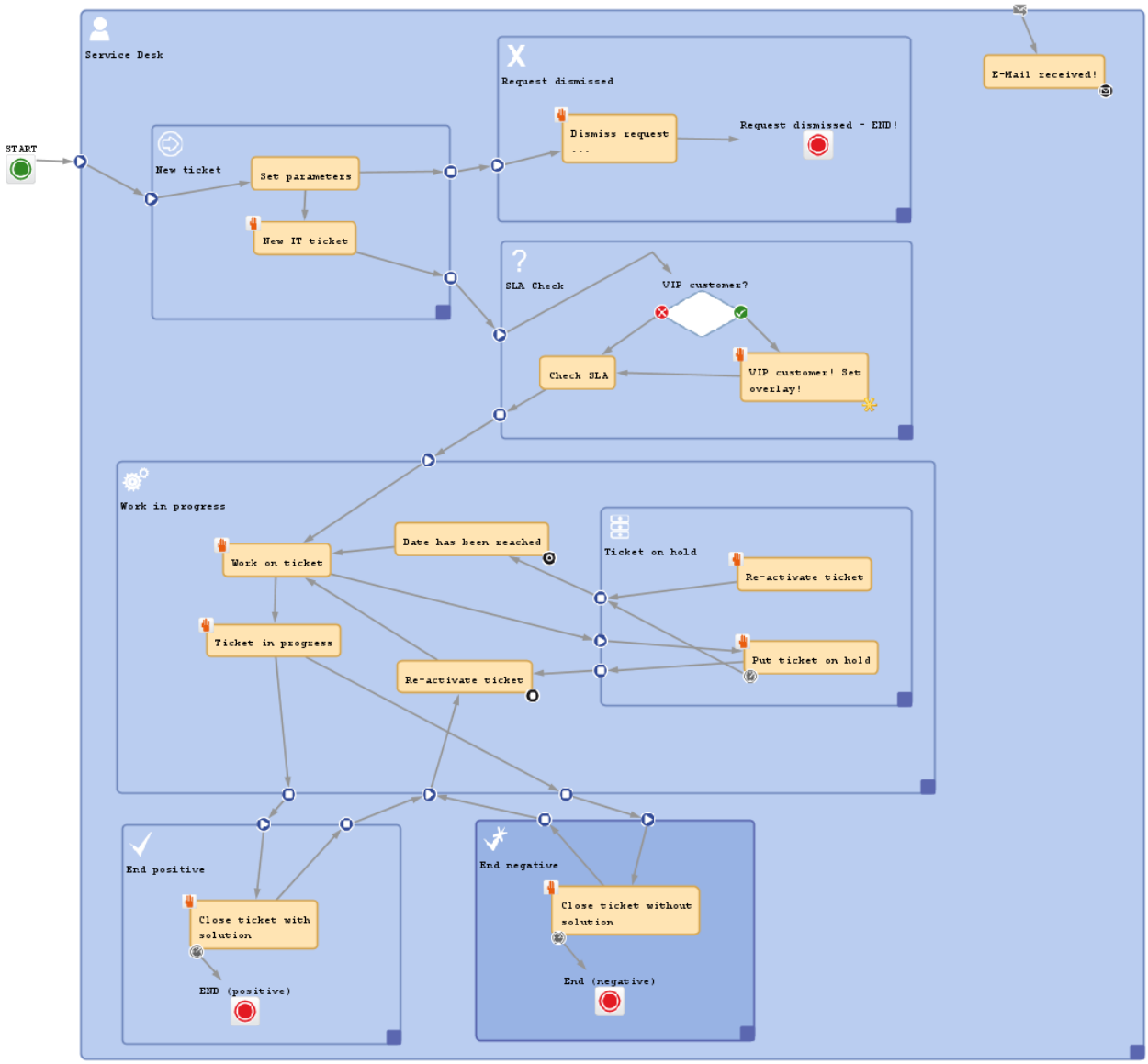


Fig. 1: ConSol*CM Process Designer - Workflow with Global Scope

Please keep in mind that you can always use triggers in inner scopes which will then consume the event (see section Firing Order of Business Event Triggers as an example for business event triggers). For example, if you would like to use a mail trigger in the entire process in the global scope but you need a certain reaction of the ticket in the *Finished* scope, you can use a mail trigger which is attached to the *Finished* scope.

7.1.2 Variant B: Use of Three or More Main Scopes

An alternative way to construct a workflow is to use three or more main scopes:

- New tickets
- In progress (only here, a mail trigger is applied)
- Closed tickets (in one or more separate scopes)

The following picture shows an example for a workflow which has been built according to this principle.

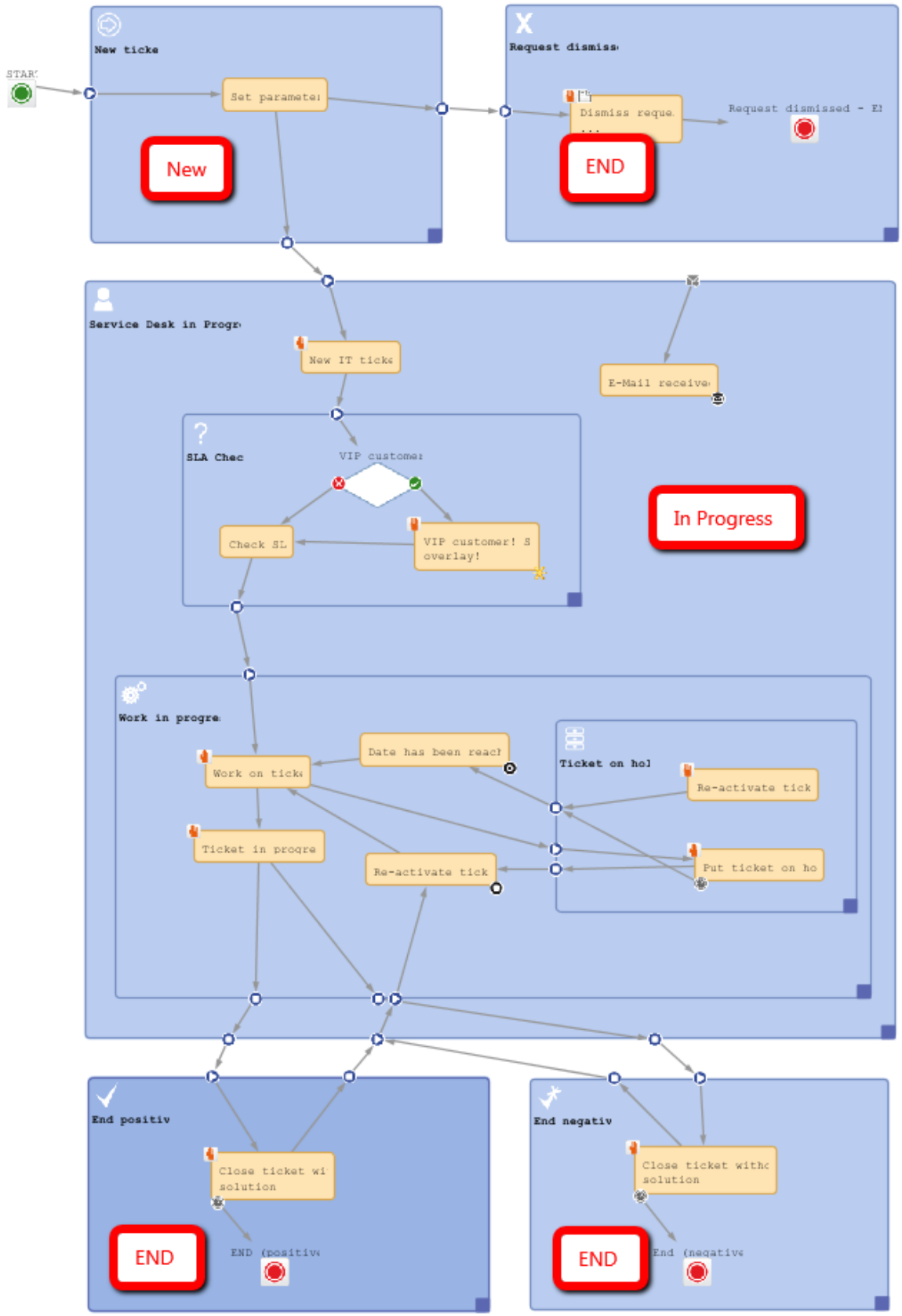


Fig. 2: ConSol*CM Process Designer - Workflow with Three Types of Main Scopes

7.2 The Position of the START Node

The best position of the START Node depends on the use of triggers in the following scope. If time triggers are used in the first scope, where tickets are forwarded after the start node, the start node should be placed outside the scope. In case the start node is placed inside the first scope, the time trigger might not be initialized correctly. So place the start node in the default scope.

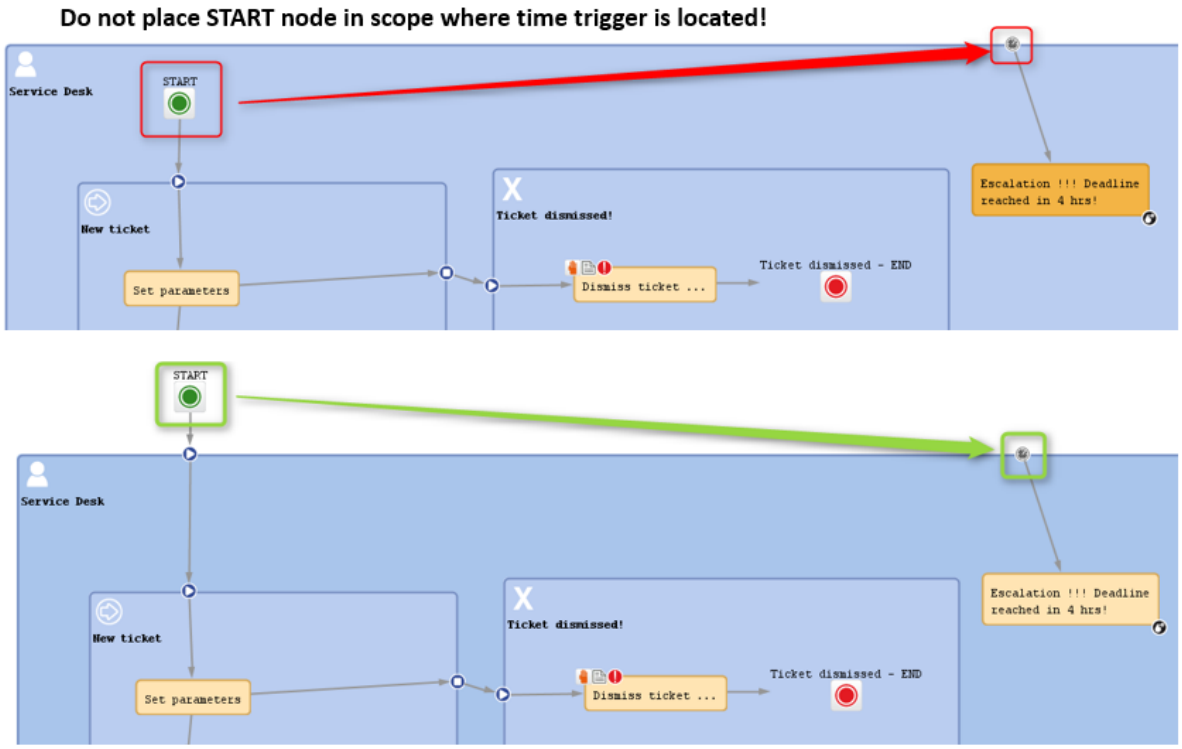


Fig. 3: ConSol*CM Process Designer - Position of START Node

7.3 Store Some Workflow Scripts in the Admin-Tool

For scripts, which are used over and over again in workflow activity and/or precondition scripts, it might be better to store them in the *Script* section of the Admin-Tool and call them from the workflow script.

7.3.1 When to Use Admin-Tool Workflow Scripts

We would neither recommend to always use this method nor would we advise against it. We will illustrate the advantages and disadvantages of this approach and you can then decide for yourself where in your system you want to apply it.

The **advantages** of storing workflow scripts in the Admin-Tool are the following:

- The script is stored only once and has to be maintained/changed at only one place.
- Changes of the scripts are executed in the system just in-time, no deployment (as for workflows) is required.

The **disadvantages** of storing workflow scripts in the Admin-Tool are the following:

- The process logic is stored at two separate places, i.e. you always have to work with the Process Designer as well as with the Admin-Tool to see the entire process.
- The Script Editor in the Admin-Tool is not as comfortable as the Workflow Script Editor.
- Most objects have to be imported into Admin-Tool scripts, because they are not present implicitly.
- A workflow export alone is not sufficient to move the workflow, because scripts in the Admin-Tool are not included in the export.

7.3.2 How to Use Admin-Tool Workflow Scripts

Admin-Tool scripts which are used in the workflow have to be of type *Workflow*. An Admin-Tool script is always called from the workflow using the interface *ScriptProvider*.

Calling an Admin-Tool script from the workflow

```
def scriptProvider = scriptProviderService.createDatabaseProvider("scriptName.groovy")
def r = scriptExecutionService.execute(scriptProvider)
```

Calling an Admin-Tool script from the workflow with use of parameters

```
// Create the scriptProvider for the required Admin-Tool script, here "scriptName.groovy"
def scriptProvider = scriptProviderService.createDatabaseProvider("scriptName.groovy")

// Define a HashMap with the key-value pairs which you would like to pass to the Admin-Tool
def params = [ "templateName": "newCustomer" ]

// Execute the script. The passed parameters are available in the Admin-Tool script. In the
// example, the variable templateName does not have to be defined in the Admin-Tool script
// but it is present based on the definition in the passed HashMap.
// The variable r will contain the return value of the script or Null if there is no return
// value
def r = scriptExecutionService.execute(scriptProvider, params)
```

7.4 Consider the Use of Trigger Combinations Well

> **Attention:**
>
> Beware of unnecessary trigger executions! They will consume resources and slow down application performance.

Example 1:
This example shows many business event triggers in one big *global* scope.

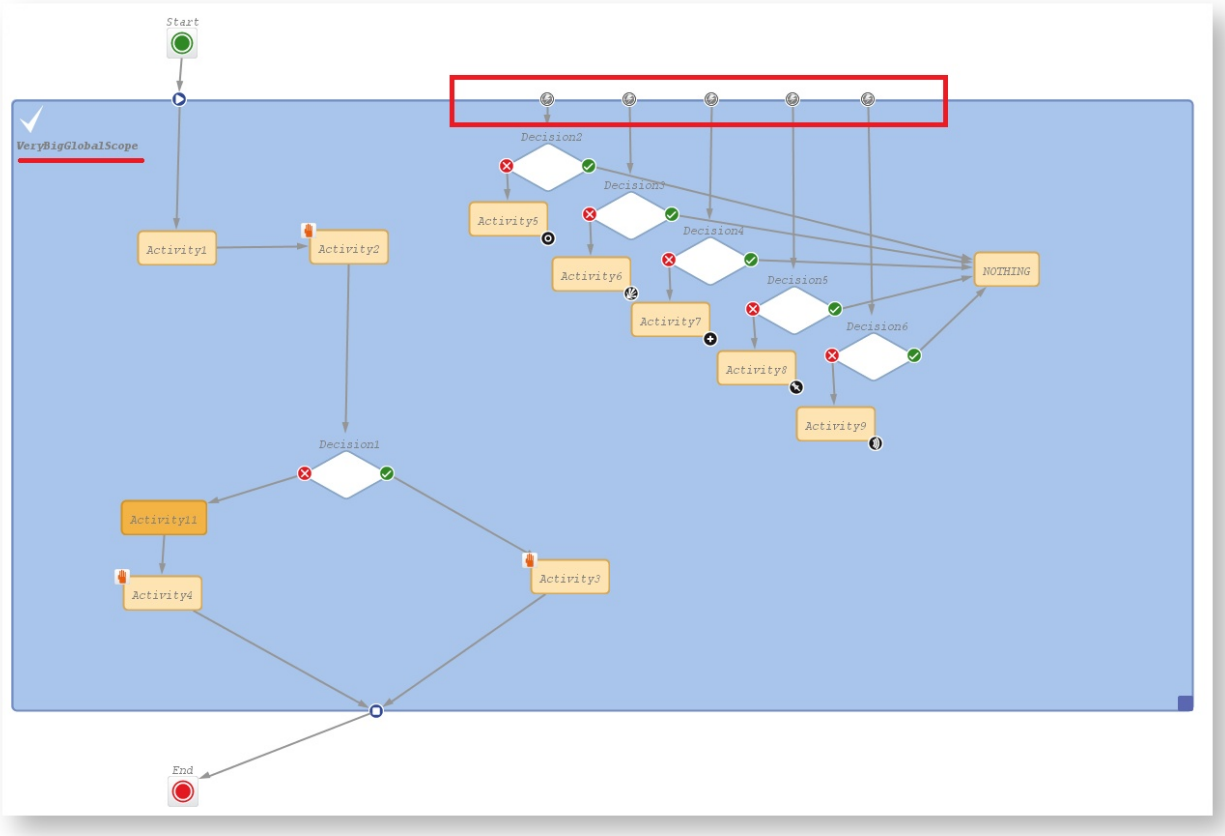


Fig. 4: ConSol*CM Process Designer - Scope with Triggers

Example 2:

If it is possible, please use triggers in the smallest scope possible (in this example, the trigger with *Decision6* was moved to a smaller scope).

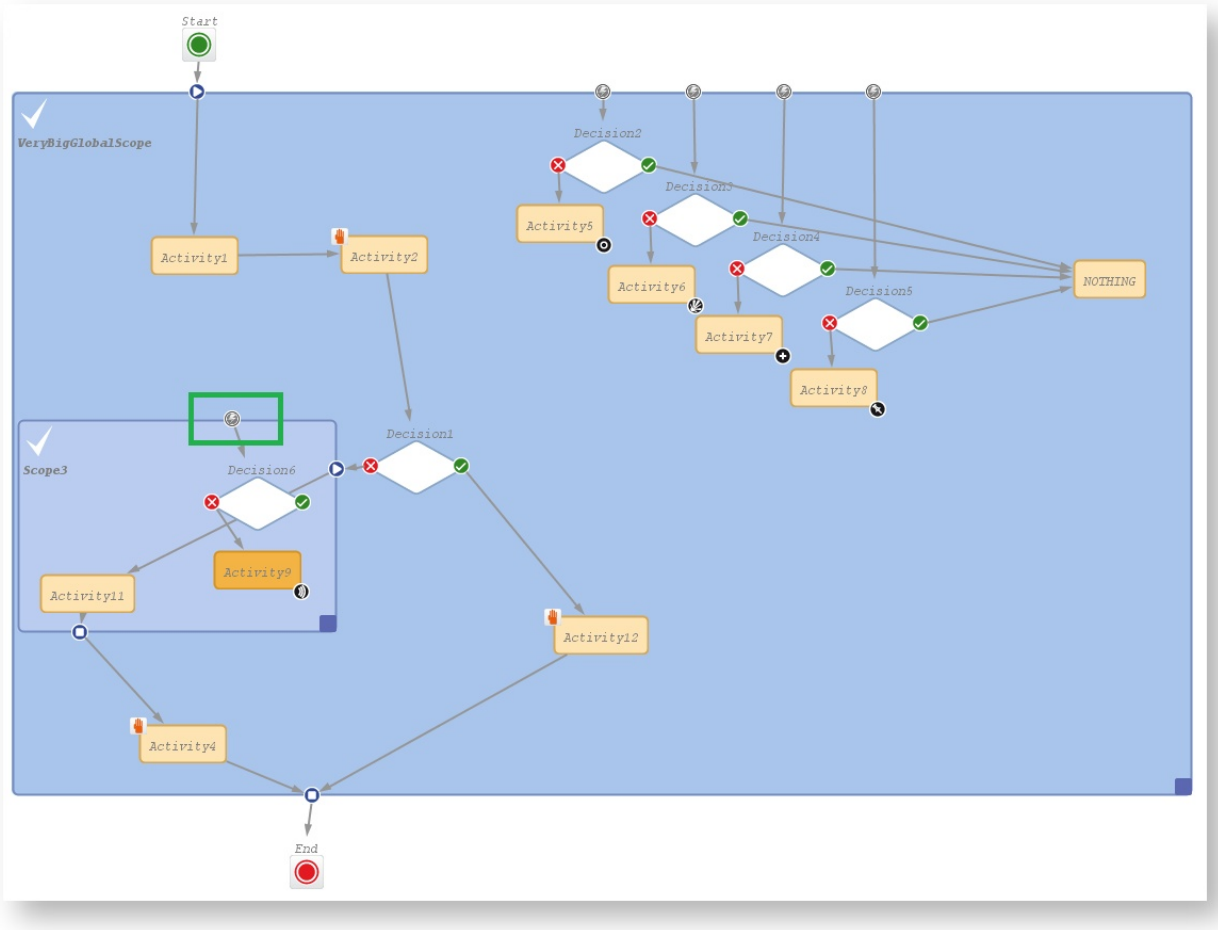


Fig. 5: ConSol*CM Process Designer - Move Trigger to Smaller Scope

Example 3:
If it is **not** possible to move triggers to smaller scopes and you do not want to call all of the triggers while executing some activity, move this activity to an outside scope without any triggers.

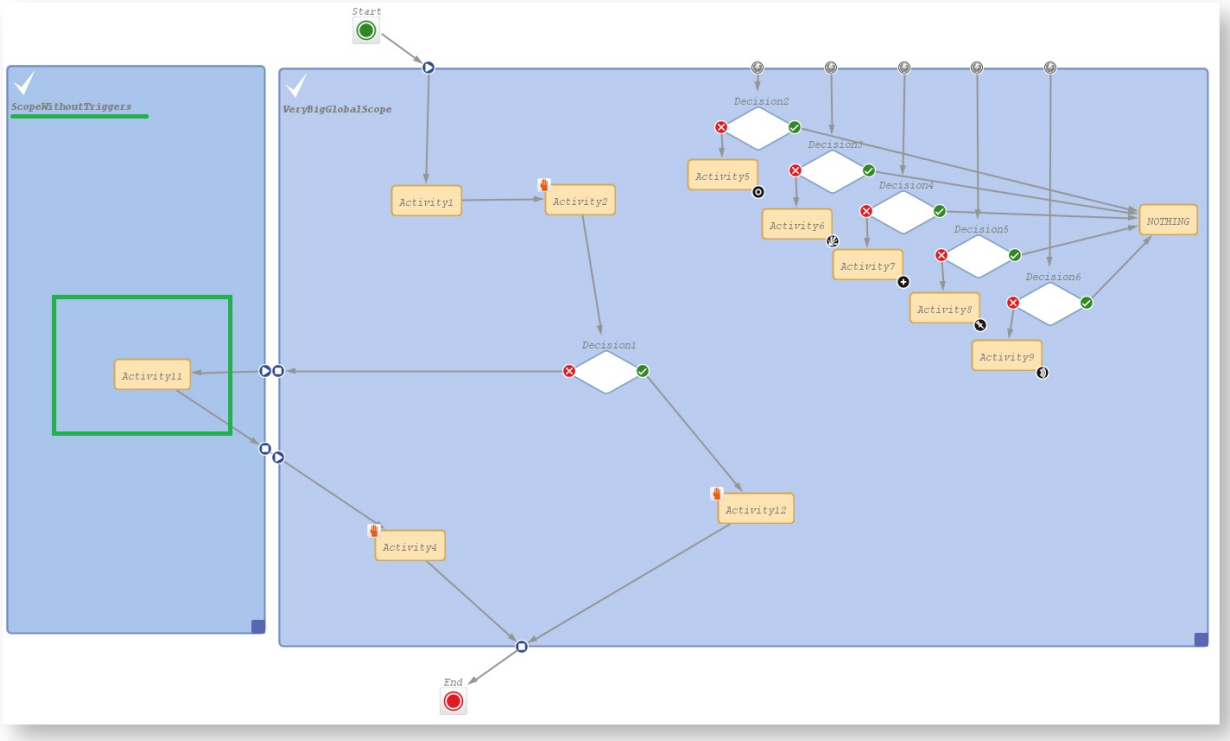


Fig. 6: ConSol*CM Process Designer - Separate Scopes with and without Triggers

In this example, the position of *Activity11* is optimized. It triggered many *Decision* calls and all of them went to *NOTHING*. Executing *Acitivity11* outside of the global scope keeps a good quality of workflow performance!

7.5 Do Not Trigger Ticket Update Events If Not Really Required

> ⚠ **Attention:**
>
> Beware of unnecessary ticket update events (Java class *TicketUpdateEvent*)!

For example, assigning the current engineer (the engineer who is logged in and working with the Web Client) to a ticket can be done in two ways. In one solution a ticket update event is fired, in the other this does not happen. If it is not necessary for a business case to throw a *TicketUpdateEvent*, avoid it, because an unnecessary call of *TicketUpdateEvent* causes a decrease in performance.

Code which triggers TicketUpdateEvent

```
//this method throws a TicketUpdateEvent after assigning the current engineer to the ticket
workflowApi.assignEngineer(workflowApi.currentEngineer)
```

Code which does not trigger TicketUpdateEvent

```
//this method does NOT throw a TicketUpdateEvent!
ticket.setEngineer(workflowApi.currentEngineer)
```

7.6 How to Use the Disable Auto Update Parameter

Attention:

Use the *disable auto update* flag for workflow components with care!

Please remember that a ticket udpate event is by default fired after every activity execution. A ticket update event is an operation that has a great impact and must be used with care!

To avoid performance problems, you can use the *disable auto update* flag. It depends on the business logic, if it makes sense to use this flag or not.

For example, when we have a series of automatic activities, a good practice is:

- The **1st** automatic activity has the *disable auto update* flag **on**.
  (It will **not** call the ticket update service method after activity execution.)
- The **2nd** automatic activity has the *disable auto update* flag **on**.
  (It will **not** call the ticket update service method after activity execution.)
- The **3rd** automatic activity has the *disable auto update* flag **on**.
  (It will **not** call the ticket update service method after activity execution.)
  ...
- The **last** automatic activity has the *disable auto update* flag **off**.
  (It **will** call *TicketUpdateEvent* **once**, at the **end** of the pipeline!)

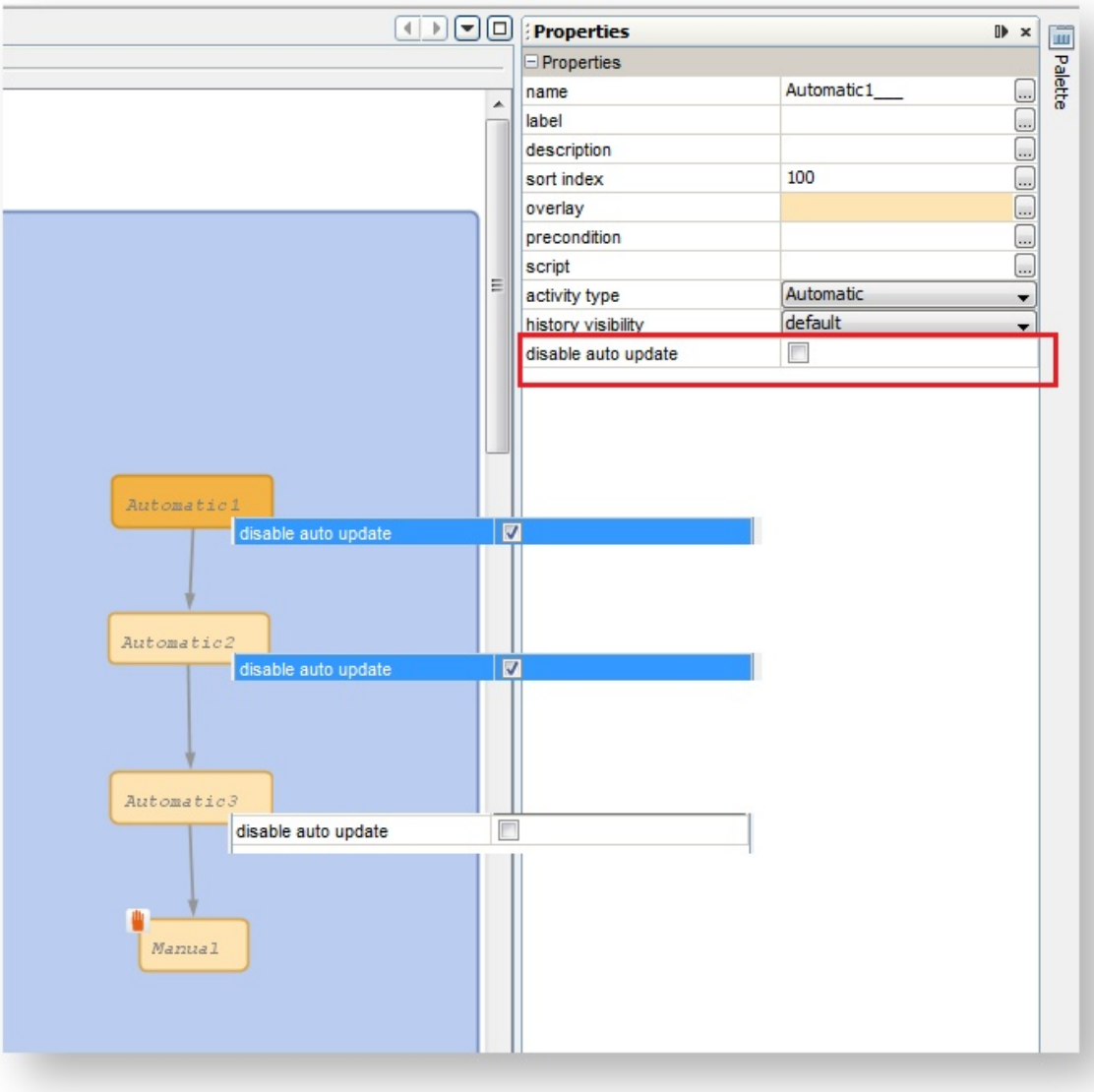


Fig. 7: ConSol*CM Process Designer - Activities with "disable auto update" Option

7.7 Avoid Self-Triggering Business Event Triggers

When you use a business event trigger which is followed by an automatic activity, be careful that in this automatic activity the fields or objects, which trigger the business event trigger, are not changed again (which would fire the trigger again)!

If the use case requires that the fields, which caused the firing of the trigger, have to be changed again, then the logic, where the fields are changed, has to be placed in an activity outside the scope which hosts the trigger.

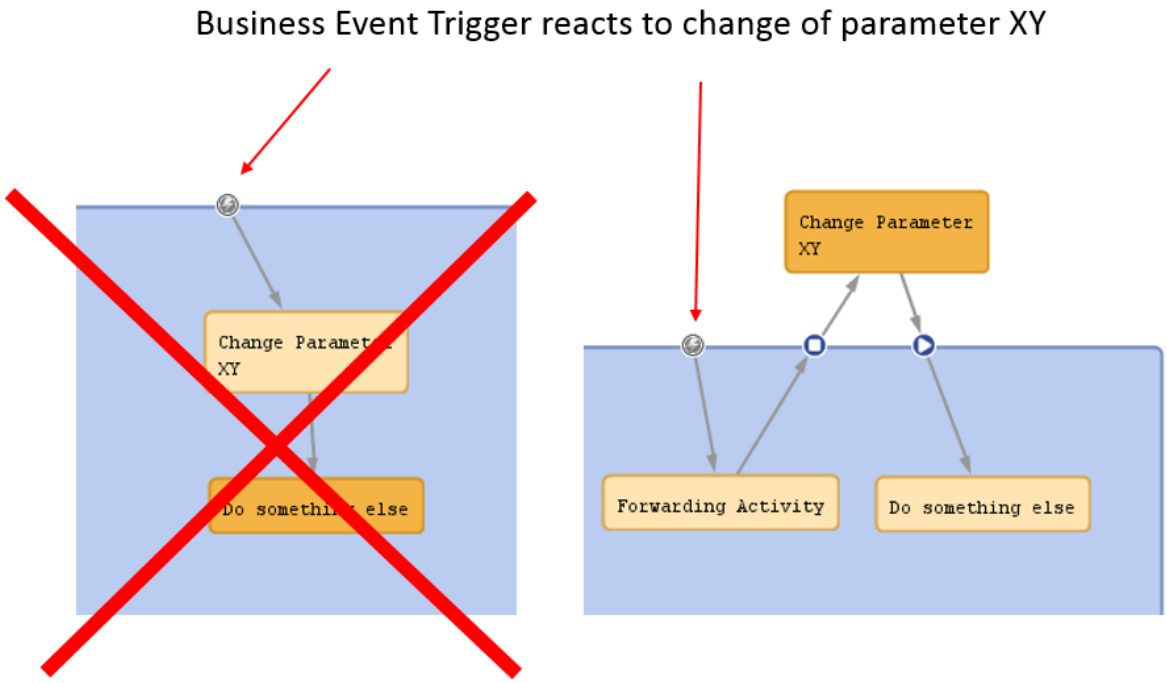


Fig. 8: ConSol*CM Process Designer - Avoiding Self-Triggering Business Event Triggers

8 Deploying Workflows

8.1 Introduction and Workflow Life Cycle

During the development of a workflow you use the following functions which reflect the workflow life cycle:

- Load  the workflow or create a new workflow, e.g. version 1.2.
- Edit the workflow.
- Save  the workflow as a new version. A new version number will be used, e.g. 2.0.
- Continue editing the workflow.
- Save  the workflow in the current version, e.g. version 2.0.
- Continue editing the workflow.
- Deploy  the workflow. This will *save* and *deploy* the workflow, e.g. version 3.0.
  A deployed workflow always has an increased major number compared to the last saved version.
  The workflow which was active/deployed before is now no longer active, but the new version of the workflow is in operation at once. The ConSol*CM system does not have to be stopped.
  The new version is marked in bold characters and with status *currently deployed* in the workflow list which is opened for the *Load* and *Delete* operations.
  After this step, the next saved version will be saved as *new version*.

Attention:

Make sure you are aware of the number of tickets which have to be transferred when a new workflow is deployed! The deploy operation might take some time in large environments! See section Actions During Workflow Deployment.

8.2 Engineer Rights Required for Workflow Deployment

An engineer who is supposed to deploy workflows must have at least one role with one of the following access rights:

- **Global Permissions:**
  Administrate
- **Workflow Permissions:**
  Deploy workflow

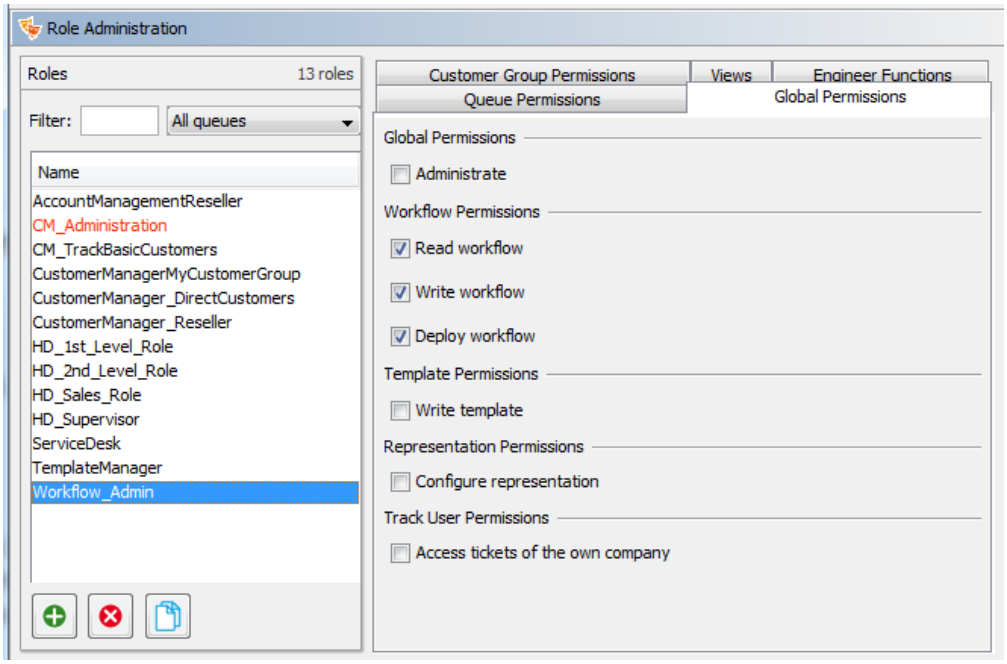


Fig. 1: ConSol*CM Admin-Tool - Engineer Permissions for Deploying Workflows

8.3 Actions During Workflow Deployment

When a workfow is deployed, it will be active at once. Thus, consider well what will happen to open tickets which are in a queue where the new workflow will be applied. They will be transferred to the new workflow.

In case you have performed one or more of the following steps:

- removed one or more activities
- added one or more automatic activities
- added one or more triggers

the following actions will be initiated after you have pressed the *Deploy* button**.**

You will be prompted for a decision concerning the open tickets in the respective queues which cannot stay at their previous position within the process because the workflow architecture was changed:

1. Stay as close as possible to the previous position (default).
2. Let all those tickets start the process from the beginning.

In case you choose the first option (*keep position*), the following actions will be performed:

1. The transfer of tickets starts.
2. The name of the ticket's last executed activity is compared to the names in the current workflow definition. If the ticket's activity is no longer in the workflow definition, a new target activity for the ticket must be found.
3. The *History* for the ticket is loaded. The transfer engine iterates over all activities executed from the beginning of the process instance and tries to find one which would be suitable, i.e. which
    a. is still present in the workflow definition,
    b. is not a trigger target element,
    c. is not a dead end activity.

Each ticket which cannot keep its position will be moved to the suitable position according to those criteria. In any case the tickets will be moved backwards, never forwards, within the workflow.

For a *summary* of all ticket transfers click on *View* in the main menu and select *Show ticket transfer history*.

- **Workflow name**
  Name of the workflow.
- **Version**
  Version of the old workflow.
- **Start time**
  Start of the transfer. Will be the start time of the *Deploy* operation.
- **End time**
  End of the transfer. After this time the new workflow will be in full operation.

- **Transferred tickets**
  Number of tickets which have been transferred, i.e. which had to be touched by the system during workflow deployment. Should be identical to the sum of open tickets in all queues which use the workflow.
- **Details**
  Additional information concerning the deployment with ticket transfer.

In the bottom right corner of the Process Designer GUI, the overall status of the ticket transfer is displayed.

9 Appendix A - List of Annotations

9.1 Alphabetical List of Field Annotations (up to Version 6.9.3)

	Name	Annotation Type	Description	Values	Comment
A	accuracy	validation	For date fields, to define the level of detail displayed.	date (default)	Show date without time.
				date-time	Show date with time.
				only-time	Show only time, no date.
B	boolean-type	component-type	Definition of the layout of a boolean field.	check box (default)	Field that can be checked (set to *false* by default).
				radio	2 radio buttons (yes/no) for selection (only one can be active).
				select	Drop-down-field with 2 values (yes/no).
C	colspan	layout	Defines how many columns are reserved for the field in the layout.	<number>	Number of columns.
	contact search result column	search-result	Identifies whether the field should be presented in the search result by default.	true	Remove the annotation if the field should not be visible by default.

	Name	Annotation Type	Description	Values	Comment
	contains contacts	ticket contact relation	Used only for list field definition, indicates that the defined fields can hold contact information.	true / false	Necessary to distinguish if the list is shown with the contact (true) or with the ticket (false).
D	dialable	phone commander	Defines a field with a phone number.	true	Remove the annotation if the field should not hold a dialable phone number. **Version 6.9 and higher:** Used with CM/Phone only. Marks a phone number as automatically dialable for outgoing calls for the CTI system.
E	email	validation	Used for e-mail addresses to check if the format is correct, i.e. if <name>@<domain> has been entered.	true	May be used with *string* custom fields. Remove the annotation if the format should not be checked.

	Name	Annotation Type	Description	Values	Comment
	enum field with ticket color	ticket display	Defines the background color of the ticket icon for ticket list and ticket.	true / false	The field has to exist within enum administration where lists, values, and colors are defined.
	enum-in-search-type	component-type	Defines whether an enum field used in a search accepts search over multiple values.	single (default) / multiple	Accepts search over multiple values if value *multiple* is set.
	enum-type	component-type	Layout definition of list display.	select (default)	Drop-down list for selection.
				radio	List of radio buttons to select (only one option can be active)
				autocomplete	Drop-down list for selection where the field is an input field used to filter the list.

	Name	Annotation Type	Description	Values	Comment
F	field-group	layout	Allows grouping of fields in *view* mode. Annotation is ignored in *edit* mode.	<string>	To group fields the same *string* value has to be set in the annotation of each field. Two or more custom fields are bound when they share the same value of this annotation. The group of coupled custom fields is shown only if all of them have values set.
	field indexed	indexing	Used to indicate that the field may be indexed.	transitive	All data is displayed (ticket and customer).
				unit	Used for customer data. Only the unit and the parent unit (i.e. company) is given as a search result, no tickets are provided.
				local	Used for customer data. Only the unit is given as a search result, no company and no tickets are displayed.

	Name	Annotation Type	Description	Values	Comment
				not indexed	Field is not indexed.
	fieldsize	layout	Displayed field size within ticket layout.	<rows>;<cols>	For *string* custom fields with annotation *text-type* and value *textarea*. <rows> defines the number of displayed rows and <cols> defines the number of characters displayed per row. Used only for layout purposes.
				<number>	For enum custom fields. Defines how many values are directly visible in the list box. Used only for layout purposes.

	Name	Annotation Type	Description	Values	Comment
	format	validation	Used to check the correct format of date fields.	<date format>	The pattern for the date is based on *SimpleDateFormat*, e.g. dd.mm.yyyy. Remember to set the proper *colspan* when you want to add hours/minutes. See http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html for date format reference.
G	groupable	cmweb-common	Enables grouping in the ticket list.	true	Used only with enum custom fields. Remove the annotation if you want to disable grouping.

	Name	Annotation Type	Description	Values	Comment
L	label-group	layout	Indicates a group of fields along with its descriptive label in *view* mode. Annotation is ignored in *edit* mode.	<string>	Indicates a group of custom fields along with its descriptive label. The annotation is used in *presentation* mode, ignored in *edit* mode. The group can have exactly one label (a custom field of type *string* with assigned additional annotation *text-type* with value *label*). The label is shown when at least one custom field from its group has a value set. All fields with the same label value are grouped and displayed under this label. The annotation *label-group* has to be assigned to the label, too.

	Name	Annotation Type	Description	Values	Comment
	label-in-view	layout	Shows custom field value as a label in *view* mode. Annotation is ignored in *edit* mode.	true	Remove the annotation if the label should not be visible in *view mode*.
	ldapid ***only version 6.9 and higher***	contact authentication	Used in a data object group of type *customer*, for the data object group field which contains the LDAP ID for CM/Track authentication.		Indicates that this field will be used as an LDAP ID in the authentication process. Data type *string* is required. Since the definition is made on customer group level, the LDAP authentication can be run in mixed mode. I.e. use LDAP for some customer groups and regular authentication for other customer groups.
	leave-trailing-zeros	common	Used for the display of fixed point numbers.	true / false	Remaining zeros of the fractional part are not cut off when value is *true*.

	Name	Annotation Type	Description	Values	Comment
	list-type	component-type	Disables the *add* and/or *delete* options for custom fields of type *list* or *struct*.	fixed-size	It is not possible to add or delete fields/rows.
				non-shrinkable	It is not possible to delete fields/rows.
				non-growable	It is not possible to add fields/rows.
M	matches	validation	Checks if input of *string* custom fields matches the given RegEx.	<string>	May be used with *string* custom fields.
	maxLength	validation	Defines the maximum length of input for *string* custom fields.	<number>	May be used with *string* custom fields.
	maxValue	validation	Defines the maximum value for *number* custom fields.	<number>	May be used with *number* custom fields, i.e. *number* and *fixed-point number*.
	minLength	validation	Defines the minimum length of input for *string* custom fields.	<number>	May be used with *string* custom fields.

	Name	Annotation Type	Description	Values	Comment
	minValue	validation	Defines the minimum value for *number* custom fields.	<number>	May be used with *number* custom fields, i.e. *number* and *fixed-point number*.
N	no-history-field	performance	Indicates that a single custom field should not be historized. Overwrites the group annotation *no-history*.	true / false	Annotation is active if value is set to *true*. For fields that should be stored but not be visible in history use annotation *visibility configuration*.
O	order-in-result	layout	Shows field as a column in the search result list at given position.	<number>	The columns are sorted in ascending order.
P	password	contact authentication	Indicates that this field will be used as a password in the authentication process.	<string>	Used for CM/Track.
	position	layout	Defines the position of a field within a grid layout.	<number>;<number>	Values define *row* and *column* (row;column), numbering starts at 0;0. If no values are set, the custom field will take the next free grid cell.

	Name	Annotation Type	Description	Values	Comment
			Defines the position of a field within a list (struct).	0;<number>	Only the *column* value is used, the *row* value is ignored.
R	readonly	common	Used to indicate that the custom field cannot be modified.	true / false	Field is read only if value is set to *true*. Lack of value or any value except *false* is also treated as *true*.
	reportable	dwh	Indicates that the field is reportable and that it should be transferred to the DWH.	true / false	Field is reportable if value is set to *true*.
	required	validation	Indicates that this is a required field.	true / false	Field is required if value is set to *true*. The user cannot save the ticket without having entered a value in a required field. In the Web Client, required fields are marked by a red asterisk.
	rowspan	layout	Indicates how many rows within the layout are occupied by this field.	<number>	Number of rows.

	Name	Annotation Type	Description	Values	Comment
S	sortable	cmweb-common	Used to enable sorting of the ticket list.	true	Used for custom fields of type *DATE* or of type *enum*. Remove the annotation if you want to disable sorting. For *enum* fields : Works only if order index is set for all values of the *enum* field.
T	text-type	component-type	Defines the possible types of a *string* field.	text (default)	Single-line input field.
				textarea	Multi-line input field.
				password	Input field for passwords. Password will be displayed as ******* in *view* mode.
				label	Input will be displayed as a label, i.e. the field is displayed only, no input is possible.

	Name	Annotation Type	Description	Values	Comment
				url	Input will be displayed as URL in *view* mode. String has to match the following pattern: "^((?:mailto\:|(?:(?:ht|f)tps?)\://)1\S+)(?: (?:|)?(.*))?$" Example: "http://consol.de ConSol*"
	ticket-list-colspan	layout	Defines how many columns are occupied by the field in the ticket list box.	<number>	Number of columns.
	ticket-list-position	layout	Defines the position of the field in the ticket list box.	<number>;<number>	Values define *row* and *column* (row;column), numbering starts at 0;0.
	ticket-list-rowspan	layout	Defines how many rows are occupied by the field in the ticket list box.	<number>	Number of rows.
U	username	contact authentification	Indicates that this field will be used as a login name in the authentication process.	true / false	Used for CM/Track.

	Name	Annotation Type	Description	Values	Comment
V	visibility	common	Defines when the field is visible.	edit	Field will be displayed in *edit* mode.
				view	Field will be displayed in *view* mode.
				none	Field is not visible.
					If any other or no value is set the field will always be visible.
	visibility configuration	visibility	Indicates the visibility of this field in history.	on every level	Field is shown on every level of history.
				2nd level and 3rd level	Field is shown only on the 2nd and the 3rd level of history.
				only 3rd level	Field is shown only on the 3rd level of history.

9.2 Alphabetical List of Group Annotations (Version 6.8 and Older)

	Name	Annotation Type	Description	Values	Comment
C	contact history template name	ticket contact relation	Describes the contact information shown in ticket history.	<template name>	Format is specified within the template definition. Name of template is referenced here.
	contact-template-contact-ticket-page	contact-templates	Used to display short information about a contact in the ticket and contact pages.	<template name>	Format is specified within the template definition. Name of template is referenced here. If this annotation is not configured, *contact-template-default* will be used.

	Name	Annotation Type	Description	Values	Comment
	contact-template-default	contact-templates	Used to display short information about contacts.	<template name>	Format is specified within the template definition. Name of template is referenced here. If this annotation is not configured, deprecated *unit search template name* will be used.
	contact-template-dragged	contact-templates	Used to display short information about a contact when contact is dragged.	<template name>	Format is specified within the template definition. Name of template is referenced here. If this annotation is not configured, *contact-template-default* will be used.

	Name	Annotation Type	Description	Values	Comment
	contact-template-email	contact-templates	Used to display short information about a contact for auto-completion of e-mail addressee.	<template name>	Format is specified within the template definition. Name of template is referenced here. If this annotation is not configured, *contact-template-default* will be used.
	contact-template-quick-search	contact-templates	Used to display short information about a contact in the quick search result list.	<template name>	Format is specified within the template definition. Name of template is referenced here. If this annotation is not configured, *contact-template-default* will be used.
	contact-template-search	contact-templates	Used to display short information about a contact in the contact search result list.	<template name>	Format is specified within the template definition. Name of template is referenced here. If this annotation is not configured, *contact-template-default* will be used.

	Name	Annotation Type	Description	Values	Comment
	contact-template-ticket-list	contact-templates	Used to display short information about a contact in the ticket list.	<template name>	Format is specified within the template definition. Name of template is referenced here. If this annotation is not configured, *contact-template-default* will be used.
	contact-template-ticket-reference	contact-templates	Used to display short information about a contact in the ticket reference section.	<template name>	Format is specified within the template definition. Name of template is referenced here. If this annotation is not configured, *contact-template-default* will be used.
	contact-template-ticket-search	contact-templates	Used to display short information about a contact in the ticket search result list.	<template name>	Format is specified within the template definition. Name of template is referenced here. If this annotation is not configured, *contact-template-default* will be used.

	Name	Annotation Type	Description	Values	Comment
	contact-template-workspace-favourite	contact-templates	Used to display short information about a contact in the workspace and favourites sections.	<template name>	Format is specified within the template definition. Name of template is referenced here. If this annotation is not configured, *contact-template-default* will be used.
G	group-visibility	common	Defines the default visibility of a custom field group.	true / false	The annotation can be overwritten on field level.
N	no-history	performance	Indicates that all custom fields belonging to this group will not be historized.	true / false	Possible values are *true* if this annotation should be active or *false* which is the same like removing the annotation. Use this annotation if you want to prevent history for all/many fields in a group. If you only want to prevent historization for a single/some field(s), use the annotation *no-history-field* on field level.

	Name	Annotation Type	Description	Values	Comment
O	open-at-create	layout	Allows custom field groups to be visible during ticket creation even if they are hidden.	true	Remove the annotation if the group should not be visible.
R	reportable group	dwh	Indicates that all custom fields belonging to this group are reportable and should be transferred to CMRF.	true / false	A value has to be set. Annotation is active if value is set to *true*.
S	show-contact-in-ticket-list	layout	Indicates that the custom field group (contact) should be shown in the ticket list.	true	This annotation can only be assigned to groups with the annotation *unit is a contact*. Remove the annotation if the contact should not be shown in the ticket list.
	show-in-group-section	layout	Defines that a custom field group is displayed in the *Groups* section.	true	Without this annotation the group is shown in the ticket header.
U	unit is a contact	ticket contact relation	Indicates that the custom field group describes contact data.	true / false	Group is shown with contact when *true* or with ticket when *false*.

	Name	Annotation Type	Description	Values	Comment
	unit search template name ***deprecated***	indexing	Template used to display short information about found contacts.	<template name>	Format is specified within the template definition. Name of template is referenced here.

9.3 Alphabetical List of Group Annotations (Version 6.9 and Higher)

	Name	Annotation Type	Description	Values	Comment
A	auto-open-group	layout	The group will be opened initially. More than one value can be entered as comma-separated list (can be used for the *customer* annotation).	ticket:create	Group is opened initially when a new ticket is created.
				customer:create	Group is opened initially when a new customer is created.
				customer:view	Group is opened when the customer (contact or company) page is opened.
G	group-visibility	common	Defines the default visibility of a custom field group.	true / false	The annotation can be overwritten on field level.

	Name	Annotation Type	Description	Values	Comment
N	no-history	performance	Indicates that all custom fields belonging to this group will not be historized.	true / false	Possible values are *true* if this annotation should be active or *false* which is the same like removing the annotation. Use this annotation if you want to prevent history for all/many fields in a group. If you only want to prevent historization for a single/some field(s), use the annotation *no-history-field* on field level.
R	reportable group	dwh	Indicates that all custom fields belonging to this group are reportable and should be transferred to CMRF.	true / false	A value has to be set. Annotation is active if value is set to *true*.
S	show-contact-in-ticket-list		Obsolete! Use page customization! accordionTicketList.mainCustomerDescriptionVisible={true, false}	obsolete	

	Name	Annotation Type	Description	Values	Comment
	show-in-group-section	layout	Defines that a custom field group is displayed in the *Groups* section (as tab).	true / false	Without this annotation the group is shown in the non-tabbed ticket data or contact section.
U	unit is a contact ***deprecated***	ticket contact relation		true/false	Removed in version 6.9.0.

10 Appendix B - Glossary

	Term	Explanation
A	Access Rights	Permissions of an engineer to view or make changes to tickets in the Web Client. Access rights are always assigned to a group, never to single engineers/users.
	ACIM	Activity item - entry in the history section of a ticket (e.g. comment, e-mail, attachment, time booking entry).
	AD	Microsoft Active Directory - an LDAP-based directory service for Microsoft Windows domain networks.
	Additional customer	Customer (contact or company) besides the main customer, e.g. an employee of the company. For additional customers, customer roles can be assigned.
	Admin-Tool	Graphical application to configure and manage a ConSol*CM system. Uses Java Web Start.
B	BI	Business Intelligence: Methods, technologies, and architectures to transform data into useful information for business purposes.
C	CMDB	ConSol*CM Database - the working database of the CM system.
	CMRF	ConSol*CM Reporting Framework: JEE application which synchronizes data between the ConSol*CM database and the DWH.

	Term	Explanation
	CM/Office	A standard module of ConSol*CM which enables the engineer via ConSol*CM/Web Client to work with MS-Word documents pre-filled with ConSol*CM ticket or customer parameters.
	CM/Track	Consol*CM web portal: Provides customer access to the ConSol*CM system.
	Company	A data object of type company. Often this is a real company or an institution, but it can also be something else, like a machine or a ship.
	Contact	The customer who has a question or service request.
	CTI	Computer telephony integration, a description for any technology that allows interactions on a telephone and a computer to be integrated or coordinated
	Customer	General term for customer objects in ConSol*CM. A customer can be a contact or a company. Technically, a customer is a data object. The respective java class is Unit.
D	Data object	A customer, contact, or a company. Former *Unit*.
	Data object group	A group of fields where data for customers (contact or company) can be stored. Similar to custom field group for ticket data.
	Data object group field	A field where data for customers (contact or company) can be stored. Similar to custom field for ticket data.

	Term	Explanation
	DWH	Data Warehouse: ConSol*CM database used for reporting and data analysis.
E	Engineer	User who has a login to the Web Client and who has to manage the tasks defined in the tickets.
	ESB	Enterprise Service Bus: Software architecture used for communication between mutually interacting software applications in a service-oriented architecture (SOA).
	ERP system	Enterprise Resource Planning: Often used for this type of enterprise management software.
	ETL	Extract Transform Load: Extracts data from one source (this can be a database or another source), transforms it, and loads it into a database, e.g. a data warehouse.
F	FlexCDM	The *Flexible Customer Data Model*, the Customer Data Model which has been introduced in ConSol*CM in version 6.9. For each customer group, a specific customer data model can be defined.
G	GUI	Graphical User Interface
I	IMAP	Internet Message Access Protocol: Internet standard protocol to access e-mail on a remote e-mail server. Can be used as plain IMAP or as secure IMAP (IMAPs). In the latter case the proper certificates are required.
J	Java EE	Java Enterprise Edition

	Term	Explanation
	JMS	Java Message Service: Java EE component used to send messages between JMS clients.
K	KPI	Key Performance Indicator - parameter used for performance measurement for companies, projects etc.
L	LDAP	Lightweight Directory Access Protocol: Application protocol to access and maintain directory information over an IP network.
M	Mailbox	Destination to which e-mail messages are delivered. Mailboxes are managed on a mail server. ConSol*CM can access one or more separate mailboxes to retrieve e-mails.
	Main customer	The customer who is the main customer of a ticket. Starting with ConSol*CM version 6.9, this can be either a contact or a company.
	Mule	An open source Java-based Enterprise Service Bus (ESB).
P	PCDS	Page Customization Definition Section
	Pentaho	Pentaho™ is a business intelligence (BI) suite which is available as open source version and as enterprise edition.
	POP	Post Office Protocol: Internet standard protocol to retrieve e-mails from a remote server via TCP/IP. Can be used as plain POP or as secure POP (POPs). In the latter case the proper certificates are required.

	Term	Explanation
	Portal	CM/Track: Provides customer access to ConSol*CM.
	Process Designer	Graphical application to model and program ConSol*CM workflows. Uses Java Web Start.
Q	Queue	Comprises tickets from the same domain and makes sure that all tickets of this domain are treated in the same way. A queue always has one workflow. Access rights and other parameters are defined based on queues.
R	RDBMS	Relational Database Management System: E.g. Oracle ® , MS SQL Server ® , MySQL.
	REST	Representational State Transfer: Method to transfer data via a network, based on HTTP.
	Role	Defines the access permissions and views of an engineer.
S	Script	Program written for a special run-time environment that can interpret and automate the execution of tasks. In ConSol*CM, scripts are stored in the Admin-Tool and are stored as scripts for activities in workflows.
	SMTP	Simple Message Transfer Protocol: Standard protocol to send e-mails.

	Term	Explanation
T	TAPI	Telephony Application Programming Interface, a Microsoft Windows API, which provides computer telephony integration and enables PCs running Microsoft Windows to use telephone services
	Template	Pre-formatted example concerning layout, text, and/or data, e.g. for e-mails or CM/Office.
	Ticket	Incident, service case, or other request of an internal or external customer. A ticket is the object which runs through the process (defined by the workflow).
V	View	A selection of tickets based on scopes from one or from different workflows, assigned to a role, and visible in the ticket list of the ConSol*CM/Web Client.
W	Workflow	Models a process that should be managed using ConSol*CM step by step.

11 Appendix C - System Properties

The lists provide explanation for all available ConSol*CM system properties. You can define properties in the Admin-Tool, in the *Configuration* section.

11.1 System Properties Ordered by Module

Module	Property	Explanation
cmas-app-admin-tool	admin.tool.session.check.interval	*Description:* Admin-Tool inactive (ended) sessions check time interval (in seconds) *Type:* Integer *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* 30 *Since:* 6.7.5
cmas-app-admin-tool	autocomplete.enabled ***only version 6.9 and higher***	*Description:* If the flag is missing or its value is *false*, then the *Auto complete address* tab is hidden in AT. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* true *Since:* 6.9.2.0
cmas-core-cache	cache-cluster-name	*Description:* JBoss cache cluster name *Type:* String *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* 635a6de1-629a-4129-8299-2d98633310f0 *Since:* 6.4.0
cmas-core-cache	eviction.event.queue.size	*Description:* *Type:* Integer *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* 200000 *Since:* 6.4.0

Module	Property	Explanation
cmas-core-cache	eviction.max.nodes	*Description:* *Type:* Integer *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* 100000 *Since:* 6.4.0
cmas-core-cache	eviction.wakeup.interval	*Description:* *Type:* Integer *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* 3000 *Since:* 6.4.0
cmas-core-index-common	big.task.minimum.size	*Description:* How many parts task at least should have to be handled by indexer with low priority. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 15 (default) *Since:* 6.8.3
cmas-core-index-common	disable.admin.task.auto.commit	*Description:* All tasks created for index update will be automatically executed right after creation. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.6.1
cmas-core-index-common	index.attachment	*Description:* Describes if content of attachments is indexed. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* true *Since:* 6.4.3

Module	Property	Explanation
cmas-core-index-common	index.history	*Description:* Describes if unit and ticket history are indexed. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.1.0
cmas-core-index-common	index.status	*Description:* Status of the indexer, possible values RED, YELLOW, GREEN, will be displayed in the Admin-Tool. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* GREEN *Since:* 6.6.1
cmas-core-index-common	index.task.worker.threads	*Description:* How many threads will be used to execute batch index tasks (synchronization, administrative, and repair tasks). *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 1 (default) (we recommend to use a value not larger than 2) *Since:* 6.6.14, 6.7.3
cmas-core-index-common	index.version.current	*Description:* Holds information about current (possibly old) index version. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 1 (default) *Since:* 6.7.0

Module	Property	Explanation
cmas-core-index-common	index.version.newest	*Description:* Holds information about which index version is considered newest. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 1 (default) *Since:* 6.7.0
cmas-core-index-common	indexed.assets.per.thread.in.memory	*Description:* How many assets should be loaded into memory at once during indexing per one thread. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 200 (default) *Since:* 6.8.0
cmas-core-index-common	indexed.engineers.per.thread.in.memory	*Description:* How many engineers should be loaded into memory at once during indexing per one thread. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 300 (default) *Since:* 6.6.14, 6.7.3
cmas-core-index-common	indexed.tickets.per.thread.in.memory	*Description:* How many tickets should be loaded into memory at once during indexing per one thread. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 100 (default) *Since:* 6.6.14, 6.7.3

Module	Property	Explanation
cmas-core-index-common	indexed.units.per.thread.in.memory	*Description:* How many units should be loaded into memory at once during indexing per one thread. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 200 (default) *Since:* 6.6.14, 6.7.3
cmas-core-index-common	synchronize.master.address	*Description:* Value of *-Dcmas.http.host.port* informing how to connect to indexing master server. Default null. Since 6.6.17 this value is configurable in setup to designate initial indexing master server. Please note that changing this value is only allowed when all cluster nodes index changes receivers are stopped. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 127.0.0.1:80 *Since:* 6.6.0
cmas-core-index-common	synchronize.master.security.token	*Description:* The password for accessing the index snapshot via URL, e.g. for index synchronizaton or for back-ups. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* token *Since:* 6.6.0

Module	Property	Explanation
cmas-core-index-common	synchronize.master.security.user	*Description:* The user name for accessing the index snapshot via URL, e.g. for index synchronizaton or for back-ups. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* user *Since:* 6.6.0
cmas-core-index-common	synchronize.master.timeout.minutes	*Description:* How much time master server may constantly fail until new master gets elected with index fix procedure. Default 5. Since 6.6.17 this value is configurable in setup where zero means that master server will never change (failover mechanism is off). *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 5 *Since:* 6.6.0
cmas-core-index-common	synchronize.megabits.per.second	*Description:* How much bandwidth can master server consume to transfer index changes to all slave servers. Default 85. Please do not use all available bandwidth to transfer index changes between hosts. This will most probably partition cluster as some subsystems will not be able to communicate. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 85 *Since:* 6.6.0

Module	Property	Explanation
cmas-core-index-common	synchronize.sleep.millis	*Description:* How often each slave server polls master server for index changes. Default 1000. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 1000 *Since:* 6.6.0
cmas-core-security	admin.email	*Description:* The e-mail address of the ConSol*CM administrator. The value which you have entered during system set-up is used initially. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* maz@consol.de *Since:* 6.0
cmas-core-security	admin.login	*Description:* The name of the ConSol*CM administrator. The value which you have entered during system set-up is used initially. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* admin *Since:* 6.0
cmas-core-security	authentication.method	*Description:* User authentication method (internal CM database or LDAP authentication). Allowed values are *LDAP* or *DATABASE*. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* DATABASE *Since:* 6.0

Module	Property	Explanation
cmas-core-security	contact.authentication.method ***only version 6.9 and higher***	*Description:* Indicates contact authentication method, where possible values are *DATABASE* or *LDAP* or *LDAP,DATABASE* or *DATABASE,LDAP.* *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Since:* 6.9.3.0
cmas-core-security	contact.inherit.permissions.only.to.own.customer.group ***only version 6.9 and higher***	*Description:* Indicates whether authenticated contact inherits all customer group permissions from representing engineer (false) or only permission to own customer group (true). *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Since:* 6.9.2.3
cmas-core-security	kerberos.v5.enabled	*Description:* Flag which indicates whether SSO via Kerberos is enabled. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false (default if Kerberos has not been enabled during system set-up) *Since:* 6.2.0
cmas-core-security	kerberos.v5.username.regex	*Description:* Regular expression used for mapping Kerberos principal to CM user login. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* (.*)@.* *Since:* 6.2.0

Module	Property	Explanation
cmas-core-security	ldap.authentication	*Description:* Authentication method used when using LDAP authentication. *Type:* String *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* simple *Since:* 6.0
cmas-core-security	ldap.basedn	*Description:* Base DN used for looking up LDAP user accounts when using LDAP authentication. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* OU=accounts,DC=consol,DC=de *Since:* 6.0
cmas-core-security	ldap.contact.name.basedn ***only version 6.9 and higher***	*Description:* Base path to search for contact DN by LDAP ID (e.g. ou=accounts,dc=consol,dc=de). *Type:* String *Restart required:* No *System:* No *Optional:* Yes *Since:* 6.9.3.0
cmas-core-security	ldap.contact.name.password ***only version 6.9 and higher***	*Description:* Password to lookup contact DN by LDAP ID. If not set, anonymous account is used. *Type:* String *Restart required:* No *System:* No *Optional:* Yes *Since:* 6.9.3.0

Module	Property	Explanation
cmas-core-security	ldap.contact.name.providerurl ***only version 6.9 and higher***	*Description:* Address of the LDAP server (ldap[s]://host:port). *Type:* String *Restart required:* No *System:* No *Optional:* Yes *Since:* 6.9.3.0
cmas-core-security	ldap.contact.name.searchattr ***only version 6.9 and higher***	*Description:* Attribute to search for contact DN by LDAP ID (e.g. uid). *Type:* String *Restart required:* No *System:* No *Optional:* Yes *Since:* 6.9.3.0
cmas-core-security	ldap.contact.name.userdn ***only version 6.9 and higher***	*Description:* User DN to lookup contact DN by LDAP ID. If not set, anonymous account is used. *Type:* String *Restart required:* No *System:* No *Optional:* Yes *Since:* 6.9.3.0
cmas-core-security	ldap.initialcontextfactory	*Description:* Class name for initial context factory of LDAP implementation when using LDAP authentication. If it is not set, com.sun.jndi.ldap.LdapCtxFactory is being used as a value. *Type:* String *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* com.sun.jndi.ldap.LdapCtxFactory *Since:* 6.0

Module	Property	Explanation
cmas-core-security	ldap.password	*Description:* Password for connecting to LDAP to lookup users (when using LDAP authentication). Only needed if lookup cannot be done anonymously. *Type:* Password *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.1.2
cmas-core-security	ldap.providerurl	*Description:* LDAP provider (when using LDAP authentication). *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* ldap://ldap.consol.de:389 *Since:* 6.0
cmas-core-security	ldap.searchattr	*Description:* Search attribute for looking up LDAP entry connected to CM6 login. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* uid *Since:* 6.0
cmas-core-security	ldap.userdn	*Description:* LDAP user for connecting to LDAP to lookup users (when using LDAP authentication). Only needed if lookup cannot be done anonymously. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.1.2

Module	Property	Explanation
cmas-core-server	attachment.allowed.types	*Description:* Comma-separated list of allowed filename extensions (if no value defined, all file extensions are allowed). *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* txt,zip,doc *Since:* 6.5.0
cmas-core-server	attachment.max.size	*Description:* Maximum attachment size in MB *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 100 *Since:* 6.4.0
cmas-core-server	config.data.version	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 11 *Since:* 6.0
cmas-core-server	defaultCommentClassName	*Description:* Default text class name for comments *Type:* String *Restart required:* No *System:* No *Optional:* Yes *Example value:* *Since:* 6.3.0
cmas-core-server	defaultIncommingMailClassName	*Description:* Default text class name for incoming mails *Type:* String *Restart required:* No *System:* No *Optional:* Yes *Example value:* *Since:* 6.3.0

Module	Property	Explanation
cmas-core-server	defaultOutgoingMailClassName	*Description:* Default text class name for outgoing mails *Type:* String *Restart required:* No *System:* No *Optional:* Yes *Example value:* *Since:* 6.3.0
cmas-core-server	fetchSize.strategy	*Description:* Strategy selected to set fetch size on jdbc result sets. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* FetchSizePageBasedStrategy, FetchSizeThresholdStrategy, FetchSizeFixedStrategy *Since:* 6.8.4.1
cmas-core-server	fetchSize.strategy.FetchSizeFixedStrategy.value	*Description:* Sets fetch size value if selected strategy to set fetch size is *FetchSizeFixedStrategy*. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 150 *Since:* 6.8.4.1
cmas-core-server	fetchSize.strategy.FetchSizePageBasedStrategy.limit	*Description:* Sets max fetch size value if selected strategy to set fetch size is *FetchSizePageBasedStrategy*. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 10000 *Since:* 6.8.4.1

Module	Property	Explanation
cmas-core-server	fetchSize.strategy.FetchSizeThresholdStrategy.value	*Description:* Sets fetch size threshold border values if selected strategy to set fetch size is *FetchSizeThresholdStrategy*. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 150,300,600,1000 *Since:* 6.8.4.1
cmas-core-server	last.config.change	*Description:* Random UUID created during last change in config *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* 2573c7b7-2bf5-47ff-b5a2-bad31951a266 *Since:* 6.1.0, 6.2.1
cmas-core-server	ldap.certificate.basedn	*Description:* Base DN for certificates location in LDAP tree. If not provided, ldap.basedn is taken. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* OU=accounts,DC=consol,DC=de *Since:* 6.8.4
cmas-core-server	ldap.certificate.content.attribute	*Description:* LDAP attribute name used where certificate data is stored in LDAP tree. Default value is: usercertificate. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* usercertificate *Since:* 6.8.4

Module	Property	Explanation
cmas-core-server	ldap.certificate.password	*Description:* LDAP Certificates manager password. If not set, ldap.password is taken. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.8.4
cmas-core-server	ldap.certificate.providerurl	*Description:* LDAP Certificates provider URL. If not set, ldap.providerurl is taken. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* ldap://ldap.consol.de:389 *Since:* 6.8.4
cmas-core-server	ldap.certificate.searchattr	*Description:* LDAP attribute name used to search for certificate in LDAP tree. Default value is: mail. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* mail *Since:* 6.8.4
cmas-core-server	ldap.certificate.userdn	*Description:* LDAP Certificates manager DN. If not set, ldap.userdn is taken. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.8.4

Module	Property	Explanation
cmas-core-server	mail.notification.engineerChange	*Description:* Flag if notification mail should be sent when engineer of ticket is changed. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* true *Since:* 6.1.0
cmas-core-server	mail.notification.sender	*Description: From* address for notification mails when engineer of ticket is changed. If not set, *cmas-core-security admin.email* is used instead. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* cm6notification@cm6installation *Since:* 6.6.3
cmas-core-server	mail.smtp.email	*Description:* SMTP mail URL for outgoing mails *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* smtp://mail.consol.de:25 *Since:* 6.0
cmas-core-server	mail.smtp.envelopesender	*Description:* Mail address used as sender in SMTP envelope. If not set, the *From:* address of the mail is used. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* mysender@mydomain.com *Since:* 6.5.7

Module	Property	Explanation
cmas-core-server	max.licences.perUser	*Description:* Sets max licenses single user can use (e.g logging in from different browsers). By default this value is not restricted. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 10 *Since:* 6.8.4.5
cmas-core-server	monitoring.engineer.login ***only version 6.9 and higher***	*Description:* Login of monitoring engineer *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* bartek *Since:* 6.9.3.0
cmas-core-server	monitoring.unit.login ***only version 6.9 and higher***	*Description:* Login of monitoring unit *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* bartek *Since:* 6.9.3.0
cmas-core-server	server.session.archive.reaper.interval	*Description:* Server archived sessions' reaper interval (in seconds) *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 60 *Since:* 6.7.1

Module	Property	Explanation
cmas-core-server	server.session.archive.timeout	*Description:* Server sessions archive validity timeout (in days). After this time session info is removed from DB. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 31 *Since:* 6.7.1
cmas-core-server	server.session.reaper.interval	*Description:* Server inactive (ended) sessions' reaper interval (in seconds) *Type:* Integer *Restart required:* Only Session Service *System:* Yes *Optional:* No *Example value:* 60 *Since:* 6.6.1, 6.7.1
cmas-core-server	server.session.timeout	*Description:* Server session timeout (in seconds) for connected clients. Each client can overwrite this timeout with custom value using its ID (ADMIN_TOOL, WEB_CLIENT, WORKFLOW_EDITOR, TRACK (before 6.8 please use PORTER), ETL, REST) appended to property name, e.g. server.session.timeout.ADMIN_TOOL *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 1800 *Since:* 6.6.1, 6.7.1

Module	Property	Explanation
cmas-core-server	tickets.delete.size	*Description:* Property that defines a number of tickets deleted per transaction. By default it is set to 10. *Type:* Integer *Restart required:* Only Session Service *System:* Yes *Optional:* No *Example value:* 10 *Since:* 6.8.1
cmas-core-server	ticket.delete.timeout	*Description:* Transaction timeout (in seconds) for deleting tickets *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 60 *Since:* 6.1.3
cmas-core-server	unit.replace.batchSize	*Description:* Describes number of objects to be processed in unit replace action. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 5 *Since:* 6.8.2
cmas-core-server	unit.replace.timeout	*Description:* Transaction timeout (seconds) of unit replacement action step. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 120 *Since:* 6.8.2

Module	Property	Explanation
cmas-core-server	unused.content.remover.cluster.node.id ***only version 6.9 and higher***	*Description:* Value of a cmas.clusternode.id designating node which will remove unused ticket attachments and unit content entries. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 1 (assuming cluster node started with -Dcmas.clusternode.id=1 parameter) *Since:* 6.9.0.0
cmas-core-server	unused.content.remover.enabled ***only version 6.9 and higher***	*Description:* Flag whether unused ticket attachments and unit content entries removal should take place. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* true *Since:* 6.9.0.0
cmas-core-server	unused.content.remover.polling.minutes ***only version 6.9 and higher***	*Description:* How often unused ticket attachments and unit content entries should be checked for removal. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 15 *Since:* 6.9.0.0

Module	Property	Explanation
cmas-core-server	unused.content.remover.ttl.minutes ***only version 6.9 and higher***	*Description:* Minimum interval after which unused ticket attachments and unit content entries can be removed. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 1440 *Since:* 6.9.0.0
cmas-core-shared	cluster.mode	*Description:* Flag if CMAS is running in cluster. *Type:* Boolean *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* false *Since:* 6.1.0
cmas-core-shared	data.directory	*Description:* Directory for CMAS data (e.g. index) *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* C:\Users\user\cmas *Since:* 6.0
cmas-dwh-server	autocommit.cf.changes	*Description:* *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.7.0

Module	Property	Explanation
cmas-dwh-server	batch-commit-interval	*Description:* Number of objects in a JMS message. Higher value means better transfer performance and bigger memory usage. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 100 *Since:* 6.0.0
cmas-dwh-server	dwh.mode	*Description:* Current mode of DWH data transfer. Possible values are OFF, ADMIN, LIVE *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* OFF *Since:* 6.0.1
cmas-dwh-server	ignore-queues	*Description:* By adding a comma separated list of queue names it is configured that tickets of these queues are not transferred to the DWH. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* QueueName1,QueueName2,QueueName3 *Since:* 6.6.19 *Removed in:* 6.8.1

Module	Property	Explanation
cmas-dwh-server	is.cmrf.alive	*Description:* As a starting point time of sending last message to CMRF should be used. If response from CMRF is not received after value (in seconds) it should create a DWH operation status with error message that CMRF is down. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 1200 *Since:* 6.7.0
cmas-dwh-server	java.naming.factory.initial	*Description:* Factory class for DWH context factory. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* org.jnp.interfaces.NamingContextFactory *Since:* 6.0.1
cmas-dwh-server	java.naming.factory.url.pkgs	*Description:* *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* org.jboss.naming:org.jnp.interfaces *Since:* 6.0.1
cmas-dwh-server	java.naming.provider.url	*Description:* URL of naming provider *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* localhost *Since:* 6.0.1

Module	Property	Explanation
cmas-dwh-server	notification.error.description	*Description:* Text for error mails from DWH *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* Error occurred *Since:* 6.0.1
cmas-dwh-server	notification.error.from	*Description: From* address for error mails from DWH *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.0.1
cmas-dwh-server	notification.error.subject	*Description:* Subject for error mails from DWH *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* Error occurred *Since:* 6.0.1
cmas-dwh-server	notification.error.to	*Description: To* address for error mails from DWH *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* maz@consol.de *Since:* 6.0.1
cmas-dwh-server	notification.finished_successfully.description	*Description:* Text for mails from DWH when transfer finished successfully. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* Transfer finished successfully *Since:* 6.0.1

Module	Property	Explanation
cmas-dwh-server	notification.finished_successfully.from	*Description: From* address for mails from DWH when transfer finished successfully. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.0.1
cmas-dwh-server	notification.finished_successfully.subject	*Description:* Subject for mails from DWH when transfer finished successfully. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* Transfer finished successfully *Since:* 6.0.1
cmas-dwh-server	notification.finished_successfully.to	*Description: To* address for mails from DWH when transfer finished successfully. *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* maz@consol.de *Since:* 6.0.1
cmas-dwh-server	notification.finished_unsuccessfully.description	*Description:* Text for mails from DWH when transfer finished unsuccessfully. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* Transfer finished unsuccessfully *Since:* 6.0.1

Module	Property	Explanation
cmas-dwh-server	notification.finished_unsuccessfully.from	*Description: From* address for mails from DWH when transfer finished unsuccessfully. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.0.1
cmas-dwh-server	notification.finished_unsuccessfully.subject	*Description:* Subject for mails from DWH when transfer finished unsuccessfully. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* Transfer finished unsuccessfully *Since:* 6.0.1
cmas-dwh-server	notification.finished_unsuccessfully.to	*Description: To* address for mails from DWH when transfer finished unsuccessfully. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* maz@consol.de *Since:* 6.0.1
cmas-dwh-server	notification.host	*Description:* Mail (SMTP) server hostname for sending DWH mails *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* mail.consol.de *Since:* 6.1.0

Module	Property	Explanation
cmas-dwh-server	notification.password	*Description:* Password for sending DWH mails (optional) *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.1.0
cmas-dwh-server	notification.port	*Description:* SMTP port for sending DWH mails *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 25 *Since:* 6.1.0
cmas-dwh-server	notification.protocol	*Description:* The protocol used for sending emails from DWH. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* pop3\
cmas-dwh-server	notification.username	*Description:* (SMTP) User name for sending DWH mails *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* maz *Since:* 6.1.0
cmas-dwh-server	skip-ticket	*Description:* Tickets are not transferred during transfer/update. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.6.19 *Removed in:* 6.8.1

Module	Property	Explanation
cmas-dwh-server	skip-ticket-history	*Description:* History of ticket is not transferred during transfer/update. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.6.19 *Removed in:* 6.8.1
cmas-dwh-server	skip-unit	*Description:* Units are not transferred during transfer/update. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.6.19 *Removed in:* 6.8.1
cmas-dwh-server	skip-unit-history	*Description:* History of unit is not transferred during transfer/update. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.6.19 *Removed in:* 6.8.1
cmas-dwh-server	split.history	*Description:* Changes the SQL that fetches the history for the tickets during DWH transfer not to all tickets at once but only for one ticket per SQL. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* false *Since:* 6.8.0

Module	Property	Explanation
cmas-dwh-server	unit.transfer.order	*Description:* Define in which order unit custom field groups should be transferred to the DWH. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value: company;customer* *Since:* 6.6.19 *Removed in:* 6.8.1
cmas-esb-core	esb.directory	*Description:* Directory used by ESB (Mule) *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* C:\Users\user\cmas\mule *Since:* 6.0
cmas-esb-mail	mail.attachments.validation.info.sender	*Description:* Sets *From* header of attachments type error notification mail. As a default the e-mail address of the administrator which you have entered during system set-up is used. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* admin@consolcm.com *Since:* 6.7.5

Module	Property	Explanation
cmas-esb-mail	mail.attachments.validation.info.subject	*Description:* Sets subject of attachments type error notification mail. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* Mail was not processed because its attachments were rejected!!! *Since:* 6.7.5
cmas-esb-mail	mail.callname.pattern	*Description:* Regular expression for subject of incoming mails. Available as TICKET_NAME_PATTERN_FORMAT in incoming mail scripts. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* .*?Ticket\s+\((\S+)\).* *Since:* 6.0
cmas-esb-mail	mail.cluster.node.id	*Description:* Only the node whose mail.cluster.node.id equals cmas.clusternode.id will start the Mule ESB mail services. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* unspecified *Since:* 6.6.5
cmas-esb-mail	mail.db.archive	*Description:* If property is set to *true*, incoming e-mails are archived in the database. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* false (default) *Since:* 6.8.5.5

Module	Property	Explanation
cmas-esb-mail	mail.delete.read	*Description:* Determines whether CM deletes messages fetched via IMAP(S). Setting value to *true* will cause deletion of messages after fetching. Default is to not delete messages fetched via IMAP(S). Note: Messages fetched via POP3(S) will always be deleted. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* true *Since:* 6.7.3
cmas-esb-mail	mail.encryption	*Description:* If property is set to *true*, the encrypt check box in the Ticket E-Mail Editor is checked by default. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* true (default = false) *Since:* 6.8.4.0

Module	Property	Explanation
cmas-esb-mail	mail.incoming.uri	*Description:* URL for incoming mails *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* pop3://cm-incoming-user:password@localhost:10110 *Since:* 6.0 This value should not be edited here using the system properties pop-up window, but the mailboxes should be configured using the file card *E-mail* in the Admin-Tool (see *ConSol*CM Administrator Manual* section *File Card E-mail*). Using this standard feature all entries are controlled - i.e. for each mailbox which is added, CM establishes a test connection during mailbox set-up. That way it is not possible to enter wrong values.
cmas-esb-mail	mail.max.restarts	*Description:* Maximum number of mail service restarts before giving up *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 3 *Since:* 6.0

Module	Property	Explanation
cmas-esb-mail	mail.mime.strict	*Description:* If set to *false*, mail addresses are not parsed for strict MIME compliance. Default is *true*, which means check for strict MIME compliance. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.6.17, 6.7.3
cmas-esb-mail	mail.mule.service	*Description: From* address for mails sent by Mule service *Type:* EMail *Restart required:* No *System:* Yes *Optional:* No *Example value:* maz@consol.de *Since:* 6.0
cmas-esb-mail	mail.polling.interval	*Description:* Mail polling interval in ms *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 60000 *Since:* 6.0
cmas-esb-mail	mail.process.error	*Description: To* address for error mails from Mule. As a default the e-mail address of the administrator which you have entered during system set-up is used. *Type:* EMail *Restart required:* No *System:* Yes *Optional:* No *Example value:* maz@consol.de *Since:* 6.0

Module	Property	Explanation
cmas-esb-mail	mail.process.retry.attempts	*Description:* Number of retries when processing mail *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 3 *Since:* 6.0.2
cmas-esb-mail	mail.process.timeout	*Description:* Mail processing timeout in seconds *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 60 *Since:* 6.1.3
cmas-esb-mail	mail.redelivery.retry.count	*Description:* Indicates the number of retries of re-delivering an e-mail from the CM system. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 3 *Since:* 6.1.0
cmas-setup-hibernate	hibernate.dialect	*Description:* The dialect used by hibernate. Usually set during initial setup (depending on the database system). *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* org.hibernate.dialect.MySQL5InnoDBDialect *Since:* 6.0

Module	Property	Explanation
cmas-setup-manager	initialized	*Description:* Flag if CMAS is initialized. If this value is missing or not *true*, setup will be performed. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* true *Since:* 6.0 Be careful with using this property!!! When you set the value to *false*, the ConSol*CM server will perform the system set-up at the next start, i.e. all data of the existing system is lost, including system properties!!!
cmas-setup-scene	scene	*Description:* Scene file which was imported during setup (can be empty). *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* vfszip:/P:/dist/target/jboss/server/cmas/deploy/cm-dist-6.5.1-SNAPSHOT.ear/APP-INF/lib/dist-scene-6.5.1-SNAPSHOT.jar/META-INF/cmas/scenes/helpdesk-sales_scene.jar/ *Since:* 6.0

Module	Property	Explanation
cmas-workflow-engine	jobExecutor.adminMail	*Description:* Mail which will get notified about job execution problems (when retry counter is exceeded). *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* admin@consol.de *Since:* 6.8.0
cmas-workflow-engine	jobExecutor.idleInterval.seconds	*Description:* Determines how often job executor thread will look for new jobs to execute. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 5 (default) *Since:* 6.8.0
cmas-workflow-engine	jobExecutor.jobMaxRetries	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 5 (default) *Since:* 6.8.0
cmas-workflow-engine	jobExecutor.jobMaxRetriesReachedSubject	*Description:* (rev.54593) *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* Job max retries reached. Job was removed!!! (default) *Since:* 6.8.0

Module	Property	Explanation
cmas-workflow-engine	jobExecutor.lockTimeout.seconds	*Description:* How long the job can be locked (marked for execution) by job executor. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 360 (default) *Since:* 6.8.0
cmas-workflow-engine	jobExecutor.lockingLimit	*Description:* Number of job locked at once (marked for execution) by job executor thread *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 10 (default) *Since:* 6.8.0
cmas-workflow-engine	jobExecutor.mailFrom	*Description:* Mail which will be set as *From* header during admin notifications. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* jobexecutor@consol.de *Since:* 6.8.0
cmas-workflow-engine	jobExecutor.maxInactivityInterval.minutes	*Description:* Number of minutes of allowed job executor inactivity (e.g. when it is blocked by long timer execution). After this time executors threads are restarted. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes. Default value is set to 30 minutes. *Example value:* 15 (default) *Since:* 6.9.2.0

Module	Property	Explanation
cmas-workflow-engine	jobExecutor.threads	*Description:* Number of job execution threads *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 1 (default) *Since:* 6.8.0
cmas-workflow-engine	jobExecutor.timerRetryInterval.seconds	*Description:* Determines how long job executor thread will wait after job execution error. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 10 (default) *Since:* 6.8.0
cmas-workflow-engine	jobExecutor.txTimeout.seconds	*Description:* Transaction timeout used for job execution *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 60 (default) *Since:* 6.8.0
cmweb-server-adapter	checkUserOnlineIntervalInSeconds	*Description:* The interval in seconds to check which users are online (default 180sec = 3min). *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 180 *Since:* 6.0

Module	Property	Explanation
cmweb-server-adapter	cmoffice.enabled	*Description:* Flag if CM/Office is enabled. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.4.0
cmweb-server-adapter	commentRequiredForTicketCreation	*Description:* Flag if comment is a required field for ticket creation. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* true (default) *Since:* 6.2.0
cmweb-server-adapter	customizationVersion	*Description:* *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* cd58453e-f3cc-4538-8030-d15e8796a4a7 *Since:* 6.5.0

Module	Property	Explanation
cmweb-server-adapter	data.optimization	*Description:* Defines optimization to be applied on response data. So far, the following values are supported (for setting more than one value, separate values by '|'): MINIFICATION and COMPRESSION. MINIFICATION minifies HTML data by e.g. stripping whitespaces and comments. COMPRESSION applies gzip compression to HTTP response. (Note: If you are running in cluster mode and want to test different configurations in parallel, you can set different values for each cluster node by specifying property data.optimization.*nodeId* to override default property.) *Type:* String *Restart required:* COMPRESSION can be switched on/off without restart, MINIFICATION requires restart. *System:* Yes *Optional:* Yes *Example value:* MINIFICATION|COMPRESSION
cmweb-server-adapter	defaultContentEntryClassName	*Description:* Default text class for new acims *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* default_class *Since:* 6.3.0

Module	Property	Explanation
cmweb-server-adapter	defaultNumberOfCustomFieldsColumns	*Description:* Default number of columns for custom fields *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 3 *Since:* 6.2.0
cmweb-server-adapter	favoritesSizeLimit	*Description:* Maximum number of items in favorites list *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 10 *Since:* 6.0
cmweb-server-adapter	globalSearchResultSizeLimit	*Description:* Maximum number of items in global (Q&E) search result *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 10 *Since:* 6.0
cmweb-server-adapter	helpFilePath	*Description:* URL for online help. If not empty, *Help* button is displayed in Web Client. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* http://www.consol.de *Since:* 6.2.1

Module	Property	Explanation
cmweb-server-adapter	hideTicketSubject	*Description:* If set to *true*, ticket subject is hidden. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.2.1
cmweb-server-adapter	mail.from	*Description:* Use this address if set instead of engineer e-mail address during mail conversation. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.1.2

Module	Property	Explanation
cmweb-server-adapter	mail.reply.to	*Description:* When set, Web Client will display reply-to field on mail send, prefilled with this value. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.0.1 Please see also *ConSol *CM Administrator Manual* section *Queue Administration* . When you set the REPLY TO address in the outgoing e-mail script, the *mail.reply.to* system property must not be set (because it would overwrite the configured value)! That means when you use one outgoing e-mail script for a queue you have to define outgoing e-mail scripts for all queues because the *mail.reply.to* property can no longer be used.
cmweb-server-adapter	mailTemplateAboveQuotedText	*Description:* Indicates behavior of mail template in the Ticket E-Mail Editor when another mail is quoted, i.e. forwarded or replied to. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.2.4

Module	Property	Explanation
cmweb-server-adapter	maxSizePerPagemapInMegaBytes	*Description:* Maximum size (in MB) for each Wicket pagemap *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 15 *Since:* 6.3.5
cmweb-server-adapter	pagemapLockDurationInSeconds	*Description:* Number of seconds to pass before pagemap is considered to be locked for too long. *Type:* Integer *Restart required:* Yes *System:* Yes *Optional:* Yes *Example value:* 60 *Since:* 6.7.3
cmweb-server-adapter	postActivityExecutionScriptName	*Description:* Defines the name for the script which should be executed after every workflow activity (see *ConSol*CM Administrator Manual* section *Admin-Tool Scripts - Default Workflow Activity Script*). If no script should be executed, leave the value empty. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* postActivityExecutionHandler *Since:* 6.2.0
cmweb-server-adapter	queuesExcludedFromGS	*Description:* Comma-separated list of queue names which are excluded from global search. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.0

Module	Property	Explanation
cmweb-server-adapter	rememberMeLifetimeInMinutes	*Description:* Lifetime for *remember me* in minutes *Type:* Integer *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* 1440 *Since:* 6.0
cmweb-server-adapter	request.scope.transaction	*Description:* It allows to disable request scope transaction. By default one transaction is used per request. Setting this property to *false* there will cause one transaction per service method invocation. *Type:* Boolean *Restart required:* Yes *System:* Yes *Optional:* Yes *Example value:* true *Since:* 6.8.1
cmweb-server-adapter	searchPageSize	*Description:* Default page size for search results *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 20 *Since:* 6.0
cmweb-server-adapter	searchPageSizeOptions	*Description:* Options for page size for search results *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* 10|20|30|40|50|75|100 *Since:* 6.0

Module	Property	Explanation
cmweb-server-adapter	serverPoolingInterval	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 5 *Since:* 6.1.0
cmweb-server-adapter	supportEmail	*Description:* *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.0
cmweb-server-adapter	themeOverlay	*Description:* Name of used theme overlay *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* kyoEUR *Since:* 6.0
cmweb-server-adapter	ticketListRefreshIntervalInSeconds	*Description:* Refresh interval for ticket list (in seconds) *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 180 *Since:* 6.0
cmweb-server-adapter	ticketListSizeLimit	*Description:* Maximum number of tickets in ticket list *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 100 *Since:* 6.0

Module	Property	Explanation
cmweb-server-adapter	unitIndexSearchResultSizeLimit	*Description:* Maximum number of units in unit search result (e.g. when searching for contact) *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 5 *Since:* 6.0
cmweb-server-adapter	urlLogoutPath	*Description:* URL which is used when user logs out. (If no value is set, logout leads to login-mask.) *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* http://intranet.consol.de *Since:* 6.3.1
cmweb-server-adapter	webSessionTimeoutInMinutes	*Description:* Session timeout in minutes *Type:* Integer *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* 180 *Removed in:* 6.7.1 *Replaced by:* server.session.timeout
cmweb-server-adapter	wicketAjaxRequestHeaderFilterEnabled	*Description:* This enables filter for Wicket AJAX requests, coming from stale pages with Wicket 1.4 scripting (CM6 pre-6.8.0), after update to CM6 post-6.8.0. *Type:* Boolean *Restart required:* Yes *System:* Yes *Optional:* Yes *Example value:* false *Since:* 6.8.1

Module	Property	Explanation
cmas-workflow-jbpm	fetchLock.interval	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 5000 *Removed in:* 6.8.0
cmas-workflow-jbpm	fetchLock.timeout	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 15000 *Removed in:* 6.8.0
cmas-workflow-jbpm	jobExecutor.idleInterval	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 45000 *Removed in:* 6.8.0 *Replaced by:* jobExecutor.idleInterval.seconds
cmas-workflow-jbpm	jobExecutor.jobExecuteRetryNumber	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 5 *Removed in:* 6.8.0 *Replaced by:* jobExecutor.jobMaxRetries
cmas-workflow-jbpm	jobExecutor.timerRetryInterval	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 10000 *Removed in:* 6.8.0 *Replaced by:* jobExecutor.timerRetryInterval.seconds

Module	Property	Explanation
cmas-workflow-jbpm	mail.sender.address	*Description: From* address for mails from the workflow engine *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* maz@consol.de *Removed in:* 6.8.0 *Replaced by:* jobExecutor.mailFrom
cmas-workflow-jbpm	outdated.lock.age	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 60000 *Removed in:* 6.8.0 *Replaced by:* jobExecutor.lockTimeout.seconds
cmas-workflow-jbpm	refreshTimeInCaseOfConcurrentRememberMeRequests	*Description:* It sets the refresh time (in seconds) after which page will be reloaded in case of concurrent *remember me* requests. This feature prevents one user from occupying many licenses. Please increase that time if sessions are still occupying. *Type:* Integer *Restart required:* Yes *System:* Yes *Optional:* Yes *Example value:* 5 *Since:* 6.8.2

11.2 System Properties Ordered by Property Name

Module	Property	Explanation
cmas-core-security	admin.email	*Description:* The e-mail address of the ConSol*CM administrator. The value which you have entered during system set-up is used initially. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* maz@consol.de *Since:* 6.0
cmas-core-security	admin.login	*Description:* The name of the ConSol*CM administrator. The value which you have entered during system set-up is used initially. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* admin *Since:* 6.0
cmas-app-admin-tool	admin.tool.session.check.interval	*Description:* Admin Tool inactive (ended) sessions check time interval (in seconds) *Type:* Integer *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* 30 *Since:* 6.7.5

Module	Property	Explanation
cmas-core-server	attachment.allowed.types	*Description:* Comma-separated list of allowed filename extensions (if no value defined, all file extensions are allowed). *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* txt,zip,doc *Since:* 6.5.0
cmas-core-server	attachment.max.size	*Description:* Maximum attachment size in MB *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 100 *Since:* 6.4.0
cmas-core-security	authentication.method	*Description:* User authentication method (internal CM database or LDAP authentication). Allowed values are LDAP or DATABASE *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* DATABASE *Since:* 6.0
cmas-dwh-server	autocommit.cf.changes	*Description:* *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.7.0

Module	Property	Explanation
cmas-app-admin-tool	autocomplete.enabled ***only version 6.9 and higher***	*Description:* If the flag is missing or its value is *false*, then the *Auto complete address* tab is hidden in AT. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* true *Since:* 6.9.2.0
cmas-dwh-server	batch-commit-interval	*Description:* Number of objects in a JMS message. Higher value means better transfer performance and bigger memory usage. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 100 *Since:* 6.0.0
cmas-core-index-common	big.task.minimum.size	*Description:* How many parts task at least should have to be handled by indexer with low priority. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 15 (default) *Since:* 6.8.3
cmas-core-cache	cache-cluster-name	*Description:* JBoss cache cluster name *Type:* String *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* 635a6de1-629a-4129-8299-2d98633310f0 *Since:* 6.4.0

Module	Property	Explanation
cmweb-server-adapter	checkUserOnlineIntervalInSeconds	*Description:* The interval in seconds to check which users are online (default 180sec = 3min). *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 180 *Since:* 6.0
cmas-core-shared	cluster.mode	*Description:* Flag if CMAS is running in cluster. *Type:* Boolean *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* false *Since:* 6.1.0
cmweb-server-adapter	cmoffice.enabled	*Description:* Flag if CM/Office is enabled. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.4.0
cmweb-server-adapter	commentRequiredForTicketCreation	*Description:* Flag if comment is a required field for ticket creation. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* true (default) *Since:* 6.2.0
cmas-core-server	config.data.version	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 11 *Since:* 6.0

Module	Property	Explanation
cmas-core-security	contact.authentication.method ***only version 6.9 and higher***	*Description:* Indicates contact authentication method, where possible values are *DATABASE or LDAP or LDAP,DATABASE or DATABASE,LDAP.* *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Since:* 6.9.3.0
cmas-core-security	contact.inherit.permissions.only.to.own.customer.group ***only version 6.9 and higher***	*Description:* Indicates whether authenticated contact inherits all customer group permissions from representing engineer (false) or only permission to own customer group (true). *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Since:* 6.9.2.3
cmweb-server-adapter	customizationVersion	*Description:* *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* cd58453e-f3cc-4538-8030-d15e8796a4a7 *Since:* 6.5.0
cmas-core-shared	data.directory	*Description:* Directory for CMAS data (e.g. index) *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* C:\Users\user\cmas *Since:* 6.0

Module	Property	Explanation
cmweb-server-adapter	data.optimization	*Description:* Defines optimization to be applied on response data. So far, the following values are supported (for setting more than one value, separate values by '|'): MINIFICATION and COMPRESSION. MINIFICATION minifies HTML data by e.g. stripping whitespaces and comments. COMPRESSION applies gzip compression to HTTP response. (Note: If you are running in cluster mode and want to test different configurations in parallel, you can set different values for each cluster node by specifying property data.optimization.*nodeId* to override default property.) *Type:* String *Restart required:* COMPRESSION can be switched on/off without restart, MINIFICATION requires restart *System:* Yes *Optional:* Yes *Example value:* MINIFICATION|COMPRESSION
cmas-core-server	defaultCommentClassName	*Description:* Default text class name for comments *Type:* String *Restart required:* No *System:* No *Optional:* Yes *Example value:* *Since:* 6.3.0

Module	Property	Explanation
cmweb-server-adapter	defaultContentEntryClassName	*Description:* Default text class for new acims *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* default_class *Since:* 6.3.0
cmas-core-server	defaultIncommingMailClassName	*Description:* Default text class name for incoming mails *Type:* String *Restart required:* No *System:* No *Optional:* Yes *Example value:* *Since:* 6.3.0
cmweb-server-adapter	defaultNumberOfCustomFieldsColumns	*Description:* Default number of columns for custom fields *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 3 *Since:* 6.2.0
cmas-core-server	defaultOutgoingMailClassName	*Description:* Default text class name for outgoing mails *Type:* String *Restart required:* No *System:* No *Optional:* Yes *Example value:* *Since:* 6.3.0
cmas-core-index-common	disable.admin.task.auto.commit	*Description:* All tasks created for index update will be automatically executed right after creation. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.6.1

Module	Property	Explanation
cmas-dwh-server	dwh.mode	*Description:* Current mode of DWH data transfer. Possible values are OFF, ADMIN, LIVE *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* OFF *Since:* 6.0.1
cmas-esb-core	esb.directory	*Description:* Directory used by ESB (Mule) *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* C:\Users\user\cmas\mule *Since:* 6.0
cmas-core-cache	eviction.event.queue.size	*Description:* *Type:* Integer *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* 200000 *Since:* 6.4.0
cmas-core-cache	eviction.max.nodes	*Description:* *Type:* Integer *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* 100000 *Since:* 6.4.0
cmas-core-cache	eviction.wakeup.interval	*Description:* *Type:* Integer *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* 3000 *Since:* 6.4.0

Module	Property	Explanation
cmweb-server-adapter	favoritesSizeLimit	*Description:* Maximum number of items in favorites list *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 10 *Since:* 6.0
cmas-workflow-jbpm	fetchLock.interval	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 5000 *Removed in:* 6.8.0
cmas-workflow-jbpm	fetchLock.timeout	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 15000 *Removed in:* 6.8.0
cmas-core-server	fetchSize.strategy	*Description:* Strategy selected to set fetch size on jdbc result sets. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* FetchSizePageBasedStrategy, FetchSizeThresholdStrategy, FetchSizeFixedStrategy *Since:* 6.8.4.1

Module	Property	Explanation
cmas-core-server	fetchSize.strategy.FetchSizeFixedStrategy.value	*Description:* Sets fetch size value if selected strategy to set fetch size is *FetchSizeFixedStrategy*. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 150 *Since:* 6.8.4.1
cmas-core-server	fetchSize.strategy.FetchSizePageBasedStrategy.limit	*Description:* Sets max fetch size value if selected strategy to set fetch size is *FetchSizePageBasedStrategy*. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 10000 *Since:* 6.8.4.1
cmas-core-server	fetchSize.strategy.FetchSizeThresholdStrategy.value	*Description:* Sets fetch size threshold border values if selected strategy to set fetch size is *FetchSizeThresholdStrategy*. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 150,300,600,1000 *Since:* 6.8.4.1
cmweb-server-adapter	globalSearchResultSizeLimit	*Description:* Maximum number of items in global (Q&E) search result *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 10 *Since:* 6.0

Module	Property	Explanation
cmweb-server-adapter	helpFilePath	*Description:* URL for online help. If not empty, *Help* button is displayed in Web Client. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* http://www.consol.de *Since:* 6.2.1
cmas-setup-hibernate	hibernate.dialect	*Description:* The dialect used by hibernate. Usually set during initial setup (depending on the database system). *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* org.hibernate.dialect.MySQL5InnoDBDialect *Since:* 6.0
cmweb-server-adapter	hideTicketSubject	*Description:* If set to *true*, ticket subject is hidden. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.2.1
cmas-dwh-server	ignore-queues	*Description:* By adding a comma separated list of queue names it is configured that tickets of these queues are not transferred to the DWH. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* QueueName1,QueueName2,QueueName3 *Since:* 6.6.19 *Removed in:* 6.8.1

Module	Property	Explanation
cmas-core-index-common	index.attachment	*Description:* Describes if content of attachments is indexed. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* true *Since:* 6.4.3
cmas-core-index-common	index.history	*Description:* Describes if unit and ticket history are indexed. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.1.0
cmas-core-index-common	index.status	*Description:* Status of the indexer, possible values RED, YELLOW, GREEN, will be displayed in the Admin-Tool. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* GREEN *Since:* 6.6.1
cmas-core-index-common	index.task.worker.threads	*Description:* How many threads will be used to execute batch index tasks (synchronization, administrative, and repair tasks). *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 1 (default) (we recommend to use a value not larger than 2) *Since:* 6.6.14, 6.7.3

Module	Property	Explanation
cmas-core-index-common	index.version.current	*Description:* Holds information about current (possibly old) index version. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 1 (default) *Since:* 6.7.0
cmas-core-index-common	index.version.newest	*Description:* Holds information about which index version is considered newest. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 1 (default) *Since:* 6.7.0
cmas-core-index-common	indexed.assets.per.thread.in.memory	*Description:* How many assets should be loaded into memory at once during indexing per one thread. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 200 (default) *Since:* 6.8.0
cmas-core-index-common	indexed.engineers.per.thread.in.memory	*Description:* How many engineers should be loaded into memory at once during indexing per one thread. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 300 (default) *Since:* 6.6.14, 6.7.3

Module	Property	Explanation
cmas-core-index-common	indexed.tickets.per.thread.in.memory	*Description:* How many tickets should be loaded into memory at once during indexing per one thread. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 100 (default) *Since:* 6.6.14, 6.7.3
cmas-core-index-common	indexed.units.per.thread.in.memory	*Description:* How many units should be loaded into memory at once during indexing per one thread. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 200 (default) *Since:* 6.6.14, 6.7.3

Module	Property	Explanation
cmas-setup-manager	initialized	*Description:* Flag if CMAS is initialized. If this value is missing or not *true*, setup will be performed. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* true *Since:* 6.0 Be careful with using this property!!! When you set the value to *false*, the ConSol*CM server will perform the system set-up at the next start, i.e. all data of the existing system is lost, including system properties!!!
cmas-dwh-server	is.cmrf.alive	*Description:* As a starting point time of sending last message to CMRF should be used. If response from CMRF is not received after value (in seconds) it should create a DWH operation status with error message that CMRF is down. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 1200 *Since:* 6.7.0

Module	Property	Explanation
cmas-dwh-server	java.naming.factory.initial	*Description:* Factory class for DWH context factory. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* org.jnp.interfaces.NamingContextFactory *Since:* 6.0.1
cmas-dwh-server	java.naming.factory.url.pkgs	*Description:* *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* org.jboss.naming:org.jnp.interfaces *Since:* 6.0.1
cmas-dwh-server	java.naming.provider.url	*Description:* URL of naming provider *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* localhost *Since:* 6.0.1
cmas-workflow-engine	jobExecutor.adminMail	*Description:* Mail which will get notified about job execution problems (when retry counter is exceeded). *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* admin@consol.de *Since:* 6.8.0

Module	Property	Explanation
cmas-workflow-jbpm	jobExecutor.idleInterval	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 45000 *Removed in:* 6.8.0 *Replaced by:* jobExecutor.idleInterval.seconds
cmas-workflow-engine	jobExecutor.idleInterval.seconds	*Description:* Determines how often job executor thread will look for new jobs to execute. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 5 (default) *Since:* 6.8.0
cmas-workflow-jbpm	jobExecutor.jobExecuteRetryNumber	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 5 *Removed in:* 6.8.0 *Replaced by:* jobExecutor.jobMaxRetries
cmas-workflow-engine	jobExecutor.jobMaxRetries	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 5 (default) *Since:* 6.8.0

Module	Property	Explanation
cmas-workflow-engine	jobExecutor.jobMaxRetriesReachedSubject	*Description:* (rev.54593) *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* Job max retries reached. Job was removed!!! (default) *Since:* 6.8.0
cmas-workflow-engine	jobExecutor.lockingLimit	*Description:* Number of job locked at once (marked for execution) by job executor thread *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 10 (default) *Since:* 6.8.0
cmas-workflow-engine	jobExecutor.lockTimeout.seconds	*Description:* How long the job can be locked (marked for execution) by job executor. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 360 (default) *Since:* 6.8.0
cmas-workflow-engine	jobExecutor.mailFrom	*Description:* Mail which will be set as *From* header during admin notifications. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* jobexecutor@consol.de *Since:* 6.8.0

Module	Property	Explanation
cmas-workflow-engine	jobExecutor.maxInactivityInterval.minutes	*Description:* Number of minutes of allowed job executor inactivity (e.g. when it is blocked by long timer execution). After this time executors threads are restarted. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes. Default value is set to 30 minutes. *Example value:* 15 (default) *Since:* 6.9.2.0
cmas-workflow-engine	jobExecutor.threads	*Description:* Number of job execution threads *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 1 (default) *Since:* 6.8.0
cmas-workflow-jbpm	jobExecutor.timerRetryInterval	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 10000 *Removed in:* 6.8.0 *Replaced by:* jobExecutor.timerRetryInterval.seconds
cmas-workflow-engine	jobExecutor.timerRetryInterval.seconds	*Description:* Determines how long job executor thread will wait after job execution error. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 10 (default) *Since:* 6.8.0

Module	Property	Explanation
cmas-workflow-engine	jobExecutor.txTimeout.seconds	*Description:* Transaction timeout used for job execution *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 60 (default) *Since:* 6.8.0
cmas-core-security	kerberos.v5.enabled	*Description:* Flag which indicates whether SSO via Kerberos is enabled. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false (default if Kerberos has not been enabled during system set-up) *Since:* 6.2.0
cmas-core-security	kerberos.v5.username.regex	*Description:* Regular expression used for mapping Kerberos principal to CM user login. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* (.*)@.* *Since:* 6.2.0
cmas-core-server	last.config.change	*Description:* Random UUID created during last change in config *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* 2573c7b7-2bf5-47ff-b5a2-bad31951a266 *Since:* 6.1.0, 6.2.1

Module	Property	Explanation
cmas-core-security	ldap.authentication	*Description:* Authentication method used when using LDAP authentication. *Type:* String *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* simple *Since:* 6.0
cmas-core-security	ldap.basedn	*Description:* Base DN used for looking up LDAP user accounts when using LDAP authentication. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* OU=accounts,DC=consol,DC=de *Since:* 6.0
cmas-core-server	ldap.certificate.basedn	*Description:* Base DN for certificates location in LDAP tree. If not provided, ldap.basedn is taken. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* OU=accounts,DC=consol,DC=de *Since:* 6.8.4
cmas-core-server	ldap.certificate.content.attribute	*Description:* LDAP attribute name used where certificate data is stored in LDAP tree. Default value is: usercertificate. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* usercertificate *Since:* 6.8.4

Module	Property	Explanation
cmas-core-server	ldap.certificate.password	*Description:* LDAP Certificates manager password. If not set, ldap.password is taken. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.8.4
cmas-core-server	ldap.certificate.providerurl	*Description:* LDAP Certificates provider URL. If not set, ldap.providerurl is taken. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* ldap://ldap.consol.de:389 *Since:* 6.8.4
cmas-core-server	ldap.certificate.searchattr	*Description:* LDAP attribute name used to search for certificate in LDAP tree. Default value is: mail. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* mail *Since:* 6.8.4
cmas-core-server	ldap.certificate.userdn	*Description:* LDAP Certificates manager DN. If not set, ldap.userdn is taken. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.8.4

Module	Property	Explanation
cmas-core-security	ldap.contact.name.basedn ***only version 6.9 and higher***	*Description:* Base path to search for contact DN by LDAP ID (e.g. ou=accounts,dc=consol,dc=de) *Type:* String *Restart required:* No *System:* No *Optional:* Yes *Since:* 6.9.3.0
cmas-core-security	ldap.contact.name.password ***only version 6.9 and higher***	*Description:* Password to lookup contact DN by LDAP ID. If not set, anonymous account is used. *Type:* String *Restart required:* No *System:* No *Optional:* Yes *Since:* 6.9.3.0
cmas-core-security	ldap.contact.name.providerurl ***only version 6.9 and higher***	*Description:* Address of the LDAP server (ldap[s]://host:port) *Type:* String *Restart required:* No *System:* No *Optional:* Yes *Since:* 6.9.3.0
cmas-core-security	ldap.contact.name.searchattr ***only version 6.9 and higher***	*Description:* Attribute to search for contact DN by LDAP ID (e.g. uid) *Type:* String *Restart required:* No *System:* No *Optional:* Yes *Since:* 6.9.3.0
cmas-core-security	ldap.contact.name.userdn ***only version 6.9 and higher***	*Description:* User DN to lookup contact DN by LDAP ID. If not set, anonymous account is used. *Type:* String *Restart required:* No *System:* No *Optional:* Yes *Since:* 6.9.3.0

Module	Property	Explanation
cmas-core-security	ldap.initialcontextfactory	*Description:* Class name for initial context factory of LDAP implementation when using LDAP authentication. If it is not set, com.sun.jndi.ldap.LdapCtxFactory is being used as value. *Type:* String *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* com.sun.jndi.ldap.LdapCtxFactory *Since:* 6.0
cmas-core-security	ldap.password	*Description:* Password for connecting to LDAP to lookup users (when using LDAP authentication). Only needed if lookup cannot be done anonymously. *Type:* Password *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.1.2
cmas-core-security	ldap.providerurl	*Description:* LDAP provider (when using LDAP authentication). *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* ldap://ldap.consol.de:389 *Since:* 6.0

Module	Property	Explanation
cmas-core-security	ldap.searchattr	*Description:* Search attribute for looking up LDAP entry connected to CM6 login. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* uid *Since:* 6.0
cmas-core-security	ldap.userdn	*Description:* LDAP user for connecting to LDAP to lookup users (when using LDAP authentication). Only needed if lookup cannot be done anonymously. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.1.2
cmas-esb-mail	mail.attachments.validation.info.sender	*Description:* Sets *From* header of attachments type error notification mail. As a default the e-mail address of the administrator which you have entered during system set-up is used. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* admin@consolcm.com *Since:* 6.7.5

Module	Property	Explanation
cmas-esb-mail	mail.attachments.validation.info.subject	*Description:* Sets subject of attachments type error notification mail. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* Mail was not processed because its attachments were rejected!!! *Since:* 6.7.5
cmas-esb-mail	mail.callname.pattern	*Description:* Regular expression for subject of incoming mails. Available as TICKET_NAME_PATTERN_FORMAT in incoming mail scripts. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* .*?Ticket\s+\((\S+)\).* *Since:* 6.0
cmas-esb-mail	mail.cluster.node.id	*Description:* Only the node whose mail.cluster.node.id equals cmas.clusternode.id will start the Mule ESB mail services. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* unspecified *Since:* 6.6.5
cmas-esb-mail	mail.db.archive	*Description:* If property is set to *true*, incoming e-mails are archived in the database. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* false (default) *Since:* 6.8.5.5

Module	Property	Explanation
cmas-esb-mail	mail.delete.read	*Description:* Determines whether CM deletes messages fetched via IMAP(S). Setting value to *true* will cause deletion of messages after fetching. Default is to not delete messages fetched via IMAP(S). Note: Messages fetched via POP3(S) will always be deleted. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* true *Since:* 6.7.3
cmas-esb-mail	mail.encryption	*Description:* If property is set to *true*, the encrypt check box in the Ticket E-Mail Editor is checked by default. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* true (default = false) *Since:* 6.8.4.0
cmweb-server-adapter	mail.from	*Description:* Use this address if set instead of engineer e-mail address during mail conversation. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.1.2

Module	Property	Explanation
cmas-esb-mail	mail.incoming.uri	*Description:* URL for incoming mails *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* pop3://cm-incoming-user:password@localhost:10110 *Since:* 6.0 This value should not be edited here using the system properties pop-up window, but the mailboxes should be configured using the file card *E-mail* in the Admin-Tool (see *ConSol*CM Administrator Manual* section *File Card E-mail*). Using this standard feature all entries are controlled - i.e. for each mailbox which is added, CM establishes a test connection during mailbox set-up. That way it is not possible to enter wrong values.
cmas-esb-mail	mail.max.restarts	*Description:* Maximum number of mail service restarts before giving up *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 3 *Since:* 6.0

Module	Property	Explanation
cmas-esb-mail	mail.mime.strict	*Description:* If set to *false*, mail addresses are not parsed for strict MIME compliance. Default is *true*, which means check for strict MIME compliance. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.6.17, 6.7.3
cmas-esb-mail	mail.mule.service	*Description: From* address for mails sent by Mule service *Type:* EMail *Restart required:* No *System:* Yes *Optional:* No *Example value:* maz@consol.de *Since:* 6.0
cmas-core-server	mail.notification.engineerChange	*Description:* Flag if notification mail should be sent when engineer of ticket is changed. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* true *Since:* 6.1.0
cmas-core-server	mail.notification.sender	*Description: From* address for notification mails when engineer of ticket is changed. If not set, *cmas-core-security admin.email* is used instead. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* cm6notification@cm6installation *Since:* 6.6.3

Module	Property	Explanation
cmas-esb-mail	mail.polling.interval	*Description:* Mail polling interval in ms *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 60000 *Since:* 6.0
cmas-esb-mail	mail.process.error	*Description: To* address for error mails from Mule. As a default the e-mail address of the administrator which you have entered during system set-up is used. *Type:* EMail *Restart required:* No *System:* Yes *Optional:* No *Example value:* maz@consol.de *Since:* 6.0
cmas-esb-mail	mail.process.retry.attempts	*Description:* Number of retries when processing mail *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 3 *Since:* 6.0.2
cmas-esb-mail	mail.process.timeout	*Description:* Mail processing timeout in seconds *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 60 *Since:* 6.1.3

Module	Property	Explanation
cmas-esb-mail	mail.redelivery.retry.count	*Description:* Indicates the number of retries of re-delivering an e-mail from the CM system. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 3 *Since:* 6.1.0
cmweb-server-adapter	mail.reply.to	*Description:* When set, Web Client will display reply-to field on mail send, prefilled with this value. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.0.1 Please see also *ConSol *CM Administrator Manual* section *Queue Administration* . When you set the REPLY TO address in the outgoing e-mail script, the *mail.reply.to* system property must not be set (because it would overwrite the configured value)! That means when you use one outgoing e-mail script for a queue you have to define outgoing e-mail scripts for all queues because the *mail.reply.to* property can no longer be used.

Module	Property	Explanation
cmas-workflow-jbpm	mail.sender.address	*Description: From* address for mails from the workflow engine *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* maz@consol.de *Removed in:* 6.8.0 *Replaced by:* jobExecutor.mailFrom
cmas-core-server	mail.smtp.email	*Description:* SMTP mail URL for outgoing mails *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* smtp://mail.consol.de:25 *Since:* 6.0
cmas-core-server	mail.smtp.envelopesender	*Description:* Mail address used as sender in SMTP envelope. If not set, the *From:* address of the mail is used. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* mysender@mydomain.com *Since:* 6.5.7
cmweb-server-adapter	mailTemplateAboveQuotedText	*Description:* Indicates behavior of mail template in the Ticket E-Mail Editor when another mail is quoted, i.e. forwarded or replied to. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.2.4

Module	Property	Explanation
cmas-core-server	max.licences.perUser	*Description:* Sets max licenses single user can use (e.g logging in from different browsers). By default this value is not restricted. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 10 *Since:* 6.8.4.5
cmweb-server-adapter	maxSizePerPagemapInMegaBytes	*Description:* Maximum size (in MB) for each Wicket pagemap *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 15 *Since:* 6.3.5
cmas-core-server	monitoring.engineer.login ***only version 6.9 and higher***	*Description:* Login of monitoring engineer *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* bartek *Since:* 6.9.3.0
cmas-core-server	monitoring.unit.login ***only version 6.9 and higher***	*Description:* Login of monitoring unit *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* bartek *Since:* 6.9.3.0

Module	Property	Explanation
cmas-dwh-server	notification.error.description	*Description:* Text for error mails from DWH *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* Error occurred *Since:* 6.0.1
cmas-dwh-server	notification.error.from	*Description: From* address for error mails from DWH *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.0.1
cmas-dwh-server	notification.error.subject	*Description:* Subject for error mails from DWH *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* Error occurred *Since:* 6.0.1
cmas-dwh-server	notification.error.to	*Description: To* address for error mails from DWH *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* maz@consol.de *Since:* 6.0.1
cmas-dwh-server	notification.finished_successfully.description	*Description:* Text for mails from DWH when transfer finished successfully. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* Transfer finished successfully *Since:* 6.0.1

Module	Property	Explanation
cmas-dwh-server	notification.finished_successfully.from	*Description: From* address for mails from DWH when transfer finished successfully. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.0.1
cmas-dwh-server	notification.finished_successfully.subject	*Description:* Subject for mails from DWH when transfer finished successfully. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* Transfer finished successfully *Since:* 6.0.1
cmas-dwh-server	notification.finished_successfully.to	*Description: To* address for mails from DWH when transfer finished successfully. *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* maz@consol.de *Since:* 6.0.1
cmas-dwh-server	notification.finished_unsuccessfully.description	*Description:* Text for mails from DWH when transfer finished unsuccessfully. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* Transfer finished unsuccessfully *Since:* 6.0.1

Module	Property	Explanation
cmas-dwh-server	notification.finished_unsuccessfully.from	*Description: From* address for mails from DWH when transfer finished unsuccessfully. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.0.1
cmas-dwh-server	notification.finished_unsuccessfully.subject	*Description:* Subject for mails from DWH when transfer finished unsuccessfully. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* Transfer finished unsuccessfully *Since:* 6.0.1
cmas-dwh-server	notification.finished_unsuccessfully.to	*Description: To* address for mails from DWH when transfer finished unsuccessfully. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* maz@consol.de *Since:* 6.0.1
cmas-dwh-server	notification.host	*Description:* Mail (SMTP) server hostname for sending DWH mails *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* mail.consol.de *Since:* 6.1.0

Module	Property	Explanation
cmas-dwh-server	notification.password	*Description:* Password for sending DWH mails (optional) *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.1.0
cmas-dwh-server	notification.port	*Description:* SMTP port for sending DWH mails *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 25 *Since:* 6.1.0
cmas-dwh-server	notification.protocol	*Description:* The protocol used for sending emails from DWH. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* pop3\
cmas-dwh-server	notification.username	*Description:* (SMTP) User name for sending DWH mails *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* maz *Since:* 6.1.0
cmas-workflow-jbpm	outdated.lock.age	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 60000 *Removed in:* 6.8.0 *Replaced by:* jobExecutor.lockTimeout.seconds

Module	Property	Explanation
cmweb-server-adapter	pagemapLockDurationInSeconds	*Description:* Number of seconds to pass before pagemap is considered to be locked for too long. *Type:* Integer *Restart required:* Yes *System:* Yes *Optional:* Yes *Example value:* 60 *Since:* 6.7.3
cmweb-server-adapter	postActivityExecutionScriptName	*Description:* Defines the name for the script which should be executed after every workflow activity (see *ConSol*CM Administrator Manual* section *Admin-Tool Scripts - Default Workflow Activity Script*). If no script should be executed, leave the value empty. *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* postActivityExecutionHandler *Since:* 6.2.0
cmweb-server-adapter	queuesExcludedFromGS	*Description:* Comma-separated list of queue names which are excluded from global search. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.0

Module	Property	Explanation
cmas-workflow-jbpm	refreshTimeInCaseOfConcurrentRememberMeRequests	*Description:* It sets the refresh time (in seconds) after which page will be reloaded in case of concurrent *remember me* request s. This feature prevents one user from occupying many licenses. Please increase that time if sessions are still occupying. *Type:* Integer *Restart required:* Yes *System:* Yes *Optional:* Yes *Example value:* 5 *Since:* 6.8.2
cmweb-server-adapter	rememberMeLifetimeInMinutes	*Description:* Lifetime for *remember me* in minutes *Type:* Integer *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* 1440 *Since:* 6.0
cmweb-server-adapter	request.scope.transaction	*Description:* It allows to disable request scope transaction. By default one transaction is used per request. Setting this property to *false* there will cause one transaction per service method invocation. *Type:* Boolean *Restart required:* Yes *System:* Yes *Optional:* Yes *Example value:* true *Since:* 6.8.1

Module	Property	Explanation
cmas-setup-scene	scene	*Description:* Scene file which was imported during setup (can be empty). *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* vfszip:/P:/dist/target/jboss/server/cmas/deploy/cm-dist-6.5.1-SNAPSHOT.ear/APP-INF/lib/dist-scene-6.5.1-SNAPSHOT.jar/META-INF/cmas/scenes/helpdesk-sales_scene.jar/ *Since:* 6.0
cmweb-server-adapter	searchPageSize	*Description:* Default page size for search results *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 20 *Since:* 6.0
cmweb-server-adapter	searchPageSizeOptions	*Description:* Options for page size for search results *Type:* String *Restart required:* No *System:* Yes *Optional:* No *Example value:* 10|20|30|40|50|75|100 *Since:* 6.0
cmweb-server-adapter	serverPoolingInterval	*Description:* *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 5 *Since:* 6.1.0

Module	Property	Explanation
cmas-core-server	server.session.archive.reaper.interval	*Description:* Server archived sessions' reaper interval (in seconds) *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 60 *Since:* 6.7.1
cmas-core-server	server.session.archive.timeout	*Description:* Server sessions archive validity timeout (in days). After this time session info is removed from DB. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 31 *Since:* 6.7.1
cmas-core-server	server.session.reaper.interval	*Description:* Server inactive (ended) sessions' reaper interval (in seconds) *Type:* Integer *Restart required:* Only Session Service *System:* Yes *Optional:* No *Example value:* 60 *Since:* 6.6.1, 6.7.1

Module	Property	Explanation
cmas-core-server	server.session.timeout	*Description:* Server session timeout (in seconds) for connected clients. Each client can overwrite this timeout with custom value using its ID (ADMIN_TOOL, WEB_CLIENT, WORKFLOW_EDITOR, TRACK (before 6.8 please use PORTER), ETL, REST) appended to property name, e.g. server.session.timeout.ADMIN_TOOL *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 1800 *Since:* 6.6.1, 6.7.1
cmas-dwh-server	skip-ticket	*Description:* Tickets are not transferred during transfer/update. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.6.19 *Removed in:* 6.8.1
cmas-dwh-server	skip-ticket-history	*Description:* History of ticket is not transferred during transfer/update. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.6.19 *Removed in:* 6.8.1

Module	Property	Explanation
cmas-dwh-server	skip-unit	*Description:* Units are not transferred during transfer/update. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.6.19 *Removed in:* 6.8.1
cmas-dwh-server	skip-unit-history	*Description:* History of unit is not transferred during transfer/update. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* false *Since:* 6.6.19 *Removed in:* 6.8.1
cmas-dwh-server	split.history	*Description:* Changes the SQL that fetches the history for the tickets during DWH transfer not to all tickets at once but only for one ticket per SQL. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* false *Since:* 6.8.0
cmweb-server-adapter	supportEmail	*Description:* *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Since:* 6.0

Module	Property	Explanation
cmas-core-index-common	synchronize.master.address	*Description:* Value of *-Dcmas.http.host.port* informing how to connect to indexing master server. Default null. Since 6.6.17 this value is configurable in setup to designate initial indexing master server. Please note that changing this value is only allowed when all cluster nodes index changes receivers are stopped. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 127.0.0.1:80 *Since:* 6.6.0
cmas-core-index-common	synchronize.master.security.token	*Description:* The password for accessing the index snapshot via URL, e.g. for index synchronizaton or for back-ups. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* token *Since:* 6.6.0
cmas-core-index-common	synchronize.master.security.user	*Description:* The user name for accessing the index snapshot via URL, e.g. for index synchronizaton or for back-ups. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* user *Since:* 6.6.0

Module	Property	Explanation
cmas-core-index-common	synchronize.master.timeout.minutes	*Description:* How much time master server may constantly fail until new master gets elected with index fix procedure. Default 5. Since 6.6.17 this value is configurable in setup where zero means that master server will never change (failover mechanism is off). *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 5 *Since:* 6.6.0
cmas-core-index-common	synchronize.megabits.per.second	*Description:* How much bandwidth can master server consume to transfer index changes to all slave servers. Default 85. Please do not use all available bandwidth to transfer index changes between hosts. This will most probably partition cluster as some subsystems will not be able to communicate. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 85 *Since:* 6.6.0
cmas-core-index-common	synchronize.sleep.millis	*Description:* How often each slave server polls master server for index changes. Default 1000. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 1000 *Since:* 6.6.0

Module	Property	Explanation
cmweb-server-adapter	themeOverlay	*Description:* Name of used theme overlay *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* kyoEUR *Since:* 6.0
cmas-core-server	ticket.delete.timeout	*Description:* Transaction timeout (in seconds) for deleting tickets *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 60 *Since:* 6.1.3
cmweb-server-adapter	ticketListRefreshIntervalInSeconds	*Description:* Refresh interval for ticket list (in seconds) *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 180 *Since:* 6.0
cmweb-server-adapter	ticketListSizeLimit	*Description:* Maximum number of tickets in ticket list *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 100 *Since:* 6.0

Module	Property	Explanation
cmas-core-server	tickets.delete.size	*Description:* Property that defines a number of tickets deleted per transaction. By default it is set to 10. *Type:* Integer *Restart required:* Only Session Service *System:* Yes *Optional:* No *Example value:* 10 *Since:* 6.8.1
cmweb-server-adapter	unitIndexSearchResultSizeLimit	*Description:* Maximum number of units in unit search result (e.g. when searching for contact) *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 5 *Since:* 6.0
cmas-core-server	unit.replace.batchSize	*Description:* Describes number of objects to be processed in unit replace action. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 5 *Since:* 6.8.2
cmas-core-server	unit.replace.timeout	*Description:* Transaction timeout (seconds) of unit replacement action step. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 120 *Since:* 6.8.2

Module	Property	Explanation
cmas-dwh-server	unit.transfer.order	*Description:* Define in which order unit custom field groups should be transferred to the DWH. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value: company;customer* *Since:* 6.6.19 *Removed in:* 6.8.1
cmas-core-server	unused.content.remover.cluster.node.id ***only version 6.9 and higher***	*Description:* Value of a cmas.clusternode.id designating node which will remove unused ticket attachments and unit content entries. *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* 1 (assuming cluster node started with -Dcmas.clusternode.id=1 parameter) *Since:* 6.9.0.0
cmas-core-server	unused.content.remover.enabled ***only version 6.9 and higher***	*Description:* Flag whether unused ticket attachments and unit content entries removal should take place. *Type:* Boolean *Restart required:* No *System:* Yes *Optional:* No *Example value:* true *Since:* 6.9.0.0

Module	Property	Explanation
cmas-core-server	unused.content.remover.polling.minutes ***only version 6.9 and higher***	*Description:* How often unused ticket attachments and unit content entries should be checked for removal. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 15 *Since:* 6.9.0.0
cmas-core-server	unused.content.remover.ttl.minutes ***only version 6.9 and higher***	*Description:* Minimum interval after which unused ticket attachments and unit content entries can be removed. *Type:* Integer *Restart required:* No *System:* Yes *Optional:* No *Example value:* 1440 *Since:* 6.9.0.0
cmweb-server-adapter	urlLogoutPath	*Description:* URL which is used when user logs out. (If no value is set, logout leads to login-mask.) *Type:* String *Restart required:* No *System:* Yes *Optional:* Yes *Example value:* http://intranet.consol.de *Since:* 6.3.1
cmweb-server-adapter	webSessionTimeoutInMinutes	*Description:* Session timeout in minutes *Type:* Integer *Restart required:* Yes *System:* Yes *Optional:* No *Example value:* 180 *Removed in:* 6.7.1 *Replaced by:* server.session.timeout

Module	Property	Explanation
cmweb-server-adapter	wicketAjaxRequestHeaderFilterEnabled	*Description:* This enables filter for Wicket AJAX requests, coming from stale pages with Wicket 1.4 scripting (CM6 pre-6.8.0), after update to CM6 post-6.8.0. *Type:* Boolean *Restart required:* Yes *System:* Yes *Optional:* Yes *Example value:* false *Since:* 6.8.1

12 Appendix D - Trademarks

- Microsoft® – Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. See Microsoft trademark web page
- Microsoft® Office – Microsoft and Microsoft Office are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. See Microsoft trademark web page
- Windows® operating system – Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. See Microsoft trademark web page
- Microsoft® Active Directory® – Microsoft and Microsoft Active Directory are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. See Microsoft trademark web page
- Microsoft® Word® – Microsoft and Microsoft Word are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. See Microsoft trademark web page
- Microsoft® SQL Server® – Microsoft and Microsoft SQL Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. See Microsoft trademark web page
- MuleSoft™ and Mule ESB™ are among the trademarks of MuleSoft, Inc. See Mule Soft web page
- Oracle® – Oracle is a registered trademark of Oracle Corporation and/or its affiliates. See Oracle trademarks web page
- Oracle® WebLogic – Oracle is a registered trademark of Oracle Corporation and/or its affiliates. See Oracle trademarks web page
- Pentaho® – Pentaho and the Pentaho logo are registered trademarks of Pentaho Inc. See Pentaho trademark web page

13 Index

P

Q

R

S

T

W