Workflows

Release 15.4.1.3

CONTACT Software

Oct 29, 2018

Contents

CHAPTER 1

Introduction

Note: Product development has high demands with respect to process support. Characteristic features are, for example, a high degree of parallelism and unforeseen changes and corrections in the course of a project. Conventional solutions soon reach their limits here. *CONTACT Workflows* offers comprehensive support for these requirements.

CONTACT Workflows supports a wide range of business processes directly in the system. They combine interactive and automated *tasks* in structure of temporal and logical dependencies called *workflow*.

A *workflow* is active or started if its status is **Execution**. A *task* is active or started if its status is **Execution**.

A *workflow* can either be instantiated directly by authorized users or from a template defined by domain experts. Before starting, a consistency check ensures, among other things, that the responsibility for all interactive *tasks* has been defined and that all *system tasks* have the necessary *parameters*.

There are two main tasks for administrators regarding *CONTACT Workflows*: *Monitoring* (page 3) and *Error Handling* (page 5).

This manual also contains technical details on individual components.

Note: For details on the general application and definition of a *workflow*, please read the wf_user:workflow_user.

CHAPTER 2

Interfaces to Other Applications

2.1 Digital Signature (cs.dsig)

The application *CONTACT Digital Signatures* adds a new task type to *CONTACT Workflows*. In order to complete these tasks, a responsible user has to apply their digital signature to it or specific objects.

2.2 Engineering Change Management (cs.ec)

CONTACT Engineering Changes uses *CONTACT Workflows* to model the individual process steps occurring in an Engineering Change (“Engineering Change Request”, “Engineering Change Order” and “Engineering Change Notification”).

2.3 Tasks (cs.taskmanager)

CONTACT Workflows contains a plugin for *CONTACT Tasks* to integrate its tasks.

CHAPTER 3

Monitoring

Note: This chapter contains information about the technical monitoring of *CONTACT Workflows*. For information about monitoring your own currently running *workflow*, please read the wf_user:workflow_user.

3.1 Workflow Service

If the *Workflow Service* is not up and running, the only effect on the running system is that *system tasks* in status **Execution** will not be processed. Your *workflows* are effectively paused, which can lead to delays in business processes.

An overview of pending and failed jobs can be found in the queue (available in the navigation menu under *Administration/Konfiguration* -> *Adminstration* -> *Processes* -> *System Task Jobs*). Successfully completed jobs are removed from the queue.

3.1.1 Impersonation

Each *system task* is processed in a separate subprocess of the service. To make sure access checks and writing logs is handled correctly, the subprocess impersonates the user who started the *workflow*. They are the only unambiguous choice of a person in charge of the *workflow*, since the configuration also allows naming a role as being responsible. In the database you will find the user's ID in the attribute `started_by` of the relation `cdbwf_process`.

Warning: After deactivating a user account, all *system tasks* of *workflows* started by this user will fail.

3.1.2 License Check

A service user account is used for license checks while processing a *system task*. The account is specified in the service arguments: `--svcuser`.

CONTACT Workflows includes the user account `cs.workflow.svcuser` for this purpose.

Note: The *workflow service* checks the licenses of the *workflow owner* when running a *system task*, but does not occupy them permanently to avoid disrupting interactive operation.

3.1.3 Logging

Log behavior of the subprocesses can be defined in `$CADDOK_BASE/etc/wfqueue.conf`.

3.2 Email Notifications

Users may opt in to email notifications whenever certain events occur.

In order for these to be sent, an email server must be configured (see environment variables `CADDOK_MAIL*` and `CADDOK_SMTP*` in the `site.conf`).

It is also possible to deactivate email notifications from *CONTACT Workflows* completely by setting the environment variable `CADDOK_STOP_EMAIL_NOTIFICATION` (also in `site.conf`). However, this is primarily intended for test and development systems.

CHAPTER 4

Error Handling

In order to assess whether the failure of individual *tasks* is in line with expectations or not, you should first familiarize yourself with the internal logic of *tasks* and *constraints* and read the wf_user:workflow_user.

In addition, actual errors may occur, the cause and treatment of which are dealt with in this section.

Errors during the execution of a *workflow* or a *task* always result in a change of the *workflow's* status to 🚫 **Failed**. In some cases the status of the *task* is also changed to ✖ **Discarded**.

We recommend the following procedure in case of unexpected errors in *CONTACT Workflows*:

1. Analysis
2. Mitigation
3. Repetition

Some steps may not be necessary in some cases.

4.1 Analysis

If a *workflow* has terminated prematurely due to some unknown reason, the last entries in the *workflow's protocol* should be analyzed first. In most cases, the reason for the termination can be found there in plain text.

Unexpected errors can also write a "traceback" into the *protocol*, which the error message from PowerScript including the call stack. To interpret a traceback, some knowledge of PowerScript is required.

Further sources of information are the server logs of *workflow service* and its queue.

4.2 Mitigation

Troubleshooting depends on the cause of the error. Some errors do not require administrative action, others do. In this section you will find examples of some error classes.

4.2.1 Premature Termination of System Tasks

While the status of some interactive *tasks* can be changed explicitly to 🚫 **Rejected** by the person responsible, this can also happen when processing *system tasks* for various reasons.

An example: A *system task* "Status Change" should change the status of its *briefcase contents* to `200`.

Example *Preconditions not met*

In our example, if there is an object in the *briefcase contents* for which no status transition from its current status to `200` is defined, the *task* will fail.

If the person responsible for a predecessor task was expected to establish the necessary preconditions for the status change to work, that user is at fault and there is no need for administrative action.

Example *Faulty workflow design*

An error in the *workflow's* design is, for example, if

- preconditions for the status change should have been established automatically, or
- no status change should have been attempted if it is impossible.

These kinds of errors have to be corrected by correcting the template. If the faulty *workflow* is not based on a template, one of the users responsible for the *workflow* has to make a copy of the *workflow* to be corrected.

Example *Error caused by task logic*

If the error is caused by the logic of the *task* itself, it has to be fixed by a developer. This is usually indicated by the fact that the *protocol* contains a PowerScript traceback.

4.2.2 Faulty Configuration

If a *task* is not configured correctly, it can also fail. Examples are missing required attributes or *parameters*. These rules are checked as soon as the *workflow's* status is changed to 🔄 **Execution** in order to detect such cases at an early stage. However, most aspects can still be changed after the *workflow* has started.

Configuration errors have their cause in the design of the *workflow* and have to be solved by a user responsible for the *workflow*.

4.2.3 System Errors

Errors can also be caused by faulty customizing or programming. In this case, further analysis is required.

4.3 Repetition

If the results of the failed *workflow* are not satisfactory, a corrected copy of the *workflow* can be started. If the original *workflow* was based on a template, the template should be corrected and reinstantiated instead.

If parts of the original *workflow* (which have already been processed successfully, for example) are no longer relevant for the repetition, they can simply be removed before starting the *workflow's* copy.

CHAPTER 5

Workflow Templates

Usually repeatable *workflows* are controlled by templates. Domain experts can create templates, optionally assign some rules and release the templates for usage.

5.1 Transporting Templates

Since *workflow* templates are inherently part of the business logic, they should be handled accordingly and kept in sync between all systems (such as development, test and production).

The following tool chain supports this:

1. `cs/workflow/updates/tools/export_process.py`
2. `cdbexp`
3. `cdbimp`

Warning: To avoid ID conflicts, you must take precautions. Possible solutions can be found in the section *Avoiding ID Conflicts* (page 8).

Warning: Only the *workflow* is transported along with its structure (*tasks*, *briefcases*, *constraints*...) and information (*Protocol*, activities). Excluded from the transport are referenced objects (but not the references themselves), such as *briefcase contents* and attachments to posts and comments in the Activity Stream.

5.1.1 Procedure

First of all, the *workflow* to be transported should be tested extensively in the source system. It is recommended to always transport *workflows* only in a fixed order (e.g. from development to testing, from testing to production).

Then, a control file for `cdbexp` can be created with the supplied command line tool `export_process`. The control file will contain the database queries needed to export the *workflow* and its structure:

```
powerscript -m cs.workflow.updates.tools.export_process P00000000 > P.ctl
```

If the call succeeds, the output file (`P.ctl` in the above example) can be used as input for `cdbexp`:

```
cdbexp -c P.ctl -o P.exp
```

The output file `P.exp` now contains the exported *workflow* structure that can be imported in the target system (please manage potential ID conflicts):

```
cdbimp P.exp
```

5.1.2 Avoiding ID Conflicts

Workflow IDs (`cdbwf_process.cdb_process_id`) and *task* IDs (`cdbwf_task.task_id`) are sequential numbers by default (`P000...` and `T000...`, respectively). This mechanism is not safe to use across multiple systems, so conflicts must be proactively avoided when transporting *workflows* and *tasks*. See below for some possible strategies.

Separate ID Ranges

The simplest way is to have each system use its own ID range, such a dedicated prefix for IDs (For example, development systems might use *workflow* IDs `X000...` instead of `T000....`).

Using this strategy, nothing else has to be done when transporting templates.

Separating IDs Manually

The export is a simple text file. With a suitable editor you can replace potentially conflicting IDs. Additionally, this solution allows for use of slightly more descriptive template IDs, since both IDs of *workflows* and *tasks* can also be short, unique texts.

Joining ID Ranges

If you want to use uniform IDs for *workflow* templates across all systems, you should at least separate the ID ranges of the templates from those of the instances.

Then, whenever a template is created, this should immediately be mirrored in all systems to "reserve" the same number. This placeholder can then be deleted in the target systems immediately before transporting the finished template.

CHAPTER 6

Configuration

6.1 Briefcases

This section contains technical requirements for using *briefcases*.

6.1.1 Briefcase Contents

A *briefcase* may contain objects of multiple classes. An object must fulfill all of the following requirements to be usable as *briefcase contents*:

- It must have the attribute `cdb_object_id`.
- It must have a corresponding PowerScript class.

Additionally, its PowerScript class may be derived from the class `cs.workflow.briefcases.BriefcaseContent` to re-use predefined event handlers (e.g. for the operation *Workflow/New*).

By default, some classes (like *Documents*) are already preconfigured for use as *briefcase contents*.

Other classes in your system can be configured for use as *briefcase contents* by meeting the above requirements.

Relationships

Briefcases always reference a *workflow*. Local *briefcases* are also linked to any number of *tasks* within their *workflow*.

Linking objects to a *briefcase* is done via the platform class `cdbfolder_content` and therefore links have no direct relation to the *workflow*.

To configure a relationship between objects of a class and their *workflow* links, you can use the class `cdbwf_process_content`, which contains the *briefcase contents* of each *workflow* redundantly.

This class can also be used as a PowerScript reference. The corresponding PowerScript class is `cs.workflow.briefcases.BriefcaseReference`.

6.1.2 Automatically Granting Access Rights

Workflows can automatically grant editing rights for *briefcase contents* to persons responsible for currently running *tasks*. For further information, see the User's Manual.

This feature is based on a relationship access profile and is restricted to a few predefined types (e.g. Parts, Documents, etc.) by default. You can extend automatically granting rights to other types by creating a relationship between the class `cdbwf_briefcase` (as *Referer*) and your target class (as *Reference*).

Use the following parameters to do so:

Referer `cdbwf_briefcase`

Relationship Class `cdbfolder_content`

Reference Your target class

Keymap (Referer) `cdb_object_id=cdb_folder_id`

Keymap (Reference) `cdb_object_id=cdb_content_id`

Relationship Access Profile `cdbwf_assign_rights`

When defining a profile, you should make sure the outcome matches the expectation depending on the *edit mode*.

Contents of **Info** *briefcases* should be readable, but not writeable (if write permissions are not granted by any other means), while *contents* of **Edit** *briefcases* should be both readable and writeable. For example, the standard configuration for documents grants these permissions

Edit Mode	Permission
Info	`read` and `read_file`
Edit	`read`, `read_file`, `save`, `accept`, `lock` and `unlock`

Hint: You can customize the profile to assign additional rights. Please contact your system administrator about this.

Accessing Folder Contents

In order for a user to be able to use objects in *briefcases*, they must be granted the `cdbwf_obj_obj_info` or `cdbwf_obj_edit` right on the *briefcase*, depending on the *edit mode*. This is to prevent a *workflow* from granting permissions not granted to the user performing the assignment.

The permissions `cdbwf_obj_info` and `cdbwf_obj_edit` are derived from permissions of the objects assigned to the *briefcase* (via the relationship access profile `cdbwf_assign_rights`) and are always checked when

- an object is added to the *briefcase contents*,
- the *edit mode* of a *briefcase* is changed,
- a *briefcase* is linked to another *task* or
- a *workflow* or *task* is started.

By default, permissions (on Documents, for example) are restricted so that the assignment does result in granting access rights the user performing the assignment is denied.

For certain objects, it may be desirable to never (or only by certain user groups) use them in *workflows* or *briefcases*, which may eventually grant write access. One reason could be the danger of releasing objects to the *CONTACT Collaboration Portal*. In this case, restricting which objects can be used as *briefcase contents* is recommended.

6.1.3 Adding Missing "Attachments" Briefcases to Workflow Templates

If a *workflow* template is missing the global "Attachments" *briefcase*, it will not be added automatically after an update. The following PowerScript code example will add the *briefcase* for all *workflow* templates that don't have it yet:

```
from cs.workflow.processes import Process

for process in Process.KeywordQuery(is_template="1"):
    process.make_attachments_briefcase()
```

6.2 Constraints

6.2.1 Object Rules Usable in the Workflow Designer

Constraints are object rules, but not all rules are suitable to be used as a *constraint*. The *Workflow Designer* only shows rules that have been released for the definition of *constraints*.

By default, only rules whose name begin with `wf-designer:` are visible. By modifying the PowerScript list `cs.workflow.designer.catalogs.ConstraintsCatalog.__catalog_rules_conditions__`, this convention can be adapted to your needs.

The list contains SQL conditions. The *Workflow Designer* offers all object rules matching at least one of these conditions for defining *constraints*.

Example

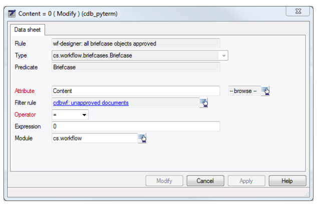
```
from cs.workflow.designer.catalogs import ConstraintsCatalog
catalog_conditions = ConstraintsCatalog.__catalog_rules_conditions__
catalog_conditions.append("name LIKE 'constraint-rule:%'")
```

6.2.2 Defining Your Own Object Rules for Constraints

When evaluating a *constraint* that is not referencing a specific *briefcase*, the object rule is checked against the *task* of the *constraint*. Otherwise the check is performed against the *briefcase*.

Example *All documents in briefcase are released*

The object rule should match a *briefcase* if the *briefcase* contains no documents or approved documents only. If the *briefcase* contains at least one unreleased document, the object rule should not match.



Content = 0 (Modify) (cdb_pyterm)

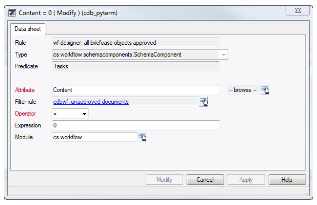
Data sheet

Rule	wf-designer: all briefcase objects approved
Type	cs.workflow.briefcases.Briefcase
Predicate	Briefcase
Attribute	Content
Filter rule	cdbwf: unapproved documents
Operator	=
Expression	0
Module	cs.workflow

Fig. 6.1: Object Rule *All documents in briefcase are released*

The `Content` attribute contains the *briefcase contents* as a list. Additionally, the filter "Document is not released" is applied. The final list is expected to be empty (e.g. its length is expected to be 0).

Example *All documents of task are released*



Content = 0 (Modify) (cdb_pyterm)

Data sheet

Rule	wf-designer: all briefcase objects approved
Type	cs.workflow.schemacomponents.SchemaComponent
Predicate	Tasks
Attribute	Content
Filter rule	cdbwf: unapproved documents
Operator	=
Expression	0
Module	cs.workflow

Fig. 6.2: Object Rule *All documents of task are released*

This example works just like the previous one, except this rule is meant to be checked against a *task*. In this case, `Content` contains the contents of all *briefcases* of the *task*.

To be even more specific, *tasks* also provides the attributes `InfoContent` and `EditContent`, which only provide the content of the respective *edit mode*.

6.3 Forms

Forms use the Maskenkonfiguration and are displayed just like masks are, with the *form* itself as the context object. For a mask to be used as a *form*, a form template must be created, named and released.

A *form's* data is stored as `JSON` (JavaScript Object Notation) in its long text attribute `cdbwf_form_contents_txt`.

6.3.1 Form Templates

Under *Administration/Konfiguration -> Katalogverwaltung -> Processes -> Form Templates* you can create new form templates or find existing ones. The following field are relevant when creating new ones:

Name Name of the *form* in all login languages. Should clearly state its intended purpose.

Mask Name of a Maskenkonfiguration. Defines both the appearance and the attribute set of the *form*. In addition, the usual mechanisms (catalogues, mandatory fields and presets) of the mask are used.

Warning: Mask groups are not supported, regular masks only.

Warning: Only explicit mandatory fields are evaluated. Since *forms* are contextless, mandatory fields from the Data Dictionary are not applied.

Form templates can only be used in *workflows* if their status is ✓ **Released**. Revoking the status ✓ **Released** has no effect on *forms* already instantiated from the template.

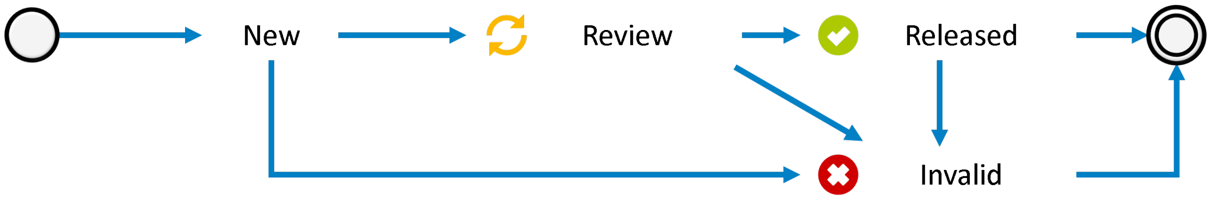


Fig. 6.3: Status Network of a Form Template

6.3.2 Access Rights

Warning: This section describes the default access configuration. Actual access in your system can differ due to customization.

Access Rights of Form Templates

Condition	Access Right	Granted For
Status **New**	• Full access	• *Administrator* • *Administrator: Master Data* • *cdbwf: Process Administrator* • *cdbwf: Process Library Manager*
Status not **New**	• Read • Status Change	• *Administrator* • *Administrator: Master Data* • *cdbwf: Process Administrator* • *cdbwf: Process Library Manager*

Access Rights of Forms

Read access on *forms* is unrestricted by default.

Access rights for *forms* are granted to responsible persons of a *task* by the relevant assignment to a *briefcase*.

Persons responsible for a *workflow* are granted the same access rights as on the *tasks*. Permissions on *forms* are mapped by the `Workflow components` relationship access profile to the *workflow* right `edit schema`.

6.4 System Tasks

Please note that *system tasks* will be executed by the *workflow service*.

6.4.1 Configuring System Tasks

In addition to the preconfigured *system task definitions*, you can add your own. The necessary steps for this are explained below in the *Example: Creating a Custom System Task* (page 15).

Adding a System Task Definition

Create a new *system task definition* first by navigating to *Administration/Konfiguration -> Katalogverwaltung -> Processes -> System Task Definitions* and choosing *New*.

The following fields may be filled in in the resulting dialog:

Identifier A unique identifier

Name (de) German name

Name (en) English name

Fully Qualified Python Name The name of a callable PowerScript object (typically a function) that implements the *system task* logic.

Implementing System Tasks

First, write the PowerScript function for the new *system task definition*. It must be accessible under the Python name entered in the dialog.

The system calls the function with the following parameters for each active *system task* of this definition:

task The *system task* as a `cdb.object.Object`.

content A PowerScript dictionary containing the *briefcase contents*. The keys are the possible *edit modes* (“info”, “edit”) of an object. The values are lists of the objects representing the respective contents.

******kwargs**** The remaining parameters correspond to the *parameters* configured in the definition (also see *Defining Parameters* (page 14)).

Error Handling

System task implementations should explicitly handle all possible exceptions and convert them into user-readable messages.

If you want to cancel the *system task* after an error, you can throw a `cs.workflow.systemtasks.TaskCancelledException` instead of the original exception. The system will also log the error message.

To cancel the entire *workflow*, use the class `cs.workflow.systemtasks.ProcessAbortedException` instead.

Customize Display in the Workflow Designer

The operation *Import image* in the context menu of a *system task definition* lets you define a *system task definition’s* graphical representation in the *Workflow Designer*.

Defining Parameters

For each *system task definition* a list of required *parameters* can be specified. Each of these *parameters* must exist for a *system task* to be processed, otherwise its status will immediately be changed to 🚫 **Rejected** upon being 🔄 **Execution**.

Note: To view and edit *parameters* in the *Workflow Designer*, it needs a suitable plugin. Please contact your system administrator about this.

Configuring the Included System Task “Status Change”

The included *system task* “Status Change” can unlock objects automatically, depending on the configuration. The setting `cs.workflow.status_change_unlock` must contain a valid value for the class name of the object (in the settings’s “Area” field) for the `unlock` parameter of the method `cdb.objects.Object.ChangeState`. No superclasses are taken into account, only the exact class names of the objects are.

By default, this is only defined for each of the classes *document*, *model* and *cdb_wsp* with the value *1* (unlock if permission is granted).

6.4.2 Example: Creating a Custom System Task

In this example, we will create our own *system task*, which copies *briefcase contents* and uses its own *parameters* to preset the copy operations.

Note: We assume that the `namespace` of our system is `demo` and that a custom module named `demo.plm` exists.

Creating the Definition

We will change the following parameters for our spin on the Copy *system task*:

Identifier demo_copy_objects

Name (de) Demo Objekte kopieren

Name (en) Copy Objects Demo

Fully Qualified Python Name `demo.plm.workflow.demo_copy_objects`

Additionally, you may *import an image for the new definition* (page 14).

Implementing the Function

Create the file `workflow.py` in the custom module `demo.plm` with the following contents:

```python
# coding: utf-8
"Example of a custom workflow system task"
from cdb import constants
from cdb import util
from cdb.objects import operations
from cs.workflow.systemtasks import TaskCancelledException

def demo_copy_objects(task, content, **kwargs):
    """
    Copy all objects in content["info"] (using kwargs to preset the copy
    operation) and put them in the "edit" briefcase.

    Only allows for exactly one "edit" briefcase.
    """
    if len(task.EditBriefcases) != 1:
        msg = util.get_label(
            "cdbwf_only_one_out_briefcase") % task.GetDescription()
        raise TaskCancelledException(msg)
    briefcase = task.EditBriefcases[0]

    for obj in content["info"]:
        copyobj = operations.operation(constants.kOperationCopy,
                                       obj,
                                       operations.form_input(obj, **kwargs))

        # unlock new object
        try:
            cd = obj.GetClassDef()
            if constants.kOperationUnlock in [
                oi.get_opname() for oi in cd.getOperationInfos()
            ]:
                operations.operation(constants.kOperationUnlock, copyobj)
        except RuntimeError:
            task.addProtocol(
                "cannot unlock object %s" % obj.GetDescription(),
                msgtype=protocols.MSGINFO)

        operations.operation(constants.kOperationNew,
                             FolderContent,
                             cdb_folder_id=briefcase.cdb_object_id,
                             cdb_content_id=copyobj.cdb_object_id)
```

Testing

Log on to the system again. In the *Workflow Designer*, you can now use the new *system task* to create *tasks*.

Note: The *parameters* for this *system task* can only be defined via the data sheet of the *system task* for now. For further information, see *Defining Parameters* (page 14).

6.5 Workflow Designer

6.5.1 Catalog Filters

The following catalogs used in the *Workflow Designer* use default filters not accessible to users:

Catalog	Default Filter
cs.workflow.designer.catalogs.ConstraintsCatalog	[“name LIKE ‘wf-designer:%’”]
cs.workflow.designer.catalogs.FormTemplateCatalog	[“status = 20”]
cs.workflow.designer.catalogs.OperationCatalog	[“name LIKE ‘CDB_%’”]

Filters are Python lists or sets containing SQL where conditions. During runtime, these are “OR”-concatenated and combined with user input before being applied.

See Workflows API for details on how to customize these filters.

6.6 Miscellaneous

6.6.1 Complete Prematurely

The option *Complete prematurely* (available for *tasks* of type *Approval* and *Examination*) is deactivated by default in the *Workflow Designer*.

To enable this option, set the property `wffo` to `true`.

6.6.2 Optional Performance Optimization

By default, the *Workflow Designer* checks permissions at startup to show or hide interactive elements depending on those permissions. This contradicts the usual pattern that elements are always active and permissions are only checked after actual interaction.

To deactivate these initial permission checks (for write permissions only, read permission on the *Workflow* itself is of course still required), you can set the default setting `cs.workflow.designer/check_access_proactively` to `0` or create user settings to set this specifically for individual users.

Additionally, the following optimization of the relationship access profiles configuration is recommended:

- Remove the assignment of the access profile `Workflow components` to the relationships `cdbwf_p2task`, `cdbwf_ag2task`, `cdbwf_p2component` and `cdbwf_p2aggr`.

- Set up a new relationship `cdbwf_p2all_components`, which links a *Workflow* with all objects of the class `cdbwf_process_component` of its `cdb_process_id` (regardless of their placement in the structure). This relationship then uses the access profile `Workflow components` to speed up the permission checks on structural elements.

Note: An SQL script for setting up these recommended changes is available at `cs/workflow/updates/tools/optional_performance_optimization.sql`.

CHAPTER 7

FAQ

7.1 System Tasks are not Completed

Check if the *workflow service* is up and running.

If the service is up and running and individual *system tasks* are still not completed, the associated jobs may have failed. You can restart the jobs as follows:

```
UPDATE mq_wfqueue SET cdbmq_state='W' WHERE cdbmq_state='F'
```

CHAPTER 8

Glossary

Completion Task

Special *task group* which is executed after the regular end of its *workflow*. The term is used synonymously in the *Workflow Designer* with *tasks* within this *task group*.

Completion tasks are usually used for exception handling or notifications.

- Example

Task

Element of a *workflow* that combines input data with a task description. The description is textual for interactive tasks and a PowerScript function for *system tasks*. Input data is linked to the task via *briefcases*. The behavior of a :term‘task‘ is determined by its wf_user:task_types.

- wf_user:create_tasks
- wf_user:task_olc

CONTACT Tasks offers users an overview of their tasks in status **Execution** and the opportunity to easily complete them.

Edit Mode

Controls the wf_admin:briefcase_contents_access. Possible values:

Database Value	Name	Permissions Granted (Example)
0	**Info**	Read (read, read_file)
1	**Edit**	Write (accept, lock, save, unlock)

- Details
- Setup

Constraint

You can use constraints to define conditions that are checked immediately after the status change of a *task* in **Execution**. If there is at least one violated constraint for the *task*, the status of the *task* is set to **Discarded**. Constraints are object rules.

- Usage
- Setup

Form

Forms enable capturing and processing structured data (metadata) in *workflows* and their *tasks*.

A form is always instantiated from a *template*.

- Usage
- Setup

Briefcase

Briefcases aggregate input or output data (see *briefcase contents*), either for an entire *workflow* ("global briefcase") or individual *tasks* ("local briefcase").

The assignment of a briefcase to a *workflow* or a *task* additionally contains an *edit mode* for additional access rights management.

- Usage
- Setup

Briefcase Contents Object assigned as content to a *briefcase*. Possible contents are *forms*, files, or business objects that serve as input or output data of a *workflow* or a *task*.

Parameter

Filter Parameter *System tasks* may contain parameters serving as input for processing them. Parameters of the *system task definition* are mandatory.

Protocol The protocol of a *workflow* contains messages (both regular and unexpected events) and their timestamps that are useful for monitoring and subsequent analysis.

- Additional Information

Task Group

Task Groups are containers for *tasks*, which do not have their own logic apart from the execution order of their child *tasks*.

Sequential task groups run their child *tasks* strictly one after another, parallel task groups run them at the same time. Task Groups only reach status **Completed** when all child *tasks* are in a final status (**Completed**, **Rejected** or **Discarded**).

Workflows behave like sequential task groups themselves.

Task groups are not explicitly displayed in the *Workflow Designer*. *Tasks* displayed in a column share a parallel parent task group, *tasks* displayed in a row are either direct children of the *workflow* or of a mutual sequential parent task group.

System Task

A *task* which is not processed interactively by a user, but automatically by a PowerScript function. System tasks in status **Execution** are run asynchronously by the *workflow service*.

- Usage
- Setup

System Task Definition Logic, name and appearance of a *system task* are determined by their definition. Each *system task* must reference a definition via its attribute `task_definition_id`.

Your system administrator can add new definitions to expand the stock of usable *system task* types. Please read the section in the administrator's manual.

Template A template is an object that cannot be used itself, but can be copied to get a usable object. Copying a template is called "instantiating".

Objects that can be created from templates are *workflows* and *forms*.

Workflow

A workflow contains temporal and logical dependency structures between *task groups* and interactive and automatic *tasks*. It is able to model entire business processes in some instances.

A workflow may also be a *template*.

- Menu access: *Processes –> Workflows*

Workflow Designer

Web application supporting the graphical design of a *workflow*.

- Usage

Workflow Owner The user who changes the status of a *workflow* to **Execution** will from then on be considered its "owner". They are personally responsible for the correctness of the *workflow* design at that time.

Workflow Service This service runs *system tasks* in status **Execution**. It is implemented as a "Message Queue". Whenever a *system task's* status is set to **Execution**, a job which controls the execution by the service is created.

The service's name is *cs.workflow.services.WFServer*, that of the queue *wfqueue*.

- Additional Information
- Message Queues
- Services

Workflow Category Categories allow for logically grouping *workflows*. They let users preselect *workflow* templates more easily. Possible category names could be *Tender*, *Engineering Change Order* or *Request for Information*.

- Menu access *Administration/Configuration -> Catalog Administration -> Processes -> Workflow Categories*

List of Figures

List of Tables

Index