OTRS ITSM Manual

Release 2024.6.1

OTRS AG

Sep 06, 2024

CONTENTS

1 Introduction **3**

2 Configuration Management **5**

2.1 Administrator Interface . . . 5
2.1.1 Users, Groups & Roles . . . 5
2.1.1.1 Groups . . . 5
2.1.2 Processes & Automation . . . 6
2.1.2.1 Process Management . . . 6
2.1.2.2 Web Services . . . 24
2.1.3 Administration . . . 26
2.1.3.1 General Catalog . . . 26
2.1.3.2 Import and Export . . . 28
2.1.3.3 System Configuration . . . 34
2.1.4 CMDB Settings . . . 37
2.1.4.1 Configuration Items . . . 37
2.2 Agent Interface . . . 42
2.2.1 Configuration Items . . . 43
2.2.1.1 Create Configuration Item . . . 43
2.2.1.2 Configuration Item List . . . 44
2.2.1.3 Configuration Item Detail View . . . 44
2.2.2 Customers . . . 47
2.2.3 Customer Users . . . 48
2.3 External Interface . . . 49

CHAPTER

ONE

INTRODUCTION

This manual is intended for OTRS::ITSM administrators and users to provide information on the basic use of OTRS::ITSM by IT service managers, IT service staff (agents) and end users (customers). Information pertaining to the installation, configuration and administration of OTRS::ITSM is only provided if there are differences to the OTRS core product or for functions, which only exist in OTRS::ITSM.

IT is expected to consistently deliver high service quality in an increasingly complex field. In this context, effective and efficient incident and problem management are indispensable. However, IT service management remains a task almost impossible if there is no consistent and up-to-date database with information about the state and configuration of the IT infrastructure.

The IT Infrastructure Library®, short ITIL®, is a series of books published by the United Kingdom's Office of Government Commerce (OGC), which generically combine best practice approaches to designing, providing, operating and managing IT services. ITIL does not focus on the technology but the services provided by the IT and comprises information on processes, roles, responsibilities, potential problem fields/resolutions, and definitions of terms.

ITIL has established itself as de facto standard over the past years and its circulation in IT organizations has contributed considerably to the development of a collective awareness and consistent terminology for IT service management. However, ITIL only describes **who should do what** and what should be considered along the way. In order to cover as wide a user group as possible, it does not or to a little extent address the issue of how to do things. Therefore, no implementable information is given for particular industries, companies, or manufacturers.

In December 2005, the ITIL based ISO/IEC 20000 industry standard for IT service management was published. IT organizations can apply for ISO/IEC 20000 certification and prove their conformity.

The continuing boom caused demand for IT service management tools, which could represent the ITIL-based processes. So far, only proprietary solutions existed. Because of their considerable complexity, most of these tools are only affordable for large companies and effective in large IT departments.

The development of OTRS::ITSM was started as a result of the great success of the OTRS framework in order to combine the globally accepted, public ITIL recommendations with the benefits of open-source software.

OTRS::ITSM was the first real-world ITIL compliant IT service management solution on open-source basis, built on the solid basis of OTRS with its over thousands known OTRS installations and its community.

OTRS::ITSM is practically-oriented. This was accomplished by developing it in collaboration with ITIL consultants and with some of OTRS Groups' customers.

The service-desk and ticket system solution OTRS is the basis for the ITIL compliant IT service management solution OTRS::ITSM, its incident management, problem management, service level management, configuration management modules, and integrated CMDB.

Just like ITIL, OTRS::ITSM does not claim to be an *out-of-the-box* solution for all tasks and questions arising in IT service management. It is, in fact, supposed to serve as a flexible, stable and easy to understand

information platform that can be adapted to meet the requirements of virtually every organization.

Therefore, please excuse us for bringing the following to your attention: The use of an ITIL aligned tool such as OTRS::ITSM only makes sense if processes, people, and products (IT services) are truly ITIL aligned.

Without the thoughtful tailoring of generic ITIL processes to meet the requirements of the specific business scenario, OTRS::ITSM will not achieve a discernible improvement of the key performance indicators of IT service management.

You should also be aware of the fact that successful ITIL implementation projects typically take up to a year and longer. Their scope and impact on the organization is not to be underestimated. However, we would like to mention that a neatly implemented ITIL aligned ITSM tool can help to save time and money, as the process support of the tool aids and accelerates the process of organizational realignment.

Note: The implementation of OTRS::ITSM is based on ITIL v4.

OTRS::ITSM supports the following features and processes, which are usually designed during the first phase of an ITIL implementation:

- Incident Management
- Problem Management
- Service Level Management
- Configuration Management Database

A more detailed description of use and adaptation of the system can be found in the following sections. Please note that the each OTRS::ITSM package can be installed independently and that their names correspond to their respective ITIL topics.

Note: The ITSM packages are installed into **OTRS** by the *Customer Solution Team*. In case of *On-Premise* systems, the customer can install the packages from the package manager, when the *Customer Solution Team* added the selected packages to the repository. To install a package, please contact the *Customer Solution Team* via support@otrs.com or in the OTRS Portal.

CHAPTER

TWO

CONFIGURATION MANAGEMENT

The configuration management database (CMDB) is not a database in the technical sense, but a conceptual IT model, which is indispensable for efficient IT service management. All IT components and inventories are managed in the CMDB. Configuration management exceeds asset management, often incorrectly used as a synonym, as it does not only document assets from a financial point of view, but captures information regarding the relationship between components, specifications, or their location. Thus IT support can quickly access information on the interdependence of IT services and the IT components (aka. configuration items or CIs) necessary for them.

This package provides a tool to import and export configuration items in the CSV format.

2.1 Administrator Interface

This chapter describes the new features that are available in the administrator interface after installation of the package.

2.1.1 Users, Groups & Roles

After installation of the package a new group is added to the system.

2.1.1.1 Groups

After installation of the package a new group is added to the system. The group management screen is available in the *Groups* module of the *Users, Groups & Roles* group.

New Group

After installation of the package the following group is added to the system:

itsm-configitem
: Group for accessing the *Asset Management* screens of the agent interface.

Note: The primary administrator user (root@localhost) is added to the group with permission *rw* by default.

See also:

To set the correct permissions for other users, check the following relations:

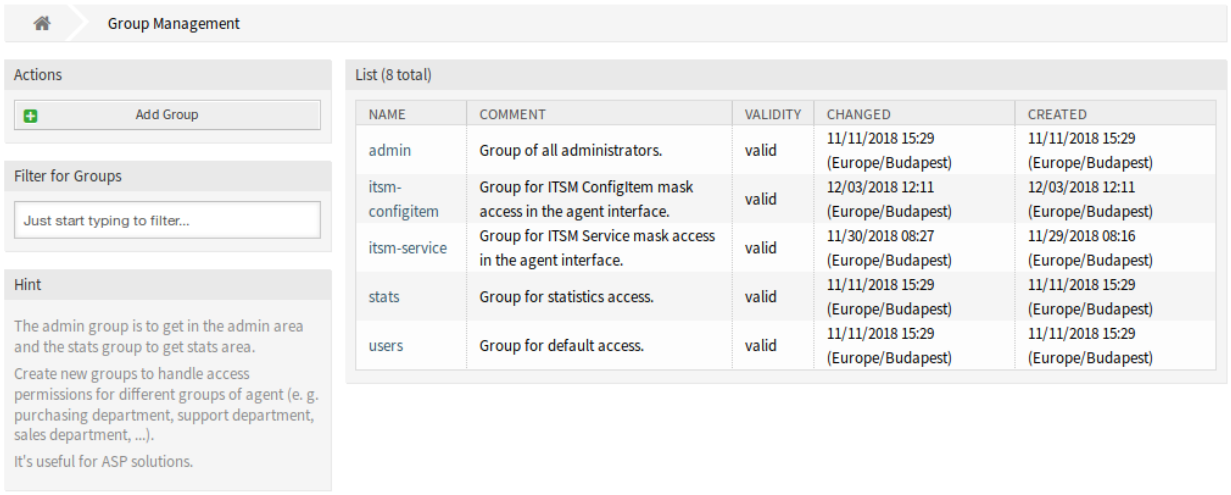

Fig. 1: Group Management Screen

- *Agents* *Groups*
- *Customers* *Groups*
- *Customer Users* *Groups*
- *Roles* *Groups*

2.1.2 Processes & Automation

After installation of the package two new scripts are added to the script task activity element of process management and some new operations are added to the generic interface.

2.1.2.1 Process Management

After installation of the package five new modules are added to script task activities and sequence flow actions of process management.

Process Modules

To see the new modules:

1. Go to the *Process Management* screen of the administrator interface.
2. Create a new process or select an existing process that contains a script task activity.
3. Click on the *Activities* item in the *Available Process Elements* widget in the left sidebar.
4. Create a new script task activity or edit an existing one.
5. Select one of the new scripts in the *Script* drop-down.
    - `ITSMConfigItemDataPull`
    - `ITSMConfigItemDataPush`

- `LinkWithITSMConfigItem`
- `TicketLinkITSMConfigItem`
- `TicketUpdateByLinkedCI`

6. Click on the *Save* button, if the *Configure* button is not visible next to the *Script* drop-down.
7. Click on the *Configure* button to add parameters (key-value pairs) for the script.

`ITSMConfigItemDataPull`

A module to fetch data from a linked ITSM configuration item.

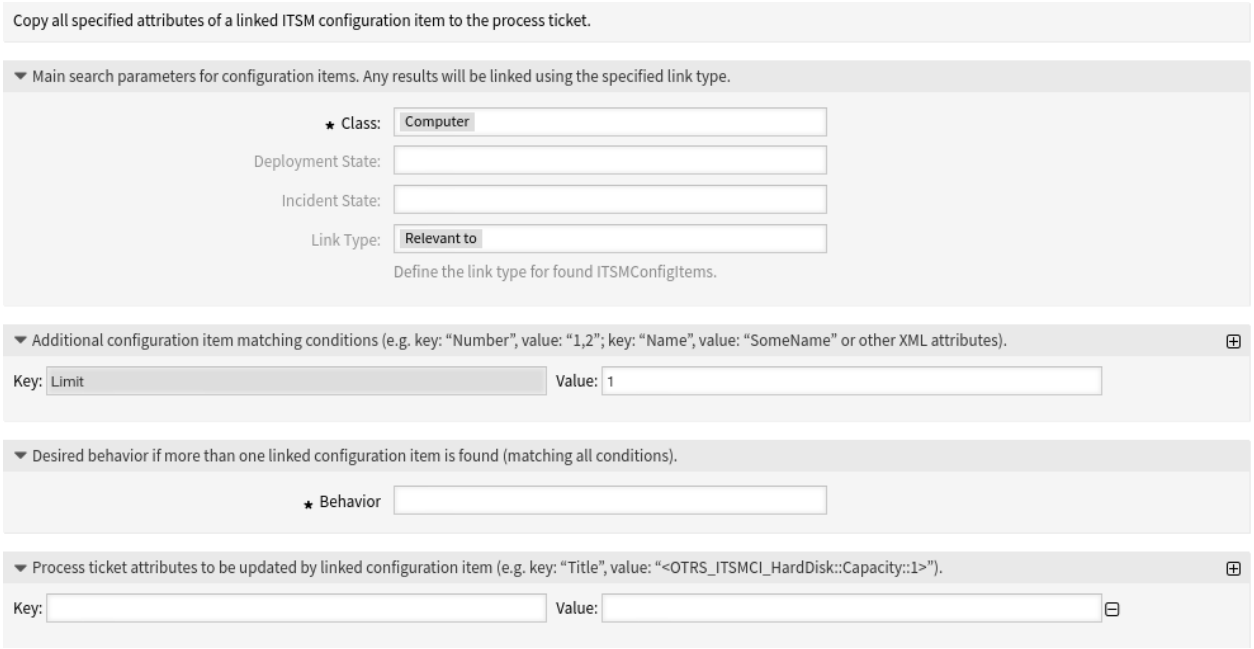

Fig. 2: Process Management Module ITSMConfigItemDataPull

Main search parameters section
The following parameters can be used for restrictions:

- Class *
- Deployment State
- Incident State
- Link Type

Additional configuration item condition section
This section is used to search for configuration items.

Filters can be added with key-value pairs. There is logical AND relation between the filters if more than one filter is added. Logical OR relation can be added by multiple values separated by `,`.

The key `Limit` limits the number of configuration items returned.

Desired behavior section
If more than one configuration items are found in the section above, the desired behavior can be defined here.

Possible values:

- Copy attributes from the configuration item that was found first
- Copy attributes from the configuration item that was found last
- Ignore configuration item, do not copy anything

Process ticket attributes section

With this module the process ticket attributes can be updated. The key is the attribute of the process ticket. The value can be a pre-defined text, an attribute from the linked configuration item in form of an OTRS tag or a concatenation of both. The `<OTRS_ITSMCI_*>` OTRS tag prefix can be used here.

Examples:

Key	Value
`Priority`	`5 very high`
`DynamicField_Capacity`	`<OTRS_ITSMCI_HardDisk::Capacity::1>`
`Title`	`From: <OTRS_ITSMCI_Name>`

See also:

See the *ITSMConfigItemDataPull* and the *ConfigItemSearch()* API reference.

`ITSMConfigItemDataPush`

A module to insert data to a linked ITSM configuration items.

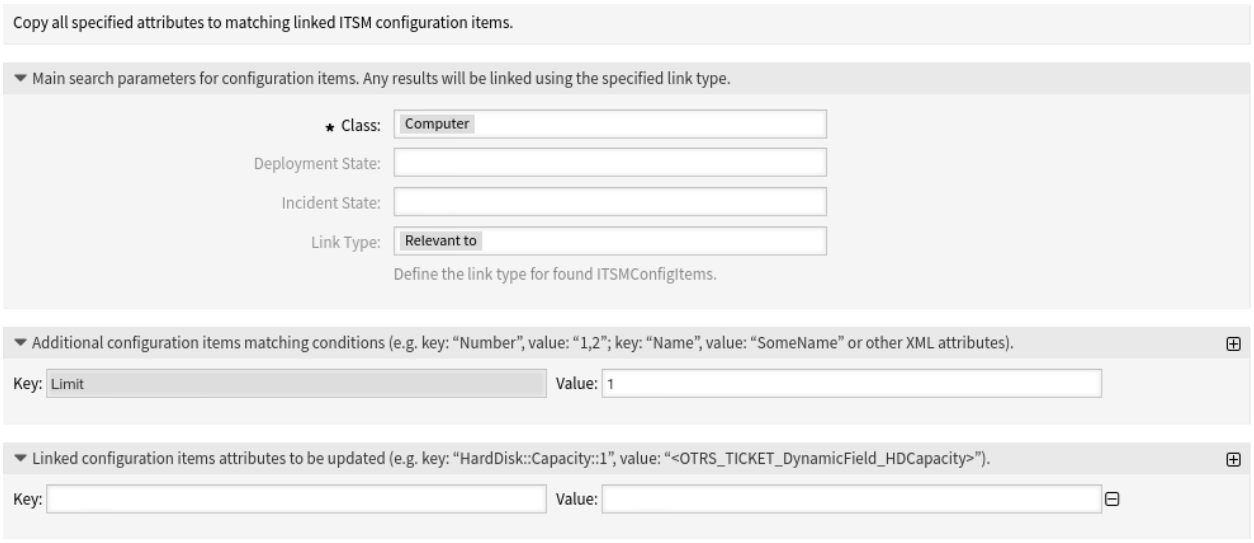

Fig. 3: Process Management Module ITSMConfigItemDataPush

Main search parameters section

The following parameters can be used for restrictions:

- Class *
- Deployment State
- Incident State

- Link Type

Additional configuration item condition section

This section is used to search for configuration items.

Filters can be added with key-value pairs. There is logical AND relation between the filters if more than one filter is added. Logical OR relation can be added by multiple values separated by `,`.

The key `Limit` limits the number of configuration items returned.

Linked configuration item attributes section

Here can be set the linked configuration item attributes to be updated. The key is the attribute of the linked configuration item. The value can be a pre-defined text, an attribute from the process ticket in form of an OTRS tag or a concatenation of both. The `<OTRS_TICKET_*>` OTRS tag prefix can be used here.

Examples:

Key	Value
`ConfigItemCreateTime-NewerDate`	`2021-10-20 12:23:34`
`HardDisk::Capacity::1`	`<OTRS_TICKET_DynamicField_HDCapacity>`
`Name`	`Process: <OTRS_TICKET_DynamicField_ProcessManagementProcessID`

See also:

See the *ITSMConfigItemDataPush* and the *ConfigItemSearch()* API reference.

LinkWithITSMConfigItem

A module to link ITSM configuration items.

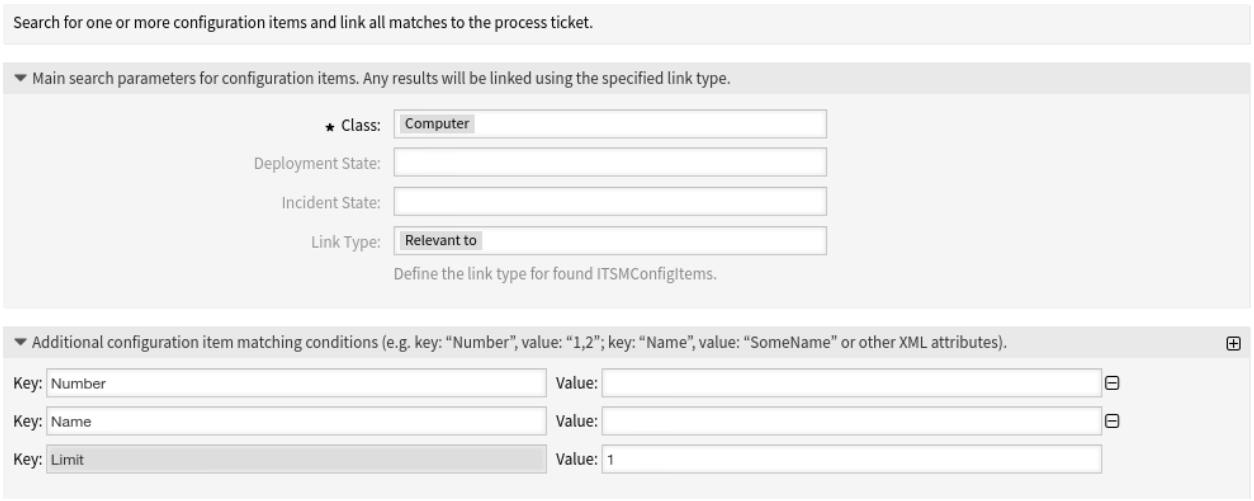

Fig. 4: Process Management Module LinkWithITSMConfigItem

Main search parameters section

The following parameters can be used for restrictions:

- Class *
- Deployment State

- Incident State
- Link Type

Additional configuration item condition section

This section is used to search for configuration items.

Filters can be added with key-value pairs. There is logical AND relation between the filters if more than one filter is added. Logical OR relation can be added by multiple values separated by `,`.

The key `Limit` limits the number of configuration items returned.

See also:

See the *LinkWithITSMConfigItem* and the *ConfigItemSearch()* API reference.

`TicketLinkITSMConfigItem`

A module to perform an extended configuration item search and link search results to ticket.

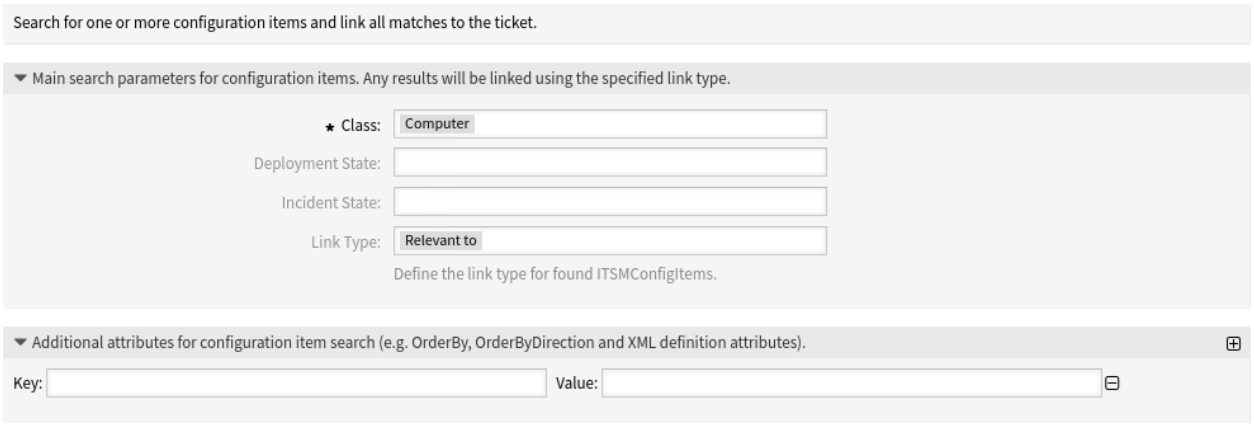

Fig. 5: Process Management Module TicketLinkITSMConfigItem

Main search parameters section

The following parameters can be used for restrictions:

- Class *
- Deployment State
- Incident State
- Link Type

Additional configuration item condition section

This section is used to search for configuration items.

Filters can be added with key-value pairs. There is logical AND relation between the filters if more than one filter is added. Logical OR relation can be added by multiple values separated by `,`.

You can sort the search results if you specify the key `OrderBy` and the attribute of a configuration item as value.

You can influence the sorting order with the key `OrderByDirection` and the values `Up` or `Down`.

If no sorting is specified, sorting is performed automatically in descending order based on the ID of the configuration items.

See also:

See the *TicketLinkITSMConfigItem* and the *ConfigItemSearch()* API reference.

`TicketUpdateByLinkedCI`

A module to set ticket attributes based on a linked configuration item.

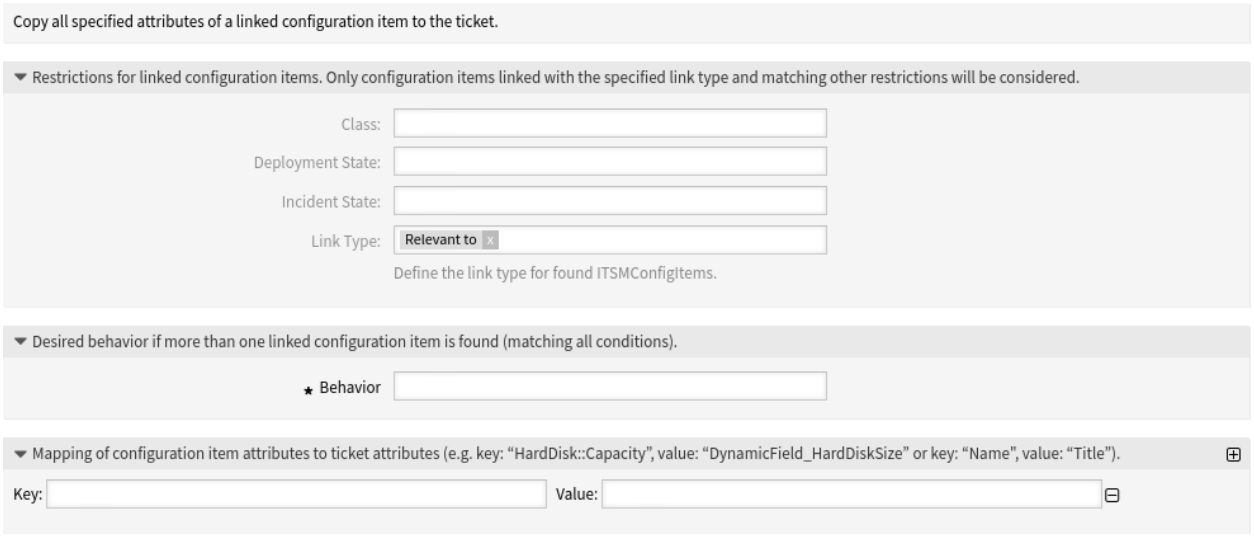

Fig. 6: Process Management Module TicketUpdateByLinkedCI

Main search parameters section
The following parameters can be used for restrictions:

- Class
- Deployment State
- Incident State
- Link Type

Desired behavior section
If more than one configuration items are found in the section above, the desired behavior can be defined here.

Possible values:

- Copy attributes from the configuration item that was found first
- Copy attributes from the configuration item that was found last
- Ignore configuration item, do not copy anything

Mapping configuration item to ticket section
With this module the configuration item attributes to ticket attributes can be mapped. The key is the attribute of the configuration item. The value is the attribute of the ticket.

Examples:

Key	Value
HardDisk::Capacity	DynamicField_HDCapacity
Name	Title

See also:

See the *TicketUpdateByLinkedCI* and the *ConfigItemSearch()* API reference.

API Reference

These API references are not available online, but they are included in this manual.

`ITSMConfigItemDataPull` API

```
NAME

Kernel::System::ProcessManagement::Modules::ITSMConfigItemDataPull - A module to␣
↪fetch data from a linked ITSM configuration item.

DESCRIPTION

All ITSMConfigItemDataPull functions.

PUBLIC INTERFACE

new()

Don't use the constructor directly, use the ObjectManager instead:

    my $ITSMConfigItemDataPullObject = $Kernel::OM->Get(
↪'Kernel::System::ProcessManagement::Modules::ITSMConfigItemDataPull');

Run()

Run Data

my $Success = $ITSMConfigItemDataPullObject->Run(
    UserID                          => 123,
    Ticket                          => \%Ticket,  # required
    ProcessEntityID                 => 'P123',
    ActivityEntityID                => 'A123',
    SequenceFlowEntityID            => 'T123',
    SequenceFlowActionEntityID      => 'TA123',
    Config => {

        ConfigITSMConfigItemSearch => {
            ClassIDs     => [9, 8, 7, 6],             # (optional)
            DeplStateIDs => [1, 2, 3, 4],             # (optional)
            InciStateIDs => [1, 2, 3, 4],             # (optional)
            LinkType     => 'RelevantTo::Source',     # (optional)
        },

        ConfigSearchKeyValueList => {
```

(continues on next page)

(continued from previous page)

```
            Name         => 'The Name',                # (optional)

            # configuration items with created time after ...
            ConfigItemCreateTimeNewerDate => '2006-01-09 00:00:01',  # (optional)
            # configuration items with created time before then ....
            ConfigItemCreateTimeOlderDate => '2006-01-19 23:59:59',  # (optional)

            # configuration items with changed time after ...
            ConfigItemChangeTimeNewerDate => '2006-01-09 00:00:01',  # (optional)
            # configuration items with changed time before then ....
            ConfigItemChangeTimeOlderDate => '2006-01-19 23:59:59',  # (optional)

            # XML attributes (defined by class)
            'ElementA::ElementB' => '%contentA%',
            'ElementA::ElementB' => '%contentC%,%contentD%,%contentE%,
        },

        UserID => 123,     # optional,to override the UserID from the logged user

        ConfigDropdown => {
            Behavior => 'NoCopy',       #  'NoCopy', 'CopyFirstLinked' or
↪'CopyLastLinked' only
        },

        UserID => 123,     # optional,to override the UserID from the logged user

        # Value set:
        #   * Key is the attribute of the linked ITSM configuration item where the␣
↪data will be set,
        #   * Value is the value is the value to be set, supporting smart tags <OTRS_
↪ITSMCI_*> from the resulting linked
        #       configuration item after match and behavior filters
        #
        #   Example:
        #   * To set process ticket title to be exactly the linked configuration item␣
↪first element of field someDefinitionField::Sub
        #     Title => '<OTRS_ITSMCI_someDefinitionField::Sub::1>',
        #     where 'Sub' is a sub field of field 'someDefinitionField'
        #   * To set process ticket title to be exactly the linked configuration item␣
↪second element of field someDefinitionField
        #     Title => '<OTRS_ITSMCI_someDefinitionField2::2>',
        #   * To set the process ticket dynamic field NameX to an static value (not␣
↪real pull):
        #     DynamicField_NameX => 'someValue',
    }
);

- `Ticket` contains the result of TicketGet including DynamicFields.
```

ITSMConfigItemDataPush API

```
NAME

Kernel::System::ProcessManagement::Modules::ITSMConfigItemDataPush - A module to␣
↪insert data to a linked ITSM configuration items.

DESCRIPTION

All ITSMConfigItemDataPush functions.

PUBLIC INTERFACE

new()

Don't use the constructor directly, use the ObjectManager instead:

    my $ITSMConfigItemDataPushObject = $Kernel::OM->Get(
↪'Kernel::System::ProcessManagement::Modules::ITSMConfigItemDataPush');

Run()

Run Data

my $Success = $ITSMConfigItemDataPullObject->Run(
    UserID                          => 123,
    Ticket                          => \%Ticket,  # required
    ProcessEntityID                 => 'P123',
    ActivityEntityID                => 'A123',
    SequenceFlowEntityID            => 'T123',
    SequenceFlowActionEntityID      => 'TA123',
    Config => {

        ConfigITSMConfigItemSearch => {
            ClassIDs     => [9, 8, 7, 6],             # (optional)
            DeplStateIDs => [1, 2, 3, 4],             # (optional)
            InciStateIDs => [1, 2, 3, 4],             # (optional)
            LinkType     => 'RelevantTo::Source',     # (optional)
        },

        ConfigSearchKeyValueList => {
            Name         => 'The Name',               # (optional)

            # configuration items with created time after ...
            ConfigItemCreateTimeNewerDate => '2006-01-09 00:00:01',  # (optional)
            # configuration items with created time before then ....
            ConfigItemCreateTimeOlderDate => '2006-01-19 23:59:59',  # (optional)

            # configuration items with changed time after ...
            ConfigItemChangeTimeNewerDate => '2006-01-09 00:00:01',  # (optional)
            # configuration items with changed time before then ....
            ConfigItemChangeTimeOlderDate => '2006-01-19 23:59:59',  # (optional)

            # XML attributes (defined by class)
            'ElementA::ElementB' => '%contentA%',
            'ElementA::ElementB' => '%contentC%,%contentD%,%contentE%,
        }.
```

(continues on next page)

(continued from previous page)

```
        UserID => 123,      # optional,to override the UserID from the logged user

        ConfigDropdown => {
            Behavior => 'NoCopy',       #  'NoCopy', 'CopyFirstLinked' or
↪'CopyLastLinked' only
        },

        UserID => 123,      # optional,to override the UserID from the logged user

        # Value set:
        #   * Key is the attribute of the linked ITSM configuration items where the␣
↪data will be pushed,
        #   * Value is the value is the value to be set, supporting smart tags <OTRS_
↪TICKET_*> from the current process ticket e.g.
        #
        #   Example:
        #   * To set linked ITSM configuration items first element of␣
↪someDefinitionField::Sub to be exactly the process ticket QueueID:
        #     'someDefinitionField::Sub::1' => '<OTRS_Ticket_QueueID>',
        #      Where 'Sub' is a sub field of 'someDefinitionField' field
        #   * To set linked ITSM configuration items second element of␣
↪someDefinitionField2 to be the concatenation of
        #     some text and the content of the process ticket dynamic field␣
↪ExternalField2:
        #     'someDefinitionField2::2' => 'Some text <OTRS_Ticket_DynamicField_
↪ExternalField2>',
        #   * To set linked ITSM configuration items last element of␣
↪someDefinitionField::Sub to be an static text:
        #     'someDefinitionField::Sub' => 'Some text',
    }
);

- `Ticket` contains the result of TicketGet including DynamicFields.
```

`LinkWithITSMConfigItem` API

```
NAME

Kernel::System::ProcessManagement::Modules::LinkWithITSMConfigItem - A module to link␣
↪ITSM configuration items.

DESCRIPTION

All LinkWithITSMConfigItem functions.

PUBLIC INTERFACE

new()

Don't use the constructor directly, use the ObjectManager instead:

    my $LinkWithITSMConfigItemObject = $Kernel::OM->Get(
↪'Kernel::System::ProcessManagement::Modules::LinkWithITSMConfigItem');
```

(continues on next page)

(continued from previous page)

```
Run()

Run Data

my $Success = $LinkWithITSMConfigItem->Run(
    UserID                          => 123,
    Ticket                          => \%Ticket,  # required
    ProcessEntityID                 => 'P123',
    ActivityEntityID                => 'A123',
    SequenceFlowEntityID            => 'T123',
    SequenceFlowActionEntityID      => 'TA123',
    Config => {

        ConfigITSMConfigItemSearch => {
            ClassIDs     => [9, 8, 7, 6],             # (optional)
            DeplStateIDs => [1, 2, 3, 4],             # (optional)
            InciStateIDs => [1, 2, 3, 4],             # (optional)
            LinkType     => 'RelevantTo::Source',     # (optional)
        },

        ConfigSearchKeyValueList => {
            Number       => 'The ConfigItem Number',  # (optional)
            Name         => 'The Name',               # (optional)

            # configuration items with created time after ...
            ConfigItemCreateTimeNewerDate => '2006-01-09 00:00:01',  # (optional)
            # configuration items with created time before then ....
            ConfigItemCreateTimeOlderDate => '2006-01-19 23:59:59',  # (optional)

            # configuration items with changed time after ...
            ConfigItemChangeTimeNewerDate => '2006-01-09 00:00:01',  # (optional)
            # configuration items with changed time before then ....
            ConfigItemChangeTimeOlderDate => '2006-01-19 23:59:59',  # (optional)

            # XML attributes (defined by class)
            'ElementA::ElementB' => '%contentA%',
            'ElementA::ElementB' => '%contentC%,%contentD%,%contentE%,
        }.

        UserID => 123,     # optional,to override the UserID from the logged user
    }
);

- `Ticket` contains the result of TicketGet including DynamicFields.
```

TicketLinkITSMConfigItem API

```
NAME

Kernel::System::ProcessManagement::Modules::TicketLinkITSMConfigItem - A module to␣
↪perform an extended configuration item search and link search results to ticket.

DESCRIPTION

All TicketLinkITSMConfigItem functions.

PUBLIC INTERFACE

new()

Don't use the constructor directly, use the ObjectManager instead:

    my $TicketLinkITSMConfigItemObject = $Kernel::OM->Get(
↪'Kernel::System::ProcessManagement::Modules::TicketLinkITSMConfigItem');

Run()

Run Data

my $Success = $TicketLinkITSMConfigItemObject->Run(
    UserID                          => 123,
    Ticket                          => \%Ticket,   # required
    ProcessEntityID                 => 'P123',
    ActivityEntityID                => 'A123',
    SequenceFlowEntityID            => 'T123',
    SequenceFlowActionEntityID      => 'TA123',
    Config => {
        UserID                      => 123,         # optional, to override the␣
↪UserID from the logged user
    }
);

- `Ticket` contains the result of TicketGet including DynamicFields.
- `Config` is the Config Hash stored in a Process::SequenceFlowAction's Config key.
```

TicketUpdateByLinkedCI API

```
NAME

Kernel::System::ProcessManagement::Modules::TicketUpdateByLinkedCI - A module to set␣
↪ticket attributes based on a linked configuration item.

DESCRIPTION

All TicketUpdateByLinkedCI functions.

PUBLIC INTERFACE

new()
```

(continues on next page)

(continued from previous page)

```
Don't use the constructor directly, use the ObjectManager instead:

my $TicketUpdateByLinkedCIObject = $Kernel::OM->Get(
↪'Kernel::System::ProcessManagement::Modules::TicketUpdateByLinkedCI');

Run()

Run Data

my $Success = $TicketUpdateByLinkedCIObject->Run(
    UserID                          => 123,
    Ticket                          => \%Ticket,                    # required
    ProcessEntityID                 => 'P123',
    ActivityEntityID                => 'A123',
    SequenceFlowEntityID            => 'T123',
    SequenceFlowActionEntityID      => 'TA123',
    Config  => {
        ConfigITSMConfigItemSearch => {
            ClassID                 => 123,                         # optional
            DeplStateIDs            => [123],                       # optional
            InciStateIDs            => [123],                       # optional
            LinkType                => 'someType::someDirection',   # optional
        },
        ConfigDropdown => {
            Behavior                => 'NoCopy',                    # 'CopyFirstLinked
↪', 'CopyLastLinked'
        },
        'someDefinitionField::Sub'  => 'DynamicField_someName',     # example optional
        'someDefinitionField2'      => 'someTicketAttribute',       # example optional
        UserID                      => 123,                         # optional, to␣
↪override the UserID from the logged user
    }
);

- `Ticket` contains the result of TicketGet including DynamicFields.
- `Config` is the Config Hash stored in a Process::SequenceFlowAction's Config key.
```

`ConfigItemCreate()` API

```
perform ConfigItemCreate Operation. This will return the created config item number.

 my $Result = $OperationObject->Run(
     Data => {
         UserLogin         => 'some agent login',            # UserLogin or␣
↪AccessToken is required
         AccessToken       => 'eyJhbGciOiJIUzI1NiJ9[...],'

         Password  => 'some password',                   # if UserLogin is sent then
                                                         #   Password is required
         ConfigItem => {
             Number       => '111',                      # optional
             Class        => 'Configuration Item Class',
             Name         => 'The Name',
             DeplState    => 'deployment state',
```

(continues on next page)

(continued from previous page)

```
            InciState    => 'incident state',
            CIXMLData    => $ArrayHashRef,              # it depends on the␣
↪Configuration Item class and definition

            Attachment => [
                {
                    Content     => 'content'            # base64 encoded
                    ContentType => 'some content type'
                    Filename    => 'some fine name'
                },
                # ...
            ],
            #or
            #Attachment => {
            #    Content     => 'content'
            #    ContentType => 'some content type'
            #    Filename    => 'some fine name'
            #},
        },
    },
);

$Result = {
    Success         => 1,                       # 0 or 1
    ErrorMessage    => '',                      # in case of error
    Data            => {                        # result data payload after Operation
        ConfigItemID => 123,                    # Configuration Item  ID number in␣
↪OTRS::ITSM (Service desk system)
        Number       => 2324454323322           # Configuration Item  Number in␣
↪OTRS::ITSM (Service desk system)
        Error => {                              # should not return errors
                ErrorCode    => 'ConfigItemCreate.ErrorCode'
                ErrorMessage => 'Error Description'
        },
    },
};
```

`ConfigItemDelete()` API

```
perform ConfigItemDelete Operation. This function is able to return
one or more ConfigItem entries in one call.

my $Result = $OperationObject->Run(
    Data => {
        UserLogin         => 'some agent login',            # UserLogin or␣
↪CustomerUserLogin or AccessToken is
                                                            # required
        CustomerUserLogin => 'some customer login',
        AccessToken       => 'eyJhbGciOiJIUzI1NiJ9[...]',

        Password          => 'some password',               # if UserLogin or␣
↪customerUserLogin is sent then
                                                            #   Password is required
        ConfigItemID      => '32,33',                       # required, could be␣
```

(continues on next page)

(continued from previous page)

```
↪coma separated IDs or an Array
    },
);

$Result = {
    Success         => 1,                           # 0 or 1
    ErrorMessage    => '',                          # in case of error
    Data            => {                            # result data payload after Operation
        ConfigItemID => [123, 456],          # Configuration Item IDs number in␣
↪OTRS::ITSM (Service desk system)
        Error => {                                  # should not return errors
                ErrorCode    => 'ConfigItemDelete.ErrorCode'
                ErrorMessage => 'Error Description'
        },
    },
};
```

`ConfigItemGet()` API

```
perform ConfigItemGet Operation. This function is able to return
one or more ConfigItem entries in one call.

my $Result = $OperationObject->Run(
    Data => {
        UserLogin         => 'some agent login',            # UserLogin or␣
↪AccessToken is
        AccessToken       => 'eyJhbGciOiJIUzI1NiJ9[...]',   # required
        Password          => 'some password',               # if UserLogin is sent␣
↪then Password is required
        ConfigItemID      => '32,33',                       # required, could be␣
↪coma separated IDs or an Array
        Attachments       => 1,                             # Optional, 1 as default.
↪ If it's set with the value 1,
                                                            # attachments for␣
↪articles will be included on ConfigItem data
    },
);

$Result = {
    Success      => 1,                                  # 0 or 1
    ErrorMessage => '',                                 # In case of an error
    Data         => {
        ConfigItem => [
            {

                Number             => '20101027000001',
                ConfigItemID       => 123,
                Name               => 'some name',
                Class              => 'some class',
                VersionID          => 123,
                LastVersionID      => 123,
                DefinitionID       => 123,
                InciState          => 'some incident state',
                InciStateType      => 'some incident state type',
```

(continues on next page)

(continued from previous page)

```
                DeplState          => 'some deployment state',
                DeplStateType      => 'some deployment state type',
                CurInciState       => 'some incident state',
                CurInciStateType   => 'some incident state type',
                CurDeplState       => 'some deployment state',
                CurDeplStateType   => 'some deployment state type',
                CreateTime         => '2010-10-27 20:15:00'
                CreateBy           => 123,
                CIXMLData          => $XMLDataHashRef,

                Attachment => [
                    {
                        Content            => "xxxx",     # actual attachment␣
↪contents, base64 enconded
                        ContentType        => "application/pdf",
                        Filename           => "StdAttachment-Test1.pdf",
                        Filesize           => "4.6 KBytes",
                        Preferences        => $PreferencesHashRef,
                    },
                    {
                        # . . .
                    },
                ],
            },
            {
                # . . .
            },
        ],
    },
};
```

`ConfigItemSearch()` API

```
ConfigItemSearch()

return a configuration item list as an array reference

my $ConfigItemIDs = $ConfigItemObject->ConfigItemSearch(
    Number       => 'The ConfigItem Number',  # (optional)
    ClassIDs     => [9, 8, 7, 6],             # (optional)
    DeplStateIDs => [1, 2, 3, 4],             # (optional)
    InciStateIDs => [1, 2, 3, 4],             # (optional)
    CreateBy     => [1, 2, 3],                # (optional)
    ChangeBy     => [3, 2, 1],                # (optional)

    # configuration items with created time after ...
    ConfigItemCreateTimeNewerDate => '2006-01-09 00:00:01',  # (optional)
    # configuration items with created time before then ....
    ConfigItemCreateTimeOlderDate => '2006-01-19 23:59:59',  # (optional)

    # configuration items with changed time after ...
    ConfigItemChangeTimeNewerDate => '2006-01-09 00:00:01',  # (optional)
    # configuration items with changed time before then ....
    ConfigItemChangeTimeOlderDate => '2006-01-19 23:59:59',  # (optional)
```

(continues on next page)

(continued from previous page)

```
    OrderBy => [ 'ConfigItemID', 'Number' ],                  # (optional)
    # default: [ 'ConfigItemID' ]
    # (ConfigItemID, Number, ClassID, DeplStateID, InciStateID,
    # CreateTime, CreateBy, ChangeTime, ChangeBy)

    # Additional information for OrderBy:
    # The OrderByDirection can be specified for each OrderBy attribute.
    # The pairing is made by the array indices.

    OrderByDirection => [ 'Down', 'Up' ],                     # (optional)
    # default: [ 'Down' ]
    # (Down | Up)

    Limit          => 122,  # (optional)
    UsingWildcards => 0,    # (optional) default 1
);
```

```
ConfigItemSearchExtended()

return a configuration item list as an array reference

my $ConfigItemIDs = $ConfigItemObject->ConfigItemSearchExtended(
    Number       => 'The ConfigItem Number',  # (optional)
    Name         => 'The Name',               # (optional)
    ClassIDs     => [9, 8, 7, 6],             # (optional)
    DeplStateIDs => [1, 2, 3, 4],             # (optional)
    InciStateIDs => [1, 2, 3, 4],             # (optional)

    # configuration items with created time after ...
    ConfigItemCreateTimeNewerDate => '2006-01-09 00:00:01',  # (optional)
    # configuration items with created time before then ....
    ConfigItemCreateTimeOlderDate => '2006-01-19 23:59:59',  # (optional)

    # configuration items with changed time after ...
    ConfigItemChangeTimeNewerDate => '2006-01-09 00:00:01',  # (optional)
    # configuration items with changed time before then ....
    ConfigItemChangeTimeOlderDate => '2006-01-19 23:59:59',  # (optional)

    What => [                                                # (optional)
        # each array element is a and condition
        {
            # or condition in hash
            "[%]{'ElementA'}[%]{'ElementB'}[%]{'Content'}" => '%contentA%',
            "[%]{'ElementA'}[%]{'ElementC'}[%]{'Content'}" => '%contentA%',
        },
        {
            "[%]{'ElementA'}[%]{'ElementB'}[%]{'Content'}" => '%contentB%',
            "[%]{'ElementA'}[%]{'ElementC'}[%]{'Content'}" => '%contentB%',
        },
        {
            # use array reference if different content with same key was searched
            "[%]{'ElementA'}[%]{'ElementB'}[%]{'Content'}" => ['%contentC%', '
↪%contentD%', '%contentE%'],
            "[%]{'ElementA'}[%]{'ElementC'}[%]{'Content'}" => ['%contentC%', '
↪%contentD%', '%contentE%'],
```

(continues on next page)

(continued from previous page)

```
        },
    ],

    PreviousVersionSearch => 1,  # (optional) default 0 (0|1)

    OrderBy => [ 'ConfigItemID', 'Number' ],                  # (optional)
    # default: [ 'ConfigItemID' ]
    # (ConfigItemID, Number, Name, ClassID, DeplStateID, InciStateID,
    # CreateTime, CreateBy, ChangeTime, ChangeBy)

    # Additional information for OrderBy:
    # The OrderByDirection can be specified for each OrderBy attribute.
    # The pairing is made by the array indices.

    OrderByDirection => [ 'Down', 'Up' ],                     # (optional)
    # default: [ 'Down' ]
    # (Down | Up)

    Limit          => 122,  # (optional)
    UsingWildcards => 0,    # (optional) default 1
);
```

`ConfigItemUpdate()` API

```
perform ConfigItemUpdate Operation. This will return the updated configuration item␣
↪number.

 my $Result = $OperationObject->Run(
     Data => {
         UserLogin => 'some agent login',               # UserLogin or AccessToken is
         AccessToken => 123,                            # required

         Password  => 'some password',                  # if UserLogin is sent then␣
↪Password is required

         ReplaceExistingData => 0,                      # optional, 0 or 1, default 0
                                                        # this will replace the␣
↪existing XML data and attachments
         ConfigItemID => 123,

         ConfigItem   => {
             Class     => 'Configuration Item Class',
             Name      => 'The Name',
             DeplState => 'deployment state',
             InciState => 'incident state',
             CIXMLData => $ArrayHashRef,                # it depends on the␣
↪Configuration Item class and definition

             Attachment => [
                 {
                     Content     => 'content'           # base64 encoded
                     ContentType => 'some content type'
                     Filename    => 'some fine name'
                 },
```

(continues on next page)

(continued from previous page)

```
                # ...
            ],
            # or
            #Attachment => {
            #   Content     => 'content'
            #   ContentType => 'some content type'
            #   Filename    => 'some fine name'
            #},
        },
    },
);

$Result = {
    Success         => 1,                       # 0 or 1
    ErrorMessage    => '',                      # in case of error
    Data            => {                        # result data payload after Operation
        ConfigItemID => 123,                    # Configuration Item  ID number in␣
↪OTRS::ITSM (Service desk system)
        Number       => 2324454323322           # Configuration Item  Number in␣
↪OTRS::ITSM (Service desk system)
        Error => {                              # should not return errors
                ErrorCode    => 'ConfigItemUpdate.ErrorCode'
                ErrorMessage => 'Error Description'
        },
    },
};
```

2.1.2.2 Web Services

This package adds some new operations for creating, changing, retrieving, deleting and searching configuration items via generic interface. The following operations are available:

- `ConfigItemCreate()`
- `ConfigItemDelete()`
- `ConfigItemGet()`
- `ConfigItemSearch()`
- `ConfigItemUpdate()`

See also:

For more information please take a look at the WSDL file located in `development/webservices/GenericConfigItemConnectorSOAP.wsdl` of your instance.

New Operations

These new operations are available in the *Web Services* module of the *Processes & Automation* group:

- `ConfigItem::ConfigItemCreate`
- `ConfigItem::ConfigItemDelete`
- `ConfigItem::ConfigItemGet`
- `ConfigItem::ConfigItemSearch`
- `ConfigItem::ConfigItemUpdate`

To use these operations:

1. Add or edit a web service.
2. Select a *Network transport* in the *OTRS as provider* widget and save the web service.
3. The new operations are available in the *Add Operation* field of the *OTRS as provider* widget.

See also:

Check the API references in *Process Management* chapter for more information.

Examples for Usage

The following examples give a quick look about how to use the API for basic actions.

1. Create configuration item
   - URL: `/api/agent/config-item/create`
   - Method: POST
   - Payload:

```
{
    "ConfigItem": {
        "Class": "Computer",
        "Name": "test name for new config item",
        "DeplState": "Production",
        "InciState": "Operational",
        "CIXMLData": {
            "Seriennummer": "SNR1"
            "NIC": {
                 "NIC": "test",
                 "IPoverDHCP": "Yes"
            }
        }
    }
}
```

2. Update configuration item
   - URL: `/api/agent/config-item/4/update` where `4` is the ID of the configuration item to be updated
   - Method: POST
   - Payload:

```
{
    "ConfigItemID": "4",
    "ConfigItem": {
        "Class": "Computer",
        "Name": "test name for new config item",
        "DeplState": "Production",
        "InciState": "Operational",
        "CIXMLData": {
            "Seriennummer": "SNR2"
            "NIC": {
                 "NIC": "test",
                 "IPoverDHCP": "Yes"
            }
        }
    }
}
```

Note: The `Class` is required to be transmitted but will not affect the configuration item when updating. If you update a configuration item in the class `Location` and transmit the class `Computer` the configuration item will stay in the class `Location`.

3. Get configuration item
   - URL: `/api/agent/config-item/4` where `4` is the ID of the configuration item to be fetched
   - Method: GET
4. List configuration items
   - URL: `/api/agent/config-item/list`
   - Method: POST

2.1.3 Administration

After installation of the package some new classes will be available in the *General Catalog* and a new module will be available in the administrator interface.

2.1.3.1 General Catalog

ITSM configuration management adds some new classes to the *General Catalog*. The general catalog management screen is available in the *General Catalog* module of the *Administration* group.

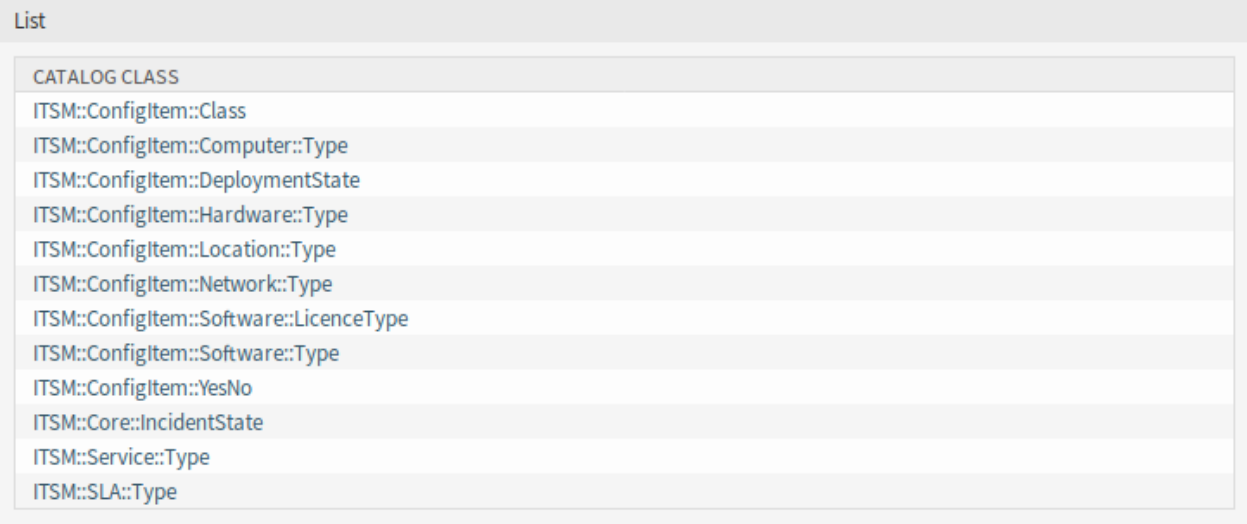

Fig. 7: General Catalog Class List Screen

New Classes

ITSM::ConfigItem::Class
A class for configuration item classes.

See also:

The class definition of configuration item classes can be managed in the *Configuration Items* module of the *CMDB Settings* group.

ITSM::ConfigItem::Computer::Type
A class for computer types, that can be selected in *Configuration Items* when adding or editing configuration items of type computer.

ITSM::ConfigItem::DeploymentState
A class for deployment states, that can be selected in *Configuration Items* when adding or editing configuration items.

ITSM::ConfigItem::Hardware::Type
A class for hardware types, that can be selected in *Configuration Items* when adding or editing configuration items of type hardware.

ITSM::ConfigItem::Location::Type
A class for location types, that can be selected in *Configuration Items* when adding or editing configuration items of type location.

ITSM::ConfigItem::Network::Type
A class for network types, that can be selected in *Configuration Items* when adding or editing configuration items of type network.

ITSM::ConfigItem::Software::LicenceType
A class for software license types, that can be selected in *Configuration Items* when adding or editing configuration items of type software.

ITSM::ConfigItem::Software::Type
A class for software types, that can be selected in *Configuration Items* when adding or editing configuration items of type software.

ITSM::ConfigItem::YesNo
This class contains the items *Yes* and *No*.

2.1.3.2 Import and Export

Use this screen to create import and export templates. The import/export template management screen is available in the *Import and Export* module of the *Administration* group.

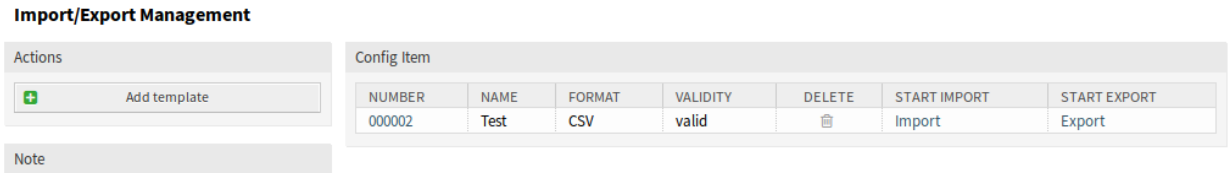

Fig. 8: Import/Export Template Management Screen

Manage Import/Export Templates

To create a new template:

1. Click on the *Add Template* button in the left sidebar.
2. Fill in the required fields in all steps.
3. Click on the *Finish* button.

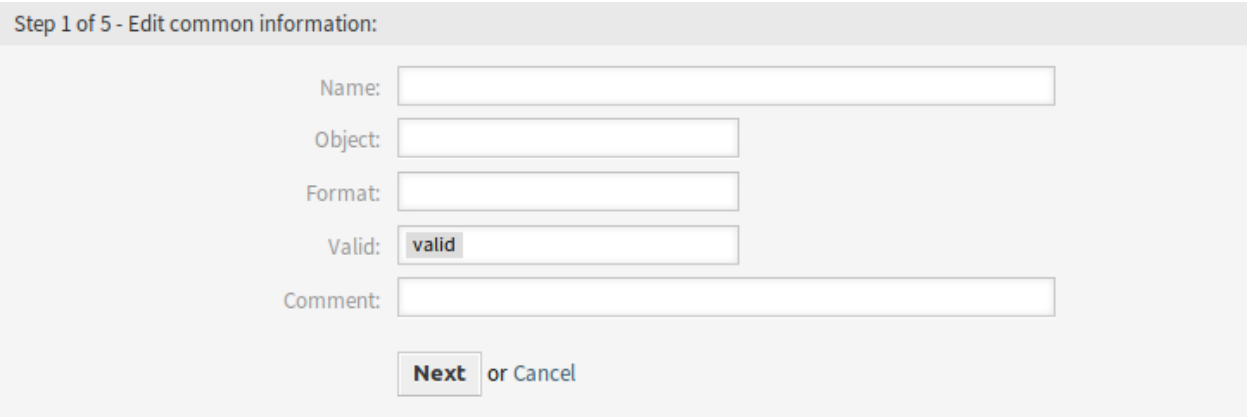

Fig. 9: Create New Import/Export Template Screen

To edit a template:

1. Click on a template in the list of templates.
2. Modify the fields in all steps.
3. Click on the *Finish* button.

To delete a template:

1. Click on the trash icon in the list of templates.
2. Click on the *Confirm* button.

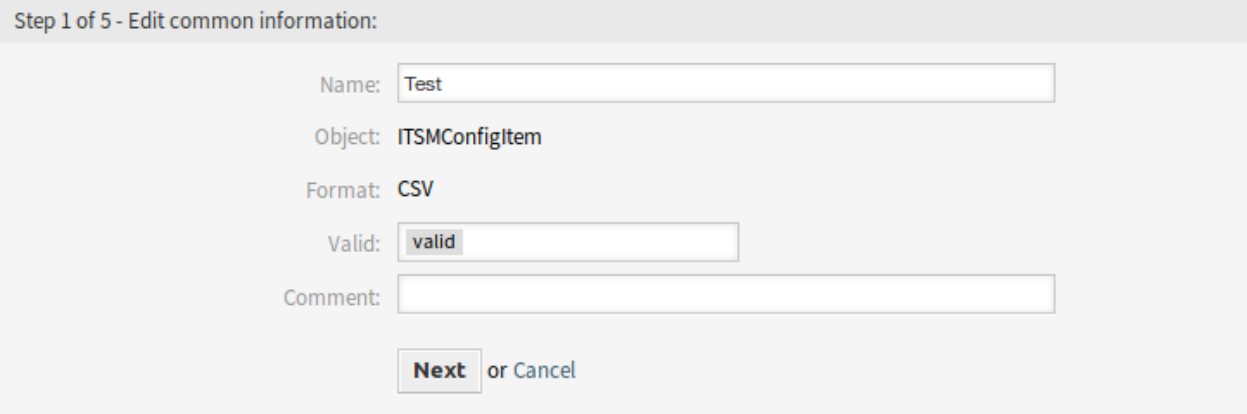

Fig. 10: Edit Import/Export Template Screen

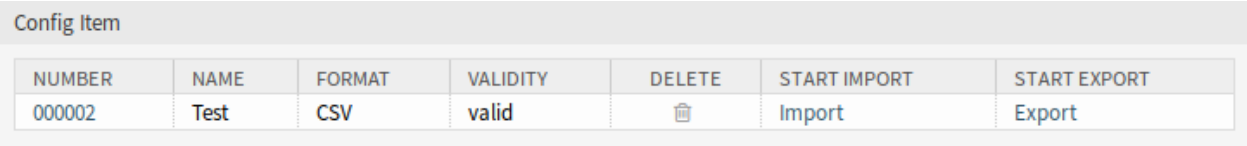

Fig. 11: Delete Import/Export Template Screen

To import data based on a template:

1. Click on the *Import* link in the list of templates.
2. Click on the *Browse*··· button and select a CSV file.
3. Click on the *Start Import* button.

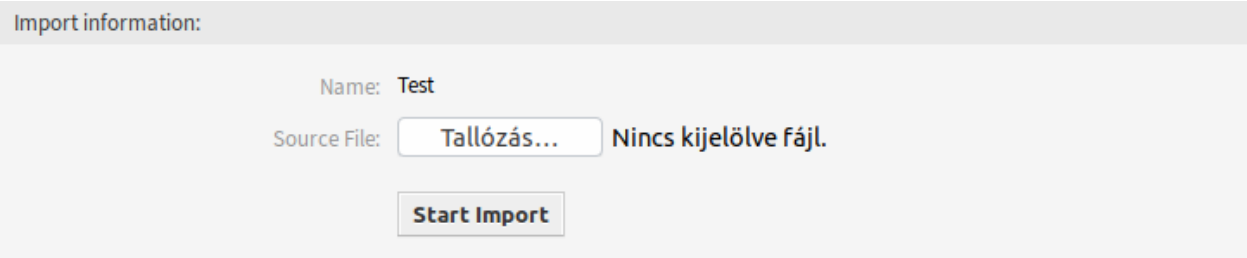

Fig. 12: Import Data Screen

To export data based on a template:

1. Click on the *Export* link in the list of templates.
2. Choose a location in your computer to save the `Export.csv` file.

Import/Export Template Settings

The following settings are available when adding this resource. The fields marked with an asterisk are mandatory.

Note: Import/Export package is meant to be independent. This means, that the following settings can be different if no configuration items will be imported or exported.

Edit Common Information

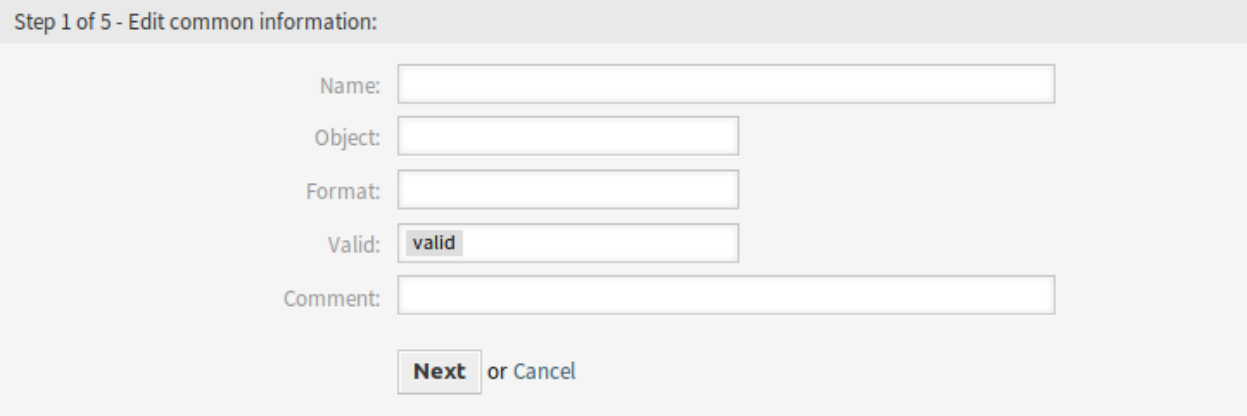

Fig. 13: Edit Common Information Screen

Name *
: The name of this resource. Any type of characters can be entered to this field including uppercase letters and spaces. The name will be displayed in the overview table.

Object *
: Select the object type you want to import to or export from.

Format *
: Select the import and export format.

Validity *
: Set the validity of this resource. Each resource can be used in OTRS only, if this field is set to *valid*. Setting this field to *invalid* or *invalid-temporarily* will disable the use of the resource.

Comment
: Add additional information to this resource. It is recommended to always fill this field as a description of the resource with a full sentence for better clarity.

Edit Object Information

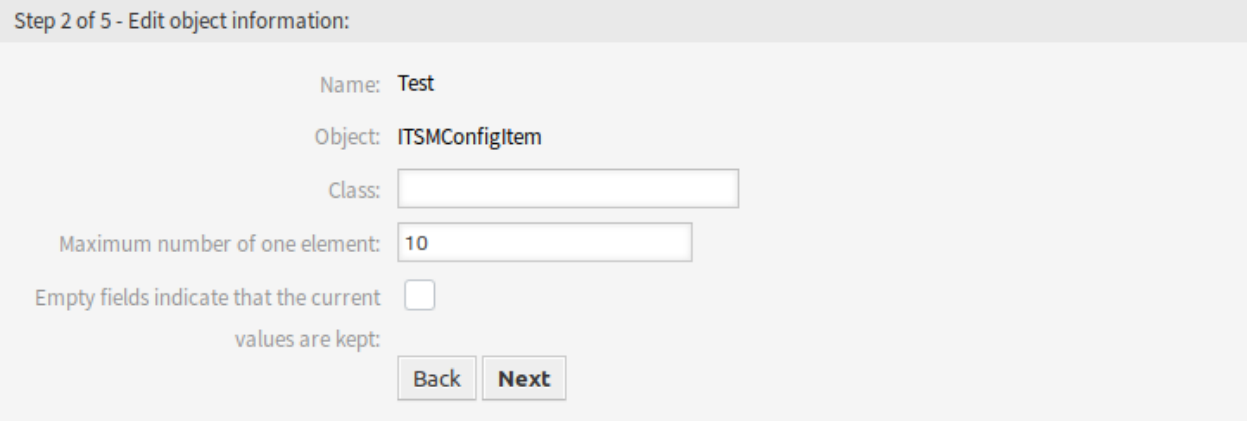

Fig. 14: Edit Object Information Screen

Name
This is a read only field from the previous step. Use the *Back* button to edit it.

Object
This is a read only field from the previous step. Use the *Back* button to edit it.

Class *
Select the class that is needed to be affected by the import and export.

Maximum number of one element *
Specify the maximum number of columns per array attribute that can be mapped when mapping the import from or export to the CSV file.

Empty fields indicate that the current values are kept
Select this checkbox if the empty field should keep the data in OTRS. Otherwise the data will be overwritten with blank value.

Edit Format Information

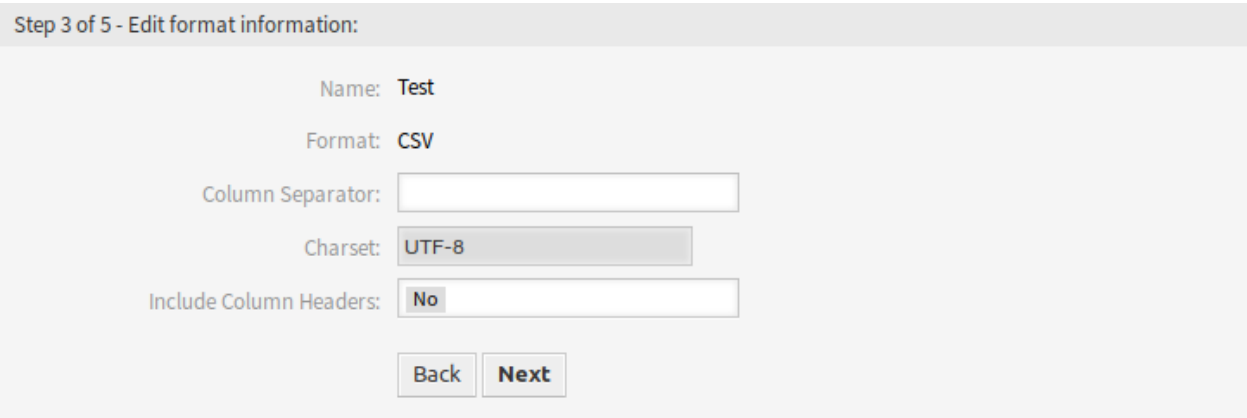

Fig. 15: Edit Format Information Screen

Name
This is a read only field from the previous step. Use the *Back* button to edit it.

Format
This is a read only field from the previous step. Use the *Back* button to edit it.

Column Separator *
Select a column separator for CSV file.

Charset
Select a character encoding for the CSV file.

Include Column Headers
Specify if column headers should be included or not.

Edit Mapping Information

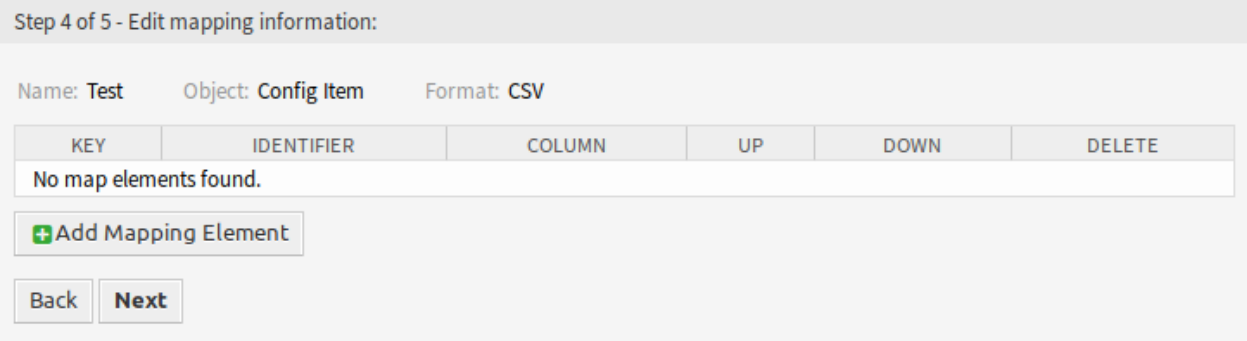

Fig. 16: Edit Mapping Information Screen

Click on the *Add Mapping Element* button to add element from the class. You can also specify if this element is an identifier. The order of the elements is sortable.

Edit Search Information

Template Name
This is a read only field from the previous step. Use the *Back* button to edit it.

Restrict export per search
You can add search term for each attribute of the selected class to restrict the import and export functions. The possible fields are listed below this field.

Note: The other fields come from the back end driver, and can be different depending on the used object to be imported or exported.

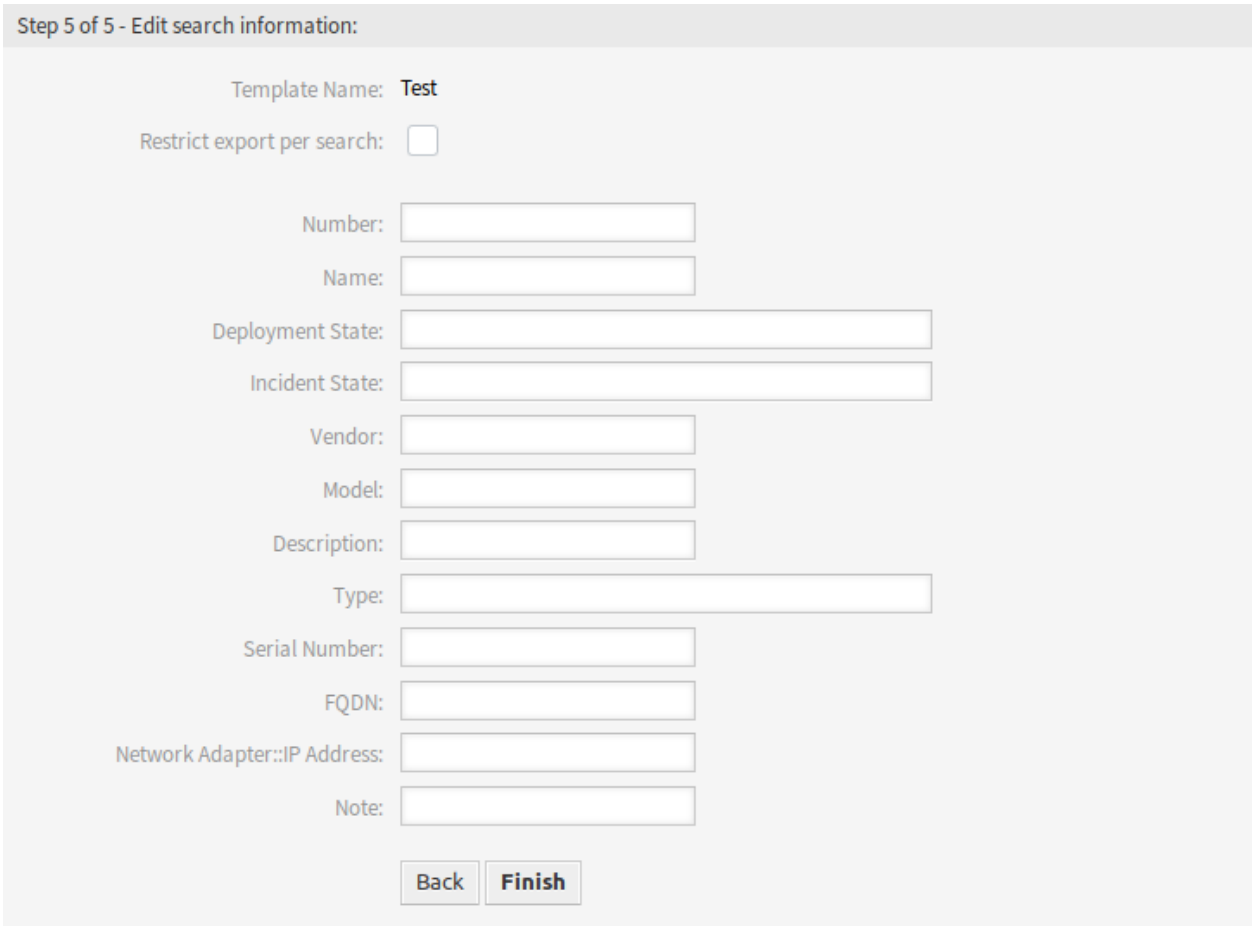

Fig. 17: Edit Search Information Screen

2.1.3.3 System Configuration

Displaying Configuration Item Class Specific Columns

Configuration item class specific columns (i.e. the capacity of a hard disc) are not shown in the configuration overview list and in the configuration item organizer item list by default.

In order to display configuration item field values as table columns, the YAML configuration of the lists needs to be extended.

The following example shows how to add the field `Computer::HardDisk::1` and `Computer::HardDisk::1::Capacity::1` of the class `Computer` to the configuration item overview list:

1. Search for the setting `AgentFrontend::ConfigItemList###DefaultConfig`.
2. Add the following to the YAML configuration:

```
Columns:
  Computer::HardDisk::1:
    IsVisible: 2
  Computer::HardDisk::1::Capacity::1:
    IsVisible: 2
```

Fig. 18: Example YAML Configuration of the Configuration Item Overview

3. Deploy the modified configuration.

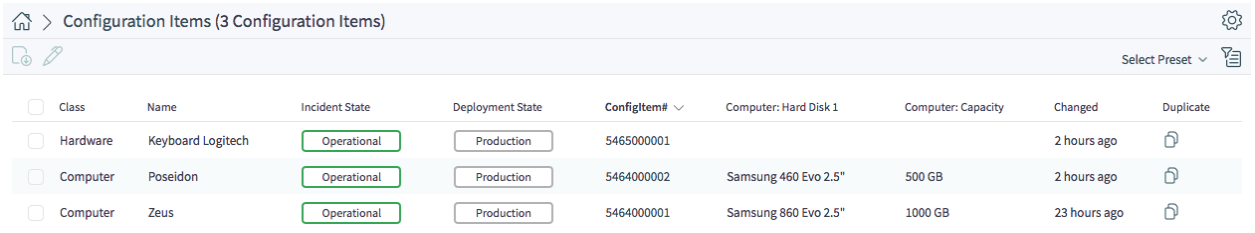

Fig. 19: Configuration Item Overview with Class Specific Columns

Applying Configuration Item Class Specific Filters

The following example shows how to apply the filter for the `Computer::Model` field of the class `Computer` in the configuration item overview list.

First you need to make sure, that the relevant class filter is also applied, otherwise all class-specific filters will be simply ignored.

This can be done via the `ClassIDs` filter, which takes the ID of the class as the value. Please follow the steps below to get the ID of the class and apply the filters.

1. Go to the *Configuration Items* management screen in the administrator interface.
2. Click on the relevant class in the list.

   The `ClassID` is now shown in the URL, for example in this case the ID is `22`:

   ```
   ``otrs/index.pl/?Action=AdminITSMConfigItem;Subaction=DefinitionList;ClassID=22``
   ```

3. Search for the setting `AgentFrontend::ConfigItemList###DefaultConfig`.
4. Set both the filter for the class to the value determined in step 2 and the field filter to the desired value:

   ```
   ActiveFilters:
     ClassIDs:
       Value:
       - 22
     Computer::Model:
       Value: ModelA
   ```

5. Deploy the modified configuration.

Note: The fields that can be filtered need to have `Searchable: 1` set in their class definitions. See *here* for more information.

Applying Configuration Item Filters for All Classes

The following example shows how to apply the filters for the common `Owner` and `CustomerID` fields which are used in all classes in the configuration item overview list.

1. Search for the setting `AgentFrontend::ConfigItemList###DefaultConfig`.
2. Add the following to the YAML configuration:

   ```
   AvailableSearchInAllClassesFilters:
   - Owner
   - CustomerID
   ```

3. Deploy the modified configuration.

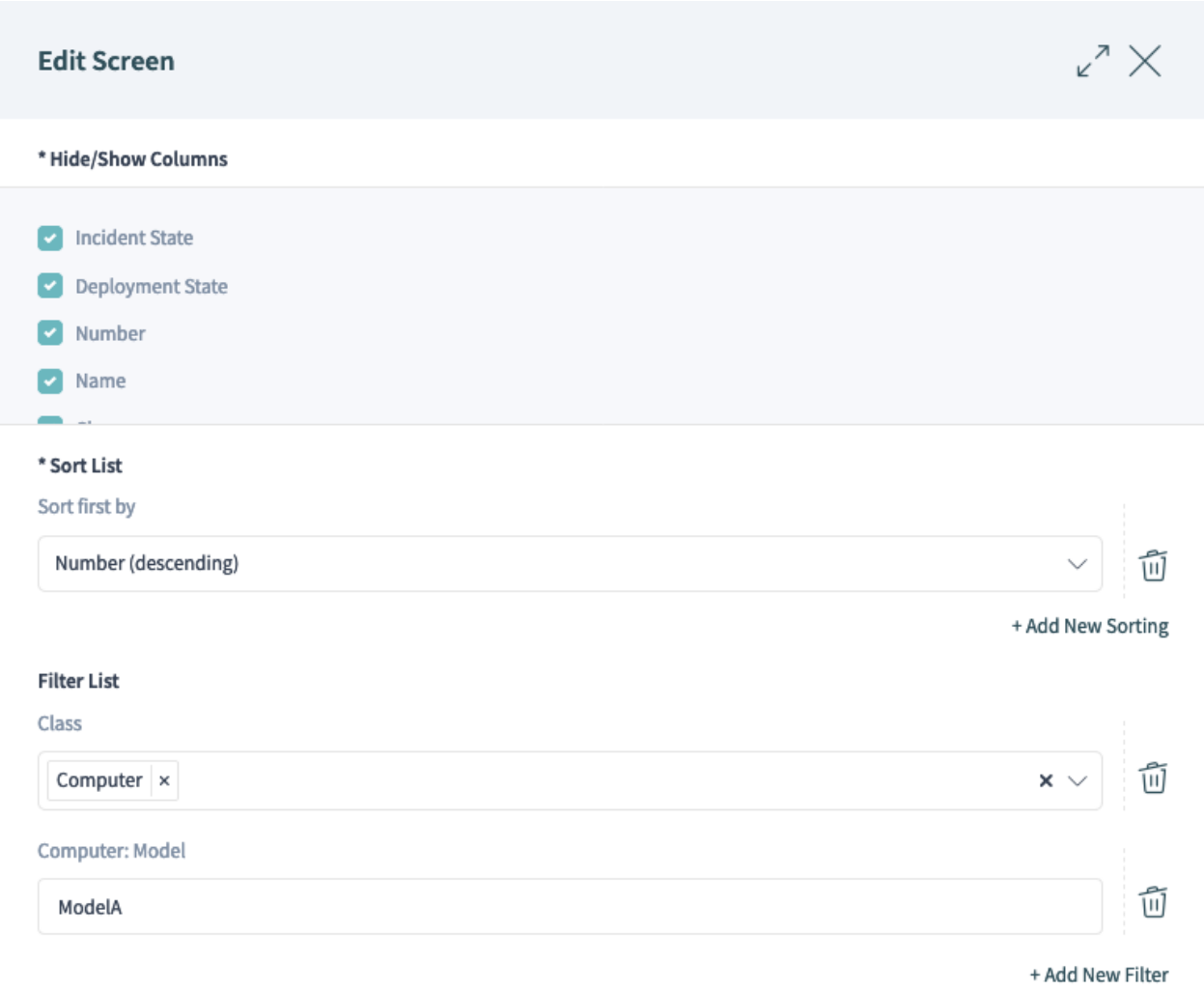

Fig. 20: Configuration Item Overview with Class Specific Filter

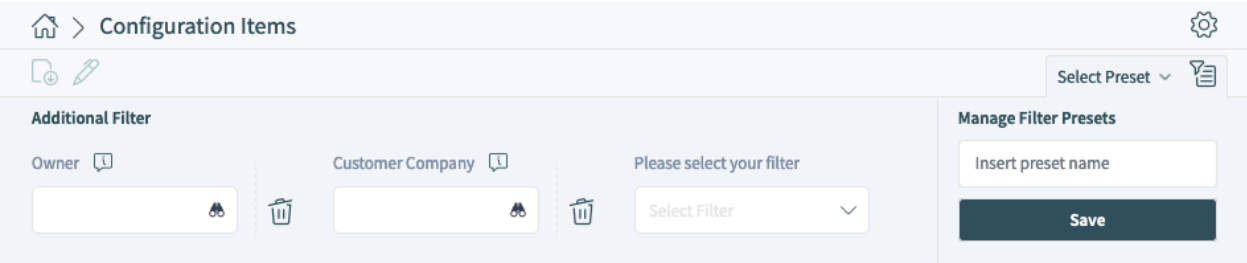

Fig. 21: Configuration Item Overview with All Class Filters

2.1.4 CMDB Settings

After installation of the package a new group *CMDB Settings* will be available with a new module in the administrator interface.

2.1.4.1 Configuration Items

Use this screen to manage class definition of configuration item classes. The configuration item class management screen is available in the *Configuration Items* module of the *CMDB Settings* group.

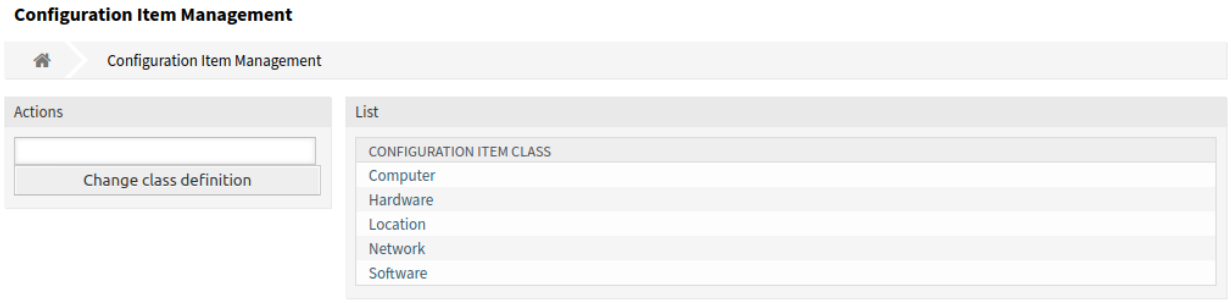

Fig. 22: Configuration Item Management Screen

To add or edit the class definition of a configuration item class:

1. Select a class from the drop-down menu in the left sidebar.
2. Click on the *Change class definition* button.
3. Add or edit the class definition in YAML format.
4. Click on the *Save* or *Save and finish* button.

To see the class definition of a configuration item class:

1. Click on a class name in the list of classes.
2. Select a version by clicking on a class name in the list of class versions.

See also:

New configuration item classes can be added in General Catalog module in the administrator interface.

Class Definition Types and Form Elements

Multiple input field types can be used when defining a class. These input field types are used to generate the edit form for creating new or editing already existing configuration items.

The following block is an example of a form field called *Operating System*.

```
---
- Key: OperatingSystem
  Name: Operating System
  Input:
    Type: Text
```

(continues on next page)

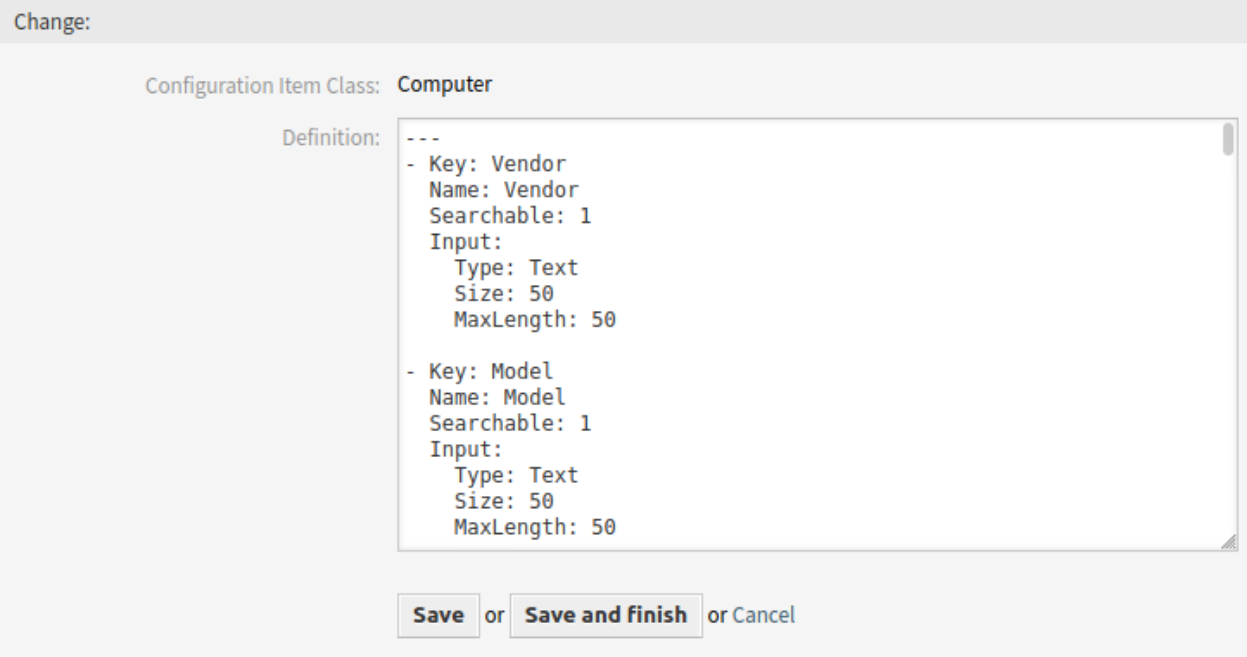

Fig. 23: Edit Configuration Item Class Definition Screen

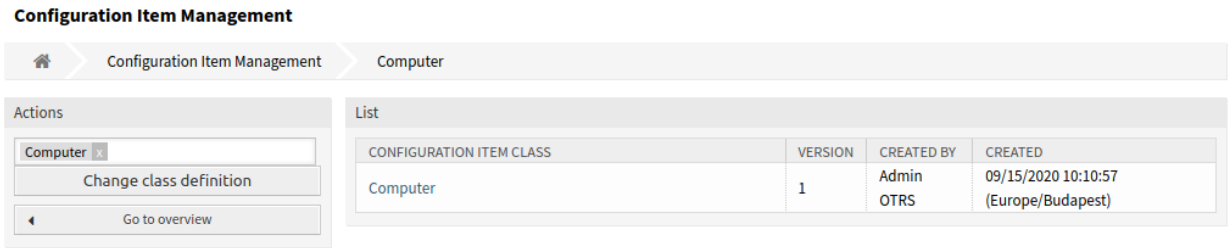

Fig. 24: Configuration Item Class Versions Screen

(continued from previous page)

```
    Size: 50
    MaxLength: 100
```

The following settings are available when adding or editing this resource. The fields marked with an asterisk are mandatory.

Key *
Must be unique and only accept alphabetic and numeric characters. If this is changed, data will not be readable from old definitions.

Name *
The label of the field in the form. Any type of characters can be entered to this field including uppercase letters and spaces.

Note: It is recommended to always use English words for names.

See also:

Names can be translated into other languages with custom language files. For more information, see the Custom Language File chapter in the administration manual.

Searchable
Defines whether the field is searchable or not. Possible values are *0* or *1*.

Input *
Initiates the definition of the input field. An input field can contain the following attributes:

Type *
Defines the type of the element. Must be placed indented as a logical block. Possible values are:

- `Customer`: A drop-down list for select a customer user from the database back end. The field can be used with wildcards (*).
- `CustomerCompany`: A drop-down list for select a customer from the database back end.
- `Date`: A field for select a date.
- `DateTime`: A field for select date and time.
- `Dummy`: This field is used to give the other elements a structure. It has usually `Sub` structures.
- `GeneralCatalog`: A drop-down list for select a general catalog class. The general catalog class must be defined before use it as input type. The items of the general catalog class will be the options of the drop-down list.
- `Integer`: A drop-down list with integer numbers.
- `Text`: A single text field.
- `TextArea`: A text field with multiple rows.

Required
Defines whether the field is mandatory or not. Possible values are *0* or *1*.

Size
Defines the size of the text field. The value must be a positive integer.

MaxLength
Defines the maximum amount of characters that can be entered in the text field. The value must be a positive integer.

RegEx
A regular expression to restrict the possible values of the text field.

RegExErrorMessage
The displayed error message if the input does not match to definition given in the regular expression.

Class
The name of the class to be used for the drop-down list. Required for type `GeneralCatalog`.

Translation
Defines whether the items of a general catalog have to be translated. Possible values are: *0* or *1*.

YearPeriodPast
Defines how many years in the past are available for selection from the present year in a date or date/time field. The value must be a non-negative integer.

YearPeriodFuture
Defines how many years in the future are available for selection from the present year in a date or date/time field. The value must be a non-negative integer.

ValueMin
Defines the minimum value for an integer field.

ValueMax
Defines the maximum value for an integer field.

ValueDefault
Defines the default value for an integer field.

CountMin
Defines at least how many of the current input types are available. The value must be a non-negative integer.

CountMax
Defines at most how many of the current input types are available. The value must be a non-negative integer.

CountDefault
Defines how many field should be displayed by default. The value must be a non-negative integer.

Sub
Defines a sub-element in the input field. The sub-element can contain its own input fields. It is useful if you have certain properties under a main property.

SuppressVersionAdd
This can be used to suppress creating a new version of a configuration item, when an attribute has changed. Possible values are `UpdateLastVersion` and `Ignore`.

- `UpdateLastVersion`: If this value is set and there is no other updated attribute, the attribute is updated in the current version without creating a new version.
- `Ignore`: If this value is set and there is no other updated attribute, nothing will be done, and no new version is created.

Class Definition Reference

The following class definition is an example for all possible options.

Note: The `CustomerID` and `Owner` are special keys, since these keys are used in *Customers* and *Customer Users* to assign configuration items automatically to customers and customer users by default.

```
---
- Key: OperatingSystem
  Name: Operating System
  Searchable: 1
  Input:
    Type: Text
    Required: 1
    Size: 50
    MaxLength: 100
    RegEx: Linux|MacOS|Windows|Other
    RegExErrorMessage: The operating system is unknown.
  CountMin: 0
  CountMax: 5
  CountDefault: 1

- Key: Description
  Name: Description
  Searchable: 0
  Input:
    Type: TextArea
    Required: 0
  CountMin: 0
  CountMax: 1
  CountDefault: 0

- Key: Type
  Name: Type
  Searchable: 1
  Input:
    Type: GeneralCatalog
    Class: ITSM::ConfigItem::Software::Type
    Required: 1
    Translation: 1

- Key: CustomerID
  Name: Customer Company
  Searchable: 1
  Input:
    Type: CustomerCompany

- Key: Owner
  Name: Owner
  Searchable: 1
  Input:
    Type: Customer

- Key: LicenseKey
  Name: License Key
  Searchable: 1
```

(continues on next page)

(continued from previous page)

```
  Input:
    Type: Text
    Size: 50
    MaxLength: 50
    Required: 1
  CountMin: 0
  CountMax: 100
  CountDefault: 0
  Sub:
  - Key: Quantity
    Name: Quantity
    Input:
      Type: Integer
      ValueMin: 1
      ValueMax: 1000
      ValueDefault: 1
      Required: 1
    CountMin: 0
    CountMax: 1
    CountDefault: 0

  - Key: ExpirationDate
    Name: Expiration Date
    Input:
      Type: Date
      Required: 1
      YearPeriodPast: 20
      YearPeriodFuture: 10
    CountMin: 0
    CountMax: 1
    CountDefault: 0

- Key: LastUsed
  Name: Last Used
  Input:
    Type: DateTime
    Required: 1
  CountMin: 0
  CountMax: 1
  CountDefault: 0
  SuppressVersionAdd: UpdateLastVersion
```

2.2 Agent Interface

This chapter describes the new features that are available in the agent interface after installation of the package.

2.2.1 Configuration Items

After installation of the package a new menu section will be available in the main menu.

Note: In order to grant users access to the *Asset Management* menu, you need to add them as member to the group *itsm-configitem*.

2.2.1.1 Create Configuration Item

Use this screen to add new configuration items to the configuration management database.

To add a configuration item:

1. Select a class from the list of classes.
2. Fill in the required fields.
3. Click on the *Create* button.

Fig. 25: Create Configuration Item Screen

See also:

The fields in the *Properties* widget can be very different on each classes. To see the available fields, check the *Configuration Items* module in the administrator interface.

Warning: The maximum number of 20,000 configuration items should not be exceeded. Exceeding this limit may affect the system performance.

2.2.1.2 Configuration Item List

This screen gives an overview of configuration items. Configuration items have an *Incident State* column, which includes two state types:

- Operational
- Incident

For each state type, any number of states can be registered. The state of a configuration item affects the service state, which will be dynamically calculated and displayed in the Services screen of the agent interface.

See also:

To enable the dynamic calculation, activate the following system configuration settings:

- `ITSMConfigItem::SetIncidentStateOnLink`
- `ITSMConfigItem::LinkStatus::TicketTypes`

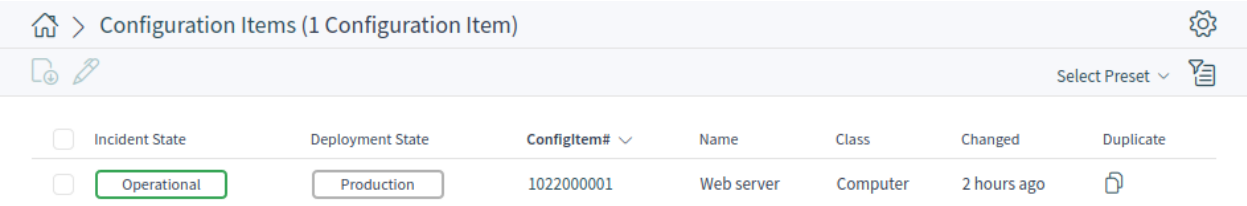

Fig. 26: Configuration Item List

2.2.1.3 Configuration Item Detail View

Use this screen to see the details of a configuration item. The configuration item detail view is available if you select a configuration item from a configuration item list.

Configuration Item Detail View Widgets

Like other business object detail views, the configuration item detail view is also highly customizable. Some of the following widgets are displayed with the default installation, but others have to be added in the screen configuration.

Configuration Item Information Widget
: This widget shows information about the configuration item.

Configuration Item Version Details Widget
: This widget shows the configuration item versions. Any change on a configuration item will create a new version. Clicking on a version in this widget will expand the details.

Attachments Widget
: This widget can be used to display attachments to the configuration item. The attachments can be downloaded and, for the images, a preview function is supported.

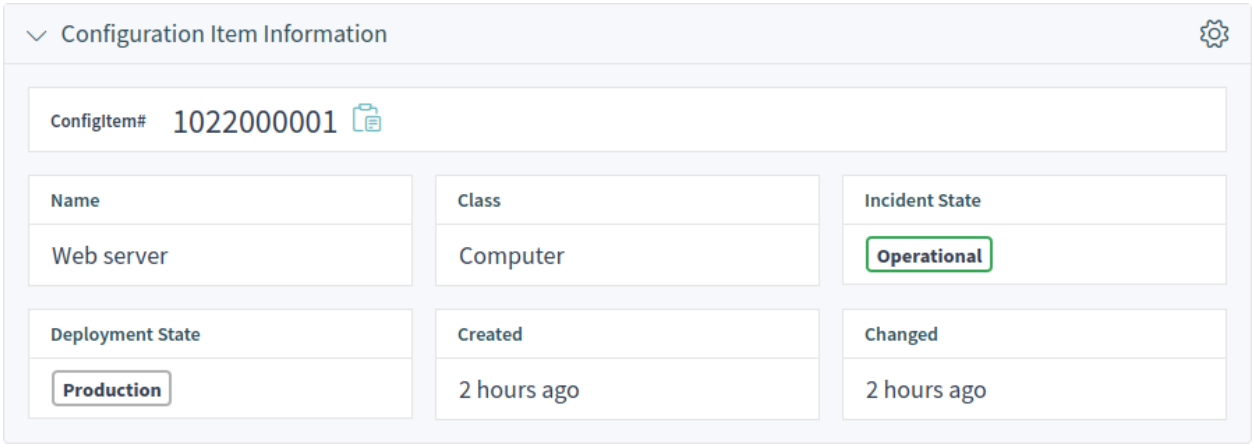

Fig. 27: Configuration Item Information Widget

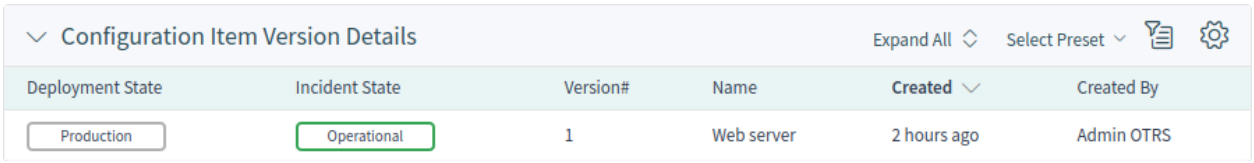

Fig. 28: Configuration Item Version Details Widget

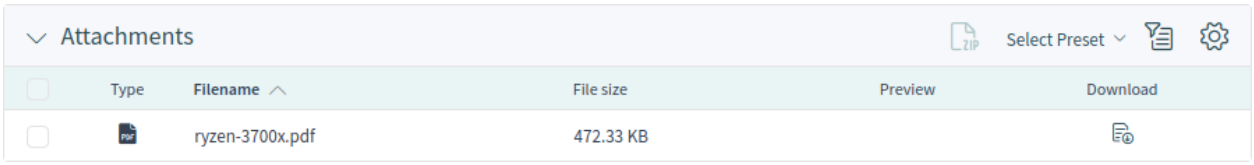

Fig. 29: Attachments Widget

Configuration Item Detail View Actions

The following actions are available in the ticket detail view.

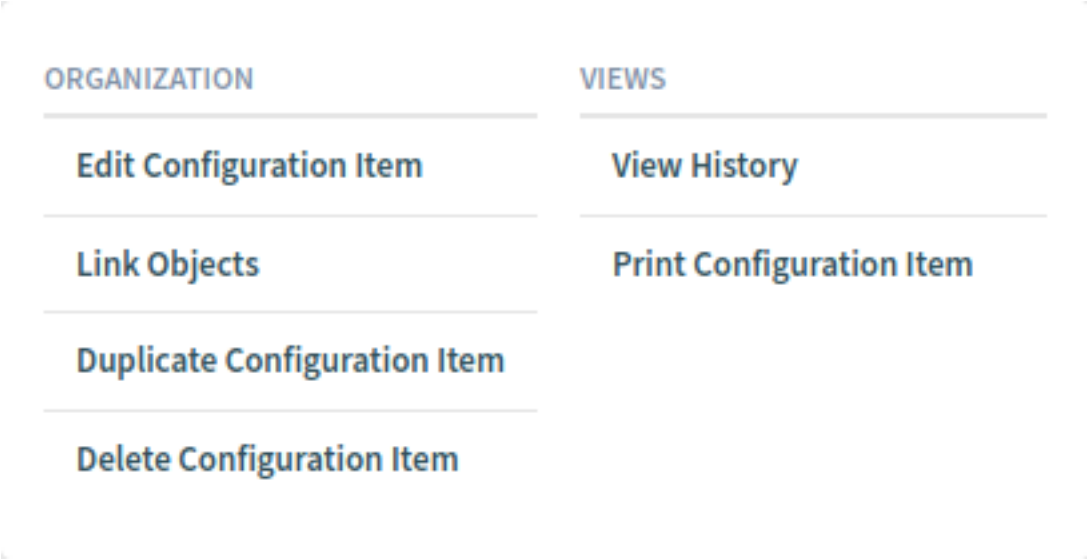

Fig. 30: Configuration Item Detail View Actions

Organization
This column groups the following actions together:

Edit Configuration Item
This action allows agents to edit the configuration item.

Link Objects
This action allows agents to link other business objects to the configuration item.

Duplicate Configuration Item
This action allows agents to duplicate the configuration item.

Delete Configuration Item
This action allows agents to delete the configuration item.

Views
This column groups the following actions together:

View History
This action allows agents to view the history of the configuration item. The history contains all operations that happened with the configuration item in the past, along with timestamp and username of the person who took the action.

Print Configuration Item
This action allows agents to print the configuration item to a PDF file and to download it.

2.2.2 Customers

After installation of the package a new widget named *Configuration Item List* will be available in the customer detail view.

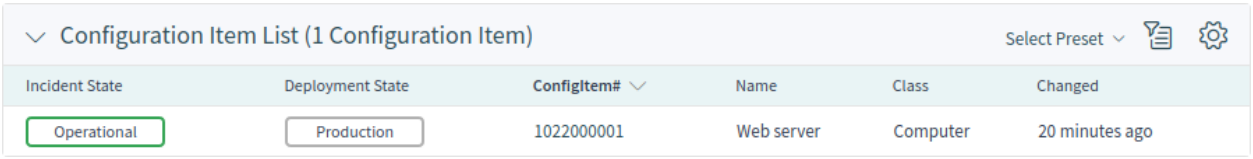

Fig. 31: Configuration Item List Widget

This widget displays the configuration items that are assigned to the customer.

The assignment is done via attribute `CustomerID` by default. If the configuration item uses different attribute for linking, you should change it in the system configuration settings.

See also:

See `AgentFrontend::CustomerCompanyDetailView::WidgetType###ConfigItemList` system configuration setting for more information.

The default setting is:

```
ClassBasedCustomerIDSearch:
  Computer: CustomerID
  Hardware: CustomerID
  Location: CustomerID
  Network: CustomerID
  Software: CustomerID
ClassBasedCustomerSearch:
  Computer: CustomerID
  Hardware: CustomerID
  Location: CustomerID
  Network: CustomerID
  Software: CustomerID
```

You also need to have this `CustomerID` attribute in the class definition to display the assigned configuration items. Check the existing class definitions in the *Configuration Items* module.

If your class definition doesn't contain the `CustomerID` attribute, then you have to add it manually.

```
- Key: CustomerID
  Name: Customer Company
  Searchable: 1
  Input:
    Type: CustomerCompany
```

2.2.3 Customer Users

After installation of the package a new widget named *Configuration Item List* will be available in the customer user detail view.

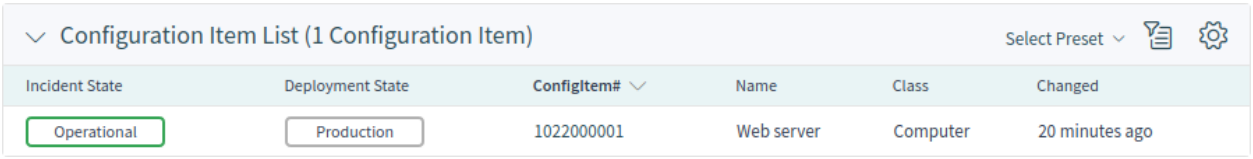

Fig. 32: Configuration Item List Widget

This widget displays the configuration items that are assigned to the customer user.

The assignment is done via attribute `Owner` by default. If the configuration item uses different attribute for linking, you should change it in the system configuration settings.

See also:

See `AgentFrontend::CustomerUserDetailView::WidgetType###ConfigItemList` system configuration setting for more information.

The default setting is:

```
ClassBasedCustomerUserSearch:
  Computer: Owner
  Hardware: Owner
  Location: Owner
  Network: Owner
  Software: Owner
ClassBasedCustomerSearch:
  Computer: Owner
  Hardware: Owner
  Location: Owner
  Network: Owner
  Software: Owner
```

You also need to have this `Owner` attribute in the class definition to display the assigned configuration items. Check the existing class definitions in the *Configuration Items* module.

If your class definition doesn't contain the `Owner` attribute, then you have to add it manually.

```
- Key: Owner
  Name: Owner
  Searchable: 1
  Input:
    Type: Customer
```

2.3 External Interface

This package has no external interface.