ITSMConfigurationManagement

Release master

Rother OSS GmbH

Dec 16, 2024

Table of Contents

CHAPTER 1

Overview

1.1 Summary

This documentation is an overview of the functionality of the OTOBO package "ITSMConfigurationManagement".

With this package, you can extend the functionality of OTOBO to manage assets and Configuration Items and link them to tickets, Customer Users, and more. This allows you to keep track of these items and have a unified solution for managing both your tickets and assets.

We provide a Basic Configuration based on a selection of the most commonly used Configuration Items to get you up and running quickly. If you are interested in configuring your own Configuration Items, you should be able to do so using the Advanced Configuration.

1.2 What is a Configuration Management Database (CMDB) ?

A Configuration Management Database (CMDB) is a centralized hub that keeps track of all the resources your organization owns, including their current state, location, and setup. It's essentially a single source of truth for all IT assets, configurations, service requests, and incidents. Plus, it can also monitor ITIL processes like change management and problem management.

One of the main reasons you'd want to maintain a CMDB is to keep your infrastructure and systems in check. Having access to accurate data - think what IT assets you have, where they are, how they're set up, and how they relate to each other - is crucial for managing their lifecycle, making changes, handling incidents, and scaling your business.

The database serves as a trusted source of information and is a fundamental building block when following ITIL practices. It gives you a complete view of all your IT configuration items (CIs), their attributes, and relationships. But here's the thing: a CMDB is more than just a massive inventory list for tracking physical assets. It also manages process, documentation, human capital, and how they all fit together with your overall IT infrastructure.

Identifying what we call Configuration Items (or CIs) is a critical step in building a CMDB. According to ITIL, a CI is basically anything that needs to be managed to deliver an IT service. That includes everything from buildings and services to software, documentation, users, and hardware. And get this - these CIs can vary wildly in terms of size and scope depending on how they're configured, used, or related to each other internally and externally.

1.3 The Relationship Between Asset Management and the Configuration Management Database (CMDB)

Asset Management and the Configuration Management Database (CMDB) are two closely related disciplines in IT management, each with its own focus. Asset Management centers on the entire lifecycle of IT resources, including procurement, deployment, maintenance, and decommissioning. Meanwhile, a CMDB focuses on configuration data and the relationships between components within the IT infrastructure.

The OTOBO CMDB serves as a central repository for tracking configuration items (CIs) and their interdependencies, providing critical context for operations and troubleshooting. For example, it helps you understand how systems are connected and the impact that changes to one component might have on others. On the other hand, Asset Management takes a financial and operational perspective, monitoring costs, ownership, and the utilization of resources.

There is some overlap between the two, as both systems often manage information about the same IT assets, such as servers or applications. By integrating these systems, you gain a comprehensive view of your IT environment: the OTOBO CMDB provides technical insights and relationship mapping, while Asset Management optimizes cost efficiency and resource usage. Together, they enable more effective IT asset management, cost reduction, and risk minimization through improved transparency.

1.4 Common Usecases and Main Benefits

Incident Management and Troubleshooting - In the event of an IT outage, technicians can quickly analyze the affected configuration items (CIs) and their dependencies. A CMDB helps pinpoint the root cause of the issue and assess potential impacts on other systems, significantly accelerating the resolution process.

Change Management - When making changes to IT infrastructure, such as system upgrades or patches, the CMDB enables risk-aware planning. It highlights which systems might be affected by the change, ensuring that potential disruptions are identified and mitigated in advance.

Asset Management and IT Compliance - A CMDB facilitates accurate tracking of IT assets, including hardware, software, licenses, warranties, and ownership. This helps organizations stay compliant with regulatory requirements while maintaining better control over IT costs and asset lifecycles. A CMDB with up-to-date data and a clear history of configuration items (CIs) significantly reduces the effort required for future audits.

Capacity and Resource Planning - The CMDB provides visibility into infrastructure resources and their current usage. This allows IT teams to optimize resource allocation and plan for future needs more effectively, avoiding over- or underutilization.

Documentation and Knowledge Management - Serving as a centralized knowledge base, a CMDB documents all configurations, systems, and their interdependencies. This is particularly valuable when onboarding new team members or planning IT projects, as it ensures comprehensive understanding and better decision-making.

Proactive Risk Analysis - By offering transparency into system dependencies, the CMDB helps identify potential vulnerabilities and risks before they escalate. IT teams can proactively address weaknesses, improving overall infrastructure resilience.

1.5 Further Reading

If you are unfamiliar with the concept of a Configuration Management Database, the article on Wikipedia can provide introductory information.

More information about IT Infrastructure Library (ITIL) Practices can be found here.

CHAPTER 2

Configuration

2.1 System Requirements

2.1.1 Framework

OTOBO 11.0.x

2.1.2 Packages

ITSMCore 11.0.1

2.2 Basic Configuration

This section provides a basic approach on how to set up an initial CMDB for productive use. It comes with a set of ready to adopt and most common Configuration Item (CIs) classes. You'll save valuable time and can start right away mapping your infrastructure and most valuable assets.

2.2.1 Assign Agent Permissions

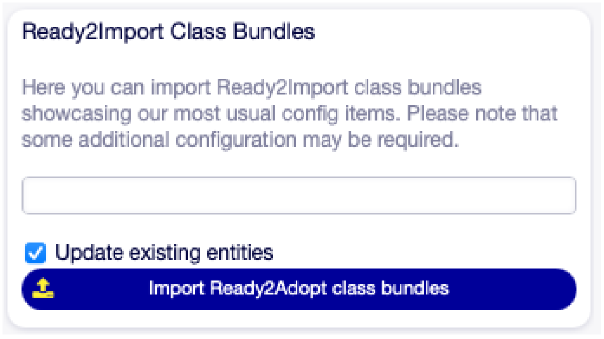
Agents who need access to CIs in the Configuration Management Database (CMDB) must be assigned to the group "itsm-configitem".

2.2.2 Import the Ready to Adopt ConfigItem Classes

Attention: Importing the Ready2Import classes will create a large number of dynamic fields and is not automatically reversible. We recommend trying this on a non-productive system first.

OTOBO provides the option to import class bundles that showcase some of the most commonly used Configuration Items (CIs). To get started, perform the following steps:

1. Open the admin view of your OTOBO Web UI.
2. Navigate to **Config Items** in the **CMDB Settings** section.
3. Locate the **Ready2Import Class Bundles** panel on the left side and select the class bundle you want to import.



Ready2Import Class Bundles

Here you can import Ready2Import class bundles showcasing our most usual config items. Please note that some additional configuration may be required.

4. Click the button to import the Ready2Import class bundles.

2.2.3 Assign Customer Permissions (Optional)

To provide customer access to CIs, enable the following system configuration settings:

- CustomerFrontend::Module###CustomerITSMConfigItem
- CustomerFrontend::Module###CustomerITSMConfigItemSearch
- CustomerFrontend::Module###CustomerITSMConfigItemZoom
- Customer::ConfigItem::PermissionConditions###01

If needed, the permission condition can be customized and expanded as described in Permissions and Access Restrictions.

2.2.4 Change Common Settings for Config Item Classes (Optional)

CI classes offer a ton of flexibility when it comes to tailoring them to your needs. Here's what you need to do to change the general attributes of a CI class:

1. Open the admin view of your OTOBO Web UI.
2. Navigate to **General Catalog** in the **Administration** section.
3. Locate the **Catalog Class** table in the middle, then click on the **ITSM::ConfigItem::Class** entry.
4. Find the **Name** table right in the middle, then click on the class name of the specific class you're looking to modify.

You can customize attributes like permission groups that control who gets access to CIs in a particular class, as well as categories. A more detailed description is provided in General Catalog Classes of the CMDB.

2.2.5 Change Advanced Settings for Config Item Classes (Optional)

You can also tweak the look and feel of a CI class, as well as its associated data, right from the Admin Config Item screen.

1. Open the admin view of your OTOBO Web UI.
2. Navigate to **Config Items** in the **CMDB Settings** section.
3. Find the **Actions** panel on the left side and select the class you want to edit.
4. Click the button **Change class definition**.

In the YAML editor on the page you can configure various things, such as Dynamic Fields used as attributes for the Class and the appearance of CIs belonging to the class in the zoom views. This is covered more in-depth in Config Item Definitions.

2.2.6 Customize Attributes Shown in Config Item Overview (Optional)

For configuring the attributes available and visible in the Config Item overviews, the following system configuration settings can be used:

- ITSMConfigItem::Frontend::AgentITSMConfigItem###ClassColumnsAvailable
- ITSMConfigItem::Frontend::AgentITSMConfigItem###ClassColumnsDefault
- ITSMConfigItem::Frontend::CustomerITSMConfigItem###ClassColumnsAvailable
- ITSMConfigItem::Frontend::CustomerITSMConfigItem###ClassColumnsDefault

2.3 Custom Configuration and Advanced Features

2.3.1 General Catalog Classes of the CMDB

Classes

Define the Config Item Classes (ITSM::ConfigItem::Class) available on the system, and get their basic settings in place. This includes (but isn't limited to):

- **Access Group**: Controls user (agent) access to this class.
- **Name Module**: No functionality in the default OTOBO Core installation. Allows extensions to automatically set CI names.
- **Number Module**: Module used to define visible CI Numbers (in contrast to their internal IDs - compare TicketNumber and -ID)
- **Version-String Module**: Versions can either be incremental (1, 2, 3,…) (default), defined by a Template Toolkit expression from the respectively current attributes of the CI, or (if left empty) be manually defined when adding/editing CIs in the frontend.
- **Version-String Expression**: Currently only used in combination with the version string module "Template Toolkit"
- **Version Trigger**: Defines which attributes trigger the creation of a new version when changed. See Config Item Versioning and History.
- **Categories**: Which categories this class should be listed in in the CI Overview of the agent interface.

Categories

A collection of categories (ITSM::ConfigItem::Class::Category) which are used to organize classes in the Agent Frontend.

Config Item Roles

Config Item Roles available on the system (ITSM::ConfigItem::Role). Can be defined and used in Admin->ConfigItem to define common sets of attributes between CI classes into a section, making it reusable across different areas.

Deployment States

The Deployment State of a CI describes the current stage it's reached in its lifecycle (e.g. from planned to decommissioned). You can maintain and customize the field values for the Deployment State in the General Catalog (ITSM::ConfigItem::DeploymentState). Plus, you can assign a unique color code to each Deployment State, providing a quick visual cue for the current status.

Incident States

The Incident State of a CI describes its current production state (e.g., operational, incident). You can maintain and customize the available field values for this state in the General Catalog (ITSM::Core::IncidentState). The Incident State is displayed as a traffic light system in views (like the TreeView).

2.3.2 Config Item Definitions

General Structure

A CI definition outlines all the 'Pages' that appear on the ConfigItemZoom masks and implicitly determines which attributes belong to a class. Each attribute corresponds to a dynamic field of the "ConfigItem" object type, which must exist.

Structurally, the 'Pages' in the definitions YAML contain a layout with 'Sections' that bundle information blocks and some additional attributes. These 'Sections' are later defined in more detail and may include, for example, a list of dynamic fields to be displayed. Sections can also be embedded as 'Roles' and separately defined within each role's definition.

Example class "Domain":

```
Pages:
  - Name: Domain Information
    Layout:
      Columns: 2
      ColumnWidth: 1fr 1fr
    Interfaces:
      - Agent
      - Customer
    Content:
      - Section: Domain Info::Summary
        RowStart: 1
        ColumnStart: 1
      - Section: Domain Info::Backlinks
```

```
          RowStart: 2
          ColumnStart: 1
        - Section: Domain Info::Description
          RowStart: 1
          RowSpan: 2
          ColumnStart: 2
   - Name: Accounting
     Layout:
        Columns: 2
        ColumnWidth: 1fr 1fr
     Interfaces:
        - Agent
     Groups:
        - accounting
     Content:
        - Section: Accounting

Sections:
   Accounting:
      Content:
         - Header: Accounting
         - DF: ITSMAccounting-InventoryNumber
         - Grid:
            Columns: 3
            ColumnWidth: 1fr 1fr 1fr
            Rows:
               -
                  - DF: ITSMAccounting-DateOfInvoice
                  - DF: ITSMAccounting-OrderDate
                  - DF: ITSMAccounting-WarrantyDate
               -
                  - DF: ITSMAccounting-CostUnit
                  - DF: ITSMAccounting-InvestmentCosts
                  - DF: ITSMAccounting-OperatingCosts
               -
                  - DF: ITSMAccounting-OrderNumber
                  - DF: ITSMAccounting-InvoiceNumber
                  - DF: ITSMAccounting-ReferenceToAccount

# here the latest version of the role "ITSM Domain Information" is included as
↪"Domain Info"
# and its sections can be used on the pages above
Roles:
   Domain Info:
      Name: ITSM Domain Information
```

Setting Reference

Pages

- **Name** (required): The name of the page.
- **Layout** (required): The layout of the grid on this page.
  - **Columns**: Number of columns. Example: “`Columns: 3`” displays three columns.
  - **ColumnWidth**: Basic CSS defining the column widths.

- **Interfaces** (optional): An array containing the interfaces on which this page is available (default: [Agent]).
- **Groups** (optional): If defined, an array containing groups a user must belong to in order to view this page.
- **Content** (required): The content, i.e., an array of sections and their positions on this page.
  - **Section** (required): The section name.
  - **RowStart** (optional): The starting row in the page grid of this section.
  - **RowSpan** (optional): The number of rows this section should span.
  - **ColumnStart** (optional): The starting column in the page grid of this section.
  - **ColumnSpan** (optional): The number of columns this section should span.

Sections

The **Sections** subsection of the YAML contains a hash of the sections referenced in the **Pages**. The following keys are valid for each section:

- **Type** (optional): Defines the type of the section. Depending on the type, other attributes may or may not be available. Available types include:
  - **DynamicFields** (default): A standard section containing dynamic fields.
  - **Description**: A rich text description that may contain images that can be defined in the CI editing masks.
  - **ConfigItemLinks**: Displays ConfigItems linked via dynamic fields (not used for edit masks).
  - **ReferencedSection**: Displays a section of a referenced CI in a reference dynamic field (not used for edit masks).

Type: DynamicFields

An additional key, **Content**, is mandatory. This works like content in ticket masks. Additionally, a header for the section can be provided.

- **Header** (optional): A header for this section.
- **DF**: A dynamic field (the name).
  - **Mandatory** (optional): Set to 1 if the field is mandatory in edit masks.
  - **Readonly** (optional): Set to 1 if the field is read-only in edit masks (only for basic field types).
  - **Label** (optional): Overrides the field label in edit masks.
- **Grid**: A multi-column section of dynamic fields.
  - **Columns**: Number of columns.
  - **ColumnWidth** (optional): Column widths (e.g., "1fr 40px 2fr"; "%" is not supported).
  - **Rows**: A matrix of dynamic fields (array of arrays).

Example:

```
Sections:
   Info:
      Content:
         - DF: Computer-OS
         - DF: Owner
           Mandatory: 1
```

```
          Label: In front of the monitor
        - Grid:
           Columns: 2
           ColumnWidth: 1fr 1fr
           Rows:
              # First row
              -
                 - DF: DateBought
                   Readonly: 1
                 - DF: DateWarranty
              # Second row
              -
                 - DF: Computer-Application
                 - DF: Computer-LicenseKey
```

Type: Description

No additional settings are available.

Type: ConfigItemLinks

Lists linked Config Items.

- **Header** (optional): A header for this section.
- **LinkedAs** (optional): Source (default), Target, or Both.
- **LinkTypes** (optional): An array of link types.

Type: ReferencedSection

- **ReferenceField** (mandatory): The reference field containing the referenced Config Item.
- **SectionName** (mandatory): The section name of the referenced Config Item to display.
- **FieldListPre** (optional): Dynamic fields of this Config Item rendered before the referenced section.
- **FieldListPost** (optional): Dynamic fields rendered after the referenced section.

2.3.3 Config Item Versioning and History

Config Items (CIs) have a general history listing all changes made over time in list form. This is the standard reference to find when and who made which changes to a CI. Additionally snapshots of CIs can be stored as (historic) CI versions, which can be selected and viewed. This feature is automatically enabled, if Version Triggers are set in the CI classes in the general catalog.

The following features and settings have important interactions with CI versions:

- Reference dynamic fields can statically link to a CI version instead of dynamically to the latest version of a CI.
- Version view for the customers can be enabled or disabled via the SysConfig in ITSM-ConfigItem::Frontend::CustomerITSMConfigItemZoom
- To ensure audit-proof documentation, you'll need to disable the option to delete config items.

2.3.4 Dynamic Field Specifics

Fields for Configuration Item (CI) classes are defined as Dynamic Fields with the new Object Type **ITSM ConfigItem**. You can define custom fields for Configuration Item (CI) classes by creating dynamic fields associated with the ITSM ConfigItem object type. To utilize these fields, simply define them as dynamic fields for the CI object type and then reference them in your CI class definition.

Some dynamic fields that might be particularly useful for your Configuration Management Database (CMDB) include:

- **Reference**
  - **Agent**: This field allows you to refer to an Agent in OTOBO. It can be useful for defining and identifying the person responsible for a Configuration Item (e.g., the IT administrator, who is also an Agent).
  - **ConfigItem**: This field allows you to refer to another Configuration Item in OTOBO. It can be useful for linking multiple Configuration Items together or defining a hierarchy, such as Country <-> Subsidiary <-> Building <-> Room.
  - **ConfigItemVersion**: This field allows you to refer to a specific version of another Configuration Item in OTOBO.
  - **Customer**: This field allows you to refer to a Customer in OTOBO. It can be valuable for defining the company or department associated with a Configuration Item. Additionally, this allows the Configuration Item to appear in the Customer Information Center widget.
  - **Customer User**: This field allows you to refer to a Customer User in OTOBO. It can be useful for defining the owner of a Configuration Item, such as an internal Customer User (employee). With this, you can also view the Configuration Item in the Customer User Information Center widget.
- **Dynamic Field Lens** The **Lens** field lets you view specific values from a referenced object (like through a periscope). If you have a reference to a Configuration Item (via a **DF Reference ConfigItem**), you can use the Lens field to view or change values from that referenced Configuration Item. It is often a good idea to set this field as **Readonly** to prevent changes to these values.

  Required settings for the Lens field:

  - **Referenced DF**: Defines the Dynamic Field where the referenced object is mentioned. This is the Configuration Item whose values you want to display.
  - **Attribute DF of the referenced object**: This is the (Dynamic) Field within the referenced object from which you want to view the value.
  - You can make this field **Readonly** via the **Readonly: 1** option.

  Example: You can reference a **Server ConfigItem** and view values like the **IP Address** and **Server Owner** through the Lens fields.
- **General Catalog** The **General Catalog** field is essentially a dropdown field. You define the values for this field in the **Admin > General Catalog** section. The difference between a General Catalog field and a regular **Dropdown** Dynamic Field is that with the General Catalog, you define the values globally, allowing them to be reused across multiple Dynamic Fields.

  For example, you could have two fields with the same dropdown values but different names or labels for the Dynamic Field. These fields might appear in different CI classes, but both would pull their options from the same General Catalog.

2.3.5 Config Item Links

Coming soon.

2.3.6 Import/Export

1. **From CSV**: You can import and export data as a .csv file, which can even be automated - for example, by dropping a .csv file into a specific directory that's set up to import regularly into OTOBO via a CronJob (command: bin/otobo.Console.pl Admin::ITSM::ImportExport::Import).
2. **Using Web Services**: Another option is to use web services to import and export data. For instance, you could set it up so that whenever a new config item is created or changed in a distributed system connected to the OTOBO CMDB via a web service, a new config item is also created or an existing one updated in the CMDB.

2.3.7 Command-Line Operations

The following commands can be executed using bin/otobo.Console.pl:

- **Admin::ITSM::Configitem::ClassExport** Export dynamic field classes and their respective dynamic fields.
- **Admin::ITSM::Configitem::ClassImport** Import dynamic field classes and their respective dynamic fields.
- **Admin::ITSM::Configitem::Delete** Delete Configuration Items (CIs) either completely, by class, or by deployment state and number.
- **Admin::ITSM::Configitem::DumpGraph** Export the graph of Configuration Items as an SVG file named OTOBO_ITSM_Config_Items.svg.
- **Admin::ITSM::Configitem::ListDuplicates** List Configuration Items with non-unique names.
- **Admin::ITSM::Configitem::UpgradeTo11** Upgrade the complete Configuration Management Database (CMDB) from OTOBO 10 to OTOBO 11. This includes changing all Configuration Item definitions, preparing dynamic fields for each attribute, and migrating the data.
- **Maint::ITSM::Configitem::RebuildLinkTable** Rebuild the database table configitem_link based on dynamic fields of type Reference. Only fields linking Configuration Items are affected.

2.3.8 Permissions and Access Restrictions

You can use Access Control Lists (ACLs) to set rules for who can view or hide config items. If you are not familiar with the way access control is handled in OTOBO, you can read more about that in the OTOBO 11 Administration Manual.

- **ACLs**
  - New Object Type: **ITSM ConfigItem**
  - **Match settings**:
    - **ConfigItem**
    - **Frontend**
    - **User**
  - **Change settings**:

* ConfigItem
* Form

2.3.9 Configuration: Web Services

The ITSMConfigurationManagement package also extends the existing web service capabilities. Before diving into this section, make sure you're familiar with the basics of OTOBO web services. You can find an introduction in OTOBO 11 Administration Manual: Webservices.

Note: To keep things easy to read, not all combinations of possible setups are shown here. Instead we provide some basic examples, which serve the purpose to provide a sufficient illustration of possibilites. Here are the presets we're using - and keep in mind that these can be customized to fit your needs:

Authentication: BasicAuth method with username and password

Transport: HTTP::REST

Mapping Type: Simple

Attention: Every operation, as well as the invoker listed and described below, requires authentication via an agent account. The accoubt also needs to have the permissions enabled for the respective action **and** config item(s). Customer user access to these modules is currently not available.

Operations

ConfigItemGet

With this operation, it is possible to fetch config item data from an OTOBO system. A simple example web service definition could look as follows:

```
---
Debugger:
  DebugThreshold: debug
  TestMode: '0'
Description: ''
FrameworkVersion: 11.0.x
Provider:
  Operation:
    ConfigItemGet:
      Description: Get Config Item data.
      MappingInbound:
        Type: Simple
      MappingOutbound:
        Type: Simple
      Type: ConfigItem::ConfigItemGet
  Transport:
    Config:
      AdditionalHeaders: ~
      KeepAlive: ''
      MaxLength: '1000000'
      RouteOperationMapping:
        ConfigItemGet:
          Route: /ConfigItemGet/:ConfigItemID
    Type: HTTP::REST
RemoteSystem: ''
```

Based on this definition, a request may look like this:

```
curl -X POST --header "Content-Type: application/json"
--data '{
    "UserLogin": "AgentUser",
    "Password": "Password",
    "Attachments": 1
}'
https://YOURSERVER/otobo/nph-genericinterface.pl/Webservice/<WEBSERVICE_NAME>/
↪ConfigItemGet/<ConfigItemID>
```

Possible parameters to pass are:

UserLogin Either UserLogin and Password or SessionID are required.

Password Passwords are passed in plain text.

SessionID The SessionID may be retrieved by using a SessionCreate web service operation.

ConfigItemID Mandatory. Can be comma separated IDs or an Array.

> **Note:** In the above curl example, the ConfigItemID is appended to the URL string. This is possible because the example web service definition contains a route mapping which allows this (Route: /ConfigItemGet/:ConfigItemID).

Attachments optional, 1 as default. If it's set with the value 1, attachments for articles will be included on ConfigItem data

Resulting data may be returned as follows:

```
{
  "ConfigItem": [
    {
      "Attachment": "",
      "ChangeBy": 2,
      "ChangeTime": "2024-11-14 09:06:47",
      "Class": "Building",
      "ConfigItemID": 2,
      "CreateBy": 2,
      "CreateTime": "2024-11-14 09:06:47",
      "CurDeplState": "Production",
      "CurDeplStateType": "productive",
      "CurInciState": "Operational",
      "CurInciStateType": "operational",
      "DeplState": "Production",
      "DeplStateType": "productive",
      "Description": "Some meaningful Description",
      "DynamicField_Location-ReferenceToSubsidiary": [
        1
      ],
      "InciState": "Operational",
      "InciStateType": "operational",
      "LastVersionID": 2,
      "Name": "Building 01",
      "Number": "1042000001",
      "VersionID": 2,
      "VersionString": "1"
    }
  ]
}
```

ConfigItemSearch

With this operation, it is possible to search for config items based on a wide variety of search query options. A simple example web service definition could look as follows:

```
---
Debugger:
  DebugThreshold: debug
  TestMode: '0'
Description: ''
FrameworkVersion: 11.0.x
Provider:
  Operation:
    ConfigItemSearch:
      Description: Search for Config Items.
      MappingInbound:
        Type: Simple
      MappingOutbound:
        Type: Simple
      Type: ConfigItem::ConfigItemSearch
  Transport:
    Config:
      AdditionalHeaders: ~
      KeepAlive: ''
      MaxLength: '1000000'
      RouteOperationMapping:
        ConfigItemSearch:
          Route: /ConfigItemSearch/
    Type: HTTP::REST
RemoteSystem: ''
```

Based on this definition, a request may look like this:

```
curl -X POST --header "Content-Type: application/json"
--data '{
    "UserLogin": "AgentUser",
    "Password": "Password",
    "Name": "Building*",
    "DynamicField_FieldNameA": {
        "Empty": 1
    },
    "DynamicField_FieldNameB": {
        "Equals": "SomeString"
    },
    "DynamicField_FieldNameC": {
        "GreaterThan": "1970-01-01 00:00:01",
        "SmallerThan": "1980-12-31 23:59:59"
    },
    "DynamicField_FieldNameD": {
        "GreaterThanEquals": "1970-01-01 00:00:01",
        "SmallerThanEquals": "1980-12-31 23:59:59"
    }
}'
https://YOURSERVER/otobo/nph-genericinterface.pl/Webservice/<WEBSERVICE_NAME>/
↪ConfigItemSearch
```

Possible parameters to pass are:

UserLogin Either UserLogin and Password or SessionID are required.

Password Passwords are passed in plain text.

SessionID The SessionID may be retrieved by using a SessionCreate web service operation.

ConfigItemID Optional. Can be either a String or an array of Strings. Use ConfigItemSearch as a config item filter on a single config item or a predefined config item list.

Number Optional. Can be either a String or an array of Strings. SQL placeholders like % are allowed.

Name Optional. Can be either a String or an array of Strings. SQL placeholders like % are allowed.

Classes Optional. Has to be an Array of Strings.

ClassIDs Optional. Has to be an Array of Strings.

DeplStates Optional. Deployment state names of config items as an Array of Strings.

DeplStateIDs Optional. Deployment state IDs of config items as an Array of Strings.

CurDeplStates Optional. Current deployment state names of config itemsas an Array of Strings.

CurDeplStateIDs Optional. Current deployment state IDs of config items as an Array of Strings.

InciStates Optional. Incident state names of config items as an Array of Strings.

InciStateIDs Optional. Incident state IDs of config items as an Array of Strings.

CurInciStates Optional. Current incident state names of config items as an Array of Strings.

CurInciStateIDs Optional. Current incident state IDs of config items as an Array of Strings.

Limit Optional. Maximum number of config items returned, default is 10000.

CreateBy Optional. Agent user ID of the agent who created the config item as an Array of Strings.

ChangeBy Optional. Agent user ID of agent who performed the latest change to the config item as an Array of Strings.

DynamicFields At least one operator must be specified. Operators will be connected with AND, values in an operator with OR. You can also pass more than one argument to an operator: [‘value1’, ‘value2’]. Available operators are:

- **Empty**: will return dynamic fields without a value, set to 0 to search fields with a value present
- **Equals**
- **Like**: ‘Equals’ operator with wild card support
- **GreaterThan**
- **GreaterThanEquals**
- **SmallerThan**
- **SmallerThanEquals**

Caution: The set of operators available is restricted for some dynamic field types. For example, dynamic fields of type checkbox only support the operators ‘Empty’ and ‘Equals’.

ConfigItemCreateTimeOlderMinutes Optional. Filter for config items that have been created more than … minutes ago. Provide the number of minutes as a String.

ConfigItemCreateTimeNewerMinutes Optional. Filter for config items that have been created less than … minutes ago. Provide the number of minutes as a String.

ConfigItemCreateTimeNewerDate Optional. Filter for config items that have been created later than the given time stamp. Provide the time in DateTime-String using format 'YYYY-MM-DD HH:MM:SS'

ConfigItemCreateTimeOlderDate Optional. Filter for config items that have been created before the given time stamp. Provide the time in DateTime-String using format 'YYYY-MM-DD HH:MM:SS'

ConfigItemLastChangeTimeOlderMinutes Optional. Filter for config items that have been updated more than … minutes ago. Provide the number of minutes as a String.

ConfigItemLastChangeTimeNewerMinutes Optional. Filter for config items that have been updated less than … minutes ago. Provide the number of minutes as a String.

ConfigItemLastChangeTimeNewerDate Optional. Filter for config items that have been updated later than the given time stamp. Provide the time in DateTime-String using format 'YYYY-MM-DD HH:MM:SS'

ConfigItemLastChangeTimeOlderDate Optional. Filter for config items that have been updated before the given time stamp. Provide the time in DateTime-String using format 'YYYY-MM-DD HH:MM:SS'

OrderBy Optional. The direction to order results in. Possible values are: 'Down', 'Up'. Also possible as array for sub sorting.

SortBy Optional. The attribute to sort results by. Possible values are: ConfigItem, Number, Name, DeplState, InciState, CurDeplState, CurInciState, Age, Created, Changed. Also possible as array for sub sorting

Resulting data may be returned as follows:

```
{
  "ConfigItemID": [
    "2",
    "1"
  ]
}
```

ConfigItemUpsert

With this operation, it is possible to add or update one or more config items based on the sent data. A simple example web service definition could look as follows:

```
---
Debugger:
  DebugThreshold: debug
  TestMode: '0'
Description: ''
FrameworkVersion: 11.0.x
Provider:
  Operation:
    ConfigItemUpsert:
      Description: Add or update one or more Config Items.
      Identifier41:
      - Number
      [...]
      Identifier74:
      - Number
      MappingInbound:
        Type: Simple
      MappingOutbound:
        Type: Simple
      Type: ConfigItem::ConfigItemUpsert
  Transport:
    Config:
      AdditionalHeaders: ~
      KeepAlive: ''
      MaxLength: '1000000'
      RouteOperationMapping:
        ConfigItemUpsert:
          Route: /ConfigItemUpsert/
    Type: HTTP::REST
RemoteSystem: ''
```

Based on this definition, a request may look like this:

```
curl -X POST --header "Content-Type: application/json"
--data '{
    "UserLogin": "AgentUser",
    "Password": "Password",
    "ConfigItem": [
        {
          "Attachment": "",
          "ChangeBy": 2,
          "ChangeTime": "2024-11-14 09:06:47",
          "Class": "Building",
          "ConfigItemID": 2,
          "CreateBy": 2,
          "CreateTime": "2024-11-14 09:06:47",
          "CurDeplState": "Production",
          "CurDeplStateType": "productive",
          "CurInciState": "Operational",
          "CurInciStateType": "operational",
          "DeplState": "Production",
          "DeplStateType": "productive",
          "Description": "Some meaningful Description",
          "DynamicField_Location-ReferenceToSubsidiary": [
            1
          ],
          "InciState": "Operational",
          "InciStateType": "operational",
          "LastVersionID": 2,
          "Name": "Building 01",
          "Number": "1042000001",
          "VersionID": 2,
          "VersionString": "1"
        }
    ]
}'
https://YOURSERVER/otobo/nph-genericinterface.pl/Webservice/<WEBSERVICE_NAME>/
↪ConfigItemUpsert
```

Hint: Note that, based on the version trigger configured per class in the admin general catalog interface, a new version may or may not be created based on the data provided via request.

Possible parameters to pass are:

UserLogin Either UserLogin and Password or SessionID are required.

Password Passwords are passed in plain text.

SessionID The SessionID may be retrieved by using a SessionCreate web service operation.

ConfigItem Either a single config item data hash or an array of config item data hashes to add or update.

Hint: The system determines wether to perform an insertion or update based on the identifier configured per class in the web service configuration.

Resulting data may be returned as follows:

```
{
  "ConfigItem": [
    {
      "ConfigItemID": "1",
      "Name": "Subsidiary 01"
    },
    {
      "ConfigItemID": "3",
      "Name": "Building 02"
    }
  ]
}
```

ConfigItemDelete

With this operation, it is possible to delete an existing config item. A simple example web service definition could look as follows:

```
---
Debugger:
  DebugThreshold: debug
  TestMode: '0'
Description: ''
FrameworkVersion: 11.0.x
Provider:
  Operation:
    ConfigItemDelete:
      Description: Delete a Config Item.
      MappingInbound:
        Type: Simple
      MappingOutbound:
        Type: Simple
      Type: ConfigItem::ConfigItemDelete
  Transport:
    Config:
      AdditionalHeaders: ~
      KeepAlive: ''
      MaxLength: '1000000'
      RouteOperationMapping:
        ConfigItemDelete:
          Route: /ConfigItemDelete/:ConfigItemID
    Type: HTTP::REST
RemoteSystem: ''
```

Based on this definition, a request may look like this:

```
curl -X POST --header "Content-Type: application/json"
--data '{
    "UserLogin": "AgentUser",
    "Password": "Password",
}'
https://YOURSERVER/otobo/nph-genericinterface.pl/Webservice/<WEBSERVICE_NAME>/
↪ConfigItemDelete/<ConfigItemID>
```

Possible parameters to pass are:

UserLogin Either UserLogin and Password or SessionID are required.

Password Passwords are passed in plain text.

SessionID The SessionID may be retrieved by using a SessionCreate web service operation.

ConfigItemID Required. Can be comma separated IDs or an Array.

Note: In the above curl example, the ConfigItemID is appended to the URL string. This is possible because the example web service definition contains a route mapping which allows this (Route: /ConfigItemGet/:ConfigItemID).

Resulting data may be returned as follows:

```
{
  "ConfigItemID": [
    "4"
  ]
}
```

Invoker

ConfigItemCreate

Using the ConfigItemCreate invoker, it is possible to send config item data to a remote endpoint. A simple example web service definition could look as follows:

```
---
Debugger:
  DebugThreshold: debug
  TestMode: '0'
Description: ''
FrameworkVersion: 11.0.x
RemoteSystem: ''
Requester:
  Invoker:
    ConfigItemCreate:
      Description: 'Send Config Item data to remote system.'
      Events:
      - Asynchronous: '1'
        Condition:
          Condition:
            '1':
              Fields:
                Number:
                  Match: .+
                  Type: Regexp
              Type: and
          ConditionLinking: and
        Event: ConfigItemCreate
      MappingInbound:
        Type: Simple
      MappingOutbound:
        Type: Simple
      Type: ConfigItem::ConfigItemCreate
  Transport:
    Config:
      Authentication:
        AuthType: BasicAuth
        BasicAuthPassword: Password
        BasicAuthUser: AgentUser
      DefaultCommand: GET
      Host: https://YOURSERVER/otobo/nph-genericinterface.pl/Webservice/<WEBSERVICE_
↪NAME>
      InvokerControllerMapping:
        ConfigItemCreate:
          Controller: /ConfigItemCreate/
      Timeout: '120'
    Type: HTTP::REST
```

Without any mapping applied, the outgoing config item data will look like the following:

```
{
  "ChangeBy": "2",
  "ChangeTime": "1970-01-01 00:00:01",
  "Class": "Client Software",
  "ClassID": "45",
  "ConfigItemID": "1",
  "CreateBy": "2",
  "CreateTime": "1970-01-01 00:00:01",
  "CurDeplState": "Production",
  "CurDeplStateID": "2",
  "CurDeplStateType": "productive",
  "CurInciState": "Operational",
  "CurInciStateID": "1",
  "CurInciStateType": "operational",
  "DefinitionID": "1",
  "DeplState": "Production",
  "DeplStateID": "2",
  "DeplStateType": "productive",
  "Description": "Some meaningful description",
  "DynamicField_ITSM-LensToLicenseCount": "2",
  "DynamicField_ITSM-LensToLicensePeriodFrom": "1970-01-01 00:00:00",
  "DynamicField_ITSM-LensToLicensePeriodUntil": "2069-12-31 00:00:00",
  "DynamicField_ITSM-LensToLicenseType": "ApacheLizenz",
  "DynamicField_ITSM-ReferenceToLicense": "Name of License",
  "InciState": "Operational",
  "InciStateID": "1",
  "InciStateType": "operational",
  "LastVersionID": "1",
  "Name": "ClientSoftware",
  "Number": "1045000002",
  "VersionID": "1",
  "VersionString": "1"
}
```

Note: Note that values of dynamic fields are transformed. E.g. reference fields will not be included with their value ID, but with a readable value. For reference dynamic fields of type ConfigItem, ConfigItemVersion or Ticket, this value can be configured with the attribute 'Display value'.

ConfigItemFetch

Using the ConfigItemFetch invoker, it is possible to fetch data from a remote endpoint and add or update one or more config items based on that data. A simple example web service definition could look as follows:

```
---
Debugger:
  DebugThreshold: debug
  TestMode: '0'
Description: ''
FrameworkVersion: 11.0.x
RemoteSystem: ''
Requester:
  Invoker:
    ConfigItemFetch:
      Description: Fetch Config Item data.
      Events:
        - Asynchronous: '1'
          Condition:
            Condition:
              '1':
                Fields:
                  Title:
                    Match: .+
                    Type: Regexp
                Type: and
            ConditionLinking: and
          Event: TicketQueueUpdate
      MappingInbound:
        Type: Simple
      MappingOutbound:
        Type: Simple
      Type: ConfigItem::ConfigItemFetch
  Transport:
    Config:
      Authentication:
        AuthType: BasicAuth
        BasicAuthPassword: Password
        BasicAuthUser: AgentUser
      DefaultCommand: GET
      Host: https://YOURSERVER/otobo/nph-genericinterface.pl/Webservice/<WEBSERVICE_
↪NAME>
      InvokerControllerMapping:
        ConfigItemFetch:
          Controller: /ConfigItemFetch/
      Timeout: '120'
    Type: HTTP::REST
```

The ConfigItemFetch invoker works slightly similar to the ConfigItemUpsert operation in terms that it adds or updates one or more config items based on the defined identifier attributes. Upon being triggered by the configured events (in the example above TicketQueueUpdate), the remote system is requested and the returned data are processed.

Mapping of incoming data

Response data structures of a remote system may vary, but an example is nonetheless helpful for understanding. Given the following data structure:

```
[
  {
      "currency": "EUR",
      "currency_symbol": "€",
      "customer_id": 1,
      "customer_ip_address": "127.0.0.1",
      "customer_note": "",
      "discount_tax": "0.00",
      "discount_total": "0.00",
      "fee_lines": [],
      "id": 200,
      "next_payment_date": "01 January 1970",
      "number": "200",
      "parent_id": 0,
      "parent_order_id": 100,
      "recurring_amount": 42,
      "refunds": [],
      "shipping_lines": [],
      "shipping_tax": "0.00",
      "shipping_total": "0.00",
      "status": "pending",
      "subscription_id": 100,
      "subscriptions_expiry_date": "01 January 1970",
      "tax_lines": [],
      "total": "0.05",
      "total_tax": "0.00",
      "transaction_id": "",
      "user_name": "AgentUser"
  },
  {
      "currency": "EUR",
      "currency_symbol": "€",
      "customer_id": 1,
      "customer_ip_address": "127.0.0.1",
      "customer_note": "",
      "discount_tax": "0.00",
      "discount_total": "0.00",
      "fee_lines": [],
      "id": 198,
      "next_payment_date": "01 January 1970",
      "number": "198",
      "parent_id": 0,
      "parent_order_id": 98,
      "recurring_amount": 42,
      "refunds": [],
      "shipping_lines": [],
      "shipping_tax": "0.00",
      "shipping_total": "0.00",
      "status": "processing",
      "subscription_id": 98,
      "subscriptions_expiry_date": "01 January 1970",
      "tax_lines": [],
      "total": "0.00",
      "total_tax": "0.00",
      "transaction_id": "",
      "user_name": "AgentUser"
  }
]
```

The data stems from a WordPress system. A ConfigItemFetch invoker receiving such data needs to also contain a mapping to select and transform it into a data structure for creating or updating a config item. The usable config item attributes are:

Hint: Note that, based on the version trigger configured per class in the admin general catalog interface, a new version may or may not be created based on the data fetched via request.

The data provided by the remote system which is requested by the invoker need to be mapped to the usual config item data, which are the following:

Number Optional. Number of config item, will be generated if not provided.

Name Optional in case a name module is configured for the class. The Name of the config item.

Class Mandatory. The name of the class in which the config item should be created.

VersionString Optional if a version string module is configured for the class. The version identifier of the config item.

DeploymentState Mandatory. The name of the deployment state the config item is in.

IncidentState Mandatory. The name of the incident state the config item is in.

Description Optional. The description text if a description is configured for the respective class.

DynamicFields Optional. The data of the config item dynamic fields.

Hint: The system determines wether to perform an insertion or update based on the identifier configured per class in the web service configuration.

Attention: When implementing the following mapping, the presets listed at the start of the page get overwritten in the following points:

Mapping Type: Simple -> XSLT

This also implies that the above webservice config has to be adapted in this case.

Given the incoming data and the needed config item data structure, a mapping may look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes"/>
  <!-- Main template for transformation -->
  <xsl:template match="/RootElement">
    <data>
      <xsl:for-each select="data">
        <ConfigItems>

          <!-- Straight date conversion for next_payment_date -->
          <DynamicField_SUBSC-NextPaymentDate>
            <xsl:variable name="inputDate" select="next_payment_date"/>
            <xsl:variable name="day" select="format-number(substring-before(
↪$inputDate, ' '), '00')"/>
            <xsl:variable name="monthName" select="substring-before(substring-after(
↪$inputDate, ' '), ' ')"/>
            <xsl:variable name="year" select="substring-after(substring-after(
↪$inputDate, ' '), ' ')"/>

            <!-- Conversion of month name into two-digit number -->
            <xsl:variable name="month">
              <xsl:choose>
                <xsl:when test="$monthName = 'January'">01</xsl:when>
                <xsl:when test="$monthName = 'February'">02</xsl:when>
                <xsl:when test="$monthName = 'March'">03</xsl:when>
                <xsl:when test="$monthName = 'April'">04</xsl:when>
                <xsl:when test="$monthName = 'May'">05</xsl:when>
                <xsl:when test="$monthName = 'June'">06</xsl:when>
                <xsl:when test="$monthName = 'July'">07</xsl:when>
                <xsl:when test="$monthName = 'August'">08</xsl:when>
                <xsl:when test="$monthName = 'September'">09</xsl:when>
                <xsl:when test="$monthName = 'October'">10</xsl:when>
                <xsl:when test="$monthName = 'November'">11</xsl:when>
                <xsl:when test="$monthName = 'December'">12</xsl:when>
              </xsl:choose>
            </xsl:variable>

            <!-- Composing the date in format YYYY-MM-DD HH:MM:SS -->
            <xsl:value-of select="concat($year, '-', $month, '-', $day, ' 00:00:00')"/
↪>
          </DynamicField_SUBSC-NextPaymentDate>

          <!-- Conditional printing and date conversion for subscriptions_expiry_date␣
↪-->
          <xsl:variable name="expiryDate" select="subscriptions_expiry_date"/>
          <xsl:if test="normalize-space($expiryDate) != '' and $expiryDate != '---'">
            <DynamicField_SUBSC-SubscriptionsExpiryDate>

              <!-- Conversion of expiryDate in preferred format -->
              <xsl:variable name="expiryDay" select="format-number(substring-before(
↪$expiryDate, ' '), '00')"/>
              <xsl:variable name="expiryMonthName" select="substring-before(substring-
↪after($expiryDate, ' '), ' ')"/>
              <xsl:variable name="expiryYear" select="substring-after(substring-after(
↪$expiryDate, ' '), ' ')"/>
              <xsl:variable name="expiryMonth">
                <xsl:choose>
                    <xsl:when test="$expiryMonthName = 'January'">01</xsl:when>
                    <xsl:when test="$expiryMonthName = 'February'">02</xsl:when>
                    <xsl:when test="$expiryMonthName = 'March'">03</xsl:when>
                    <xsl:when test="$expiryMonthName = 'April'">04</xsl:when>
```

```
                        <xsl:when test="$expiryMonthName = 'May'">05</xsl:when>
                        <xsl:when test="$expiryMonthName = 'June'">06</xsl:when>
                        <xsl:when test="$expiryMonthName = 'July'">07</xsl:when>
                        <xsl:when test="$expiryMonthName = 'August'">08</xsl:when>
                        <xsl:when test="$expiryMonthName = 'September'">09</xsl:when>
                        <xsl:when test="$expiryMonthName = 'October'">10</xsl:when>
                        <xsl:when test="$expiryMonthName = 'November'">11</xsl:when>
                        <xsl:when test="$expiryMonthName = 'December'">12</xsl:when>
                    </xsl:choose>
                  </xsl:variable>

                  <!-- Printing the formatted date -->
                  <xsl:value-of select="concat($expiryYear, '-', $expiryMonth, '-',
↪$expiryDay, ' 00:00:00')"/>
                </DynamicField_SUBSC-SubscriptionsExpiryDate>
              </xsl:if>

              <!-- Printing other fields without conversion -->
              <Class>Subscription</Class>
              <DeploymentState>Production</DeploymentState>
              <IncidentState>Operational</IncidentState>
              <Name><xsl:value-of select="parent_order_id"/></Name>
              <DynamicField_SUBSC-OrderID><xsl:value-of select="parent_order_id"/></
↪DynamicField_SUBSC-OrderID>
              <DynamicField_SUBSC-RecurringAmount><xsl:value-of select="recurring_amount"/
↪></DynamicField_SUBSC-RecurringAmount>
              <DynamicField_SUBSC-SubscStatus><xsl:value-of select="status"/></
↪DynamicField_SUBSC-SubscStatus>
              <DynamicField_SUBSC-SubscriptionID><xsl:value-of select="subscription_id"/>
↪</DynamicField_SUBSC-SubscriptionID>
              <DynamicField_SUBSC-UserName><xsl:value-of select="user_name"/></
↪DynamicField_SUBSC-UserName>
            </ConfigItems>
          </xsl:for-each>
        </data>
      </xsl:template>
    </xsl:stylesheet>
```

The combination of the incoming data and the mapping above will result in the following data for config item creation or update, respectively:

```
[
  {
    "Class": "Subscription",
    "DeploymentState": "Production",
    "DynamicField_SUBSC-NextPaymentDate": "1970-01-01 00:00:00",
    "DynamicField_SUBSC-OrderID": "100",
    "DynamicField_SUBSC-RecurringAmount": "42",
    "DynamicField_SUBSC-SubscStatus": "pending",
    "DynamicField_SUBSC-SubscriptionID": "100",
    "DynamicField_SUBSC-SubscriptionsExpiryDate": "1970-01-01 00:00:00",
    "DynamicField_SUBSC-UserName": "AgentUser",
    "IncidentState": "Operational",
    "Name": "100"
  },
  {
    "Class": "Subscription",
    "DeploymentState": "Production",
    "DynamicField_SUBSC-NextPaymentDate": "1970-01-01 00:00:00",
    "DynamicField_SUBSC-OrderID": "98",
    "DynamicField_SUBSC-RecurringAmount": "42",
    "DynamicField_SUBSC-SubscStatus": "pending",
    "DynamicField_SUBSC-SubscriptionID": "98",
    "DynamicField_SUBSC-SubscriptionsExpiryDate": "1970-01-01 00:00:00",
    "DynamicField_SUBSC-UserName": "AgentUser",
    "IncidentState": "Operational",
    "Name": "100"
  }
]
```

ConfigItemUpdate

Using the ConfigItemUpdate invoker, it is possible to send config item data to a remote endpoint. A simple example web service definition could look as follows:

```
---
Debugger:
  DebugThreshold: debug
  TestMode: '0'
Description: ''
FrameworkVersion: 11.0.x
RemoteSystem: ''
Requester:
  Invoker:
    ConfigItemUpdate:
      Description: 'Send Config Item data to remote system.'
      Events:
      - Asynchronous: '1'
        Condition:
          Condition:
            '1':
              Fields:
                Number:
                  Match: .+
                  Type: Regexp
              Type: and
          ConditionLinking: and
        Event: VersionCreate
      MappingInbound:
        Type: Simple
      MappingOutbound:
        Type: Simple
      Type: ConfigItem::ConfigItemUpdate
  Transport:
    Config:
      Authentication:
        AuthType: BasicAuth
        BasicAuthPassword: Password
        BasicAuthUser: AgentUser
      DefaultCommand: GET
      Host: https://YOURSERVER/otobo/nph-genericinterface.pl/Webservice/<WEBSERVICE_
↪NAME>
      InvokerControllerMapping:
        ConfigItemUpdate:
          Controller: /ConfigItemUpdate/
      Timeout: '120'
    Type: HTTP::REST
```

Without any mapping applied, the outgoing config item data will look like the following:

```
{
  "ChangeBy": "2",
  "ChangeTime": "1970-01-01 00:00:01",
  "Class": "Client Software",
  "ClassID": "45",
  "ConfigItemID": "1",
  "CreateBy": "2",
  "CreateTime": "1970-01-01 00:00:01",
  "CurDeplState": "Production",
  "CurDeplStateID": "2",
  "CurDeplStateType": "productive",
  "CurInciState": "Operational",
  "CurInciStateID": "1",
  "CurInciStateType": "operational",
  "DefinitionID": "1",
  "DeplState": "Production",
  "DeplStateID": "2",
  "DeplStateType": "productive",
  "Description": "Some meaningful description",
  "DynamicField_ITSM-LensToLicenseCount": "2",
  "DynamicField_ITSM-LensToLicensePeriodFrom": "1970-01-01 00:00:00",
  "DynamicField_ITSM-LensToLicensePeriodUntil": "2069-12-31 00:00:00",
  "DynamicField_ITSM-LensToLicenseType": "ApacheLizenz",
  "DynamicField_ITSM-ReferenceToLicense": "Name of License",
  "InciState": "Operational",
  "InciStateID": "1",
  "InciStateType": "operational",
  "LastVersionID": "1",
  "Name": "ClientSoftware",
  "Number": "1045000002",
  "VersionID": "1",
  "VersionString": "1"
}
```

Note: Note that values of dynamic fields are transformed. E.g. reference fields will not be included with their value ID, but with a readable value. For reference dynamic fields of type ConfigItem, ConfigItemVersion and Ticket, this value can be configured with the attribute 'Display value'.

It is also possible to trigger a ConfigItemFetch invoker manually via console command:

```
bin/otobo.Console.pl Maint::WebService::TriggerConfigItemFetch <WEBSERVICE_NAME>
↪<INVOKER_NAME>
```

2.3.10 Migration from OTOBO 10 / OTRS

bin/otobo.Console.pl Admin::ITSM::Configitem::UpgradeTo11

This command upgrades the entire Configuration Management Database (CMDB) from OTOBO 10 to OTOBO 11. During the process, all Configuration Item definitions will be updated, a dynamic field will be created for each Configuration Item attribute, and the data will be migrated.

The migration is performed in several steps to allow for customization. However, you can bypass this behavior by using the `--use-defaults` flag, which is only recommended for test systems.

If you plan to use attributes globally (i.e., shared across multiple Configuration Item classes), you can use the `--no-namespace` flag. This prevents creating a separate dynamic field for each class, which is particularly useful for attributes like "Owner" or "Customer" that aren't tied to a specific class. You can also make more detailed customizations during the migration if needed. Customization options are available after the first and second steps of the process:

1. In the first step, all existing classes are evaluated, and attribute maps are generated to match the previous attributes with the new Dynamic Fields that will be created. These maps are then saved to the working directory. You can review the map for each class and freely modify the Dynamic Field names, including their namespaces (the part before the "-"), as long as you follow standard Dynamic Field naming rules, such as maintaining uniqueness (shared across object types like ticket Dynamic Fields). This also allows you to make Dynamic Fields common across multiple classes, if desired.

2. The definitions are prepared during this step, including both the class definitions and the Dynamic Field definitions. By default, each Configuration Item (CI) class will have just one page and one section containing all Dynamic Fields in a long list. On a production system where you plan to actively use the migrated data, it's recommended to spend some time organizing the Dynamic Fields into different sections, or even multiple pages, to achieve a cleaner and more user-friendly layout. You might find it helpful to review the ready-to-use classes for inspiration. Since all legacy versions of the classes are being migrated—and older versions can vary significantly—a large number of files may be generated. You'll need to decide whether to focus on improving only the latest definition for each class or to refine all versions.

2.3.11 Ready2Import Class Bundles

Through the Config Item management, you can import the Ready2Adopt Class Bundles as described in the Basic Configuration. These include the most commonly used configuration attributes. Please note that some additional configurations may be required to map a Config Item according to your requirements.

We currently provide the following Class Bundles:

- IT-Servicemanagement

Which provides a host of Config Item types typically used in IT service management, such as:

- Infrastructure
- Location
- Contract
- Workplace
- Software
- Network Infrastructure
- …

… and many more.

2.3.12 Relations

Coming soon.

2.4 Configuration Reference

2.4.1 Core::BulkAction

ITSMConfigItem::Frontend::BulkFeature

Enables configuration item bulk action feature for the agent frontend to work on more than one configuration item at a time.

ITSMConfigItem::Frontend::BulkFeatureGroup

Enables configuration item bulk action feature only for the listed groups.

2.4.2 Core::DynamicFields::DriverRegistration

DynamicFields::Driver###ConfigItem

DynamicField backend registration.

DynamicFields::Driver###ConfigItemVersion

DynamicField backend registration.

2.4.3 Core::DynamicFields::ObjectTypeRegistration

DynamicFields::ObjectType###ITSMConfigItem

DynamicField object registration.

2.4.4 Core::Elasticsearch::Settings

Elasticsearch::ExcludedCIClasses

ConfigItems of the following classes will not be stored on the Elasticsearch server. To apply this to existing CIs, the CI migration has to be run via console, after changing this option.

Elasticsearch::ExcludedCIDeploymentStates

ConfigItems with the following deployment states will not be stored on the Elasticsearch server. To apply this to existing CIs, the CI migration has to be run via console, after changing this option.

Elasticsearch::ConfigItemStoreFields

Fields stored in the configuration item index which are used for other things besides fulltext searches. For the complete functionality all fields are mandatory.

Elasticsearch::ConfigItemSearchFields

Fields of the configuration item index, used for the fulltext search. Fields are also stored, but are not mandatory for the overall functionality. Inclusion of attachments can be disabled by setting the entry to 0 or deleting it.

Elasticsearch::QuickSearchShow###ConfigItem

Objects to search for, how many entries and which attributs to show. ConfigItem attributes have to explicitly be stored via Elasticsearch.

2.4.5 Core::Event::ITSMConfigItem

ITSMConfigItem::EventModulePost###100-History

Config item event module that enables logging to history in the agent interface.

ITSMConfigItem::EventModulePost###300-DefinitionConfigItemUpdate

Config item event module that updates config items to their current definition.

ITSMConfigItem::EventModulePost###400-LinkSync

Config item event module that updates the table configitem_ĺink.

ITSMConfigItem::EventModulePost###500-RecalcCurInciState

Config item event module updates the current incident state.

ITSMConfigItem::EventModulePost###4000-ScriptDynamicFields

Evaluate all script fields.

DynamicField::EventModulePost###200-DynamicFieldSync

Dynamic field event module that marks config item definitions as out of sync, if containing dynamic fields change.

ITSMConfigItem::EventModulePost###1000-GenericInterface

Performs the configured action for each event (as an Invoker) for each configured Webservice.

2.4.6 Core::Event::Ticket

ITSMConfigItem::EventModulePost###042-ITSMConfigItemTicketStatusLink

Event module to set configitem-status on ticket-configitem-link.

Ticket::EventModulePost###042-ITSMConfigItemTicketStatusLink

Event module to set configitem-status on ticket-configitem-link.

2.4.7 Core::GeneralCatalog

GeneralCatalogPreferences###DeploymentStates

Parameters for the deployment states in the preferences view of the agent interface.

GeneralCatalogPreferences###DeploymentStatesColors

Parameters for the deployment states color in the preferences view of the agent interface.

GeneralCatalogPreferences###NameModule

Parameters for the name module for config item classes in the preferences view of the agent interface.

GeneralCatalogPreferences###NumberModule

Parameters for the number module for config item classes in the preferences view of the agent interface.

GeneralCatalogPreferences###VersionStringModule

Parameters for the version string module for config item classes in the preferences view of the agent interface.

GeneralCatalogPreferences###VersionStringExpression

Parameters for the version string template toolkit module for config item classes in the preferences view of the agent interface.

GeneralCatalogPreferences###VersionTrigger

Parameters for the version trigger for config item classes in the preferences view of the agent interface.

GeneralCatalogPreferences###Categories

Parameters for the categories for config item classes in the preferences view of the agent interface.

GeneralCatalogPreferences###Permissions

Parameters for the example permission groups of the general catalog attributes.

2.4.8 Core::ITSMConfigItem

ITSMConfigItem::CINameRegex

Defines regular expressions individually for each ConfigItem class to check the ConfigItem name and to show corresponding error messages.

ITSMConfigItem::Permission::Class###010-ClassGroupCheck

Module to check the group responsible for a class.

ITSMConfigItem::Permission::Item###010-ItemClassGroupCheck

Module to check the group responsible for a configuration item.

ITSMConfigItem::Hook

The identifier for a configuration item, e.g. ConfigItem#, MyConfigItem#. The default is ConfigItem#.

UniqueCIName::EnableUniquenessCheck

Enables/disables the functionality to check ITSM onfiguration items for unique names. Before enabling this option you should check your system for already existing config items with duplicate names. You can do this with the console command Admin::ITSM::Configitem::ListDuplicates.

UniqueCIName::UniquenessCheckScope

Check for a unique name only within the same ConfigItem class ('class') or globally ('global'), which means every existing ConfigItem is taken into account when looking for duplicates.

Customer::ConfigItem::PermissionConditions###01

Define a set of conditions under which a customer is allowed to see a config item. Conditions can optionally be restricted to certain customer groups. Name is the only mandatory attribute. If no other options are given, all config items will be visible under that category.

Customer::ConfigItem::PermissionConditions###02

Define a set of conditions under which a customer is allowed to see a config item. Conditions can optionally be restricted to certain customer groups. Name is the only mandatory attribute. If no other options are given, all config items will be visible under that category.

Customer::ConfigItem::PermissionConditions###03

Define a set of conditions under which a customer is allowed to see a config item. Conditions can optionally be restricted to certain customer groups. Name is the only mandatory attribute. If no other options are given, all config items will be visible under that category.

Customer::ConfigItem::PermissionConditions###04

Define a set of conditions under which a customer is allowed to see a config item. Conditions can optionally be restricted to certain customer groups. Name is the only mandatory attribute. If no other options are given, all config items will be visible under that category.

Customer::ConfigItem::PermissionConditions###05

Define a set of conditions under which a customer is allowed to see a config item. Conditions can optionally be restricted to certain customer groups. Name is the only mandatory attribute. If no other options are given, all config items will be visible under that category.

Customer::ConfigItem::PermissionConditionColumns###Default

Customer::ConfigItem::PermissionConditionColumns###01

Customer::ConfigItem::PermissionConditionColumns###02

Customer::ConfigItem::PermissionConditionColumns###03

Customer::ConfigItem::PermissionConditionColumns###04

Customer::ConfigItem::PermissionConditionColumns###05

CMDBTreeView::DefaultDepth

Defines the default relations depth to be shown.

CMDBTreeView::ShowLinkLabels

Defines if the link type labels must be shown in the node connections.

2.4.9 Core::ITSMConfigItem::ACL

ITSMConfigItemACLKeysLevel1Match

Defines which items are available in first level of the ITSM Config Item ACL structure.

ITSMConfigItemACLKeysLevel1Change

Defines which items are available in first level of the ITSM Config Item ACL structure.

ITSMConfigItemACLKeysLevel2::Possible

Defines which items are available in second level of the ITSM Config Item ACL structure.

ITSMConfigItemACLKeysLevel2::PossibleAdd

Defines which items are available in second level of the ITSM Config Item ACL structure.

ITSMConfigItemACLKeysLevel2::PossibleNot

Defines which items are available in second level of the ITSM Config Item ACL structure.

ITSMConfigItemACLKeysLevel2::Properties

Defines which items are available in second level of the ITSM Config Item ACL structure.

ITSMConfigItemACLKeysLevel2::PropertiesDatabase

Defines which items are available in second level of the ITSM Config Item ACL structure.

ITSMConfigItemACLKeysLevel3::Actions###100-Default

Defines which items are available for 'Action' in third level of the ITSM Config Item ACL structure.

ConfigItemACL::ACLPreselection

Whether the execution of ConfigItemACL can be avoided by checking cached field dependencies. This can improve loading times of formulars, but has to be disabled, if ACLModules are to be used for ITSMConfigItem- and Form-ReturnTypes.

ConfigItemACL::Autoselect

Whether fields should be automatically filled (1), and in that case also be hidden from ticket formulars (2).

2.4.10 Core::ImportExport::ObjectBackend::ModuleRegistration

ImportExport::ObjectBackendRegistration###ITSMConfigItem

Object backend module registration for the import/export module.

2.4.11 Core::LinkObject

LinkObject::DefaultSubObject###ITSMConfigItem

Defines the default subobject of the class 'ITSMConfigItem'.

LinkObject::ITSMConfigItem::ShowColumns

Defines the shown columns of CIs in the link table complex view for all CI classes. If there is no entry, then the default columns are shown.

LinkObject::ITSMConfigItem::ShowColumnsByClass

Defines the shown columns of CIs in the link table complex view, depending on the CI class. Each entry must be prefixed with the class name and double colons (i.e. Computer::). There are a few CI-Attributes that common to all CIs (example for the class Computer: Computer::Name, Computer::CurDeplState, Computer::CreateTime). To show individual CI-Attributes as defined in the CI-Definition, the following scheme must be used (example for the class Computer): Computer::HardDisk::1, Computer::HardDisk::1::Capacity::1, Computer::HardDisk::2, Computer::HardDisk::2::Capacity::1. If there is no entry for a CI class, then the default columns are shown.

2.4.12 Core::LinkStatus

ITSMConfigItem::SetIncidentStateOnLink

Set the incident state of a CI automatically when a Ticket is Linked to a CI.

ITSMConfigItem::LinkStatus::TicketTypes

Defines which type of ticket can affect the status of a linked CI.

ITSMConfigItem::LinkStatus::DeploymentStates

Defines the relevant deployment states where linked tickets can affect the status of a CI.

ITSMConfigItem::LinkStatus::IncidentStates

Defines the order of incident states from high (e.g. cricital) to low (e.g. functional).

ITSMConfigItem::LinkStatus::LinkTypes

Defines which type of link (named from the ticket perspective) can affect the status of a linked CI.

2.4.13 Core::Stats

Stats::DynamicObjectRegistration###ITSMConfigItem

Module to generate ITSM config item statistics.

2.4.14 Daemon::SchedulerCronTaskManager::Task

Daemon::SchedulerCronTaskManager::Task###TriggerConfigItemFetch-01

Triggers ConfigItemFetch invoker automatically.

Daemon::SchedulerCronTaskManager::Task###TriggerConfigItemFetch-02

Triggers ConfigItemFetch invoker automatically.

Daemon::SchedulerCronTaskManager::Task###TriggerConfigItemFetch-03

Triggers ConfigItemFetch invoker automatically.

2.4.15 Frontend::Admin

Events###ITSMConfigItem

List of all Package events to be displayed in the GUI.

2.4.16 Frontend::Admin::ModuleRegistration

Frontend::Module###AdminITSMConfigItem

Frontend module registration for the agent interface.

Frontend::Module###AdminGenericInterfaceInvokerConfigItem

Frontend module registration for the agent interface.

2.4.17 Frontend::Admin::ModuleRegistration::AdminOverview

Frontend::NavigationModule###AdminITSMConfigItem

Admin area navigation for the agent interface.

2.4.18 Frontend::Admin::ModuleRegistration::Loader

Loader::Module::AdminITSMConfigItem###002-ITSMConfigItem

Loader module registration for the agent interface.

Loader::Module::AdminGenericInterfaceInvokerConfigItem###007-ITSMConfigurationManagement

Loader module registration for the agent interface.

2.4.19 Frontend::Admin::ModuleRegistration::MainMenu

Frontend::Navigation###AdminITSMConfigItem###003-ITSMConfigItem

Main menu item registration.

2.4.20 Frontend::Admin::View::ITSMConfigItemDefinition

ITSMConfigItem::Frontend::AdminITSMConfigItem###EditorRows

Defines the number of rows for the CI definition editor in the admin interface.

2.4.21 Frontend::Agent::ITSMConfigItem::MenuModule

ITSMConfigItem::Frontend::MenuModule###000-Back

Shows a link in the menu to go back in the configuration item zoom view of the agent interface.

ITSMConfigItem::Frontend::MenuModule###200-History

Shows a link in the menu to access the history of a configuration item in the its zoom view of the agent interface.

ITSMConfigItem::Frontend::MenuModule###300-Edit

Shows a link in the menu to edit a configuration item in the its zoom view of the agent interface.

ITSMConfigItem::Frontend::MenuModule###400-Print

Shows a link in the menu to print a configuration item in the its zoom view of the agent interface.

ITSMConfigItem::Frontend::MenuModule###500-Link

Shows a link in the menu that allows linking a configuration item with another object in the config item zoom view of the agent interface.

ITSMConfigItem::Frontend::MenuModule###550-TreeView

Shows a link in the menu to display the configuration item links as a Tree View.

ITSMConfigItem::Frontend::MenuModule###600-Duplicate

Shows a link in the menu to duplicate a configuration item in the its zoom view of the agent interface.

ITSMConfigItem::Frontend::MenuModule###700-ConfigItemDelete

Shows a link in the menu to delete a configuration item in its zoom view of the agent interface.

2.4.22 Frontend::Agent::ITSMConfigItem::MenuModulePre

ITSMConfigItem::Frontend::PreMenuModule###100-Zoom

Shows a link in the menu to zoom into a configuration item in the configuration item overview of the agent interface.

ITSMConfigItem::Frontend::PreMenuModule###200-History

Shows a link in the menu to access the history of a configuration item in the configuration item overview of the agent interface.

ITSMConfigItem::Frontend::PreMenuModule###300-Duplicate

Shows a link in the menu to duplicate a configuration item in the configuration item overview of the agent interface.

2.4.23 Frontend::Agent::ITSMConfigItem::Permission

ITSMConfigItem::Frontend::AgentITSMConfigItem###Permission

Required permissions to use the ITSM configuration item screen in the agent interface.

ITSMConfigItem::Frontend::AgentITSMConfigItemEdit###Permission

Required permissions to use the edit ITSM configuration item screen in the agent interface.

ITSMConfigItem::Frontend::AgentITSMConfigItemAdd###Permission

Required permissions to use the add ITSM configuration item screen in the agent interface.

ITSMConfigItem::Frontend::AgentITSMConfigItemHistory###Permission

Required permissions to use the history ITSM configuration item screen in the agent interface.

ITSMConfigItem::Frontend::AgentITSMConfigItemPrint###Permission

Required permissions to use the print ITSM configuration item screen in the agent interface.

ITSMConfigItem::Frontend::AgentITSMConfigItemZoom###Permission

Required permissions to use the ITSM configuration item zoom screen in the agent interface.

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###Permission

Required permissions to use the ITSM configuration item search screen in the agent interface.

ITSMConfigItem::Frontend::AgentITSMConfigItemTreeView###Permission

Required permissions to use the ITSM configuration item tree view screen in the agent interface.

ITSMConfigItem::Frontend::AgentITSMConfigItemBulk###Permission

Required permissions to use the bulk ITSM configuration item screen in the agent interface.

2.4.24 Frontend::Agent::ITSMConfigItemAttachment::Permission

ITSMConfigItem::Frontend::AgentITSMConfigItemAttachment###Permission

Required permissions to use the ITSM configuration item attachment action in the agent interface.

2.4.25 Frontend::Agent::ITSMConfigItemOverview

ITSMConfigItem::Frontend::Overview###Small

Defines an overview module to show the small view of a configuration item list.

2.4.26 Frontend::Agent::LinkObject

LinkObject::ComplexTable::SettingsVisibility###ITSMConfigItem

Define Actions where a settings button is available in the linked objects widget (LinkObject::ViewMode = "complex"). Please note that these Actions must have registered the following JS and CSS files: Core.AllocationList.css, Core.UI.AllocationList.js, Core.UI.Table.Sort.js, Core.Agent.TableFilters.js and Core.Agent.LinkObject.js.

2.4.27 Frontend::Agent::ModuleRegistration

Frontend::Module###AgentITSMConfigItem

Frontend module registration for the agent interface.

Frontend::Module###AgentITSMConfigItemZoom

Frontend module registration for the agent interface.

Frontend::Module###AgentITSMConfigItemEdit

Frontend module registration for the agent interface.

Frontend::Module###AgentITSMConfigItemPrint

Frontend module registration for the agent interface.

Frontend::Module###AgentITSMConfigItemDelete

Frontend module registration for the agent interface.

Frontend::Module###AgentITSMConfigItemTreeView

Frontend module registration for the agent interface.

Frontend::Module###AgentITSMConfigItemAdd

Frontend module registration for the agent interface.

Frontend::Module###AgentITSMConfigItemSearch

Frontend module registration for the agent interface.

Frontend::Module###AgentITSMConfigItemHistory

Frontend module registration for the agent interface.

Frontend::Module###AgentITSMConfigItemBulk

Frontend module registration for the agent interface.

Frontend::Module###AgentITSMConfigItemAttachment

Frontend module registration for the agent interface.

2.4.28 Frontend::Agent::ModuleRegistration::Loader

Loader::Module::AgentITSMConfigItem###003-ITSMConfigItem

Loader module registration for the agent interface.

Loader::Module::AgentITSMConfigItemZoom###003-ITSMConfigItem

Loader module registration for the agent interface.

Loader::Module::AgentITSMConfigItemEdit###003-ITSMConfigItem

Loader module registration for the agent interface.

Loader::Module::AgentITSMConfigItemTreeView###003-ITSMConfigItem

Loader module registration for the agent interface.

Loader::Module::AgentITSMConfigItemAdd###003-ITSMConfigItem

Loader module registration for the agent interface.

Loader::Module::AgentITSMConfigItemSearch###003-ITSMConfigItem

Loader module registration for the agent interface.

Loader::Module::AgentITSMConfigItemHistory###003-ITSMConfigItem

Loader module registration for the agent interface.

2.4.29 Frontend::Agent::ModuleRegistration::MainMenu

Frontend::Navigation###AgentITSMConfigItem###003-ITSMConfigItem

Main menu item registration.

Frontend::Navigation###AgentITSMConfigItemAdd###003-ITSMConfigItem

Main menu item registration.

Frontend::Navigation###AgentITSMConfigItemSearch###003-ITSMConfigItem

Main menu item registration.

Frontend::Navigation###AgentITSMConfigItemBulk###003-ITSMConfigItem

Main menu item registration.

2.4.30 Frontend::Agent::ModuleRegistration::MainMenu::Search

Frontend::Search###ConfigItem

Configuration item search backend router of the agent interface.

Frontend::Search::JavaScript###ConfigItem

JavaScript function for the search frontend.

2.4.31 Frontend::Agent::View::AgentITSMConfigItemBulk

ITSMConfigItem::Frontend::AgentITSMConfigItemBulk###DynamicField

Dynamic fields shown in the additional ITSM field screen of the agent interface.

2.4.32 Frontend::Agent::View::AgentITSMConfigItemEdit

ITSMConfigItem::Frontend::AgentITSMConfigItemEdit###DynamicField

Dynamic fields shown in the additional ITSM field screen of the agent interface.

2.4.33 Frontend::Agent::View::AgentITSMConfigItemHistory

ITSMConfigItem::Frontend::AgentITSMConfigItemPrint###DynamicField

Dynamic fields shown in the additional ITSM field screen of the agent interface.

2.4.34 Frontend::Agent::View::AgentITSMConfigItemZoom

ITSMConfigItem::Frontend::AgentITSMConfigItemZoom###DynamicField

Dynamic fields shown in the additional ITSM field screen of the agent interface.

2.4.35 Frontend::Agent::View::ConfigItemSearch

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###ExtendedSearchCondition

Allows extended search conditions in config item search of the agent interface. With this feature you can search e. g. config item name with this kind of conditions like "(key1*&&*key2)" or "(key1*||*key2)".

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###SearchPageShown

Number of config items to be displayed in each page of a search result in the agent interface.

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###SortBy::Default

Defines the default config item attribute for config item sorting of the config item search result of the agent interface.

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###Order::Default

Defines the default config item order in the config item search result of the agent interface. Up: oldest on top. Down: latest on top.

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###Defaults###Number

Defines the default shown config item search attribute for config item search screen.

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###Defaults###Name

Defines the default shown config item search attribute for config item search screen.

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###Defaults###DeploymentStateIDs

Defines the default shown config item search attribute for config item search screen.

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###Defaults###IncidentStateIDs

Defines the default shown config item search attribute for config item search screen.

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###Defaults###CustomerID

Defines the default shown config item search attribute for config item search screen.

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###Defaults###CustomerUserLogin

Defines the default shown config item search attribute for config item search screen.

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###Defaults###ITSMConfigItemCreateTimePoint

Default data to use on attribute for config item search screen. Example: "ITSMConfigItemCreateTimePointFormat=year;ITSMConfigItemCreateTimePointStart=Last;ITSMConfigItemCreateTimePoint=2;".

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###Defaults###ITSMConfigItemCreateTimeSlot

Default data to use on attribute for config item search screen. Example: "ITSMConfigItemCreateTimeStartYear=2010;ITSMConfigItemCreateTimeStartMonth=10;ITSMConfigItemCreateTimeStartDay=4;ITSMConfigItemCreate

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###Defaults###ITSMConfigItemChangeTimePoint

Defines the default shown config item search attribute for config item search screen.

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###Defaults###ITSMConfigItemChangeTimeSlot

Defines the default shown config item search attribute for config item search screen.

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###Defaults###DynamicField

Defines the default shown config item search attribute for config item search screen. Example: "Key" must have the name of the Dynamic Field in this case 'X', "Content" must have the value of the Dynamic Field depending on the Dynamic Field type, Text: 'a text', Dropdown: '1', Date/Time: 'Search_DynamicField_XTimeSlotStartYear=1974; Search_DynamicField_XTimeSlotStartMonth=01; Search_DynamicField_XTimeSlotStartDay=26; Search_DynamicField_XTimeSlotStartHour=00; Search_DynamicField_XTimeSlotStartMinute=00; Search_DynamicField_XTimeSlotStartSecond=00; Search_DynamicField_XTimeSlotStopYear=2013; Search_DynamicField_XTimeSlotStopMonth=01; Search_DynamicField_XTimeSlotStopDay=26; Search_DynamicField_XTimeSlotStopHour=23; Search_DynamicField_XTimeSlotStopMinute=59; Search_DynamicField_XTimeSlotStopSecond=59;' and or 'Search_DynamicField_XTimePointFormat=week; Search_DynamicField_XTimePointStart=Before; Search_DynamicField_XTimePointValue=7';.

2.4.36 Frontend::Agent::View::CustomerInformationCenter

AgentCustomerInformationCenter::Backend###0060-CIC-ITSMConfigItemCustomerCompany

Parameters for the dashboard backend of the customer company config item overview of the agent interface . "Limit" is the number of entries shown by default. "Group" is used to restrict the access to the plugin (e. g. Group: admin;group1;group2;). "Default" determines if the plugin is enabled by default or if the user needs to enable it manually. "CacheTTLLocal" is the cache time in minutes for the plugin.

2.4.37 Frontend::Agent::View::CustomerUserInformationCenter

AgentCustomerUserInformationCenter::Backend###0060-CUIC-ITSMConfigItemCustomerUser

Parameters for the dashboard backend of the customer company config item overview of the agent interface . "Limit" is the number of entries shown by default. "Group" is used to restrict the access to the plugin (e. g. Group: admin;group1;group2;). "Default" determines if the plugin is enabled by default or if the user needs to enable it manually. "CacheTTLLocal" is the cache time in minutes for the plugin.

2.4.38 Frontend::Agent::View::ITSMConfigItem

ITSMConfigItem::Frontend::AgentITSMConfigItem###DefaultColumns

Columns that can be filtered in the config item overview of the agent interface. Note: Only Config Item attributes and Dynamic Fields (DynamicField_NameX) are allowed.

ITSMConfigItem::Frontend::AgentITSMConfigItem###SearchLimit

Defines the search limit for the AgentITSMConfigItem screen.

ITSMConfigItem::Frontend::AgentITSMConfigItem###DefaultCategory

The default category which is shown, if none is selected.

ITSMConfigItem::Frontend::AgentITSMConfigItem###ClassColumnsAvailable

Defines the available columns of CIs in the config item overview depending on the CI class. Each entry must consist of a class name and an array of available fields for the corresponding class. Dynamic field entries have to honor the scheme DynamicField_FieldName.

ITSMConfigItem::Frontend::AgentITSMConfigItem###ClassColumnsDefault

Defines the default displayed columns of CIs in the config item overview depending on the CI class. Each entry must consist of a class name and an array of available fields for the corresponding class. Dynamic field entries have to honor the scheme DynamicField_FieldName.

2.4.39 Frontend::Agent::View::ITSMConfigItemBulk

ITSMConfigItem::Frontend::AgentITSMConfigItemBulk###DeplState

Sets the deployment state in the configuration item bulk screen of the agent interface.

ITSMConfigItem::Frontend::AgentITSMConfigItemBulk###InciState

Sets the incident state in the configuration item bulk screen of the agent interface.

2.4.40 Frontend::Agent::View::ITSMConfigItemDelete

ITSMConfigItem::Frontend::AgentITSMConfigItemDelete###Permission

Required privileges to delete config items.

2.4.41 Frontend::Agent::View::ITSMConfigItemEdit

ITSMConfigItem::Frontend::AgentITSMConfigItemEdit###RichTextWidth

Defines the width for the rich text editor component for this screen. Enter number (pixels) or percent value (relative).

ITSMConfigItem::Frontend::AgentITSMConfigItemEdit###RichTextHeight

Defines the height for the rich text editor component for this screen. Enter number (pixels) or percent value (relative).

2.4.42 Frontend::Agent::View::ITSMConfigItemHistory

ITSMConfigItem::Frontend::HistoryOrder

Shows the config item history (reverse ordered) in the agent interface.

2.4.43 Frontend::Agent::View::ITSMConfigItemSearch

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###SearchCSVData

Data used to export the search result in CSV format.

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###SearchCSVDynamicField

Dynamic Fields used to export the search result in CSV format.

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###Defaults###SearchPreviousVersions

Defines the default shown config item search attribute for config item search screen.

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###SearchLimit

Defines the search limit for the AgentITSMConfigItemSearch screen.

ITSMConfigItem::Frontend::AgentITSMConfigItemSearch###DynamicField

Dynamic fields shown in the config item search screen of the agent interface.

2.4.44 Frontend::Agent::View::Preferences

PreferencesGroups###ConfigItemOverviewSmallPageShown

Parameters for the pages (in which the configuration items are shown).

PreferencesGroups###ConfigItemOverviewFilterSettings

Parameters for the column filters of the small config item overview. Please note: setting 'Active' to 0 will only prevent agents from editing settings of this group in their personal preferences, but will still allow administrators to edit the settings of another user's behalf. Use 'PreferenceGroup' to control in which area these settings should be shown in the user interface.

2.4.45 Frontend::Base::DynamicFieldScreens

DynamicFieldScreens###ITSMConfigurationManagement

This configuration defines all possible screens to enable or disable dynamic fields.

DefaultColumnsScreens###ITSMConfigurationManagement

This configuration defines all possible screens to enable or disable default columns.

2.4.46 Frontend::Base::Loader

Loader::Agent::CommonJS###100-ConfigurationManagement

List of JS files to always be loaded for the agent interface.

2.4.47 Frontend::Base::NavBarModule

Frontend::AdminModuleGroups###200-ITSMConfigurationManagement

Defines available groups for the admin overview screen.

2.4.48 Frontend::Customer::ITSMConfigItemOverview

ITSMConfigItem::Frontend::CustomerOverview###Small

Defines an overview module to show the small view of a configuration item list.

2.4.49 Frontend::Customer::ModuleRegistration

CustomerFrontend::Module###CustomerITSMConfigItem

Frontend module registration for the customer interface.

CustomerFrontend::Module###CustomerITSMConfigItemSearch

Frontend module registration for the customer interface.

CustomerFrontend::Module###CustomerITSMConfigItemZoom

Frontend module registration for the customer interface.

CustomerFrontend::Module###CustomerITSMConfigItemAttachment

Frontend module registration for the customer interface.

2.4.50 Frontend::Customer::ModuleRegistration::Loader

Loader::Module::CustomerITSMConfigItem###003-ITSMConfigItem

Loader module registration for the customer interface.

Loader::Module::CustomerITSMConfigItemSearch###003-ITSMConfigItem

Loader module registration for the customer interface.

Loader::Module::CustomerITSMConfigItemZoom###003-ITSMConfigItem

Loader module registration for the customer interface.

2.4.51 Frontend::Customer::ModuleRegistration::MainMenu

CustomerFrontend::Navigation###CustomerITSMConfigItem###003-ITSMConfigItem

Main menu item registration.

2.4.52 Frontend::Customer::View::ConfigItemSearch

ITSMConfigItem::Frontend::CustomerITSMConfigItemSearch###Permission

Required permissions to use the ITSM configuration item search screen in the customer interface.

ITSMConfigItem::Frontend::CustomerITSMConfigItemSearch###DeploymentState

Includes deployment states in the config item search of the customer interface.

ITSMConfigItem::Frontend::CustomerITSMConfigItemSearch###IncidentState

Includes incident states in the config item search of the customer interface.

ITSMConfigItem::Frontend::CustomerITSMConfigItemSearch###ExtendedSearchCondition

Allows extended search conditions in config item search of the customer interface. With this feature you can search e. g. config item name with this kind of conditions like "(key1*&&*key2)" or "(key1*||*key2)".

ITSMConfigItem::Frontend::CustomerITSMConfigItemSearch###SearchPageShown

Number of config items to be displayed in each page of a search result in the customer interface.

ITSMConfigItem::Frontend::CustomerITSMConfigItemSearch###SortBy::Default

Defines the default config item attribute for config item sorting of the config item search result of the customer interface.

ITSMConfigItem::Frontend::CustomerITSMConfigItemSearch###Order::Default

Defines the default config item order in the config item search result of the customer interface. Up: oldest on top. Down: latest on top.

ITSMConfigItem::Frontend::CustomerITSMConfigItemSearch###Defaults###Number

Defines the default shown config item search attribute for config item search screen.

ITSMConfigItem::Frontend::CustomerITSMConfigItemSearch###Defaults###Name

Defines the default shown config item search attribute for config item search screen.

ITSMConfigItem::Frontend::CustomerITSMConfigItemSearch###Defaults###DeploymentStateIDs

Defines the default shown config item search attribute for config item search screen.

ITSMConfigItem::Frontend::CustomerITSMConfigItemSearch###Defaults###IncidentStateIDs

Defines the default shown config item search attribute for config item search screen.

ITSMConfigItem::Frontend::CustomerITSMConfigItemSearch###Defaults###DynamicField

Defines the default shown config item search attribute for config item search screen. Example: "Key" must have the name of the Dynamic Field in this case 'X', "Content" must have the value of the Dynamic Field depending on the Dynamic Field type, Text: 'a text', Dropdown: '1', Date/Time: 'Search_DynamicField_XTimeSlotStartYear=1974; Search_DynamicField_XTimeSlotStartMonth=01; Search_DynamicField_XTimeSlotStartDay=26; Search_DynamicField_XTimeSlotStartHour=00; Search_DynamicField_XTimeSlotStartMinute=00; Search_DynamicField_XTimeSlotStartSecond=00; Search_DynamicField_XTimeSlotStopYear=2013; Search_DynamicField_XTimeSlotStopMonth=01; Search_DynamicField_XTimeSlotStopDay=26; Search_DynamicField_XTimeSlotStopHour=23; Search_DynamicField_XTimeSlotStopMinute=59; Search_DynamicField_XTimeSlotStopSecond=59;' and or 'Search_DynamicField_XTimePointFormat=week; Search_DynamicField_XTimePointStart=Before; Search_DynamicField_XTimePointValue=7';.

2.4.53 Frontend::Customer::View::ITSMConfigItem

ITSMConfigItem::Frontend::CustomerITSMConfigItem###DefaultColumns

Columns that can be filtered in the config item overview of the customer interface. Note: Only Config Item attributes and Dynamic Fields (DynamicField_NameX) are allowed.

ITSMConfigItem::Frontend::CustomerITSMConfigItem###SearchLimit

Defines the search limit for the CustomerITSMConfigItem screen.

ITSMConfigItem::Frontend::CustomerITSMConfigItem###ClassColumnsAvailable

Defines the available columns of CIs in the config item overview depending on the CI class. Each entry must consist of a class name and an array of available fields for the corresponding class. Dynamic field entries have to honor the scheme DynamicField_FieldName.

ITSMConfigItem::Frontend::CustomerITSMConfigItem###ClassColumnsDefault

Defines the default displayed columns of CIs in the config item overview depending on the CI class. Each entry must consist of a class name and an array of available fields for the corresponding class. Dynamic field entries have to honor the scheme DynamicField_FieldName.

2.4.54 Frontend::Customer::View::ITSMConfigItemOverview

ITSMConfigItem::Frontend::CustomerITSMConfigItem###DynamicField

Dynamic fields shown in the config item overview screen of the customer interface.

ITSMConfigItem::Frontend::CustomerITSMConfigItemSearch###DynamicField

Dynamic fields shown in the config item overview screen of the customer interface.

2.4.55 Frontend::Customer::View::ITSMConfigItemSearch

ITSMConfigItem::Frontend::CustomerITSMConfigItemSearch###SearchLimit

Defines the search limit for the CustomerITSMConfigItemSearch screen.

2.4.56 Frontend::Customer::View::ITSMConfigItemZoom

ITSMConfigItem::Frontend::CustomerITSMConfigItemZoom###VersionsEnabled

Customers can see historic CI versions.

ITSMConfigItem::Frontend::CustomerITSMConfigItemZoom###VersionsSelectable

Customers have the possibility to manually switch between historic CI versions.

ITSMConfigItem::Frontend::CustomerITSMConfigItemZoom###GeneralInfo

Which general information is shown in the header.

2.4.57 GenericInterface::Invoker

GenericInterface::Invoker::ConfigItemFetch::Classes

For every webservice (key) an array of classes (value) can be defined on which the import is restricted. For all chosen classes, or all existing classes the identifying attributes will have to be chosen in the invoker config.

2.4.58 GenericInterface::Invoker::ModuleRegistration

GenericInterface::Invoker::Module###Elasticsearch::ConfigItemManagement

GenericInterface module registration for the invoker layer.

GenericInterface::Invoker::Module###ConfigItem::ConfigItemFetch

GenericInterface module registration for the ConfigItemFetch invoker layer.

2.4.59 GenericInterface::Operation::ConfigItemCreate

GenericInterface::Operation::ConfigItemCreate###Permission

Defines Required permissions to create ITSM configuration items using the Generic Interface.

2.4.60 GenericInterface::Operation::ConfigItemDelete

GenericInterface::Operation::ConfigItemDelete###Permission

Defines Required permissions to delete ITSM configuration items using the Generic Interface.

2.4.61 GenericInterface::Operation::ConfigItemGet

GenericInterface::Operation::ConfigItemGet###Permission

Defines Required permissions to get ITSM configuration items using the Generic Interface.

2.4.62 GenericInterface::Operation::ConfigItemSearch

GenericInterface::Operation::ConfigItemSearch###Permission

Defines Required permissions to search ITSM configuration items using the Generic Interface.

GenericInterface::Operation::ConfigItemSearch###SearchLimit

Maximum number of config items to be displayed in the result of this operation.

GenericInterface::Operation::ConfigItemSearch###SortBy::Default

Defines the default config item attribute for config item sorting of the config item search result of this operation.

GenericInterface::Operation::ConfigItemSearch###Order::Default

Defines the default config item order in the config item search result of the this operation. Up: oldest on top. Down: latest on top.

2.4.63 GenericInterface::Operation::ConfigItemUpdate

GenericInterface::Operation::ConfigItemUpdate###Permission

Defines Required permissions to update ITSM configuration items using the Generic Interface.

2.4.64 GenericInterface::Operation::ModuleRegistration

GenericInterface::Operation::Module###ConfigItem::ConfigItemCreate

GenericInterface module registration for the operation layer.

GenericInterface::Operation::Module###ConfigItem::ConfigItemGet

GenericInterface module registration for the operation layer.

GenericInterface::Operation::Module###ConfigItem::ConfigItemUpdate

GenericInterface module registration for the operation layer.

GenericInterface::Operation::Module###ConfigItem::ConfigItemSearch

GenericInterface module registration for the operation layer.

GenericInterface::Operation::Module###ConfigItem::ConfigItemDelete

GenericInterface module registration for the operation layer.

CHAPTER 3

About

3.1 Contact

Rother OSS GmbH
Email: hello@otobo.io
Web: https://otobo.io

3.2 Version

Author: Rother OSS GmbH / Version: 11.0 / Date of release: 2024-12-16

3.3 Terms and Conditions

Published by: Rother OSS GmbH, (https://otobo.io), Oberwalting 31, 94339 Leiblfing, Germany.

CHAPTER 4

Sacrifice to Sphinx