Configuration Layer Objects Reference Guide

Introduction to the Configuration Layer Objects

4/10/2025

Contents

Introduction to the Configuration Layer Objects

> **Important**
>
> Content on this page comes from the Platform SDK Developer's Guide, but is copied here to provide a better understanding of the reference material included in this guide.

The Genesys Configuration Layer is a database containing information about the objects in your contact center environment. You may need to get information about these objects. You may also want to add, update, or delete them. The Configuration Platform SDK gives you the means to do that.

This article contains information that is common to all of these Configuration Layer objects.

General Parameters

The following parameters are common to objects of all types. They will not be described again in the listings for individual objects.

- `DBID` — An identifier of this object in the Configuration Database. Generated by Configuration Server, it is unique within an object type. Identifiers of deleted objects are not used again. Read-only.
- `state` — Current object state. Mandatory. Refer to `CfgObjectState` in section Variable Types.

> **Tip**
>
> Change in the state of a parent object will cause the states of all its child objects to change accordingly. Configuration Server will provide a notification for each elementary change. Changing the state of a parent object will not be allowed unless the client application has privileges to change all of the child objects of this parent object.

- `userProperties` — In objects, a pointer to the list of user-defined properties. In delta objects, a pointer to a list of user-defined properties added to the existing list. Parameter `userProperties` has the following structure: Each key-value pair of the primary list (`TKVList *userProperties`) uses the key for the name of a user-defined section, and the value for a secondary list, that also has the `TKVList` structure and specifies the properties defined within that section. Each key-value pair of the secondary list uses the key for the name of a user-defined property, and the value for its current setting. User properties can be defined as variables of integer, character, or binary type. Names of sections must be unique within the primary list. Names of properties must be unique within the secondary list.

> **Tip**
> Configuration Server is not concerned with logical meanings of user-defined sections, properties, or their values.

- `deletedUserProperties` — A pointer to the list of deleted user-defined properties. Has the same structure as parameter `userProperties` above. A user-defined property is deleted by specifying the name of the section that this property belongs to, and the name of the property itself with any value. A whole section is deleted by specifying the name of that section and an empty secondary list.

- `changedUserProperties` — A pointer to the list of user-defined properties whose values have been changed. Has the same structure as parameter `userProperties` above. A value of a user-defined property is changed by specifying the name of the section that this property belongs to, the name of the property itself, and a new value of that property.

- `flexibleProperties` — In objects, a pointer to the list of additional properties. In delta objects, a pointer to a list of user-defined properties added to the existing list. This parameter has the following structure: Each key-value pair of the primary list (`TKVList * flexibleProperties`) uses the key for the name of the section, and the value for a secondary list, that also has the `TKVList` structure and specifies either properties defined within that section or another section name. Each key-value pair of the secondary list uses the key for the name of a property, and the value for its current setting. Properties can be defined as variables of integer, character, or binary type or as the name of another list of properties. Names of sections must be unique within the primary list. Names of properties must be unique within the list. The data structure within the `flexibleProperties` property is object-type specific and hard-coded within Configuration Server. Each key-value in the `TKVList * flexibleProperties` is controlled and processed by Configuration Server only in the same manner as any other property in contrast with user-properties the contents of which are not Configuration Server concerned. If the structure of the property's `Extension` is not specified, the value is `NULL`. For more information, see the detailed object descriptions in this document.

Configuration Object Association

Configuration Objects can be associated with each other in a number of different ways that can be generally classified as follows:

- Parent-child relationship, where a child object cannot be created without a parent and will be deleted automatically if its parent object is deleted. Most of the object types will have an explicit reference to their parents which is marked with an asterisk in the specification below. For the object types that do not have such a reference, it is implied that their parent is the Service Provider (that is, the imaginary tenant with `DBID = 1`).
- Exclusive association, where an object cannot be associated in the same manner with more than one other object.
- Non-exclusive association, where an object can be associated in the same manner with more than one other object. Unless expressly noted otherwise, a reference to the DBID of another object without an asterisk indicates a non-exclusive assignment.

The parameters of all object-related structures are optional unless otherwise noted. However, all variables of character type must be initialized at the time an object is created. The variables of character type that are not mandatory may be initialized with an empty string (the recommended

default value unless otherwise noted). The variables of character type that are mandatory may not be initialized with an empty string. Variables of character type may accept values of up to 255 symbols in length unless otherwise noted. The recommended default value for optional parameters of other types is `zero` or `NULL`, unless otherwise noted.

Filters

Filters are used to specify more precisely the kind of information that the client application is interested in. Filters reduce both volumes of data communicated by Configuration Server and data-processing efforts on the client side. Filters are structured as key-value pairs where the value of each key defines a certain condition of data selection. Filter keys are defined as variables of integer type unless otherwise noted.

> ### Important
> Although your application can use "and" to combine multiple filters when retrieving a set of matching configuration objects, specifying a DBID value as one of the filters causes all other filters in that request to be ignored. This is by design, as only a single configuration object can match the specified DBID value. However, this behavior could create unexpected results if your application intended to use filters as a method for checking whether a known configuration object also matches additional filter values.

Here is a list of common filter types:

- `folder_dbid` — A unique identifier of a folder. If specified, Configuration Server will return information only about objects of specific type located under specified folder. See also the description of the `ConfGetObjectInfo` function.
- delegate_dbid — A unique identifier of an account on behalf of which current query is to be executed. Produced result set will be calculated using a superposition of the registered account permissions and that passed in `delegate_dbid` filter. Must be used in conjunction with delegate_type filter in order to specify account type (`CFGPerson` or `CFGAccessGroup`).
- `delegate_type` — Object type of the account (`CFGPerson` or `CFGAccessGroup`) on behalf of which the current query is to be executed. Must be used in conjunction with `delegate_dbid`.
- `object_path` — A flag that causes Configuration Server to return a full path of the object in the folder hierarchy for every object in the result set. The path string will be returned in the `cfgDescription` field of the `CFGObjectInfo` event.
- `cmp_insensitive` — A flag that causes Configuration Server to perform case-insensitive comparison of string values in the filter. Supported from Configuration Server 7.2.000.00.
- `read_folder_dbid` — A flag that causes Configuration Server to return a Folder DBID for every object in the result set. The folder will be returned in the `cfgExtraInfo3` field of the `CFGObjectInfo` event. Supported from Configuration Server 7.2.000.00.