Red Hat Enterprise Linux 10

Using IdM API

Using IdM API with Python scripts

Last Updated: 2025-11-26

Red Hat Enterprise Linux 10 Using IdM API

Using IdM API with Python scripts

Abstract

The IdM API contains examples for using various types of request. Administrators and developers can use the IdM API to write custom scripts in Python to integrate IdM with third-party applications.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting feedback through Jira (account required)

1. Log in to the Jira website.
2. Click **Create** in the top navigation bar
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

CHAPTER 1. INTRODUCTION TO IDM API

You can access the services of the Red Hat Identity Management with command-line and web-based interfaces. With the Identity Management API, you can interact with Identity Management services through the third-party applications and scripts that are written in Python.

The Identity Management API has the JavaScript Object Notation Remote Procedure Call (JSON-RPC) interface. To use the automation for various important parts, access the Identity Management API through Python. For example, you can retrieve metadata from the server with all available commands.

CHAPTER 2. BASICS OF IDM API

You can use the IdM API to automate the access to IdM environment with your custom scripts.

2.1. INITIALIZING IDM API

To use the IdM API, first initialize it in your environment.

Prerequisites

- The IdM server or IdM client package is installed.
- A valid Kerberos ticket is issued.

Procedure

1. To initialize the IdM API, include the following code in the beginning of your script:

```
from ipalib import api

api.bootstrap(context="server")
api.finalize()
```

2. To establish a connection with the LDAP server, add the following logic to your script after API initialization:

```
if api.env.in_server:
    api.Backend.ldap2.connect()
else:
    api.Backend.rpcclient.connect()
```

- If you run your script on the IdM server, this logic allows your script to connect directly to LDAP server.
- If you run your script on the IdM client, the script uses the Remote Procedure Call (RPC) client.

Additional resources

- IdM API context

2.2. RUNNING IDM API COMMANDS

You can run IdM API commands within your script. To run an IdM API command, use the **api.Command** structure in your script.

Prerequisites

- The IdM API is initialized. For more information, see Initializing IdM API.

Procedure

- For example, to list the information about user, include the following code in your script:

```
api.Command.user_show("user_name", no_members=True, all=True)
```

In this example, you also pass arguments and options to the command **user_show**.

Additional resources

- IPA API Commands

2.3. IDM API COMMANDS OUTPUT STRUCTURE

Each IdM API command has four sections for its output. These sections contain various information about the command execution.

IdM API output structure

result

This section provides the result of the command. It contains various details about the command operation, such as options and arguments which were passed to the command.

values

This section indicates the argument for the command.

messages

This section shows various information which **ipa** tool provides after the execution of the command.

summary

This section shows the summary for the operation.

In this example, your script executes the **add_user** command:

```
api.Command.user_add("test", givenname="a", sn="b")
```

The output structure of that command is below:

```
{
    "result": {
        "displayname": ["a b"],
        "objectclass": [
            "top",
            "person",
            "organizationalperson",
            "inetorgperson",
            "inetuser",
            "posixaccount",
            "krbprincipalaux",
            "krbticketpolicyaux",
            "ipaobject",
            "ipasshuser",
            "ipaSshGroupOfPubKeys",
            "mepOriginEntry",
            "ipantuserattrs",
        ],
        "cn": ["a b"],
        "gidnumber": ["1445000004"],
        "mail": ["test@ipa.test"],
        "krbprincipalname": [ipapython.kerberos.Principal("test@IPA.TEST")],
        "loginshell": ["/bin/sh"],
        "initials": ["ab"],
        "uid": ["test"],
        "uidnumber": ["1445000004"],
        "sn": ["b"],
        "krbcanonicalname": [ipapython.kerberos.Principal("test@IPA.TEST")],
        "homedirectory": ["/home/test"],
        "givenname": ["a"],
        "gecos": ["a b"],
        "ipauniqueid": ["9f9c1df8-5073-11ed-9a56-fa163ea98bb3"],
        "mepmanagedentry": [
            ipapython.dn.DN("cn=test,cn=groups,cn=accounts,dc=ipa,dc=test")
        ],
        "has_password": False,
        "has_keytab": False,
        "memberof_group": ["ipausers"],
        "dn": ipapython.dn.DN("uid=test,cn=users,cn=accounts,dc=ipa,dc=test"),
    },
    "value": "test",
    "messages": [
        {
            "type": "warning",
            "name": "VersionMissing",
            "message": "API Version number was not sent, forward compatibility not guaranteed.
Assuming server's API version, 2.248",
            "code": 13001,
            "data": {"server_version": "2.248"},
        }
    ],
    "summary": 'Added user "test"',
}
```

2.4. LISTING THE IDM API COMMANDS AND PARAMETERS

You can list information about the IdM API command and its parameters by using the commands **command_show** and **param_show**.

Prerequisites

- The IdM API is initialized. For more information, see Initializing IdM API.

Procedure

1. To display information about **user_add** command, execute the following code:

    ```
    api.Command.command_show("user_add")
    ```

    The result for this command is as follows:

    ```
    {
        "result": {
            "name": "user_add",
            "version": "1",
    ```

```
        "full_name": "user_add/1",
        "doc": "Add a new user.",
        "topic_topic": "user/1",
        "obj_class": "user/1",
        "attr_name": "add",
    },
    "value": "user_add",
    "messages": [
        {
            "type": "warning",
            "name": "VersionMissing",
            "message": "API Version number was not sent, forward compatibility not guaranteed.
Assuming server's API version, 2.251",
            "code": 13001,
            "data": {"server_version": "2.251"},
        }
    ],
    "summary": None,
}
```

2. To display information about the **givenname** parameter for the **user_add** command, execute the following code:

```
api.Command.param_show("user_add", name="givenname")
```

The result for this command is as follows:

```
{
    "result": {
        "name": "givenname",
        "type": "str",
        "positional": False,
        "cli_name": "first",
        "label": "First name",
    },
    "value": "givenname",
    "messages": [
        {
            "type": "warning",
            "name": "VersionMissing",
            "message": "API Version number was not sent, forward compatibility not guaranteed.
Assuming server's API version, 2.251",
            "code": 13001,
            "data": {"server_version": "2.251"},
        }
    ],
    "summary": None,
}
```

2.5. USING BATCHES FOR EXECUTING IDM API COMMANDS

You can execute multiple IdM API commands with a single call by using the **batch** command. The following example shows how to create multiple IdM users.

Prerequisites

Prerequisites

- The IdM API is initialized. For more information, see Initializing IdM API.

Procedure

- To create 100 IdM users in one batch, include the following code into your script:

```
batch_args = []
for i in range(100):
    user_id = "user%i" % i
    args = [user_id]
    kw = {
        'givenname' : user_id,
        'sn' : user_id
    }
    batch_args.append({
        'method' : 'user_add',
        'params' : [args, kw]
    })
ret = api.Command.batch(*batch_args)
```

2.6. IDM API CONTEXT

IdM API context determines which plug-ins the API uses. Specify the context during API initialization. For example on how to use the IdM API context, see Initializing IdM API.

IdM API context

server

Set of plug-ins which validate arguments and options that are passed to IdM API commands for execution.

client

Set of plug-ins which validate arguments and options that are forwarded to the IdM server for execution.

installer

Set of plug-ins which are specific to the installation process.

updates

Set of plug-ins which are specific to the updating process.

CHAPTER 3. IDM API AND IDM CLI COMMANDS COMPARISON

You can use the IdM API commands in the Python interactive console.

The IdM API commands are different from the **ipa** tool commands:

Command naming structure

The **ipa** CLI commands use the hyphen, as in **user-add**, but IdM API commands use the underscore instead, as in **user_add**.

Parameter naming

The parameters are different for IdM CLI commands and IdM API commands. For example, the IdM CLI **user-add** command has a parameter **first** but the IdM API **user_add** command has a parameter **givenname**.

Date format

The following date formats are available for IdM CLI:

- **%Y%m%d%H%M%SZ**
- **%Y-%m-%dT%H:%M:%SZ**
- **%Y-%m-%dT%H:%MZ**
- **%Y-%m-%dZ**
- **%Y-%m-%d %H:%M:%SZ**
- **%Y-%m-%d %H:%MZ**
  Additionally, the IdM API can use the Python built-in class **datetime**.

Useful CLI tools:

- The **console** starts an interactive Python console, which you can use to run IdM API commands.
- The **help** command shows description of the topics and the commands and includes various examples.
- The **show-mapping** command shows the mapping between CLI parameter names and LDAP attributes.

CHAPTER 4. IDM API EXAMPLE SCENARIOS

The following examples provide you with the common scenarios of using IdM API commands.

4.1. MANAGING USERS WITH IDM API COMMANDS

The examples below show common scenarios of how you can manage IdM users with the IdM API commands.

Creating an IdM user

In this example, you create an IdM user with the username **exampleuser** and the supported user **one-time password (OTP)** authentication.

```
api.Command.user_add("exampleuser", givenname="Example", sn="User",
ipauserauthtype="otp")
```

Showing an IdM user information

In this example, you display all available information about the IdM user **exampleuser**.

```
api.Command.user_show("exampleuser", all=True)
```

Modifying an IdM user

In this example, you change the e-mail address for the IdM user **exampleuser**.

```
api.Command.user_mod("exampleuser", mail="exampleuser@example.org")
```

Searching for an IdM user

In this example, you search for all IdM users that match **exampleuser** in the IdM group **admins**.

```
api.Command.user_find(criteria="exampleuser", in_group="admins")
```

Deleting an IdM user

In this example, you delete the IdM user **exampleuser**.

```
api.Command.user_del("exampleuser")
```

To restore the user in future, use the **preserve** option. If you use this option, you can restore the user with the **user_undel** command.

Adding and removing a certificate for an IdM user

You can add or remove **Base64 encoded** certificate for a user with the **user_add_cert** and **user_remove_cert** commands. In this example, you add a certificate for a user **exampleuser**.

```
args = ["exampleuser"]
kw = {
    "usercertificate": """

MIICYzCCAcygAwIBAgIBADANBgkqhkiG9w0BAQUFADAuMQswCQYDVQQGEwJVUzEMMAoGA
1UEC
hMDSUJNMREwDwYDVQQLEwhMb2NhbCBDQTAeFw05OTEyMjIwNTAwMDBaFw0wMDEyMjM
wNDU5NT

laMC4xCzAJBgNVBAYTAlVTMQwwCgYDVQQKEwNJQk0xETAPBgNVBAsTCExvY2FsIENBMIGf
MA0

GCSqGSIb3DQEBATOPA4GNADCBiQKBgQD2bZEo7xGaX2/0GHkrNFZvlxBou9v1Jmt/PDiTMPve

    8r9FeJAQ0QdvFST/0JPQYD20rH0bimdDLgNdNynmyRoS2S/IInfpmf69iyc2G0TPyRvmHIiOZ

bdCd+YBHQi1adkj17NDcWj6S14tVurFX73zx0sNoMS79q3tuXKrDsxeuwIDAQABo4GQMIGNME

sGCVUdDwGG+EIBDQQ+EzxHZW5lcmF0ZWQgYnkgdGhlIFNlY3VyZVdheSBTZWN1cml0eSBTZ
XJ

2ZXIgZm9yIE9TLzM5MCAoUkFDRikwDgYDVR0PAQH/BAQDAgAGMA8GA1UdEwEB/wQFMAMB
Af8w

HQYDVR0OBBYEFJ3+ocRyCTJw067dLSwr/nalx6YMMA0GCSqGSIb3DQEBBQUAA4GBAMaQzt
+za
    j1GU77yzlr8iiMBXgdQrwsZZWJo5exnAucJAEYQZmOfyLiMD6oYq+ZnfvM0n8G/Y79q8nhwvu
    xpYOnRSAXFp6xSkrIOeZtJMY1h00LKp/JX3Ng1svZ2agE126JHsQ0bhzN5TKsYfbwfTwfjdWA
    Gy6Vf1nYi/rO+ryMO
  """
}

api.Command.user_add_cert(*args, **kw)
```

Enabling and disabling an IdM user

You can enable or disable an IdM user with the **user_enable** and **user_disable** commands. In this example, you disable the IdM user **exampleuser**.

```
api.Command.user_disable("exampleuser")
```

4.2. MANAGING GROUPS WITH IDM API COMMANDS

The examples below show common scenarios of how you can manage IdM groups with the IdM API commands.

Creating an IdM group

In this example, you create an IdM group **developers**, with a specified Group ID number.

```
api.Command.group_add("developers", gidnumber=500, description="Developers")
```

Adding a user as a member to an IdM group

In this example, you add the **admin** user to the **developers** group.

```
api.Command.group_add_member("developers", user="admin")
```

Adding a service as a member to an IdM group

In this example, you add the **HTTP/server.ipa.test** service to the **developers** group.

```
api.Command.group_add_member("developers", service="HTTP/server.ipa.test")
```

Adding a group as a subgroup to an IdM group

In this example, you add another group, **admins**, to the **developers** group.

```
api.Command.group_add_member("developers", group="admins")
```

Adding IdM group managers

In this example, you add the **bob** user as a group manager for the **developers** group.

```
api.Command.group_add_member_manager("developers", user="bob")
```

Finding an IdM group

You can search for an IdM group using various parameters. In this example, you find all groups that the user **bob** is managing.

```
api.Command.group_find(membermanager_user="bob")
```

Displaying IdM group information

In this example, you display group information about the **developers** group, without the members list.

```
api.Command.group_show("developers", no_members=True)
```

Modifying an IdM group

In this example, you convert a non-POSIX group **testgroup** to a POSIX group.

```
api.Command.group_mod("testgroup", posix=True)
```

Removing members from an IdM group

In this example, you remove the **admin** user from the **developers** group.

```
api.Command.group_remove_member("developers", user="admin")
```

Removing IdM group managers

In this example, you remove the user **bob** as a manager from the **developers** group.

```
api.Command.group_remove_member_manager("developers", user="bob")
```

Removing an IdM group

In this example, you remove the **developers** group.

```
api.Command.group_del("developers")
```

4.3. MANAGING ACCESS CONTROL WITH IDM API COMMANDS

The examples below show common scenarios of how you can manage access control with the IdM API commands.

Adding a permission for creating users

In this example, you add a permission for creating users.

```
api.Command.permission_add("Create users", ipapermright='add', type='user')
```

Adding a permission for managing group membership

In this example, you add a permission for adding users to groups.

```
api.Command.permission_add("Manage group membership", ipapermright='write', type='group',
attrs="member")
```

Adding a privilege for the user creation process

In this example, you add a privilege for creating users, adding them to groups, and managing user certificates.

```
api.Command.permission_add("Create users", ipapermright='add', type='user')
api.Command.permission_add("Manage group membership", ipapermright='write', type='group',
attrs="member")
api.Command.permission_add("Manage User certificates", ipapermright='write', type='user',
attrs='usercertificate')

api.Command.privilege_add("User creation")
api.Command.privilege_add_permission("User creation", permission="Create users")
api.Command.privilege_add_permission("User creation", permission="Manage group
membership")
api.Command.privilege_add_permission("User creation", permission="Manage User certificates")
```

Adding a role using a privilege

In this example, you add a role using the privilege created in the previous example.

```
api.Command.role_add("usermanager", description="Users manager")
api.Command.role_add_privilege("usermanager", privilege="User creation")
```

Assigning a role to a user

In this example, you assign the **usermanager** role to the user **bob**.

```
api.Command.role_add_member("usermanager", user="bob")
```

Assigning a role to a group

In this example, you assign the **usermanager** role to the **managers** group.

```
api.Command.role_add_member("usermanager", group="managers")
```

4.4. MANAGING SUDO RULES WITH IDM API COMMANDS

The examples below show common scenarios of how you can manage sudo rules with the IdM API commands.

Creating a sudo rule

In this example, you create a sudo rule that holds time change commands.

```
api.Command.sudorule_add("timechange")
```

Creating a sudo command

In this example, you create the **date** sudo command.

```
api.Command.sudocmd_add("/usr/bin/date")
```

Attaching a sudo command to a sudo rule

In this example, you attach the **date** sudo command to the **timechange** sudo rule.

```
api.Command.sudorule_add_allow_command("timechange", sudocmd="/usr/bin/date")
```

Creating and attaching groups of sudo commands

In this example, you create multiple sudo commands, add them to a newly created **timecmds** sudo command group, and attach the group to the **timechange** sudo rule.

```
api.Command.sudocmd_add("/usr/bin/date")
api.Command.sudocmd_add("/usr/bin/timedatectl")
api.Command.sudocmd_add("/usr/sbin/hwclock")
api.Command.sudocmdgroup_add("timecmds")
api.Command.sudocmdgroup_add_member("timecmds", sudocmd="/usr/bin/date")
api.Command.sudocmdgroup_add_member("timecmds", sudocmd="/usr/bin/timedatectl")
api.Command.sudocmdgroup_add_member("timecmds", sudocmd="/usr/sbin/hwclock")
api.Command.sudorule_add_allow_command("timechange", sudocmdgroup="timecmds")
```

Denying sudo commands

In this example, you deny the **rm** command to be run as sudo.

```
api.Command.sudocmd_add("/usr/bin/rm")
api.Command.sudorule_add_deny_command("timechange", sudocmd="/usr/bin/rm")
```

Adding a user to a sudo rule

In this example, you add the user **bob** to the **timechange** sudo rule.

```
api.Command.sudorule_add_user("timechange", user="bob")
```

Making a sudo rule available only for a specified host

In this example, you restrict the **timechange** rule to be available only for the **client.ipa.test** host.

```
api.Command.sudorule_add_host("timechange", host="client.ipa.test")
```

Setting sudo rules to be run as a different user

By default, sudo rules are run as **root**. In this example, you set the **timechange** sudo rule to be run as the **alice** user instead.

```
api.Command.sudorule_add_runasuser("timechange", user="alice")
```

Setting sudo rules to be run as a group

In this example, you set the **timechange** sudo rule to be run as the **sysadmins** group.

```
api.Command.sudorule_add_runasgroup("timechange", group="sysadmins")
```

Setting a sudo option for a sudo rule

In this example, you set a sudo option for the **timechange** sudo rule.

```
api.Command.sudorule_add_option("timechange", ipasudoopt="logfile='/var/log/timechange_log'")
```

Enabling a sudo rule

In this example, you enable the **timechange** sudo rule.

```
api.Command.sudorule_enable("timechange")
```

Disabling a sudo rule

In this example, you disable the **timechange** sudo rule.

```
api.Command.sudorule_disable("timechange")
```

4.5. MANAGING HOST-BASED ACCESS CONTROL WITH IDM API COMMANDS

The examples below show common scenarios of how you can manage Host-based Access Control (HBAC) with the IdM API commands.

Creating an HBAC rule

In this example, you create a base rule that will handle SSH service access.

```
api.Command.hbacrule_add("sshd_rule")
```

Adding a user to an HBAC rule

In this example, you add the user **john** to the **sshd_rule** HBAC rule.

```
api.Command.hbacrule_add_user("sshd_rule", user="john")
```

Adding a group to an HBAC rule

In this example, you add the group **developers** to the **sshd_rule** HBAC rule.

```
api.Command.hbacrule_add_user("sshd_rule", group="developers")
```

Removing a user from an HBAC rule

In this example, you remove the user **john** from the **sshd_rule** HBAC rule.

```
api.Command.hbacrule_remove_user("sshd_rule", user="john")
```

Registering a new target HBAC service

You must register a target service before you can attach it to an HBAC rule. In this example, you register the **chronyd** service.

```
api.Command.hbacsvc_add("chronyd")
```

Attaching a registered service to an HBAC rule

In this example, you attach the **sshd** service to the **sshd_rule** HBAC rule. This service is registered in IPA by default, so there is no need to register it using **hbacsvc_add** beforehand.

```
api.Command.hbacrule_add_service("sshd_rule", hbacsvc="sshd")
```

Adding a host to an HBAC rule

In this example, you add **workstations** host group to the **sshd_rule** HBAC rule.

```
api.Command.hbacrule_add_host("sshd_rule", hostgroup="workstations")
```

Testing an HBAC rule

In this example, you use the **sshd_rule** HBAC rule against the **workstation.ipa.test** host. It targets the service **sshd** that comes from the user **john**.

```
api.Command.hbactest(user="john", targethost="workstation.ipa.test", service="sshd",
rules="sshd_rule")
```

Enabling an HBAC rule

In this example, you enable the **sshd_rule** HBAC rule.

```
api.Command.hbacrule_enable("sshd_rule")
```

Disabling an HBAC rule

In this example, you disable the **sshd_rule** HBAC rule.

```
api.Command.hbacrule_disable("sshd_rule")
```

CHAPTER 5. AUDITING IDM API OPERATIONS

Identity Management (IdM) servers use the **systemd** journal to create audit records of all IdM API operations. To audit operations and troubleshoot issues, you can query the journal to see who performed actions, when, and on which server.

5.1. OVERVIEW OF IDM API AUDITING

An IdM server records any use of the Identity Management (IdM) API in the **systemd** journal. This provides a unified method to collect logs for auditing API operations.

The **systemd** journal allows for centralized collection of logs from individual systems, which can then be queried and filtered.

Each log entry is tagged with an **IPA.API** marker and contains the following details in a structured format:

- The authenticated Kerberos principal that performed the action, or **` if the operation was performed by the `root** user directly on the server through LDAPI.
- The name of the API command that was executed.
- The result of the execution, which is either **SUCCESS** or an exception name.
- An LDAP backend instance identifier, which is the same for all operations performed as part of the same request.
- A list of arguments and options passed to the command, in JSON format.

You can query these entries with the **journalctl** utility. Using **journalctl** with the **-x** option provides a more detailed, human-readable explanation of the log entry, including links to the relevant documentation.

NOTE

All IdM API audit entries have a **MESSAGE_ID** property set to the application UID **6d70f1b493df36478bc3499257cd3b17**.

5.2. VIEWING THE IDM API AUDIT LOGS

You can view the IdM API audit logs and details of a specific entry by querying the **systemd** journal. This procedure shows how to identify and display logs of a user deletion using the IdM API.

Prerequisites

- You have root access to the IdM server.

Procedure

1. To see a list of all IdM API operations recorded in the journal, filter the journal for the **IPA.API** marker:

```
# journalctl -g IPA.API
May 23 10:30:15 idmserver.idm.example.com /usr/bin/ipa[247422]: [IPA.API] [autobind]:
```

```
user_del: SUCCESS [ldap2_140328582446688] {"uid": ["example_user"], "continue": false,
"version": "2.253"}
May 23 10:32:01 idmserver.idm.example.com /usr/bin/ipa[247555]: [IPA.API]
admin@IDM.EXAMPLE.COM: user_add: SUCCESS [ldap2_140328582446999] {"uid":
["new_user"], "givenname": "New", "sn": "User", "cn": "New User"}
May 23 10:33:10 idmserver.idm.example.com /mod_wsgi[247035]: [IPA.API]
admin@IDM.EXAMPLE.COM: ping: SUCCESS [ldap2_139910420944784] {"version":
"2.253"}
May 23 10:34:05 idmserver.idm.example.com /usr/bin/ipa[247888]: [IPA.API] [autobind]:
group_add_member: SUCCESS [ldap2_140328582447111] {"cn": "admins", "user":
"new_user"}
```

The output shows a summary of each API call, including the user, the command, the result, the unique connection ID, and the parameters used.

2. Identify the unique identifier for the specific entry you want to inspect. For example, the **user_del** call has the LDAP backend instance identifier **ldap2_140328582446688**.

3. Use **journalctl** with the **-x** option and the unique identifier value to get a detailed explanation of the user deletion log entry:

```
# journalctl -x -g ldap2_140328582446688
May 23 10:30:15 idmserver.idm.example.com /usr/bin/ipa[255232]: [IPA.API] [autobind]:
user_del: SUCCESS [ldap2_140328582446688] {"uid": ["example_user"], "continue": false,
"version": "2.253"}
-- Subject: IdM API command was executed and result of its execution was audited
-- Defined-by: FreeIPA
-- Support: https://lists.fedorahosted.org/archives/list/freeipa-users@lists.fedorahosted.org/
-- Documentation: man:ipa(1)
-- Documentation: https://freeipa.readthedocs.io/en/latest/api/index.html
-- Documentation: https://freeipa.readthedocs.io/en/latest/api/user_del.html

-- Identity Management provides an extensive API that allows to manage all aspects of IdM
deployments.

-- The following information about the API command executed is available:

-- [IPA.API] [autobind]: user_del: SUCCESS [ldap2_140328582446688] {"uid":
["example_user"], "continue": false, "version": "2.253"}

-- The command was executed by '/usr/bin/ipa' utility. If the utility name
-- is '/mod_wsgi`, then this API command came from a remote source through the IdM
-- API end-point.

-- The message includes following fields:

--   - executable name and PID ('/mod_wsgi' for HTTP end-point; in this case it
--     was '/usr/bin/ipa' command)

--   - '[IPA.API]' marker to allow searches with 'journalctl -g IPA.API'

--   - authenticated Kerberos principal or '[autobind]' marker for LDAPI-based
--     access as root. In this case it was '[autobind]'

--   - name of the command executed, in this case 'user_del'
--   - result of execution: SUCCESS or an exception name. In this case it was
--     'SUCCESS'

--   - LDAP backend instance identifier. The identifier will be the same for all
--     operations performed under the same request. This allows to identify operations
--     which were executed as a part of the same API request instance. For API
--     operations that didn't result in LDAP access, there will be
--     '[no_connection_id]' marker.

--   - finally, a list of arguments and options passed to the command is provided
--     in JSON format.

-- ---------
-- The following list of arguments and options were passed to the command
-- 'user_del' by the '[autobind]' actor:
--
-- {"uid": ["example_user"], "continue": false, "version": "2.253"}
-- ---------

-- A detailed information about Identity Management API can be found at upstream
documentation API reference:
-- https://freeipa.readthedocs.io/en/latest/api/index.html

-- For details on the IdM API command 'user_del' see
-- https://freeipa.readthedocs.io/en/latest/api/user_del.html
```