

Zusammenfassung *(automatisiert erstellt)*

Beim Starten von Containern in XNAT treten häufig Fehler auf, die durch falsche Verwendung von IDs statt Namen, unkorrekten Ressourcen, Matcher-Problemen oder fehlenden Projektaktivierungen entstehen. Das Problem liegt meist darin, dass die API in URLs und Payloads den Namen (String) anstelle der internen numerischen IDs erwartet. Debugging erfolgt durch manuelles Testen (curl/Postman), genaue Kontrolle der Ressourcendaten (Label, Filename, Endung), sowie durch Ausgaben im Skript. Das erneute Aktivieren des Wrappers mit Namen statt IDs, das Prüfen der Ressourcenstruktur und das Sicherstellen der Projektzugehörigkeit sind essentielle Schritte. Ein systematisches Logging, Matcher-Test (mit „true“) und API-Check helfen, Fehler zu lokalisieren. Das Tagging/Pushen von Docker-Images ist nur bei externen Registry-Operationen relevant, nicht bei lokalen Builds.

Systemprompt

Du bist ein intelligentes und unterstützendes KI-Assistenzsystem für alle Hochschulangehörigen der HAWK Hildesheim/Holzminden/Göttingen. Dein Ziel ist es, Studierende, Lehrende, Forschende und Mitarbeitende in ihrer akademischen Arbeit, beim Lernen, Forschen, Lehren und verwalterischen Aufgaben zu unterstützen. Dabei förderst du kollaboratives Arbeiten, wissenschaftliches Denken und eine kreative Problemlösung. Beziehe dich auf wissenschaftliche Methoden und Theorien, argumentiere sachlich und reflektiere kritisch. Sei objektiv und verzichte auf unbegründete Meinungen. Fördere akademische Integrität und unterstütze keine Plagiate. Sei inklusiv, wertschätzend und respektiere Vielfalt.

Verlauf:

Bousfiha, Tanae:

tanae@mi0082n:~/dockerfile/xnat-osa/Ziel1_osaDocker

Container\$ python3 Containerenabled.py

XNAT WEB URL:https://xnat-dev.gwdg.de

XNAT Username:tanae

XNAT Password:

Project ID:tanae_osa_Predct

Session ID:XNAT_E00428

Path to the Python script:Bonjour.py

Docker Name base image:python:3.10

What is the name of the command in XNAT:

datasciencerec

What is the description of the command in XNAT:

dmcftfix

What is the name of the Label in XNAT: ficvhtsd

What is the description of the Label in XNAT: dtascien

Dockerfile written to ./Dockerfile

Building Docker image

'datasciencerec:latest'...

Docker image 'datasciencerec:latest' built successfully.

JSON file created at command.json

Uploading command to

https://xnat-dev.gwdg.de/xapi/commands

Command created successfully.

No wrapper found for this command.

Wrapper existiert noch nicht, wird erstellt...

Wrapper-Payload: {

 "name":

 "datasciencerec_wrapper",

 "label": "ficvhtsd",

 "description": "dtascien",

 "contexts": [

 "xnat:mrSessionData"

],

 "outputs": [

```

{
  "name": "result_file",
  "description": "Result",
  "required": true,
  "mount": "output_mount",
  "path": "result.csv"
}
],
"external-inputs": [
  {
    "name": "session",
    "type": "Session",
    "required": true,
    "load-children": true
  }
],
"derived-inputs": [
  {
    "name": "csv_resource",
    "type": "Resource",
    "matcher": "@.label ==
&#039;CSV&#039;",
    "required": true,
    "provides-files-for-command-mount":
"input_mount",
    "load-children": true,
    "derived-from-wrapper-input":
"session",
    "multiple": false
  },
  {
    "name": "input_file",
    "type": "File",
    "matcher": "@.name =~
\".*\\.(csv|tsv|txt)$\"",
    "required": true,
    "load-children": true,
    "derived-from-wrapper-input":

```

```

    &quot;csv_resource&quot;;,
    &quot;multiple&quot;: false
  },
  {
    &quot;name&quot;: &quot;input_file_name&quot;,
    &quot;type&quot;: &quot;string&quot;,
    &quot;required&quot;: true,
    &quot;load-children&quot;: true,
    &quot;derived-from-wrapper-input&quot;:
&quot;input_file&quot;;,
    &quot;multiple&quot;: false
  }
],
&quot;output-handlers&quot;: [
  {
    &quot;name&quot;: &quot;output&quot;,
    &quot;accepts-command-output&quot;:
&quot;result_file&quot;;,
    &quot;as-a-child-of&quot;: &quot;session&quot;;,
    &quot;type&quot;: &quot;Resource&quot;;,
    &quot;label&quot;: &quot;Results&quot;;,
    &quot;format&quot;: &quot;csv&quot;;
  }
]
}

```

Wrapper creation failed: 400 - Invalid command:

Command "datasciencerec" - wrapper
name "datasciencerec_wrapper" is not
unique.

Wrapper konnte nicht erstellt werden.

mit dem Skript

```

#-----Bibliotheken-----
-----

```

```
import datetime
```

```
import json
```

```
import requests # https://wiki.xnat.org/container-service/
container-service-api
```

```
import os
```

```
import subprocess #
https://www.datacamp.com/tutorial/python-subprocess
import getpass
import sys
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequ
estWarning)
```

```
#-----erstmal Skript from the users
Fragen-----
```

```
def check_user_skript(skript_path):
    if not os.path.isfile(skript_path):
        print(f"Datei nicht gefunden:
{skript_path}",)
        return False
    if not skript_path.endswith(".py"):
        print("Datei nicht korrekt, bitte eine .py Datei
angeben.",)
        return False
    return True
```

```
#-----dockerfile
ausfüllen-----
```

```
def write_dockerfile(docker_dir, script_filename,
docker_base_image):
    dockerfile_content = f"&quot;&quot;FROM
{docker_base_image}
```

```
WORKDIR /app
```

```
COPY {script_filename} /app/{script_filename}
```

```
ENTRYPOINT ["python3",
"/app/{script_filename}"]
CMD ["example.csv",
```

```

"/app/output"]
"""
    os.makedirs(docker_dir, exist_ok=True)
    dockerfile_path = os.path.join(docker_dir,
"Dockerfile")
    with open(dockerfile_path, "w") as f:
        f.write(dockerfile_content)
    print(f"Dockerfile written to
{dockerfile_path}")
    return dockerfile_path

```

```

#-----Image
bauen-----
def build_docker_image(dockerfile_path,
docker_image_name):
    build_command = [
        "docker", "build",
        "-f", dockerfile_path, "-t",
        docker_image_name, "."
    ]
    print(f"Building Docker image
#{docker_image_name}#...")
    result = subprocess.run(build_command,
capture_output=True, text=True)
    if result.returncode == 0:
        print(f"Docker image
#{docker_image_name}# built
successfully.")
    else:
        print(f"Failed to build Docker image:
{result.stderr}")
        exit(1)

```

```

#-----User-Input-----
-----
def get_input(prompt):
    while True:
        value = input(prompt)

```

```

    if value.strip():
        return value
    else:
        print(""Cannot be empty."")

```

```

def modification():
    data = {}
    data["command_name"] =
get_input("What is the name of the command in
XNAT: ".)
    data["command_description"] =
get_input("What is the description of the command
in XNAT: ".)
    data["label_name"] =
get_input("What is the name of the Label in XNAT:
".)
    data["label_description"] =
get_input("What is the description of the Label in
XNAT: ".)
    return data

```

#-----json File

erstellen-----

```

def create_json_file(docker_image, script_filename,
mod_data):
    wrapper_name =
mod_data["command_name"].replace("
", "_").lower() + "_wrapper"
    json_file = {
        "name":
mod_data["command_name"],
        "description":
mod_data["command_description"],
        "version": "1.0",
        "image": docker_image,
        "type": "docker",
        "command-line": f"python
/app/{script_filename} /app/input/#INPUT_FILE#

```

```

/app/output";,
    "mounts": [
        {"name": "output_mount",
        "writable": True, "path":
        "/app/output"},
        {"name": "input_mount",
        "writable": False, "path":
        "/app/input"}
    ],
    "inputs": [
        {
            "name": "INPUT_FILE",
            "description":
mod_data["command_description"],
            "type": "string",
            "required": True
        }
    ],
    "outputs": [
        {
            "name": "result_file",
            "description": "Result",
            "required": True,
            "mount":
            "output_mount",
            "path": "result.csv"
        }
    ],
    "xnat": [
        {
            "name": wrapper_name,
            "label":
mod_data["label_name"],
            "description":
mod_data["label_description"],
            "contexts":
["xnat:mrSessionData"],
            "external-inputs": [

```



```

        {&quot;name&quot;:: &quot;session&quot;,
&quot;type&quot;:: &quot;Session&quot;,
&quot;required&quot;:: True, &quot;load-children&quot;::
True}

    ],
    &quot;derived-inputs&quot;:: [
        {
            &quot;name&quot;::
&quot;csv_resource&quot;,
            &quot;type&quot;:: &quot;Resource&quot;,
            &quot;matcher&quot;:: &quot;@.label ==
&#039;CSV&#039;&quot;,
            &quot;required&quot;:: True,

&quot;provides-files-for-command-mount&quot;::
&quot;input_mount&quot;,
            &quot;load-children&quot;:: True,
            &quot;derived-from-wrapper-input&quot;::
&quot;session&quot;
        },
        {
            &quot;name&quot;::
&quot;input_file&quot;,
            &quot;type&quot;:: &quot;File&quot;,
            &quot;matcher&quot;:: &quot;@.name =~
&quot;.*\\.(csv|tsv|txt)$\\&quot;&quot;,
            &quot;required&quot;:: True,
            &quot;load-children&quot;:: True,
            &quot;derived-from-wrapper-input&quot;::
&quot;csv_resource&quot;
        },
        {
            &quot;name&quot;::
&quot;input_file_name&quot;,
            &quot;type&quot;:: &quot;string&quot;,
            &quot;derived-from-wrapper-input&quot;::
&quot;input_file&quot;
        }
    ]
}

```

```

    ],
    "output-handlers": [
        {
            "name": "output",
            "accepts-command-output":
"result_file",
            "as-a-child-of":
"session",
            "type": "Resource",
            "label": "Results",
            "format": "csv"
        }
    ]
}
]
}
]
}
#-----
with open("command.json",
"w") as json_out:
    json.dump(json_file, json_out, indent=4)
    print(f"JSON file created at
command.json")
    return "command.json"

#-----Command zu XNAT
senden-----
def send_json_to_xnat(json_file_path, xnat_url,
xnat_user, xnat_password):
    url = f"{xnat_url}/xapi/commands"
    print(f"Uploading command to {url}")
    with open(json_file_path, "r") as f:
        response = requests.post(url, auth=(xnat_user,
xnat_password), json=json.load(f))
        if response.status_code == 200:
            print("Command uploaded
successfully.")
        elif response.status_code == 201:
            print("Command created successfully.")

```

```

elif response.status_code == 409:
    print("&quot;Command already exists.&quot;")
else:
    print(f"&quot;Failed to upload command:
{response.status_code} - {response.text}&quot;")

#-----Command-Liste-----
-----
def get_command_id_by_name(xnat_host, xnat_user,
xnat_password, command_name):
    url = f"&quot;{xnat_host.rstrip(&#039;/&#039;)}&quot;/xapi/co
mmands&quot;";
    resp = requests.get(url, auth=(xnat_user,
xnat_password), verify=False)
    if resp.status_code != 200:
        print(f"&quot;Error fetching commands:
{resp.status_code}&quot;")
        sys.exit(1)
    data = resp.json()
    if isinstance(data, dict) and "&quot;commands&quot;" in
data:
        command_list = data["commands"]
    else:
        command_list = data
    for command in command_list:
        if command.get("&quot;name&quot;") ==
command_name:
            return command["id"]
    print("&quot;Command not found.&quot;")
    sys.exit(1)

#-----Wrapper
auslesen/erstellen-----
def get_command_io(xnat_host, xnat_user,
xnat_password, command_id):
    url = f"&quot;{xnat_host.rstrip(&#039;/&#039;)}&quot;/xapi/co
mmands/{command_id}&quot;";
    resp = requests.get(url, auth=(xnat_user,

```

```

xnat_password), verify=False)
    if resp.status_code != 200:
        print(f'Fehler beim Abrufen des Commands:
{resp.status_code}')
        sys.exit(1)
    cmd = resp.json()
    outputs = cmd.get('outputs', [])
    external_inputs = []
    derived_inputs = []
    for wrapper in cmd.get('xnat', []):
        external_inputs =
wrapper.get('external-inputs', [])
        derived_inputs =
wrapper.get('derived-inputs', [])
        break # Nur den ersten Wrapper verwenden
    return outputs, external_inputs, derived_inputs

```

```

def get_wrapper_id_by_command_name(xnat_host,
xnat_user, xnat_password, wrapper_name):
    url = f'{xnat_host.rstrip("/")}/xapi/co
mmands'
    resp = requests.get(url, auth=(xnat_user,
xnat_password), verify=False)
    if resp.status_code != 200:
        print(f'Error fetching commands:
{resp.status_code}')
        sys.exit(1)
    for command in resp.json():
        for wrapper in command.get('wrappers',
[]):
            if wrapper.get('name') ==
wrapper_name:
                return wrapper['id']
    print('No wrapper found for this
command.')
    sys.exit(1)

```

```

def create_wrapper(xnat_host, command_id,

```

```

wrapper_name, label_name, description, xnat_user,
xnat_password, outputs, external_inputs,
derived_inputs):
    url = f"{xnat_host.rstrip('/')}/xapi/commands/{command_id}/wrappers"
    output_handlers = [
        "name": wrapper_name,
        "accepts-command-output":
    "result_file",
        "as-a-child-of": "session",
        "type": "Resource",
        "label": "Results",
        "format": "csv"
    ]
    wrapper = {
        "name": wrapper_name,
        "label": label_name,
        "description": description,
        "contexts":
    ["xnat:mrSessionData"],
        "outputs": outputs,
        "external-inputs": external_inputs,
        "derived-inputs": derived_inputs,
        "output-handlers": output_handlers
    }
    print("Wrapper-Payload:",
json.dumps(wrapper, indent=2))
    resp = requests.post(url, auth=(xnat_user,
xnat_password),
        headers={"Content-Type":
    "application/json"},
        json=wrapper, verify=False)
    # Korrekte Behandlung von Status-Code:
    if resp.status_code == 201:
        wrapper_id = resp.text.strip()
        print(f"Wrapper created successfully. ID:
    {wrapper_id}")
        return wrapper_id

```

```

elif resp.status_code == 200:
    print("&quot;Wrapper created successfully
(200).&quot;")
    # extrahiere ggf. return-id
elif resp.status_code == 409:
    print("&quot;Wrapper already exists.&quot;")
    return None
else:
    print(f"&quot;Wrapper creation failed:
{resp.status_code} - {resp.text}&quot;")
    return None

```

#-----Wrapper

Aktivierung-----

```

def enable_wrapper(xnat_host, project_id,
command_name, wrapper_id, xnat_user,
xnat_password):
    enable_url = f"&quot;{xnat_host.rstrip(&#039;/&#039;)} /
xapi/projects/{project_id}/commands/{command_name}/
wrappers/{wrapper_id}/enable&quot;";
    print(f"&quot;Enable-URL: {enable_url}&quot;") #
Debug!
    resp = requests.post(enable_url, auth=(xnat_user,
xnat_password), verify=False)
    if resp.status_code == 200:
        print("&quot;Wrapper successfully enabled.&quot;")
    else:
        print(f"&quot;Failed to enable wrapper:
{resp.status_code} - {resp.text}&quot;")

```

#-----Container in XNAT

starten-----

```

def run_container_in_xnat(xnat_host, wrapper_id,
session_id, xnat_user, xnat_password):
    url =
f"&quot;{xnat_host}/xapi/container-launcher/launch&quot;";

```

```

headers = {'Content-Type':,
'application/json'}

payload = {
    'command': wrapper_id,
    'wrapper': wrapper_id,
    'inputs': {
        'session': session_id
    }
}

print('Launching container with payload:',
json.dumps(payload, indent=2)) # Debug!
response = requests.post(url, auth=(xnat_user,
xnat_password), headers=headers, json=payload,
verify=False)

if response.status_code in [200, 201]:
    print('Container launched
successfully.')
else:
    print('Failed to launch container:
{response.status_code} - {response.text}')

#-----Main
Teil-----

```

```

def main():
    xnat_host = get_input('XNAT WEB URL:')
    xnat_user = get_input('XNAT Username:')
    xnat_password = getpass.getpass('XNAT
Password: ')
    project_id = get_input('Project ID:')
    session_id = get_input('Session ID:')
    script_path = get_input('Path to the Python
script:')
    docker_base_image = get_input('Docker Name
base image:')

```

```

if not check_user_skript(script_path):
    return

mod_data = modification()

# Stabiler Wrapper-Name (ohne Zeitstempel)
wrapper_name =
mod_data[&quot;command_name&quot;].replace(&quot;
&quot;, &quot;_&quot;).lower() + &quot;_wrapper&quot;

# Dockerfile erstellen und Image bauen
dockerfile_path = write_dockerfile(&quot;.&quot;,
os.path.basename(script_path), docker_base_image)
docker_image_name = f&quot;{mod_data[&#039;com
mand_name&#039;].lower().replace(&#039; &#039;,
&#039;_&#039;)}:latest&quot;
build_docker_image(dockerfile_path,
docker_image_name)

# JSON erstellen und an XNAT senden
json_file_path =
create_json_file(docker_image_name,
os.path.basename(script_path), mod_data)
send_json_to_xnat(json_file_path, xnat_host,
xnat_user, xnat_password)

# Wrapper anlegen oder holen
try:
    wrapper_id =
get_wrapper_id_by_command_name(xnat_host,
xnat_user, xnat_password, wrapper_name)
    print(f&quot;Wrapper existiert bereits:
{wrapper_id}&quot;.)
except SystemExit:
    print(&quot;Wrapper existiert noch nicht, wird
erstellt...&quot;.)
    command_id =
get_command_id_by_name(xnat_host, xnat_user,

```



```

xnat_password,
mod_data["command_name"])
    outputs, external_inputs, derived_inputs =
get_command_io(xnat_host, xnat_user, xnat_password,
command_id)
    wrapper_id = create_wrapper(
        xnat_host, command_id, wrapper_name,
        mod_data["label_name"],
mod_data["label_description"],
        xnat_user, xnat_password,
        outputs, external_inputs, derived_inputs
    )
    if not wrapper_id:
        print(""Wrapper konnte nicht erstellt
werden."")
        return

# Wrapper aktivieren
enable_wrapper(xnat_host, project_id,
mod_data["command_name"], wrapper_id,
xnat_user, xnat_password)

# Container starten
run_container_in_xnat(xnat_host, wrapper_id,
session_id, xnat_user, xnat_password)

if __name__ == "__main__":
    main()

```

HAWKI (gpt-4.1):

Deine Situation und die Fehlermeldung

Du bekommst beim Anlegen eines Wrappers in XNAT folgenden Fehler:

...

Wrapper creation failed: 400 - Invalid command:

Command "datasciencerec" - wrapper name "datasciencerec_wrapper" is not unique.

Wrapper konnte nicht erstellt werden.

Das bedeutet: ****Ein Wrapper mit diesem Namen (datasciencerec_t_wrapper) existiert bereits für das XNAT-Command.****

Erklärung

- In XNAT ist der Name des Wrappers pro Command eindeutig.
- Dein Skript versucht denselben Wrapper ("datasciencerec_t_wrapper") erneut anzulegen, was zu einem HTTP 400 (Bad Request) führt.
- Sofern du denselben Wrapper (mit gleichem Namen) erneut anlegen willst, wird immer dieser Fehler auftreten.

Lösungsmöglichkeiten

****1. Wrapper NICHT neu anlegen, sondern weiterverwenden.****

Falls du deinen Command (und Wrapper) testest und nicht jedes Mal einen neuen benötigst, solltest du bei Existenz einfach den bestehenden Wrapper weiterverwenden.

Dein Code versucht das bereits, aber ``get_wrapper_id_by_command_name`` funktioniert nur, wenn das Command die Struktur ``{"wrappers": [...]}`` hat. XNATs Command-API kann aber auch flacher sein.

****Lösung (sicherer Zugriff auf den Wrapper):****

Ersetze

```
```python
```

```
def get_wrapper_id_by_command_name(xnat_host, xnat_user, xnat_password,
wrapper_name):
```

```
 url = f"{xnat_host.rstrip('/')}/xapi/commands"
```

```
 resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
```

```
 if resp.status_code != 200:
```

```
 print(f"Error fetching commands: {resp.status_code}")
```

```
 sys.exit(1)
```

```
 for command in resp.json():
```

```
 for wrapper in command.get("wrappers", []):
```

```
 if wrapper.get("name") == wrapper_name:
```

```
 return wrapper["id"]
```

```

 print("No wrapper found for this command.")
 sys.exit(1)
...

durch
```python
def get_wrapper_id_by_command_name(xnat_host, xnat_user, xnat_password,
command_name, wrapper_name):
    url = f"{xnat_host.rstrip('/')}/xapi/commands"
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
    if resp.status_code != 200:
        print(f"Error fetching commands: {resp.status_code}")
        sys.exit(1)
    for cmd in resp.json(): # oder ggf. resp.json()["commands"]
        # command_name prüfen
        if cmd.get("name") == command_name:
            for wrapper in cmd.get("xnat", []): # "xnat" statt "wrappers"!
                if wrapper.get("name") == wrapper_name:
                    return wrapper.get("id") # id kann auch None sein, dann ggf. name
zurückgeben
    print("No wrapper found for this command.")
    sys.exit(1)
...

```

und passe im Main an:

```

```python
try:
 wrapper_id = get_wrapper_id_by_command_name(xnat_host, xnat_user,
xnat_password, mod_data["command_name"], wrapper_name)
 ...
...

```

**\*\*Erklärung:\*\***

Du musst beim Suchen nicht alle Commands, sondern nur das richtige Command nach einem Wrapper durchsuchen. In XNAT heißt das Feld meistens ``"xnat"``, nicht ``"wrappers"``.

**\*\*2. Wrapper überschreiben/aktualisieren (fortgeschrittener, aber seltener ratsam):\*\***  
 Wrapper können über die XNAT-REST-API aktualisiert/gelöscht werden. Das ist aber fast nie sinnvoll. Meistens genügt die Wiederverwendung (siehe oben).

**\*\*3. Neuen Wrapper-Namen verwenden (z.B. mit UUID oder Zeitstempel):\*\***

Das ist aber meist nicht sinnvoll, weil es viele fast gleiche Wrapper erzeugt und unübersichtlich wird.

#### Konkrete Empfehlung

---

- Lasse deinen Code bestehen, dass er erst sucht („existiert ein Wrapper mit dem Namen...?“)
- Falls ja, verwende ihn (aktivieren, Container starten)
- Falls nein, lege ihn neu an

**\*\*Zur Info:\*\***

Willst du einen Wrapper wirklich „neu“ definieren (z.B. weil sich die JSON-Struktur geändert hat), musst du ihn manuell in XNAT löschen (über das Webinterface oder per API), weil der Name – wie die Fehlermeldung sagt – einzigartig sein muss.

**\*\*Praktischer Tipp:\*\***

Du kannst per GUI in der XNAT-Administration prüfen, welche Commands und Wrapper angelegt sind.

#### Zusammengefasst:

- **\*\*Fehlermeldung = Wrapper existiert schon\*\***
- **\*\*Lösung = Den bestehenden Wrapper verwenden, nicht überschreiben!\*\***
- **\*\*Optional = Wrapper vor Neuanlage manuell löschen, falls du ihn ersetzen willst\*\***

**\*\*Bei weiteren Fragen (z.B. wie Wrapper in XNAT gelöscht werden):\*\*** Bitte präziser nachfragen, dann kann ein Beispiel gegeben werden.

---

**\*\*Stichworte:\*\***

- XNAT Container Service
- Wrapper API
- Namenskonflikt (Uniqueness)
- Automatisierung/Fehlerbehandlung

---

Falls du Fehler-Handling oder API-Struktur weiter analysieren möchtest, helfe ich gerne gezielt weiter!

### **Bousfiha, Tanae:**

wie soll im allgemein eine wrapper heißen damit er das akzeptiert

### **HAWKI (gpt-4.1):**

**\*\*Grundregel für Wrapper-Namen in XNAT:\*\***

- **\*\*Wrapper-Namen müssen innerhalb eines Commands eindeutig sein.\*\***
- Es gilt: **\*\*Kein Command darf zwei Wrapper mit IDENTISCHEM Namen haben\*\***.
- Der Name darf keine Sonderzeichen enthalten, meist werden nur Buchstaben, Zahlen und Unterstriche verwendet.
- Empfehlenswert: Verwende Kleinbuchstaben und Unterstriche (`lower_snake_case`).

**\*\*Typisches Namensschema\*\***

Meist macht man:

...

`<command_name>_wrapper`

...

Beispiel:

- `command_name: `datenanalyser``
- `wrapper_name: `datenanalyser_wrapper``

**\*\*Für mehrere Wrapper pro Command:\*\***

Wenn du mehrere Wrapper pro Command (für unterschiedliche Kontexte) brauchst, wähle jeweils einen **\*\*eindeutigen Suffix\*\***:

- ``datenanalyser_mrSession_wrapper``
- ``datenanalyser_subject_wrapper``
- ``datenanalyser_special_wrapper``

**\*\*Automatische Generierung von Wrapper-Namen\*\***

Falls du ein Skript schreibst, das wiederholt Wrapper anlegt (z.B. in Entwicklungszyklen), dann

- nutze z.B. einen Zeitstempel oder einen zufälligen Suffix (UUID oder Hash),
- **\*\*aber:\*\*** Vermeide viele Wrapper-Leichen! Auf Produktivsystemen reicht meist ein konsistenter Name.

**\*\*Python-Beispiel für Erzeugung eines eindeutigen Wrapper-Namens:\*\***

```
```python
import datetime
def make_wrapper_name(base_command, context=None):
    name = f"{base_command.lower()}_wrapper"
    if context:
        name += f"_{context.lower()}"
    # Optional: Zeitstempel anhängen, falls nötig
    # name += "_" + datetime.datetime.now().strftime("%Y%m%d%H%M%S")
    return name
```
```

**\*\*Best Practices:\*\***

- **\*\*Für Standardfälle reicht:\*\*** `<command_name>_wrapper``
- **\*\*Für mehrere Wrapper/verschiedene Kontexte:\*\***  
`<command_name>_<was_macht_es>_wrapper``
- **\*\*Keine genauso benannten Wrapper für dasselbe Command neu anlegen:\*\*** Wenn Fehler kommt „not unique“, existiert er schon.

**\*\*Zusätzliche Hinweise:\*\***

- Achte darauf, dass der Name weder von der Länge noch von Zeichen her durch XNAT limitiert ist (meist  $\leq 64$  Zeichen, keine Leerzeichen, keine Sonderzeichen wie `\`/`` etc.).
- Die Wrapper-Namen sind nach Anlage **\*\*nicht mehr änderbar\*\*** – nur löscher/neu-anlegbar.
- Für produktive Systeme bevorzuge wenige, konsistente Wrapper-Namen.

**\*\*Fazit:\*\***

- Pro Command nur ein Wrapper mit bestimmtem Namen möglich.
- Wähle einen Namen, der zum Command passt und ihn (falls nötig) eindeutig macht.
- Versuchst du, denselben Namen erneut zu verwenden, kommt der 400-Fehler.

**\*\*Wenn du spezielle Wrapper für verschiedene Kontexte brauchst:\*\***

Füge die Kontextinformation mit in den Namen ein!

---

**\*\*TIPP:\*\***

Wenn du experimentierst, orientiere dich an den bestehenden Namen im XNAT-Webinterface unter „Administration > Plugins > Container Service > Commands/Wrapper“ (so findest du Kollisionen).

---

**\*\*Kurzum:\*\***

Die Akzeptanz (bzw. Ablehnung) in XNAT hängt allein an der Eindeutigkeit des Wrappers NAMEN je Command.

Ein guter Name ist ``<command_name>_wrapper``, ggf.

``<command_name>_<kontext>_wrapper``.

**Bousfiha, Tanae:**

auf welche frage wird wrapper gemeint im Skript?

**HAWKI (gpt-4.1):**

Im von dir geposteten Skript bezieht sich `**„Wrapper“**` auf die Kontext-Definition eines Container-Kommandos in XNAT, damit dieses z.B. auf bestimmte Datentypen angewendet werden kann („Kontextualisierung“ des Kommandos). Der Wrapper wird im Skript `**automatisch erzeugt**`, und sein Name wird aus dem Namen des XNAT-Kommandos abgeleitet.

**### Zu deiner konkreten Frage:**

`**„Auf welche Frage wird der Wrapper-Name im Skript gemeint?“**`

**#### STEP-BY-STEP Antwort:**

---

#### 1. Nutzer\*in gibt folgende Daten am Anfang ein:

- \_What is the name of the command in XNAT:\_ !' z.B. `data`
- \_What is the description of the command in XNAT:\_ ...
- \_What is the name of the Label in XNAT:\_ ...
- \_What is the description of the Label in XNAT:\_ ...

---

#### 2. **\*\*Der Wrapper-Name wird im Skript automatisch abgeleitet:\*\***

Im Abschnitt

```
```python
```

```
wrapper_name = mod_data["command_name"].replace(" ", "_").lower() + "_wrapper"
```
```

bedeutet das:

Wenn du als Command-Name `_datasciencerec_` eingibst, wird der Wrapper-Name zu

```
!` **datasciencerec_wrapper**
```

---

#### 3. In der GUI/den Fragen taucht „Wrapper“ NICHT explizit als Eingabe für dich auf!

- \_DU gibst den Wrapper-Namen NICHT manuell ein.\_
- Der Name ist **\*\*immer abgeleitet aus deiner Antwort auf „What is the name of the command in XNAT:“\*\***.

---

#### 4. **\*\*Wo wird der Name gebraucht?\*\***

- Bei Anlage des Wrappers in XNAT (im Payload, siehe JSON und die POST-Request)
- Bei Suche nach existierendem Wrapper im Skript
- Bei Aktivierung/Starten

---

### **\*\*Zusammengefasst\*\***

- Die **\*\*Wrapper-bezogene Frage\*\***/im Skript ist:



> **"What is the name of the command in XNAT:"**

Deine Antwort darauf bestimmt sowohl den Command-Namen **als auch** (per Anhang "\_wrapper") den Wrapper-Namen.

- Einen expliziten Dialog „Wie soll der Wrapper heißen?“ gibt es im Skript **nicht**.

---

**Empfehlung**

Soll der Wrapper anders heißen (etwa: mehrere Wrapper pro Command, oder andere Konventionen), musst du im Skript die Zeile

```
python
```

```
wrapper_name = mod_data["command_name"].replace(" ", "_").lower() + "_wrapper"
```

auf eine eigene Eingabe (input) oder andere Namensgebung ändern.

---

**Kurzantwort:**

Der Wrapper-Name in deinem Skript wird **automatisch** aus deiner Antwort auf

> **"What is the name of the command in XNAT:"**

generiert.

**Bousfiha, Tanae:**

when the question comes should I answer like taht science\_wrapper or only science

**HAWKI (gpt-4.1):**

**You should answer only `science`.**

**Explanation:**

- When the script asks:

...

What is the name of the command in XNAT:

...

you provide **the command name** only, e.g.:

...

science

...

The script then **automatically builds the wrapper name** as ``science_wrapper``.

- If you answer ``science_wrapper``, the resulting wrapper name will become ``science_wrapper_wrapper``, which is generally not what you want.

**Summary Table:**

| Your input                     | Wrapper name created by script         |                                   |
|--------------------------------|----------------------------------------|-----------------------------------|
| -----                          | -----                                  | -----                             |
| <code>`science`</code>         | <code>`science_wrapper`</code>         | (the correct, intended behaviour) |
| <code>`science_wrapper`</code> | <code>`science_wrapper_wrapper`</code> | (redundant; not recommended)      |
| <code>`analysis`</code>        | <code>`analysis_wrapper`</code>        | (recommended form)                |

**Recommendation:**

When prompted for the command name, **do NOT** include `"_wrapper"` in your input.

Just enter the base name of your command (e.g., ``science``).

**Bousfiha, Tanae:**

and for the label questions?

**HAWKI (gpt-4.1):**

**For the label questions, you should also provide only the descriptive label WITHOUT adding `"_wrapper"`.**

Let's clarify both label inputs in your script:

**1. "What is the name of the Label in XNAT:"**

Here, you can use a short, descriptive word or phrase that will be shown in the XNAT UI, for example:

- Science Analysis

- My Analysis
- Subject Statistics
- Data Extraction

Do **\*\*not\*\*** add "\_wrapper" or "\_label" — just use a clear, meaningful term.

**\*\*2. "What is the description of the Label in XNAT:"\*\***

Here, you provide a more detailed, human-readable description, e.g.:

- Runs data analysis for science project
- Extracts and summarizes subject data from uploaded resources

**\*\*Summary Table:\*\***

| Script question                               | Example answer               |
|-----------------------------------------------|------------------------------|
| ----- -----                                   |                              |
| What is the name of the Label in XNAT:        | Science Analysis             |
| What is the description of the Label in XNAT: | Data analysis for my project |

**\*\*Recommendation:\*\***

Just use clear words describing the process, task, or data for the label (e.g., "Science Analysis", "Prediction Results", "QC Process").

Never add "\_wrapper" or "\_label" manually; the script and XNAT will handle where those are needed.

**\*\*In summary:\*\***

When answering label questions, **\*\*do not\*\*** add technical suffixes like "\_wrapper".

Just provide clear, human-readable names and descriptions.

**Bousfiha, Tanae:**

tanae@mi0082n:~/dockerfile/xnat-osa/Ziel1\_osaDockerContainer\$ python3

Containerenabled.py

XNAT WEB URL:https://xnat-dev.gwdg.de

XNAT Username:tanae

XNAT Password:

Project ID:tanae\_osa\_Predct

Session ID:XNAT\_E00428

Path to the Python script:Bonjour.py

Docker Name base image:python:3.10

What is the name of the command in XNAT: analysis

What is the description of the command in XNAT: analyses the data

What is the name of the Label in XNAT: Data analysis for my project

What is the description of the Label in XNAT: analyses the data in xnat

Dockerfile written to ./Dockerfile

Building Docker image 'analysis:latest'...

Docker image 'analysis:latest' built successfully.

JSON file created at command.json

Uploading command to <https://xnat-dev.gwdg.de/xapi/commands>

Command created successfully.

No wrapper found for this command.

Wrapper existiert noch nicht, wird erstellt...

Wrapper-Payload: {

```
 "name": "analysis_wrapper",
 "label": "\tData analysis for my project",
 "description": "analyses the data in xnat",
 "contexts": [
```

```
 "xnat:mrSessionData"
```

```
],
```

```
 "outputs": [
```

```
 {
 "name": "result_file",
 "description": "Result",
 "required": true,
 "mount": "output_mount",
 "path": "result.csv"
 }
],
```

```
 {
```

```
 {
```

```
 "name": "session",
```

```
 "type": "Session",
```

```
 "required": true,
```

```
 "load-children": true
```

```
 }
],
```

```
 {
```

```
 {
```

```
 "name": "session",
```

```
 "type": "Session",
```

```
 "required": true,
```

```
 "load-children": true
```

```
 }
],
```

```
 {
```

```

"derived-inputs": [
 {
 "name": "csv_resource",
 "type": "Resource",
 "matcher": "@.label == 'CSV'",
 "required": true,
 "provides-files-for-command-mount": "input_mount",
 "load-children": true,
 "derived-from-wrapper-input": "session",
 "multiple": false
 },
 {
 "name": "input_file",
 "type": "File",
 "matcher": "@.name =~ /\.?(csv|tsv|txt)$/",
 "required": true,
 "load-children": true,
 "derived-from-wrapper-input": "csv_resource",
 "multiple": false
 },
 {
 "name": "input_file_name",
 "type": "string",
 "required": true,
 "load-children": true,
 "derived-from-wrapper-input": "input_file",
 "multiple": false
 }
],
"output-handlers": [
 {
 "name": "output",
 "accepts-command-output": "result_file",
 "as-a-child-of": "session",
 "type": "Resource",
 "label": "Results",
 "format": "csv"
 }
]

```

```
]
}
```

Wrapper creation failed: 400 - Invalid command:

Command "analysis" - wrapper name "analysis\_wrapper" is not unique.

Wrapper konnte nicht erstellt werden.

**HAWKI (gpt-4.1):**

### Analyse deiner Eingaben und des Problems

Du hast alles richtig ausgefüllt (ohne „\_wrapper“ und ohne technische Zusätze):

- \*\*Command Name:\*\* `/`analysis`
- \*\*Label Name:\*\* `/`Data analysis for my project`
- \*\*Description:\*\* `/`usw.

**\*\*Das Skript generiert automatisch einen Wrapper-Namen:\*\***

`analysis\_wrapper`

**\*\*Fehlermeldung XNAT:\*\***

> Wrapper creation failed: 400 - Invalid command: Command "analysis" - wrapper name "analysis\_wrapper" is not unique.

---

## Was bedeutet das konkret?

**\*\*Im System gibt es für das Command "analysis" bereits einen Wrapper mit genau diesem Namen (`analysis\_wrapper`).\*\***

Das ist ein XNAT-Prinzip:

**\*\*Ein Wrapper-Name darf je Command nur einmal vorkommen.\*\***

**\*\*Deine Eingaben sind absolut richtig und korrekt, daran liegt es nicht.\*\***

Das ist ganz normales XNAT-Verhalten!

---

## Wie kannst du das Problem lösen?

**\*\*Du hast zwei Möglichkeiten:\*\***

### ### 1. Den bestehenden Wrapper einfach weiterverwenden

Dein Skript versucht bereits, bestehenden Wrapper zu finden.

- Überprüfe (ggf. im Skript-Code): Wird die `wrapper\_id` aus dem bestehenden Command korrekt gefunden und genutzt?
- Wenn der Wrapper bereits existiert, sollte dein Skript die Wrapper-Erstellung überspringen und mit dem bestehenden Wrapper weiterarbeiten.

**\*\*Prüfe, ob die Funktion:\*\***

```
```python
```

```
get_wrapper_id_by_command_name(xnat_host, xnat_user, xnat_password,  
command_name, wrapper_name)
```

```
```
```

**\*\*richtig verwendet wird und den bereits existierenden Wrapper zurückgibt. Dann kann der nächste Schritt (Wrapper aktivieren, Container starten) ausgeführt werden.\*\***

### ### 2. Wrapper löschen und neu anlegen (nur sinnvoll, wenn du einen neuen Wrapper möchtest)

- **\*\*Manuell in XNAT-Admin-UI:\*\***

Gehe in das Command Management (/xapi/commands), finde das Command `analysis` und lösche den existierenden Wrapper `analysis\_wrapper`.

- **\*\*Oder per API:\*\***

Du kannst Wrapper über die XNAT-REST-API ebenfalls löschen (nur zu empfehlen, wenn du weißt, was du tust).

- Danach das Skript erneut ausführen.

---

### ## Was sollte man nicht tun?

**\*\*Nicht einfach den Namen künstlich ändern ("analysis2" oder ähnliche Workarounds),\*\***

außer du möchtest tatsächlich einen neuen, anders verwendbaren Wrapper erzeugen.

---

### ## Was kannst du im Skript verbessern?

Wenn du oft experimentierst, prüfe im Skript, ob statt

```
```python
sys.exit(1)
```
```

(optional) eine Alternative kommt, wie z.B.:

- "Wrapper existiert schon, wird verwendet"
- Fortsetzen mit existierendem Wrapper
- Option zum Löschen oder Überschreiben (nur nach Rückfrage)

---

## ## Zusammenfassung

- **\*\*Deine Eingaben sind korrekt.\*\***
- **\*\*Das Problem liegt daran, dass es den Wrapper-Namen schon gibt.\*\***
- **\*\*Wrapper-Namen müssen pro Command eindeutig sein.\*\***
- **\*\*Lösung 1:\*\*** Den bereits existierenden Wrapper nutzen – sollte dein Skript unterstützen!
- **\*\*Lösung 2:\*\*** Wrapper mit gleichem Namen vorher löschen, dann Skript erneut ausführen.

> **\*\*Noch Fragen?\*\***

> Ich kann gern den Skriptteil für die Wiederverwendung des Wrappers, die Fehlermeldung oder das Löschen/Neu-Anlagen gezielt erklären!

---

**\*\*Wissenschaftlicher Hintergrund:\*\***

Dieses Verhalten sichert Datenkonsistenz in Workflow-Tools wie XNAT und ist Bestandteil des Life-Cycle-Managements von Containerkommandos und Prozessen.

---

**\*\*Merke:\*\***

Die Benennung in deinem Skript ist korrekt!

Ein Wrapper kann pro Command nur *\*einmal\** existieren.

**Bousfiha, Tanae:**



```
tanae@mi0082n:~/dockerfile/xnat-osa/Ziel1_osaDockerContainer$ python3
Containerenabled.py
XNAT WEB URL:https://xnat-dev.gwdg.de
XNAT Username:tanae
XNAT Password:
Project ID:tanae_osa_Predct
Session ID:XNAT_E00428
Path to the Python script:Bonjour.py
Docker Name base image:python:3.10
What is the name of the command in XNAT: analysis
What is the description of the command in XNAT: analyses the data
What is the name of the Label in XNAT: Data analysis for my project
What is the description of the Label in XNAT: analyses the data in xnat
Dockerfile written to ./Dockerfile
Building Docker image 'analysis:latest'...
Docker image 'analysis:latest' built successfully.
JSON file created at command.json
Uploading command to https://xnat-dev.gwdg.de/xapi/commands
Command created successfully.
No wrapper found for this command.
Wrapper existiert noch nicht, wird erstellt...
Wrapper-Payload: {
 "name": "analysis_wrapper",
 "label": "\tData analysis for my project",
 "description": "analyses the data in xnat",
 "contexts": [
 "xnat:mrSessionData"
],
 "outputs": [
 {
 "name": "result_file",
 "description": "Result",
 "required": true,
 "mount": "output_mount",
 "path": "result.csv"
 }
],
 "external-inputs": [
```

```

{
 "name": "session",
 "type": "Session",
 "required": true,
 "load-children": true
}
],
"derived-inputs": [
 {
 "name": "csv_resource",
 "type": "Resource",
 "matcher": "@.label == 'CSV'",
 "required": true,
 "provides-files-for-command-mount": "input_mount",
 "load-children": true,
 "derived-from-wrapper-input": "session",
 "multiple": false
 },
 {
 "name": "input_file",
 "type": "File",
 "matcher": "@.name =~ \".*\\.(csv|tsv|txt)$\"",
 "required": true,
 "load-children": true,
 "derived-from-wrapper-input": "csv_resource",
 "multiple": false
 },
 {
 "name": "input_file_name",
 "type": "string",
 "required": true,
 "load-children": true,
 "derived-from-wrapper-input": "input_file",
 "multiple": false
 }
],
"output-handlers": [
 {

```

```

 "name": "output",
 "accepts-command-output": "result_file",
 "as-a-child-of": "session",
 "type": "Resource",
 "label": "Results",
 "format": "csv"
 }
]
}

```

Wrapper creation failed: 400 - Invalid command:

Command "analysis" - wrapper name "analysis\_wrapper" is not unique.

Wrapper konnte nicht erstellt werden.

tanae@mi0082n:~/dockerfile/xnat-osa/Ziel1\_osaDockerContainer\$ python3

Containerenabled.py

XNAT WEB URL:https://xnat-dev.gwdg.de

XNAT Username:tanae

XNAT Password:

Project ID:tanae\_osa\_Predct

Session ID:XNAT\_E00428

Path to the Python script:Bonjour.py

Docker Name base image:python:3.10

What is the name of the command in XNAT: anlissesscan

What is the description of the command in XNAT: anylisiscn

What is the name of the Label in XNAT: analisdanscan

What is the description of the Label in XNAT: dmtxos

Dockerfile written to ./Dockerfile

Building Docker image 'anlissesscan:latest'...

Docker image 'anlissesscan:latest' built successfully.

JSON file created at command.json

Uploading command to https://xnat-dev.gwdg.de/xapi/commands

Command created successfully.

No wrapper found for this command.

Wrapper existiert noch nicht, wird erstellt...

Wrapper-Payload: {

  "name": "anlissesscan\_wrapper",

  "label": "analisdanscan",

  "description": "dmtxos",

```

 "contexts": [
 "xnat:mrSessionData"
],
 "outputs": [
 {
 "name": "result_file",
 "description": "Result",
 "required": true,
 "mount": "output_mount",
 "path": "result.csv"
 }
],
 "external-inputs": [
 {
 "name": "session",
 "type": "Session",
 "required": true,
 "load-children": true
 }
],
 "derived-inputs": [
 {
 "name": "csv_resource",
 "type": "Resource",
 "matcher": "@.label == 'CSV'",
 "required": true,
 "provides-files-for-command-mount": "input_mount",
 "load-children": true,
 "derived-from-wrapper-input": "session",
 "multiple": false
 },
 {
 "name": "input_file",
 "type": "File",
 "matcher": "@.name =~ \"*.\\.(csv|tsv|txt)$\"",
 "required": true,
 "load-children": true,
 "derived-from-wrapper-input": "csv_resource",

```

```

 "multiple": false
 },
 {
 "name": "input_file_name",
 "type": "string",
 "required": true,
 "load-children": true,
 "derived-from-wrapper-input": "input_file",
 "multiple": false
 }
],
"output-handlers": [
 {
 "name": "output",
 "accepts-command-output": "result_file",
 "as-a-child-of": "session",
 "type": "Resource",
 "label": "Results",
 "format": "csv"
 }
]
}

```

Wrapper creation failed: 400 - Invalid command:

Command "anlisesscan" - wrapper name  
 "anlisesscan\_wrapper" is not unique.

Wrapper konnte nicht erstellt werden.

## **HAWKI (gpt-4.1):**

Du bekommst wiederholt die Meldung:

```
> **Wrapper creation failed: 400 - Invalid command: Command "<name>" - wrapper
name "<name>_wrapper" is not unique.**
```

### Was bedeutet das?

- Für das jeweilige Command (egal wie es heißt: `analysis` oder `anlisesscan`) existiert in deinem XNAT-System bereits **ein** Wrapper mit exakt dem gleichen Namen (`analysis_wrapper` bzw. `anlisesscan_wrapper`).

- XNAT erlaubt `_pro Command_` nur **einzigartige Wrappernamen**.

---

**## Warum wiederholt das Skript immer dasselbe Verhalten?**

**\*\*Der eigentliche Fehler\*\*** ist, dass dein Skript einen neuen Wrapper anlegen möchte und davon ausgeht, dass noch keiner existiert.

Die Logik, um zu prüfen, ob bereits ein Wrapper existiert, funktioniert offenbar nicht korrekt!

Er findet bestehende Wrapper nicht zuverlässig – darum der erneute POST-Versuch (und XNAT verweigert das zu Recht).

---

**## Wie kannst du das Problem **sauber lösen?****

**### 1. **Wiederverwendung statt Neuanlage** von Wrappern**

**\*\*Empfehlung:\*\***

Nutze den bestehenden Wrapper, falls er bereits vorhanden ist. Dafür muss dein Skript korrekt prüfen, ob schon ein Wrapper da ist **und dann mit ihm weiterarbeiten**.

**### 2. **Entfernen** von Wrappern/(nur wenn notwendig):**

- **\*\*Manuell:\*\***

Gehe ins XNAT-Webinterface! Admin-Oberfläche! Contain  
Commands! [dein Command]! Wrappers! Löschen.

- **\*\*Per REST-API:\*\***

Mach nur Sinn, wenn du das System automatisiert wartest.

**### 3. **Korrektur deiner Wrapper-Prüffunktion****

**\*\*Im Skript ist diese Funktion entscheidend:\*\***

````python`

`def get_wrapper_id_by_command_name(xnat_host, xnat_user, xnat_password,`

```

command_name, wrapper_name):
    url = f"{xnat_host.rstrip('/')}/xapi/commands"
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
    if resp.status_code != 200:
        print(f"Error fetching commands: {resp.status_code}")
        sys.exit(1)
    for cmd in resp.json():
        if cmd.get("name") == command_name:
            for wrapper in cmd.get("xnat", []):
                if wrapper.get("name") == wrapper_name:
                    return wrapper.get("id") # oder ggf. nur wrapper_name!
    print("No wrapper found for this command.")
    sys.exit(1)
...

```

****Stelle sicher****, dass in deinem Skript DAUERHAFT nach Wrappern gesucht wird,
****bevor**** einer neu angelegt wird!
 Im Fehlerfall kannst du z.B. nicht erneut anlegen, sondern gleich mit der ID weiterarbeiten.

4. ****Debug-Tipp:****

Lass dir beim Start deines Skripts alle für dieses Command existierenden Wrapper im Terminal anzeigen, Z.B.:

```

```python
print("Bereits existierende Wrapper für dieses Command:")
for wrapper in cmd.get("xnat", []):
 print("-", wrapper.get("name"))
...

```

So siehst du, ob im System schon einer ist.

---

#### ## **\*\*Zusammenfassung:\*\***

- Der Fehler liegt nicht an deinen Eingaben, sondern an der Skriptlogik (Suche vs. Neuanlage).
- Lege nicht erneut einen Wrapper mit gleichem Namen an.

- Prüfe und verwende vorhandene Wrapper.
- Wenn wirklich ein neuer Wrapper gebraucht wird: Lösche zuvor den mit gleichem Namen im XNAT-System.
- Alternativ könntest du im Skript eine Option einbauen, einen neuen einzigartigen Namen zu generieren (aber das ist meist nicht sinnvoll).

---

### **\*\*Optional: Skriptausschnitt für bessere Fehlertoleranz\*\***

Ersetze vor dem Anlegen eines Wrappers:

```
```python
```

```
try:
```

```
    wrapper_id = get_wrapper_id_by_command_name(xnat_host, xnat_user,
xnat_password, mod_data["command_name"], wrapper_name)
```

```
    print(f"Wrapper existiert bereits: {wrapper_id}")
```

```
except SystemExit:
```

```
    print("Wrapper existiert noch nicht, wird erstellt...")
```

```
    ...
```
```

**\*\*– damit brichst du nicht ab, sondern nutzt den vorhandenen weiter.\*\***

---

### **\*\*Wissenschaftlicher Bezug:\*\***

Dieses Verhalten entspricht bewährten Prinzipien aus Software Lifecycle und Workflow-Management:

Jede Entity sollte einmalig identifizierbar sein, Redundanzen und Namenskollisionen führen sonst zu Inkonsistenzen im System.

---

**\*\*Wenn du Hilfe bei einer robusteren Wrapper-Prüfung oder bei der Analyse deiner Umgebung brauchst, kann ich dich gerne durch den Prozess lotsen – frage gern nach!\*\***

**Bousfiha, Tanae:**

can i run the container directly



container-rest-api  
Container Rest Api

GET  
/container/name/{name}  
Get Containers by database name

GET  
/containers  
Get all Containers

POST  
/containers  
Get paginated containers per request

GET  
/containers/version  
Check XNAT Version compatibility.

GET  
/containers/{containerId}/logSince/{file}  
Get Container logs

GET  
/containers/{containerId}/logs  
Get Container logs

GET  
/containers/{containerId}/logs/{file}  
Get Container logs

GET

/containers/{id}

Get Containers by database ID

DELETE

/containers/{id}

Get Container by container server ID

POST

/containers/{id}/finalize

Finalize Container

POST

/containers/{id}/kill

Kill Container

GET

/projects/{project}/containers

Get all Containers by project

GET

/projects/{project}/containers/name/{name}

Get Containers by name

POST

/projects/{project}/containers/{id}/kill

Kill Container

with those or i need something more?

**HAWKI** (gpt-4.1):

**\*\*Short answer:\*\***

No, you cannot directly launch (start) a *\*new\** container with these REST endpoints alone.

You need a "run" or "launch" endpoint which is commonly called  
`/container-launcher/launch` in XNAT's Container Service API.

---

### ### Explanation

The endpoints you listed allow you to:

- **\*\*GET\*\*** information about existing containers (by ID, name, logs, etc)
- **\*\*POST/DELETE\*\*** to kill, finalize, or query metadata of existing containers

But:

- There is **\*\*no endpoint in your list to CREATE (launch) a new container execution\*\*** for a command+session/inputs.
- Standard XNAT Container Service uses the endpoint:

...

POST /xapi/container-launcher/launch

...

or in older setups sometimes

...

POST /xapi/container/launch

...

with a payload (JSON) that defines:

- the Command or Wrapper to run
- the Inputs (e.g. session, files, etc)
- optional configuration

---

### ### Typical Launch Example

```
```http
POST https://xnat.yourserver.de/xapi/container-launcher/launch
Content-Type: application/json

{
  "command": "abcd12345",    // (or use "wrapper" key)
  "wrapper": "abcd12345",
  "inputs": {
    "session": "XNAT_E00428"
    // ...other input specifiers as needed
  }
}
```
```

This actually **RUNS** a container job (analysis, preprocessing, etc.)  
All other endpoints you listed only deal with containers that are already running or finished.

---

### ### Conclusion

- To launch/run a NEW container: **You need `/xapi/container-launcher/launch`**
- The API you listed is mainly for `_monitoring_` or `_managing_` already started containers.

---

### ### References

- [XNAT Container Service API docs (Container Launcher)](<https://wiki.xnat.org/container-service/container-service-api>)
- [XNAT Container REST API on SwaggerHub](<https://app.swaggerhub.com/apis/gbookman/xnat-container/>)

---

## **\*\*Summary Table:\*\***

| Endpoint                                          | Purpose                              | Can it start a new job? |
|---------------------------------------------------|--------------------------------------|-------------------------|
| ----- ----- -----                                 |                                      |                         |
| `/container/name/{name}`                          | Get info on existing container       | No                      |
|                                                   |                                      |                         |
| `/containers`                                     | List containers                      | No                      |
|                                                   |                                      |                         |
| `/containers/{containerId}/logs`                  | Get logs of an existing container    | No                      |
|                                                   |                                      |                         |
| `/containers/{id}`                                | Get container by ID                  | No                      |
|                                                   |                                      |                         |
| `/containers/{id}/kill`                           | Kill (stop) a running container      | No                      |
|                                                   |                                      |                         |
| `/xapi/container-launcher/launch`<br>(Required)** | <b>**Launch new container task**</b> | <b>**Yes</b>            |

---

**\*\*You need one of the `/launch` endpoints to actually start a new process.\*\***

The other endpoints are not sufficient for launching a container analysis in XNAT.

## **Bousfiha, Tanae:**

In my script I successfully upload a command in JSON form. How can I run the container after that?

## **HAWKI (gpt-4.1):**

Great question!

**\*\*After successfully uploading a command in JSON to XNAT, starting (running) the container from your script involves a call to the XNAT Container Service REST API endpoint `/xapi/container-launcher/launch`.**

Here's how you do it, step-by-step, with code example and explanation:

---

**## 1. \*\*Make sure you have:\*\***

- The command and wrapper already registered in XNAT

- The wrapper enabled for your project
- The session ID and any other required inputs

---

## ## 2. **\*\*Payload for launching a container\*\***

Typically, you send a POST to `/xapi/container-launcher/launch` with a JSON like:

```
```json
{
  "command": "<command_id>", // or
  "wrapper": "<wrapper_id>",
  "inputs": {
    "session": "<session_id>"
    // (Add other required input keys here)
  }
}
```
```

- Usually, you set `"wrapper": "<wrapper\_id>"` to launch a wrapper.
- Inputs must match the wrapper/command interface (session, resources, file names, etc.).

---

## ## 3. **\*\*Python example using `requests`\*\***

Suppose you have the variables from your script:

- `xnat\_host`
- `xnat\_user`, `xnat\_password`
- `wrapper\_id` (or `command\_id`)
- `session\_id`

You would use:

```
```python
import requests
import json
```

```

# Your XNAT server and credentials
xnat_host = "https://xnat-dev.gwdg.de"
xnat_user = "YOURUSER"
xnat_password = "YOURPASS"

# IDs obtained earlier
wrapper_id = "abcd12345" # Replace with actual
session_id = "XNAT_E00428"

# Compose API endpoint
launch_url = f"{xnat_host.rstrip('/')}/xapi/container-launcher/launch"

# Create payload
payload = {
    "wrapper": wrapper_id,
    "inputs": {
        "session": session_id
        # add more inputs if needed by your wrapper
    }
}

# Make the POST request
r = requests.post(
    launch_url,
    auth=(xnat_user, xnat_password),
    headers={"Content-Type": "application/json"},
    json=payload,
    verify=False # if you have self-signed cert
)

print(r.status_code, r.text)
if r.status_code in (200, 201):
    print("Container launched!")
else:
    print("Failed to launch:", r.text)
...

*(You can adapt this with variables from your script.)*

```

4. ****Integrate into your script (your style):****

In your script, you already define:

```
```python
def run_container_in_xnat(xnat_host, wrapper_id, session_id, xnat_user,
xnat_password):
 url = f"{xnat_host}/xapi/container-launcher/launch"
 headers = {"Content-Type": "application/json"}

 payload = {
 "command": wrapper_id, # or "wrapper"—needs to match API!
 "wrapper": wrapper_id,
 "inputs": {
 "session": session_id
 }
 }
 response = requests.post(url, auth=(xnat_user, xnat_password), headers=headers,
json=payload, verify=False)
 if response.status_code in (200, 201):
 print(f"Container launched successfully.")
 else:
 print(f"Failed to launch container: {response.status_code} - {response.text}")
```
```

****Make sure you pass the real wrapper_id!****

And that your payload matches your wrapper's (or command's) input requirements.

5. ****What if I want to launch a command (not a wrapper)?****

Most cases in XNAT use the wrapper for context (project, datatype etc.).

But for commands without a wrapper, you'd use `"command": <command_id>` and specify all required inputs.

****Conclusion/Summary:****

- You must POST to `/xapi/container-launcher/launch` with the wrapper (or command) ID and all required inputs.
- This can be integrated as a function in your script (see above).
- Always check response and status code.

****If you want, I can help you insert this into your exact script by line number—just let me know where you want to run the container from!****

****Academic Note:****

This process ensures reproducibility and provenance in scientific workflows, a key aspect in data-driven research platforms like XNAT.

****Further References****

- [XNAT Container Service Docs](<https://wiki.xnat.org/container-service/container-service-api>)
- [XNAT REST API Examples](<https://wiki.xnat.org/>)

****Let me know if you need help debugging your wrapper input/output requirements, or if your container fails to launch!****

Bousfiha, Tanae:

```
#-----Bibliotheken-----  
import datetime  
import json  
import requests # https://wiki.xnat.org/container-service/container-service-api  
import os  
import subprocess # https://www.datacamp.com/tutorial/python-subprocess
```

```
import getpass
import sys
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
```

```
#-----erstmal Skript from the users
Fragen-----
```

```
def check_user_skript(skript_path):
    if not os.path.isfile(skript_path):
        print(f"Datei nicht gefunden: {skript_path}")
        return False
    if not skript_path.endswith(".py"):
        print("Datei nicht korrekt, bitte eine .py Datei angeben.")
        return False
    return True
```

```
#-----dockerfile
ausfüllen-----
```

```
def write_dockerfile(docker_dir, script_filename, docker_base_image):
    dockerfile_content = f"&quot;&quot;FROM {docker_base_image}
```

```
WORKDIR /app
```

```
COPY {script_filename} /app/{script_filename}
```

```
ENTRYPOINT ["python3", "/app/{script_filename}"]
```

```
CMD ["example.csv", "/app/output"]
```

```
&&&
```

```
    os.makedirs(docker_dir, exist_ok=True)
    dockerfile_path = os.path.join(docker_dir, "Dockerfile")
    with open(dockerfile_path, "w") as f:
        f.write(dockerfile_content)
    print(f"Dockerfile written to {dockerfile_path}")
    return dockerfile_path
```

```
#-----Image bauen-----
def build_docker_image(dockerfile_path, docker_image_name):
    build_command = [
        "docker", "build", "-f", dockerfile_path,
        "-t", docker_image_name, "."
    ]
    print(f"Building Docker image #{docker_image_name}...")
    result = subprocess.run(build_command, capture_output=True, text=True)
    if result.returncode == 0:
        print(f"Docker image #{docker_image_name} built successfully.")
    else:
        print(f"Failed to build Docker image: {result.stderr}")
        exit(1)
```

```
#-----User-Input-----
def get_input(prompt):
    while True:
        value = input(prompt)
        if value.strip():
            return value
        else:
            print("Cannot be empty.")
```

```
def modification():
    data = {}
    data["command_name"] = get_input("What is the name of the command in XNAT: ")
    data["command_description"] = get_input("What is the description of the command in XNAT: ")
    data["label_name"] = get_input("What is the name of the Label in XNAT: ")
    data["label_description"] = get_input("What is the description of the Label in XNAT: ")
    return data
```

```
#-----json File erstellen-----
def create_json_file(docker_image, script_filename, mod_data):
```

```

wrapper_name = mod_data[&quot;command_name&quot;].replace(&quot;; &quot;,
&quot;;_&quot;).lower() + &quot;;_wrapper&quot;;
json_file = {
    &quot;name&quot;:: mod_data[&quot;command_name&quot;],
    &quot;description&quot;:: mod_data[&quot;command_description&quot;],
    &quot;version&quot;:: &quot;1.0&quot;,
    &quot;image&quot;:: docker_image,
    &quot;type&quot;:: &quot;docker&quot;,
    &quot;command-line&quot;:: f&quot;python /app/{script_filename}
/app/input/#INPUT_FILE# /app/output&quot;;,
    &quot;mounts&quot;:: [
        {&quot;name&quot;:: &quot;output_mount&quot;, &quot;writable&quot;:: True,
&quot;path&quot;:: &quot;/app/output&quot;},
        {&quot;name&quot;:: &quot;input_mount&quot;, &quot;writable&quot;:: False,
&quot;path&quot;:: &quot;/app/input&quot;}
    ],
    &quot;inputs&quot;:: [
        {
            &quot;name&quot;:: &quot;INPUT_FILE&quot;,
            &quot;description&quot;:: mod_data[&quot;command_description&quot;],
            &quot;type&quot;:: &quot;string&quot;,
            &quot;required&quot;:: True
        }
    ],
    &quot;outputs&quot;:: [
        {
            &quot;name&quot;:: &quot;result_file&quot;,
            &quot;description&quot;:: &quot;Result&quot;,
            &quot;required&quot;:: True,
            &quot;mount&quot;:: &quot;output_mount&quot;,
            &quot;path&quot;:: &quot;result.csv&quot;
        }
    ],
    &quot;xnat&quot;:: [
        {
            &quot;name&quot;:: wrapper_name,
            &quot;label&quot;:: mod_data[&quot;label_name&quot;],
            &quot;description&quot;:: mod_data[&quot;label_description&quot;],

```

```

    &quot;contexts&quot;:: [&quot;xnat:mrSessionData&quot;],
    &quot;external-inputs&quot;:: [
        {&quot;name&quot;:: &quot;session&quot;, &quot;type&quot;::
&quot;Session&quot;, &quot;required&quot;:: True, &quot;load-children&quot;:: True}
    ],
    &quot;derived-inputs&quot;:: [
        {
            &quot;name&quot;:: &quot;csv_resource&quot;,
            &quot;type&quot;:: &quot;Resource&quot;,
            &quot;matcher&quot;:: &quot;@.label == &#039;CSV&#039;&quot;,
            &quot;required&quot;:: True,
            &quot;provides-files-for-command-mount&quot;::
&quot;input_mount&quot;,
            &quot;load-children&quot;:: True,
            &quot;derived-from-wrapper-input&quot;:: &quot;session&quot;
        },
        {
            &quot;name&quot;:: &quot;input_file&quot;,
            &quot;type&quot;:: &quot;File&quot;,
            &quot;matcher&quot;:: &quot;@.name =~
&quot;.*(csv|tsv|txt)$&quot;,
            &quot;required&quot;:: True,
            &quot;load-children&quot;:: True,
            &quot;derived-from-wrapper-input&quot;:: &quot;csv_resource&quot;
        },
        {
            &quot;name&quot;:: &quot;input_file_name&quot;,
            &quot;type&quot;:: &quot;string&quot;,
            &quot;derived-from-wrapper-input&quot;:: &quot;input_file&quot;
        }
    ],
    &quot;output-handlers&quot;:: [
        {
            &quot;name&quot;:: &quot;output&quot;,
            &quot;accepts-command-output&quot;:: &quot;result_file&quot;,
            &quot;as-a-child-of&quot;:: &quot;session&quot;,
            &quot;type&quot;:: &quot;Resource&quot;,
            &quot;label&quot;:: &quot;Results&quot;,

```

```

        &quot;format&quot;;: &quot;csv&quot;;
    }
]
}
]
}
#-----
with open(&quot;command.json&quot;, &quot;w&quot;) as json_out:
    json.dump(json_file, json_out, indent=4)
    print(f&quot;JSON file created at command.json&quot;)
return &quot;command.json&quot;

#-----Command zu XNAT senden-----
def send_json_to_xnat(json_file_path, xnat_url, xnat_user, xnat_password):
    url = f&quot;{xnat_url}/xapi/commands&quot;
    print(f&quot;Uploading command to {url}&quot;)
    with open(json_file_path, &quot;r&quot;) as f:
        response = requests.post(url, auth=(xnat_user, xnat_password),
json=json.load(f))
    if response.status_code == 200:
        print(&quot;Command uploaded successfully.&quot;)
    elif response.status_code == 201:
        print(&quot;Command created successfully.&quot;)
    elif response.status_code == 409:
        print(&quot;Command already exists.&quot;)
    else:
        print(f&quot;Failed to upload command: {response.status_code} -
{response.text}&quot;)

#-----Command-Liste-----
def get_command_id_by_name(xnat_host, xnat_user, xnat_password,
command_name):
    url = f&quot;{xnat_host.rstrip(&quot;/&quot;)}xapi/commands&quot;
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
    if resp.status_code != 200:
        print(f&quot;Error fetching commands: {resp.status_code}&quot;)
        sys.exit(1)
    data = resp.json()

```

```
if isinstance(data, dict) and 'commands' in data:
```

```
    command_list = data['commands']
```

```
else:
```

```
    command_list = data
```

```
for command in command_list:
```

```
    if command.get('name') == command_name:
```

```
        return command['id']
```

```
print('Command not found.')
```

```
sys.exit(1)
```

```
#-----Wrapper auslesen/erstellen-----
```

```
def get_command_io(xnat_host, xnat_user, xnat_password, command_id):
```

```
    url =
```

```
f'{xnat_host.rstrip('/')}/xapi/commands/{command_id}'
```

```
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
```

```
    if resp.status_code != 200:
```

```
        print(f'Fehler beim Abrufen des Commands: {resp.status_code}')
```

```
        sys.exit(1)
```

```
    cmd = resp.json()
```

```
    outputs = cmd.get('outputs', [])
```

```
    external_inputs = []
```

```
    derived_inputs = []
```

```
    for wrapper in cmd.get('xnat', []):
```

```
        external_inputs = wrapper.get('external-inputs', [])
```

```
        derived_inputs = wrapper.get('derived-inputs', [])
```

```
        break # Nur den ersten Wrapper verwenden
```

```
    return outputs, external_inputs, derived_inputs
```

```
def get_wrapper_id_by_command_name(xnat_host, xnat_user, xnat_password,  
wrapper_name):
```

```
    url = f'{xnat_host.rstrip('/')}/xapi/commands'
```

```
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
```

```
    if resp.status_code != 200:
```

```
        print(f'Error fetching commands: {resp.status_code}')
```

```
        sys.exit(1)
```

```
    for command in resp.json():
```

```
        for wrapper in command.get('wrappers', []):
```

```
            if wrapper.get('name') == wrapper_name:
```

```

        return wrapper[&quot;id&quot;]
    print(&quot;No wrapper found for this command.&quot;)
    sys.exit(1)

```

```

def create_wrapper(xnat_host, command_id, wrapper_name, label_name,
description, xnat_user, xnat_password, outputs, external_inputs, derived_inputs):
    url = f&quot;{xnat_host.rstrip(&#039;/&#039;)}&#039;/xapi/commands/{command_id}/wrap
pers&quot;
    output_handlers = [{
        &quot;name&quot;: &quot;output&quot;,
        &quot;accepts-command-output&quot;: &quot;result_file&quot;,
        &quot;as-a-child-of&quot;: &quot;session&quot;,
        &quot;type&quot;: &quot;Resource&quot;,
        &quot;label&quot;: &quot;Results&quot;,
        &quot;format&quot;: &quot;csv&quot;
    }]
    wrapper = {
        &quot;name&quot;: wrapper_name,
        &quot;label&quot;: label_name,
        &quot;description&quot;: description,
        &quot;contexts&quot;: [&quot;xnat:mrSessionData&quot;],
        &quot;outputs&quot;: outputs,
        &quot;external-inputs&quot;: external_inputs,
        &quot;derived-inputs&quot;: derived_inputs,
        &quot;output-handlers&quot;: output_handlers
    }
    print(&quot;Wrapper-Payload:&quot;, json.dumps(wrapper, indent=2))
    resp = requests.post(url, auth=(xnat_user, xnat_password),
        headers={&quot;Content-Type&quot;: &quot;application/json&quot;},
        json=wrapper, verify=False)
    # Korrekte Behandlung von Status-Code:
    if resp.status_code == 201:
        wrapper_id = resp.text.strip()
        print(f&quot;Wrapper created successfully. ID: {wrapper_id}&quot;)
        return wrapper_id
    elif resp.status_code == 200:
        print(&quot;Wrapper created successfully (200).&quot;)
        # extrahiere ggf. return-id

```



```

elif resp.status_code == 409:
    print("&quot;Wrapper already exists.&quot;")
    return None
else:
    print(f"&quot;Wrapper creation failed: {resp.status_code} - {resp.text}&quot;")
    return None

```

#-----Wrapper Aktivierung-----

```

def enable_wrapper(xnat_host, project_id, command_name, wrapper_id, xnat_user,
xnat_password):
    enable_url = f"&quot;{xnat_host.rstrip('&#039;')}/xapi/projects/{project_id}/co
mmands/{command_name}/wrappers/{wrapper_id}/enable&quot;";
    print(f"&quot;Enable-URL: {enable_url}&quot;") # Debug!
    resp = requests.post(enable_url, auth=(xnat_user, xnat_password), verify=False)
    if resp.status_code == 200:
        print("&quot;Wrapper successfully enabled.&quot;")
    else:
        print(f"&quot;Failed to enable wrapper: {resp.status_code} - {resp.text}&quot;")

```

#-----Container in XNAT starten-----

```

def run_container_in_xnat(xnat_host, wrapper_id, session_id, xnat_user,
xnat_password):
    url = f"&quot;{xnat_host}/xapi/container-launcher/launch&quot;";
    headers = {"&quot;Content-Type&quot;": "&quot;application/json&quot;"}

    payload = {
        "&quot;command&quot;": wrapper_id,
        "&quot;wrapper&quot;": wrapper_id,
        "&quot;inputs&quot;": {
            "&quot;session&quot;": session_id
        }
    }
    print("&quot;Launching container with payload:&quot;," json.dumps(payload,
indent=2)) # Debug!
    response = requests.post(url, auth=(xnat_user, xnat_password), headers=headers,
json=payload, verify=False)

```

```

if response.status_code in [200, 201]:
    print(f"Container launched successfully.")
else:
    print(f"Failed to launch container: {response.status_code} -
{response.text}")
```

#-----Main Teil-----

```

def main():
    xnat_host = get_input("XNAT WEB URL:")
    xnat_user = get_input("XNAT Username:")
    xnat_password = getpass.getpass("XNAT Password: ")
    project_id = get_input("Project ID:")
    session_id = get_input("Session ID:")
    script_path = get_input("Path to the Python script:")
    docker_base_image = get_input("Docker Name base image:")

    if not check_user_skript(script_path):
        return

    mod_data = modification()

    # Stabiler Wrapper-Name (ohne Zeitstempel)
    wrapper_name = mod_data["command_name"].replace(" ",
"_").lower() + "_wrapper"

    # Dockerfile erstellen und Image bauen
    dockerfile_path = write_dockerfile(".", os.path.basename(script_path),
docker_base_image)
    docker_image_name =
f"{mod_data[&#039;command_name&#039;].lower().replace(&#039; &#039;,
&#039;_&#039;)}:latest"
    build_docker_image(dockerfile_path, docker_image_name)

    # JSON erstellen und an XNAT senden
    json_file_path = create_json_file(docker_image_name,
os.path.basename(script_path), mod_data)
```

```

send_json_to_xnat(json_file_path, xnat_host, xnat_user, xnat_password)

# Wrapper anlegen oder holen
try:
    wrapper_id = get_wrapper_id_by_command_name(xnat_host, xnat_user,
xnat_password, wrapper_name)
    print(f"Wrapper existiert bereits: {wrapper_id}")
except SystemExit:
    print("Wrapper existiert noch nicht, wird erstellt...")
    command_id = get_command_id_by_name(xnat_host, xnat_user,
xnat_password, mod_data["command_name"])
    outputs, external_inputs, derived_inputs = get_command_io(xnat_host,
xnat_user, xnat_password, command_id)
    wrapper_id = create_wrapper(
        xnat_host, command_id, wrapper_name,
        mod_data["label_name"],
mod_data["label_description"],
        xnat_user, xnat_password,
        outputs, external_inputs, derived_inputs
    )
    if not wrapper_id:
        print("Wrapper konnte nicht erstellt werden.")
        return

# Wrapper aktivieren
enable_wrapper(xnat_host, project_id, mod_data["command_name"],
wrapper_id, xnat_user, xnat_password)

# Container starten
run_container_in_xnat(xnat_host, wrapper_id, session_id, xnat_user,
xnat_password)

if __name__ == "__main__":
    main()

```

here it stopped after crating th ewrapper and did not luanch the container

HAWKI (gpt-4.1):

Sehr gut beobachtet, dass dein Skript nach "Wrapper konnte nicht erstellt werden." abbricht und den Container NICHT startet. Das liegt am Flow in diesen Zeilen im ``main()`:`

```
```python
except SystemExit:
 print("Wrapper existiert noch nicht, wird erstellt...")
 ...
 wrapper_id = create_wrapper(...)
 if not wrapper_id:
 print("Wrapper konnte nicht erstellt werden.")
 return

Wrapper aktivieren
enable_wrapper(...)

Container starten
run_container_in_xnat(...)
```
```

****Das Problem:****

Wenn die Wrapper-Erstellung fehlschlägt (z.B. weil ein Wrapper schon existiert oder ein anderer Fehler auftritt), gibt deine ``create_wrapper``-Funktion ``None`` zurück und es kommt zu einem ``return``! Das Skript stoppt HIER und errechnet die nächsten Zeilen für das Aktivieren und Starten des Containers.

Lösungsempfehlung

1. Besseres Fehler-Handling und Wrapper-Check

Du solltest unterscheiden, ob der Wrapper nur schon existiert (dann weiter verwenden!), oder ob ein echter Fehler vorliegt (z.B. Netzwerkfehler, fehlende Berechtigung).

2. Fallback-Logik bei Fehlern

Das Skript soll IMMER versuchen, mit ****dem existierenden Wrapper**** fortzufahren,

bevor es abbricht.

Korrigierte Logik — Beispiel

```
```python
```

```
try:
```

```
 wrapper_id = get_wrapper_id_by_command_name(xnat_host, xnat_user,
xnat_password, wrapper_name)
```

```
 print(f"Wrapper existiert bereits: {wrapper_id}")
```

```
except SystemExit:
```

```
 print("Wrapper existiert noch nicht, wird erstellt...")
```

```
 command_id = get_command_id_by_name(xnat_host, xnat_user, xnat_password,
mod_data["command_name"])
```

```
 outputs, external_inputs, derived_inputs = get_command_io(xnat_host, xnat_user,
xnat_password, command_id)
```

```
 wrapper_id = create_wrapper(
 xnat_host, command_id, wrapper_name,
 mod_data["label_name"], mod_data["label_description"],
 xnat_user, xnat_password,
 outputs, external_inputs, derived_inputs
)
```

```
 if not wrapper_id:
```

```
 # Neu: Vielleicht existiert der Wrapper schon, dann suche ihn erneut!
```

```
 print("Wrapper konnte nicht erstellt werden, versuche existierenden Wrapper zu
verwenden ...")
```

```
 try:
```

```
 wrapper_id = get_wrapper_id_by_command_name(xnat_host, xnat_user,
xnat_password, wrapper_name)
```

```
 print(f"Existierenden Wrapper gefunden: {wrapper_id}")
```

```
 except SystemExit:
```

```
 print("Kein Wrapper gefunden oder schwerwiegender Fehler – Abbruch.")
```

```
 return
```

```
Wrapper aktivieren
```

```
enable_wrapper(xnat_host, project_id, mod_data["command_name"], wrapper_id,
xnat_user, xnat_password)
```

# Container starten

```
run_container_in_xnat(xnat_host, wrapper_id, session_id, xnat_user, xnat_password)
...
```

**\*\*Was ändert sich?\*\***

- Falls die Wrapper-Erstellung fehlschlägt, versucht das Skript, nochmal einen existierenden Wrapper zu verwenden.
- Nur wenn auch *\*das\** fehlschlägt, bricht das Skript ab.

---

### **\*\*Zusammenfassung\*\***

- Das Skript soll immer versuchen, mit *\*einem existierenden\** Wrapper fortzufahren, wenn ``create_wrapper`` keinen neuen erzeugen kann.
- So kannst du auch bei Namenskollisionen oder bei weiteren Läufen den Container immer starten.

---

### **\*\*Kürzere Fassung:\*\***

- > **\*\*Das Skript bricht ab, weil bei Fehler kein Wrapper-Objekt gefunden wird.\*\***
- > **\*\*Passe an, dass bei Wrapper-Problemen immer nochmal gezielt nach einem existierenden Wrapper gesucht wird und dann Container-Start weitergeführt wird.\*\***

---

#### Wissenschaftlicher Kontext

Robustes Fehlerhandling ist entscheidend für reproduzierbare und automatisierte Workflows – dafür sollten immer alle existierenden Artefakte sicher wiederverwendet werden, sofern nicht ein kritischer Fehler vorliegt.

---

**\*\*Wenn du möchtest, kann ich dir den vollständigen korrigierten Main-Block posten – sag gern Bescheid!\*\***

**Bousfiha, Tanae:**

ja bitte

**HAWKI (gpt-4.1):**

Sehr gerne! Hier ist die **\*\*robuste, fehlertolerante Version\*\*** deines Main-Teils – so wird immer versucht, mit einem existierenden Wrapper den Prozess weiterzuführen, auch wenn das Anlegen nicht klappt:

```
```python
def main():
    xnat_host = get_input("XNAT WEB URL:")
    xnat_user = get_input("XNAT Username:")
    xnat_password = getpass.getpass("XNAT Password: ")
    project_id = get_input("Project ID:")
    session_id = get_input("Session ID:")
    script_path = get_input("Path to the Python script:")
    docker_base_image = get_input("Docker Name base image:")

    if not check_user_skript(script_path):
        return

    mod_data = modification()
    wrapper_name = mod_data["command_name"].replace(" ", "_").lower() +
    "_wrapper"

    # Dockerfile erstellen und Image bauen
    dockerfile_path = write_dockerfile(".", os.path.basename(script_path),
    docker_base_image)
    docker_image_name = f"{mod_data['command_name'].lower().replace(' ',
    '_')}:latest"
    build_docker_image(dockerfile_path, docker_image_name)

    # JSON erstellen und an XNAT senden
    json_file_path = create_json_file(docker_image_name,
    os.path.basename(script_path), mod_data)
    send_json_to_xnat(json_file_path, xnat_host, xnat_user, xnat_password)
```
```

```

--- Wrapper anlegen oder holen (fehlerrobust!) ---
wrapper_id = None
try:
 wrapper_id = get_wrapper_id_by_command_name(xnat_host, xnat_user,
xnat_password, wrapper_name)
 print(f"Wrapper existiert bereits: {wrapper_id}")
except SystemExit:
 print("Wrapper existiert noch nicht, wird erstellt...")

 command_id = get_command_id_by_name(xnat_host, xnat_user,
xnat_password, mod_data["command_name"])
 outputs, external_inputs, derived_inputs = get_command_io(xnat_host,
xnat_user, xnat_password, command_id)
 wrapper_id = create_wrapper(
 xnat_host, command_id, wrapper_name,
 mod_data["label_name"], mod_data["label_description"],
 xnat_user, xnat_password,
 outputs, external_inputs, derived_inputs
)

if not wrapper_id:
 # Prüfe nochmal, ob Wrapper inzwischen existiert!
 print("Wrapper konnte nicht erstellt werden, versuche existierenden Wrapper
zu verwenden ...")
 try:
 wrapper_id = get_wrapper_id_by_command_name(xnat_host, xnat_user,
xnat_password, wrapper_name)
 print(f"Existierenden Wrapper gefunden: {wrapper_id}")
 except SystemExit:
 print("Kein Wrapper gefunden oder schwerwiegender Fehler – Abbruch.")
 return

--- Wrapper aktivieren ---
enable_wrapper(xnat_host, project_id, mod_data["command_name"], wrapper_id,
xnat_user, xnat_password)

--- Container starten ---

```



```
run_container_in_xnat(xnat_host, wrapper_id, session_id, xnat_user,
xnat_password)
```

```
if __name__ == "__main__":
 main()
...
```

**\*\*Hintergrund:\*\***

- Der Code versucht IMMER zum Schluss, eine gültige `wrapper\_id` zu bekommen.
- Auch wenn das Erstellen fehlschlägt (weil der Wrapper schon existiert), wird nochmal versucht, die vorhandene `wrapper\_id` zu holen.
- Nur wenn dabei ebenfalls ein Fehler auftritt (z.B. Netzwerk-Ausfall oder Command/Wrapper existiert WIRKLICH nicht), wird abgebrochen.
- So kannst du immer weiterarbeiten, solange technisch ein Wrapper im System vorhanden ist.

**\*\*Vorteil:\*\*** Egal ob du das Skript mehrfach ausführst oder Wrapper schon existieren: Das Container-Starten klappt immer, solange Wrapper und Command konsistent im XNAT-System vorhanden sind.

Wenn du noch Fragen hast, melde dich gern!

**Bousfiha, Tanae:**

tanae@mi0082n:~/dockerfile/xnat-osa/Ziel1\_osaDockerContainer\$ python3  
Containerenabled.py

XNAT WEB URL:<https://xnat-dev.gwdg.de>

XNAT Username:tanae

XNAT Password:

Project ID:tanae\_osa\_Predct

Session ID:XNAT\_E00428

Path to the Python script:Bonjour.py

Docker Name base image:python:3.10

What is the name of the command in XNAT: neutralisyis

What is the description of the command in XNAT: neutraliseri

What is the name of the Label in XNAT: neutrlserilisis

What is the description of the Label in XNAT: neutrlisiss

Dockerfile written to ./Dockerfile

Building Docker image 'neutralisyis:latest'...

Docker image 'neutralisyis:latest' built successfully.

JSON file created at command.json

Uploading command to <https://xnat-dev.gwdg.de/xapi/commands>

Command created successfully.

No wrapper found for this command.

Wrapper existiert noch nicht, wird erstellt...

Wrapper-Payload: {

  "name": "neutralisyis\_wrapper",

  "label": "neutrlserilis",

  "description": "neutrlisiss",

  "contexts": [

    "xnat:mrSessionData"

  ],

  "outputs": [

    {

      "name": "result\_file",

      "description": "Result",

      "required": true,

      "mount": "output\_mount",

      "path": "result.csv"

    }

  ],

  "external-inputs": [

    {

      "name": "session",

      "type": "Session",

      "required": true,

      "load-children": true

    }

  ],

  "derived-inputs": [

    {

      "name": "csv\_resource",

      "type": "Resource",

      "matcher": "@.label == 'CSV'",

      "required": true,

      "provides-files-for-command-mount": "input\_mount",

```

 "load-children": true,
 "derived-from-wrapper-input": "session",
 "multiple": false
 },
 {
 "name": "input_file",
 "type": "File",
 "matcher": "@.name =~ \".*\\.(csv|tsv|txt)$\"",
 "required": true,
 "load-children": true,
 "derived-from-wrapper-input": "csv_resource",
 "multiple": false
 },
 {
 "name": "input_file_name",
 "type": "string",
 "required": true,
 "load-children": true,
 "derived-from-wrapper-input": "input_file",
 "multiple": false
 }
],
"output-handlers": [
 {
 "name": "output",
 "accepts-command-output": "result_file",
 "as-a-child-of": "session",
 "type": "Resource",
 "label": "Results",
 "format": "csv"
 }
]
}

```

Wrapper creation failed: 400 - Invalid command:

Command "neutralisyis" - wrapper name  
 "neutralisyis\_wrapper" is not unique.

Wrapper konnte nicht erstellt werden, versuche existierenden Wrapper zu verwenden

...

No wrapper found for this command.

Kein Wrapper gefunden oder schwerwiegender Fehler – Abbruch.

**HAWKI** (gpt-4.1):

**\*\*Sehr gute und vollständige Fehleranalyse!\*\***

Dein Skript ist jetzt robuster gegen Wrapper-Fehler: Wenn das Anlegen fehlschlägt, versuchst du, einen existierenden Wrapper zu finden – **\*\*aber\*\*** auch das Find-Verfahren schlägt in deinem Fall fehl:

> Wrapper creation failed: ... is not unique.

> Wrapper konnte nicht erstellt werden, versuche existierenden Wrapper zu verwenden ...

> No wrapper found for this command.

> Kein Wrapper gefunden oder schwerwiegender Fehler – Abbruch.

**\*\*Das heißt.\*\***

Offenbar existiert **\*\*im System bereits ein Wrapper “neutralisyis\_wrapper” für dieses Command\*\***, aber deine Funktion ``get_wrapper_id_by_command_name`` findet ihn nicht.

---

**## Woran liegt das?**

**\*\*Die Funktion ``get_wrapper_id_by_command_name`` durchsucht nur ("wrappers": [...]),\*\***

aber XNAT speichert Wrapper-Objekte typischerweise im Feld ``"xnat"`` und nicht ``"wrappers"``!

Siehe dein JSON, das du erzeugst:

```
```json
"xnat": [
  {
    "name": "neutralisyis_wrapper",
    ...
  }
]
```

```
...
```

```
**Deine Funktion sucht aber:**
```

```
```python
```

```
for wrapper in command.get("wrappers", []):
```

```
...
```

```
Das Feld `wrappers` ist meist leer!
```

```
Richtig ist:
```

```
```python
```

```
for wrapper in command.get("xnat", []):
```

```
...
```

```
---
```

```
## **Wie löst du das konkret?**
```

```
**Ersetze deine Funktion wie folgt:**
```

```
```python
```

```
def get_wrapper_id_by_command_name(xnat_host, xnat_user, xnat_password,
wrapper_name):
```

```
 url = f"{xnat_host.rstrip('/')}/xapi/commands"
```

```
 resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
```

```
 if resp.status_code != 200:
```

```
 print(f"Error fetching commands: {resp.status_code}")
```

```
 sys.exit(1)
```

```
 # Korrektur: Jetzt auf "xnat" prüfen!
```

```
 for command in resp.json():
```

```
 for wrapper in command.get("xnat", []):
```

```
 if wrapper.get("name") == wrapper_name:
```

```
 # Typischerweise kein "id", nimm den Namen!
```

```
 return wrapper.get("id") or wrapper_name
```

```
 print("No wrapper found for this command.")
```

```
 sys.exit(1)
```

```
...
```

\*Evtl. musst du für ältere XNAT-Versionen\* auch beide Felder prüfen; meistens reicht

oben.\*

---

## **Was ist die Folge?**

- Jetzt wird dein existierender Wrapper gefunden!
- Der Ablauf läuft/ weiter zum Enable und zum Container-Start
- Die Namenskollision tritt nicht mehr als scheinbarer „Blocker“ auf.

---

## **Zusatzhinweise**

- Nach XNAT-Update sollte das Feld immer ``xnat`` heißen.
- Falls du auf mehreren Systemen testest, kannst du absichern:

```
```python
for field in ["xnat", "wrappers"]:
    for wrapper in command.get(field, []):
```

```
        ...
```
```

- Das Feld ``id`` im Wrapper ist **optional**. Wenn es fehlt, verwende den Namen (`=wrapper_name`).

---

## **Zusammenfassung**

- Der Fehler lag im Zugriff auf das falsche Feld (``wrappers`` statt ``xnat``).
- Nach Korrektur wird der existierende Wrapper gefunden und benutzt – dein Workflow ist endlich „idempotent“ und zuverlässig!

---

**Wissenschaftlicher Tipp:**

Das ist ein klassisches Beispiel für ein API-Schema-Drift-Problem! Regelmäßig Feldnamen und Objektstrukturen prüfen ist zentral für den nachhaltigen Einsatz von

Schnittstellen in Forschungsdaten-Management.

---

**\*\* Falls du möchtest, poste ich dir gern die komplette korrigierte Bescheid!\*\***

**Bousfiha, Tanae:**

ja bitte

**HAWKI (gpt-4.1):**

Sehr gerne! Hier ist die **\*\*korrigierte Version\*\*** der Funktion ``get_wrapper_id_by_command_name``, so dass sie sowohl im Feld ``"xnat"`` (Standard) als auch notfalls im Feld ``"wrappers"`` nach einem Wrapper sucht – ganz ohne klassischen Fehlerabbruch, sondern mit klarer Rückmeldung:

```
```python
def get_wrapper_id_by_command_name(xnat_host, xnat_user, xnat_password,
command_name, wrapper_name):
    url = f"{xnat_host.rstrip('/')}/xapi/commands"
    resp = requests.get(url, auth=(xnat_user, xnat_password), verify=False)
    if resp.status_code != 200:
        print(f"Error fetching commands: {resp.status_code}")
        sys.exit(1)
    data = resp.json()
    # Wenn nötig, "commands"-Verschachtelung auflösen:
    if isinstance(data, dict) and "commands" in data:
        commands = data["commands"]
    else:
        commands = data
    for command in commands:
        if command.get("name") == command_name:
            # Erst im Feld "xnat" suchen
            for wrapper in command.get("xnat", []):
                if wrapper.get("name") == wrapper_name:
                    # Falls "id" vorhanden, nimm id, sonst name
```

```

        return wrapper.get("id") or wrapper_name
    # Fallback: Ältere XNATs können wrapper auch unter "wrappers" speichern
    for wrapper in command.get("wrappers", []):
        if wrapper.get("name") == wrapper_name:
            return wrapper.get("id") or wrapper_name
    print("No wrapper found for this command.")
    sys.exit(1)
...

```

****Wichtige Hinweise:****

- Du solltest im Skript `get_wrapper_id_by_command_name(xnat_host, xnat_user, xnat_password, mod_data["command_name"], wrapper_name)` mit beiden Parametern aufrufen: `Command-Name` / und `Wrapper-Name`.
- Das Feld `"id"` im Wrapper ist manchmal nicht gesetzt. Für die meisten XNAT-Aufrufe reicht der Wrapper-Name.

****Integration:****

- Ersetze deine bisherige Funktion durch obigen Code.
- Passe die Aufrufe im Skript an, dass sowohl `command_name` als auch `wrapper_name` übergeben werden.

****Vorteil:****

Das Skript findet jetzt sauber deinen Wrapper – auch bei Leichen aus alten Läufen, auch nach Neustarts und bei XNAT-Updates.

****Falls die Funktion anderswo im Skript aufgerufen wird, hier ein Beispielaufruf:****

```

```python
wrapper_id = get_wrapper_id_by_command_name(xnat_host, xnat_user,
xnat_password, mod_data["command_name"], wrapper_name)
...

```

---

#### **\*\*Akademischer Hinweis:\*\***

Solider Umgang mit sich wandelnden API-Schemata ist wichtig für nachhaltige



( F o r s c h u n g s ) A u t o m a t i s i e r u n g .

Melde dich gern, falls du noch Fragen zur Integration oder weiteren Fehlerbehandlung hast!

**Bousfiha, Tanae:**

```
tanae@mi0082n:~/dockerfile/xnat-osa/Ziel1_osaDockerContainer$ python3
Containerenabled.py
XNAT WEB URL:https://xnat-dev.gwdg.de
XNAT Username:tanae
XNAT Password:
Project ID:tanae_osa_Predct
Session ID:XNAT_E00428
Path to the Python script:Bonjour.py
Docker Name base image:python:3.10
What is the name of the command in XNAT: simxdixi
What is the description of the command in XNAT: simixidixi
What is the name of the Label in XNAT: simixidixi
What is the description of the Label in XNAT: simixidixi
Dockerfile written to ./Dockerfile
Building Docker image 'simxdixi:latest'...
Docker image 'simxdixi:latest' built successfully.
JSON file created at command.json
Uploading command to https://xnat-dev.gwdg.de/xapi/commands
Command created successfully.
Traceback (most recent call last):
 File "/home/tanae/dockerfile/xnat-osa/Ziel1_osaDockerContainer/Containerena
bled.py", line 365, in <module>
 main()
 File "/home/tanae/dockerfile/xnat-osa/Ziel1_osaDockerContainer/Containerena
bled.py", line 334, in main
 wrapper_id = get_wrapper_id_by_command_name(xnat_host, xnat_user,
xnat_password, wrapper_name)
TypeError: get_wrapper_id_by_command_name() missing 1 required positional
argument: 'wrapper_name'
```

**HAWKI** (gpt-4.1):

Die Fehlermeldung

```
`TypeError: get_wrapper_command_name() missing 1 required positional argument:
'wrapper_name`
```

bedeutet: **\*\*Dein Funktionsaufruf benutzt aktuell noch KEINEN command\_name, sondern nur den wrapper\_name – seit deiner letzten Korrektur braucht die Funktion aber beide.\*\***

### Hintergrund

Du verwendest jetzt diese Funktion (siehe meine letzte Antwort):

```
```python
def get_wrapper_id_by_command_name(xnat_host, xnat_user, xnat_password,
command_name, wrapper_name):
    ...
```
```

**\*\*Du rufst sie aber so auf:\*\***

```
```python
wrapper_name = ... # korrekt zusammengesetzt
wrapper_id = get_wrapper_id_by_command_name(xnat_host, xnat_user,
xnat_password, wrapper_name)
# ----->      ^          ^          ^          ^
#              xnat_host   xnat_user   xnat_password wrapper_name
```
```

Der **\*\*command\_name\*\*** fehlt!

### **\*\*KORREKTE VERWENDUNG:\*\***

```
```python
wrapper_id = get_wrapper_id_by_command_name(xnat_host, xnat_user,
xnat_password, mod_data["command_name"], wrapper_name)
```
```

bzw. – entsprechend deinem Skript-Kontext:

```
```python
wrapper_id = get_wrapper_id_by_command_name(
    xnat_host,
    xnat_user,
    xnat_password,
    mod_data["command_name"], # <-- DAS musst du zusätzlich übergeben!
```
```

```
 wrapper_name
)
...
```

**\*\*Überall\*\*** dort, wo du ``get_wrapper_id_by_command_name`` aufrufst, musst du jetzt die Signatur **\*\*mit `command_name` und `wrapper_name`** verwenden!

---

**### \*\*Beispiel (Ausschnitt):\*\***

```
```python
# --- Wrapper anlegen oder holen (fehlerrobust!) ---
wrapper_id = None
try:
    wrapper_id = get_wrapper_id_by_command_name(
        xnat_host, xnat_user, xnat_password, mod_data["command_name"],
        wrapper_name
    )
    print(f"Wrapper existiert bereits: {wrapper_id}")
except SystemExit:
    ...
...
```
```

**### \*\*Zusammenfassung\*\***

- Die Funktion braucht jetzt **\*\*5 Parameter\*\***, nicht mehr 4.
- Passe dies an JEDER Stelle an, wo du sie aufrufst.

---

**#### \*\*Transfer auf dein Skript:\*\***

Ersetze also:

```
```python
wrapper_id = get_wrapper_id_by_command_name(xnat_host, xnat_user,
xnat_password, wrapper_name)
...
```
```

durch

```
```python
wrapper_id = get_wrapper_id_by_command_name(xnat_host, xnat_user,
```

```
xnat_password, mod_data["command_name"], wrapper_name)
...
```

Dann läuft es!

Bousfiha, Tanae:

```
tanae@mi0082n:~/dockerfile/xnat-osa/Ziel1_osaDockerContainer$ python3
Containerenabled.py
XNAT WEB URL:https://xnat-dev.gwdg.de
XNAT Username:tanae
XNAT Password:
Project ID:tanae_osa_Predct
Session ID:XNAT_E00428
Path to the Python script:Bonjour.py
Docker Name base image:python:3.10
What is the name of the command in XNAT: fiktonbase
What is the description of the command in XNAT: fiktionbase data
What is the name of the Label in XNAT: fiktionbasedata
What is the description of the Label in XNAT: fiktionbasedata
Dockerfile written to ./Dockerfile
Building Docker image &#039;fiktonbase:latest&#039;...
Docker image &#039;fiktonbase:latest&#039; built successfully.
JSON file created at command.json
Uploading command to https://xnat-dev.gwdg.de/xapi/commands
Command created successfully.
Wrapper existiert bereits: 91
Enable-URL: https://xnat-dev.gwdg.de/xapi/projects/tanae_osa_Predct/commands/fikt
onbase/wrappers/91/enable
Failed to enable wrapper: 404 - &lt;!doctype html&gt;&lt;html
lang=&quot;en&quot;&gt;&lt;head&gt;&lt;title&gt;HTTP Status 404 – Not
Found&lt;/title&gt;&lt;style type=&quot;text/css&quot;&gt;body
{font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b
{color:white;background-color:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3
{font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-color:
#525D76;border:none;}&lt;/style&gt;&lt;/head&gt;&lt;body&gt;&lt;h1&gt;HTTP Status
404 – Not Found&lt;/h1&gt;&lt;hr class=&quot;line&quot;
/&gt;&lt;p&gt;&lt;b&gt;Type&lt;/b&gt; Status
Report&lt;/p&gt;&lt;p&gt;&lt;b&gt;Description&lt;/b&gt; The origin server did not find a
```

current representation for the target resource or is not willing to disclose that one exists.</p><hr class="line" /><h3>Apache Tomcat/9.0.43 (Debian)</h3></body></html>

Launching container with payload: {

 "command": 70,

 "wrapper": 91,

 "inputs": {

 "session": "XNAT_E00428"

 }

}

Failed to launch container: 404 - <!doctype html><html

lang="en"><head><title>HTTP Status 404 – Not

Found</title><style type="text/css">body

{font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b

{color:white;background-color:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3

{font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-color:

#525D76;border:none;}</style></head><body><h1>HTTP Status

404 – Not Found</h1><hr class="line"

/><p>Type Status

Report</p><p>Description The origin server did not find a

current representation for the target resource or is not willing to disclose that one

exists.</p><hr class="line" /><h3>Apache Tomcat/9.0.43

(Debian)</h3></body></html>

HAWKI (gpt-4.1):

Analyse deines Problems

Du bist jetzt schon sehr weit!

Die Command- und Wrapper-Erkennung funktioniert.

Aber:

****Beide REST-Requests (`enable` und `launch`) geben Fehler 404 ("not found") zurück.****

Beispiel:

...

Enable-URL: https://xnat-dev.gwdg.de/xapi/projects/tanae_osa_Predct/commands/fikt_onbase/wrappers/91/enable

```
Failed to enable wrapper: 404 - ...
Launching container with payload: {
  "command": 70,
  "wrapper": 91,
  "inputs": {
    "session": "XNAT_E00428"
  }
}
Failed to launch container: 404 - ...
...
---
```

Das bedeutet der Fehler

- **404** bedeutet immer: „Diese Ressource existiert nicht.“ Der Server kennt die URL nicht oder sie ist im Kontext (Projekt, Command, Wrapper, Berechtigung) nicht registriert.
- Die IDs `command: 70`` und `wrapper: 91`` existieren anscheinend intern in XNAT, funktionieren aber in diesen URLs nicht.

Häufigste Ursachen für **404** bei enable/launch

1. **Falsche Wrapper-ID/Command-ID verwendet**

- Es gibt einen Unterschied zwischen interner numerischer ID (`id: 91``) und dem **Wrapper-Namen** (z.B. `"fiktonbase_wrapper"`).
- In der REST-API muss manchmal der Name und nicht die Datenbank-ID verwendet werden.

2. **Wrapper NICHT im Projekt aktiviert**

- Vor dem Launch muss der Wrapper für das Projekt tatsächlich aktiviert sein.
- Falls der "enable"-Aufruf fehlschlägt, ist der Wrapper im Projekt nicht verknüpft.

3. **IDs passen nicht zum Projekt-Kontext**

- Wenn du eine ID aus einer globalen Liste in einer Projekt-spezifischen URL verwendest, kommt 404.

4. ****Der Wrapper ist im Command zwar verfügbar, aber nicht aktiviert oder nicht korrekt zugeordnet.****

Was ist zu TUN?

1. ****Unterscheide: Braucht die API den Wrapper-Namen oder die Wrapper-ID?****

- Prüfe in der [XNAT Container REST API

Doku](<https://wiki.xnat.org/container-service/container-service-api>), ob

`/xapi/projects/{project}/commands/{command}/wrappers/{id}/enable` die ****ID**** oder den ****Namen**** des Wrappers erwartet.

- Bei vielen XNAT-Setups ist dies ****der Name des Wrappers**** (z.B.

`"fiktonbase_wrapper"`) – keinesfalls die DB-ID (91).

- Gleiches gilt für den `"command"-Parameter im Payload für den Launch: oft der Bezeichner (bspw. `"fiktonbase"`) oder der Name, nicht die technische ID.

****Teste:****

- Gibt es eine URL `/xapi/projects/tanae_osa_Predct/commands/fiktonbase/wrappers/fiktonbase_wrapper/enable` (mit ****Namen statt ID****)?

- Entsprechend auch für den Launch: `"wrapper": "fiktonbase_wrapper"`.

2. ****Prüfe nochmal das Mapping in deinem Skript und im XNAT-UI****

- Ist der Wrapper dem Projekt korrekt zugeordnet?

3. ****Passe die Skript-Aufrufe an:****

****Ändere die Enable-Funktion auf Name:****

```
```python
```

```
enable_url = f"{xnat_host.rstrip('/')}/xapi/projects/{project_id}/commands/{command_name}/wrappers/{wrapper_name}/enable"
```

```
```
```

statt

```
```python
```

```
/wrappers/{wrapper_id}/enable
```

```
...
```

```
Passe analog den Start/Launch-Payload an:
```

```
```python
```

```
payload = {
```

```
    "command": command_name,
```

```
    "wrapper": wrapper_name,
```

```
    ...
```

```
}
```

```
```
```

```
4. **Teste im Web-Browser:**
```

- Gehe zur REST-API-URL mit dem Namen, nicht der Zahl – liefert sie 200 oder wieder 404?

```

```

```
Korrigiertes Beispiel für deinen Code
```

```
`enable_wrapper`:
```

```
```python
```

```
def enable_wrapper(xnat_host, project_id, command_name, wrapper_name,  
xnat_user, xnat_password):
```

```
    enable_url = f"{xnat_host.rstrip('/')}/xapi/projects/{project_id}/commands/{command  
_name}/wrappers/{wrapper_name}/enable"
```

```
    print(f"Enable-URL: {enable_url}")
```

```
    resp = requests.post(enable_url, auth=(xnat_user, xnat_password), verify=False)
```

```
    if resp.status_code == 200:
```

```
        print("Wrapper successfully enabled.")
```

```
    else:
```

```
        print(f"Failed to enable wrapper: {resp.status_code} - {resp.text}")
```

```
...
```

```
#### Launch-Payload (beispielhaft):
```

```
Im Funktionsaufruf:
```

```
```python
```

```
run_container_in_xnat(xnat_host, wrapper_name, session_id, xnat_user,
xnat_password, command_name)
```



...

Und dann:

```
```python
def run_container_in_xnat(xnat_host, wrapper_name, session_id, xnat_user,
xnat_password, command_name):
    url = f"{xnat_host}/xapi/container-launcher/launch"
    headers = {"Content-Type": "application/json"}
    payload = {
        "command": command_name,
        "wrapper": wrapper_name,
        "inputs": {"session": session_id}
    }
    print("Launching container with payload:", json.dumps(payload, indent=2))
    response = requests.post(url, auth=(xnat_user, xnat_password), headers=headers,
json=payload, verify=False)
    if response.status_code in [200, 201]:
        print("Container launched successfully.")
    else:
        print(f"Failed to launch container: {response.status_code} - {response.text}")
...

```

Fazit & Empfehlung

- ****In REST-URLs und im Payload lieber Namen (Strings), keine Integer-IDs verwenden****, sofern die API nicht explizit Integer verlangt.
- ****Fehler 404**** bei enable/launch: Sehr häufiges Zeichen, dass die ID falsch war oder Wrapper einfach nicht (richtig) aktiviert ist.

Bitte passe die Aufrufe auf String-Werte (Wrapper-Namen, Command-Namen) an – damit sollte die 404 verschwinden.

Wissenschaftlich

Das ist ein klassisches Mapping-Problem bei API/Namensraum
Name in relationalen APIs und wissenschaftlichen Plattformen, die mit
Objekthierarchien arbeiten.

****Bei weiteren Fehlern (immer noch 404, trotz Anpassung), poste die neuen URLs
und ggf. den aktuellen Auszug deiner enable- und run_container-Funktionen, dann
helfe ich gezielt weiter!****

Bousfiha, Tanae:

command-event-mapping-rest-api
Command Event Mapping Rest Api

GET
/commandeventmapping
Get all Command-Event Mappings

POST
/commandeventmapping
Create a Command-Event Mapping

GET
/commandeventmapping/{id}
Get a Command-Event-Mapping

DELETE
/commandeventmapping/{id}
Delete a Command-Event Mapping

POST

/commandeventmapping/{id}/convert

Convert a Command-Event Automation item to an Event Service Subscription

PUT

/commandeventmapping/{id}/disable

Disable a Command-Event Mapping

PUT

/commandeventmapping/{id}/enable

Enable a Command-Event Mapping

HAWKI (gpt-4.1):

Die `Command Event Mapping Rest Api` in XNAT ist ein anderer Mechanismus als das direkte Wrapper-Aktivieren für ein konkretes Projekt (wie bei `/xapi/projects/{project}/commands/{command}/wrappers/{wrapper}/enable`).

Mit Event-Mappings kannst du festlegen, dass bestimmte Wrapper automatisch bei bestimmten "Events" (z./B. Session erstellt, Datei hochgeladen) im System oder projektweit ausgeführt werden.

****Was ist ein Command-Event-Mapping?***

- Es verknüpft einen Wrapper mit einem ****Event**** (z./B. „Neu angelegt“ ! dann führe Wrapper XY aus).
- Es ist nicht das Gleiche wie das einfache manuelle Aktivieren eines Wrappers (damit ein User ihn per Knopfdruck im UI starten kann).
- Die Endpunkte:
 - `/commandeventmapping` (und `/commandeventmapping/{id}`): Zum Listen, Erstellen, Aktualisieren, Löschen von Event-Mappings.
 - `/commandeventmapping/{id}/enable` und `/disable`: (De)aktivieren ein Event Mapping.
 - `/commandeventmapping/{id}/convert`: Migration ins „Event Service“-System (für moderne Automatisierungswflows in XNAT).

****Wann brauchst du diese?***

- ****Nur****, wenn du möchtest, dass dein Wrapper automatisch bei systemweiten oder projektspezifischen Ereignissen getriggert wird (_Automation_).

- ****Nicht notwendig****, wenn du nur "per Hand" Container-Wrapper für ein Projekt aktivieren/ausführen willst.

****Dein Use Case bis jetzt****

- Du willst einen Container(-Wrapper) manuell/für ein Projekt aktivieren und dann mit Nutzereingaben ausführen.
- Dafür brauchst du keinen Command-Event-Mapping-Rest-API-Aufruf.
- Das bisherige Problem (404) hat nichts mit Event-Mapping zu tun, sondern mit der richtigen Nutzung von Namen/IDs beim Wrapper-Enable und Container-Launch.

Praxis-Fazit:

****Für deine Skript-Lösung:****

- Ignoriere die Command Event Mapping Rest Api_, du brauchst sie ****nicht**** für das reine "Wrapper für Projekt aktivieren und Container starten".
- Stelle sicher, dass du für alle `/enable`/`/launch`-Actions den ****Namen**** des Wrappers verwendest (siehe vorherige Antwort).
- Teste Wrapper-Aktivierung und -Launch weiter wie bisher – aber pass ggf. die URL-Parameter (Name statt ID) an!

****Nur falls Event-gesteuerte Automatisierung (z. B. nach Upload/Creation):****

- Dann wäre die Command-Event-Mapping-Rest-API relevant, um deinen Wrapper mit einem Systemereignis zu verknüpfen.

****Zusammengefasst:****

Für das „normale“ (manuelle oder Skript-basierte) Starten/Aktivieren eines Wrappers/Containers brauchst du die Command-Event-Mapping-API nicht. Diese wird nur benötigt, wenn Workflows ****automatisch**** durch ein Ereignis ausgelöst werden sollen.

Falls du dazu ein Automatisierungsszenario möchtest (z.B. "Sühne führe mein Skript aus"), melde dich, dann erkläre ich gern, wie du ein entsprechendes Mapping erzeugst.

Bousfiha, Tanae:

wie kan ich das problem beheben und den container mit meinem skript debuggen oder runnen

HAWKI (gpt-4.1):

Das ist eine sehr wichtige und praxisnahe Frage, die sich viele Forschende oder Entwickler an der HAWK aber auch an anderen Hochschulen stellen! Nachfolgend findest du eine **wissenschaftliche und praxisnahe Antwort**, die sowohl das konkrete Debugging als auch typische Ursachen, Lösungswege und systematische Vorgehensweisen abdeckt.

1. Fehlerursachen: Warum schlägt das Starten/Debuggen (bisher) fehl?

In XNAT-Container-Workflows gibt es/vor allem drei Hürden:

- **IDs statt Namen benutzt:** Viele XNAT-REST-Endpunkte/numerische Datenbank-ID eines Commands/Wrappers, sondern deren **Name** (z.B. `"fiktonbase_wrapper"` statt `91`).
- **Wrapper wurde nicht (richtig) für das Projekt aktiviert** (404 bei `/enable`)
- **Eingaben/Projekt-Kontext fehlen** (z.B. ist der Wrapper z für das richtige Projekt aktiv)

2. Systematisches Debuggen und Korrektur

Schritt 1: Wrapper-/Command-Aktivierung und Start-URL prüfen

Prüfe, ob du für „Enable Wrapper“ und den Launch im Skript wirklich den **Namen** verwendest, z.B.:

```
python
```

```
# Nutze Namen, nicht IDs!
```

```
enable_url = f"{xnat_host.rstrip('/')}/xapi/projects/{project_id}/commands/{command_n
```

```
ame}/wrappers/{wrapper_name}/enable"
```

```
...
```

und für den Start (Payload):

```
```python
```

```
payload = {
```

```
 "command": command_name, # Name, z.B. "fiktonbase"
```

```
 "wrapper": wrapper_name, # Name, z.B. "fiktonbase_wrapper"
```

```
 "inputs": {
```

```
 "session": session_id
```

```
 # ggf. weitere Inputs je nach Wrapper-Definition!
```

```
 }
```

```
}
```

```
...
```

\*Baue das explizit im Skript ein, statt IDs zu nehmen!\*

```

```

### Schritt 2: \*\*REST-Requests außerhalb des Skripts debuggen\*\*

Nutze [Postman](https://www.postman.com/) oder `curl`, um d  
Launch-Request manuell gegen deinen XNAT-Server zu schicken

```
```sh
```

```
curl -u <user>:<password> -X POST "https://xnat-dev.gwdg.de/xapi/projects/tanae_os  
a_Predct/commands/fiktonbase/wrappers/fiktonbase_wrapper/enable"
```

```
...
```

Du bekommst sofort zurück, ob die URL stimmt (200/201 OK) oder nicht (404 Not Found).

Das gleiche kannst du für den Launch machen:

```
```sh
```

```
curl -u <user>:<password> -X POST -H "Content-Type: application/json" \
```

```
 -d '{"command":"fiktonbase","wrapper":"fiktonbase_wrapper","inputs":{"session":"XN
AT_E00428"}}' \
```

```
 "https://xnat-dev.gwdg.de/xapi/container-launcher/launch"
```

```
...
```

- Wenn das manuell funktioniert, ist dein Skript fehlerhaft.
- Wenn 404 auch hier/kommt, stimmt wahrscheinlich der Name nicht.

---

### ### Schritt 3: \*\*Überprüfe deine Daten in XNAT Web-UI\*\*

- Sieh unter „Admin > Container Service > Commands“, welche Commands/Wrappers existieren und wie die Namen heißen.
- Prüfe im Projekt „Commands“/„Wrappers“ regelmäßig, ob sie für das Projekt aktiviert sind.

---

### ### Schritt 4:/\*\*Im Skript Logging/Debug-Ausgaben nutzen\*\*

Baue ZWISCHEN jedem REST-Request ein ``print("I call the following URL ...")`` und lasse dir Rückgabewerte direkt anzeigen, etwa:

```
```python
print("Enable-URL:", enable_url)
print("Payload:", payload)
print("Status:", response.status_code)
print("Response:", response.text)
```
```

> Damit kannst du ggf. kleine Tippfehler im Namen, im Projektnamen oder fehlende/doppelte Unterstriche direkt erkennen.

---

### ### Schritt 5: Fehlerhandlung robuster machen

Achte darauf, wie du `wrapper_id` und `wrapper_name` unterscheidest und benutze im Zweifel die Namen/in den REST-Aufrufen!

---

### ## 3. Beispiel: vollständige "Enable & Launch"-Funktionen (anpassen!)

Deine Funktionen sollten dann SO aussehen:

```
```python
def enable_wrapper(xnat_host, project_id, command_name, wrapper_name,
xnat_user, xnat_password):
    enable_url = f"{xnat_host.rstrip('/')}/xapi/projects/{project_id}/commands/{command
_name}/wrappers/{wrapper_name}/enable"
    print(f"Enable URL: {enable_url}")
    response = requests.post(enable_url, auth=(xnat_user, xnat_password),
verify=False)
    print("Enable Response:", response.status_code, response.text)
    if response.status_code == 200:
        print("Wrapper enabled")
    else:
        print("Could not enable wrapper")

def run_container_in_xnat(xnat_host, command_name, wrapper_name, session_id,
xnat_user, xnat_password):
    url = f"{xnat_host.rstrip('/')}/xapi/container-launcher/launch"
    payload = {
        "command": command_name,
        "wrapper": wrapper_name,
        "inputs": {"session": session_id}
    }
    print(f"Launching container, payload:\n{json.dumps(payload, indent=2)}")
    response = requests.post(url, auth=(xnat_user, xnat_password), json=payload,
verify=False)
    print("Launch Response:", response.status_code, response.text)
    if response.status_code in [200, 201]:
        print("Container launched successfully!")
    else:
        print("Could not launch container")

# Im main-Aufruf:
enable_wrapper(xnat_host, project_id, command_name, wrapper_name, xnat_user,
xnat_password)
```



```
run_container_in_xnat(xnat_host, command_name, wrapper_name, session_id,  
xnat_user, xnat_password)  
``
```

4. Alternativen & Tipps für mehr Debugging

- Prüfe auch, ob der User, mit dem du dich verbindest, wirklich Projekt-Admin ist!
- Wenn Launch-Status >202 ist, prüfe die XNAT-Logs auf dem Server (Systemadministrator fragen, falls kein Zugang).
- Nutze die `-v`` (verbose) Option bei ``curl`` für mehr Details über den Request/Response-Dialog.

5. Fazit

- Auf „Namen vs. ID“ achten
- Manuell im Browser oder per ``curl``/Postman testen
- Im Skript alle URLs und Payloads ausgeben und alles vergleichen
- Immer XNAT-Logs konsultieren, falls REST-responses nicht helfen (404, 403, 500 etc.)
- Wrapper nach jeder kleineren Änderung neu für das Projekt aktivieren

****Wenn du ganz konkret eine einzelne curl- oder requests-Zeile zum Debuggen brauchst, sag einfach, welchen Command/Wrapper du testen willst!****

Wissenschaftlicher Hinweis:

Reproduzierbares Debugging ist zentral für Forschungsdatenprozesse und Containerisierung – dokumentiere deine Kombinationen aus Project, Command, Wrapper und Inputs, damit du und deine Kolleg*innen nachvollziehen können, was wie (und warum nicht) funktioniert.

****Melde dich gern mit konkreten Zwischenständen – ich unterstütze dich Schritt für Schritt weiter!****

Bousfiha, Tanae:

kann das sein weil ich die image nicht getaggt oder gepusht auf dockerhub?

HAWKI (gpt-4.1):

Sehr gute und reflektierte Frage!

Hier die wissenschaftlich fundierte und praxisorientierte Antwort:

****Kurze Antwort:****

Das von dir beschriebene Problem (404 bei Enable/Launch) liegt mit sehr hoher Wahrscheinlichkeit ****nicht**** am fehlenden Tagging oder daran, dass du das `Docker-Image nicht auf DockerHub gepusht hast / — / jedenfalls Image-Build lokal oder via privaten Registry auf dem selben Host betreibt` (was in Entwicklungsumgebungen Standard ist).

Warum meistens nicht das Problem?

- ****Lokaler Build/Run:****

Wenn dein XNAT so konfiguriert ist, dass der XNAT-Container Service auf dem gleichen Docker-Host arbeitet, reicht es, das Image wie von dir gebaut lokal vorzuhalten (z.B. ``fiktonbase:latest``). Dann muss das Image ****nicht**** aus DockerHub geloadet werden, d.h. Push ist nicht nötig.

- ****404-Fehler:****

Ein 404 bedeutet: Die Ressource ist am Server ****nicht bekannt****/existiert nicht (Wrapper oder Command oder Aktivierungs-/Start-URL).

Hier würde bei fehlendem Image eher der Container nicht starten oder im Log auftauchen ("image not found: ..."), aber ****nicht 404****, sondern ein Fehler beim Container-Start (oft 500er-Fehler oder interne Logs).

Wann wird Push/Tagging relevant?

1. ****Image in private/public Registry nötig:****

Wenn der XNAT-Container-Service auf einem anderen Host arbeitet *(Remote Docker

Daemon)*, ****muss das Image in einer für XNAT erreichbaren Registry liegen**** (z.B. DockerHub, GitLab, eigene Registry).

2. ****Image-Tag relevant:****

Wenn du nicht mit `:latest` arbeitest oder die JSON explizit ein anderes Tag angibt, dann muss das passende Tag als Image vorliegen.

3. ****Fehlendes Image:****

****Dann kommt (im XNAT-Container-Log):****

- "Error: No such image: fiktonbase:latest"
- ****ODER**** ein 500er-Fehler beim Container-Launch, aber kein 404 bei `/enable`!

Wie kannst du das testen?

- ****Lokal:****

Nach deinem Skript-lauf, gib in der Shell ein:

```
```bash
docker images | grep fiktonbase
```
```

Du siehst: Name, Tag, Image-ID.

Das reicht für einen lokalen XNAT, der auf dem Host Docker ausführt.

- ****Im Fehlerfall:****

Wenn du ein Image benutzt, das ****nicht lokal vorliegt****, bekommst du den Fehler oft erst beim Container-Start, nicht schon beim Enablen oder Registrieren des Commands/Wrappers.

- ****Im Log:****

Fehler wie „image not found“ stehen oft in den XNAT-Container-Logs, nicht als 404 im REST-Request.

Empfehlung: Was solltest du tun?

1. ****Teste zunächst, ob das Image lokal verfügbar ist**** *(ohne Push)*, indem du den Standardlauf versuchst.
2. ****Erst wenn der Container mit dem Fehler „image not found“ im Log abstürzt:****
 - Tagge das Image exakt wie in deinem `command.json` als `repository:tag`.
 - Pushe es auf DockerHub, wenn dein XNAT-Setup auf eine externe Registry

zugreifen muss.

- Im JSON solltest du ggf. `"image": "deinusername/fiktonbase:latest"` angeben und passend bauen/pushen.

Zusammengefasst:

- ****404 bei Enable/Launch:**** Fast immer ein Problem mit Wrapper/Command-Namen/ID oder Berechtigung (nicht Docker-Image!).
- ****Fehlendes Image:**** Typischerweise Fehler erst beim eigentlichen Ausführen („image not found“).
- ****Image muss NUR dann getaggt/gepusht werden,**** wenn XNAT/Container-Service nicht auf demselben Docker-Host läuft, sondern Images aus einer Registry zieht.

****TIPP:****

Suche im XNAT WebUI nach Logs, sobald ein Container-Job scheduliert ist!

**** Nur wenn explizit „image not found“ kommt! ’ docker push n**

Wenn du unsicher bist, ob du Push brauchst, poste gern die relevante `image`-Zeile aus deinem `command.json` sowie, ob XNAT und Docker auf demselben Host laufen (oder frag deinen Systemadmin/dein Rechenzentrum).

****Fragen?****

Melde dich gern mit Logs/Ausgaben, dann kann ich ganz konkret beurteilen, ob Push/Tagging relevant ist!

Bousfiha, Tanae:

when ich den container manuelle ausführen möchte bekomme ich dies efehelr meldung

Error

Error undefined: error

Could not launch UI with value: "XNAT_E00428" for root element: "session".

HAWKI (gpt-4.1):

Das ist eine häufige, aber oft undurchsichtige Fehlermeldung, die bei XNAT Container-Workflows auftreten kann – insbesondere im Zusammenhang mit Wrappers und Session-IDs:

****Fehlermeldung:****

> Error undefined: error

> Could not launch UI with value: "XNAT_E00428" for root element: "session".

Was bedeutet das?

- ****"root element: session":****

Dein Container-Wrapper ist so konfiguriert, dass er eine Session (z.B. `XNAT_E00428`) als Root-Objekt erwartet.

- ****"Could not launch UI with value: ...":****

XNAT kann die Session-ID nicht korrekt auflösen. Das passiert oft, wenn der Kontext im Wrapper oder in den Projektrechten nicht korrekt gesetzt ist.

****Typische Ursachen und Lösungsansätze****

1. ****Session-ID stimmt nicht oder hat keine passenden Ressourcen****

- Prüfe, ob die Session-ID wirklich existiert (im XNAT-UI den Scan suchen).
- Prüfe, ob an der Session die erwarteten Daten/Ressourcen (z.B. ein "CSV", Bilddaten usw.) hinterlegt sind.

2. ****Wrapper-Kontext und Inputs stimmen nicht mit den XNAT-Daten überein****

- Dein Wrapper verlangt `external-inputs` vom Typ ****Session**** (z.B. `"name": "session", "type": "Session"`).
- ****Aber:**** Wenn dem Wrapper z.B. ein MR-Scan (statt einer Session) oder eine Ressource fehlt, funktioniert es nicht.
- Prüfe, ob der Kontext (`contexts`: z.B. `xnat:mrSessionData`) und die Inputs/Ressource-Matcher ****zur ausgewählten Session passen****.

3. ****Session nicht im richtigen Projekt****

- Die Session muss sich im selben Projekt befinden, in dem auch der Wrapper aktiviert ist.

4. ****Ressourcen-Matcher im Wrapper zu restriktiv****

- Prüfe im Wrapper (JSON), ob die Matcher (Beispiel: `@.label == 'CSV'`) auf ein

vorhandenes Resource-Label passen.

- Wenn deine Session kein Resource-Label "CSV" hat, wird sie nicht "gefunden" und verursacht Fehler.

5. ****Berechtigungen fehlen****

- Dein User braucht mindestens Leserechte auf der Session und (meist) Execute auf den Wrapper.

****Was tun? — Schrittweise Fehlerdiagnose und Lösung****

1. Existiert die Session wirklich?

- Suche in XNAT nach „Sessions“ und prüfe, ob `XNAT_E00428` existiert und im richtigen Projekt liegt.

2. Hat die Session die geforderten Ressourcen?

- Gehe im UI in die Details der Session `XNAT_E00428`.
- Siehst du dort unter „Manage files“ oder „Resources“ einen Eintrag mit Label „CSV“, oder wie im Wrapper verlangt?
- Wenn nein:
 - **Lege die gewünschte Datei/Resource mit dem richtigen Label an!****

3. Stimmen die Context/Matcher-Einstellungen im Wrapper?

- Überprüfe im Wrapper-Payload den Context:
 - ` "contexts": ["xnat:mrSessionData"]`
- Passt das zum Typ deiner Session (MR, CT, PET)? Möglicher Wert:
****xnat:mrSessionData**** für MR, ****xnat:ctSessionData**** für CT usw.
- Passen die "Matcher"-Angaben zur Session und Resource?

4. Teste den Wrapper für eine andere Session

- Evtl. läuft der Wrapper mit anderen Sessions problemlos (andere Daten, Projektzugehörigkeit).

5. Logs prüfen

- Prüfe, ob die Server-Logs (XNAT und Container-Service) detailliertere Fehler bieten.

****Typische Sofortmaßnahmen****

- Lade eine „Test-Ressource“ unter deiner Session mit dem passenden Label hoch

und starte dann den Job erneut.

- Passe ggf. im Wrapper die Matcher so an, dass sie nicht auf ein Label prüfen, sondern allgemeiner sind, zum Test:

```
```json
"matcher": "true"
```
```

Dann findest du heraus, ob das Problem am Label liegt.

****Fazit****

- Session-ID existieren und zum Kontext passen.
- Richtige(n) Resource(s) mit richtigem Label an der Session.
- Wrapper-Kontext und Matcher-Einstellungen kontrollieren.
- Projekt- und Wrapper-Aktivierung korrekt?
- Rechte prüfen.

****Falls du das Wrapper-JSON posten willst (aus command.json), kann ich konkret sagen, ob die Matcher und Inputs diese Session überhaupt finden können und was ggf. geändert werden muss!****

Melde dich, wenn du einen Wrapper-Abschnitt oder Session-Info posten möchtest – dann bekommst du gezielt Unterstützung!

Bousfiha, Tanae:

```
{
  "name": "fiktonbase",
  "description": "fiktionbase data",
  "version": "1.0",
  "image": "fiktonbase:latest",
  "type": "docker",
  "command-line": "python /app/Bonjour.py
/app/input/#INPUT_FILE# /app/output",
  "mounts": [
    {
      "name": "output_mount",
      "writable": true,
      "path": "/app/output"
```

```

    },
    {
        "name":: "input_mount",
        "writable":: false,
        "path":: "/app/input"
    }
],
"environment-variables":: {},
"ports":: {},
"inputs":: [
    {
        "name":: "INPUT_FILE",
        "description":: "fiktionbase data",
        "type":: "string",
        "required":: true,
        "select-values":: []
    }
],
"outputs":: [
    {
        "name":: "result_file",
        "description":: "Result",
        "required":: true,
        "mount":: "output_mount",
        "path":: "result.csv"
    }
],
"xnat":: [
    {
        "name":: "fiktonbase_wrapper",
        "label":: "fiktionbasedata",
        "description":: "fiktionbasedata",
        "contexts":: [
            "xnat:mrSessionData"
        ]
    }
],
"external-inputs":: [
    {
        "name":: "session",

```



```

    &quot;type&quot;:: &quot;Session&quot;,
    &quot;required&quot;:: true,
    &quot;load-children&quot;:: true
  }
],
&quot;derived-inputs&quot;:: [
  {
    &quot;name&quot;:: &quot;csv_resource&quot;,
    &quot;type&quot;:: &quot;Resource&quot;,
    &quot;matcher&quot;:: &quot;@.label == &#039;CSV&#039;&quot;,
    &quot;required&quot;:: true,
    &quot;provides-files-for-command-mount&quot;:: &quot;input_mount&quot;,
    &quot;load-children&quot;:: true,
    &quot;derived-from-wrapper-input&quot;:: &quot;session&quot;,
    &quot;multiple&quot;:: false
  },
  {
    &quot;name&quot;:: &quot;input_file&quot;,
    &quot;type&quot;:: &quot;File&quot;,
    &quot;matcher&quot;:: &quot;@.name =~ \&quot;.*\\.(csv|tsv|txt)$\&quot;&quot;,
    &quot;required&quot;:: true,
    &quot;load-children&quot;:: true,
    &quot;derived-from-wrapper-input&quot;:: &quot;csv_resource&quot;,
    &quot;multiple&quot;:: false
  },
  {
    &quot;name&quot;:: &quot;input_file_name&quot;,
    &quot;type&quot;:: &quot;string&quot;,
    &quot;required&quot;:: true,
    &quot;load-children&quot;:: true,
    &quot;derived-from-wrapper-input&quot;:: &quot;input_file&quot;,
    &quot;multiple&quot;:: false
  }
],
&quot;output-handlers&quot;:: [
  {
    &quot;name&quot;:: &quot;output&quot;,
    &quot;accepts-command-output&quot;:: &quot;result_file&quot;,

```

```

        &quot;as-a-child-of&quot;:: &quot;session&quot;,
        &quot;type&quot;:: &quot;Resource&quot;,
        &quot;label&quot;:: &quot;Results&quot;,
        &quot;format&quot;:: &quot;csv&quot;,
        &quot;tags&quot;:: []
    }
]
}
],
&quot;container-labels&quot;:: {},
&quot;generic-resources&quot;:: {},
&quot;ulimits&quot;:: {},
&quot;secrets&quot;:: []
}

```

so sieht dne command json und die session existiert
und den external Skript sieht so aus

```

import sys
import os

```

Argumente

```

input_file_name = sys.argv[1]
output_dir = sys.argv[2]

```

Absoluter Pfad zum Input-File im Docker-Container

```

input_file = os.path.join(&quot;/app/input&quot;, input_file_name)

```

Output-Verzeichnis anlegen

```

os.makedirs(output_dir, exist_ok=True)
output_path = os.path.join(output_dir, &quot;result.csv&quot;)

```

Prüfung und Ausgabe

```

if not os.path.exists(input_file):
    with open(output_path, &quot;w&quot;) as f:
        f.write(&quot;input file not found\n&quot;)
    sys.exit(0)

```

```

if os.path.getsize(input_file) == 0:

```

```

    with open(output_path, &quot;w&quot;) as f:

```

```
f.write(&quot;input file is empty\n&quot;)\n\nsys.exit(0)
```

```
with open(output_path, &quot;w&quot;) as f:\n\n    f.write(&quot;success\n&quot;)
```

istthere any mistakes ?

HAWKI (gpt-4.1):

****Sehr gute, strukturierte Darstellung!****

Dein `command.json` und dein Python-Script sehen auf den ersten Blick technisch sauber aus. Allerdings gibt es ****ein zentrales Problem**** mit Bezug auf die Ressourcenstruktur in XNAT, das die Launch-Fehler ("Could not launch UI with value... for root element: session") verursacht.

Analyse & Empfehlungen

1. **Matcher und Resource-Label**

Dein Wrapper erwartet als required input:

```
```\njson\n{\n  "name": "csv_resource",\n  "type": "Resource",\n  "matcher": "@.label == 'CSV'",\n  "required": true,\n  "provides-files-for-command-mount": "input_mount",\n  "load-children": true,\n  "derived-from-wrapper-input": "session",\n  "multiple": false\n}\n```\n
```

Das bedeutet:

- **\*\*Deine MR-Session (`XNAT\_E00428`) muss eine Resource mit Label `CSV` enthalten!\*\***

- **\*\*Diese Resource muss mindestens eine Datei (z.B. .csv, .tsv, .txt) enthalten.\*\***

### ### 2. **\*\*input\_file Matcher\*\***

```
```json
{
  "name": "input_file",
  "type": "File",
  "matcher": "@.name =~ \".*\\.(csv|tsv|txt)$\"",
  "required": true,
  ...
}
```

Es wird gesucht: Eine Datei in der Resource mit Namen/Endung `.csv`/` .tsv`/` .txt``.

****Wenn beides fehlt, erscheint genau dein Fehler!****

3. ****Checkliste für dich****

- ****Gibt es im Session-UI zu ``XNAT_E00428`` unter "Manage files/Resources" einen Eintrag ``CSV``?****

- NEIN: `!` Muss hochgeladen werden!`

- JA: Enthält die Resource tatsächlich eine Datei namens ``...csv``, ``...tsv`` oder ``...txt``?

- NEIN: Eine Datei der passenden Endung hinzufügen.

****Nur DANN kann dein Wrapper/Command UI erfolgreich gestartet werden.****

4. ****Python Script-Argumente****

Dein Skript ist robust/– es nimmt:

- ``sys.argv[1]`` `!` Filename`

- ``sys.argv[2]`` `!` Output-Verzeichnis`

****Das passt exakt zu deinem command-line und mount im Docker:****

```
```json
"command-line": "python /app/Bonjour.py /app/input/#INPUT_FILE# /app/output",
```

```
"mounts": [
 {"name": "input_mount", ... "/app/input"},
 {"name": "output_mount", ... "/app/output"}
]
...
```

**\*\*Das bedeutet\*\*:**

Das Resource-File wird nach `/app/input` gemountet, und der Filename wird als Argument übergeben.

\_! P a s s t e b e n f a l l s ! \_

---

### ### 5. **\*\*Testing/Debugging-Tipp\*\***

- Gehe im XNAT-UI ins Projekt, rufe die Session `XNAT\_E00428` auf.
- „Manage Files“ > **\*\*Ressource hinzufügen\*\*** (Create Resource) mit Label **\*\*CSV\*\***.
- Lade eine Datei z.B. `test.csv` hoch.
- **\*\*Starte das Command nochmal.\*\***

---

### ### 6. **\*\*Optional: Matcher für Debugging aufweichen\*\***

Für den Anfang könntest du in `command.json` temporär den Matcher lockern:

```
``json
"matcher": "true"
...
```

Dann werkt das System mit jeder beliebigen Resource. (Später wieder spezifizieren!)

---

## ## Zusammenfassung

- **\*\*Dein Fehler liegt sehr wahrscheinlich daran, dass keine oder keine passende Resource (`CSV`) an der Session vorhanden ist.\*\***
- **\*\*Deine Script- und JSON-Konfiguration sind ansonsten in Ordnung.\*\***
- **\*\*Füge testhalber eine CSV-Resource samt passender Datei hinzu – dann sollte**

alles laufen.\*\*

---

### ### Wissenschaftlicher Kontext

Das Vorgehen entspricht dem Standardmodell wissenschaftlicher Workflow-Engines: Ressourcen und Matcher müssen exakt zu den tatsächlichen Metadaten und Dateistrukturen in der Datengrundlage passen. Dieses Prinzip sichert die Reproduzierbarkeit und verhindert fehlerhafte oder unvollständige Analyseläufe.

---

**\*\*Falls es nach diesem Test immer noch einen Fehler gibt, nenne gern den exakten Aufbau deiner Resources/Ressourcenstruktur (Screenshot oder Beschreibung), dann kann ich noch gezielter helfen!\*\***

### **Bousfiha, Tanae:**

doch es gibt eine datei unter Resources >>CSV>>correct\_patient.csv

### **HAWKI (gpt-4.1):**

Danke für die Rückmeldung! Das hilft, gezielter zu analysieren.

**\*\*If your XNAT Session (XNAT\_E00428) has a resource "CSV" and a file called correct\_patient.csv, then in principle the wrapper inputs and matcher should work.\*\***  
Dennoch gibt es noch einige Fallstricke, die du prüfen solltest:

---

### ### Mögliche Ursachen, warum es trotzdem scheitert

#### 1. **\*\*Groß-/Kleinschreibung (Case Sensitivity)\*\***

- Prüfe genau, wie das Resource Label lautet:
  - Dein Matcher fragt nach `@.label == 'CSV'`
  - Wenn das Label z.B. ``csv``, ``Csv`` oder ähnlich geschrieben ist, schlägt der Matcher fehl.

- XNAT ist hierbei case-sensitive!
- Lösung: Stelle sicher, dass das Label exakt **\*\*CSV\*\*** (Großbuchstaben) ist.

## 2. **\*\*Ressourcen-Typ/Kategorie\*\***

- Die Datei **\*\*korrekt\*\*** als Resource hochgeladen? Sie darf **\*\*nicht\*\*** z.B. im Scan-Bereich oder als „Attachment“ landen, sondern nur unter der „CSV“-Resource **\*\*auf Session-Level\*\*** liegen.

- Lösung: Im XNAT UI unter der Session! Resources! CSV prüfen und ggf. neu hochladen.

## 3. **\*\*Dateiendung/Doppelte Dateierweiterung\*\***

- Die Datei muss eine der im Matcher angegebenen Endungen haben:
  - `.csv`, `.tsv`, `.txt`
- Prüfe, ob die Datei ungewollte doppelte Endungen oder Leerzeichen enthält (Bsp.: `correct\_patient.csv.txt`).

## 4. **\*\*Session-Kontext passt zum Wrapper\*\***

- In deinem JSON steht:

```
```json
"contexts": [ "xnat:mrSessionData" ]
```
```

- Deine Session muss eine MR-Session sein (im XNAT-UI: Typ muss MR sein).
- XNAT unterscheidet zwischen MR, CT, PET etc.

## 5. **\*\*Nutzer-Berechtigungen\*\***

- Stelle sicher, dass dein XNAT-Benutzer die Datei wirklich im System sehen/lesen darf.

## 6. **\*\*Wrapper ist wirklich aktiviert\*\***

- Wrapper muss für das **\*\*Projekt\*\*** (tanae\_osa\_Predct) aktiv (enabled) sein!
  - Im UI prüfen unter: Admin! Container Service! Activation

---

## ## Weitere Debug-Möglichkeiten und Tipps

### ### Matcher zum Testen lockern

Ändere testweise im command.json den Matcher auf:

```
```json
"matcher": "true"
```
```

Damit solltest du im UI alle Ressourcen angezeigt bekommen. Wenn es dann funktioniert, weißt du: Der Fehler ist ein zu enger Matcher.

#### ### REST Request Log

Rufe das Command im XNAT-Webinterface auf. Schau dann sofort in die Server-Logs (`cat catalina.out` oder im Web unter „System > Logs“): Gibt es einen Hinweis, WELCHE Ressourcen gefunden wurden oder nicht?

#### ### API-Check

Du kannst mit z.B. curl alle Resources der Session anzeigen lassen:

```
```sh
curl -u <user>:<pass>
"https://xnat-dev.gwdg.de/data/experiments/XNAT_E00428/resources"
```
```

Antwort sollte CSV/anzeigen, sonst stimmt etwas nicht!

#### ### XNAT-UI-Bestätigung

Gehe zu:

- Projekte !' tanae\_osa\_Predct
- Browse !' Wähle XNAT\_E00428
- "Manage Files" !' Siehst du dort unter Resources eine Zeile darunter "correct\_patient.csv"?

---

#### ## Zusammenfassung: Typische Fehlerquellen

- Label mit Groß-/Kleinschreibung stimmt nicht
- Datei-Endung nicht genau `.csv`, `.tsv` oder `.txt`
- Datei liegt nicht direkt unter der Session-Resource „CSV“ (falsche Hierarchie)
- Wrapper nicht enabled oder im falschen Projekt
- Session ist kein MR-SessionType (falscher Kontext)

---



**\*\*Debug-Tipp:\*\***

Wenn alles passt, ändere den Matcher (nur für dich, nicht produktiv!) auf `"matcher": "true"`. Geht es jetzt? Dann prüfe, ob das Label im System exakt „CSV“ heißt (und nicht „csv“), oder setzt den Matcher spezifischer, falls gewünscht.

---

Du kannst auch konkret folgende Informationen (gern anonymisiert) posten:

- Screenshot aus XNAT „Manage Files“ für deine Session
- Den Abschnitt „Resources“ als JSON/REST-Output
- Den genauen Namen und die Endung der Datei

Dann kann ich ganz gezielt sagen, an welcher Stelle der Workflow noch „klemmt“!

---

**\*\*Akademischer Tipp:\*\***

Feingranulare Label-Prüfung und saubere Kontextdefinitionen sind zentrale Bestandteile von wissenschaftlichen Datenworkflows – kleine Inkonsistenzen führen schnell zu nicht startbaren Analysen. Eine einheitliche Datenstruktur ist der Schlüssel zur erfolgreichen Automatisierung.

**Bousfiha, Tanae:**

```
tanae@mi0082n:~/dockerfile/xnat-osa/Ziel1_osaDockerContainer$ curl -u tanae:123
"https://xnat-dev.gwdg.de/data/experimen
ts/XNAT_E00428/resources"
{"ResultSet":{"Result":["file_count":"1",
"xnat_abstractresource_id":"703","cat_desc":"
","cat_id":"XNAT_E00428","format":"&q
uot;,"label":"CSV","category":"resources"
,"element_name":"xnat:resourceCatalog","file_size"
:"99","content":"","tags":""}],
"totalRecords": "1","title": "Resources"}}
```

**HAWKI (gpt-4.1):**

Sehr gut! Deine `curl`-Antwort zeigt eindeutig:

- Es gibt **eine Resource mit Label `CSV`** an deiner Session **XNAT\_E00428**.
- In dieser Resource befindet sich **eine Datei** (file\_count: 1).

Das heißt:

**Der Resource-Matcher in deinem Wrapper sollte grundsätzlich greifen!**

---

#### Was ist jetzt noch zu prüfen, wenn das UI/Launch trotzdem nicht geht?

##### 1. **Dateiname und Endung prüfen**

Du hattest angegeben:

> unter Resources >> CSV >> correct\_patient.csv

Bitte prüfe auf der Kommandozeile (oder notfalls UI), wie die Datei **exakt** heißt:

- Ist der Name **correct\_patient.csv** (genau diese Endung)?
- Liegt die Datei **wirklich direkt** in der Resource "CSV"?
- Keine Leerzeichen/Tippfehler?
- Groß-/Kleinschreibung stimmt?

---

##### 2. **REST-Call für die Dateien testen**

Prüfe nochmal direkt per REST, welche Datei dort steckt:

```
``bash
```

```
curl -u tanae:123
```

```
"https://xnat-dev.gwdg.de/data/experiments/XNAT_E00428/resources/CSV/files"
```

```
``
```

Die Antwort sollte **mindestens den Dateinamen** mit richtiger Endung (`.csv``, `.tsv``, `.txt``) enthalten.

---

##### 3. **Wrapper- und Command-Matcher wie im JSON**

Deine Konfiguration verlangt:

- `@.label == 'CSV'` (also Label \*exakt\* `CSV`)
- Datei: Matcher `"@.name =~ \".\*\\.(csv|tsv|txt)\$\"`

---

#### ##### 4. \*\*Session-Typ\*\*

Gehe sicher, dass **XNAT\_E00428** eine **MR-Session** ist (weil Kontext: `"contexts": [ "xnat:mrSessionData" ]`).

Falls es keine MR-Session ist, den Kontext im Wrapper ggf. auf den passenden Session-Typ ändern!

---

#### ##### 5. \*\*Testweise Matcher lockern\*\*

Um Fehler im Matcher komplett auszuschließen, setze im Wrapper zum Test:

```
``json
"matcher": "true"
``
```

Dann sollte jede Datei gematcht werden.

Wenn das UI/der Job dann startet, liegt es definitiv am Matcher (Case, Regex etc.).

---

#### ##### 6. \*\*Logs im XNAT-UI und Server prüfen\*\*

Schaue parallel nach Fehlerhinweisen im XNAT-Log (oder System-Log-Tab im Webinterface).

---

#### ### Zusammenfassung (was jetzt tun?)

1. \*\*Prüfe mit obigem Curl\*\* die Dateinamen und Endungen.
2. \*\*Passe ggf. den Matcher an\*\* (Regex, Case).
3. \*\*Teste, ob das Lockern des Matchers hilft\*\*.
4. \*\*Session-Typ kontrollieren\*\* (MR-Session?).

> Tipp: Poste das Ergebnis von

> ``curl -u tanae:123``

"https://xnat-dev.gwdg.de/data/experiments/XNAT\_E00428/resources/CSV/files"

falls Unsicherheit besteht, dann kann ich das Matcher-Problem direkt sehen und ggf. den Regex an dein Setup anpassen.

---

**\*\*Akademischer Hintergrund:\*\***

Solche Fehler sind typisch, wenn Automatisierungs-Workflows mit komplexen Matchern und heterogenen Rohdaten konstruiert werden. Advanced Data Management lebt von sauberen Metadaten und klaren Strukturen – eine wichtige Kompetenz im Forschungsdatenmanagement!

Melde dich gern mit dem Dateinamen-Output oder weiteren Fragen – dann finden wir die genaue Stelle, an der es klemmt.

**Bousfiha, Tanae:**

tanae@mi0082n:~/dockerfile/xnat-osa/Ziel1\_osaDockerContainer\$ python3

Containerenabled.py

XNAT WEB URL:https://xnat-dev.gwdg.de

XNAT Username:tanae

XNAT Password:

Project ID:tanae\_osa\_Predct

Session ID:XNAT\_E00428

Path to the Python script:Bonjour.py

Docker Name base image:python:3.10

What is the name of the command in XNAT: examinascan

What is the description of the command in XNAT: examinscan

What is the name of the Label in XNAT: examinscan

What is the description of the Label in XNAT: examine scans in xnat

Dockerfile written to ./Dockerfile

Building Docker image 'examinascan:latest';...

Docker image 'examinascan:latest'; built successfully.

JSON file created at command.json

Uploading command to https://xnat-dev.gwdg.de/xapi/commands

Command created successfully.

Wrapper existiert bereits: 92

Enable-URL: [https://xnat-dev.gwdg.de/xapi/projects/tanae\\_osa\\_Predct/commands/examinascan/wrappers/92/enable](https://xnat-dev.gwdg.de/xapi/projects/tanae_osa_Predct/commands/examinascan/wrappers/92/enable)

Failed to enable wrapper: 404 - <!doctype html><html

lang="en"><head><title>HTTP Status 404 – Not

Found</title><style type="text/css">body

{font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b

{color:white;background-color:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3

{font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-color:

#525D76;border:none;}</style></head><body><h1>HTTP Status

404 – Not Found</h1><hr class="line"

/><p><b>Type</b> Status

Report</p><p><b>Description</b> The origin server did not find a current representation for the target resource or is not willing to disclose that one

exists.</p><hr class="line" /><h3>Apache Tomcat/9.0.43

(Debian)</h3></body></html>

Launching container with payload: {

  "command": 71,

  "wrapper": 92,

  "inputs": {

    "session": "XNAT\_E00428"

  }

}

Failed to launch container: 404 - <!doctype html><html

lang="en"><head><title>HTTP Status 404 – Not

Found</title><style type="text/css">body

{font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b

{color:white;background-color:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3

{font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-color:

#525D76;border:none;}</style></head><body><h1>HTTP Status

404 – Not Found</h1><hr class="line"

/><p><b>Type</b> Status

Report</p><p><b>Description</b> The origin server did not find a current representation for the target resource or is not willing to disclose that one

exists.</p><hr class="line" /><h3>Apache Tomcat/9.0.43

(Debian)</h3></body></html>

**HAWKI** (gpt-4.1):

Du hast wieder das Problem, das viele in der XNAT-Automatisierung haben:  
\*\*Du nutzt für `enable` und den Container-Start die numerische Wrapper- und Command-ID (`92`, `71`) statt den NAMEN!\*\*  
Das führt zu \*\*404-Fehlern\*\* bei fast allen aktuellen XNAT-Container-Services.

**Warum?**

Die API erwartet in der Enable-URL und im Launch-Payload den **Namen** (String), nicht die interne Datenbank-ID (Zahl).

---

**So behebst du das Problem Schritt für Schritt:**

**1. Verwende Wrapper- und Command-Name, NICHT die ID**

- Zum Aktivieren (`enable`) verwendest du:

```
...
/xapi/projects/tanae_osa_Predct/commands/examinascan/wrappers/examinascan_w
rapper/enable
...
(nicht .../wrappers/92/enable)
```

- Für den Launch/Container-Start muss das Payload so aussehen:

```
```json  
{  
  "command": "examinascan",  
  "wrapper": "examinascan"
```

Bousfiha, Tanae:

was war schwer um den wrapper zu fixieren

HAWKI (gpt-4.1):

Reflexionsfrage:

Was war schwer am Fixieren des Wrappers?

Das ist eine kluge Frage für die Entwicklungspraxis und für das Lernen von technischen Schnittstellen. Aus wissenschaftlicher Sicht lässt sich das so beantworten (und hilft dir oder deinem Team bei der Prozess-Reflexion, was du beim nächsten Automatisierungs / Container-Projekt beachten solltest)

1. ****Fehlende oder intransparente Dokumentation der API****

Die offizielle XNAT Container Service API dokumentiert nicht immer eindeutig, wann ****numerische IDs**** und wann ****Wrapper-Namen**** ("name" statt "id") einzusetzen sind. Viele Plattformen liefern beides (ID und Name), aber REST-Endpunkte akzeptieren häufig nur einen davon.

! Das Fehlen eines konsistenten Benutzungsschemas erschwert führt zu Frustration.

2. ****Inkonsistenz zwischen der XNAT-Web-GUI und der API****

In der Weboberfläche werden Namen angezeigt. Die API-JSONs enthalten aber sowohl IDs als auch Namen — je nach Kontext wertet XNAT aber nur einen aus. Der Unterschied fällt beim Testen oft nicht direkt auf.

3. ****Automatische Wrapper-Verwaltung ist nicht robust****

Viele Skripte (siehe deines) erzeugen automatisch Wrapper, ohne vorher gründlich zu prüfen, ob exakt einer zum Command existiert oder bereits „enabled“ ist, oder ob eine Kollision besteht.

Das erschwert insbesondere das Wiederverwenden und Debuggen bereits bestehender Wrapper bei mehrfachen Testdurchläufen.

4. ****REST-Fehler 404 sind in XNAT wenig spezifisch****

Ein HTTP 404 „not found“ kann alles bedeuten:

Das Objekt gibt es nicht,

es gibt kein Mapping mehr,

Projekt-Kontext passt nicht,

****oder****: *falscher Typ (Name statt ID oder umgekehrt).*

Die eigentliche Fehlerursache bleibt versteckt.

5. **Komplexe Wrapper-Metadaten (Kontext, Matcher, Ressourcen)**

Insbesondere, wenn Ressourcen, Labels, File-Endungen oder Kontextdefinitionen (session, scan etc.) nicht exakt zu den aktuellen XNAT-Daten passen, gibt es keine hilfreiche Fehlermeldung, sondern lediglich ein unerklärliches Abbrechen oder einen UI-Fehler.

Was war also schwer am Fixieren des Wrappers?

- **Richtig zu erkennen, dass die API in URLs und Payloads den Namen erwartet und nicht die ID.**
- **In JSONs und Aufrufen immer genau nachzuvollziehen, welche Werte in welchem Feld gebraucht werden (name vs. id, case sensitivity!).**
- **Sicherzustellen, dass der Wrapper nicht nur existiert, sondern auch korrekt für das Projekt aktiviert und mit passenden Ressourcen gematcht werden kann.**
- **Den Überblick über bereits existierende Wrapper zu behalten und konsequent mit dem passenden weiterzuarbeiten (statt Chaos durch viele doppelte Wrapper zu erzeugen).**
- **Fehlermeldungen richtig zu deuten und systematisch nach logischen Fehlern in der Chain (Command ! Wrapper ! Activation ! Launch ! Input-Feld) suchen.**

Fazit/Leitfrage für die Reflexion:

Das Schwierigste am Fixieren des Wrappers war die korrekte Unterscheidung und Verwendung von Namen und IDs — sowohl in URLs als auch in Payloads — sowie die Interpretation unspezifischer Fehlermeldungen und das Nachvollziehen der komplexen Zusammenhänge zwischen Ressourcen, Metadaten und Projekt-Kontexten.

****Tipp:**** Für nachhaltige Forschung ist es essenziell, Workflows und Fehlerquellen zu dokumentieren – z.B. indem du deine Lessons Learned und Stolperfallen notierst und in eine Team-Doku oder ein Forschungsdatenmanagement-System einpflegst. Das hilft mittel- bis langfristig auch Kolleg*innen enorm!