

Dstar Tool

The `dstar_tool` helps you discover available Datastar utilities and generate ready-to-run Python scripts for them. Utilities are pre-built jobs that run on the Optilogic platform — covering categories such as standard utilities (Python-based), AI agents, and machine learning models. Rather than manually constructing task configuration, you point `dstar_tool` at a utility by name and it generates a script with all parameters stubbed out and ready to fill in.

Docs and Examples

After installing the package, pull the bundled PDF guides and example scripts onto disk. Neither command takes any options, and both fail with an error if the target folder already exists.

Create a local `docs` folder containing the PDF guides (`datastar.pdf` and `dstar_tool.pdf`) and the `Changelog.txt`:

```
dstar_tool docs
```

Create a local `examples` folder containing the bundled example scripts:

```
dstar_tool examples
```

CLI Usage

The `list`, `file`, `raw`, and `create` commands read an `app.key` file from the current working directory. All of these commands require this file to be present. `docs` and `examples` do not require an `app.key`.

List available utilities

List all utilities:

```
dstar_tool list
```

Filter by jobType (values match the utilities API exactly):

```
dstar_tool list utility-python  
dstar_tool list ai-agent  
dstar_tool list machine-learning
```

Generate a task script

Pass the utility name exactly as shown in the list output:

```
dstar_tool create "Convert Currency"  
dstar_tool create "Modeler Agent"  
dstar_tool create "Geographic Clustering"
```

This writes a `.py` file to the current directory named after the utility. The file contains a complete script with the correct task class (`RunUtilityTask`, `RunAIAgentTask`, or `RunMachineLearningTask`) and all parameters stubbed out. Required parameters are marked with a `# required` comment.

Dump raw utilities JSON

Write the full utilities API response to a file for inspection:

```
dstar_tool file utilities.json
```

Programmatic Usage

The same functionality is available in code via `UtilityTool`.

List utilities

```
from datastar.dstar_tool import UtilityTool  
  
tool = UtilityTool(app_key="op_xxxx...")  
  
# All utilities as a formatted string  
print(tool.list())  
  
# Filtered by jobType  
print(tool.list("ai-agent"))  
print(tool.list("machine-learning"))  
print(tool.list("utility-python"))  
  
# As a list of dicts for programmatic inspection  
utilities = tool.get_utilities()  
ml_utilities = tool.get_utilities("machine-learning")
```

Generate a task script as a string

`create` returns the generated script as a string rather than writing a file, so you can use it however you need — print it, write it to a custom path, execute it dynamically, or embed it in a UI.

```

from datastar.dstar_tool import UtilityTool

tool = UtilityTool(app_key="op_xxxx...")
code = tool.create("Geographic Clustering")
print(code)

```

Example output for an ML utility:

```

# Utility: Geographic Clustering (v1.2.0)
# Clusters facilities or customers by geographic proximity
from datastar import Project, RunMachineLearningTask

project = Project.create("MY PROJECT")
macro = project.add_macro(name="My Macro")

properties = {
    "database-name": "<REQUIRED: Database Name>", # required; Database
Name
    "input-table-name": "<REQUIRED: Input Table>", # required; Input Table
    "output-table-name": "<REQUIRED: Output Table>", # required; Output
Table
    "clustering-model": "kmeans", # Clustering Model
    "n-clusters": 10, # Number of Clusters
    "distance-unit": "mi", # Distance Unit
}

command_args =
RunMachineLearningTask.build_command_args_from_properties(properties)

run_utility_task = macro.add_run_machine_learning_task(
    name="Run Geographic Clustering",
    description=" ",
    utility_name="Geographic Clustering",
    utility_configuration={"properties": properties},
    run_configuration={"commandArgs": command_args},
    auto_join=False,
)

```