

# API Reference

## Table of contents

<b>1</b>	<b>Core API Functions</b>	<b>1</b>
1.1	create_llm . . . . .	1
1.2	create_default_tools . . . . .	2
1.3	build_agent . . . . .	3
<b>2</b>	<b>Interacting with the Agent</b>	<b>3</b>
<b>3</b>	<b>File Operation Helpers</b>	<b>3</b>
3.1	write_file . . . . .	4
3.2	create_from_template . . . . .	4
3.3	py_to_ipynb . . . . .	4
3.4	create_project_scaffold . . . . .	4

This page provides a reference for the public API of the `code_agent` package.

## 1 Core API Functions

The `code_agent` package exposes several key functions for building and interacting with the agent.

### 1.1 create\_llm

This function initializes a Language Model (LLM) instance based on your configuration.

```

from code_agent import create_llm
from code_agent.main import load_config

config = load_config()
llm = create_llm(config)

```

#### Arguments:

- `cfg` (dict): A dictionary containing LLM configuration options (e.g., `ollama_model`, `ollama_host`, `ollama_port`). This is typically loaded from `llm_config.json`.

#### Returns:

- An instance of `langchain_core.language_models.BaseChatModel` (e.g., `ChatOllama`).

## 1.2 create\_default\_tools

This function creates a list of standard tools that the agent can use.

```

from code_agent.agents.base_agent import create_default_tools
from code_agent import create_llm
from code_agent.main import load_config

config = load_config()
llm = create_llm(config)
tools = create_default_tools(root_dir=".", llm=llm)

```

#### Arguments:

- `root_dir` (str, optional): The root directory for file operations. Defaults to the current working directory.
- `llm` (`BaseChatModel`, optional): An LLM instance, required for tools that involve language model interaction (e.g., `SearchExplainTool`).

#### Returns:

- A list of `langchain_core.tools.BaseTool` instances.

### 1.3 build\_agent

This is the primary function to construct the LangGraph-based agent. It binds the provided tools to the LLM, creating a `Runnable` that represents the agent's core logic.

```
from code_agent import build_agent, create_llm
from code_agent.agents.base_agent import create_default_tools
from code_agent.main import load_config

config = load_config()
llm = create_llm(config)
tools = create_default_tools(root_dir=".", llm=llm)
agent_runnable = build_agent(llm=llm, tools=tools)
```

#### Arguments:

- `llm` (`BaseChatModel`): The initialized LLM instance.
- `tools` (`Iterable[BaseTool]`): A list of tools for the agent to use.

#### Returns:

- A `langchain_core.runnables.Runnable` instance, which is the compiled LangGraph agent.

## 2 Interacting with the Agent

Once you have `agent_runnable`, you can interact with it using its `invoke` or `stream` methods.

```
from code_agent.main import Runnable, HumanMessage
# Assuming agent_runnable is already created as shown above
response = agent_runnable.invoke({"messages": [HumanMessage(content="Hello, agent!")]})

# For streaming intermediate steps (as used in the CLI)
# for event in agent_runnable.stream({"messages": [HumanMessage(content="Improve file.py")]})
#     # Process events
#     pass
```

## 3 File Operation Helpers

The `code_agent` package also provides direct utility functions for common file operations:

### 3.1 write\_file

Writes content to a specified file.

```
from code_agent import write_file
from pathlib import Path

file_path = Path("my_new_file.txt")
write_file(file_path, "This is some content.")
```

### 3.2 create\_from\_template

Creates a file from a template, with optional variable replacement.

```
from code_agent import create_from_template
from pathlib import Path

template_path = Path("template.txt")
template_path.write_text("Hello, {name}!")
output_path = Path("output.txt")
create_from_template(template_path, output_path, replace_vars={"name": "World"})
```

### 3.3 py\_to\_ipynb

Converts a Python script to a Jupyter notebook.

```
from code_agent import py_to_ipynb
from pathlib import Path

script_path = Path("my_script.py")
script_path.write_text("print('Hello from Python!')")
notebook_path = Path("my_notebook.ipynb")
py_to_ipynb(script_path, notebook_path)
```

### 3.4 create\_project\_scaffold

Generates a new project scaffold.

```
from code_agent import create_project_scaffold
from pathlib import Path

project_root = Path("./my_new_project")
create_project_scaffold(str(project_root), project_name="my_new_project")
```