API Reference

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

Core API Functions

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

`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`).

`create_default_tools`

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

```python
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.

`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.

```python
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.

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
```

File Operation Helpers

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

`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.")
```

`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"})
```

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)
```

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")
```