File Breakdown: `src/utils/csv_function_mapper.py`

File Location

`src/utils/csv_function_mapper.py`

Overview

The `csv_function_mapper.py` file implements the `FunctionMapLoader` class, which is responsible for loading function definitions from CSV files and mapping them to actual Python functions. This utility enables a flexible, configuration-driven approach to function registration and discovery, allowing the system to define function mappings in CSV files rather than hardcoding them.

Key Responsibilities

- Load function definitions from CSV files
- Register Python functions for availability in the system
- Map function names from CSVs to actual callable functions
- Provide fallbacks when CSV loading fails
- Support enhanced function maps with additional metadata
- Handle path resolution for finding CSV files

Core Functionality

Class Definition

```python
class FunctionMapLoader:
    """
    Loads function maps from CSV files and maps them to actual Python functions.
    This class bridges the CSV definitions with callable Python functions.
    """

    def __init__(self, base_path=None):
        """
        Initialize the function map loader.

        Parameters:
            base_path (str, optional): Base directory where function map CSVs are
stored.
                If None, will use default path based on module location.
        """
        if base_path is None:
            # Default path is in the utils/function maps directory
            self.base_path = os.path.join(
                os.path.dirname(os.path.abspath(__file__)),
                "function maps"
            )
        else:
            self.base_path = base_path

        # Create the directory if it doesn't exist
        os.makedirs(self.base_path, exist_ok=True)

        self.function_registry = {}
```

Function Registration

Methods to register functions with the system:

```
@log_function_call
def register_function(self, key: str, function: Callable):
    """
    Register a Python function to be available for mapping.

    Parameters:
        key (str): The key or name of the function
        function (Callable): The actual Python function to be called
    """
    self.function_registry[key] = function

@log_function_call
def register_functions(self, functions_dict: Dict[str, Callable]):
    """
    Register multiple functions at once.

    Parameters:
        functions_dict (Dict[str, Callable]): Dictionary mapping function keys to
actual functions
    """
    self.function_registry.update(functions_dict)
```

Function Map Loading

The main method for loading function maps from CSV files:

```
@log_function_call
def load_function_map(self, agent_name: str, enhanced=True) -> Dict[str, Callable]:
    """
    Load a function map CSV for a specific agent and map it to actual Python
functions.

    Parameters:
        agent_name (str): Name of the agent (e.g., 'Nova', 'Emil')
        enhanced (bool, optional): Whether to use the enhanced CSV. Defaults to True.

    Returns:
        Dict[str, Callable]: Dictionary mapping function keys to actual Python
functions
    """
    # Determine the CSV file path
    filename = f"{agent_name}_function_map{'_enhanced' if enhanced else ''}.csv"
    csv_path = os.path.join(self.base_path, filename)

    # Check if file exists
    if not os.path.exists(csv_path):
        print(f"Warning: Function map CSV not found: {csv_path}")
        # Try finding it in the agent directory
        agent_dir_path = os.path.join(
            os.path.dirname(os.path.dirname(os.path.abspath(__file__))),
            "agents",
            filename
        )
        if os.path.exists(agent_dir_path):
            csv_path = agent_dir_path
            print(f"Found function map in agents directory: {csv_path}")
        else:
            print(f"No function map found for agent {agent_name}")
            return {}

    try:
        # Load the CSV into a DataFrame
        df = pd.read_csv(csv_path)

        # Create a function map
        function_map = {}

        # Process each row in the CSV
        for _, row in df.iterrows():
            key = row['Key']
            function_name = row['Function'] if 'Function' in row else key

            # Check if this function is registered
            if function_name in self.function_registry:
                function_map[key] = self.function_registry[function_name]
            else:
                print(f"Warning: Function '{function_name}' is not registered for key
'{key}'")

        print(f"Successfully loaded {len(function_map)} functions for agent
{agent_name}")
        return function_map
    except Exception as e:
        print(f"Error loading function map for {agent_name}: {str(e)}")
        return {}
```

Key Features

1. **CSV-Based Configuration**: Enables function definitions to be managed in CSV files
2. **Function Registration**: Provides a clear API for registering callable functions
3. **Dynamic Mapping**: Maps function names from CSVs to actual Python functions
4. **Path Resolution**: Searches multiple locations for CSV files
5. **Error Handling**: Gracefully handles missing files and registration errors

6. **Logging**: Uses decorators for function call logging
7. **Enhanced Mode**: Supports enhanced CSVs with additional metadata

Integration

- Used in `main.py` to load function maps for all agents
- Provides function mappings for Nova, Emil, Ivan, and Lola
- Works with the function registry in `functions_registery.py`
- Enables configuration-driven function discovery

Workflow

1. Initialize the function map loader
2. Register all available functions with the loader
3. For each agent, load its function map from CSV
4. Map function keys from the CSV to registered functions
5. Return the completed function map for agent initialization
6. Fallback to hardcoded mappings if CSV loading fails

Implementation Notes

- Uses pandas for CSV parsing
- Searches multiple directories to find CSV files
- Logs warnings for unregistered functions
- Handles both standard and enhanced CSV formats
- Creates the function maps directory if it doesn't exist
- Provides detailed error information when loading fails