CARLA 0.9.15 Client and Agent Modules: Structured Summary and Practical Usage with ROS 2 Humble Bridge

Based on Official CARLA 0.9.15 Documentation
(`core_world/#the-client` and `adv_agents`)
IRLab 2025, IES Uni Kassel
Sören, Nikita, Iman

1. Introduction

In the CARLA 0.9.15 simulator, the **Client** and the **World** constitute two fundamental abstractions required to operate the simulation and its actors. CARLA 0.9.15 is implemented on **Unreal Engine 4.26**, which provides the rendering and physics backbone for all simulation features. The Client is one of the core architectural elements of CARLA. It establishes a connection with the server, retrieves information about the current world state, and issues commands to modify it through user-defined scripts. After identifying itself to the server, the Client accesses the World instance, which represents the active simulation environment and provides interfaces for controlling actors, weather, lighting, and other parameters.

Beyond these basic operations, the Client also provides access to advanced CARLA modules and functionalities, including the Traffic Manager, the Recorder, and command batching utilities. Command batches, in particular, are designed to apply multiple actions—such as spawning or destroying large numbers of actors—in a single synchronized step. More complex modules are introduced in specialized sections of the CARLA documentation.

CARLA's **Agent** scripts extend this functionality by enabling autonomous vehicle behavior within the simulated environment. An Agent allows a vehicle either to follow a random, continuous route or to navigate to a specific destination using route planning and control mechanisms. Agents obey traffic lights and react to dynamic obstacles such as vehicles and pedestrians. Three primary agent types are available, and their operational parameters—including target speed, braking distance, and tailgating behavior—can be adjusted. Furthermore, actor classes can be subclassed or modified to define custom agent logic, enabling the creation of user-specific driving behaviors for advanced testing and simulation scenarios.

2. Client: Interface for World Control

The `carla.Client` class is the communication gateway between user scripts and the CARLA server. It manages the connection, world instances, and global simulation parameters.

2.1. Connection and Initialization

A client connects through TCP ports to the CARLA server. The primary port is `2000`, and a secondary port `2001` is used automatically. An optional third argument sets the number of worker threads (default 0 = use all). Timeouts prevent indefinite blocking.

```
import carla
client = carla.Client('localhost', 2000)
client.set_timeout(30.0)
world = client.get_world()
```

Client and server module versions must match; this can be verified via `get_client_version()` and `get_server_version()`.

2.2. World Connection and Management

The world can be loaded or reloaded dynamically:

```
print(client.get_available_maps())
world = client.load_world('Town05')
# client.reload_world() reloads the same map
```

Each reload destroys the previous simulation instance and spawns a new episode ID, while the Unreal Engine process remains active.

2.3. Command Batching and Utilities

Batching reduces communication latency by grouping actor commands:

```
client.apply_batch([carla.command.DestroyActor(x) for x in vehicles])
```

The Client also provides:

- **Traffic Manager** — controls all vehicles set to autopilot to create realistic traffic.
- **Recorder** — saves and replays entire simulations using frame-level snapshots.

2.4. World Features and Settings

- **Actors and Blueprints:** spawn vehicles, pedestrians, and sensors from the blueprint library.
- **Weather:** configured through `carla.WeatherParameters`, e.g. cloudiness, precipitation, sun angle.
- **Lights:** managed by `carla.LightManager` and categorized by `LightGroup`.
- **Debug Helper:** draws primitives (boxes, lines, arrows) for debugging.
- **Snapshots:** capture per-frame states of all actors.
- **WorldSettings:** define fixed delta time, synchronous mode, and rendering options.

```
settings = world.get_settings()
settings.synchronous_mode = True
settings.fixed_delta_seconds = 0.05
world.apply_settings(settings)
```

Note: In CARLA 0.9.15, weather parameters modify only the rendered appearance of the scene (e.g., clouds, rain, lighting) and have no influence on the underlying vehicle physics or surface friction.

3. Agents: Autonomous Vehicle Control

Agents in CARLA 0.9.15 are Python classes that implement route planning, control, and behaviour logic for vehicles.

3.1. Architecture Overview

- **Planning and Control Modules:**
  - `controller.py` — combined longitudinal/lateral PID controller.
  - `global_route_planner.py` — builds a waypoint graph of the map topology.
  - `local_planner.py` — generates short-range waypoints and control targets.
- **Agent Behaviours:**
  - `basic_agent.py` — implements the *BasicAgent* that follows routes, avoids collisions, obeys lights, but ignores stop signs.
  - `behavior_agent.py` — defines the *BehaviorAgent* that follows speed limits, observes traffic rules, and adjusts aggression through behaviour profiles (*cautious*, *normal*, *aggressive*).

3.2. Behaviour Parameters (0.9.15)

Defined in `behavior_types.py`: `max_speed`, `speed_lim_dist`, `speed_decrease`, `safety_time`, `min_proximity_threshold`, `braking_distance`, and `tailgate_counter`. These tune aggressiveness, following distance, and braking reactions.

3.3. Implementation Example

```python
from agents.navigation.behavior_agent import BehaviorAgent
agent = BehaviorAgent(vehicle, behavior='normal')
agent.set_destination(random.choice(spawn_points).location)

while True:
    vehicle.apply_control(agent.run_step())
    if agent.done():
        agent.set_destination(random.choice(spawn_points).location)
```

3.4. Custom Agent Definition

Every custom agent must implement initialization and `run_step()`.

```python
import carla
from agents.navigation.basic_agent import BasicAgent

class CustomAgent(BasicAgent):
    def __init__(self, vehicle, target_speed=20, debug=False):
        super().__init__(vehicle, target_speed, debug)

    def run_step(self, debug=False):
        control = carla.VehicleControl()
        # Custom logic
        return control
```

Example scripts in `PythonAPI/examples/automatic_control.py` demonstrate both agent types.

4. Integration with ROS 2 Humble

For CARLA 0.9.15, the `carla_ros_bridge` provides native ROS 2 Humble compatibility. The bridge converts CARLA sensor streams into ROS 2 messages and synchronizes them with simulation ticks.

4.1. Typical Workflow

1. Launch the CARLA server: `$ ./CarlaUE4.sh -quality-level=Low`
2. Start the ROS 2 bridge: `$ ros2 launch carla_ros_bridge carla_ros_bridge.launch.py`
3. Connect the Python client and enable synchronous mode.
4. Spawn the ego vehicle and attach sensors via blueprints.
5. Run an `Agent` to control movement.
6. Record topics using `$ ros2 bag record /carla/ego_vehicle/* /carla/imu /carla/lidar`

4.2. Scenario Dimensions

- **Environment:** adjust `WeatherParameters` (cloudiness, rain, fog, sun altitude).
- **Lighting:** use `LightManager` and `LightGroup` to alter illumination at night.
- **Map and Routes:** rotate among `Town01-Town10`, define start/goal points.
- **Traffic:** spawn varying numbers of NPC vehicles and pedestrians under the Traffic Manager.
- **Ego Behaviour:** switch among agent profiles (*cautious*, *normal*, *aggressive*).
- **Sensor Rig:** modify sensor pose, field of view, resolution, and tick rate.
- **Events:** spawn or remove actors dynamically to create occlusion or near-miss events.

4.3. Scenario Examples

1. **Weather Sweep:** vary sun altitude, rain, and cloudiness at fixed routes.
2. **Traffic Ladder:** increase NPC density for congestion scenarios.
3. **Behaviour Variation:** repeat identical route with different agent profiles.
4. **Night Mode:** set `sun_altitude_angle < 0`, enable street and building lights.
5. **Sensor Layout Change:** adjust camera height, LiDAR offset, and frame rate.

4.4. Checklist for High-Quality Datasets

- Run in **synchronous mode** with fixed delta.
- Store per-episode metadata (map, weather, agent profile).
- Vary weather, traffic, and agent behaviour.
- Re-spawn world to avoid residual states.
- Record via `ros2 bag record` to synchronized `.db3`.
- Use consistent sensor calibration and time stamps.

Note: The parameter `fixed_delta_seconds` defines the simulation time step (Δt) between consecutive frames. Using a fixed Δt ensures that each physics update and sensor measurement occurs at uniform temporal intervals, enabling reproducible, time-synchronized datasets across all sensor modalities.

5. Version Considerations

This document targets CARLA **0.9.15** and ROS 2 **Humble**. Later CARLA versions (e.g. 0.10) migrate to Unreal Engine 5 and restructure lighting APIs. For 0.9.15, `LightManager`, `TrafficManager`, and the ROS 2 bridge described here should be fully supported and stable.

6. Conclusion

This report provides a concise technical summary of the Client and Agent modules in CARLA 0.9.15, derived directly from the official documentation. The Client–World interface defines how external scripts connect to and control the simulator, while the Agent framework provides autonomous driving behaviours based on route planning and control logic. Built on Unreal Engine 4.26, CARLA 0.9.15 enables synchronous simulation stepping to maintain temporal alignment of sensor data and control actions. In combination with the ROS 2 Humble bridge, these components support synchronized sensor streaming, automated scenario execution, and consistent data recording under varied environmental and behavioural conditions. The presented workflow and scenario design guidelines form a practical foundation for creating realistic and repeatable simulation datasets for autonomous-driving research.