대한기계학회 IT지능융합부문 2025년 동계 옴니버스 강습회

NVIDA Omniverse 입문 및 Simulation 기초 실습

신 동 훈

Prof. and Director

Research in Intelligent Mobility Systems Lab.

Korea Maritime & Ocean University

© copyrights KMOU RiMS

2025. 02. 28

(Proudly Assisted by 윤택한, 우태걸)

사단법인 대한기계학회 KSME The Korean Society of Mechanical Engineers

RiMS

Contents

Contents

NVIDIA Omniverse 내 기능을 활용한 데이터 추출/적용 방법 소개

- **Isaac sim Physics Vehicle을 활용한 차량 데이터 추출 및 데이터 적용 방법**

*** 주행데이터 csv 파일 추출 로직**

*** 추출된 csv 파일**

*** 추출된 csv 파일을 이용한 주행 시뮬레이션 적용 로직**

- Omniverse 시뮬레이션 **차량 주행 데이터 추출을 위한 로직 구현**
  - 실제 주행 시뮬레이션 내에서 **차량의 속도, 가속도, 각속도** 등의 **파라미터 값**들을 실시간으로 **확인 및 추출**할 수 있는 **기반 구축**
  - 시뮬레이션 차량과 Omniverse 시뮬레이션 內 기능인 **Action Graph**와 **Python Scripts**의 **상호작용**을 통해 **주행 데이터 추출**

NVIDIA Omniverse 내 기능을 활용한 데이터 추출/적용 방법 소개

- **Isaac sim Physics Vehicle을 활용한 차량 데이터 추출 및 데이터 적용 방법**

*** 주행데이터 csv 파일 추출 로직**

*** 추출된 csv 파일**

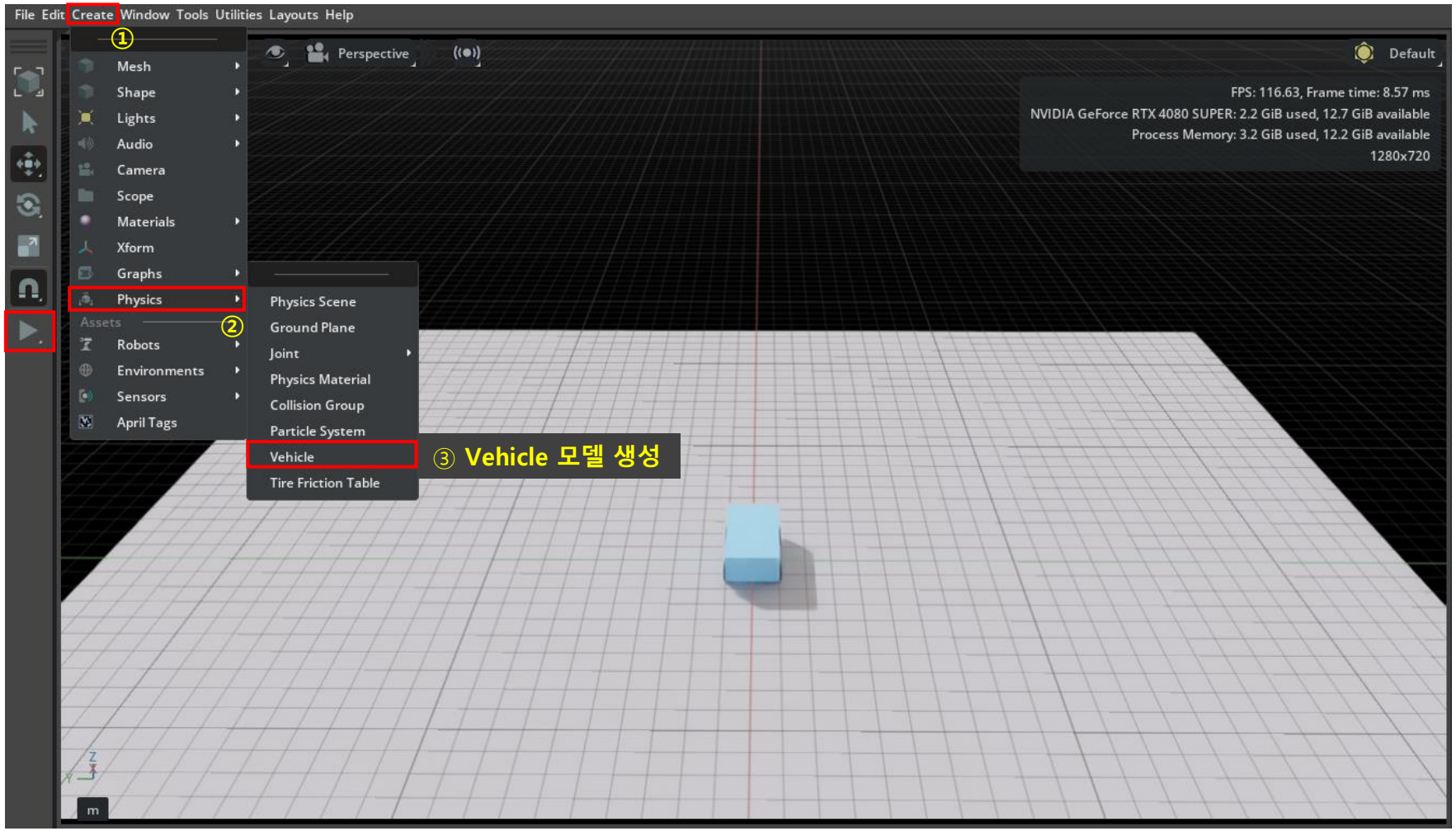
*** 추출된 csv 파일을 이용한 주행 시뮬레이션 적용 로직**

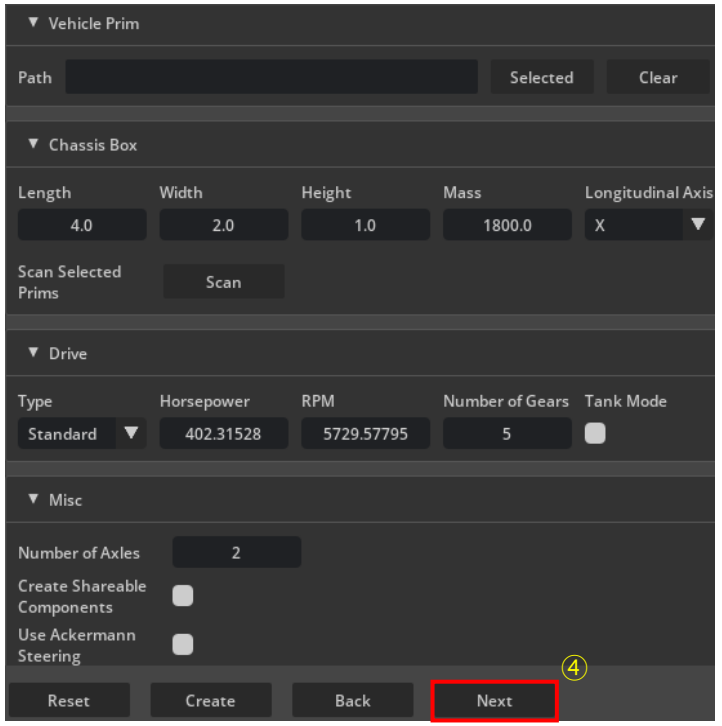
- Omniverse 시뮬레이션 **차량 주행 데이터 적용을 위한 로직 구현**
  - 실제 주행 시뮬레이션 내에서 **차량의 속도, 가속도, 각속도** 등의 **파라미터 값**들을 실시간으로 **적용**할 수 있는 **기반 구축**
  - 시뮬레이션 차량에 추출된 **데이터를 Action Graph**와 **Python Scripts**의 **상호작용을 통해 가상 주행 시뮬레이션 실행**

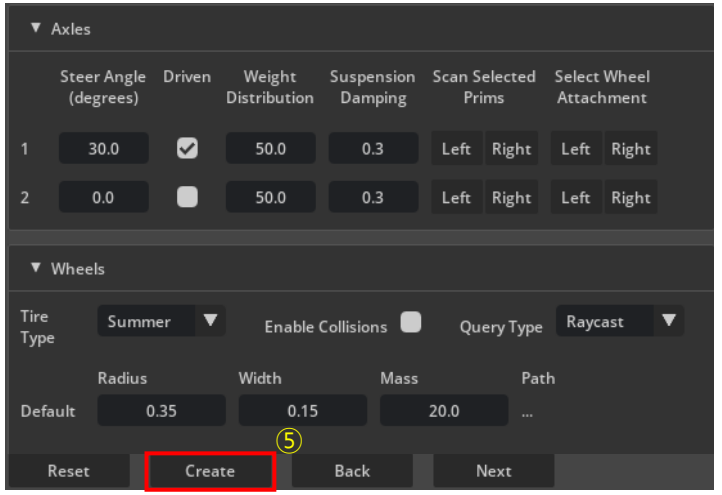
Contents

Action Graph 기반 데이터 추출 로직 구현 및 실습

- **Vehicle 모델 생성**

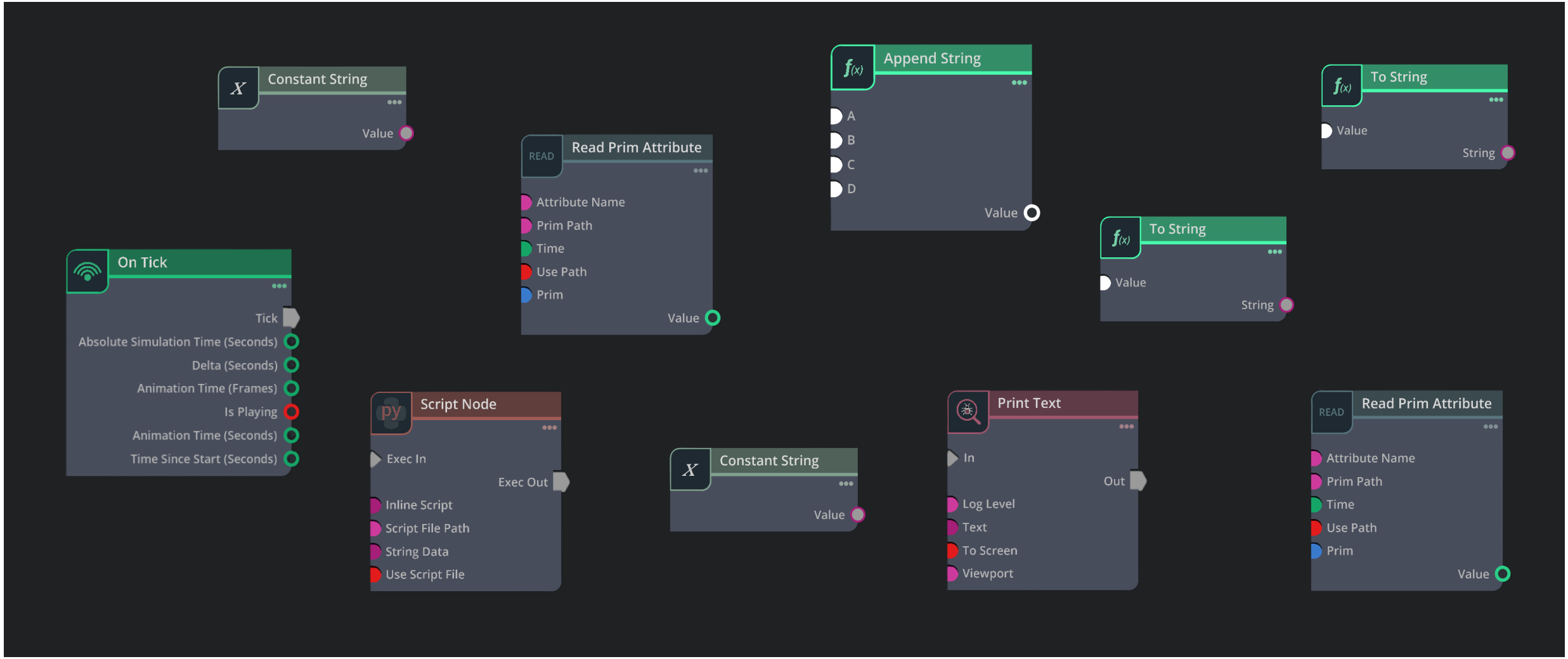






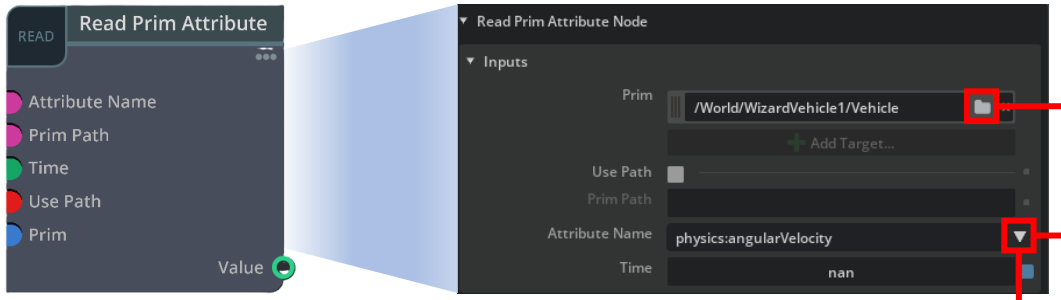
Action Graph 기반 데이터 추출 로직 구현 및 실습: Quiz

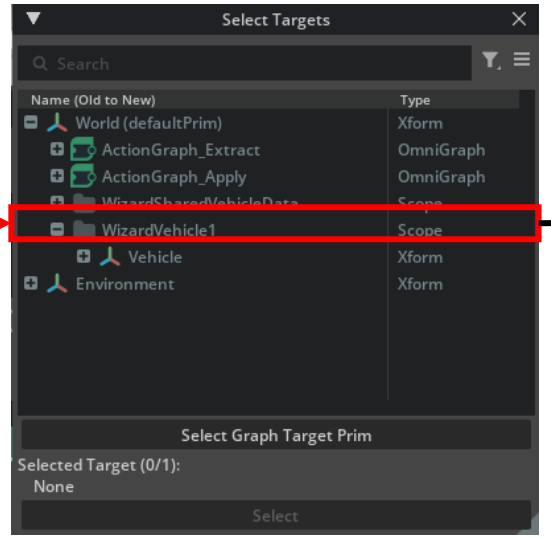
- Vehicle Velocity 데이터 추출을 위한 Action Graph



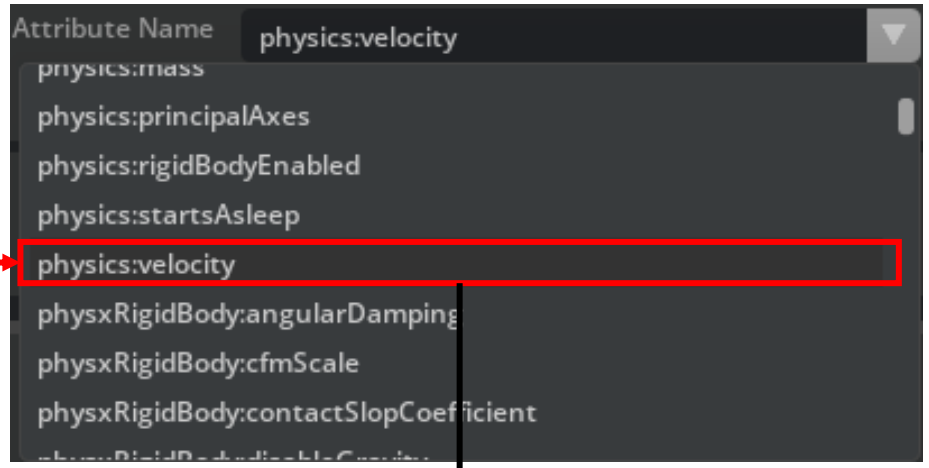
Action Graph 기반 데이터 추출 로직 구현 및 실습: 데이터 추출 노드 [1/3]

- **Vehicle Prim의 Linear/Angular Velocity 데이터 추출 노드**

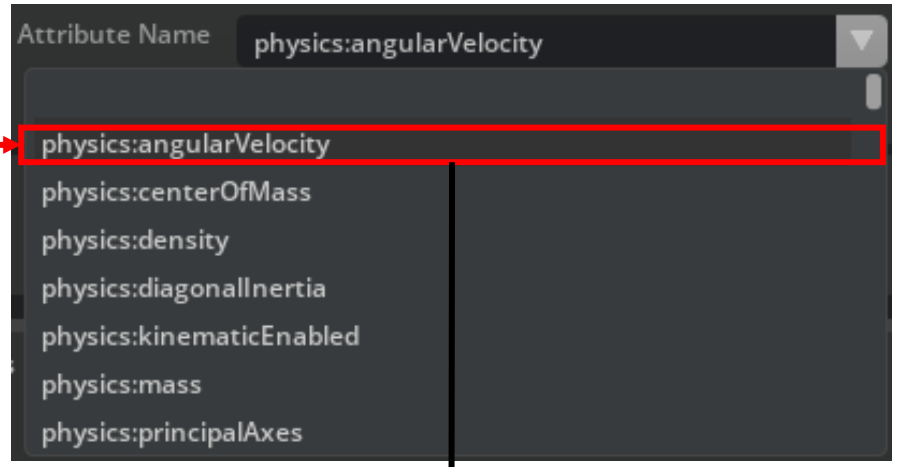




속성을 불러올 Prim 객체 설정
(WizardVehicle1 → Vehicle)



객체의 불러올 속성 설정
(physics:Velocity)



객체의 불러올 속성 설정
(physics:angularVelocity)

Action Graph 기반 데이터 추출 로직 구현 및 실습: 데이터 추출 노드 [2/3]

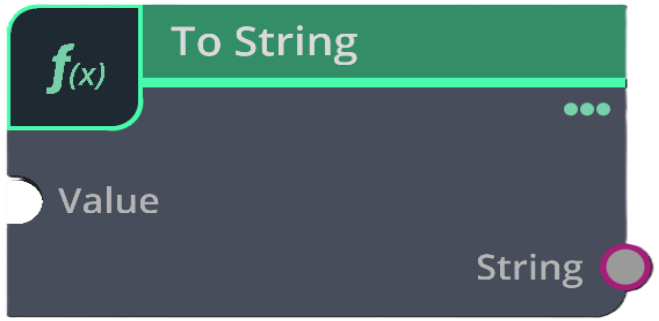
- 추출된 Linear/Angular Velocity 데이터의 String 타입 변환 및 결합 노드

Angular Velocity(ω_x, ω_y, ω_z)

Data Type: GfVec3f
Unit: deg/s

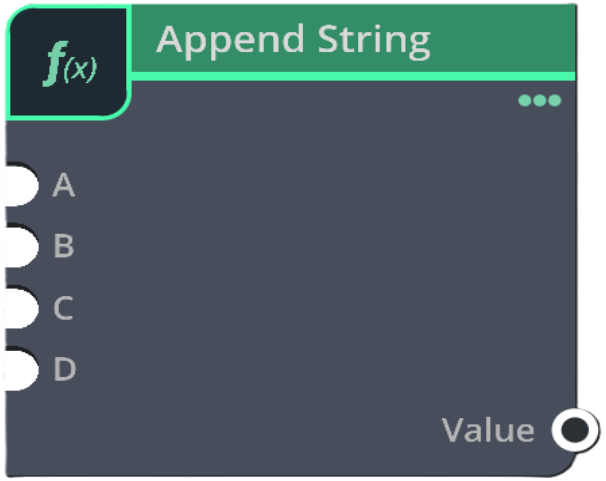
Linear Velocity(v_x, v_y, v_z)

Data Type: GfVec3f
Unit: m/s



<추출된 데이터를 String 타입으로
강제 형변환 시키는 노드 >

<시뮬레이션 화면 내 Default로
출력되는 String 설정 노드>



<여러 String 타입의 데이터를
하나로 결합 시켜주는 노드>

실시간 추출된 차량 데이터 (Data Type: String)

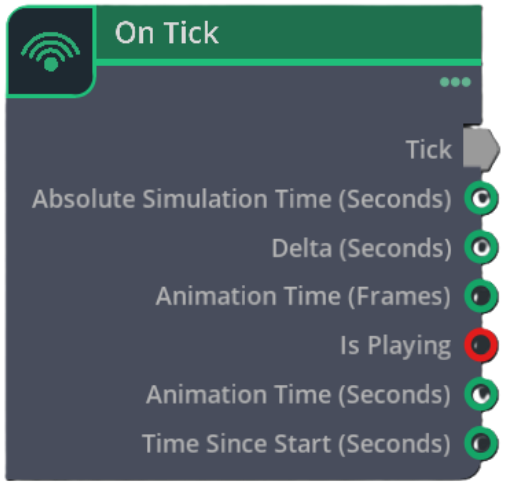
Angular Velocity: $[\omega_x, \omega_y, \omega_z]$ Linear Velocity: $[v_x, v_y, v_z]$

```
Angular Velocity : [0.0157618, 0.741693, 0.00769744]  Linear Velocity : [2.4416, -0.00873407, -0.0118046]
```

추후, 시뮬레이션 실행결과 위 이미지와 같이 화면 상단에
String 형태로 결합된 주행 데이터가 실시간 출력되면 성공!

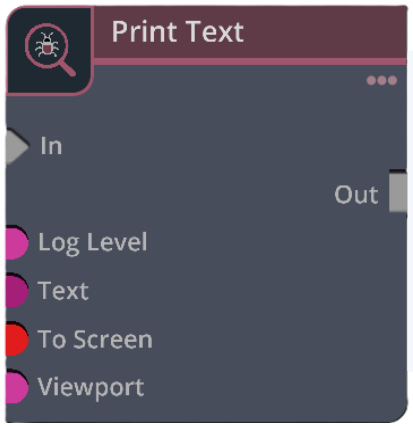
Action Graph 기반 데이터 추출 로직 구현 및 실습: 데이터 추출 노드 [3/3]

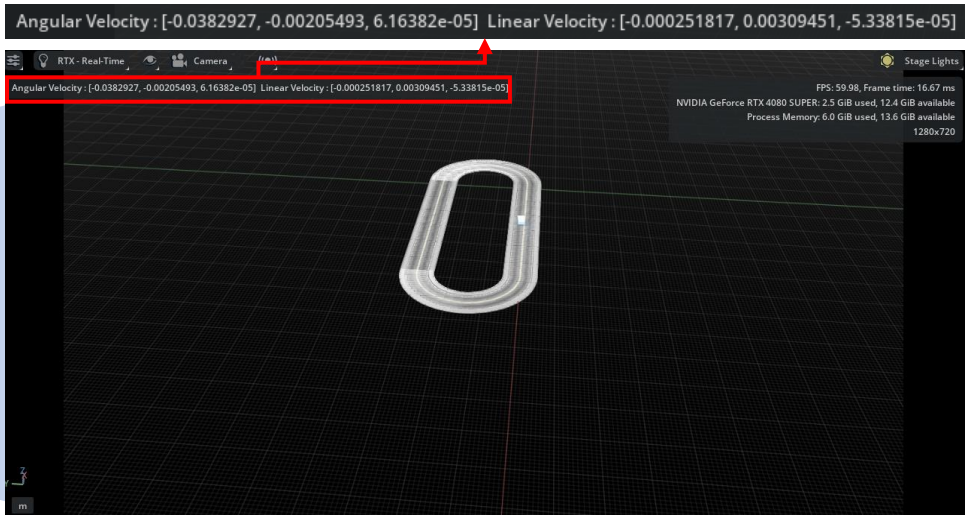
- **Python Script를 통한 Vehicle Velocity Data 실시간 추출**



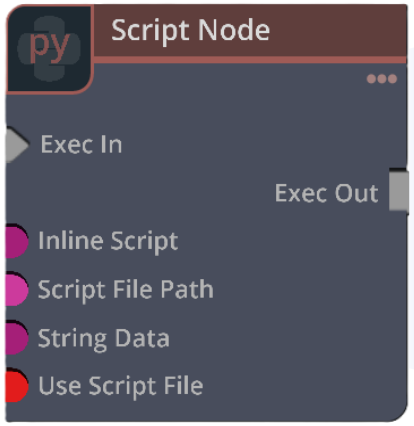
<On Tick>
: 시뮬레이션이 실행되는 동안, 매 프레임마다 노드를 실행시키는 트리거 역할 수행

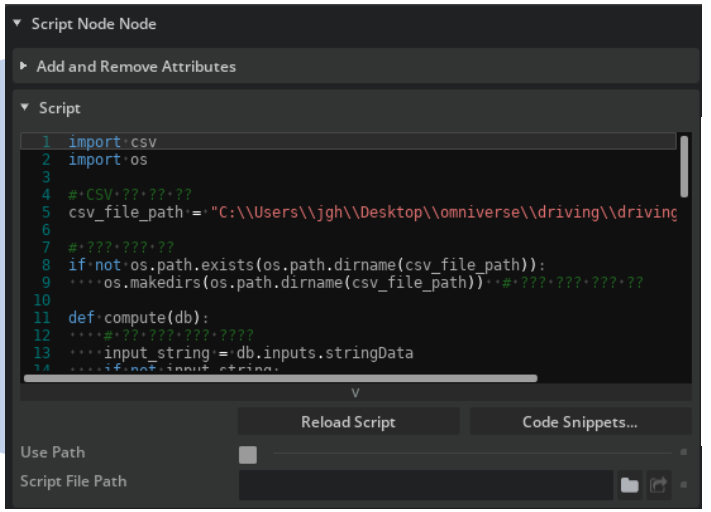
실시간 실행이 요구되는 모든 노드에게 가장 중요한 노드!!





실시간 Velocity Data Monitoring 가능



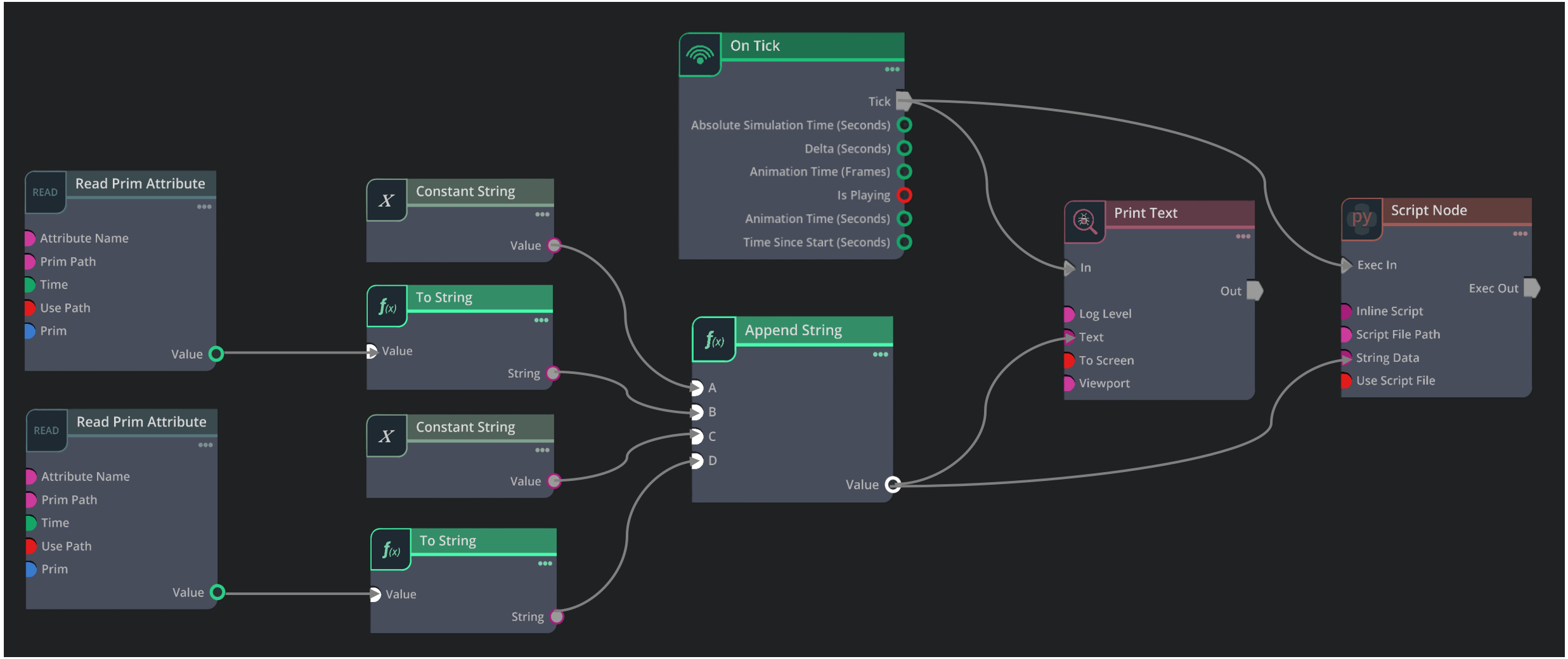


csv file

Python Script를 통한 Vehicle Velocity Data 추출 가능

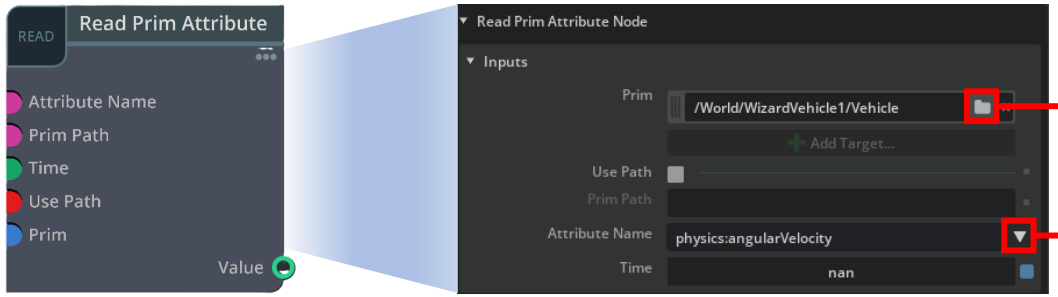
Action Graph 기반 데이터 추출 로직 구현 및 실습: Answer

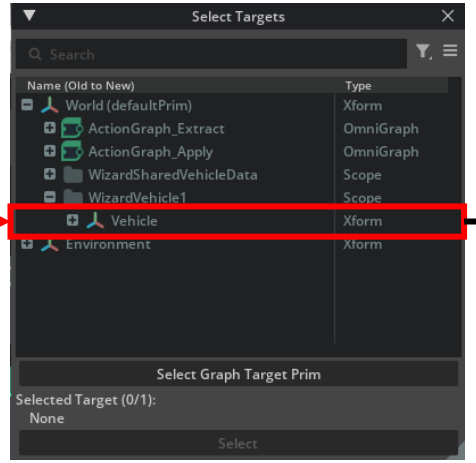
- Vehicle Velocity 데이터 추출을 위한 Action Graph



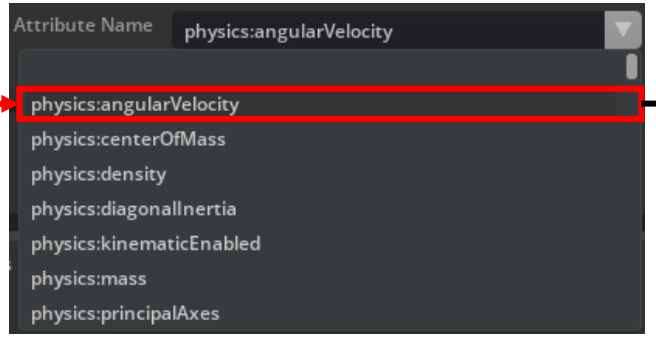
Action Graph 기반 데이터 추출 로직 구현 및 실습 [1/3]

- **Vehicle Prim의 Velocity 데이터 가져오기**

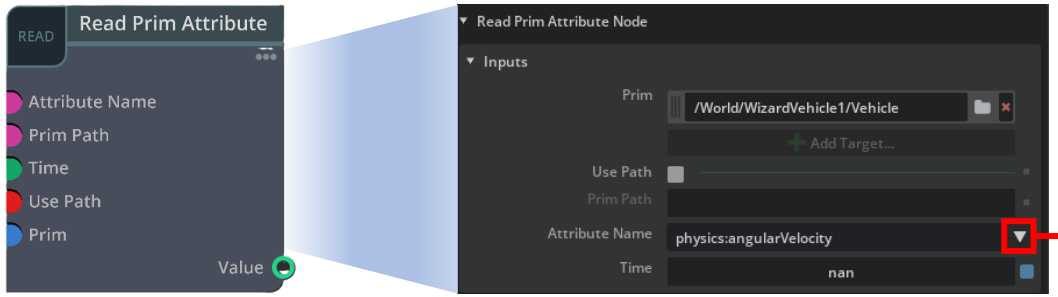


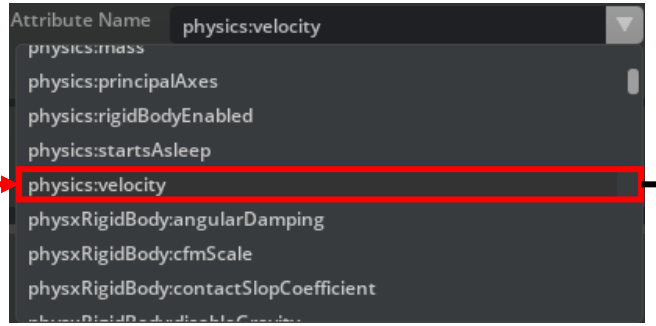


속성을 불러올 Prim 객체 설정
(WizardVehicle1 → Vehicle)



객체의 불러올 속성 설정
(physics:angularVelocity)

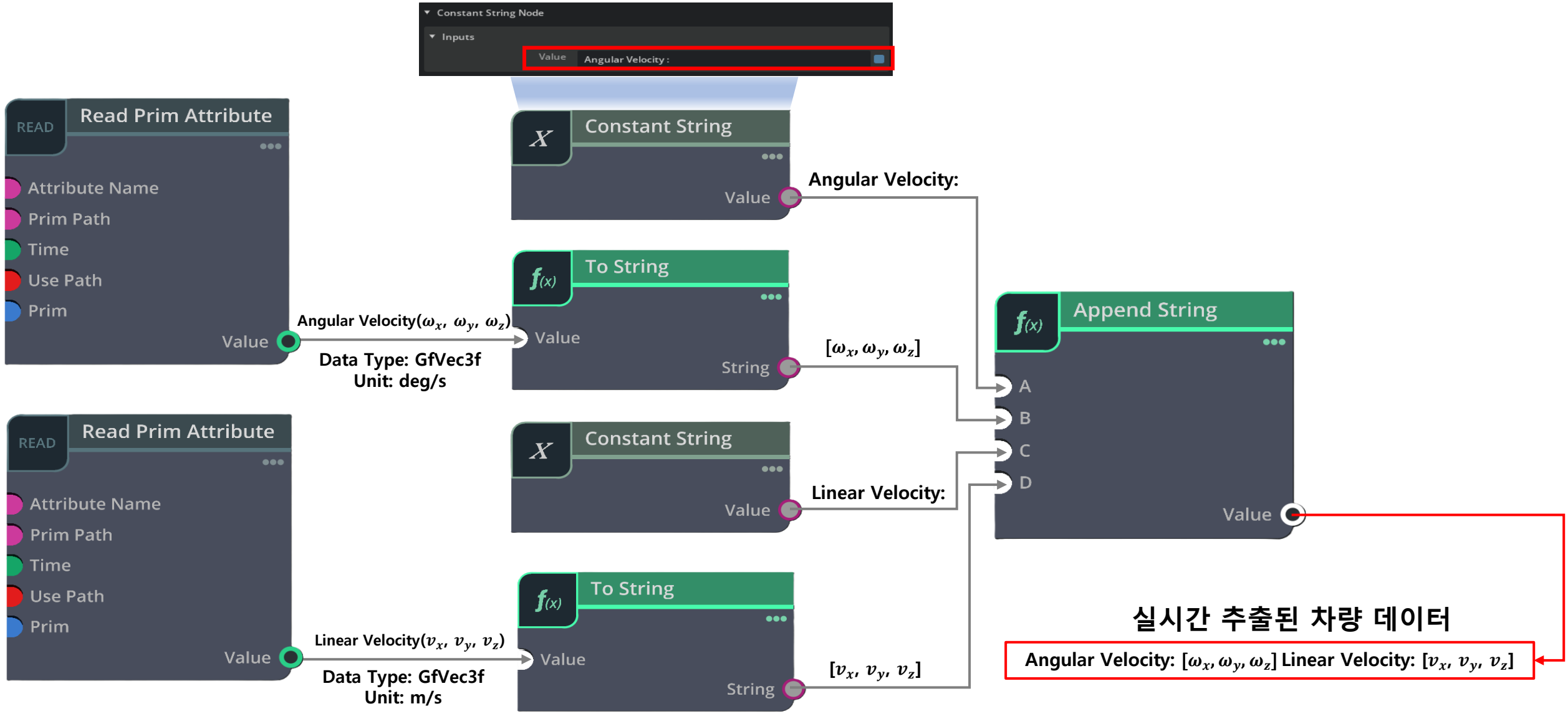




객체의 불러올 속성 설정
(physics:Velocity)

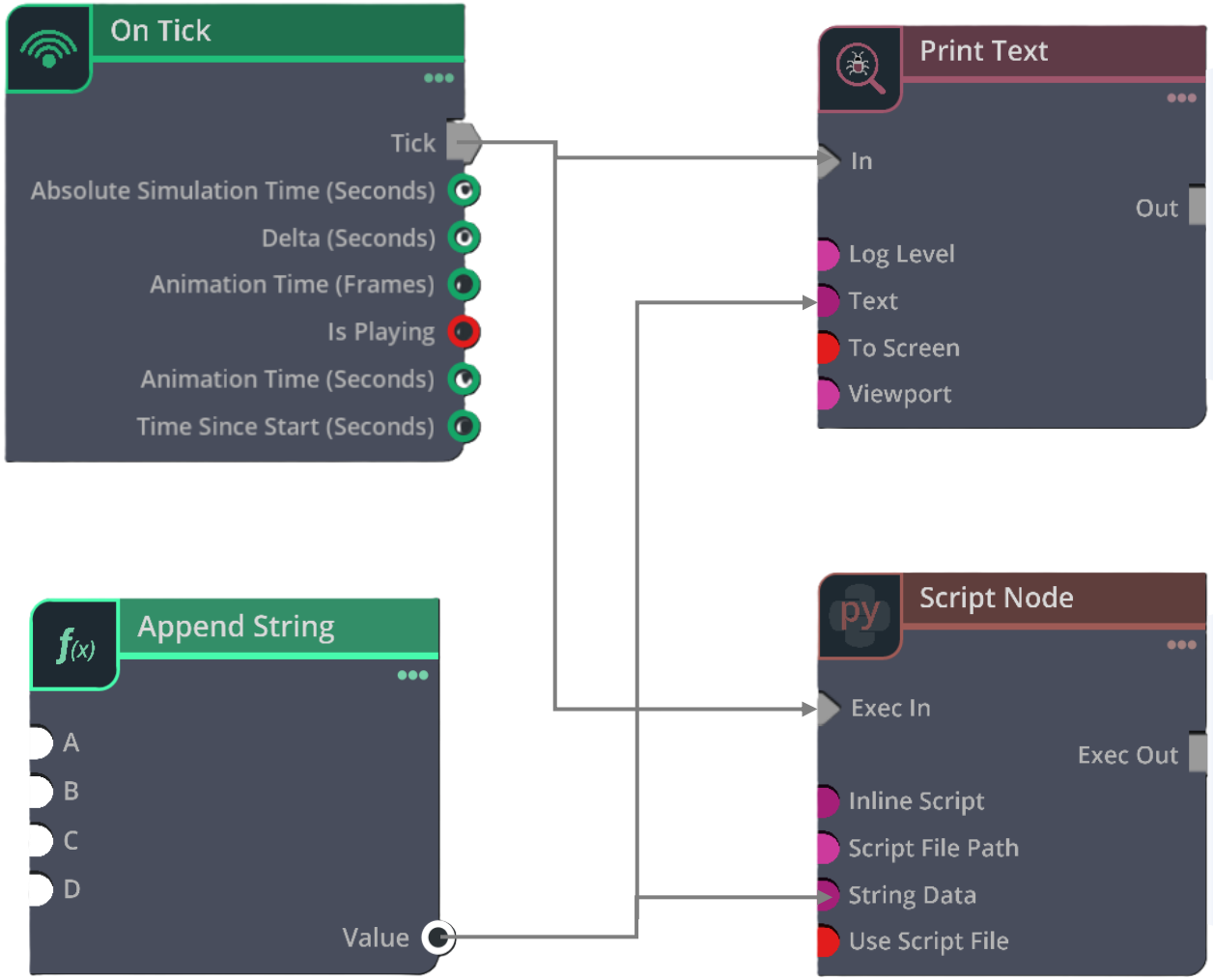
Action Graph 기반 데이터 추출 로직 구현 및 실습 [2/3]

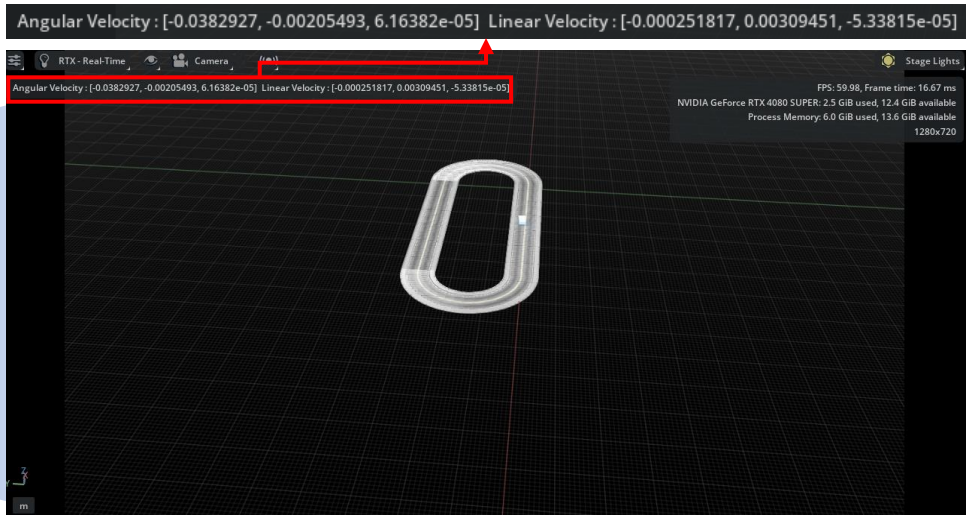
- 불러온 차량 Velocity 데이터 String 타입으로 변환 및 결합



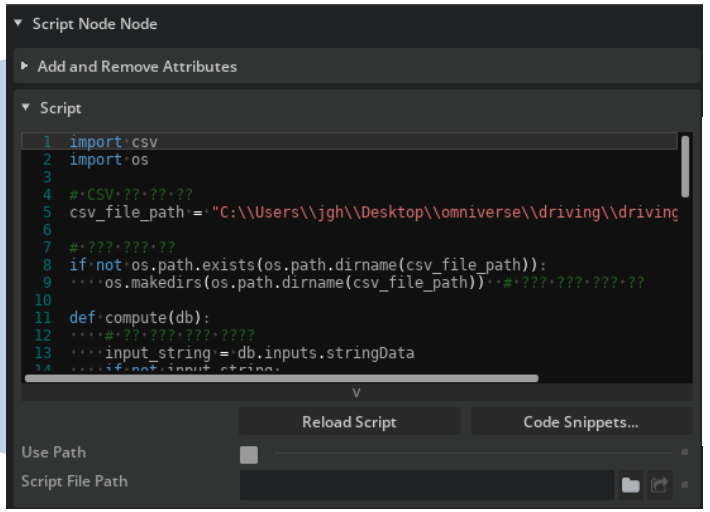
Action Graph 기반 데이터 추출 로직 구현 및 실습 [3/3]

- **Python Script를 통한 Vehicle Velocity Data 실시간 추출 [1/2]**





실시간 Velocity Data Monitoring 가능



csv file

Python Script를 통한 Vehicle Velocity Data 추출 가능

Action Graph 기반 데이터 추출 로직 구현 및 실습 [3/3]

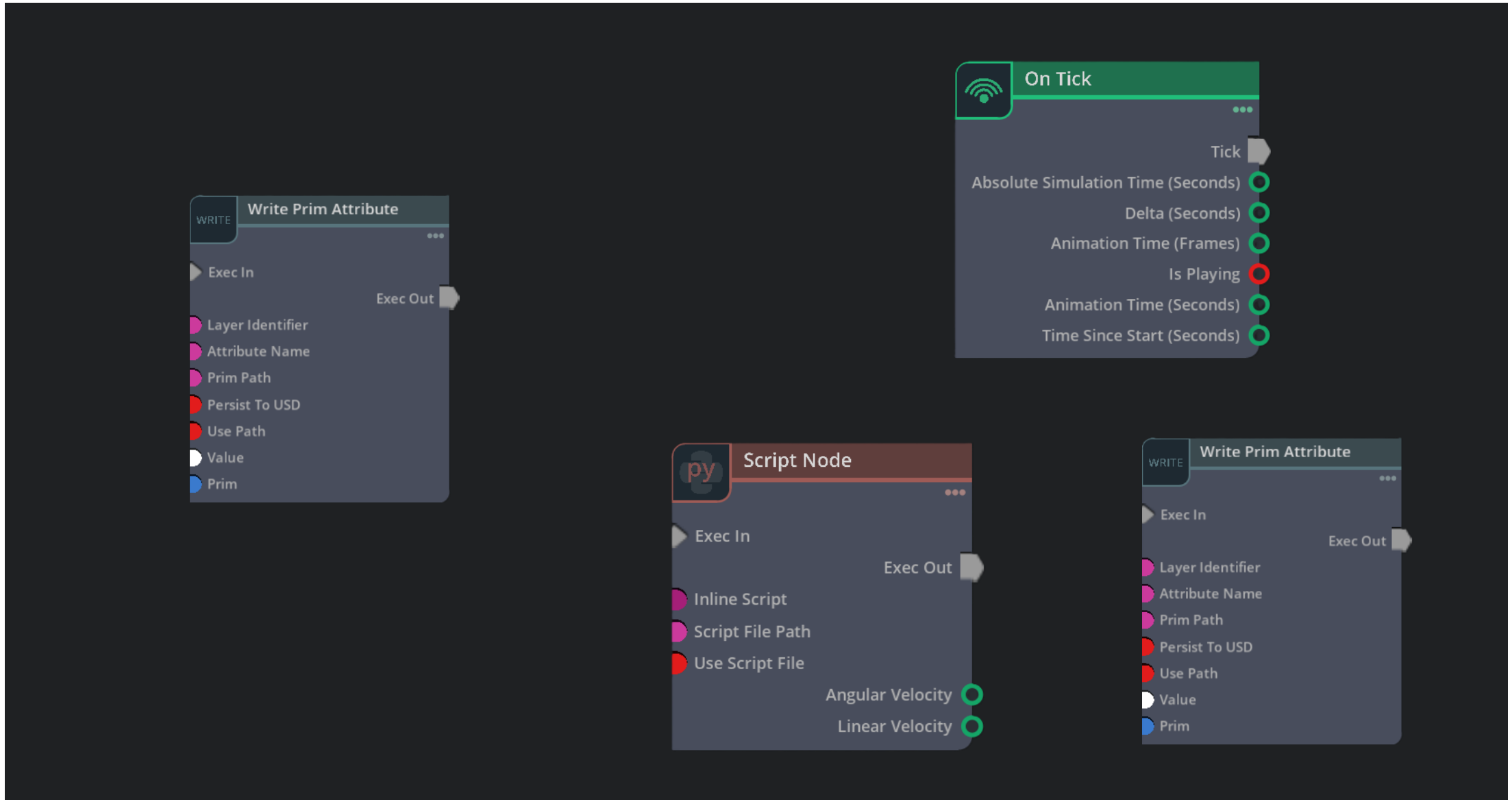
- **Python Script를 통한 Vehicle Velocity Data 실시간 추출 [2/2]**

```
import csv
import os

# CSV 파일 경로 (사용자 환경에 맞게 변경 필요)
#예시: csv_file_path =
"C:\\Users\\jgh\\Desktop\\omniverse\\driving\\driving_data_velocity.csv"
csv_file_path = "C:\\path\\to\\your\\csvfile.csv"

# 폴더가 없으면 생성
if not os.path.exists(os.path.dirname(csv_file_path)):
    os.makedirs(os.path.dirname(csv_file_path))  # 지정된 경로의 폴더를 생성

def compute(db):
    # 입력 문자열 데이터 가져오기
    input_string = db.inputs.stringData
    if not input_string:
        print("Debug: No data received in stringData") # 데이터가 없을 경우 경고 출력
        return

    # 수신된 데이터 출력
    print(f"Debug: Received data: {input_string}")

    # 입력 문자열에서 Angular Velocity와 Linear Velocity 추출
    try:
        # 문자열을 파싱하여 Angular Velocity 부분 추출
        angular_velocity_part = input_string.split("Angular Velocity :
")[1].split("  Linear Velocity : ")[0]
        # Linear Velocity 부분 추출
        linear_velocity_part = input_string.split("Linear Velocity : ")[1]

        # 문자열을 리스트 형태로 변환
        angular_velocity = eval(angular_velocity_part)
        linear_velocity = eval(linear_velocity_part)

        # CSV 파일이 존재하지 않으면 헤더 작성
        if not os.path.exists(csv_file_path):
            print(f"Debug: File does not exist. Creating new file at
{csv_file_path}")
            with open(csv_file_path, mode="w", newline="") as file:
                writer = csv.writer(file)
                # CSV 헤더 작성
                writer.writerow(["Angular Velocity X", "Angular Velocity Y",
"Angular Velocity Z",
                                 "Linear Velocity X", "Linear Velocity Y",
"Linear Velocity Z"])  # 헤더 추가

        # 데이터 저장
        with open(csv_file_path, mode="a", newline="") as file:
            writer = csv.writer(file)
            # Angular Velocity와 Linear Velocity 값을 CSV 파일에 추가
            writer.writerow([angular_velocity[0], angular_velocity[1],
angular_velocity[2],
                             linear_velocity[0], linear_velocity[1],
linear_velocity[2]])  # 데이터 추가
            print(f"Debug: Data written to file: {angular_velocity},
{linear_velocity}")  # 저장된 데이터 출력

    except Exception as e:
        print(f"Error: {e}")  # 오류 발생 시 오류 메시지 출력
```

Contents

Action Graph 기반 데이터 적용 로직 구현: Quiz

- 추출된 데이터 적용을 위한 Action Graph

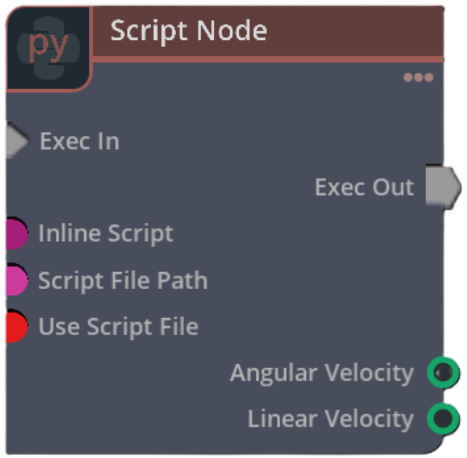


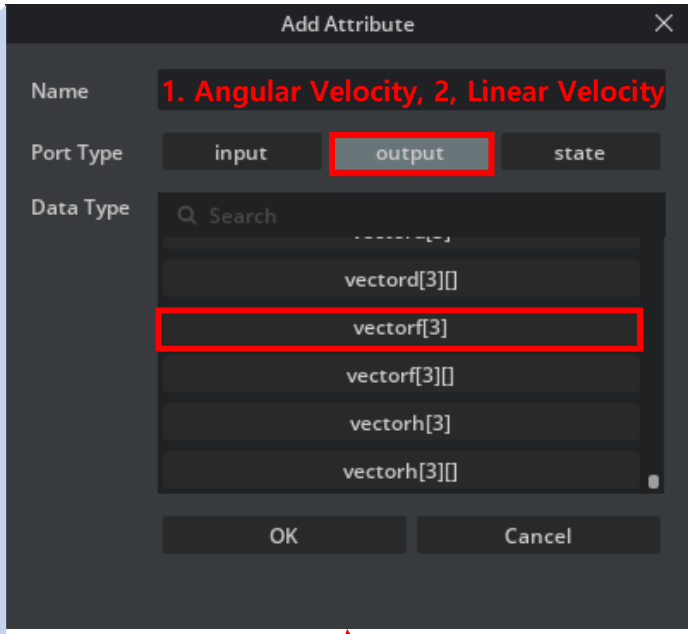
Action Graph 기반 데이터 적용 로직 구현: 데이터 적용 노드 [1/2]

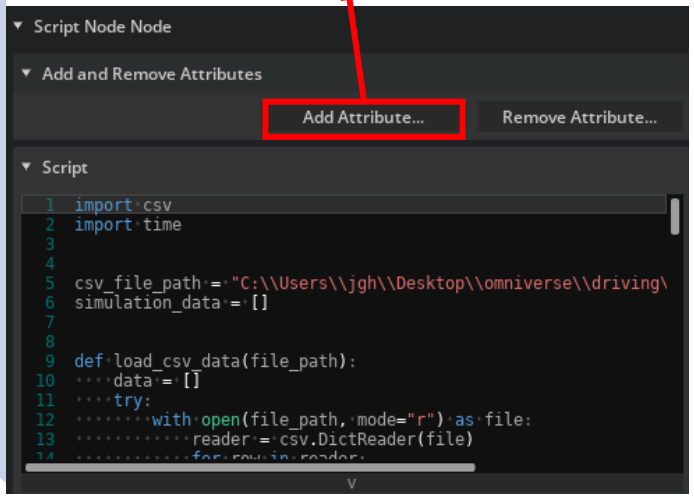
- Python Script를 통한 추출된 Vehicle Velocity Data Import

?

시뮬레이션이 실행되는 동안,
매 프레임마다 노드를 실행시키는
트리거 역할 수행 노드 연결



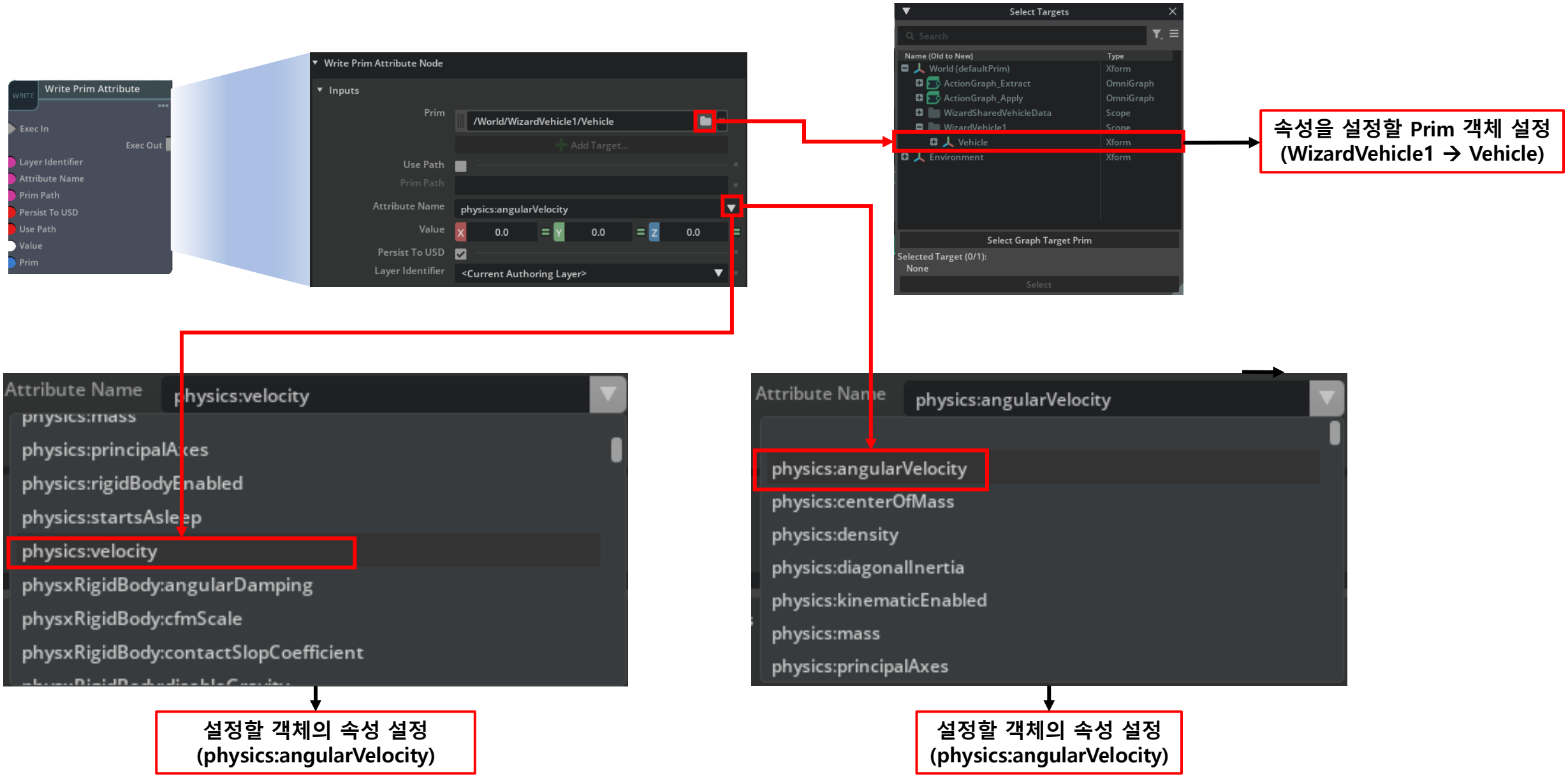




Python Script를 통해,
Angular Velocity, Linear Velocity data를
매 프레임마다 Omniverse 내로 Import

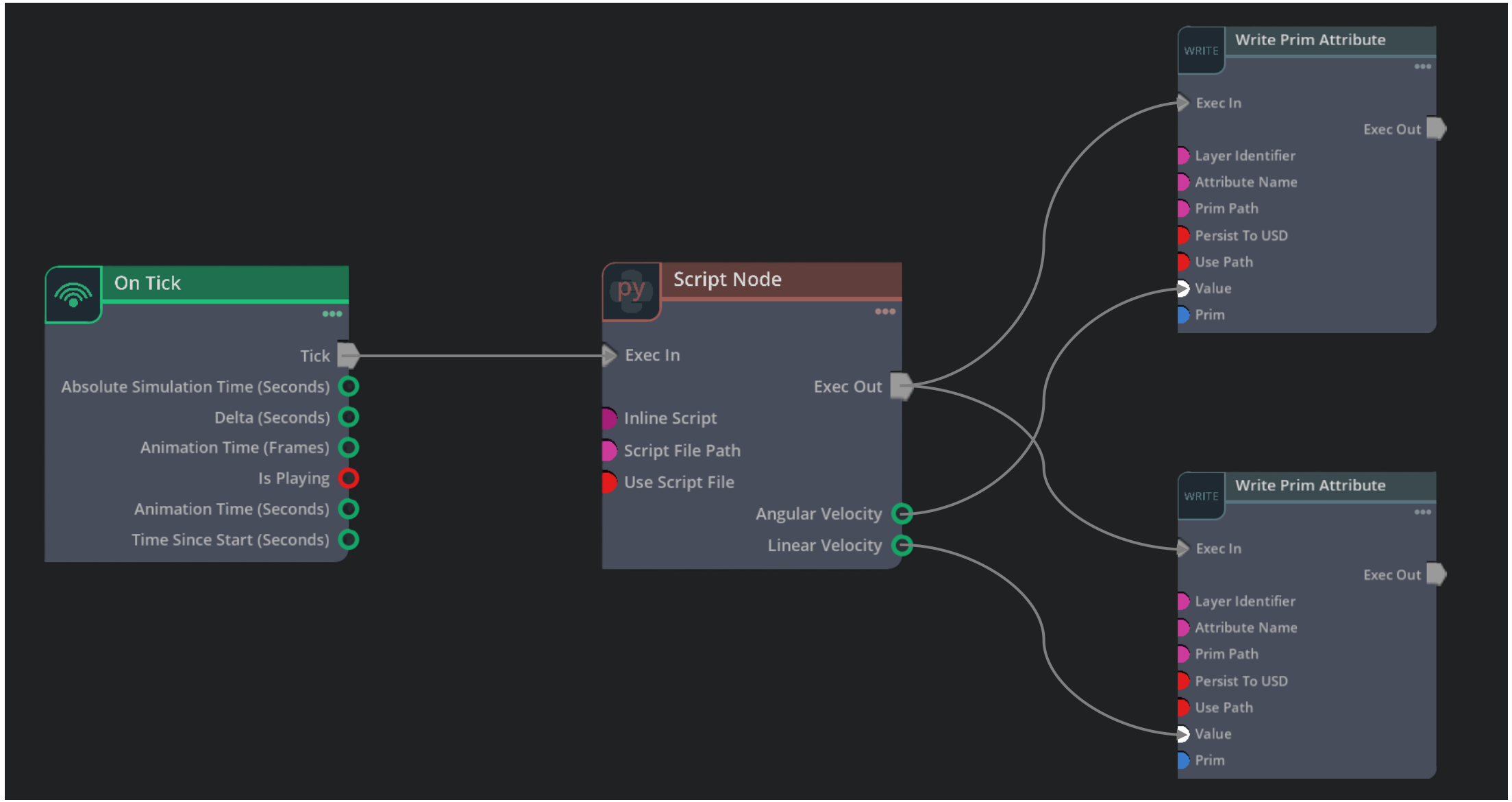
Action Graph 기반 데이터 적용 로직 구현: 데이터 적용 노드 [2/2]

- 실시간 Import된 Linear/Angular Velocity 데이터를 Vehicle Prim에 적용



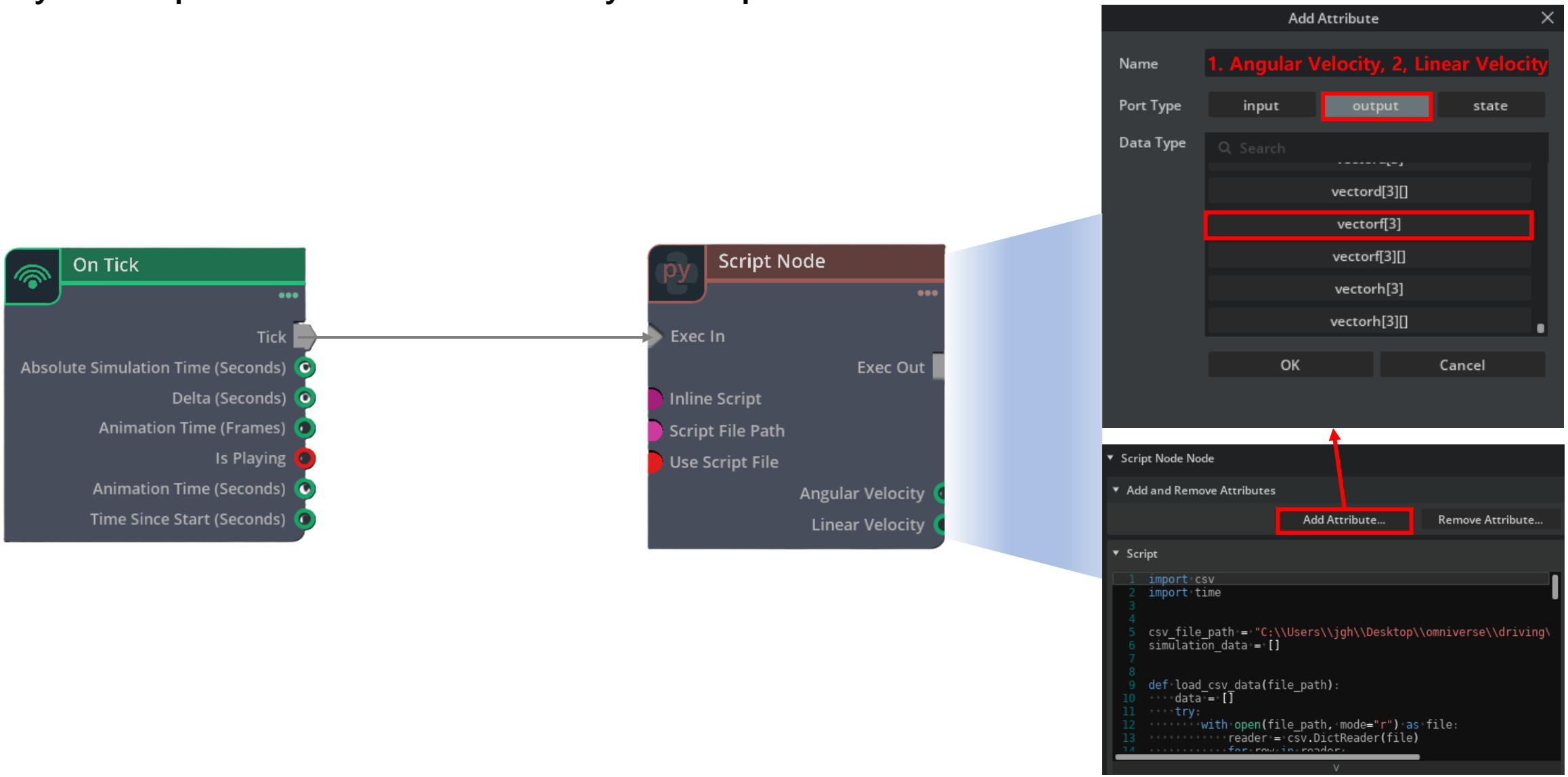
Action Graph 기반 데이터 적용 로직 구현: Answer

- 추출된 데이터 적용을 위한 Action Graph



Action Graph 기반 데이터 적용 로직 구현 [1/3]

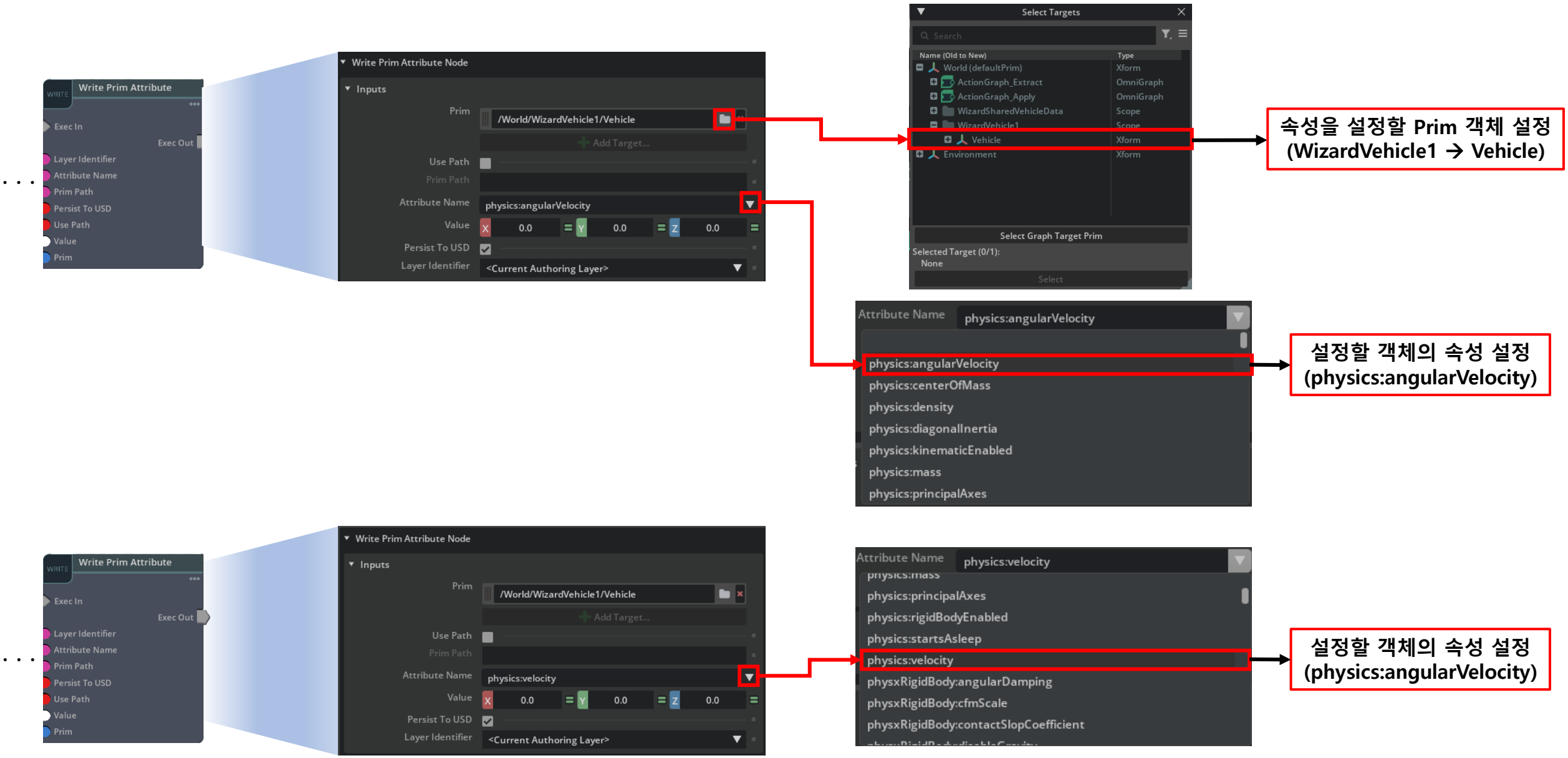
- **Python Script를 통한 추출된 Vehicle Velocity Data Import**



Python Script를 통한 Angular Velocity, Linear Velocity Import

Action Graph 기반 데이터 적용 로직 구현 [2/3]

- Import된 Vehicle Velocity 데이터를 Vehicle Prim에 적용



Action Graph 기반 데이터 추출 로직 구현 및 실습 [3/3]

- **Python Script를 통한 Vehicle Velocity Data 실시간 적용 [1/2]**

```
import csv
import time

# CSV 파일 경로 (사용자 환경에 맞게 변경 필요)
csv_file_path = "C:\\path\\to\\your\\csvfile.csv"

# 시뮬레이션 데이터를 저장할 리스트
simulation_data = []

# CSV 데이터 로드
simulation_data = load_csv_data(csv_file_path)

# 원본 및 타겟 프레임 레이트 설정
source_fps = 93.3  # 원본 데이터의 FPS
target_fps = 93.3  # 목표 FPS

# 데이터 다운샘플링 (현재 FPS가 같으므로 변경 없음)
simulation_data = downsample_data(simulation_data, target_fps, source_fps)

# 프레임 지속 시간 계산
target_frame_duration = 1.0 / target_fps  # 각 프레임의 시간 간격
last_update_time = time.time()  # 마지막 업데이트 시간 기록

frame = 0  # 현재 프레임 인덱스
is_playing = True  # 시뮬레이션 실행 여부
```

```
def load_csv_data(file_path):
    """
    CSV 파일을 로드하여 각 행의 데이터를 딕셔너리 리스트로 변환
    """
    data = []
    try:
        with open(file_path, mode="r") as file:
            reader = csv.DictReader(file)
            for row in reader:
                try:
                    # 각 행에서 각 속도 데이터를 가져와서 리스트로 변환
                    angular_velocity = [
                        float(row.get("Angular Velocity X", 0.0)),
                        float(row.get("Angular Velocity Y", 0.0)),
                        float(row.get("Angular Velocity Z", 0.0))]
                    linear_velocity = [
                        float(row.get("Linear Velocity X", 0.0)),
                        float(row.get("Linear Velocity Y", 0.0)),
                        float(row.get("Linear Velocity Z", 0.0))]

                    # 변환된 데이터를 리스트에 추가
                    data.append({
                        "angular_velocity": angular_velocity,
                        "linear_velocity": linear_velocity
                    })
                except ValueError as e:
                    print(f"Error parsing row {row}: {e}")
        print("CSV file loaded successfully!")
    except Exception as e:
        print(f"Error loading CSV file: {e}")
    return data
```

Action Graph 기반 데이터 추출 로직 구현 및 실습 [3/3]

- **Python Script를 통한 Vehicle Velocity Data 실시간 적용 [2/2]**

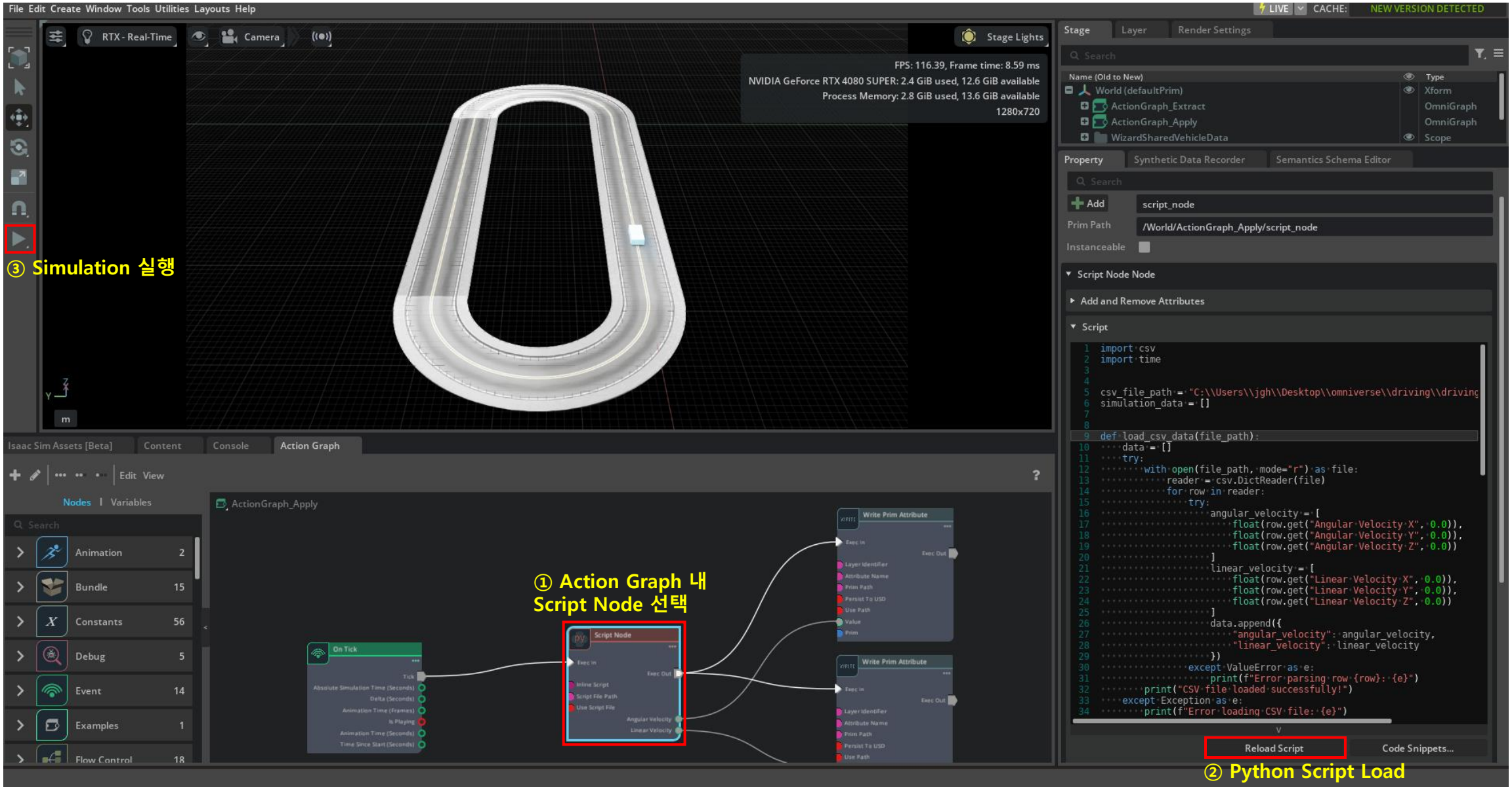
```python
def downsample_data(data, target_fps, source_fps):
    """
    원본 FPS에서 타겟 FPS로 다운샘플링 (프레임 수 줄이기)
    """
    step = int(source_fps / target_fps)  # 몇 프레임마다 하나를 선택할지 결정
    return data[::step]  # 지정한 간격으로 데이터를 샘플링하여 반환

def compute(db):
    """
    프레임마다 실행되는 함수. 시뮬레이션 데이터를 기반으로 출력값을 갱신.
    """
    global frame, simulation_data, last_update_time, is_playing

    # 시뮬레이션이 정지된 경우 종료
    if not is_playing:
        print("Playback stopped.")
        return

    # 현재 시간이 마지막 업데이트 시간보다 프레임 간격보다 적다면 대기
    current_time = time.time()
    if current_time - last_update_time < target_frame_duration:
        return  # 아직 프레임을 갱신할 시간이 되지 않음

    # 모든 데이터를 재생한 경우 종료
    if frame >= len(simulation_data):
        print("End of simulation data. Stopping playback.")
        db.outputs.angularVelocity = (0.0, 0.0, 0.0)  # 정지 상태로 설정
        db.outputs.linearVelocity = (0.0, 0.0, 0.0)
        is_playing = False  # 재생 중지
        return
```

```python
    # 현재 프레임의 데이터 가져오기
    data = simulation_data[frame]
    angular_velocity = data["angular_velocity"]
    linear_velocity = data["linear_velocity"]

    # 출력 값 업데이트
    db.outputs.angularVelocity = tuple(angular_velocity)
    db.outputs.linearVelocity = tuple(linear_velocity)

    # 10프레임마다 로그 출력
    if frame % 10 == 0:
        print(f"Frame {frame}: Angular Velocity = {angular_velocity}, Linear
Velocity = {linear_velocity}")

    # 다음 프레임으로 이동
    frame += 1
    last_update_time = current_time
```

Contents

추출된 데이터 기반 Simulation 실습

- 추출된 데이터 import를 통한 차량 Simulation



Contents

Q & A

Thank You