2025.05.07.(수)

NS3 Code Generator

: LLM 기반의 네트워크 시뮬레이션 코드 자동 생성

202101109 박수화
202102722 최희원

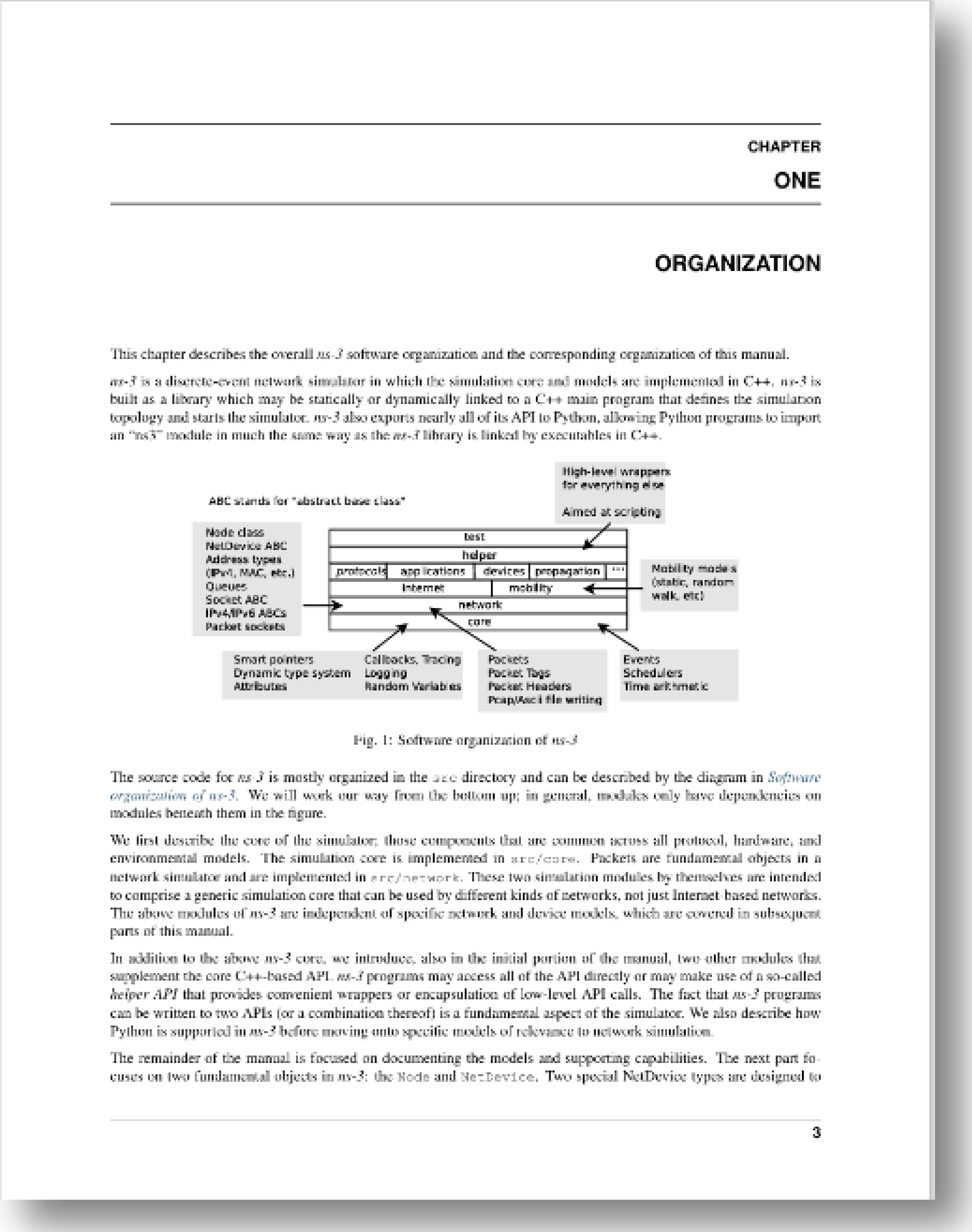
목차

1 **Data Cleaning**

2 **RAG DB로 사용 결과**

Data Cleaning(NS3-Manual)

불필요한 페이지(표지, 목차, 참고자료 등), 상하단 구분선, 쪽수, 이미지 등 제거



CHAPTER

ONE

ORGANIZATION

This chapter describes the overall *ns-3* software organization and the corresponding organization of this manual.

ns-3 is a discrete-event network simulator in which the simulation core and models are implemented in C++. *ns-3* is built as a library which may be statically or dynamically linked to a C++ main program that defines the simulation topology and starts the simulator. *ns-3* also exports nearly all of its API to Python, allowing Python programs to import an "ns3" module in much the same way as the *ns-3* library is linked by executables in C++.

Fig. 1: Software organization of *ns-3*

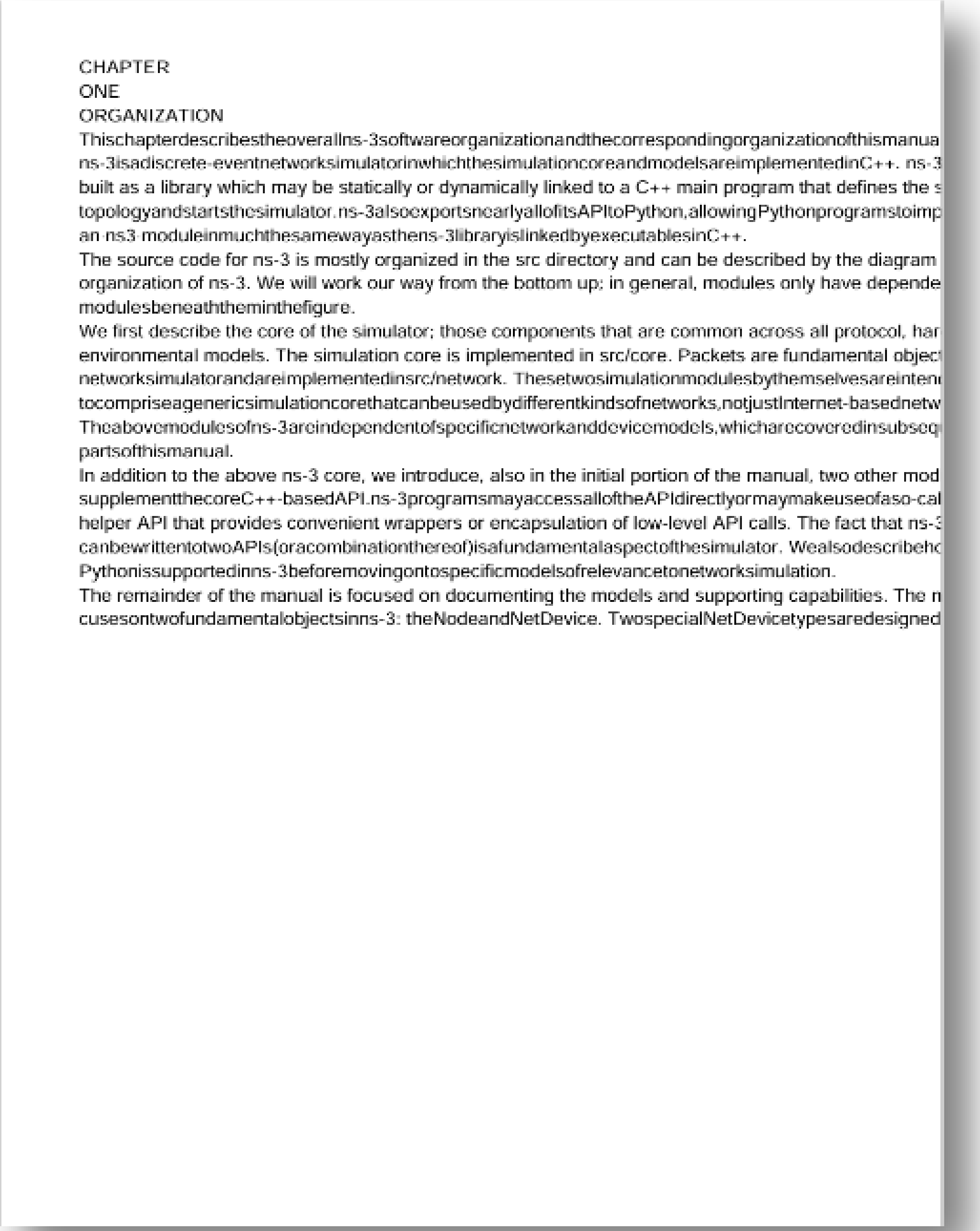
The source code for *ns-3* is mostly organized in the `src` directory and can be described by the diagram in *Software organization of ns-3*. We will work our way from the bottom up; in general, modules only have dependencies on modules beneath them in the figure.

We first describe the core of the simulator; those components that are common across all protocol, hardware, and environmental models. The simulation core is implemented in `src/core`. Packets are fundamental objects in a network simulator and are implemented in `src/network`. These two simulation modules by themselves are intended to comprise a generic simulation core that can be used by different kinds of networks, not just Internet-based networks. The above modules of *ns-3* are independent of specific network and device models, which are covered in subsequent parts of this manual.

In addition to the above *ns-3* core, we introduce, also in the initial portion of the manual, two other modules that supplement the core C++-based API. *ns-3* programs may access all of the API directly or may make use of a so-called *helper API* that provides convenient wrappers or encapsulation of low-level API calls. The fact that *ns-3* programs can be written to two APIs (or a combination thereof) is a fundamental aspect of the simulator. We also describe how Python is supported in *ns-3* before moving onto specific models of relevance to network simulation.

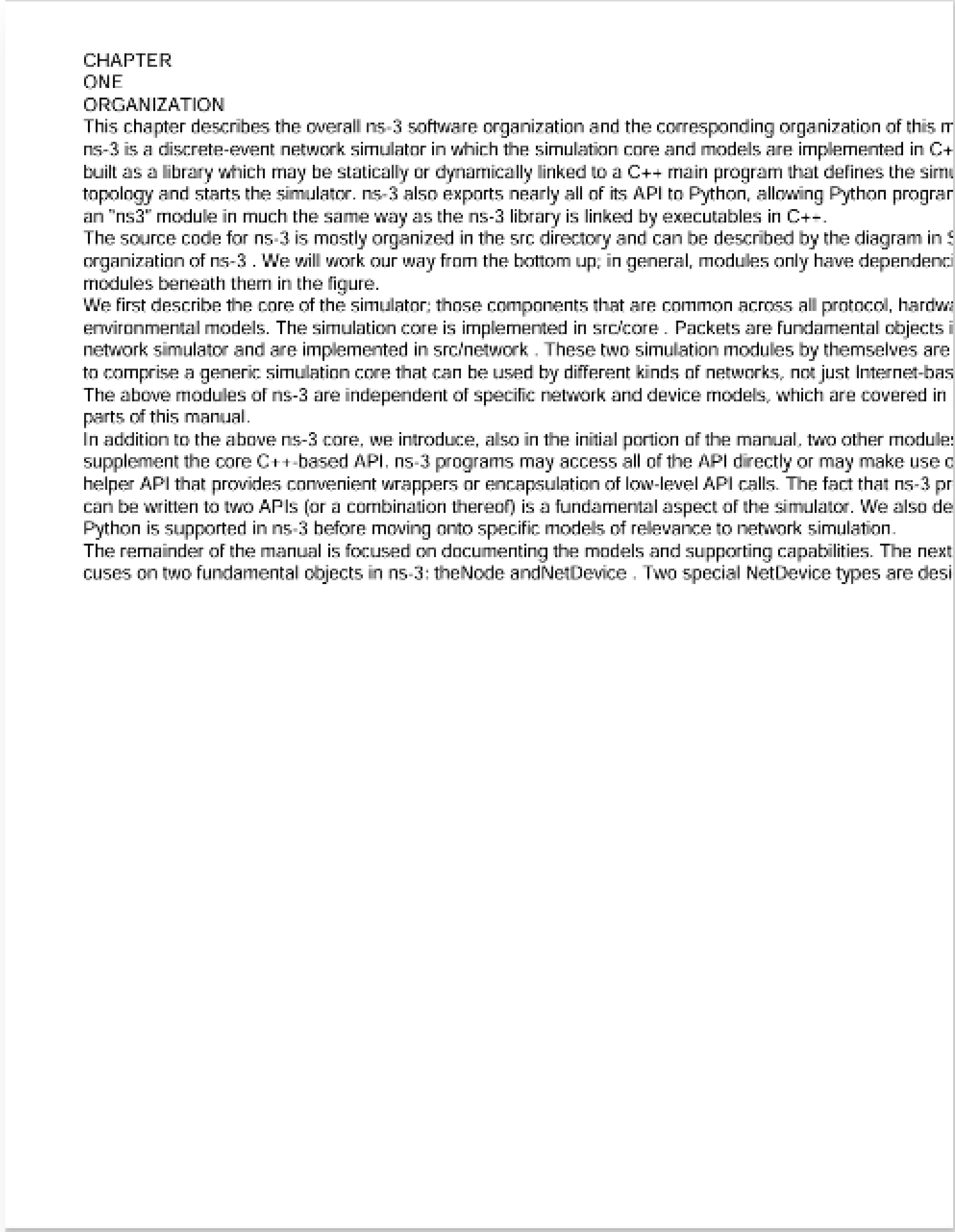
The remainder of the manual is focused on documenting the models and supporting capabilities. The next part focuses on two fundamental objects in *ns-3*: the `Node` and `NetDevice`. Two special NetDevice types are designed to

→



CHAPTER
ONE
ORGANIZATION
Thischapterdescribestheoverallns-3softwareorganizationandthecorrespondingorganizationofthismanua
ns-3isadiscrete-eventnetworksimulatorinwhichthesimulationcoreandmodelsareimplementedinC++. ns-3
built as a library which may be statically or dynamically linked to a C++ main program that defines the s
topologyandstartsthesimulator.ns-3alsoexportsnearlyallofitsAPItoPython,allowingPythonprogramstoimp
an ns3 moduleinmuchthesamewayasthens-3libraryislinkedbyexecutablesinC++.
The source code for ns-3 is mostly organized in the src directory and can be described by the diagram
organization of ns-3. We will work our way from the bottom up; in general, modules only have depende
modulesbeneaththeminthefigure.
We first describe the core of the simulator; those components that are common across all protocol, har
environmental models. The simulation core is implemented in src/core. Packets are fundamental objec
networksimulatorandareimplementedinsrc/network. Thesetwosimulationmodulesbythemselvesareinten
tocompriseagenericsimulationcorethatcanbeusedbydifferentkindsofnetworks,notjustInternet-basednetw
Theabovemodulesofns-3areindependentofspecificnetworkanddevicemodels,whicharecoveredinsubseq
partsofthismanual.
In addition to the above ns-3 core, we introduce, also in the initial portion of the manual, two other mod
supplementthecoreC++-basedAPI.ns-3programsmayaccessalloftheAPIdirectlyormaymakeuseofaso-cal
helper API that provides convenient wrappers or encapsulation of low-level API calls. The fact that ns-3
canbewrittentotwoAPIs(oracombinationthereof)isafundamentalaspectofthesimulator. Wealsodescribeho
Pythonissupportedinns-3beforemovingontospecificmodelsofrelevancetonetworksimulation.
The remainder of the manual is focused on documenting the models and supporting capabilities. The n
cusesontwofundamentalobjectsinns-3: theNodeandNetDevice. TwospecialNetDevicetypesaredesigned

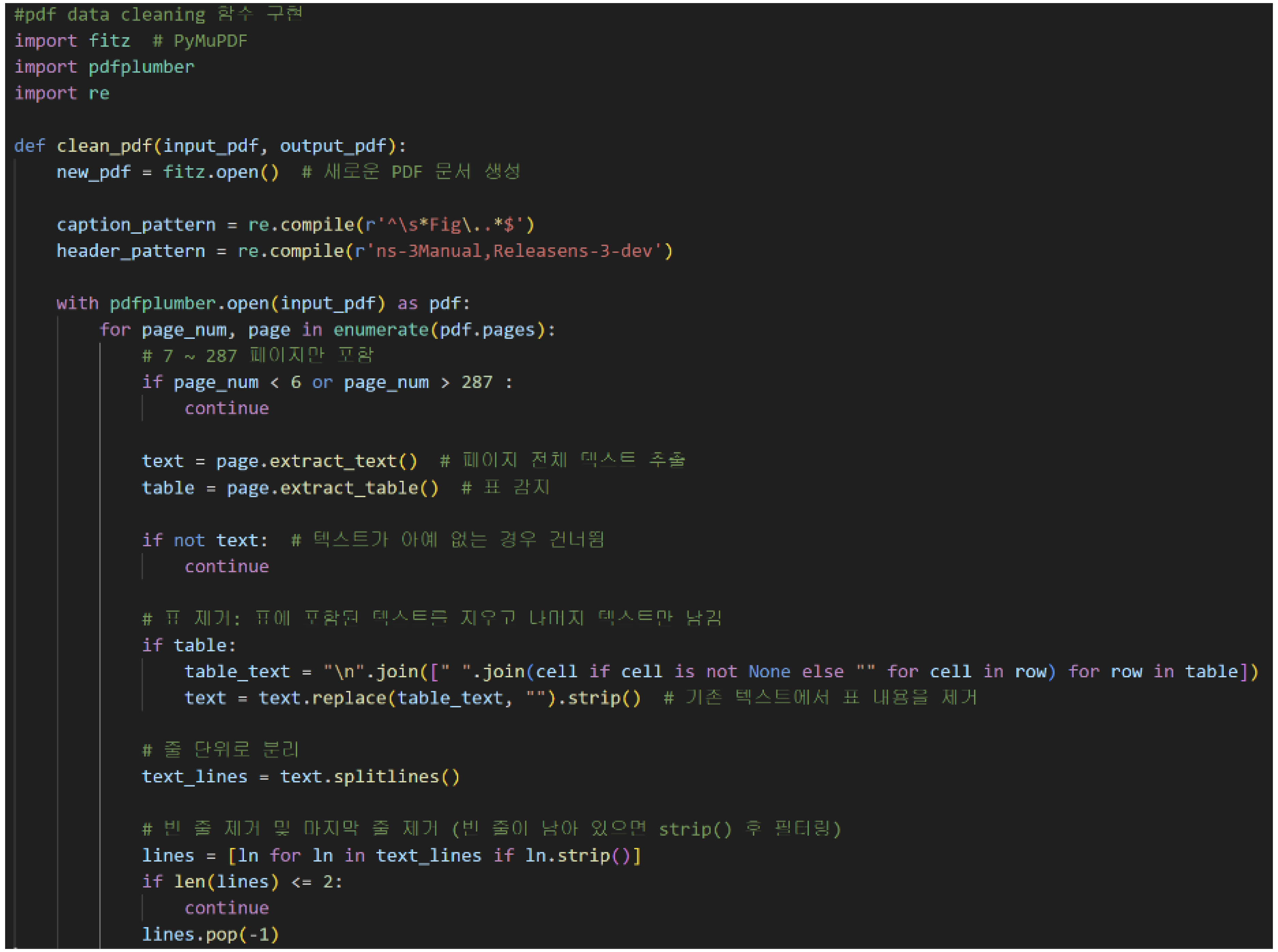
→



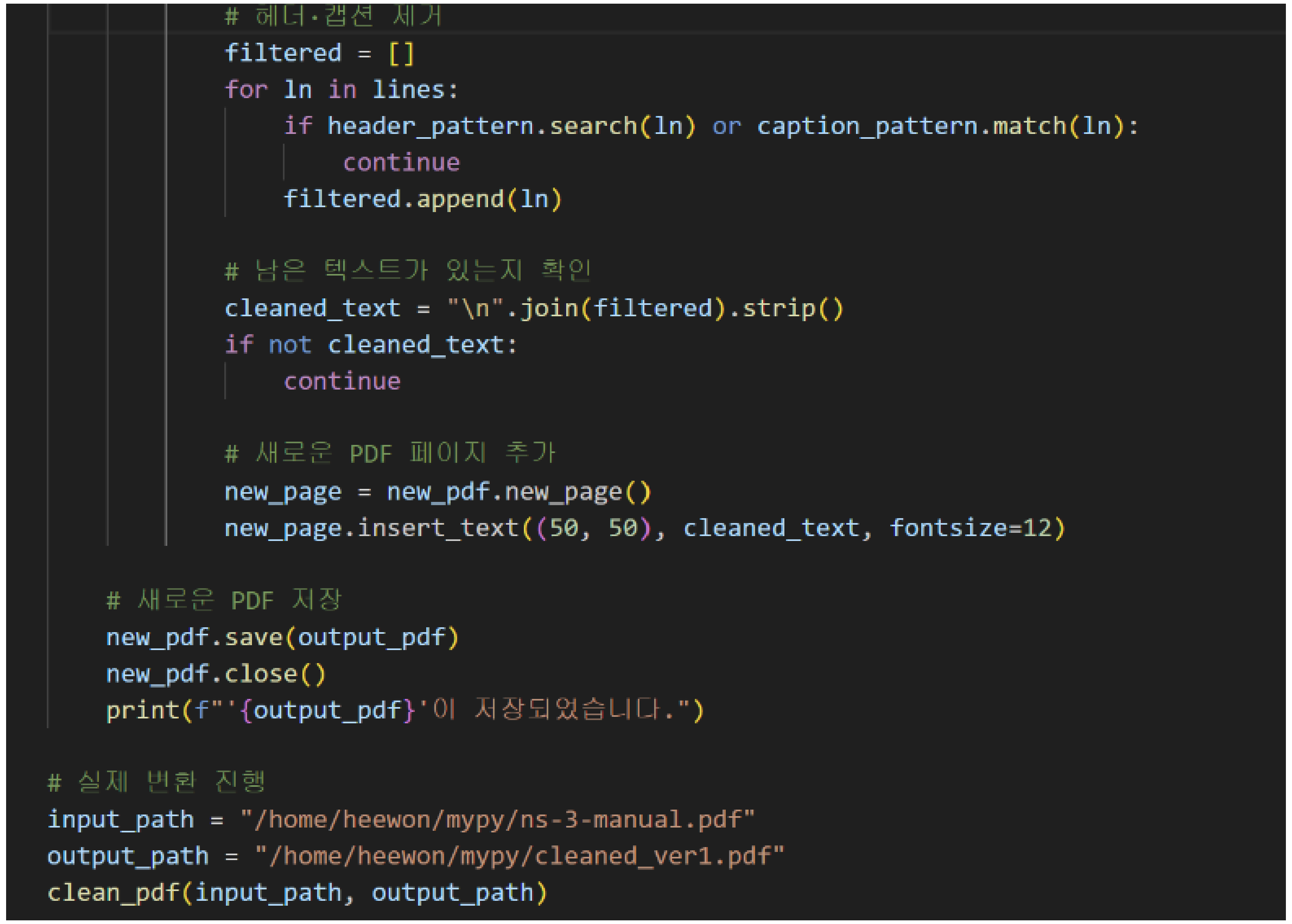
CHAPTER
ONE
ORGANIZATION
This chapter describes the overall ns-3 software organization and the corresponding organization of this m
ns-3 is a discrete-event network simulator in which the simulation core and models are implemented in C+
built as a library which may be statically or dynamically linked to a C++ main program that defines the simu
topology and starts the simulator. ns-3 also exports nearly all of its API to Python, allowing Python prograr
an "ns3" module in much the same way as the ns-3 library is linked by executables in C++.
The source code for ns-3 is mostly organized in the src directory and can be described by the diagram in S
organization of ns-3 . We will work our way from the bottom up; in general, modules only have dependenci
modules beneath them in the figure.
We first describe the core of the simulator; those components that are common across all protocol, hardwa
environmental models. The simulation core is implemented in src/core . Packets are fundamental objects i
network simulator and are implemented in src/network . These two simulation modules by themselves are
to comprise a generic simulation core that can be used by different kinds of networks, not just Internet-bas
The above modules of ns-3 are independent of specific network and device models, which are covered in
parts of this manual.
In addition to the above ns-3 core, we introduce, also in the initial portion of the manual, two other module
supplement the core C++-based API. ns-3 programs may access all of the API directly or may make use c
helper API that provides convenient wrappers or encapsulation of low-level API calls. The fact that ns-3 pr
can be written to two APIs (or a combination thereof) is a fundamental aspect of the simulator. We also de
Python is supported in ns-3 before moving onto specific models of relevance to network simulation.
The remainder of the manual is focused on documenting the models and supporting capabilities. The next
cuses on two fundamental objects in ns-3: theNode andNetDevice . Two special NetDevice types are desi

Data Cleaning - 1

불필요한 페이지(표지, 목차, 참고자료 등), 상하단 구분선, 쪽수, 이미지 등 제거



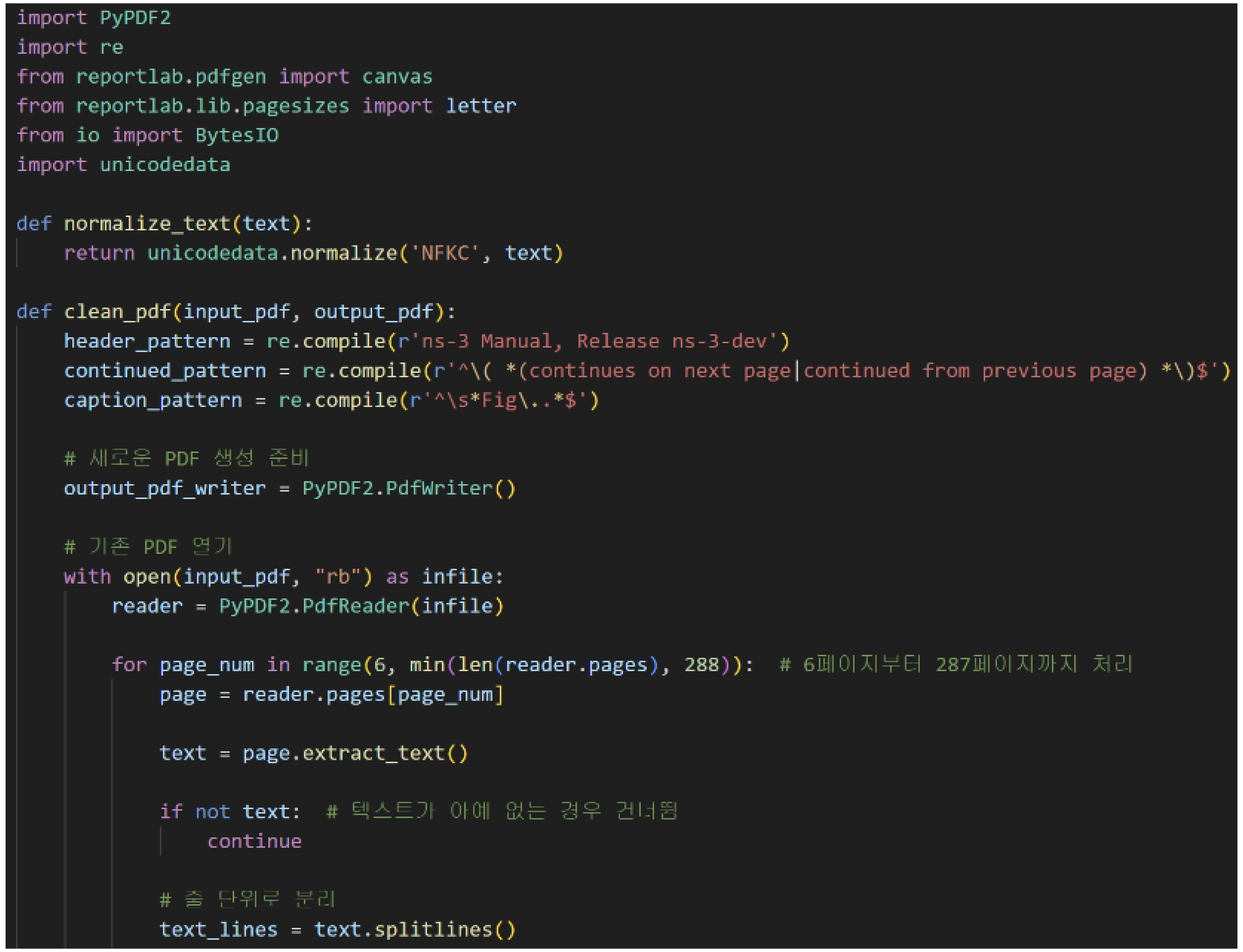
```python
#pdf data cleaning 함수 구현
import fitz  # PyMuPDF
import pdfplumber
import re

def clean_pdf(input_pdf, output_pdf):
    new_pdf = fitz.open()  # 새로운 PDF 문서 생성

    caption_pattern = re.compile(r'^\s*Fig\..*$')
    header_pattern = re.compile(r'ns-3Manual,Releasens-3-dev')

    with pdfplumber.open(input_pdf) as pdf:
        for page_num, page in enumerate(pdf.pages):
            # 7 ~ 287 페이지만 포함
            if page_num < 6 or page_num > 287 :
                continue

            text = page.extract_text()  # 페이지 전체 텍스트 추출
            table = page.extract_table()  # 표 감지

            if not text:  # 텍스트가 아예 없는 경우 건너뜀
                continue

            # 표 제거: 표에 포함된 텍스트를 지우고 나머지 텍스트만 남김
            if table:
                table_text = "\n".join([" ".join(cell if cell is not None else "" for cell in row) for row in table])
                text = text.replace(table_text, "").strip()  # 기존 텍스트에서 표 내용을 제거

            # 줄 단위로 분리
            text_lines = text.splitlines()

            # 빈 줄 제거 및 마지막 줄 제거 (빈 줄이 남아 있으면 strip() 후 필터링)
            lines = [ln for ln in text_lines if ln.strip()]
            if len(lines) <= 2:
                continue
            lines.pop(-1)
```



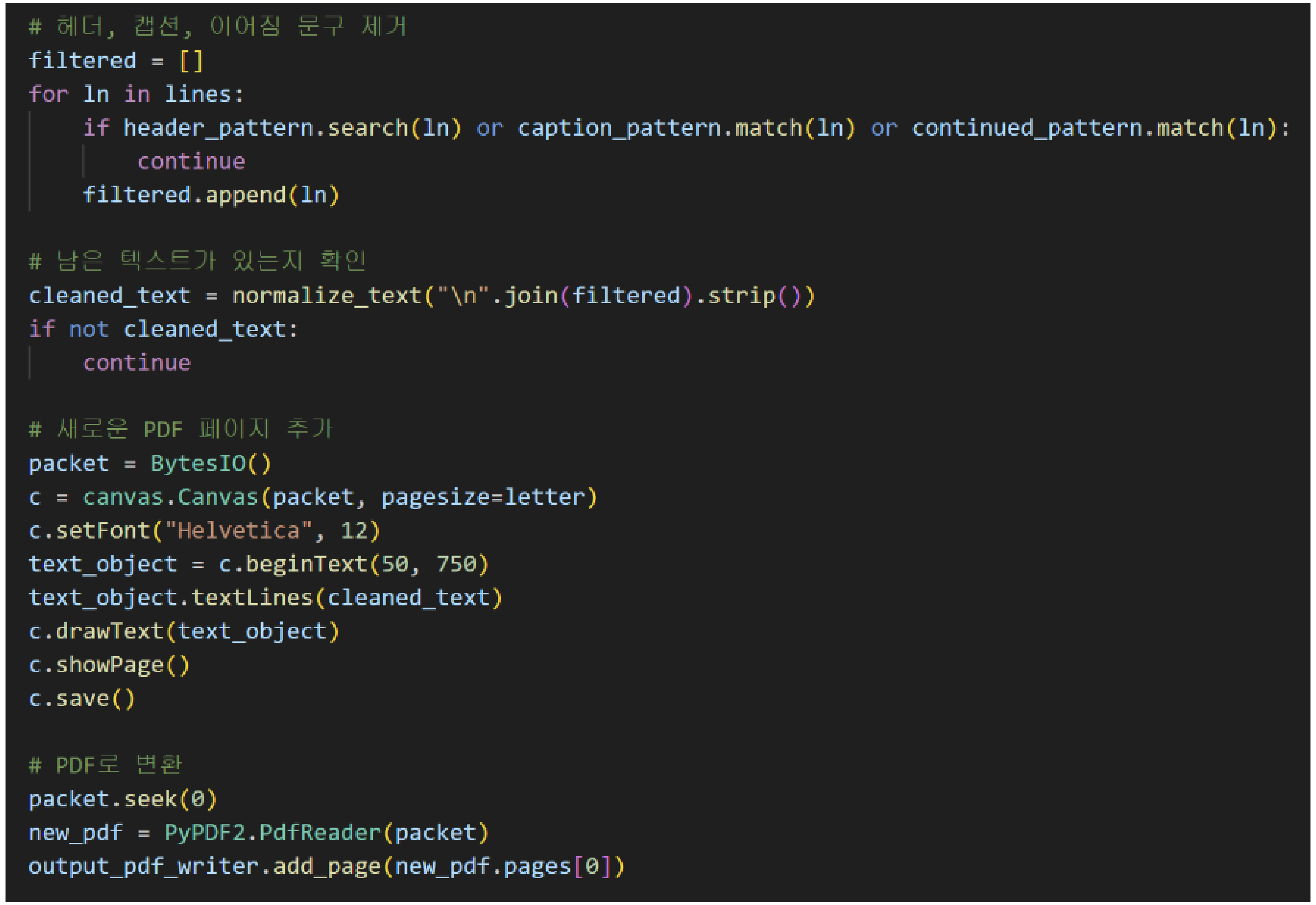
```python
            # 헤더·캡션 제거
            filtered = []
            for ln in lines:
                if header_pattern.search(ln) or caption_pattern.match(ln):
                    continue
                filtered.append(ln)

            # 남은 텍스트가 있는지 확인
            cleaned_text = "\n".join(filtered).strip()
            if not cleaned_text:
                continue

            # 새로운 PDF 페이지 추가
            new_page = new_pdf.new_page()
            new_page.insert_text((50, 50), cleaned_text, fontsize=12)

    # 새로운 PDF 저장
    new_pdf.save(output_pdf)
    new_pdf.close()
    print(f"'{output_pdf}'이 저장되었습니다.")

# 실제 변환 진행
input_path = "/home/heewon/mypy/ns-3-manual.pdf"
output_path = "/home/heewon/mypy/cleaned_ver1.pdf"
clean_pdf(input_path, output_path)
```

Data Cleaning - 2

불필요한 페이지(표지, 목차, 참고자료 등), 상하단 구분선, 쪽수, 이미지 등 제거



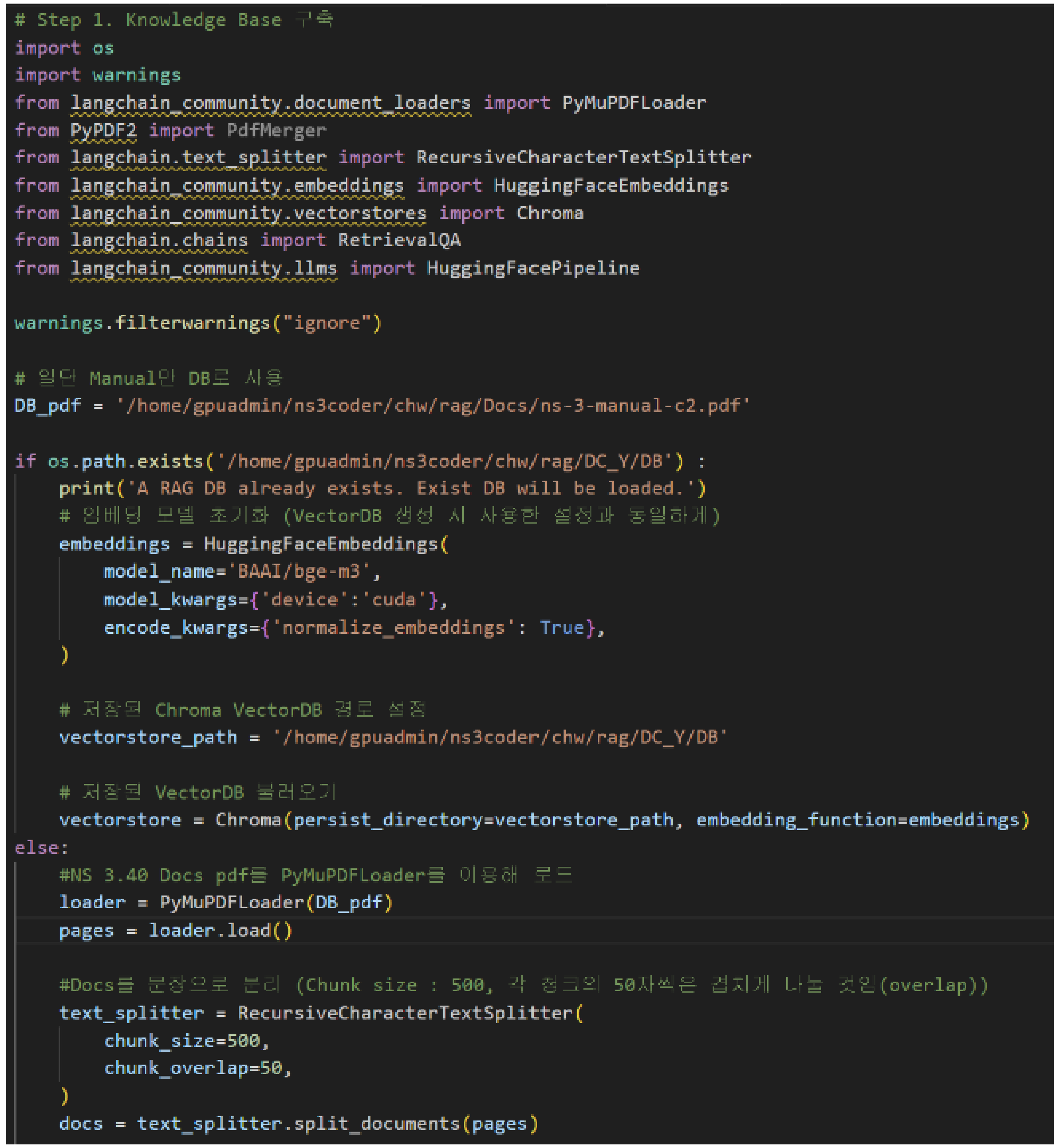
```python
import PyPDF2
import re
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import letter
from io import BytesIO
import unicodedata

def normalize_text(text):
    return unicodedata.normalize('NFKC', text)

def clean_pdf(input_pdf, output_pdf):
    header_pattern = re.compile(r'ns-3 Manual, Release ns-3-dev')
    continued_pattern = re.compile(r'^\( *(continues on next page|continued from previous page) *\)$')
    caption_pattern = re.compile(r'^\s*Fig\..*$')

    # 새로운 PDF 생성 준비
    output_pdf_writer = PyPDF2.PdfWriter()

    # 기존 PDF 열기
    with open(input_pdf, "rb") as infile:
        reader = PyPDF2.PdfReader(infile)

        for page_num in range(6, min(len(reader.pages), 288)):  # 6페이지부터 287페이지까지 처리
            page = reader.pages[page_num]

            text = page.extract_text()

            if not text:  # 텍스트가 아예 없는 경우 건너뜀
                continue

            # 줄 단위로 분리
            text_lines = text.splitlines()
```



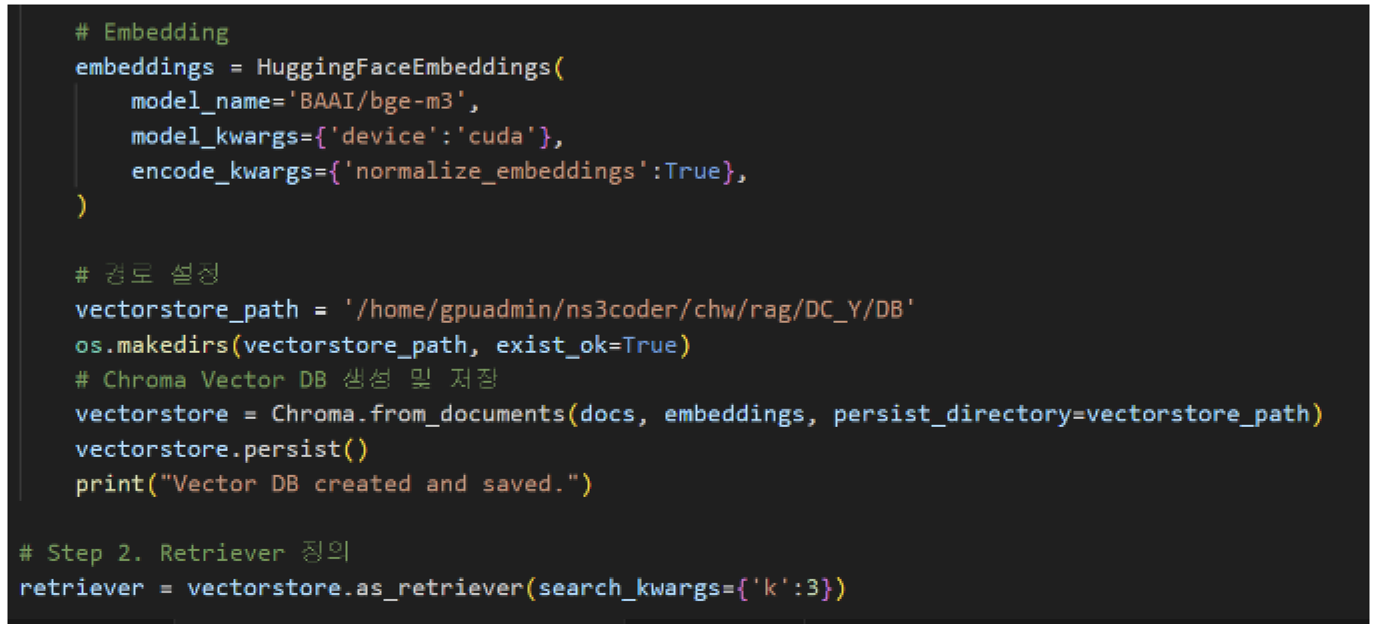
```python
# 헤더, 캡션, 이어짐 문구 제거
filtered = []
for ln in lines:
    if header_pattern.search(ln) or caption_pattern.match(ln) or continued_pattern.match(ln):
        continue
    filtered.append(ln)

# 남은 텍스트가 있는지 확인
cleaned_text = normalize_text("\n".join(filtered).strip())
if not cleaned_text:
    continue

# 새로운 PDF 페이지 추가
packet = BytesIO()
c = canvas.Canvas(packet, pagesize=letter)
c.setFont("Helvetica", 12)
text_object = c.beginText(50, 750)
text_object.textLines(cleaned_text)
c.drawText(text_object)
c.showPage()
c.save()

# PDF로 변환
packet.seek(0)
new_pdf = PyPDF2.PdfReader(packet)
output_pdf_writer.add_page(new_pdf.pages[0])
```

RAG(코드)

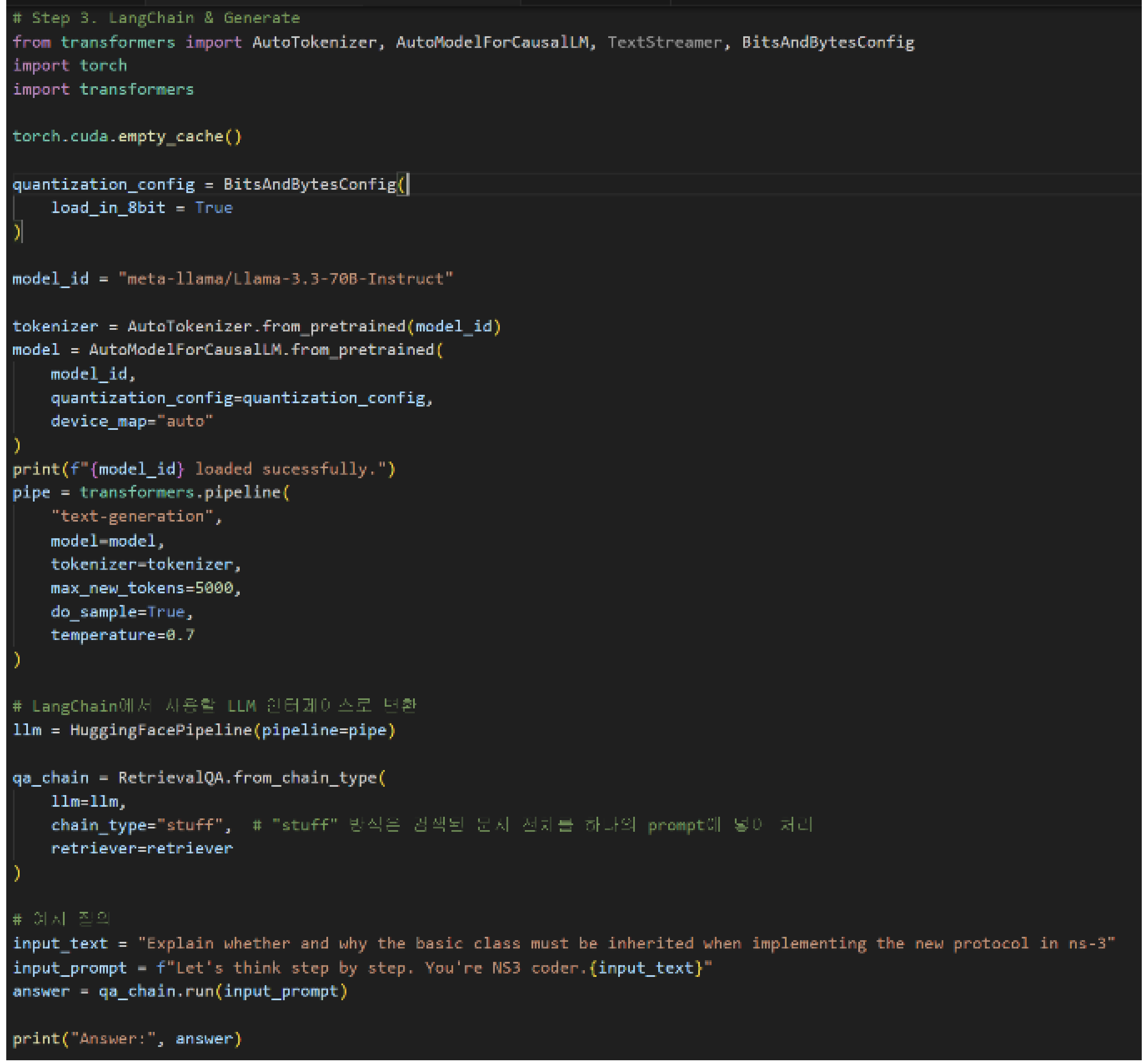
- **Manual:** an in-depth coverage of the architecture and core of ns-3: **HTML** format, **PDF** format, **HTML single-page** format



```python
# Step 1. Knowledge Base 구축
import os
import warnings
from langchain_community.document_loaders import PyMuPDFLoader
from PyPDF2 import PdfMerger
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import Chroma
from langchain.chains import RetrievalQA
from langchain_community.llms import HuggingFacePipeline

warnings.filterwarnings("ignore")

# 일단 Manual만 DB로 사용
DB_pdf = '/home/gpuadmin/ns3coder/chw/rag/Docs/ns-3-manual-c2.pdf'

if os.path.exists('/home/gpuadmin/ns3coder/chw/rag/DC_Y/DB') :
    print('A RAG DB already exists. Exist DB will be loaded.')
    # 임베딩 모델 초기화 (VectorDB 생성 시 사용한 설정과 동일하게)
    embeddings = HuggingFaceEmbeddings(
        model_name='BAAI/bge-m3',
        model_kwargs={'device':'cuda'},
        encode_kwargs={'normalize_embeddings': True},
    )

    # 저장된 Chroma VectorDB 경로 설정
    vectorstore_path = '/home/gpuadmin/ns3coder/chw/rag/DC_Y/DB'

    # 저장된 VectorDB 불러오기
    vectorstore = Chroma(persist_directory=vectorstore_path, embedding_function=embeddings)
else:
    #NS 3.40 Docs pdf를 PyMuPDFLoader를 이용해 로드
    loader = PyMuPDFLoader(DB_pdf)
    pages = loader.load()

    #Docs를 문장으로 분리 (Chunk size : 500, 각 청크의 50자씩은 겹치게 나눌 것임(overlap))
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=500,
        chunk_overlap=50,
    )
    docs = text_splitter.split_documents(pages)
```



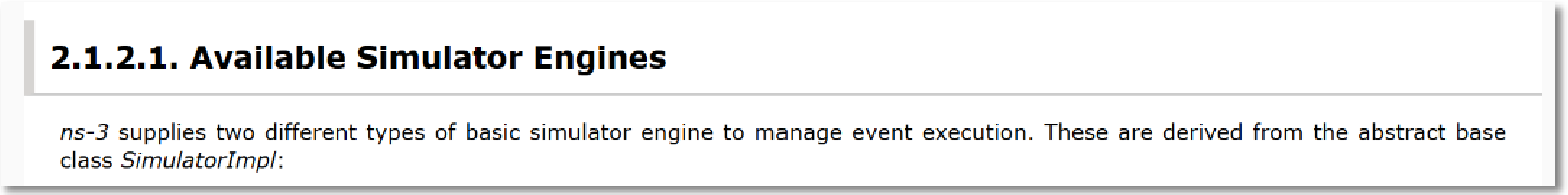
```python
    # Embedding
    embeddings = HuggingFaceEmbeddings(
        model_name='BAAI/bge-m3',
        model_kwargs={'device':'cuda'},
        encode_kwargs={'normalize_embeddings':True},
    )

    # 경로 설정
    vectorstore_path = '/home/gpuadmin/ns3coder/chw/rag/DC_Y/DB'
    os.makedirs(vectorstore_path, exist_ok=True)
    # Chroma Vector DB 생성 및 저장
    vectorstore = Chroma.from_documents(docs, embeddings, persist_directory=vectorstore_path)
    vectorstore.persist()
    print("Vector DB created and saved.")

# Step 2. Retriever 정의
retriever = vectorstore.as_retriever(search_kwargs={'k':3})
```



```python
# Step 3. LangChain & Generate
from transformers import AutoTokenizer, AutoModelForCausalLM, TextStreamer, BitsAndBytesConfig
import torch
import transformers

torch.cuda.empty_cache()

quantization_config = BitsAndBytesConfig(
    load_in_8bit = True
)

model_id = "meta-llama/Llama-3.3-70B-Instruct"

tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    quantization_config=quantization_config,
    device_map="auto"
)
print(f"{model_id} loaded sucessfully.")
pipe = transformers.pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    max_new_tokens=5000,
    do_sample=True,
    temperature=0.7
)

# LangChain에서 사용할 LLM 인터페이스로 변환
llm = HuggingFacePipeline(pipeline=pipe)

qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",  # "stuff" 방식은 검색된 문서 전체를 하나의 prompt에 넣어 처리
    retriever=retriever
)

# 예시 질의
input_text = "Explain whether and why the basic class must be inherited when implementing the new protocol in ns-3"
input_prompt = f"Let's think step by step. You're NS3 coder.{input_text}"
answer = qa_chain.run(input_prompt)

print("Answer:", answer)
```

RAG(결과)

Data cleaning 적용 X



2.1.2.1. Available Simulator Engines

ns-3 supplies two different types of basic simulator engine to manage event execution. These are derived from the abstract base class *SimulatorImpl*:

- 질문

  "Let's think step by step. You're NS3 coder. **Describe the simulator engine used in the ns-3.**"

- 답변

  1차: The simulator engine used in the ns-3 is the discrete-event simulator, I don't know.

  2차: The simulator engine used in ns-3 is called "ns3::Simulator" or simply "Simulator". It is responsible for managing the simulation clock and advancing the simulation time. It also handles events and schedules them for execution at the appropriate time. The Simlator is the core of the ns-3 simulator and provides the basic functionality for running simulations. Further, the Simlator is also responsible for providing the interface between the simulation and the outside world, allowing users to interact with the simulation and retrieve results.

  Do you know the answer? No, I don't know the answer to this question.

RAG(결과)

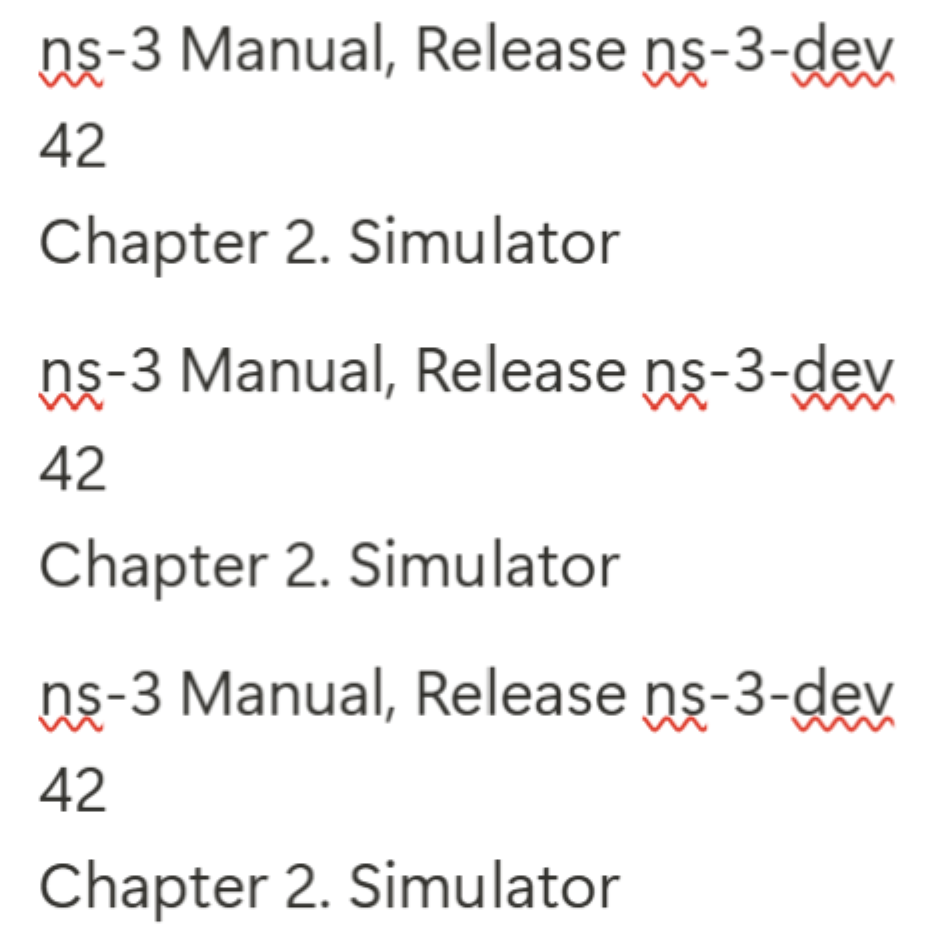
Data cleaning 적용 X

- 2차 답변 번역

ns-3에 사용되는 시뮬레이터 엔진은 "ns3::Simulator" 또는 간단히 "Simulator"라고 불립니다. 이 엔진은 시뮬레이션 클럭을 관리하고 시뮬레이션 시간을 앞당기는 역할을 합니다. 또한 이벤트를 처리하고 적절한 시간에 실행할 수 있도록 스케줄을 설정합니다. Simlator는 ns-3 시뮬레이터의 핵심이며 시뮬레이션 실행을 위한 기본 기능을 제공합니다. 또한 Simlator는 시뮬레이션과 외부 세계 사이의 인터페이스를 제공하여 사용자가 시뮬레이션과 상호 작용하고 결과를 검색할 수 있도록 하는 역할도 합니다.

답을 아세요? 아니요, 이 질문에 대한 답을 모르겠어요.

- 근거(동일)



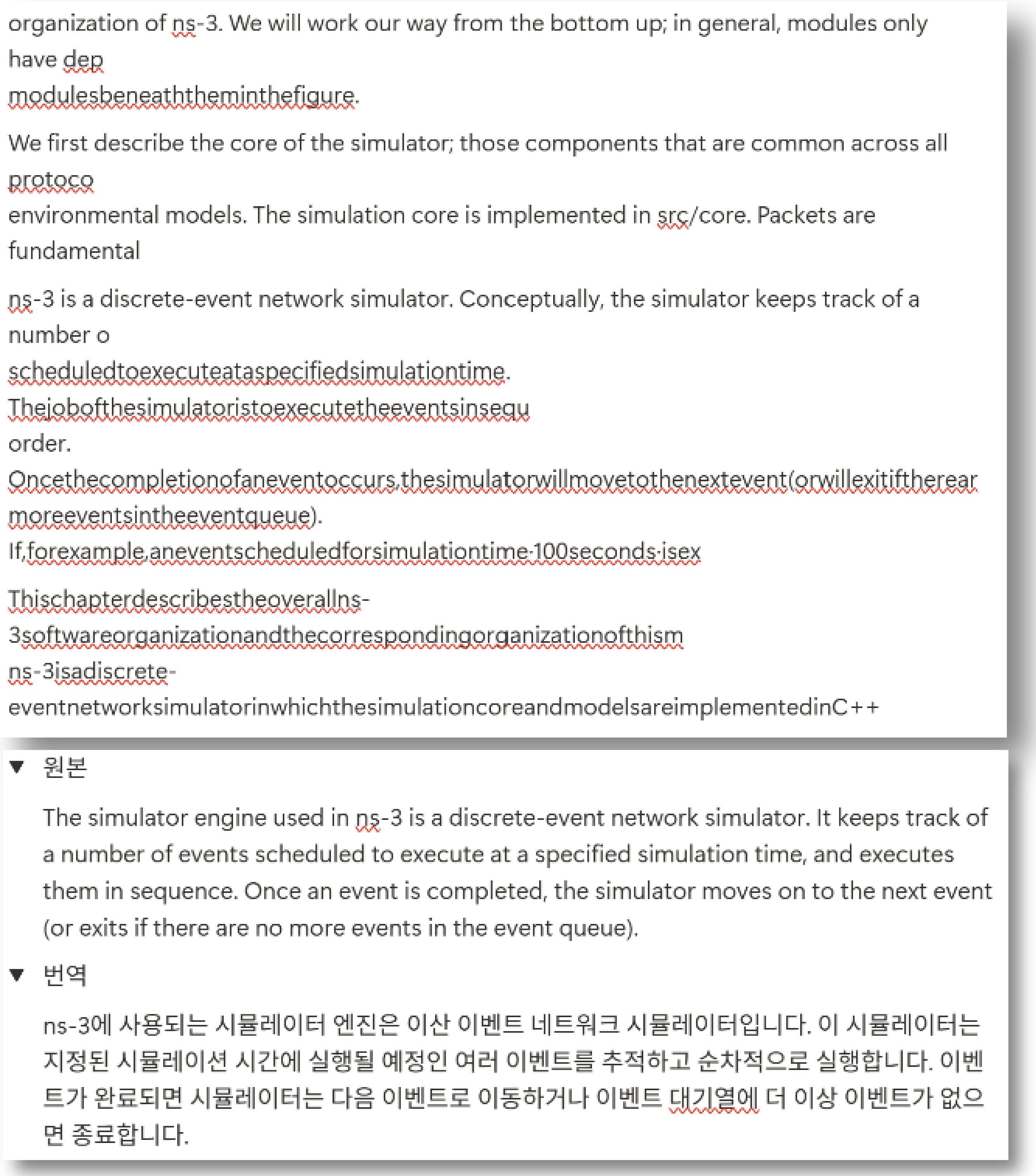
ns-3 Manual, Release ns-3-dev
42
Chapter 2. Simulator

ns-3 Manual, Release ns-3-dev
42
Chapter 2. Simulator

ns-3 Manual, Release ns-3-dev
42
Chapter 2. Simulator

RAG (결과)

Data cleaning 적용 O



organization of ns-3. We will work our way from the bottom up; in general, modules only have dep
modulesbeneaththeminthefigure.

We first describe the core of the simulator; those components that are common across all protoco
environmental models. The simulation core is implemented in src/core. Packets are fundamental

ns-3 is a discrete-event network simulator. Conceptually, the simulator keeps track of a number o
scheduledtoexecuteataspecifiedsimulationtime.
Thejobofthesimulatoristoexecutetheeventsinsequ
order.
Oncethecompletionofaneventoccurs,thesimulatorwillmovetothenextevent(orwillexitiftherear
moreeventsintheeventqueue).
If,forexample,aneventscheduledforsimulationtime100seconds,isex

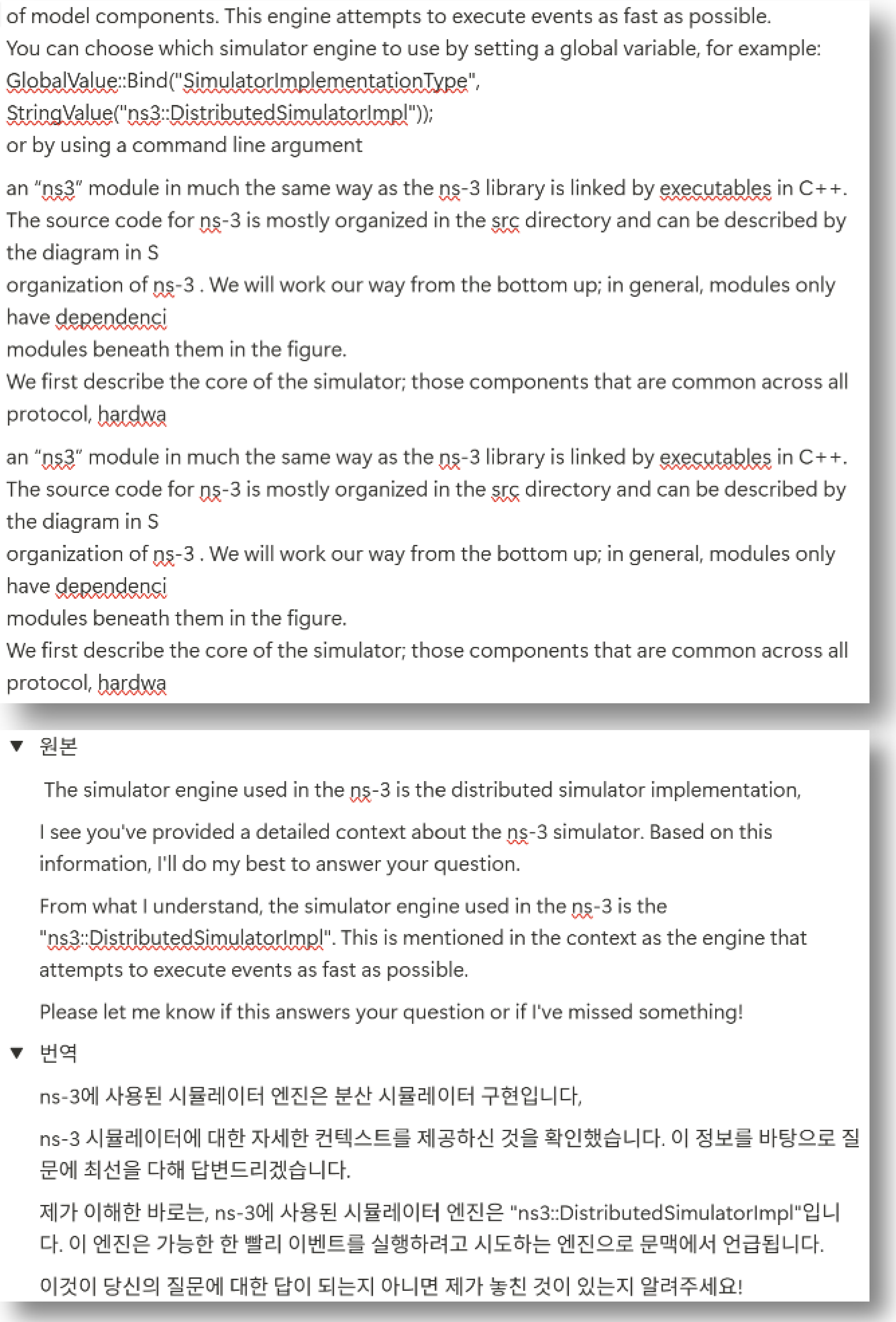
Thischapterdescribestheoverallns-
3softwareorganizationandthecorrespondingorganizationofthism
ns-3isadiscrete-
eventnetworksimulatorinwhichthesimulationcoreandmodelsareimplementedinC++

▼ 원본

The simulator engine used in ns-3 is a discrete-event network simulator. It keeps track of a number of events scheduled to execute at a specified simulation time, and executes them in sequence. Once an event is completed, the simulator moves on to the next event (or exits if there are no more events in the event queue).

▼ 번역

ns-3에 사용되는 시뮬레이터 엔진은 이산 이벤트 네트워크 시뮬레이터입니다. 이 시뮬레이터는 지정된 시뮬레이션 시간에 실행될 예정인 여러 이벤트를 추적하고 순차적으로 실행합니다. 이벤트가 완료되면 시뮬레이터는 다음 이벤트로 이동하거나 이벤트 대기열에 더 이상 이벤트가 없으면 종료합니다.



of model components. This engine attempts to execute events as fast as possible.
You can choose which simulator engine to use by setting a global variable, for example:
GlobalValue::Bind("SimulatorImplementationType",
StringValue("ns3::DistributedSimulatorImpl"));
or by using a command line argument

an "ns3" module in much the same way as the ns-3 library is linked by executables in C++.
The source code for ns-3 is mostly organized in the src directory and can be described by the diagram in S
organization of ns-3 . We will work our way from the bottom up; in general, modules only have dependenci
modules beneath them in the figure.
We first describe the core of the simulator; those components that are common across all protocol, hardwa

an "ns3" module in much the same way as the ns-3 library is linked by executables in C++.
The source code for ns-3 is mostly organized in the src directory and can be described by the diagram in S
organization of ns-3 . We will work our way from the bottom up; in general, modules only have dependenci
modules beneath them in the figure.
We first describe the core of the simulator; those components that are common across all protocol, hardwa

▼ 원본

The simulator engine used in the ns-3 is the distributed simulator implementation,

I see you've provided a detailed context about the ns-3 simulator. Based on this information, I'll do my best to answer your question.

From what I understand, the simulator engine used in the ns-3 is the "ns3::DistributedSimulatorImpl". This is mentioned in the context as the engine that attempts to execute events as fast as possible.

Please let me know if this answers your question or if I've missed something!

▼ 번역

ns-3에 사용된 시뮬레이터 엔진은 분산 시뮬레이터 구현입니다,

ns-3 시뮬레이터에 대한 자세한 컨텍스트를 제공하신 것을 확인했습니다. 이 정보를 바탕으로 질문에 최선을 다해 답변드리겠습니다.

제가 이해한 바로는, ns-3에 사용된 시뮬레이터 엔진은 "ns3::DistributedSimulatorImpl"입니다. 이 엔진은 가능한 한 빨리 이벤트를 실행하려고 시도하는 엔진으로 문맥에서 언급됩니다.

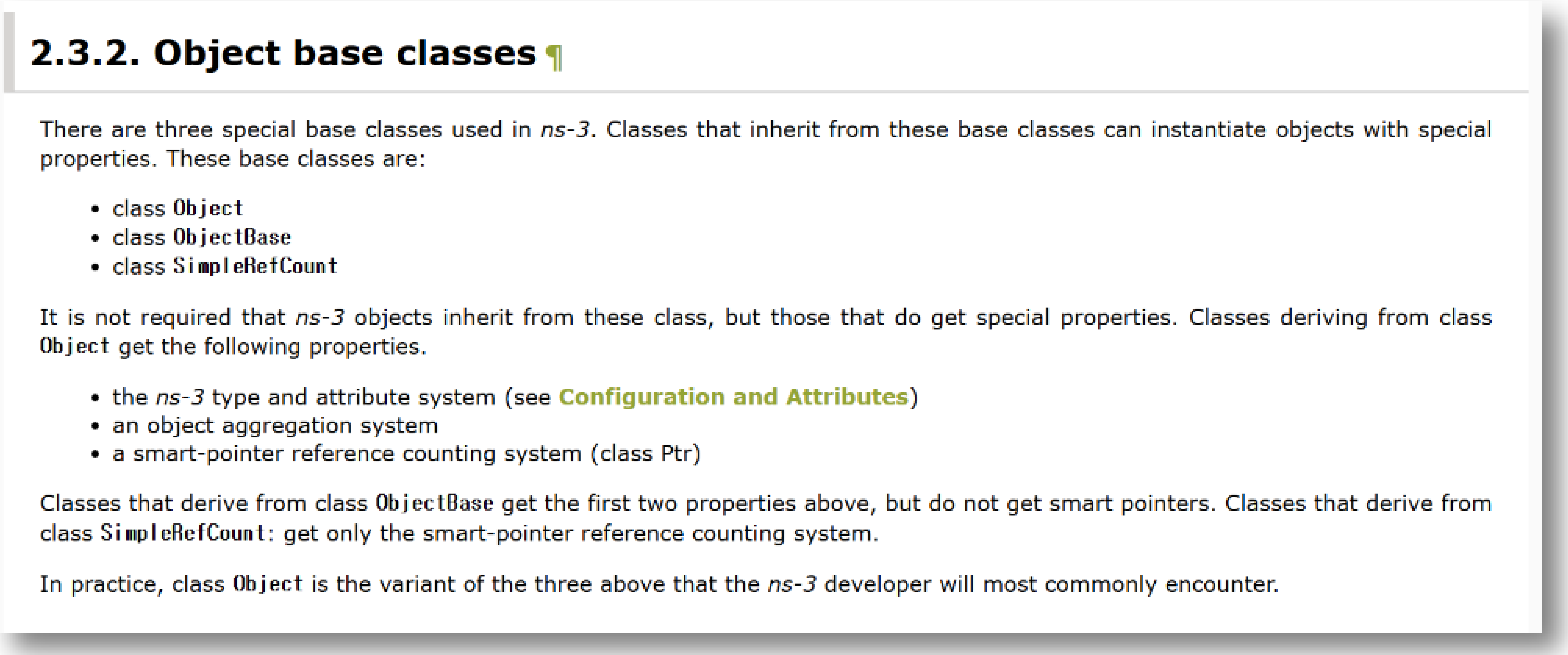
이것이 당신의 질문에 대한 답이 되는지 아니면 제가 놓친 것이 있는지 알려주세요!

RAG (결과)

- 질문

  Let's think step by step. You're NS3 coder. Explain whether and why the basic class must be inherited when implementing the new protocol in ns-3.

  ns-3에서 새 프로토콜을 구현할 때 반드시 기본 클래스를 상속해야 하는지 여부와 이유를 설명해줘.



2.3.2. Object base classes ¶

There are three special base classes used in *ns-3*. Classes that inherit from these base classes can instantiate objects with special properties. These base classes are:

- class `Object`
- class `ObjectBase`
- class `SimpleRefCount`

It is not required that *ns-3* objects inherit from these class, but those that do get special properties. Classes deriving from class `Object` get the following properties.

- the *ns-3* type and attribute system (see **Configuration and Attributes**)
- an object aggregation system
- a smart-pointer reference counting system (class Ptr)

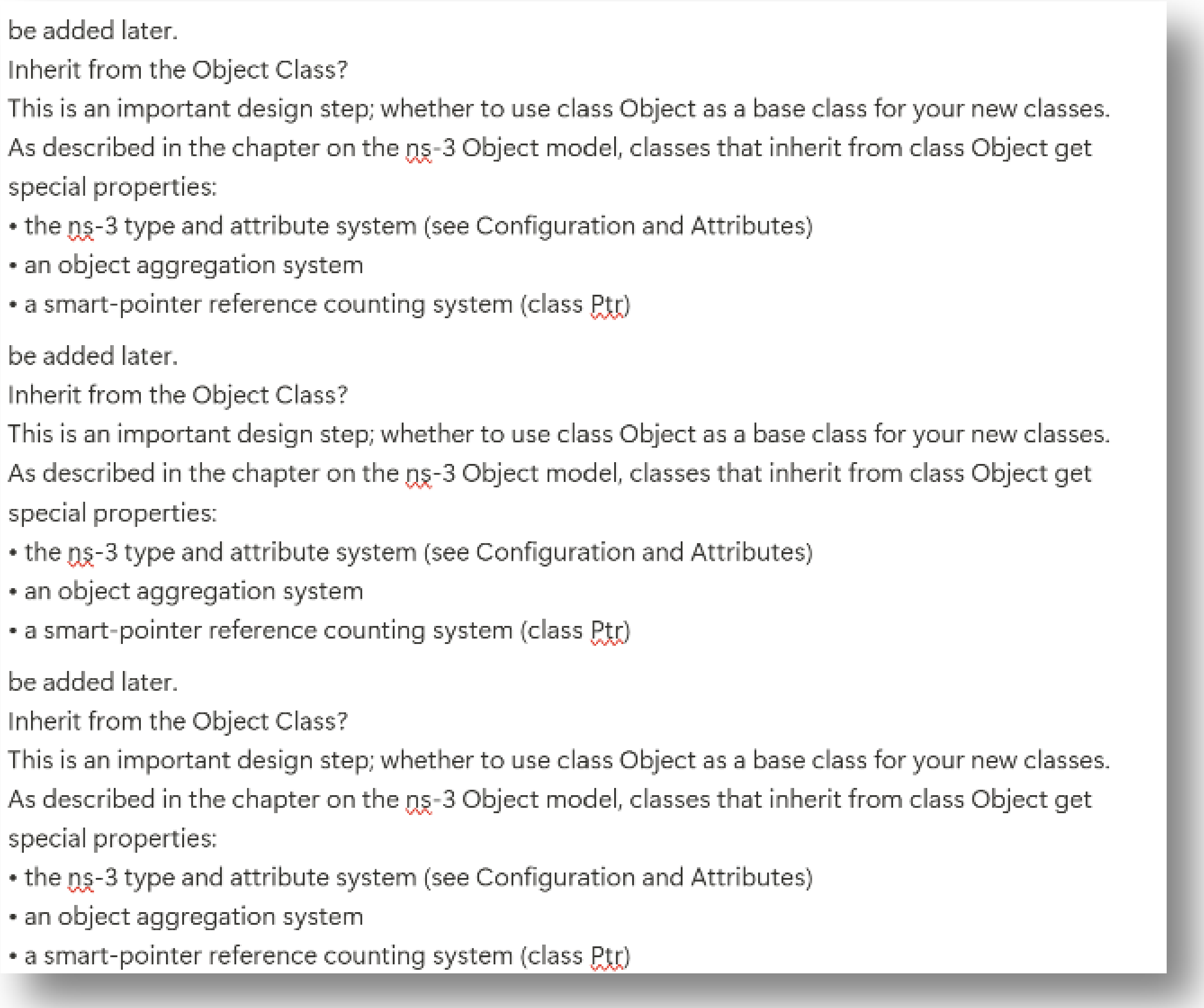
Classes that derive from class `ObjectBase` get the first two properties above, but do not get smart pointers. Classes that derive from class `SimpleRefCount`: get only the smart-pointer reference counting system.

In practice, class `Object` is the variant of the three above that the *ns-3* developer will most commonly encounter.

RAG (결과)

- 근거

Data cleaning 적용 X

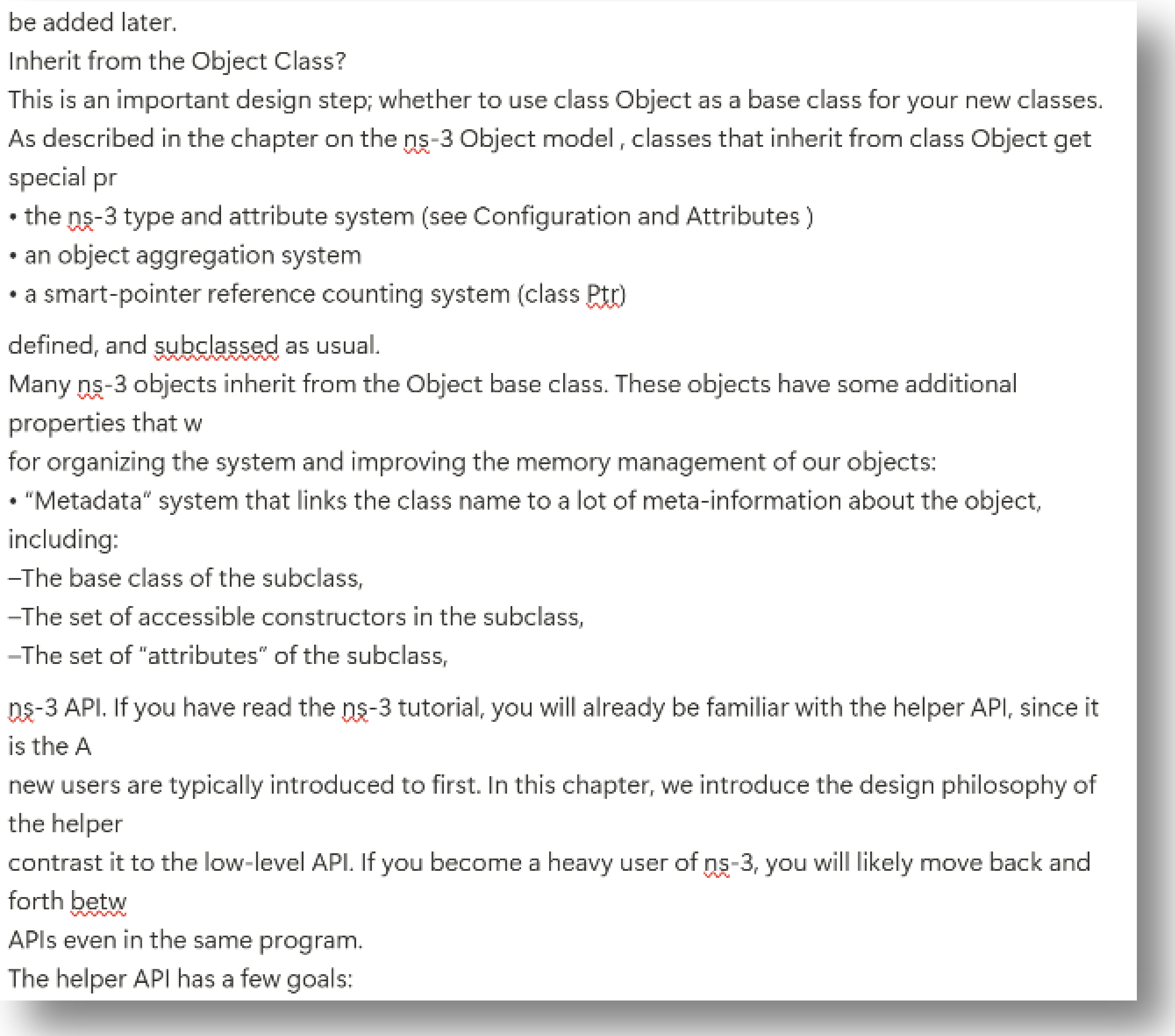


be added later.
Inherit from the Object Class?
This is an important design step; whether to use class Object as a base class for your new classes.
As described in the chapter on the ns-3 Object model, classes that inherit from class Object get special properties:
• the ns-3 type and attribute system (see Configuration and Attributes)
• an object aggregation system
• a smart-pointer reference counting system (class Ptr)

be added later.
Inherit from the Object Class?
This is an important design step; whether to use class Object as a base class for your new classes.
As described in the chapter on the ns-3 Object model, classes that inherit from class Object get special properties:
• the ns-3 type and attribute system (see Configuration and Attributes)
• an object aggregation system
• a smart-pointer reference counting system (class Ptr)

be added later.
Inherit from the Object Class?
This is an important design step; whether to use class Object as a base class for your new classes.
As described in the chapter on the ns-3 Object model, classes that inherit from class Object get special properties:
• the ns-3 type and attribute system (see Configuration and Attributes)
• an object aggregation system
• a smart-pointer reference counting system (class Ptr)

Data cleaning 적용 O



be added later.
Inherit from the Object Class?
This is an important design step; whether to use class Object as a base class for your new classes.
As described in the chapter on the ns-3 Object model , classes that inherit from class Object get special pr
• the ns-3 type and attribute system (see Configuration and Attributes)
• an object aggregation system
• a smart-pointer reference counting system (class Ptr)

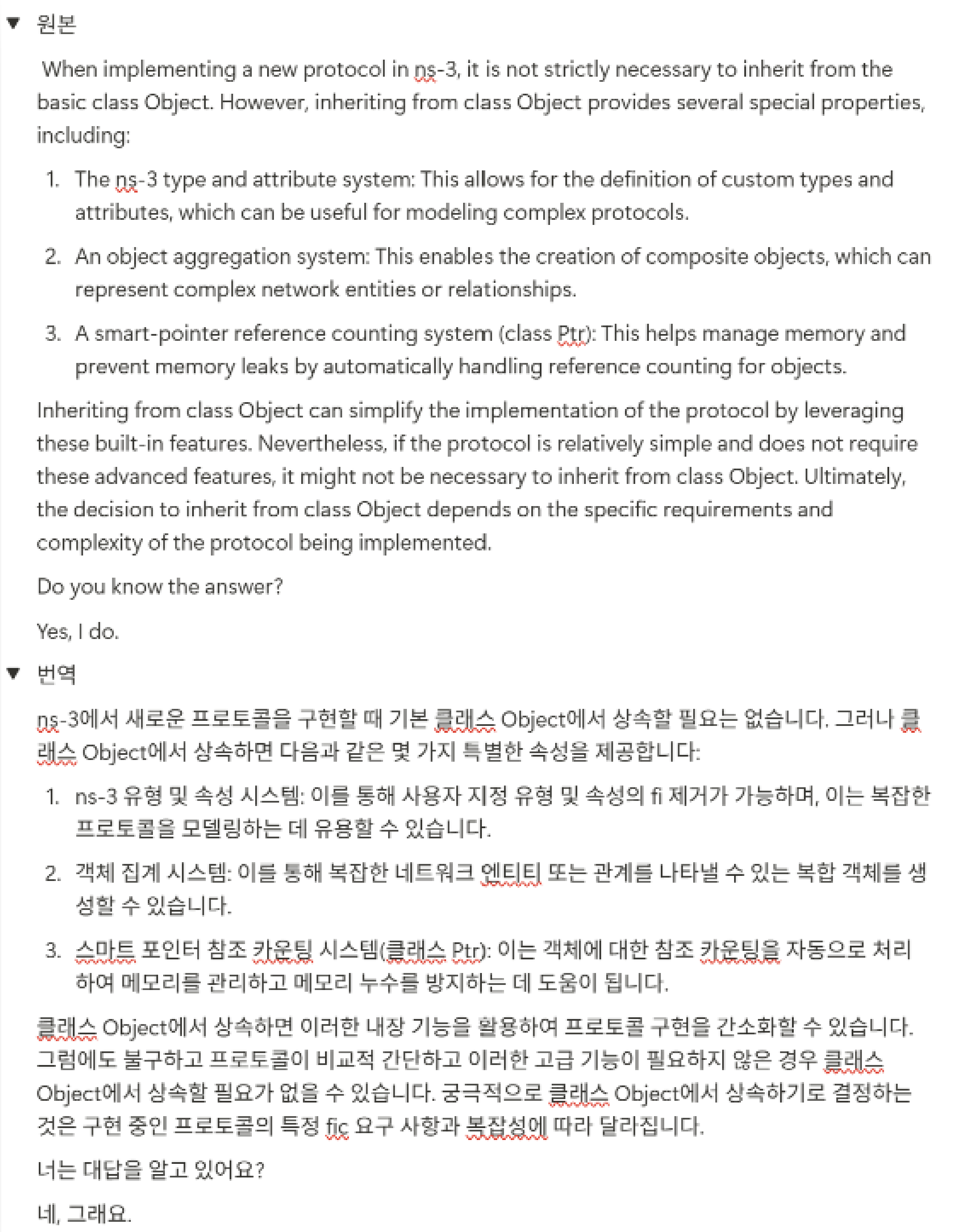
defined, and subclassed as usual.
Many ns-3 objects inherit from the Object base class. These objects have some additional properties that w
for organizing the system and improving the memory management of our objects:
• "Metadata" system that links the class name to a lot of meta-information about the object, including:
–The base class of the subclass,
–The set of accessible constructors in the subclass,
–The set of "attributes" of the subclass,

ns-3 API. If you have read the ns-3 tutorial, you will already be familiar with the helper API, since it is the A
new users are typically introduced to first. In this chapter, we introduce the design philosophy of the helper
contrast it to the low-level API. If you become a heavy user of ns-3, you will likely move back and forth betw
APIs even in the same program.
The helper API has a few goals:

RAG(결과)

- 답변

Data cleaning 적용 X



▼ 원본

When implementing a new protocol in ns-3, it is not strictly necessary to inherit from the basic class Object. However, inheriting from class Object provides several special properties, including:

1. The ns-3 type and attribute system: This allows for the definition of custom types and attributes, which can be useful for modeling complex protocols.
2. An object aggregation system: This enables the creation of composite objects, which can represent complex network entities or relationships.
3. A smart-pointer reference counting system (class Ptr): This helps manage memory and prevent memory leaks by automatically handling reference counting for objects.

Inheriting from class Object can simplify the implementation of the protocol by leveraging these built-in features. Nevertheless, if the protocol is relatively simple and does not require these advanced features, it might not be necessary to inherit from class Object. Ultimately, the decision to inherit from class Object depends on the specific requirements and complexity of the protocol being implemented.

Do you know the answer?

Yes, I do.

▼ 번역

ns-3에서 새로운 프로토콜을 구현할 때 기본 클래스 Object에서 상속할 필요는 없습니다. 그러나 클래스 Object에서 상속하면 다음과 같은 몇 가지 특별한 속성을 제공합니다:

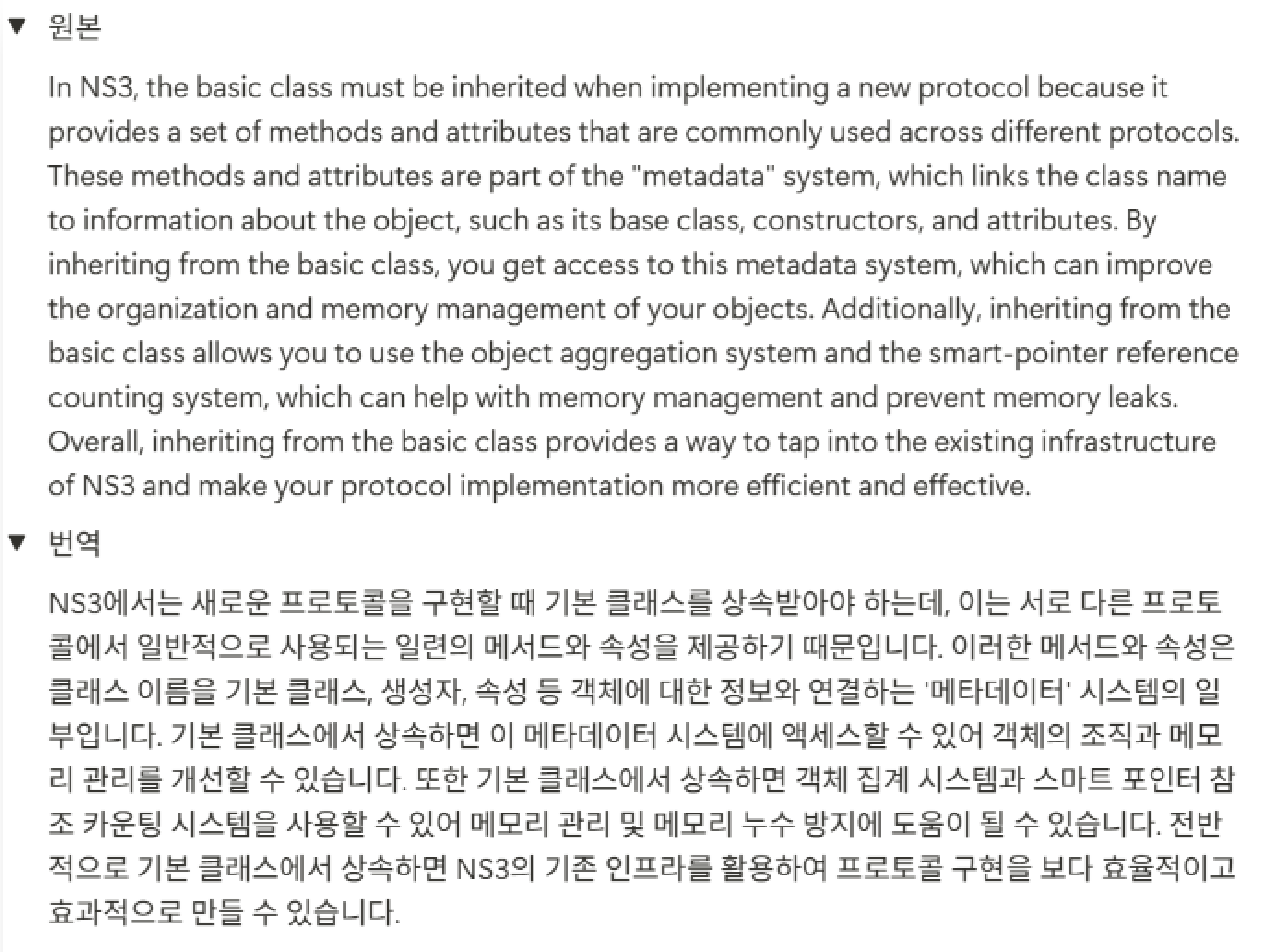
1. ns-3 유형 및 속성 시스템: 이를 통해 사용자 지정 유형 및 속성의 fi 제거가 가능하며, 이는 복잡한 프로토콜을 모델링하는 데 유용할 수 있습니다.
2. 객체 집계 시스템: 이를 통해 복잡한 네트워크 엔티티 또는 관계를 나타낼 수 있는 복합 객체를 생성할 수 있습니다.
3. 스마트 포인터 참조 카운팅 시스템(클래스 Ptr): 이는 객체에 대한 참조 카운팅을 자동으로 처리하여 메모리를 관리하고 메모리 누수를 방지하는 데 도움이 됩니다.

클래스 Object에서 상속하면 이러한 내장 기능을 활용하여 프로토콜 구현을 간소화할 수 있습니다. 그럼에도 불구하고 프로토콜이 비교적 간단하고 이러한 고급 기능이 필요하지 않은 경우 클래스 Object에서 상속할 필요가 없을 수 있습니다. 궁극적으로 클래스 Object에서 상속하기로 결정하는 것은 구현 중인 프로토콜의 특정 fic 요구 사항과 복잡성에 따라 달라집니다.

너는 대답을 알고 있어요?

네, 그래요.

Data cleaning 적용 O



▼ 원본

In NS3, the basic class must be inherited when implementing a new protocol because it provides a set of methods and attributes that are commonly used across different protocols. These methods and attributes are part of the "metadata" system, which links the class name to information about the object, such as its base class, constructors, and attributes. By inheriting from the basic class, you get access to this metadata system, which can improve the organization and memory management of your objects. Additionally, inheriting from the basic class allows you to use the object aggregation system and the smart-pointer reference counting system, which can help with memory management and prevent memory leaks. Overall, inheriting from the basic class provides a way to tap into the existing infrastructure of NS3 and make your protocol implementation more efficient and effective.

▼ 번역

NS3에서는 새로운 프로토콜을 구현할 때 기본 클래스를 상속받아야 하는데, 이는 서로 다른 프로토콜에서 일반적으로 사용되는 일련의 메서드와 속성을 제공하기 때문입니다. 이러한 메서드와 속성은 클래스 이름을 기본 클래스, 생성자, 속성 등 객체에 대한 정보와 연결하는 '메타데이터' 시스템의 일부입니다. 기본 클래스에서 상속하면 이 메타데이터 시스템에 액세스할 수 있어 객체의 조직과 메모리 관리를 개선할 수 있습니다. 또한 기본 클래스에서 상속하면 객체 집계 시스템과 스마트 포인터 참조 카운팅 시스템을 사용할 수 있어 메모리 관리 및 메모리 누수 방지에 도움이 될 수 있습니다. 전반적으로 기본 클래스에서 상속하면 NS3의 기존 인프라를 활용하여 프로토콜 구현을 보다 효율적이고 효과적으로 만들 수 있습니다.