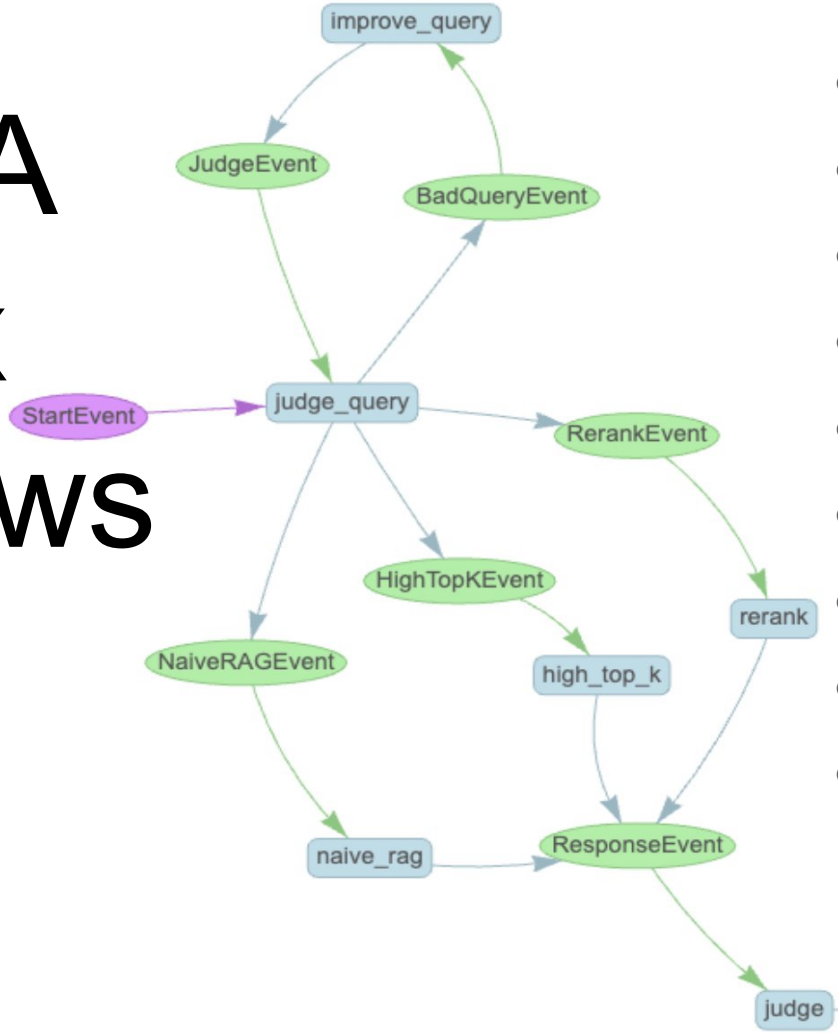
LLaMA Index Workflows



- What is Workflow?
- Why Workflow?
- Basic Workflow
- Branching and Loops
- Maintaining State
- Streaming events
- Concurrent Execution
- Nested Workflows
- Observability

What is Workflow?

A workflow is an **event-driven, step-based** way to control the execution flow of an application.

Your application is divided into sections called **Steps** which are triggered by **Events**, and themselves **emit Events** which trigger further steps.

By combining **steps and events**, you can create arbitrarily complex flows that encapsulate logic and make your application more maintainable and easier to understand.

A **step** can be anything from a single line of code to a complex agent. They can have arbitrary inputs and outputs, which are passed around by Events.

Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/

Why Workflow?

As generative AI applications become **more complex**, it becomes harder to manage the flow of data and control the execution of the application.

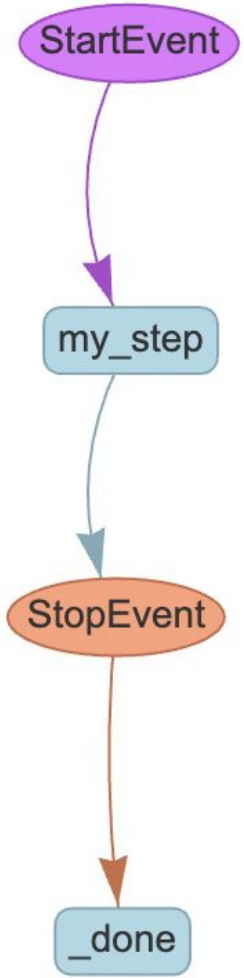
Workflows provide a way to manage this complexity by breaking the application into **smaller, more manageable pieces.**

For simple RAG pipelines and linear demos we do not expect you will need Workflows, but as your application grows in complexity, we advice for Workflow.

Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/

Basic Workflow

- Install Dependencies
- Required Imports
- Single Step Workflow
- Visualization Workflow



Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/basic_flow/

Basic Workflow

Install Dependencies

```
[1]  !pip install llama-index-core
     !pip install llama-index-utils-workflow
```

18s

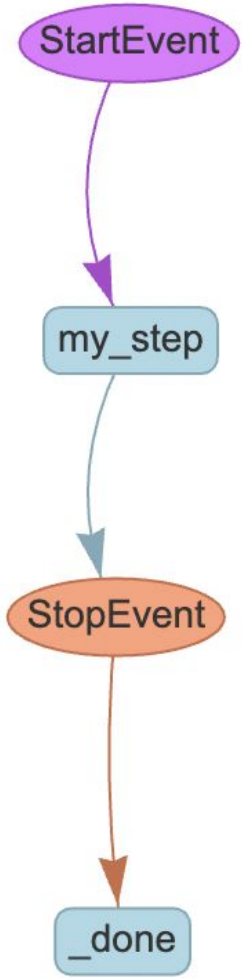
Show hidden output

Required Imports

```
[2]  from llama_index.core.workflow import (
         StartEvent,
         StopEvent,
         Workflow,
         step,
     )
```

12s

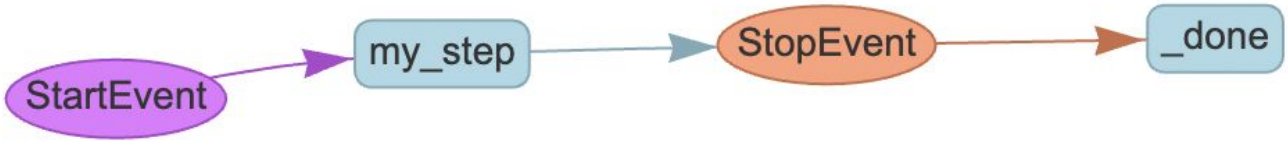
Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/basic_flow/



Basic Workflow

```python
class BasicWorkflow(Workflow):
    @step
    async def my_step(self, ev: StartEvent) -> StopEvent:
        # do something here
        return StopEvent(result="Hello, world!")


w = BasicWorkflow(timeout=10, verbose=False)
result = await w.run()
print(result)
```

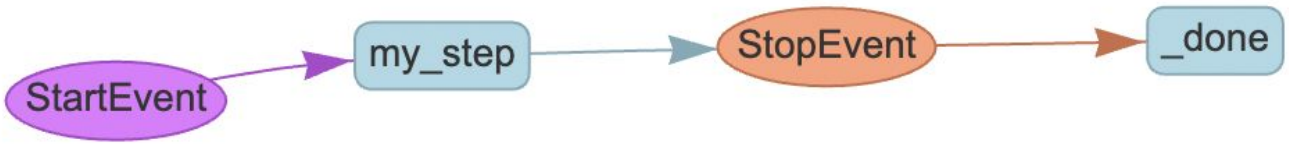
- Define a class **MyWorkflow** that inherits from **Wrokflow**
- Use the **@step** decorator to define a single step **my_step**
- The step takes a single argument, **ev,** which is an instance of **StartEvent**
- The step returns a **StopEvent** with a result of "Hello, world!"
- We create an instance of **MyWorkflow** with a timeout of 10 seconds and verbosity off
- We run the workflow and print the result



Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/basic_flow/

Basic Workflow: **Visualization**

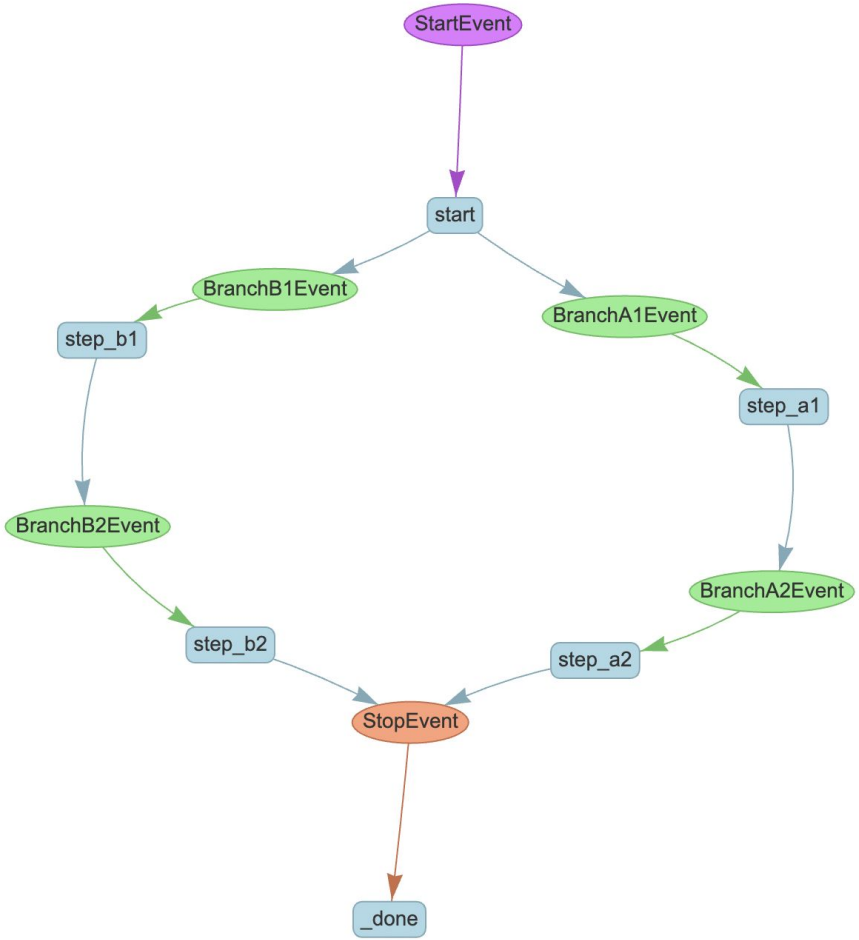
```
from llama_index.utils.workflow import draw_all_possible_flows

draw_all_possible_flows(BasicWorkflow, filename="basic_workflow1.html")
```



Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/basic_flow/

Branches in Workflow

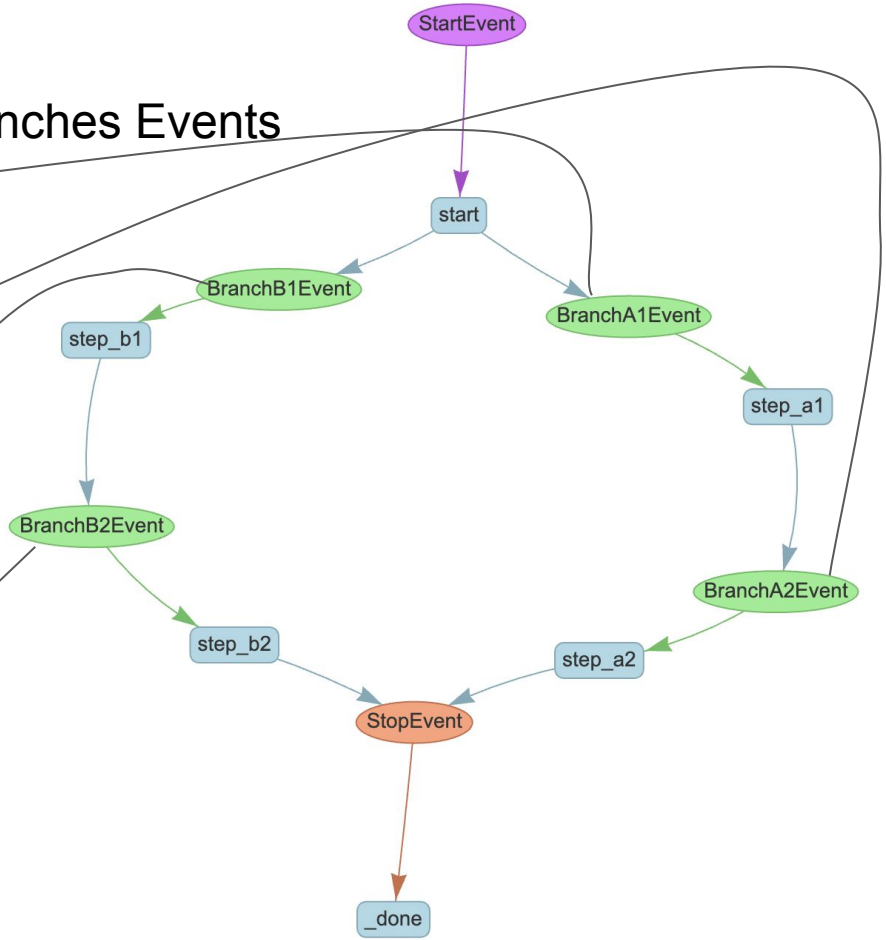
- Define Branches Events
- Define Branch Workflow
- Run Branch Workflow



Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/branches_and_loops/

Branches in Workflow: Define Branches Events

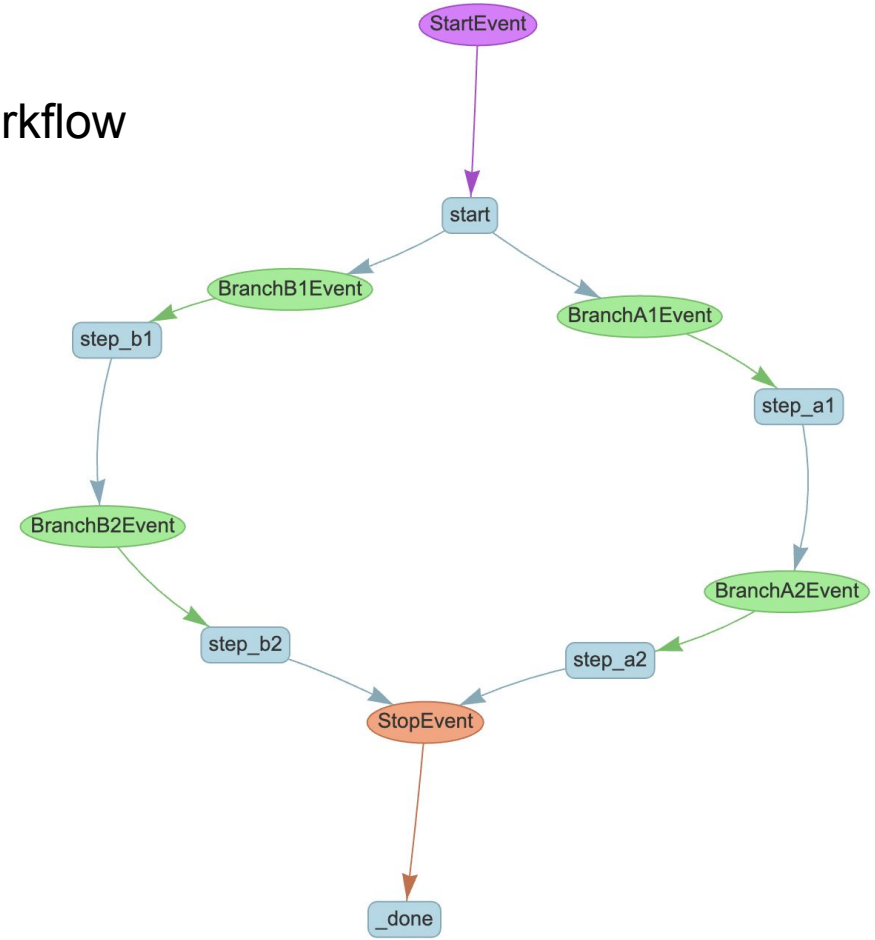
```python
class BranchA1Event(Event):
    payload: str


class BranchA2Event(Event):
    payload: str


class BranchB1Event(Event):
    payload: str


class BranchB2Event(Event):
    payload: str
```



Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/branches_and_loops/

Branches in Workflow: Define Workflow

```python
class BranchWorkflow(Workflow):
    @step
    async def start(self, ev: StartEvent) -> BranchA1Event | BranchB1Event:
        if random.randint(0, 1) == 0:
            print("Go to branch A")
            return BranchA1Event(payload="Branch A")
        else:
            print("Go to branch B")
            return BranchB1Event(payload="Branch B")

    @step
    async def step_a1(self, ev: BranchA1Event) -> BranchA2Event:
        print(ev.payload)
        return BranchA2Event(payload=ev.payload)

    @step
    async def step_b1(self, ev: BranchB1Event) -> BranchB2Event:
        print(ev.payload)
        return BranchB2Event(payload=ev.payload)

    @step
    async def step_a2(self, ev: BranchA2Event) -> StopEvent:
        print(ev.payload)
        return StopEvent(result="Branch A complete.")

    @step
    async def step_b2(self, ev: BranchB2Event) -> StopEvent:
        print(ev.payload)
        return StopEvent(result="Branch B complete.")
```



Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/branches_and_loops/

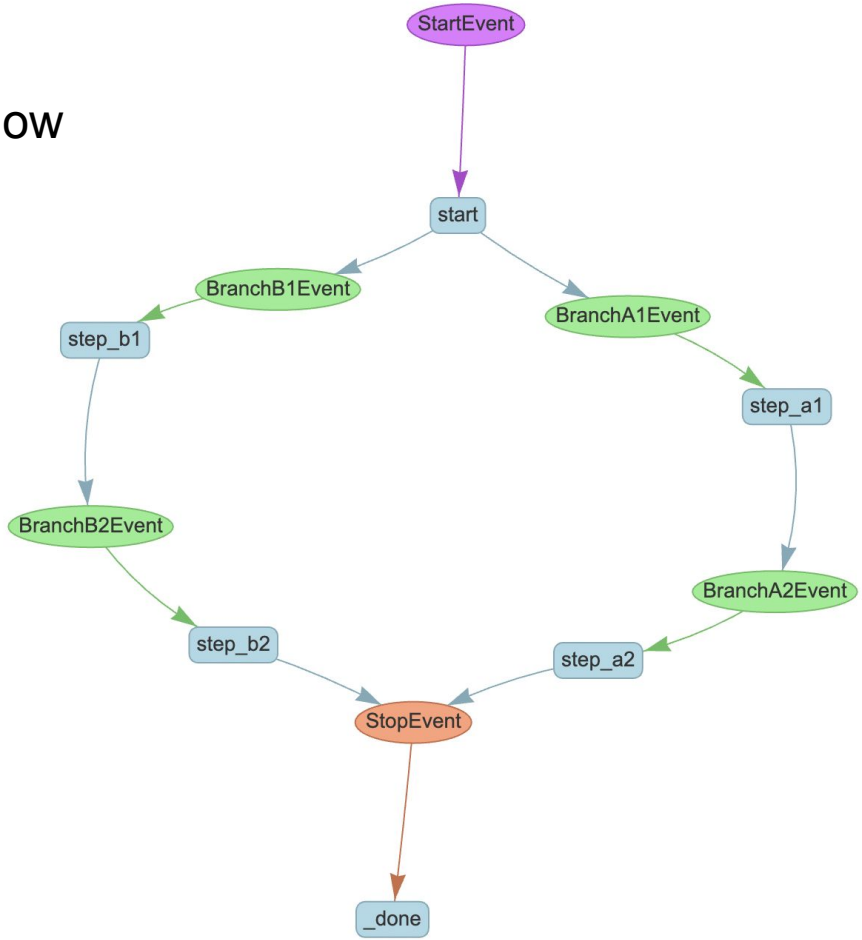
Branches in Workflow: Run Workflow

```
w = BranchWorkflow(timeout=10, verbose=False)
result = await w.run()
print(result)
```

```
Go to branch A
Branch A
Branch A
Branch A complete.
```

```
w = BranchWorkflow(timeout=10, verbose=False)
result = await w.run()
print(result)
```

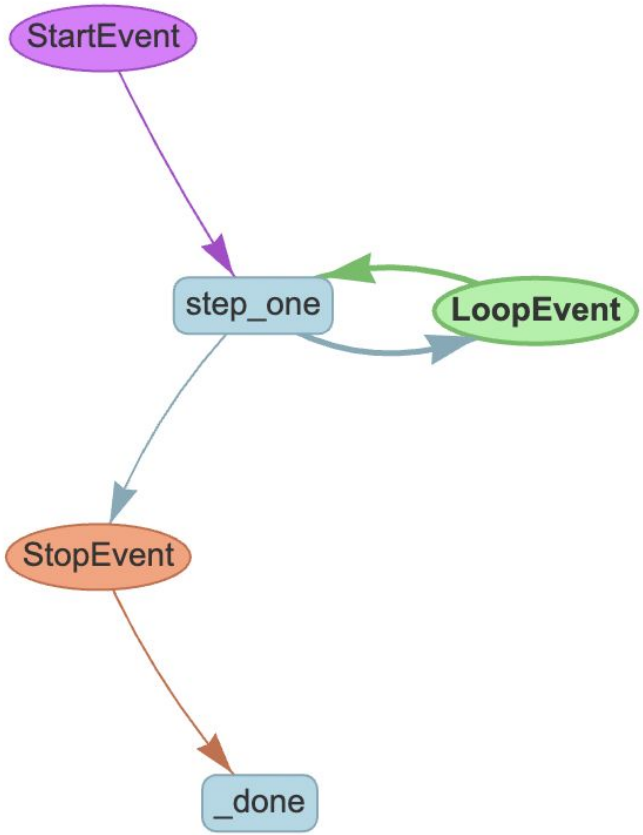
```
Go to branch B
Branch B
Branch B
Branch B complete.
```



Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/branches_and_loops/

Loops in Workflow

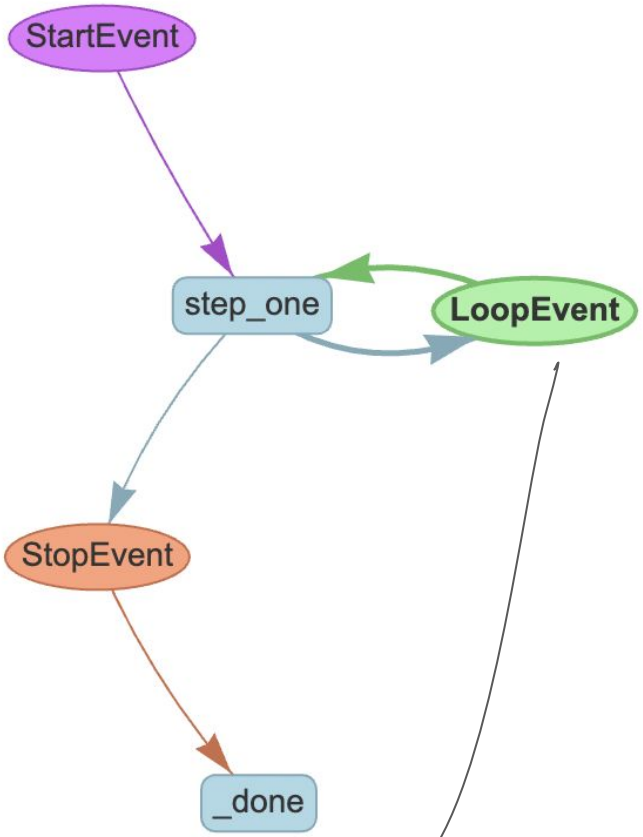
- Define Loop Events
- Define Loop Wrokflow
- Run Loop Wrokflow



Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/branches_and_loops/

Loops in Workflow: Define Loop Event

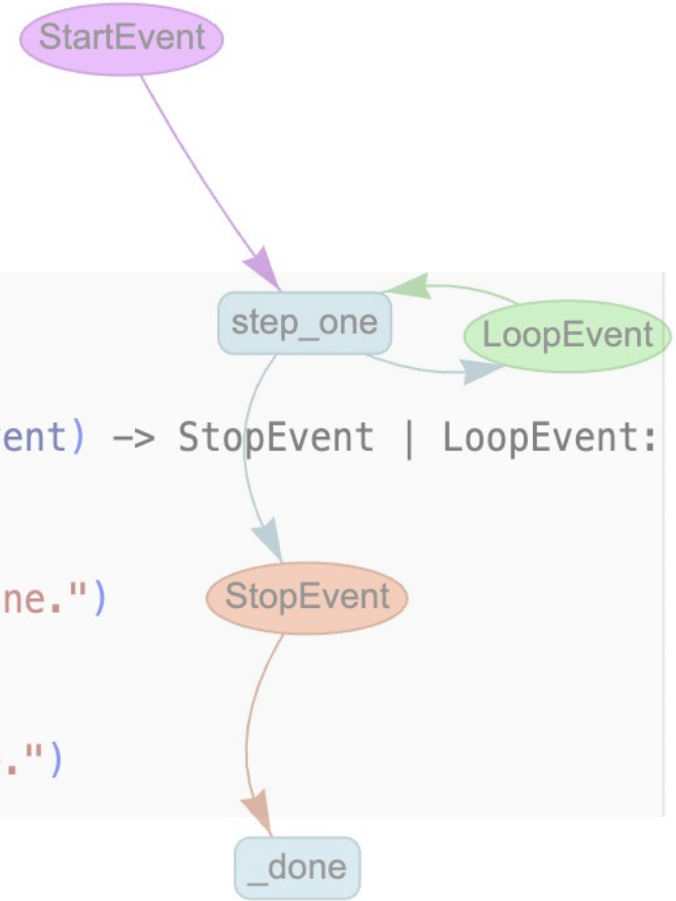
```python
from llama_index.core.workflow import (
    StartEvent,
    StopEvent,
    Workflow,
    step,
    Event
)
import random

class LoopEvent(Event):
    loop_output: str
```



Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/branches_and_loops/

Loops in Workflow: Define Loop Workflow

```python
class LoopWorkflow(Workflow):
  @step
  async def step_one(self, ev: StartEvent | LoopEvent) -> StopEvent | LoopEvent:
    if random.randint(0, 1) == 0:
      print("Bad thing happened")
      return LoopEvent(loop_output="Back to step one.")
    else:
      print("Good thing happened")
      return StopEvent(result="Stop Event complete.")
```

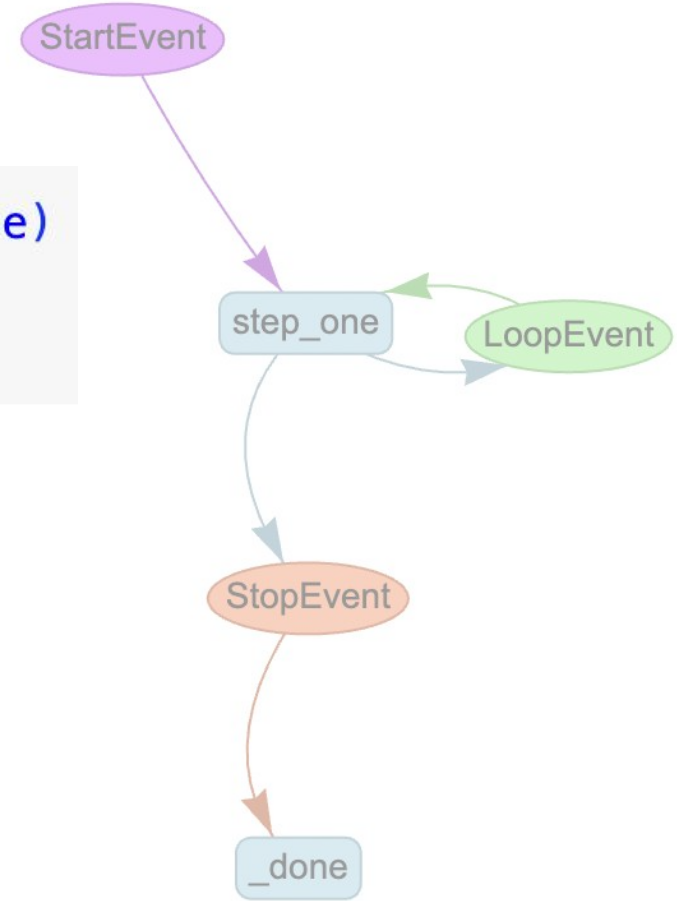


Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/branches_and_loops/

Loops in Workflow: Define Loop Workflow

```
w = LoopWorkflow(timeout=10, verbose=False)
result = await w.run()
print(result)
```

```
Bad thing happened
Bad thing happened
Bad thing happened
Bad thing happened
Good thing happened
Stop Event complete.
```

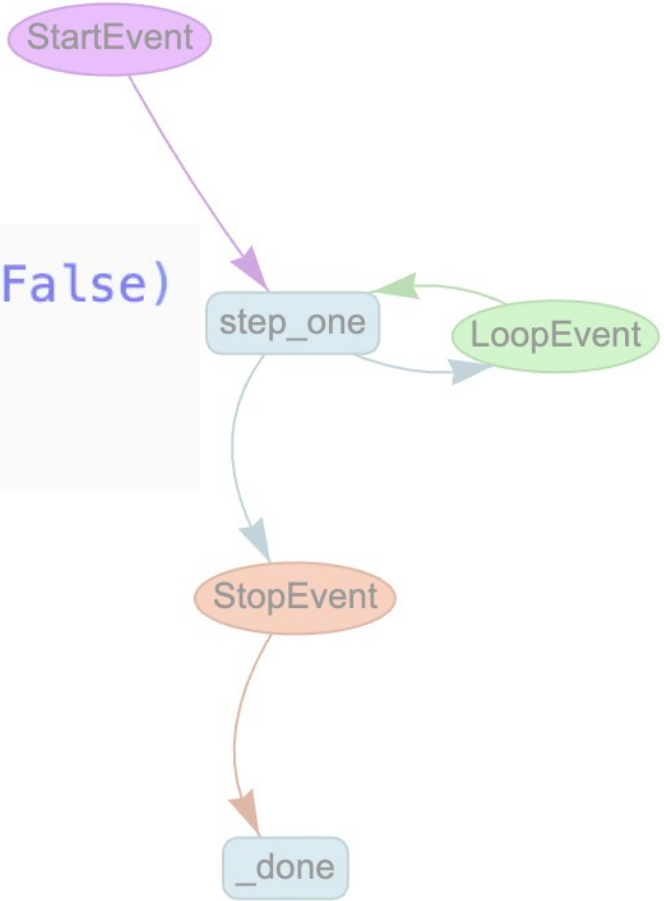


Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/branches_and_loops/

Loops in Workflow: Define Loop Workflow

```
w = LoopWorkflow(timeout=10, verbose=False)
result = await w.run()
print(result)
```

```
Bad thing happened
Good thing happened
Stop Event complete.
```



Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/branches_and_loops/

Maintaining State

```python
@step
async def start(
    self, ctx: Context, ev: StartEvent
) -> SetupEvent | StepTwoEvent:
    db = await ctx.get("some_database", default=None)
    print("DB is ", db)
    if db is None:
        print("Need to load data")
        return SetupEvent(query=ev.query)

    # do something with the query
    print("Start is called --------")
    return StepTwoEvent(query=ev.query)

@step
async def setup(self, ctx: Context, ev: SetupEvent) -> StartEvent:
    # load data
    await ctx.set("some_database", [1, 2, 3])
    query = ev.query
    print("query ------ ", query)
    return StartEvent(query=ev.query)
```

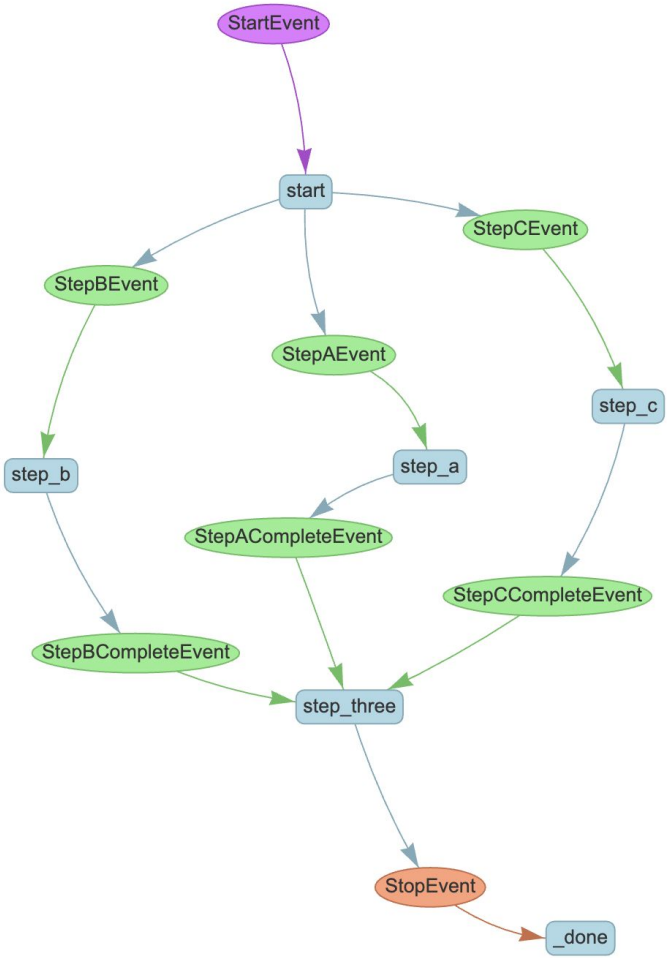
Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/state/

Streaming Events

```python
class MyWorkflow(Workflow):
    @step
    async def step_one(self, ctx: Context, ev: StartEvent) -> FirstEvent:
        ctx.write_event_to_stream(ProgressEvent(msg="Step one is happening"))
        return FirstEvent(first_output="First step complete.")
```

Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/stream/

Concurrent Execution



Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/concurrent_execution/

Concurrent Execution

```python
class ConcurrentFlow1(Workflow):
    @step
    async def start(
        self, ctx: Context, ev: StartEvent
    ) -> StepAEvent | StepBEvent | StepCEvent:
        ctx.send_event(StepAEvent(query="Query 1"))
        ctx.send_event(StepBEvent(query="Query 2"))
        ctx.send_event(StepCEvent(query="Query 3"))
```

Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/concurrent_execution/

Concurrent Execution

```python
@step
async def step_a(self, ctx: Context, ev: StepAEvent) -> StepACompleteEvent:
    print("Doing something A-ish")
    return StepACompleteEvent(result=ev.query)

@step
async def step_b(self, ctx: Context, ev: StepBEvent) -> StepBCompleteEvent:
    print("Doing something B-ish")
    return StepBCompleteEvent(result=ev.query)

@step
async def step_c(self, ctx: Context, ev: StepCEvent) -> StepCCompleteEvent:
    print("Doing something C-ish")
    return StepCCompleteEvent(result=ev.query)
```

Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/concurrent_execution/

Concurrent Execution

```python
@step
async def step_three(
    self,
    ctx: Context,
    ev: StepACompleteEvent | StepBCompleteEvent | StepCCompleteEvent,
) -> StopEvent:
    print("Received event ", ev.result)

    # wait until we receive 3 events
    if (
        ctx.collect_events(
            ev,
            [StepCCompleteEvent, StepACompleteEvent, StepBCompleteEvent],
        )
        is None
    ):
        return None

    # do something with all 3 results together
    return StopEvent(result="Done")
```

Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/concurrent_execution/

Concurrent Execution

```python
@step
async def step_three(
    self,
    ctx: Context,
    ev: StepACompleteEvent | StepBCompleteEvent | StepCCompleteEvent,
) -> StopEvent:
    print("Received event ", ev.result)

    # wait until we receive 3 events
    if (
        ctx.collect_events(
            ev,
            [StepCCompleteEvent, StepACompleteEvent, StepBCompleteEvent],
        )
        is None
    ):
        return None

    # do something with all 3 results together
    return StopEvent(result="Done")
```

Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/concurrent_execution/

Subclassing Workflow

```python
class MainWorkflow(Workflow):
    @step
    async def start(self, ev: StartEvent) -> Step2Event:
        print("Starting up")
        return Step2Event(query=ev.query)

    @step
    async def step_two(self, ev: Step2Event) -> Step3Event:
        print("Sending an email")
        return Step3Event(query=ev.query)

    @step
    async def step_three(self, ev: Step3Event) -> StopEvent:
        print("Finishing up")
        return StopEvent(result=ev.query)
```

You have an agentic workflow that does some processing and then sends an email.

You can subclass the workflow to add an extra step to send a text message as well.

Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/subclass/

Subclassing Workflow

```
class CustomWorkflow(MainWorkflow):
    @step
    async def step_two(self, ev: Step2Event) -> Step2BEvent:
        print("Sending an email")
        return Step2BEvent(query=ev.query)

    @step
    async def step_two_b(self, ev: Step2BEvent) -> Step3Event:
        print("Also sending a text message")
        return Step3Event(query=ev.query)
```

Reference:
Documentation: https://docs.llamaindex.ai/en/stable/understanding/workflows/subclass/