iWay Software
How-to's

Configuring the Neo4j Connector

Neo4j® is a highly scalable native graph database, built to leverage not only data, but also data relationships. The Neo4j Connector uses the Neo4j HTTP API to provide the capability to execute Cypher® (Neo4j graph query language) statements and run basic operations, such as create, update, delete, merge, and select nodes, or relationships against a Neo4j graph database instance.

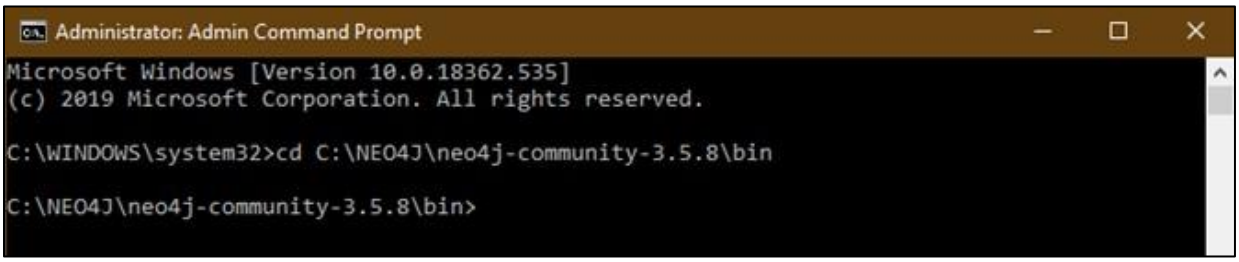
This how-to includes the following topics:

Installing the Neo4j Server on Windows Platforms

1. Download a Neo4j Server from *https://neo4j.com/download-center/*.

2. Right-click the downloaded ZIP file and extract all files to a directory (for example, C:\NEO4J).

3. Using the Command Prompt, change the directory to the bin directory of the extracted directory, as shown in the following image.



4. Run the Neo4j Server as a Windows console application or Windows service.

   Windows console application. To run the Neo4j Server as a Windows console application, you can type the *neo4j console* command to start the server, as shown in the following image.

```
Administrator: Admin Command Prompt - neo4j console
Microsoft Windows [Version 10.0.18362.535]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd C:\NEO4J\neo4j-community-3.5.8\bin

C:\NEO4J\neo4j-community-3.5.8\bin>neo4j console
2019-12-20 21:26:12.160+0000 INFO  ======== Neo4j 3.5.8 ========
2019-12-20 21:26:12.175+0000 INFO  Starting...
2019-12-20 21:26:15.082+0000 INFO  Bolt enabled on 127.0.0.1:7687.
2019-12-20 21:26:16.285+0000 INFO  Started.
2019-12-20 21:26:16.973+0000 INFO  Remote interface available at http://localhost:7474/
```

   Note: You can Ctrl-C to stop the server.

   Windows service. To run the Neo4j Server as a Windows service, the available commands for Neo4j are help, start, stop, restart, status, install-service, uninstall-service, and update-service. You can install the service with the *neo4j install-service* command, and start the service with the *neo4j start* command, as shown in the following image.

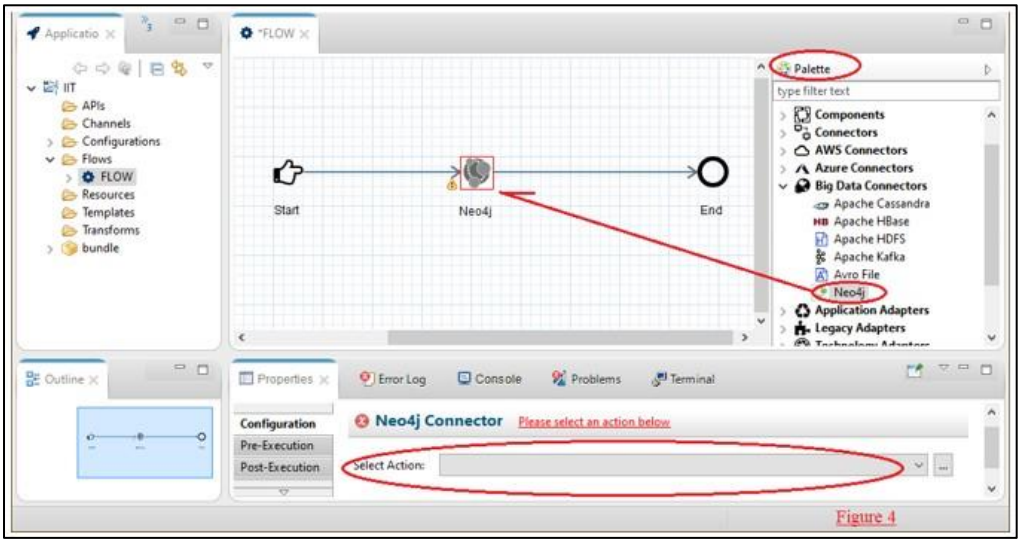
```
Administrator: Admin Command Prompt
Microsoft Windows [Version 10.0.18362.535]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd C:\NEO4J\neo4j-community-3.5.8\bin

C:\NEO4J\neo4j-community-3.5.8\bin>neo4j install-service
Neo4j service installed

C:\NEO4J\neo4j-community-3.5.8\bin>neo4j start
Neo4j service started

C:\NEO4J\neo4j-community-3.5.8\bin>_
```

Adding the Neo4j Connector to a Process Flow

After the Neo4j Server starts, follow the steps shown below to use iWay Integration Tool (iIT) to configure an object of the Neo4j Connector in a process flow and run the process flow to query the Neo4j database.

1. Launch iWay Integration Tool (iIT).
2. Create a new Application Project.
3. Create a process flow.

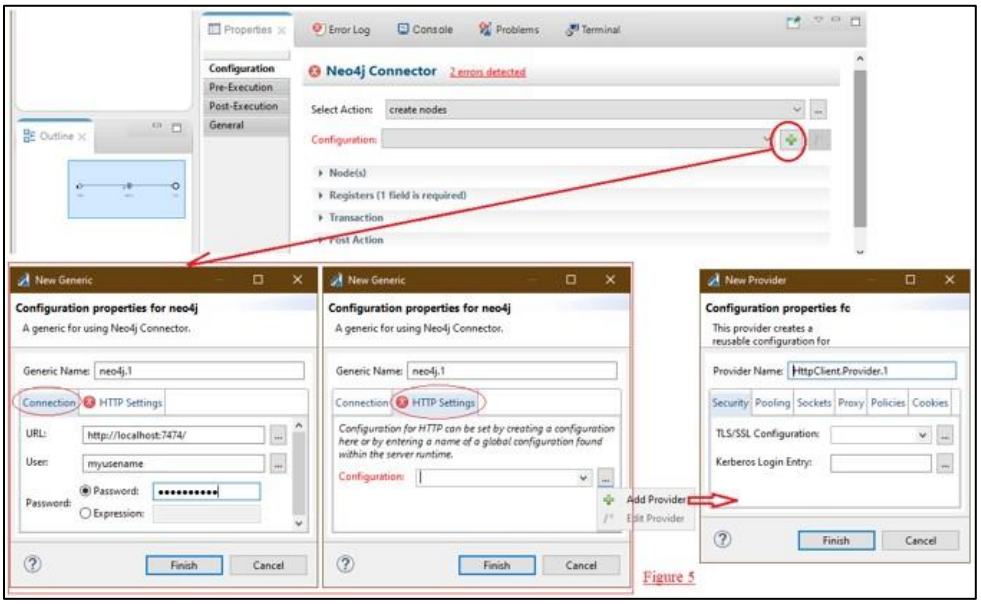
4. From the Palette, under *Big Data Connectors*, drag and drop *Neo4j* to the process flow, as shown in the following image.



5. Configure the object of the Neo4j Connector.
6. Run the created process flow.

Creating a New Neo4j Configuration

1. From the *Select Action* drop-down list, select an action which you want to execute.
2. From the *Configuration* drop-down list, select an existing configuration or click the plus (*+*) button to create a new configuration. A configuration contains Neo4j connection information and HTTP/HTTPS provider settings.
3. Fill out all required fields, as shown in the following image.



Configuration Properties

The following are configuration properties for using the Neo4j Connector.

URL. Remote interface for accessing the Neo4j Server.

User. User name for logging into the Neo4j Server.

Password. Password for the user logging into the Neo4j Server.

Configuration. Provider which creates a reusable configuration for making HTTP/HTTPS requests.

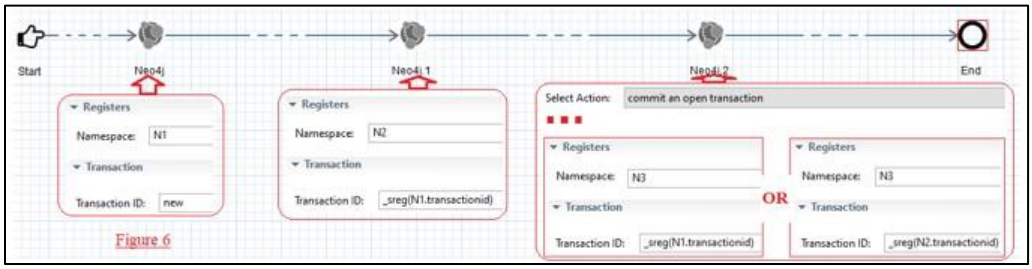
The **Namespace** and **Transaction ID** properties are used to configure transactions for most actions, except the *authenticate to access the server* action. The Neo4j transactional HTTP endpoint allows you to execute an action within the scope of a transaction. The transaction may be kept open across multiple HTTP requests, until the client chooses to commit or roll back. You can also include an action, along with a request, to begin or commit a transaction.

Namespace. Name of the special registers namespace to prepend to any created register or variable returned by the object. It can be used to uniquely identify an object of the Neo4j Connector in a process flow.

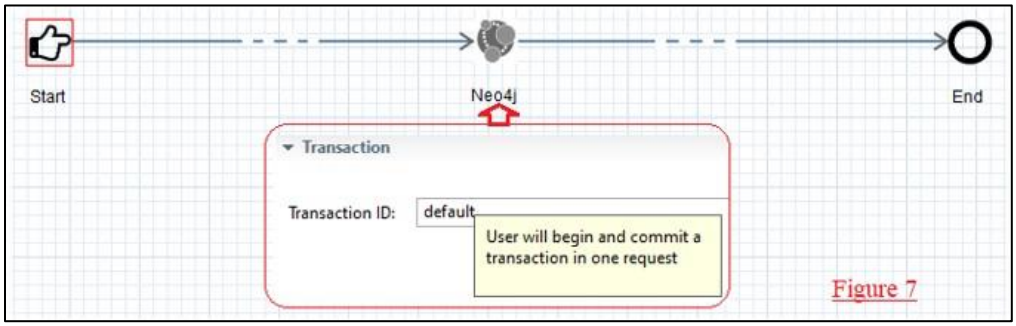
Transaction ID. Used to identify a transaction. You can open a new transaction by choosing *Open New Transaction* or use an existing opened transaction to send request(s).

To use an existing opened transaction, you can either type *_sreg([Namespace].transactionid)* or a valid transaction ID to represent the opened transaction (for example, type *_sreg(mynp.transactionid*), which is used by a Neo4j agent whose Namespace is *mynp*.

In a process flow, if one object of the Neo4j Connector opens a new transaction or uses an opened transaction to send a request, then another object of the Neo4j Connector must exist in the process to run the *commit an open transaction* or *rollback an open transaction action*, as shown in the following image.

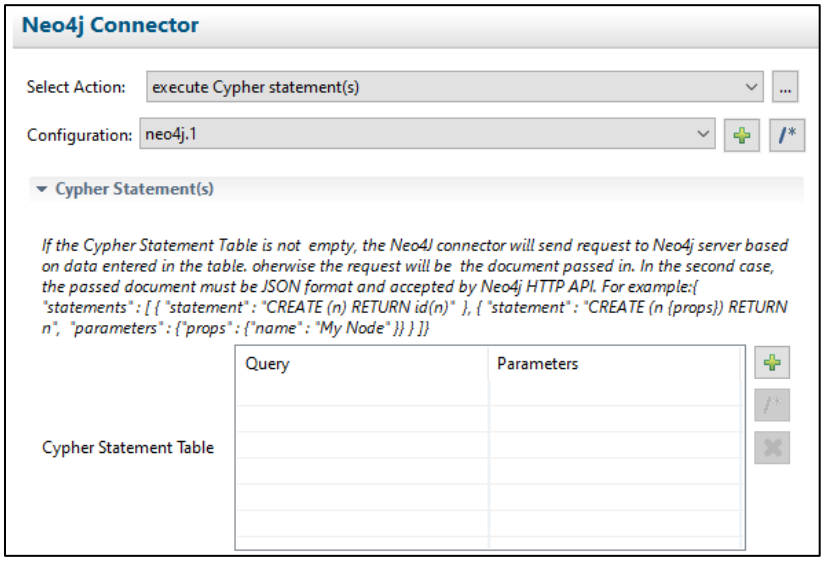


By default, the Neo4j Connector will begin and commit a transaction in one request, as shown in the following image.



Understanding Neo4j Connector Actions

The Neo4j Connector sends a request to the Neo4j Server according to a selected action. Each action is used for a particular purpose. For most actions, the Neo4j Connector makes a request based on the fields that you fill in. For *execute Cypher statement(s)* and *commit an open transaction* actions, in addition to using filled fields to make the request, the Neo4j Connector can also use an input document as the request (in this case, you must leave empty cells in the Cypher Statement Table). The Neo4j Connector can send multiple Cypher statements in the same request for the *execute Cypher statement(s)* or *commit an open transaction* action, as shown in the following image.



You can add Cypher statement(s) using the Cypher Statement Table. In the table, each row of data represents a Cypher statement. The *Query* field is a parameter of statement and is required. The *Parameters* field is a parameter of statement and is optional.

The following are samples for the *execute Cypher statement(s)* or *commit an open transaction* action:

- **Request based on input document.** Each of the following statements (Data1 through Data5) can be passed to the Neo4j Connector, and the Neo4j Connector can send it as request body to a Neo4j Server.

- **Request based on Cypher Statement Table.** Each of the statements (Data1 through Data5) can be stored in a Cypher Statement Table, as shown in the following images.

Data 1

```
{
 "statements" : [ {
  "statement" : " MATCH (n) RETURN n"
 }]
}
```

Cypher Statement(s)

If the Cypher Statement Table is not empty, the Neo4J connector will send request to Neo4j server based on data entered in the table. oherwise the request will be the document passed in. In the second case, the passed document must be JSON format and accepted by Neo4j HTTP API. For example:{ "statements" : [{ "statement" : "CREATE (n) RETURN id(n)" }, { "statement" : "CREATE (n {props}) RETURN n", "parameters" : {"props" : {"name" : "My Node" }} }]}

Cypher Statement Table

Query	Parameters
MATCH (n) RETURN n	

Data 2

```
{
 "statements" : [ {
  "statement" : "CREATE (n:MYLABEL{props}) RETURN n",
  "parameters" : {
   "props" : {
    "name" : "My TEMP Node"
   }
  }
 }]
}
```

Cypher Statement(s)

If the Cypher Statement Table is not empty, the Neo4J connector will send request to Neo4j server based on data entered in the table. oherwise the request will be the document passed in. In the second case, the passed document must be JSON format and accepted by Neo4j HTTP API. For example:{ "statements" : [{ "statement" : "CREATE (n) RETURN id(n)" }, { "statement" : "CREATE (n {props}) RETURN n", "parameters" : {"props" : {"name" : "My Node" }} }]}

Cypher Statement Table

Query	Parameters
CREATE (n:MYLABLE{props}) RETURN n	{ "props":{"name":"My TEMP Node"}}

Data 3

```
{
  "statements": [{
  "statement": " MATCH (n:HTH)  WHERE n.name = $name RETURN n, LABELS(n)",
      "parameters": {
          "name": "Alex"
      }
  }]
}
```

Cypher Statement(s)

If the Cypher Statement Table is not empty, the Neo4J connector will send request to Neo4j server based on data entered in the table. oherwise the request will be the document passed in. In the second case, the passed document must be JSON format and accepted by Neo4j HTTP API. For example:{ "statements" : [{ "statement" : "CREATE (n) RETURN id(n)" }, { "statement" : "CREATE (n {props}) RETURN n", "parameters" : {"props" : {"name" : "My Node" }} }]}

Cypher Statement Table

Query	Parameters
MATCH (n:HTH) WHERE n.name=$name RETURN n, LABELS(n)	{"name":"Alex"}

Data 4

```
{
 "statements" : [ {
     "statement" : "MATCH (n:HTH) WHERE n.name=\"Alex\" SET n = $props",
     "parameters" :
    {
        "props" : {
                "school" : "UP",
                "position" : "Developer",
                "Age":36
        }
    }
  }]
}
```

Cypher Statement(s)

If the Cypher Statement Table is not empty, the Neo4J connector will send request to Neo4j server based on data entered in the table. oherwise the request will be the document passed in. In the second case, the passed document must be JSON format and accepted by Neo4j HTTP API. For example:{ "statements" : [{ "statement" : "CREATE (n) RETURN id(n)" }, { "statement" : "CREATE (n {props}) RETURN n", "parameters" : {"props" : {"name" : "My Node" }} }]}

Cypher Statement Table

Query	Parameters
MATCH (n:HTH) WHERE n.name="Alex" SET n = $props	{"props":{"school":UP","position":"Develope","Age":36}}

Data 5

```
{
 "statements" : [ {
      "statement" : "CREATE p =(a:IBI { name:'Andy' })-[:WORKS_AT{name:'line1'}]->
                    (b:LOCATION{name:'US'})<-[:WORKS_AT {name:'line2'}]-(c: APPLE{
                    name: 'Michael' }) RETURN p"
      }, {
      "statement" : "CREATE (It:Country {name: 'Itly'}) "
      }, {
      "statement" : "MATCH (n) RETURN n "
      }, {
      "statement" : "MATCH ()-[n]-() RETURN n "
      }]
}
```

Cypher Statement(s)

If the Cypher Statement Table is not empty, the Neo4J connector will send request to Neo4j server based on data entered in the table. oherwise the request will be the document passed in. In the second case, the passed document must be JSON format and accepted by Neo4j HTTP API. For example:{ "statements" : [{ "statement" : "CREATE (n) RETURN id(n)" }, { "statement" : "CREATE (n {props}) RETURN n", "parameters" : {"props" : {"name" : "My Node" }} }]}

Cypher Statement Table

Query	Par
CREATE p =(a:IBI { name:'Andy' })-[:WORKS_AT{name:'line1'}]->(b:LOCATION{name:'US'})<-[:WORKS_AT{name:'line2'}]-(c: APPLE{ name: 'Michael' }) RETURN p	
CREATE (It:Country {name: 'Itly'})	
MATCH (n) RETURN n	
MATCH ()-[n]-() RETURN n	