



Course 12 - Graphit

Agenda



- What is Graphit
- Node types
- Node properties
- Parameters
- Variables
- Error handling
- Input parameters
- Permissions
- Versioning
- Verb
- How to invoke Graphit



What is Graphit?

- Graphit is a Fabric UI utility for designing web services with minimal coding.
- Graphit enables the visual design of JSON, XML, and CSV documents.
- If the Graphit is defined under LUT (not Web Services) it can be invoked from other Fabric components.

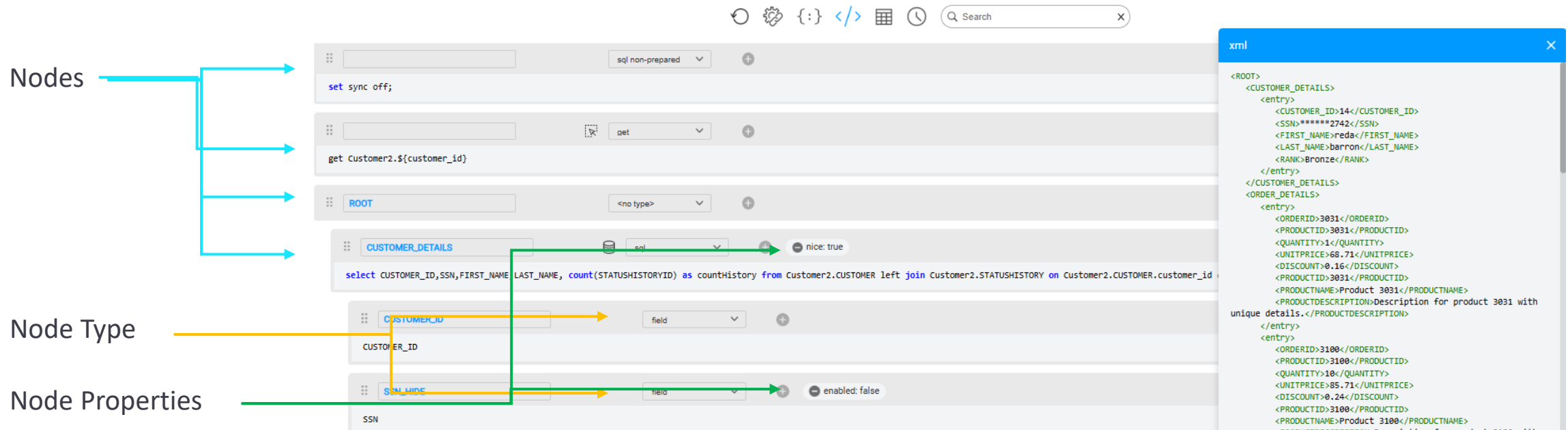
For example:

- Store API response in the LUI in advance, to have faster Web Service response)
- Send CDC messages to downstream apps

What is Graphit?

Graphit Terminology

- **Node:** Represents an output element or a functional element within the system.
- **Node Type:** Specifies the method for generating and structuring the content of a node.
- **Node Property:** Provides additional instructions to a node, such as formatting a number, specifying a database to query, or indicating whether the node is active or disabled.





Graphit Node Types and Properties

Graphit **Node Types** are organized into four logical categories

- **Field Nodes** - Represent individual data points.
Node Types: Field, Function, String, Raw
- **Loop Nodes** - Handle arrays or repeating sections.
Node Types: SQL, SQL non-prepared, Function
- **Structure Nodes** - Manage output structure.
Node Types: Condition, Group, Collect.
- **Execute Nodes** – Execute actions like GET requests or Broadway flows.
Node Types: GET, Broadway.

Graphit **Node Properties** define additional behavior for nodes, such as:

- Formatting numbers and dates.
- Specifying the database for query execution.
- Modifying node output.

Each node must be assigned a specific node type, and additional properties can be used to customize its functionality. Nodes that generate output must also have a designated name.

Graphit Node Types

- **Field** - Basic node type. Defines the node as a tag in XML/JSON format.

```
CUSTOMER_DETAILS  
sql  
+ nice: true  
select CUSTOMER_ID,SSN,FIRST_NAME,LAST_NAME, count(STATUSHISTORYID) as countHistory from Customer2.CUSTOMER 1
```

Tag name →
Tag value →

CUSTOMER_ID	field	+
CUSTOMER_ID		
SSN_HIDE	field	+ - enabled: false
SSN		
SSN	function	+
''.repeat(SSN.length - 4) + SSN.slice(-4);		
FIRST_NAME	field	+
FIRST_NAME		
LAST_NAME	field	+

Output:

```
<CUSTOMER_DETAILS>  
  <entry>  
    <CUSTOMER_ID>14</CUSTOMER_ID>  
    <SSN>*****2742</SSN>  
    <SSN_test>2939782742</SSN_test>  
    <FIRST_NAME>reda</FIRST_NAME>  
    <LAST_NAME>barron</LAST_NAME>  
    <RANK>Bronze</RANK>  
  </entry>  
</CUSTOMER_DETAILS>
```

Graphit Node Types

- **SQL** - Specifies an SQL statement / Fabric commands. Support prepared statements and binding.
 - The SQL output automatically added as nodes to the Graphit.
 - The SQL Type loops through the returned records, executing nested code for each row in the result set.
- **Function** - JavaScript code that returns the value of an XML/JSON tag.

The screenshot displays the Graphit interface with a workflow containing four nodes. The first node is a SQL node, indicated by a database icon and a dropdown menu set to 'sql'. Below it, the SQL query is shown: `select * from Customer2.CASES where status=${case_status}`. The subsequent three nodes are field nodes, each with a dropdown menu set to 'field'. They are labeled 'ACTIVITY_ID', 'CASE_ID', and 'CASE_DATE'. The final node is a function node, indicated by a dropdown menu set to 'function', with the code `STATUS.toUpperCase();`. To the right, a preview of the XML output is shown, enclosed in a blue border. The XML structure is as follows:

```
<ROOT>
  <CASES>
    <entry>
      <ACTIVITY_ID>68</ACTIVITY_ID>
      <CASE_ID>34</CASE_ID>
      <CASE_DATE>2015-10-28 00:00:00.0</CASE_DATE>
      <CASE_TYPE>Billing Issue</CASE_TYPE>
      <STATUS>OPEN</STATUS>
    </entry>
    <entry>
      <ACTIVITY_ID>70</ACTIVITY_ID>
      <CASE_ID>35</CASE_ID>
      <CASE_DATE>2016-01-06 00:00:00.0</CASE_DATE>
      <CASE_TYPE>Device Issue</CASE_TYPE>
      <STATUS>OPEN</STATUS>
    </entry>
  </CASES>
</ROOT>
```

A blue bracket on the right side of the XML output is labeled 'SQL output iterations', indicating that the XML structure is repeated for each row returned by the SQL query.

Graphit Node Types

- **SQL non-prepared**

To be used if parts of the query are dynamic (such as table name).

The screenshot displays the K2View Graphit interface. The main workspace shows a hierarchical tree of nodes. The 'ROOT' node is expanded, revealing a 'BALANCE_DETAILS' node. This node is of type 'sql non-prepared' and contains the SQL query: `select * from ${schema_name}.balance limit 2`. The 'subscriber_id' node is also visible, with the field 'subscriber_id' selected. A blue box highlights the 'nice: true' property of the 'BALANCE_DETAILS' node. On the right, a window titled 'xml' shows the XML output of the query, which is a list of two entries, each containing subscriber and balance details.

```
<ROOT>
  <BALANCE_DETAILS>
    <entry>
      <subscriber_id>19</subscriber_id>
      <balance_id>241</balance_id>
      <balance_ref_id>19</balance_ref_id>
      <available_amount>414</available_amount>
      <reset_amount>8</reset_amount>
      <reset_date>2015-10-31 00:00:00.0</reset_date>
    </entry>
    <entry>
      <subscriber_id>19</subscriber_id>
      <balance_id>242</balance_id>
      <balance_ref_id>5</balance_ref_id>
    </entry>
  </BALANCE_DETAILS>
</ROOT>
```


Graphit Node Types

- **GET** - Executes a GET command on a specified LUT and IID.
- **String** - Provides value to the tag, supporting variables like input parameters or values from previous field nodes.
- **Broadway** - Defines a Broadway flow to be executed, The flow's output parameters automatically added as nodes to the Graphit.

The screenshot displays the Graphit interface with several nodes and their configurations. Blue boxes highlight the 'get', 'string', and 'broadway' node types. A blue arrow points to the 'text' attribute in the Broadway node configuration. On the right, the XML output is shown, demonstrating the structure of the data generated by these nodes.

Node Configuration:

- get** (highlighted): Configuration includes 'get Customer2.\${customer_id}'.
- string** (highlighted): Configuration includes 'Cases staunts is \${case_status}'.
- broadway** (highlighted): Configuration includes 'broadway Customer2.bwForGraphit 'text'='This is the date: ''

XML Output:

```
<xml>
<ROOT>
  <TITLE>Cases staunts is OPEN</TITLE>
  <BROADWAY_FLOW>
    <entry>
      <RESULT_FROM_BW>This is the date: 2025-08-13
11:36:42.314</RESULT_FROM_BW>
      <RESULT_FROM_BW_2>This is an output to
Graphit</RESULT_FROM_BW_2>
    </entry>
  </BROADWAY_FLOW>
</ROOT>
```

Annotations:

- BW flow input parameters:** Indicated by a bracket on the left, pointing to the 'BROADWAY_FLOW' node.
- BW flow output parameters:** Indicated by a bracket on the left, pointing to the 'RESULT_FROM_BW' and 'RESULT_FROM_BW_2' nodes.

Graphit Node Types

- **Condition** - Implements IF-ELSE functionality, where the node's content is a simple code that returns a Boolean result.
 - The first node after the condition represents the "true" branch.
 - The next sibling node represents the "false" branch.
 - Any child nodes under the true or false branches are included based on the evaluation of the condition.

The screenshot displays the k2view Graphit editor interface. On the left, a tree view shows the structure of a 'Condition' element. The 'Condition' element is highlighted with a blue border. It contains two main branches: 'True element including child nodes' and 'False element including child nodes'. The 'True element' branch is further expanded, showing two child nodes: 'True element 1' and 'True element 2'. The 'False element' branch is also expanded, showing two child nodes: 'False element 1' and 'False element 2'. On the right, a code editor window titled 'xml' shows the corresponding XML output. The XML structure is as follows:

```
<X></X>
<Y></Y>
<ELEMENT_FALSE>
  <SUB_ELEMENT_FALSE>False element 1</SUB_ELEMENT_FALSE>
  <SUB_ELEMENT_FALSE2>False element 2</SUB_ELEMENT_FALSE2>
</ELEMENT_FALSE>
</ROOT>
```

Graphit Node Types

- **Group** - Groups several elements under a single entry tag. Used mainly with Condition nodes.

The screenshot displays the Graphit node editor interface. The top node is a 'condition' node with the label 'x<y'. Below it, there are two main branches: 'ELEMENT_TRUE' and 'ELEMENT_FALSE'. Each branch contains a 'group' node, which is highlighted with a blue box. The 'group' nodes are used to group multiple 'string' nodes. The 'ELEMENT_TRUE' branch contains two 'string' nodes labeled 'True element 2' and 'True element 1'. The 'ELEMENT_FALSE' branch contains two 'string' nodes labeled 'False element 1' and 'False element 2'. To the right, there are two XML output windows. The top window, titled 'xml', shows the XML output for the 'Result without Group' configuration, which includes nested tags for the 'group' nodes. The bottom window, also titled 'xml', shows the XML output for the 'Result with Group' configuration, where the 'group' nodes are represented by a single 'ELEMENT_TRUE' tag. A blue bracket labeled 'Result without Group' points to the top XML window.

```
<ROOT>
  <X>4</X>
  <Y>5</Y>
  <SUB_ELEMENT_TRUE2>True element 2</SUB_ELEMENT_TRUE2>
  <SUB_ELEMENT_TRUE>True element 1</SUB_ELEMENT_TRUE>
</ROOT>
```

```
<ROOT>
  <X>4</X>
  <Y>5</Y>
  <ELEMENT_TRUE>
    <SUB_ELEMENT_TRUE2>True element 2</SUB_ELEMENT_TRUE2>
    <SUB_ELEMENT_TRUE>True element 1</SUB_ELEMENT_TRUE>
  </ELEMENT_TRUE>
</ROOT>
```

Graphit Node Types

- **Row** - Presents data as output without any manipulation. For example, a header for an XML format.

The screenshot displays the K2View Graphit editor interface. The main configuration area shows a tree structure of nodes:

- ROOT** (Node Type: raw) - Contains the XML header: `<?xml version="1.0" encoding="UTF-8">`
- TITLE** (Node Type: string) - Contains the text: `Cases stauts is ${case_status}`
- BROADWAY_FLOW** (Node Type: broadway) - Contains the text: `broadway Customer2.bwForGraphit 'text'='This is the date: '`
- RESULT_FROM_BW** (Node Type: field) - Contains the text: `result`

A preview window on the right, titled 'xml', shows the resulting XML output:

```
<?xml version="1.0" encoding="UTF-8"><ROOT>
  <BROADWAY_FLOW>
    <entry>
      <RESULT_FROM_BW>This is the date:  2024-11-18
08:16:24.232</RESULT_FROM_BW>
      <RESULT_FROM_BW_2>This is an output to
Graphit</RESULT_FROM_BW_2>
    </entry>
  </BROADWAY_FLOW>
</ROOT>
```

Graphit Node Types

- **Collect** - Combines multiple datasets into a single unified array.

The screenshot shows the Graphit interface with a 'COLLECT_ELEMENT' node at the top. Below it are three input nodes: 'ORDER_DETAILS' (containing an SQL query), 'ORDERID' (a field node), and 'PRODUCTID' (a field node). The 'ORDER_DETAILS' node's query is partially visible: "NTITY,ORDERDETAILS.UNITPRICE,ORDERDETAILS.DISCOUNT,PRODUCTS.PRODUCTID,PRODUCTS.PRODUCTNAME,PRODUCTS.PRODUCTDESCRIPTION from Customer2.ORDERDETAILS left join common.PRODUCTS w".

json

```
{
  "COLLECT_ELEMENT": [
    {
      "ORDERID": 3031,
      "PRODUCTID": 3031
    },
    {
      "ORDERID": 3100,
      "PRODUCTID": 3100
    },
    {
      "ORDERID": 3802,
      "PRODUCTID": 3802
    },
    {
      "ORDERID": 5139,
      "PRODUCTID": 5139
    }
  ]
}
```

Result without
collect

json

```
{
  "ORDER_DETAILS": [
    {
      "ORDERID": 3031,
      "PRODUCTID": 3031
    },
    {
      "ORDERID": 3100,
      "PRODUCTID": 3100
    }
  ],
  "ORDER_DETAILS2": [
    {
      "ORDERID": 3802,
      "PRODUCTID": 3802
    },
    {
      "ORDERID": 5139,
      "PRODUCTID": 5139
    }
  ]
}
```

Graphit Node Properties

- **Session Provider** – Used with SQL or Non-Prepared SQL node types to specify the interface to be used for executing a query, if not defaulting to Fabric.

The screenshot shows the Graphit interface with a node named 'CUSTOMERS' of type 'sql'. The 'sessionProvider' property is highlighted with a blue box and set to 'M_CRM'. The query is 'select * from public.contract limit 2'. The output fields are 'customer_id', 'contract_id', and 'associated_line'. A preview window on the right shows the XML output.

```
<ROOT><CUSTOMERS><entry><customer_id>25</customer_id>
<contract_id>101</contract_id><associated_line>123-456-7890</associated_line><contract_description>Unlimited
call</contract_description><from_date>2024-01-01
00:00:00.0</from_date><to_date>2024-12-31 00:00:00.0</to_date>
<associated_line_fmt>1234567890</associated_line_fmt>
<reference_contract_id>102</reference_contract_id></entry><entry>
<customer_id>25</customer_id><contract_id>102</contract_id>
<associated_line>123-456-7891</associated_line>
<contract_description>Unlimited call</contract_description>
<from_date>2024-01-01 00:00:00.0</from_date><to_date>2024-12-31
00:00:00.0</to_date>
<associated_line_fmt>1234567891</associated_line_fmt>
<reference_contract_id>102</reference_contract_id></entry>
</CUSTOMERS></ROOT>
```

- **Nice** - Configures the layout of the output format. When set to True, each tag is printed on a new line and properly indented.

The screenshot shows the Graphit interface with the same 'CUSTOMERS' node and query. The 'nice' property is highlighted with a blue box and set to 'true'. The output fields are 'customer_id', 'contract_id', and 'associated_line'. A preview window on the right shows the XML output with proper indentation.

```
<ROOT>
  <CUSTOMERS>
    <entry>
      <customer_id>25</customer_id>
      <contract_id>101</contract_id>
      <associated_line>123-456-7890</associated_line>
      <contract_description>Unlimited
call</contract_description>
      <from_date>2024-01-01 00:00:00.0</from_date>
      <to_date>2024-12-31 00:00:00.0</to_date>
      <associated_line_fmt>1234567890</associated_line_fmt>
      <reference_contract_id>102</reference_contract_id>
    </entry>
    <entry>
      <customer_id>25</customer_id>
      <contract_id>102</contract_id>
      <associated_line>123-456-7891</associated_line>
      <contract_description>Unlimited
call</contract_description>
      <from_date>2024-01-01 00:00:00.0</from_date>
      <to_date>2024-12-31 00:00:00.0</to_date>
    </entry>
  </CUSTOMERS>
</ROOT>
```


Graphit Node Properties

- **One** – Determines whether the node is treated as a single value or an array. When set to True, the result is always a single entry, even if the underlying query returns multiple rows.

The screenshot displays the Graphit interface with several nodes. The 'CUSTOMER_DETAILS' node is highlighted with a blue box around its configuration area, where the 'one' property is set to 'true'. Below this node, three field nodes are visible: 'CUSTOMER_ID', 'FIRST_NAME', and 'LAST_NAME'. To the right, two XML output examples are shown. The first, titled 'Result without 'one'', shows a single entry for customer details. The second, titled 'Result with 'one'', shows a single entry for customer details, but it is preceded by an 'entry' tag, indicating that the result is treated as a single entry even if the underlying query returns multiple rows.

```
set sync off;
```

```
get Customer2.${customer_id}
```

```
ROOT
```

```
CUSTOMER_DETAILS
```

```
select CUSTOMER_ID,SSN,FIRST_NAME, LAST_NAME, count(STATUSHISTORYID) as counthistory from Customer2.CUSTOMER left join Customer2.STATUSHISTORY on Customer2.CUSTOMER.customer_id = Cu
```

```
CUSTOMER_ID
```

```
CUSTOMER_ID
```

```
FIRST_NAME
```

```
FIRST_NAME
```

```
LAST_NAME
```

```
LAST_NAME
```

```
xml
```

```
<ROOT>
  <CUSTOMER_DETAILS>
    <CUSTOMER_ID>14</CUSTOMER_ID>
    <FIRST_NAME>reda</FIRST_NAME>
    <LAST_NAME>barron</LAST_NAME>
  </CUSTOMER_DETAILS>
</ROOT>
```

```
Result without 'one'
```

```
xml
```

```
<ROOT>
  <CUSTOMER_DETAILS>
    <entry>
      <CUSTOMER_ID>14</CUSTOMER_ID>
      <FIRST_NAME>reda</FIRST_NAME>
      <LAST_NAME>barron</LAST_NAME>
    </entry>
  </CUSTOMER_DETAILS>
</ROOT>
```

Graphit Node Properties

- Entry Tag** - Specifies the tag used to enclose XML array entries. If not specified or set to None, the default value `<entry>` is applied.

The screenshot shows a Graphit query pipeline in K2View. The first node is a 'set' node with 'sync off'. The second is a 'get' node with 'Customer2.\${customer_id}'. The third is a 'ROOT' node with '<no type>' and 'nice: true'. The fourth is a 'CASES' node with 'sql' and 'entryTag: CASE' (highlighted with a blue box). Below it are three 'field' nodes for 'ACTIVITY_ID', 'CASE_ID', and 'CASE_DATE'.

Result without 'entryTag'

```
xml
<ROOT>
  <CASES>
    <CASE>
      <ACTIVITY_ID>68</ACTIVITY_ID>
      <CASE_ID>34</CASE_ID>
      <CASE_DATE>2015-10-28 00:00:00.0</CASE_DATE>
      <CASE_TYPE>Billing Issue</CASE_TYPE>
      <STATUS>Open</STATUS>
    </CASE>
  </CASES>
</ROOT>
```

```
xml
<ROOT>
  <CASES>
    <entry>
      <ACTIVITY_ID>68</ACTIVITY_ID>
      <CASE_ID>34</CASE_ID>
      <CASE_DATE>2015-10-28 00:00:00.0</CASE_DATE>
      <CASE_TYPE>Billing Issue</CASE_TYPE>
      <STATUS>Open</STATUS>
    </entry>
    <entry>
      <ACTIVITY_ID>70</ACTIVITY_ID>
      <CASE_ID>35</CASE_ID>
      <CASE_DATE>2016-01-06 00:00:00.0</CASE_DATE>
      <CASE_TYPE>Device Issue</CASE_TYPE>
      <STATUS>Open</STATUS>
    </entry>
    <entry>
      <ACTIVITY_ID>73</ACTIVITY_ID>
      <CASE_ID>37</CASE_ID>
      <CASE_DATE>2016-12-21 00:00:00.0</CASE_DATE>
      <CASE_TYPE>Billing Issue</CASE_TYPE>
      <STATUS>Unresolved</STATUS>
    </entry>
  </CASES>
</ROOT>
```

Graphit Node Properties

- **Keys** - Advanced mechanism that replaces nested queries by joining the data on the root query and grouping it with a key.

Example without keys – join query return entry per record:

The screenshot displays the K2View Graphit interface with a query graph on the left and its XML output on the right.

Query Graph:

- ROOT** node (type: <no type>) with property `nice: true`. It has a child node **CASES_INFO**.
- CASES_INFO** node (type: sql) with a query: `select Customer2.CASES.ACTIVITY_ID, Customer2.CASES.CASE_ID, Customer2.CASES.STATUS, CASE_DATE, Customer2.CASE_NOTE.NOTE_TEXT, Customer2.CASE_NOTE.NOTE_DATE from Customer2.CASES inn`.
- ACTIVITY_ID** node (type: field) with value `ACTIVITY_ID`.
- STATUS** node (type: field) with value `STATUS`.
- NOTE_TEXT** node (type: field) with value `NOTE_TEXT`.
- NOTE_DATE** node (type: field) with value `NOTE_DATE`.

XML Output:

```
<xml>
<ROOT>
  <CASES_INFO>
    <entry>
      <ACTIVITY_ID>68</ACTIVITY_ID>
      <STATUS>Open</STATUS>
      <NOTE_TEXT>I&apos;ll alert the crew. When has justice ever been as simple as a rule book?</NOTE_TEXT>
      <NOTE_DATE>2022-09-05 00:00:00.0</NOTE_DATE>
    </entry>
    <entry>
      <ACTIVITY_ID>68</ACTIVITY_ID>
      <STATUS>Open</STATUS>
      <NOTE_TEXT>I&apos;ll alert the crew. When has justice ever been as simple as a rule book?</NOTE_TEXT>
      <NOTE_DATE>2022-04-05 00:00:00.0</NOTE_DATE>
    </entry>
    <entry>
      <ACTIVITY_ID>70</ACTIVITY_ID>
      <STATUS>Open</STATUS>
      <NOTE_TEXT>I suggest you drop it, Mr. Data. Mr. Worf, you sound like a man who&apos;s asking his friend if he can start dating his sister.</NOTE_TEXT>
      <NOTE_DATE>2021-11-20 00:00:00.0</NOTE_DATE>
    </entry>
    <entry>
      <ACTIVITY_ID>70</ACTIVITY_ID>
      <STATUS>Open</STATUS>
      <NOTE_TEXT>I&apos;ve had twelve years to think about it. And if I had it to do over again, I would have grabbed the phaser and pointed it at you instead of them.</NOTE_TEXT>
      <NOTE_DATE>2024-07-09 00:00:00.0</NOTE_DATE>
    </entry>
    <entry>
      <ACTIVITY_ID>73</ACTIVITY_ID>
      <STATUS>Unresolved</STATUS>
      <NOTE_TEXT>I&apos;ve had twelve years to think about it. And if I had it to do over again, I would have grabbed the
    </entry>
  </CASES_INFO>

```

Graphit Node Properties

The **Key** should contain the “group by” element

Example after adding the Keys property:

The screenshot displays the K2View Graphit interface with several nodes and their properties. The nodes are:

- ROOT**: Properties include `<no type>`, `+ nice: true`, and `keys: CASE_ID` (highlighted with a blue box).
- CASES_INFO**: Properties include `sql` and `keys: CASE_ID` (highlighted with a blue box).
- ACTIVITY_ID**: Property is `field`.
- STATUS**: Property is `field`.
- NOTES**: Properties include `keys: NOTE_ID` and `entryTag: NOTE` (highlighted with a blue box).
- CASE_ID**: Property is `field`.

On the right, an XML preview window shows the output structure. A blue box highlights a section of the XML:

```
<ROOT>
  <CASES_INFO>
    <entry>
      <ACTIVITY_ID>68</ACTIVITY_ID>
      <STATUS>Open</STATUS>
      <NOTES>
        <NOTE>
          <CASE_ID>34</CASE_ID>
          <NOTE_TEXT>I&apos;ll alert the crew. When has
justice ever been as simple as a rule book?</NOTE_TEXT>
          <NOTE_DATE>2022-09-05 00:00:00.0</NOTE_DATE>
        </NOTE>
        <NOTE>
          <CASE_ID>34</CASE_ID>
          <NOTE_TEXT>I&apos;ll alert the crew. When has
justice ever been as simple as a rule book?</NOTE_TEXT>
          <NOTE_DATE>2022-04-05 00:00:00.0</NOTE_DATE>
        </NOTE>
      </NOTES>
    </entry>
    <entry>
      <ACTIVITY_ID>70</ACTIVITY_ID>
      <STATUS>Open</STATUS>
      <NOTES>
        <NOTE>
          <CASE_ID>35</CASE_ID>
          <NOTE_TEXT>I suggest you drop it, Mr. Data. Mr.
Worf, you sound like a man who&apos;s asking his friend if he
can start dating his sister.</NOTE_TEXT>
          <NOTE_DATE>2021-11-20 00:00:00.0</NOTE_DATE>
        </NOTE>
        <NOTE>
          <CASE_ID>35</CASE_ID>
          <NOTE_TEXT>I&apos;ve had twelve years to think
about it. And if I had it to do over again, I would have grabbed
the phaser and nointed it at you instead of them.</NOTE_TEXT>
          <NOTE_DATE>2021-11-20 00:00:00.0</NOTE_DATE>
        </NOTE>
      </NOTES>
    </entry>
  </CASES_INFO>
</ROOT>
```

Graphit Node Properties

- **Attribute** – to change a value from an element to an attribute

The screenshot displays the k2view interface with two panels. The top panel shows a node named 'ORDER_DETAILS' with a 'sql' query and a 'nice: true' property. The bottom panel shows a node named 'ORDERID' with a 'field' property. A blue box highlights the 'attribute: true' property in the bottom panel. To the right, an XML preview shows the resulting output, with a red box highlighting the element `<ORDER_DETAILS ORDERID="3031">`.

ORDER_DETAILS

sql

select ORDERDETAILS.ORDERID,ORDERDETAILS.PRODUCTID,ORDERDETAILS.QUANTITY,ORDERDETAILS.UNITPRICE,ORDERDETAILS.DISCOUNT,PRODUCTS.PRODU

ORDERID

field

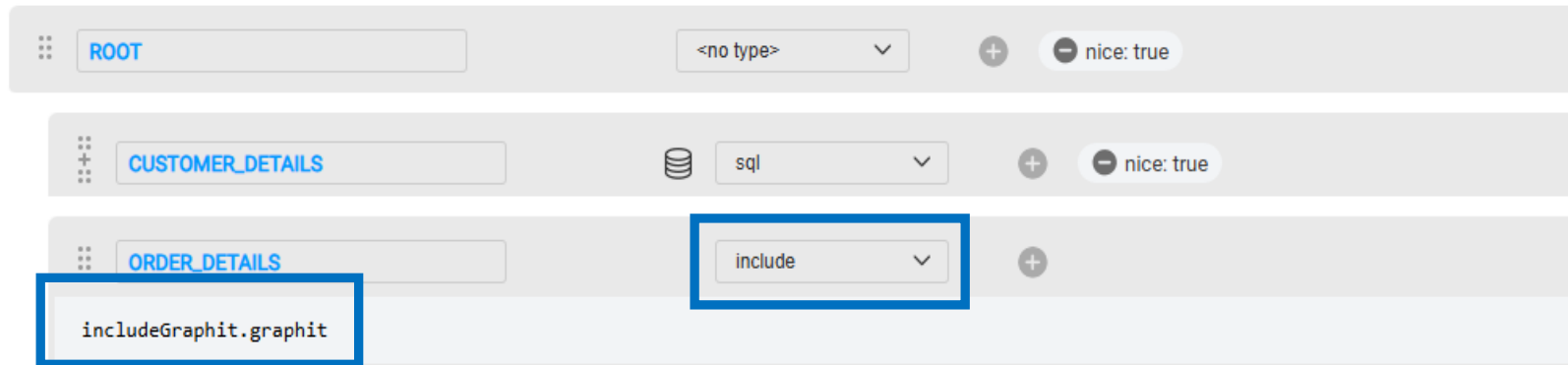
attribute: true

xml

```
</CUSTOMER_DETAILS>
<ORDER_DETAILS>
  <ORDER_DETAILS ORDERID="3031">
    <PRODUCTID>3031</PRODUCTID>
    <QUANTITY>1</QUANTITY>
    <UNITPRICE>68.71</UNITPRICE>
    <DISCOUNT>0.16</DISCOUNT>
    <PRODUCTID>3031</PRODUCTID>
    <PRODUCTNAME>Product 3031</PRODUCTNAME>
    <PRODUCTDESCRIPTION>Description for product 3031 with
unique details.</PRODUCTDESCRIPTION>
  </ORDER_DETAILS>
```

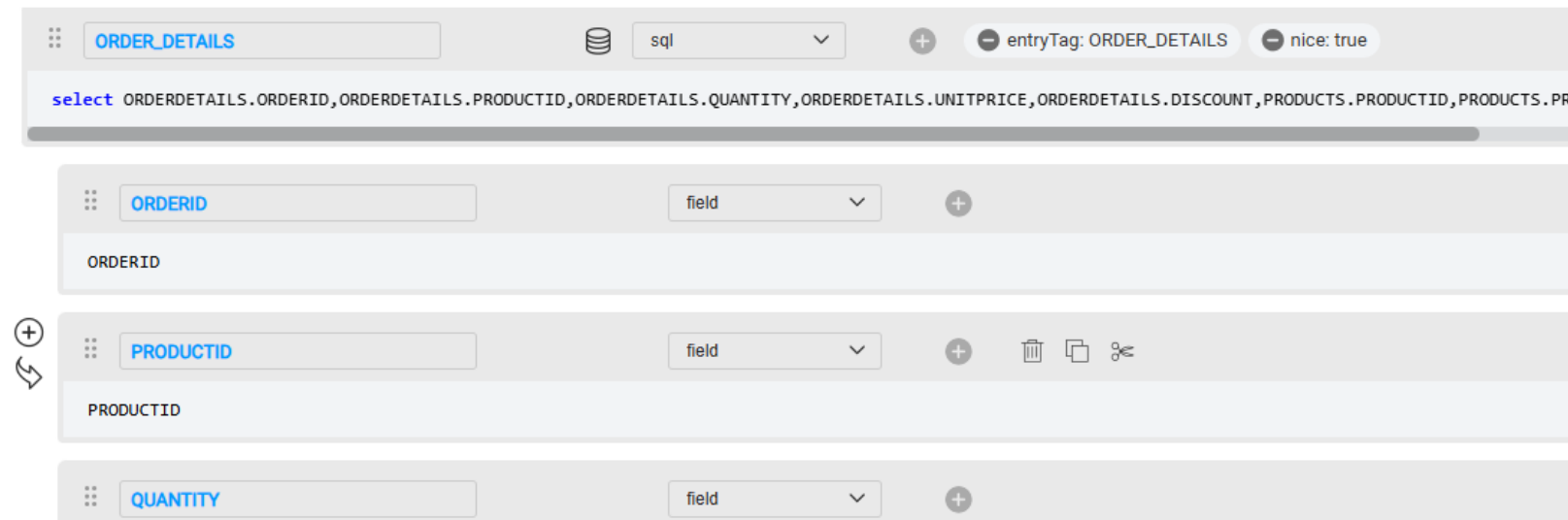
Graphit Node Properties

- **Include** – add another Graphit file to the existing Graphit



You can address any field/variables in the included graphit, as if it's written in the outer graphit

includeGraphit.graphit



Graphit Node Properties

Additional properties:

- **Show Empty** - Defines whether empty nodes are displayed in the output. Default = True. Note that this property affects the node and its child nodes.
- **Show null** - Defines whether null entries are displayed in the output. Default = True. Note that this property affects the node and its child nodes.
- Output formats:
 - **DatetimeFormat** - Controls how date/time are formatted in the output
 - **NumberFormat** - Controls how numbers are formatted in the output
- CSV - The following node properties control CSV format:
 - **csvHeader** - disables a header row
 - **csvRow** - defines the delimiter between rows in CSV format. The default value is set to the CR sign (\n).
 - **csvCol** - defines the delimiter between columns in CSV format. The default value is set to the comma character.
 - **csvEnclose** - defines the character used to enclose a value in CSV format. This is used only if the value holds a special character (csvEnclose, csvRow, csvCol).
- **Format** – When defined, the node is evaluated and added when the output format matches the format's JSON, XML or CSV value. Note that this property only affects the node where it is defined.

Graphit Variables

- Previous Field Nodes: Variables derived from earlier field nodes.
- SQL Query/Broadway Result: Data obtained from SQL queries or Broadway execution results.
- Input Parameters: User-defined inputs.
- Custom Variables in Function Node Type: Variables can be defined within a Function node (using JavaScript). Unlike Field nodes, these variables can be accessed outside the scope of the field node.

The screenshot displays the K2View Graphit interface with a data flow graph. The graph starts with a 'ROOT' node, followed by a 'CUSTOMER_DETAILS' node (SQL query), then a 'CUSTOMER_ID' node (field), and finally a 'MESSAGE' node (function). A 'MESSAGE_CUSTOMER' node (function) is also present, which outputs the 'message' variable. The 'MESSAGE' node uses the 'message' variable from the 'MESSAGE_CUSTOMER' node. The final output is an XML document.

Define variable

Use query result as variable

Use the variable from outside the loop

```
get Customer2.${customer_id}
```

ROOT

<no type>

CUSTOMER_DETAILS

sql

one: true nice: true

```
select CUSTOMER_ID,SSN,FIRST_NAME,LAST_NAME, count(STATUSHISTORYID) as countHistory from Customer2.CUSTOMER left join Customer2.STATUSHISTORY on Customer2.CUSTOMER.customer_id
```

CUSTOMER_ID

field

CUSTOMER_ID

function

```
var message = 'Not 215';  
if (CUSTOMER_ID == 215)  
  message = 'Customer 215';
```

MESSAGE_CUSTOMER

function

message

MESSAGE

function

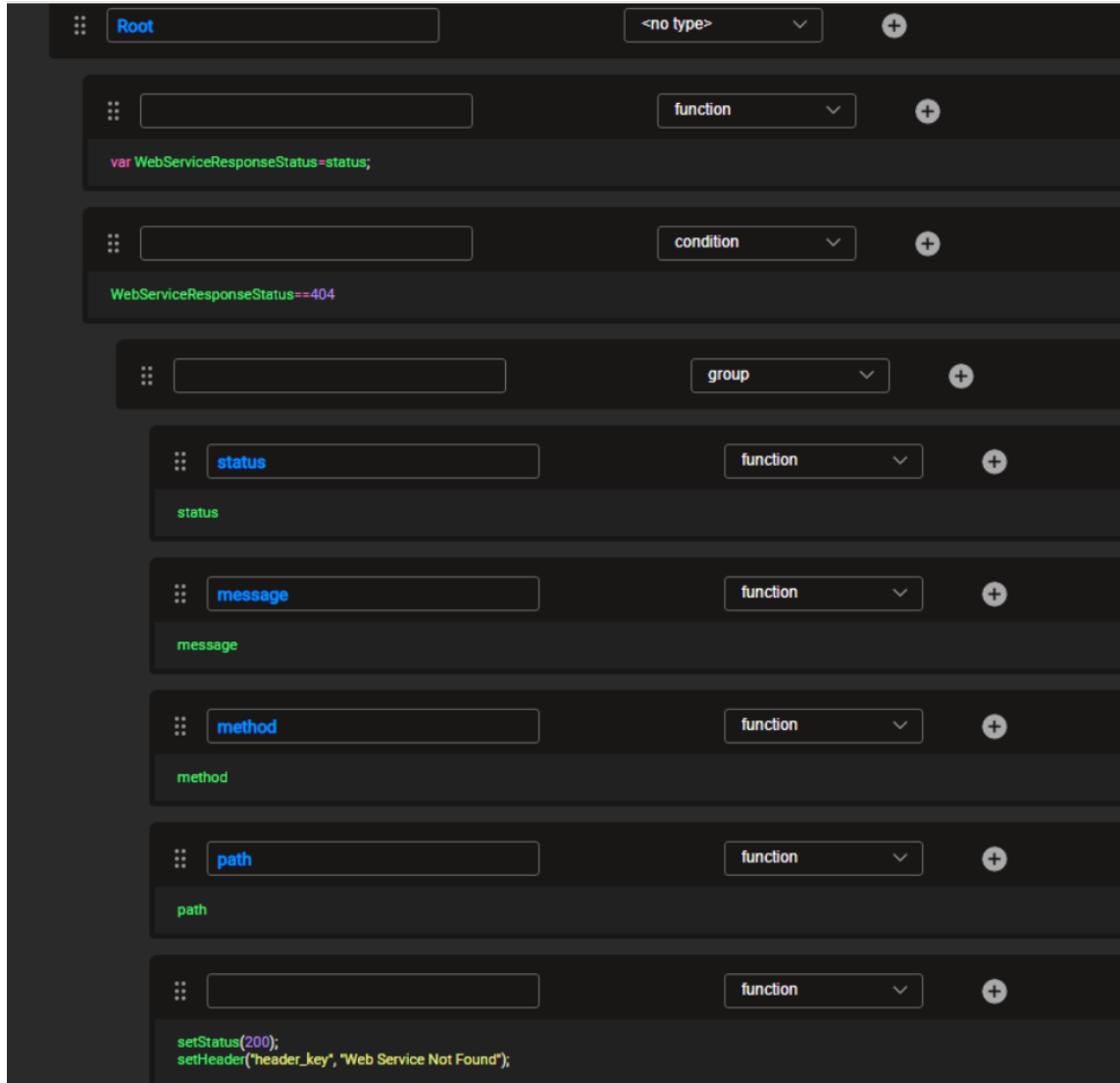
nice: true

message

xml

```
<ROOT>  
  <CUSTOMER_DETAILS>  
    <CUSTOMER_ID>14</CUSTOMER_ID>  
    <MESSAGE_CUSTOMER>Not 215</MESSAGE_CUSTOMER>  
  </CUSTOMER_DETAILS>  
  <MESSAGE>Not 215</MESSAGE>  
</ROOT>
```

Error Handling



Error.Graphit

- Executes automatically on unhandled exceptions from Fabric Web Service calls.
- Not triggered if the exception is caught in the implementation.
- Ensures a **standardized error response** in JSON or XML, matching the original request format.
- The implementer has full flexibility to determine the cause of the underlying failure that triggered the process and adjust the Web Service response accordingly, including:
 - Status – e.g., 404 for “Page Not Found” (use setStatus to override).
 - Message – custom error message text.
 - Path – the endpoint or resource path involved.
 - Method – the HTTP method used (GET, POST, etc.).
 - Header – response headers (use getHeader / setHeader to read or modify

Graphit Input parameters

Use  button to define parameters.

Use \${param_name} syntax

Binding

```
get Customer2.${customer_id}
```

Binding





```
select * from Customer2.CASES where status=${case_status}
```



URL Parameters & Properties

Input Parameters & Path

Parameters

Name	customer_id	Debug Value	14		
Name	case_status	Debug Value	Open		

☒ Expose as Endpoint

Override Path

Input parameters can be a part of a request URL path, by adding them to the path wrapped in curly brackets. For example, for a WS named 'demo' with x & y parameters, use 'demo/{x}/{y}', where request call can be: 'demo/1/2' (x=1, y=2). A combination of request parameter methods can be used too, e.g. 'demo/{x}?y='.

Url Info

GET 

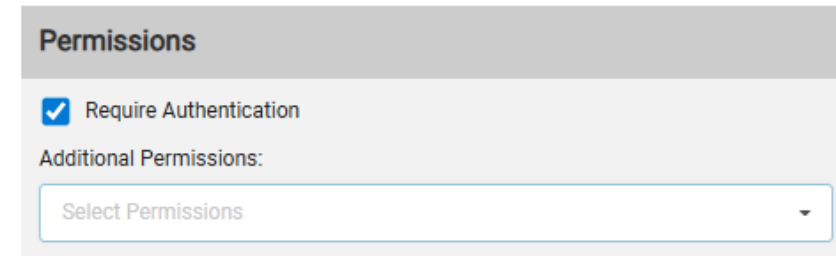
Graphit Permissions

- **Require Authentication**

- Indicates whether a web service requires authentication or not. Default is set to True.
- When set to False, it will allow calling the web service by skipping the Authentication step. This mode should be carefully used; use it only when a Web Service should be accessible for everyone, without enforcing an API key, a user/password, etc.

- **Additional Permissions**

- When a web service caller lacks the necessary role-based permissions to execute certain Fabric operations used within a Graphit, the developer can enable temporary elevated access specifically for the scope of that web service.
- To do so, select the required permissions in the Graphit's properties—these will override the caller's default permissions during execution.
- For the full list of available permissions, refer to:
https://support.k2view.com/Academy/articles/17_fabric_credentials/01_fabric_credentials_overview.html



The screenshot shows a configuration panel titled "Permissions". It contains a checked checkbox labeled "Require Authentication". Below this, the text "Additional Permissions:" is followed by a dropdown menu with the placeholder text "Select Permissions".

Graphit Versioning

Web Service Versioning

- Versioning allows multiple versions of a web service to coexist, enabling support for different client needs. The version number is typically included in the service URL, but it is optional—if omitted, the default version is 1.
- Versioning Logic
 - No version in URL → API returns the latest version.
 - Version in URL exists → API returns the specified version.
 - Version in URL higher than latest → API returns the latest version.
 - Version in URL lower than earliest → API returns an appropriate error response code.
 - Version in URL between two versions → API returns the lower of the two.
- Implementation
 - Graphit: Add the version as part of the file name, e.g., a.v1.graphit.

Graphit Verb

Methods supported by the web service, as follows:

- **GET** - get data.
- **POST** - create new data based on the data provided.
- **PUT** - update data.
- **DELETE** - delete data.

In Graphit, the verb setting is done by incorporating it as a part of the file name.
For example: "a.POST.graphit".

Note:

- Only a single verb can be set.
- When a verb is omitted, the web service is exposed with GET and POST verbs.

Invoke Graphit

Graphit files can be invoked either externally as Web Services or internally from other Fabric implementation components (if defined under LUT).

Web Service:

- Check “Expose as Endpoint” field
- Use the URLs provided in the Graphit Settings:
Note: you can override the URL path using the **Path** parameter

The screenshot shows the K2view Graphit editor interface. The main workspace displays a SQL query: `select CUSTOMER_ID,SSN,FIRST_NAME,LAST_NAME, count(STATUSHISTORYID) as countHistory`. Below the query, there are input fields for `CUSTOMER_ID`, `SSN`, `FIRST_NAME`, and `LAST_NAME`. The `SSN` field has a function applied: `'*'.repeat(SSN.length - 4) + SSN.slice(-4);`. A modal window titled "URL Parameters & Properties" is open, showing the "Expose as Endpoint" checkbox checked. The "Override Path" field is empty. The "Url Info" section shows the GET method with the URL `https://shani-mountain-solutions.eu-west3-2.cloud.k2view.com/api/Cus`. The POST method is also shown with the same URL and a JSON body: `{ "customer_id": "14" }`.

The screenshot shows a web browser displaying the XML response from the K2view API. The URL in the address bar is `shani-mountain-solutions.eu-west3-2.cloud.k2view.com/api/CustomerDetails?customer_id=14`. The browser shows the XML document tree, which is a SOAP response. The root element is `<root>`, and the main body is `<entry>`. The response contains customer details for customer ID 14, including SSN, first name, last name, and rank. It also includes order details for product 3031 and product 3100. A blue arrow points from the "POST" method in the Graphit editor to the XML response, indicating the result of the invocation.

Invoke Graphit

From Fabric Implementation Components

- Invoking From a Java Function

```
Map<String, Object> graphitParams = new HashMap<>();  
graphitParams.put("customer_id",14);  
return graphit("CustomerDetails.graphit", graphitParams);
```

- Invoking Broadway flow using Graphit Actor:

The screenshot displays a data flow tool interface. On the left, a Broadway flow is shown with two stages. Stage 1 contains a 'Customer2 IID' actor, and Stage 2 contains a 'Graphit1' actor. A data link connects the 'value' output of Stage 1 to the 'customer_id' input of Stage 2. The 'Graphit1' actor is configured with the following parameters:

- Graphit1 : Graphit
- All Fields
- Inputs
- graphit : string
- CustomerDetails
- graphitFormat : string
- JSON
- params : object
- customer_id : string

In the center, a 'Data Viewer' window displays a JSON string representing customer and order details:

```
1 [{"CUSTOMER_DETAILS": {  
2   "CUSTOMER_ID": 215,  
3   "SSN": "*****1083",  
4   "FIRST_NAME": "talieee",  
5   "LAST_NAME": "sears",  
6   "RANK": "Bronze"  
7 }},  
8 ]  
9  
10 [{"ORDER_DETAILS": {  
11   "ORDERID": 17571,  
12   "PRODUCTID": 17571,  
13   "QUANTITY": 1,  
14 }},  
15 ]
```

The 'Data Viewer' window also shows a 'JSON String' label and an 'OK' button.