Hewlett Packard
Enterprise

HPE Knowledge Graph

Software Version: 11.0

Knowledge Graph Technical Note

Document Release Date: March 2016
Software Release Date: March 2016

Legal Notices

Warranty

The only warranties for Hewlett Packard Enterprise Development LP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HPE shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HPE required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

This product includes an interface of the 'zlib' general purpose compression library, which is Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

Documentation Updates

HPE Big Data Support provides prompt and accurate support to help you quickly and effectively resolve any issue you may encounter while using HPE Big Data products. Support services include access to the Customer Support Site (CSS) for online answers, expertise-based service by HPE Big Data support engineers, and software maintenance to ensure you have the most up-to-date technology.

To access the Customer Support Site

- go to https://customers.autonomy.com

The Customer Support Site includes:

- **Knowledge Base**. An extensive library of end user documentation, FAQs, and technical articles that is easy to navigate and search.
- **Support Cases**. A central location to create, monitor, and manage all your cases that are open with technical support.
- **Downloads**. A location to download or request products and product updates.
- **Requests**. A place to request products to download or product licenses.

To contact HPE Big Data Customer Support by email or phone

- go to http://www.autonomy.com/work/services/customer-support

Support

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, visit the Knowledge Base on the HPE Big Data Customer Support Site. To do so, go to https://customers.autonomy.com, and then click **Knowledge Base**.

The Knowledge Base contains documents in PDF and HTML format as well as collections of related documents in ZIP packages. You can view PDF and HTML documents online or download ZIP packages and open PDF documents to your computer.

Contents

Introduction to Knowledge Graph

IDOL Knowledge Graph is an IDOL component that allows you to explore connections in a data set. You can define relationships between concepts or entities in IDOL documents, based on the field types in the document. Knowledge Graph then transforms the IDOL database into the set of relationships it contains. This set of relationships is called a *graph*.

You can use Knowledge Graph to enrich a database with meaningful, human-defined connections. The graph then allows you to create interesting queries to find out more about your data. For example, you can find out how many times something is connected to something else, and with what importance. There might be multiple types of connection in the graph, and each connection type might occur multiple times (represented by a count). You can also suggest new connections between entities based on their existing neighbors in the graph.

Knowledge Graph connects your data and reveals missing or hidden information. For example, you can use it to:

- find weak or fraudulent links in an e-mail chain.
- suggest a key competitive advantage
- find the fastest route to a particular market
- find an influential distributor

IDOL Knowledge Graph explicitly models relationships between parts of your data, offering an alternative view of your data to existing IDOL installations. This method of conceptualizing the data can make it easier to analyze patterns of dependency, causality, or endorsement, and to explain connections between items.

Graph Concepts

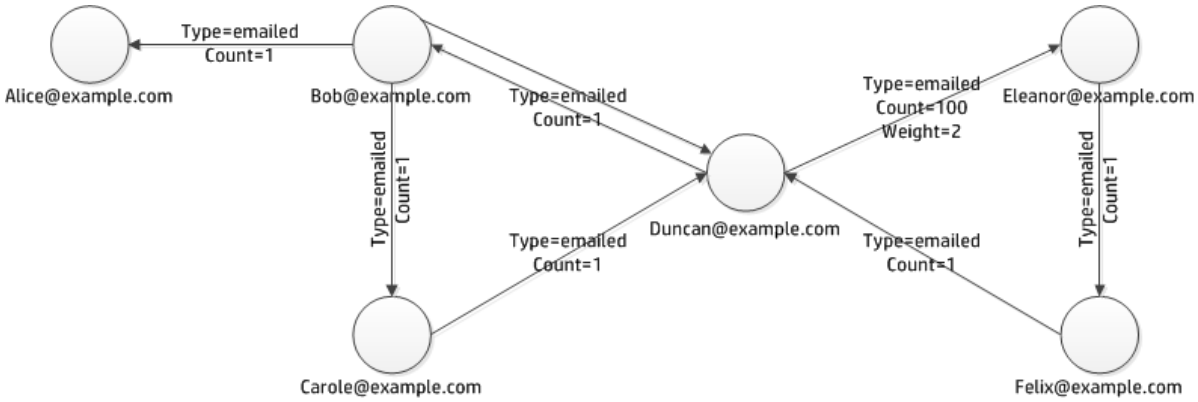
A graph is a representation of the relationships between concepts or entities.

Each concept is a *node* in the graph. The nodes have unique names, and Knowledge Graph assigns each node a unique ID. Relationships between nodes are represented by *edges*.

Edges are a directed connection between a *source node* to a *target node*. In addition, every edge has a type, and Knowledge Graph creates only one edge of each type between each unique pair of nodes (that is, a specified source and specified target). Each edge also has the following attributes:

- **Count**. The number of times a given edge occurs in the data. The first time a connection occurs in the data, Knowledge Graph creates the edge. Each subsequent time the same connection occurs, it increments the count attribute for the edge.
- **Weight**. A user-defined or automatically generated weight value. Knowledge Graph can use the weight of an edge as the distance along an edge when calculating the shortest path.

The following diagram shows an example graph.



Each node in this graph represents a person, identified by an e-mail address. All the edges have the type *emailed*.

The count represents the number of times a person has e-mailed a particular other person; for example, in this graph. Duncan has e-mailed Eleanor 100 times. The weight is a user-defined value; for example, in this graph it might correspond to the e-mail priority.

The graph also has the following features:

- **Path lengths**. A path is a sequence of edges connecting two nodes that are not necessarily adjacent to each other in the graph. The length or weight of a path is the sum of the weights of its edges. For an unweighted path, this length is the number of edges on the path between two nodes. In this graph, the length of path from Felix to Alice is three, while from Felix to Eleanor is two. For all paths, edges are directional.

  Multiple paths can exist between two nodes, and the shortest path is determined by comparing path lengths. In this graph, there are two paths between Bob and Duncan. The shortest path (unweighted) is the direct edge. However, if the direct edge had a high weight, the path from Bob to Carole to Duncan might be a shorter route.
- **In Degree**. The *in degree* of a node is the number of edges that point to that node (the number of edges where the node is the target node). In this graph, Felix and Eleanor have an in degree of one, while Duncan has an in degree of three.
- **Out Degree**. The *out degree* of a node is the number of edges that lead from a node (the number of edges where the node is the source node). In this graph, Carole, Eleanor, and Felix have an out degree of one, while Duncan and Bob have an out degree of two.

Create and Query a Graph

IDOL Knowledge Graph is an in-memory graph database, which also automatically persists its state to disk. You can configure the graph by defining edges in the server configuration file. For more information about configuration, see "Configure Knowledge Graph" on page 11.

You can index documents into Knowledge Graph in JSON format, for example by using the Connector Framework Server (CFS) to extract content from a repository. You can also index existing IDX documents into the server, by using the Python script provided in the Knowledge Graph installation.

The IDOL Knowledge Graph interface uses the ACI API. The interface includes actions to allow you to add data to your graph and to query the graph. The Knowledge Graph actions allow you to retrieve neighbors for specified nodes, paths between specified nodes, or subgraphs containing a specified set of nodes. It returns the information as IDOL XML responses.

Install and Run Knowledge Graph

You can install Knowledge Graph by using the IDOL Server installer. For more information about using this installer, refer to the *IDOL Getting Started Guide*.

You can also download the Knowledge Graph component installation package, which is a .zip file that you can extract to a location on your server.

After you install, you can configure and run Knowledge Graph.

Start Knowledge Graph

Note: Your License Server must be running before you start Knowledge Graph.

To start Knowledge Graph

- Start Knowledge Graph from the command line using the following command:

  ```
  graphserver.exe -configfile configname.cfg
  ```

  where the optional `-configfile` argument specifies the path of the configuration file that you want to use.
- Double-click the `graphserver.exe` file in the Knowledge Graph installation directory (Windows only).
- Start the Knowledge Graph service from the Windows Services dialog box (Windows only).

Send Actions to Knowledge Graph

Knowledge Graph actions are HTTP requests, which you can send, for example, from your web browser. The general syntax of these actions is:

`http://`*`host`*`:`*`port`*`/action=`*`action`*`&`*`parameters`*

where:

`host`	is the IP address or name of the machine where Knowledge Graph is installed.
`port`	is the Knowledge Graph ACI port. The ACI port is specified by the `Port` parameter in the `[Server]` section of the Knowledge Graph configuration file. For more information about the `Port` parameter, see the *Knowledge Graph Reference*.
`action`	is the name of the action you want to run.
`parameters`	are the required and optional parameters for the action.

Note: Separate individual parameters with an ampersand (&). Separate parameter names from

values with an equals sign (=). You must percent-encode all parameter values.

For more information about actions, see the *Knowledge Graph Reference*.

Display Online Help

You can display the Knowledge Graph Reference by sending an action from your web browser. The Knowledge Graph Reference describes the actions and configuration parameters that you can use with Knowledge Graph.

For Knowledge Graph to display help, the help data file (`help.dat`) must be available in the installation folder.

To display help for Knowledge Graph

1. Start Knowledge Graph.
2. Send the following action from your web browser:

   `http://host:port/action=Help`

   where:

   host is the IP address or name of the machine on which Knowledge Graph is installed.

   port is the ACI port by which you send actions to Knowledge Graph (set by the `Port` parameter in the `[Server]` section of the configuration file).

   For example:

   `http://12.3.4.56:9000/action=help`

GetStatus

You can use the `GetStatus` service action to verify the Knowledge Graph is running. For example:

`http://Host:ServicePort/action=GetStatus`

Note: You can send the `GetStatus` action to the ACI port instead of the service port. The `GetStatus` ACI action returns information about the Knowledge Graph setup.

Stop Knowledge Graph

You can stop Knowledge Graph by using one of the following procedures.

To stop Knowledge Graph

- Send the following action to the Knowledge Graph's service port.

  `http://host:ServicePort/action=stop`

where,

host	is the host name or IP address of the machine where Knowledge Graph is installed.
ServicePort	is the Knowledge Graph service port (specified in the `[Service]` section of the configuration file).

- If Knowledge Graph is running as a service, stop Knowledge Graph from the Windows Services dialog box. (Windows only)

Configure Knowledge Graph

After you install Knowledge Graph, you must configure the edges that you want to store in your graph.

You configure Knowledge Graph in the configuration file. By default, this has the name `graphserver.cfg`.

For details of all the configuration parameters that you can set in the Knowledge Graph configuration file, refer to the *Knowledge Graph Component Reference*.

Configure Edge Types

Knowledge Graph creates nodes and edges from your content according to your configuration. You must configure the edges in the Knowledge Graph configuration file before you index your data.

The `[Edges]` section contains a list of your edge types. You can also use this section to configure global settings for edge weighting. For more information, see "Configure Edge Weights" on the next page.

To configure an edge type

1. Open an example of the type of document that you want to index into Knowledge Graph to create your graph.
2. Find the fields that represent the kind of connections that you want to chart. For example:
   - In an e-mail document, you might want to create links between the `TO` and `FROM` fields to create a graph that describes e-mail connections in your organization.
   - In a wikipedia article, you might want to create a link between the title of the document and a list of other pages that the article links to. In this case, you might use CFS to add a field to your documents, containing the list of links.
3. Open the Knowledge Graph configuration file in a text editor.
4. Find the `[Edges]` configuration section, or create it if it does not exist.
5. In the list of types, add a new parameter to the list for each type that you want to create. The parameter name is the next value in the zero-based list of types. The value is the edge type. For example:

   ```
   [Edges]
   ...
   0=emailed
   1=corecipient
   ```

6. Create a new configuration section with the same name as your edge type. For example:

   ```
   [emailed]

   [corecipient]
   ```

7. For your new edge type, define the fields that you want to use to create nodes. You can use one of the following configuration parameter combinations to create nodes:
   - Set `Source` to a comma-separated list of fields whose values you want to use as source nodes. Set `Target` to a comma-separated list of fields whose values you want to use as target nodes. When a `Source` field and a `Target` field exist in a document, Knowledge Graph creates an edge between values in the source field and values in the target field, and the values become nodes. For example:

     ```
     [emailed]
     Source=FROM
     Target=TO,CC
     ```

   - Set `Nodes` to a comma-separated list of fields whose values you want to use as nodes. When these fields exist in a document, Knowledge Graph creates two edges (one in each direction) between each pair of nodes. For example:

     ```
     [corecipient]
     Nodes=CC,TO
     ```

8. Save the configuration file and restart Knowledge Graph for your changes to take effect.

Configure Edge Weights

Knowledge Graph stores a *weight* attribute for each edge. You can use this value to adjust how Knowledge Graph calculates the shortest path. You can use the following weighting methods for your graph:

- `None`. Do not assign weights to edges.
- `Field`. Use the value of a specified document field as the weight. You can configure the field to use, and define your own weight values in documents, which Knowledge Graph uses to set the weight edge attribute when it creates an edge.
- `Shortstep`. Use the *shortstep* method to automatically assign weights for all edges. This method determines the similarity of a pair of nodes by comparing the sets of out-neighbors for each node. If the nodes are identical, it assigns a weight of zero to the edge. Edges that connect nodes that are less similar have higher weights, up to infinity for distinct nodes.

For more information about these weighting methods, refer to the *Knowledge Graph Component Reference*.

To configure weighting for your edges

1. Open the Knowledge Graph configuration file in a text editor.
2. In the `[Edges]` section, set the `Weighting` parameter to the weighting method that you want to use. For example:

   ```
   [Edges]
   Weighting=Field
   ```

3. If you have set `Weighting` to `Field`, you must define the field that you want to use for weighting, by setting the `WeightFields` parameter to the name of the field.

Note: If you are not using field weighting, do not set the `WeightFields` parameter. If you set the `WeightFields` parameter when `Weighting` is not set to `Field`, Knowledge Graph logs an error and exits.

- To set a field to use for weighting for all your edge types, set the `WeightFields` parameter in the `[Edges]` section. You can use this setting as a global value for all your edge types, or as a default value, which you can override for individual edge types. For example:

  ```
  [Edges]
  Weighting=Field
  WeightFields=PRIORITY
  ```

- To set a field to use for weighting for a particular edge type, set the `WeightFields` parameter in the edge type configuration section. This setting overrides the `WeightFields` parameter in the `[Edges]` section, if it is set there. For example:

  ```
  [emailed]
  WeightFields=CONTACTWEIGHT
  ```

4. Save the configuration file and restart Knowledge Graph for your changes to take effect.
5. (Optional) If you have changed the weighting configuration and you already have documents indexed, you can use the `RegenerateWeights` action to update the weighting in your graph. For more information, refer to the *Knowledge Graph Component Reference*.

Configure the Knowledge Graph Index

Knowledge Graph stores index data in a file on disk. You can configure the name and format of this file by modifying parameters in the `[Persistence]` configuration section. For example:

```
[Persistence]
Filename=email_graph.bin
```

For more information, refer to the *Knowledge Graph Component Reference*.

Index Documents into Knowledge Graph

Knowledge Graph accepts indexed documents in JSON format. You can also use the python script `IDXtoJSON.py`, included in the Knowledge Graph installation directory to convert and index an existing IDX document. For example, you can use this script on an IDX document that you have exported from IDOL Server.

When you index a document, Knowledge Graph extracts the values of fields that you have configured as nodes in your edge configuration, and creates the graph of nodes and edges. After indexing, it creates a persistent store of the graph on disk.

Index a JSON Document

To index a JSON document, send the `IndexDocs` action to Knowledge Graph, using a POST request method, and set the `Data` parameter to the contents of the JSON document. For example:

```
http://12.3.4.56:9000/action=IndexDocs&Data=[{'title': 'document 1', 'myfield':
'value1'}, {'title': 'document 2', 'myfield': 'value 2'}]
```

Index an IDX Document

You can use the `IDXtoJSON.py` script to convert an existing IDX document to JSON and index it into Knowledge Graph.

Note: The script indexes all your documents in batches, without saving the graph index to disk. After indexing is complete, it runs a `Persist` action to save the graph index to disk, and the `RegenerateWeights` action to generate weights, if you have set the `Weighting` configuration parameter to `Shortstep`.

To run this script, you must have:

- Python version 2 or 3.
- The `Requests` module.

To index an IDX document using the script

- Open a command prompt in your Knowledge Graph installation directory, and run the following command:

  ```
  python IDXtoJSON.py --input InputPath --server ServerDetails --batchsize
  BatchSize
  ```

  where:

InputPath	is the file name and path to the input IDX file that you want to index into Knowledge Graph.

ServerDetails	is the host name and ACI port of the Knowledge Graph that you want to index into. Use the format *host:port*.
BatchSize	is the number of documents from the IDX file to include in each POST request to Knowledge Graph. The default value is 10000. **Note:** Large batch sizes can fail if the request becomes too big. If you do not want to reduce the batch size, you can increase the value of the MaxFileUploadSize configuration parameter in the [Server] section of the Knowledge Graph configuration file. For more information, refer to the *Knowledge Graph Component Reference*.

For example:

```
python IDXtoJSON.py --input /c/graphdata/graphinput.idx --server localhost:13000
--batchsize 1000
```

Use Connector Framework Server to Index Documents

The IDOL Connector Framework Server (CFS) processes files of different types and extracts the text into an IDOL document format. You can configure CFS to index into Knowledge Graph by using the IDOLACIIndexer library, which converts the normal CFS output to JSON format, which Knowledge Graph can index.

For more information about how to configure CFS and use it to retrieve documents, refer to the *Connector Framework Server Administration Guide*.

To configure CFS to index into Knowledge Graph, add the following configuration to your CFS configuration file:

```
[Indexing]
IndexerSections=Graph

[Graph]
IndexerType=Library
LibraryDirectory=IndexerDLLPath
LibraryName=IDOLACIIndexer
ACIIndexHost=GraphServerHost
ACIIndexPort=GraphServerPort
```

where,

IndexerDLLPath	is the path to your version of the IDOLACIIndexer library.
GraphServerHost	is the host name or IP address of the machine that your Knowledge Graph is installed on.
GraphServerPort	is the ACI Port of your Knowledge Graph.

By default, CFS creates XML documents. The library converts these documents to JSON. In this process, any XML attributes become JSON fields, and the JSON field name is the XML attribute name with the @ symbol as a prefix. For example, the following XML:

```
<Field1>Value1</Field1>
<Field2 attribute1="AttValue1">Value2</Field2>
```

Becomes the following in JSON format:

```
{
   "Field1" : "Value1",
   "Field2" : [
      "Value2",
      {
         "@Attribute1" : "AttValue1"
      }
   ]
}
```

Query Knowledge Graph

When you index content into Knowledge Graph it stores the nodes and edges for your graph. You can then query Knowledge Graph to find information about your data.

Knowledge Graph has the following querying actions. For full details of the actions and action parameters, refer to the *Knowledge Graph Component Reference*.

- `GetNeighbors`. Returns the neighbors of the nodes that you specify.

  For example, if your graph contains wikipedia articles, you can find a list of related concepts for a specified article. The following action finds a list of up to 10 articles that the Hewlett-Packard article links to.

  ```
  action=GetNeighbors&SourceNames=Hewlett&20Packard&20Enterprise&MaxResults=10
  ```

- `GetNeighborhood`. Returns the nodes within a specified distance of the nodes that you specify. This action also returns the edges between all the nodes in the set (input and result nodes), to create a subgraph from a starting set of nodes and closely related nodes. `GetNeighborhood` allows you to find out what a particular part of the graph looks like, without knowing the names or IDs of all the nodes.

  For example, if your graph contains wikipedia articles, you can create a neighborhood subgraph of articles that are closely related to a specified article. The following action finds up to 15 articles that are closely related to the articles on Chemistry and Physics, and any connections between them.

  ```
  action=GetNeighborhood&SourceNames=Chemistry,Physics&MaxResults=15
  ```

- `GetCommonNeighbors`. Returns neighbors of a list of nodes that you specify, together with a commonality value that indicates how many of your list of nodes the neighbor shares.

  For example, if your graph contains wikipedia articles, you can find a list of concepts that are related to two or more articles. The following action finds a list of up to 10 articles that the articles on Barack Obama and David Cameron both link to.

  ```
  action=GetCommonNeighbors&SourceNames=Barack%20Obama,David%20Cameron&MaxResults=1
  0
  ```

- `GetShortestPath`. Finds the shortest route between two specified nodes, using existing edges in the graph.

  For example, if your graph is made up of connections between places, you can find a route between them. The following action uses the graph to find the shortest route between London and Manchester. It uses the `Weighted` method, which returns the route with the lowest value for the sum of the weights of the edges. For this example, the weights might represent the physical distance or travel time between two nodes in the route.

  ```
  action=GetShortestPath&SourceName=London&TargetName=Manchester&Method=Weighted
  ```

- `GetShortestPaths`. Finds all nodes within a specified distance of a specified node, using existing edges in the graph.

  For example, if your graph is made up of connections between places, you can find all places within a specified distance of your starting point. The following action uses the graph to find all places

within a distance of 10 from Cambridge. It uses the `Weighted` method to calculate the distance. For this example, the weights might represent the physical distance or travel time between two nodes.

```
action=GetShortestPaths&SourceName=Cambridge&Method=Weighted&MaxDistance=10
```

- `GetSubgraph`. Returns a subgraph based on the set of nodes that you specify, showing all the edges that occur between these nodes.

  For example, if your graph contains a list of employees in your company and their contacts, you can create a subgraph that shows the connections in a particular department. The following action creates a subgraph for five users, where the users are specified by Knowledge Graph node ID.

  ```
  action=GetSubgraph&NodeIDs=10230,20345,20346,20347,21968
  ```

- `SuggestLinks`. Returns a list of nodes that are connected to neighbors of a node that you specify and that are not also neighbors of the specified node.

  For example, if your graph contains a list of users and their contacts, you can find friends of friends that a particular user might want to connect to. The following action finds contacts who are friends of friends of a user with the e-mail address `Alice@example.com`.

  ```
  action=SuggestLinks&NodeName=Alice%40example.com
  ```

Find Graph Details

The following actions allow you to get more information about your graph. For full details of the actions and action parameters, refer to the *Knowledge Graph Component Reference*.

- `GetNodes`. Lists the nodes in your graph. You can use this action to find the node IDs of your nodes.

  ```
  action=GetNodes&Sort=None
  ```

- `SummarizeGraph`. Returns a summary of the number of nodes and edges in your graphs, and details of the attributes stored for your edges.

  ```
  action=SummarizeGraph
  ```

Send Documentation Feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Knowledge Graph Technical Note (Knowledge Graph 11.0)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to AutonomyTPFeedback@hpe.com.

We appreciate your feedback!