# NDEx Network Data Model – The CX Format

*Last updated: June 20, 2017*

## Overview

CX is a network exchange format, designed as a flexible structure for transmission of networks. It is designed for flexibility, modularity, and extensibility, and as a message payload in common REST protocols. It is not intended as an optimized format for use in applications or for storage.

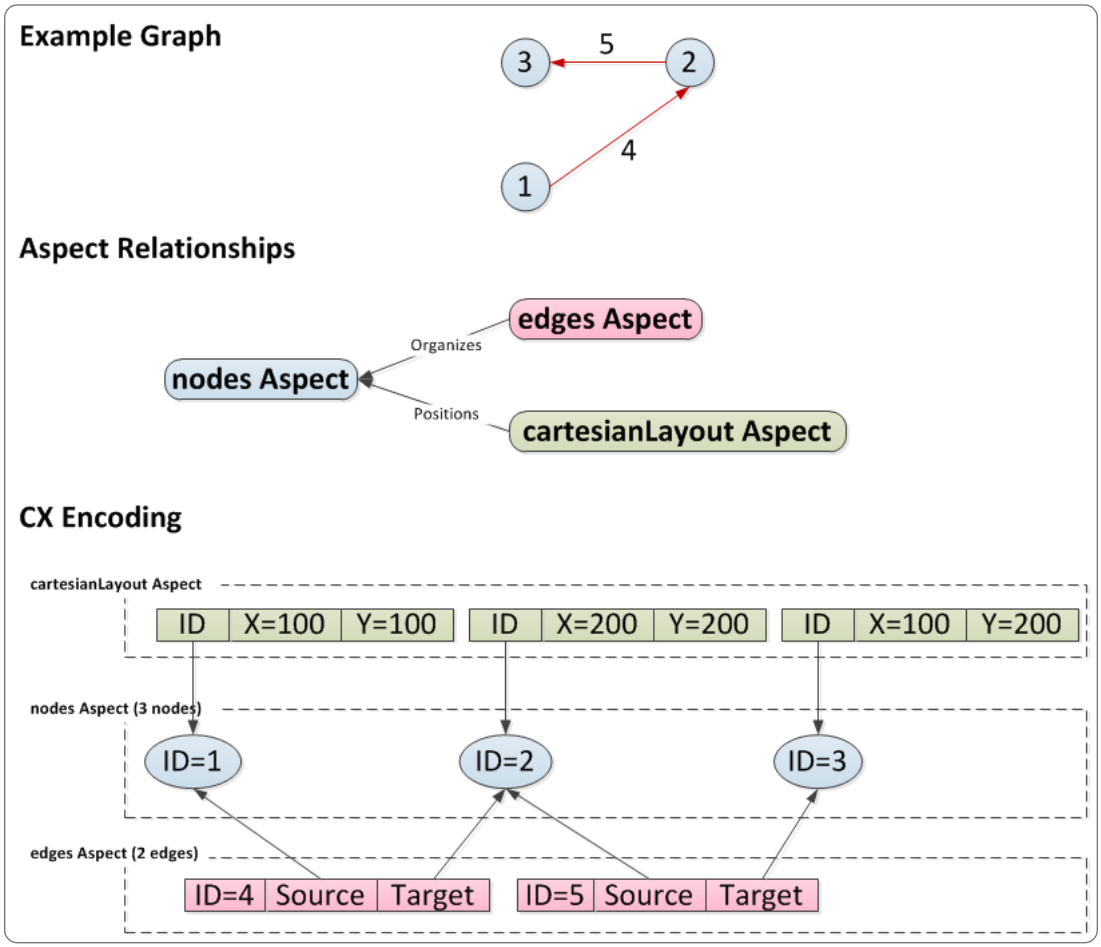
CX is "aspect-oriented", meaning that different types of information about network elements are separated into types of modules ("aspects"). Each aspect type specifies a schema for the information that it contains, typically a set of elements where the elements for that aspect type also have a defined schema. There are design guidelines for dependencies between aspects; the most basic guideline is that dependencies should be simple and minimal.

The flexibility of CX enables straightforward strategies for lossless encoding of potentially any network. At the most basic level, this means that CX imposes very few restrictions: graphs can be cyclic or acyclic and edges are implicitly directed, but formats can choose annotations schemes to override this. CX does not, itself, make any commitment to a single “correct” model of biology or graphic markup scheme.

CX is designed to facilitate streaming, potentially reducing memory footprint burden on applications processing large CX networks. In particular, in a CX stream, the elements of each aspect are broken into fragments and the fragments can be transmitted in any order.

## A Simple Example: A Small Network with Cartesian Layout

Conceptually, the network consists of three nodes, two of which are connected. The nodes aspect (named *nodes*) comprise the base aspect, while the edges (named *edges*) are an aspect composed onto the nodes, as are the cartesian coordinates (named *cartesianLayout*). The “CX Encoding” diagram shows all three aspects, focusing on how the *edges* and *cartesianLayout* annotate *nodes*.




# CX Aspects Present in Most Networks

- "networkAttributes"
  Each element specifies a name-value pair annotating the network
- "nodes"
  - Each element specifies a node in the network
  - Each node must specify its internal integer ID
  - Must also specify either:
    - a node name
    - a node name
    - or both
- "edges"
  - Elements specify edges that connect nodes
  - Each edge must specify:
    - its internal integer ID
    - The IDs of the nodes connected by the edge
  - The element may optionally specifying the type of interaction represented by the edge
- “nodeAttributes”
  - Each element specifies a name-value pair describing one node, identifying the node by its internal integer ID
- “edgeAttributes”
  - Each element specifies a name-value pair describing one edge, identifying the edge by its internal integer ID

The “nodes” and “edges” contain no other information about the network, simply the identifiers of the nodes that the network contains and the edges that connect them. The elements of other aspects typically refer to nodes and edges by their IDs. Some aspects simply annotate the network as a whole. Any reference between aspects by ID is allowed but complex schemas of linked aspect elements are considered bad practice.

Applications are free to make use of relevant aspects while ignoring others. Using a CX network is therefore a process of selection of the desired aspects followed by integration of the aspects in a task appropriate data structure. Aspects ignored by an application are referred to as "Opaque" to the application. Note that CX is not intended to be used as an internal data structure for an application, although some applications may choose to use it this way.

# Aspects and Aspect Elements

Aspects are sets of Aspect Elements, each of the same type. The Aspect defines the data structure for its elements and can hold multiple instances of that data structure. An Aspect essentially amounts to a data type, defining the distinct key values and structure for its elements.

An aspect defined by a community may evolve over time, requiring versioning. This is handled simply by defining a new aspect name. The name can include the version, but specifying the aspect version in pre-metadata would be the preferred method. A program supporting CX must therefore know which aspects it handles by name and version(s).

## Aspect Elements May Refer to Each Other by IDs

An Aspect may define an “@id” property for its elements. This is the id by which the element may be referenced by another element, either in the same aspect or in another aspect.

- Referenceable IDs defined in an aspect must be unique for the aspect for the life of the network.
- All Aspects must define the schema for their elements such that any references to elements are typed: it must explicitly define the Aspect of the referenced element. For example, if an Aspect can refer to both nodes and edges by their ids, it must use different keys to do so, otherwise the references would be ambiguous.
- Note that this typing enables placeholder objects to be created by applications reading CX when they encounter a reference to an element before the element itself while processing the CX stream, updating the placeholder as the full information becomes available.

Aspect Element ids are integers and are assigned based on a monotonically increasing counter stored in the Aspect Metadata.

References between Aspect Elements pose challenges in the case where a program changes a CX network while treating some aspects as opaque. References between modified or deleted elements and elements in the opaque aspects may become inconsistent. For this reason, each application that modifies a CX network should designate the Aspects that it asserts to be consistent after the modifications, using the Aspect Consistency Group metadata (see below).

Note that streaming filters can easily be implemented if the operate on Aspects with elements that aren’t referenced by other elements - elements can be removed incrementally without introducing inconsistencies. But in the case where the elements may be referenced by other elements, the participating Aspects must be loaded into memory to resolve inconsistencies before transmitting the filtered CX.

Note that in a given CX stream of Aspect Fragments, it is not necessary that a referenced element be defined before its reference. References will typically be resolved after the CX stream is parsed, but note that it is not required that the recipient resolve any references unless this is necessary for its function. A program might pass resolve reference only for aspects that it used and simply pass on or ignore all other aspects.

Best practice for a program that stores networks based on CX is that it should resolve all references between supported aspects and should error if there are any references that cannot be resolved, that leave incomplete “placeholder” elements.

### Numbers in Aspect Elements

Aspect elements may use numbers as supported by JSON. They are serialized as strings without quotation and are parsed by all implementations according to the JSON specification:

- A number contains an integer component that may be prefixed with an optional minus sign, which may be followed by a fraction part and/or an exponent part.
- Octal and hex forms are not allowed. Leading zeros are not allowed.
- A fraction part is a decimal point followed by one or more digits.
- An exponent part begins with the letter E in upper or lowercase, which may be followed by a plus or minus sign. The E and optional sign are followed by one or more digits.
- Numeric values that cannot be represented as sequences of digits (such as Infinity and NaN) are not permitted (in contrast, for numerical value attributes, such as nodeAttributes, which are serialized as strings within quotations, "NaN" and "null" are permitted).

The storage of these numbers for a given application parsing CX depends on the JSON implementation used. CX is only supported for implementations that pass the numeric range precondition check - they must correctly store and retrieve integers within that range. Any use of integers should be limited to that range.

An aspect may also be designed to store numbers as strings, placing the responsibility for parsing and otherwise handling these values on any CX implementations that supports the aspect.

# CX Stream Structure

First, a (mandatory) numeric range precondition element is transmitted, followed by an (optional) pre-metadata object, followed by a sequence of aspect fragments, and finally a (optional) post-metadata object is transmitted.

```
[
<numeric range precondition element>
<pre-metadata>,
<aspect-fragment 1>,
….
<aspect-fragment N>,
<post-metadata>
<status aspect>
]
```

## The Numeric Range Precondition Element

The numeric range precondition element is the first element of a CX stream. Its purpose is to ensure that the system handling the stream can correctly process large integers (which are used for identifiers of various CX elements). Currently, **281474976710655** is used as the largest integer a system has to be able to read, write and represent correctly. The Numeric Range Precondition Element is represented by the "numberVerification" aspect. It is a JSON object like this:

```
{
"numberVerification": [
{
"longNumber": 281474976710655
}
]
}
```

## Pre and Post Metadata in CX

The metadata for the CX stream is provided in two JSON objects, the pre-metadata and the post-metadata. The pre-metadata is the first object in the stream after the numberVerification aspect, and the post-metadata is the last object in the stream before the status aspect. They have the following schema, in which they break down into a set of Aspect Metadata objects, one for each Aspect present in the CX:

```
{“metaData” : [ <aspect metadata 1>, <aspect metadata 2>, …]}
...
{“metaData” : [ <aspect metadata 1>, <aspect metadata 2>, …]}
```

## Aspect Fragments

The primary CX will be rendered as a sequence of Aspect Fragments, where each Aspect Fragment holds one or more Aspect Elements. The Aspect Elements contained in all fragments of a given type compose the Aspect.

An Aspect Fragment has a simple schema:

```
{
<aspect name 1> : [<aspect-element 1>, …]
}
```

Structuring a CX network as a stream of Aspect Fragments addresses the design goal to enable incremental processing of CX, promoting the development of network processing services that are memory efficient and potentially parallelizable.

There is no fixed limit on the number of elements an Aspect Fragment can contain, though the number should be “small” -- less than 100 is safe. The larger the Aspect Fragment, the less likely the sender or receiver may have the memory necessary to represent it.

### Status Aspect

A complete CX stream ends with a status aspect object. This aspect tells the recipient if the CX is successfully generated by the source. A use case is that a source is generating a CX network incrementally and transmits the data to the recipient through a stream, when an error occurs during the creation of the CX network, the source can write out a status aspect to the CX stream with some error message in it and finish the CX network immediately. The recipient will know from this aspect that the CX is a bad object and can stop the downstream process on this network. The status aspect has this structure:

```
{
"status" : [ {
"error" : string,
"success" : boolean
} ]
}
```

The “error” field holds the error message when error occurs. “success” fields tells if the cx document is successfully generated by the source.

## Aspect Metadata

As a transfer format, Aspect Metadata is understood in the context of a network owner and a consumer, where the consumer may modify the network and return it to the owner. The owner is considered to have a complete picture of the network including all metadata and aspects associated with the network, and is considered to be authoritative as to the entire network and its parts. While the consumer can ask the owner for all or some of a network’s aspects, for every aspect returned by the owner, the owner must also return the aspect’s metadata. Additionally, for every aspect the consumer sends to the owner, it must supply the aspect’s metadata.

The owner or consumer can send part of each aspect’s metadata as pre-metadata and part as post-metadata -- the aspect’s metadata is considered to be the union of the pre-metadata and post-metadata. Each aspect must have pre-metadata containing at least a name and version field. If an aspect has post-metadata, it must have the same name field as the corresponding pre-metadata. **For optional elements, the sender must include the element values, but is not required to if it doesn’t have values for these elements**. For optional and required fields (besides name and version), the sender is free to send each field as pre- or post-metadata, the choice of which is solely the sender’s convenience. The receiver must be prepared to accept metadata values as pre-metadata, post-metadata, or (for optional fields) not at all.

Besides name, all attributes should only be specified at most one time -- there should be no redundant attributes stated in both pre-metadata and post-metadata. The union of pre-metadata and post-metadata must always be equivalent to the aspect metadata - for example, it is valid for a system to receive CX, merge pre and post metadata and then output CX with all attributes specified as only pre-metadata.

Each aspect may pose its own requirements on which additional attributes are mandatory in the corresponding metadata.

When a sender has the choice of sending an element as either pre- or post-metadata, all things being equal, it should send it as pre-metadata.

The full specification of Aspect Metadata:

```
{
name: string,
version: string,
idCounter: long,
properties: list of name value pairs in JSON object format.
elementCount: long,
consistencyGroup: long ,
checksum: string
}
```

Each Aspect Metadata contains:

- "name"
  - **Required in both pre- and post-metadata**
  - The name of the aspect
- "version"
  - Required in pre-metadata
  - version of this aspect schema
  - Version is defined according to semantic versioning rules. That is, if a service requires aspect A version 1.1.0, it would not accept aspect version 1.0.0 or aspect 2.0.0. Essentially, the minor version number indicates that required values have been added but none have been removed or had semantic meaning changes – optional values don't figure into this discussion. The major version indicates that required values have been removed or had their semantics changed
- "idCounter"
  - Integer (All Element IDs are integers)
  - This is an integer monotonically increasing ID counter used for ID generation, and represents the highest ID represented in the aspect or any of the aspect's previous versions
  - **Required if the aspect exports IDs for its aspect elements**
- "checksum"
  - A CRC/hash
  - **Optional**
  - most likely as post-metadata, Optionally checked by a CX reader
- "elementCount"
  - number (integer) of elements in this aspect
  - **Optional**
- "consistencyGroup"
  - An integer identifier shared by aspects to indicate that they are mutually consistent
  - Used to help track status of the network as different programs operate on different sets of aspects. See discussion of **Aspect Consistency Group IDs** below
  - **Required**
- "properties"
  - An aspect-defined property list
  - **Required**
  - A list of name value pairs in JSON object format. (may be an empty list [])
  - Properties that need to be fetched or updated independently of aspect data

Here is an example of pre-metadata list:

```
[
{
name:”nodes”,
version: “1.0”,
idCounter: 200,
lastUpdate: 1034334343,
elementCount: 32,
properties: [],
consistencyGroup: 1
},
{
name: “Citation”,
Version: “1.0”,
lastUpdate: 1034334343,
consistencyGroup: 1,
properties: [
{“name”: “curator”,
“value”:”Ideker Lab”}
],
}
]
```

## Aspect Consistency Group IDs

When an application modifies a CX network, it may make changes that are inconsistent with the state of aspects that it treated as opaque. Subsequent operations may need to know which aspects are expected to be consistent with each other and which may be inconsistent.

To that end, we define “consistency groups” and in the metadata for each aspect, a consistencyGroup attribute assigns the aspect to one group. All aspects in a group are asserted to be consistent with each other.

An application that modifies a CX network should therefore set the consistency group assignments for the aspects that it changed, added, or can prove to be consistent such that they are distinguished from those that may have become inconsistent.

A simple case is where an aspect is added that only refers to nodes and/or edges. It should be assigned the same consistency group as the nodes and edges.

But, for example, in the case where edges are deleted, the application should also check consistency and potentially edit all non-opaque aspects to ensure that they have no references to deleted edges. All of the modified and checked aspects are assigned a new, unique consistencyGroup attribute. All opaque aspects retain their previous consistencyGroup and can be identified as potentially inconsistent with the nodes, edges, and other aspects.

The consistencyGroup attribute is a monotonically increasing value that increments when a change occurs that could make one or more aspects inconsistent.

- When we add an aspect, it gets tagged with the current consistency group (assuming it's consistent with other aspects).
- When we update an aspect, we increment the consistency group and tag the updated aspect(s) with it.
- Aspects in the same consistencyGroup are consistent with each other while aspects in different groups may potentially reference elements that no longer exist or can in other ways be inconsistent.

# Core Aspects

## nodes

### Mandatory Aspect

Nodes are represented as (possibly multiple) lists of node key-value pairs, preceded by the keyword “nodes”. For nodes the keys are “@id” and the values are the node identifier strings that parse as integers - “node ids”.

Optionally, nodes can have a name (a single string), specified by the "n" key. Nodes can also have an optional represents attribute ( a single string), specified by the “r” key.

All node ids must be unique in the node aspect. A CX writer should not emit a node aspect in which node ids are repeated and it is good practice for any CX reader to test for uniqueness and handle malformed node aspects (either by erroring or by repairing, as appropriate to the application).

The aspect metadata must contain an idCounter representing the highest @id contained in the aspect or any previous version of the aspect as shown in the example below:

```
"nodes": [
{
"@id": 0
},
{
"@id": 1,
"n": "i am node 1",
"r": "HGNC:AKT1"
}
]
```

Condition is that all node ids (as a matter of fact, this applies to all other aspects with ids) are unique (and not “empty”). At this point, it is the responsibility of the writer to make sure this is true, but we could easily add a check for uniqueness to our parser.

## edges

Edges are represented as (possibly multiple) lists of dictionaries, preceded by the keyword “edges”. Edge dictionaries have three keys: “source” and “target” which refer to the node identifiers connected by the edge and, and an identifier which is used by other elements to refer to the edge (the “edge id”).

All edge ids must be unique in the edge aspect. A CX writer should not emit an edge aspect in which edge ids are repeated and it is good practice for any CX reader to test for uniqueness and handle malformed edge aspects (either by erroring or by repairing, as appropriate to the application).

Although a given CX network might not have any aspects that reference edges, the identifiers are always assigned for consistency and to simplify subsequent operations on the network.

Nodes can optionally also have a interaction field, specified by the "i" key.

The aspect metadata must contain an idCounter representing the highest @id contained in the aspect or any previous version of the aspect.

```
"edges": [
{ "@id": 0,
"s": 0,
"t": 1,
"i": "binds"
}
]
```

## nodeAttributes

nodeAttributes store attribute values associated with nodes. Here are the nodeAttributes element attributes:

- mandatory:
  - “po” - property of specifies the node to which the attribute applies.
  - “n” - text - attribute name
  - “v” - string or list of strings - the attribute value(s)
- optional:

- "d" - data type - attribute value data type (default is "string", see below)
- "s" - subnetwork id (see Cytoscape aspects)

```
"nodeAttributes": [
{
"n": "weight",
"po": 74,
"v": [ "2", "0.34", "2.3" ],
"d": "list_of_double"
},
{
"n": "name",
"po": 74,
"v": "Node 6"
},
{
"n": "selected",
"po": 74,
"d": "boolean",
"v": "true",
"s": 366,
}
]
```

## edgeAttributes

edgeAttributes store attribute values associated with edges. Usage:

- Cytoscape supports this aspect for both CX input and output. The edgeAttributes are mapped to the Cytoscape *edge table* for a network.
- NDEx supports this aspect, storing it in a manner that enables queries to networks based on edge attributes.

edgeAttributes element attributes:

- mandatory:
  - “po” - property of specifies the edge to which the attribute applies.
  - “n” - text - attribute name
  - “v” - string or list of strings - the attribute value(s)
- optional:
  - "d" - data type - attribute value data type (default is "string", see below)
  - "s" - subnetwork id (see Cytoscape aspects)

```
"edgeAttributes": [


{
"n": "shared name",
"po": 84,
"v": "Node 5 (undirected) Node 6"
},
{
"n": "name",
"po": 84,
"v": "Node 5 (undirected) Node 6"
},
{
"n": "interaction",
"po": 84,
"v": "undirected"
},
{
"n": "shared interaction",
"po": 84,
"v": "undirected"
},
{
"n": "selected",
"po": 84,
"d": "boolean",
"v": "true",
"s": 366,
}
]
```


## networkAttributes

networkAttributes store attribute values associated with the entire network. networkAttributes element attributes:

- mandatory:
  - “n” - text - attribute name
  - “v” - string or list of strings - the attribute value(s)
- optional:
  - "d" - data type - attribute value data type (default is "string", see below)
  - "s" - subnetwork id (see Cytoscape aspects)


```
"networkAttributes": [
{
"n": "dc:title",
"v": "Result of heat diffusion analysis"
}
]
```

## cartesianLayout

Cartesian layout elements store coordinates of nodes. Usage: Cytoscape supports this aspect for both CX input and output.

cartesianLayout element attributes:

- mandatory:
  - “node” - a node id - specifies the node to which the coordinates apply
    - “x” - number - x coordinate
    - “y” - number - y coordinate
- optional:
  - “z” - number - z coordinate
  - "view" - view id (see Cytoscape aspects)

```
"cartesianLayout": [ {
"node" : 0,
"view" : 1476,
"x" : 5.5,
"y" : 200.3
}, {
"node" : 1,
"view" : 1476,
"x" : 110.1,
"y" : 210.2
} ]
```

# Data types for Attributes

Values for node, edge, network, and hidden attributes are always encoded as JSON strings. Their data type is determined by the value of "d". If the data type "d" field is missing, the default data type of (single!) string is used. The following data types are allowed:

- boolean
- byte
- char
- double
- float
- integer
- long
- short
- string *(default)*
- list_of_boolean
- list_of_byte
- list-of_char
- list_of_double
- list_of_float
- list_of_integer
- list_of_long
- list_of_short
- list_of_string

# NDEx CX Conventions

## Handling of Identifiers

All identifiers should be either:

- <prefix>:<id> format
- a string without ‘ : ’

- a URI

## Network Attributes Treated Specially by NDEx

- “version” - string
  - NDEx will treat this attribute as the version of the network
  - Format is not controlled but best practice is to use string conforming to Semantic Versioning.
- “ndex:sourceFormat” - string
  - used by NDEx to indicate format of an original file imported, can determine semantics as well.
  - The NDEx UI will allow export options based on this value. Applications that alter a network such that it can no longer be exported in the format should remove the value.
- “name” - the name of the network
- “description” - a description of the network

## Node Attributes Treated Specially by NDEx

- “alias” - alternative identifiers for the node
  - same meaning as BioPAX “aliases”
- “relatedTo” - identifiers denoting concepts related to the node
  - same meaning as BioPAX “relatedTerms”

# CX Aspects Defined by the NDEx Project

## ndexStatus

"ndexStatus" is NOT the same as the "status" aspect. The ndexStatus aspect contains attributes that describe the state of the network when it was last stored in NDEx.

ndexStatus element attributes:

- “externalId” - string - the CX network was derived from an NDEx network with this universally unique identifier (UUID) . The CX network may be a complete rendition of all the information that was stored on NDEx or it may be some subset of the network.
- “creationTme” - timeStamp (string) - Time at which the network was created.
- “readOnly” - boolean - Content modification not permitted only if true.
- “visibility” - string - One of PUBLIC, PRIVATE. PUBLIC means it can be found or read by anyone, including anonymous users. PRIVATE is the default, means that it can only be found or read by users according to their permissions.
- integer“nodeCount” - integer - the number of node objects in the networklong<./li>
- “edgeCount” - integer - the number of edge objects in the network.
- “owner” - text - unique name of owner on the server.
- “ndexServerURl” - URI identifying the NDEx server from which the network retrieved.

```
"ndexStatus": {
"externalId": ...,
"nodeCount" : 10,
"edgeCount" : 55,
...
}
```

## citations

citations aspect elements specify literature references or other sources of information that are relevant to the network. Other aspects, such as supports, edgeCitations, or nodeCitations can link them to specific nodes and edges in the network, indicating that the citation supports the assertion represented by the given node or edge.

Citations are primarily described by five dublin core terms (http://purl.org/dc/terms). The “dc” prefix is implicitly interpreted as referencing dublin core in the context of the citations aspect.

The five primary dublin core terms defined for citations are:

- dc:title
  - http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=terms#terms-title (http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=terms#terms-title)
- dc:contributor
  - http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=terms#terms-contributor (http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=terms#terms-contributor)
- dc:identifier
  - http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=terms#terms-identifier (http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=terms#terms-identifier)
  - Ideally citations make use of the “identifier” key to link to their source. This can be a URI, but might also use the “source:id” format, where source is usually “pmid” or “doi”.
- dc:type
  - http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=terms#terms-type (http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=terms#terms-type)
- dc:description
  - http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=elements#description (http://dublincore.org/documents/2012/06/14/dcmi-terms/?v=elements#description)
  - This should be a description of the resource, and it is preferable that information that can be expressed in more specific attributes should not be in this attribute. The “description” attribute could contain the title, authors, and/or journal reference, but in that case it would not be expected to be machine parsable.

Additional attributes can be added to the citation using the “attributes” attribute, a collection of name-value pairs.

The aspect metadata must contain an idCounter representing the highest @id contained in the aspect or any previous version of the aspect.

```
"citations": [
{ "@id": 590,
"dc:identifier": "doi:10.1016/0092-8674(93)80066-N",
"dc:title": "Bcl-2 functions in an antioxidant pathway"
"dc:description": "Article from Cell, Volume 75, Issue 2, p241–251, 22 October 1993 ",
"attributes": [
{
"n" : "name1",
"v" : "value1",
},
{
"n" : "name2",
"v" : "26",
"t" : "integer",
}
]
}
]
```

## nodeCitations

Each nodeCitations element links a collection of node ids to a collection of citation ids.

```
"nodeCitations": [
{
"citations": [590],
"po": [27]
}
]
```

### edgeCitations

Each edgeCitations element links collection of edge ids to a collection of citation ids.

```
"edgeCitations": [
{
"citations": [590],
"po": [24]
}
]
```

## supports

Each support element defines text that can be used to “support” - i.e. provide evidence for - one or more nodes or edges in the network. It optionally can also specify a citation id to indicate that the text is derived from the cited publication or data source. It may also have an optional attribute “attributes” which is a collection of name-value pairs.

The aspect metadata must contain an idCounter representing the highest @id contained in the aspect or any previous version of the aspect.

```
"supports": [
{
"@id": 589,
"citation": 590,
"text": "Bcl-2 protected cells from H2O2- and menadione-induced oxidative deaths",
"attributes": [
{
"n" : "name1",
"v" : "value1",
},
{
"n" : "name2",
"v" : "26",
"t" : "integer",
}
]
}
]
```

## edgeSupports

Each edgeSupports element links a collection of edge ids to a collection of support ids.

```
"edgeSupports": [
{
"supports": [589],
"po": [24]
}
]
```

## nodeSupports

Each nodeSupports element links a collection of node ids to a collection of support ids.

```
"nodeSupports": [
{
"supports": [589],
"po": [27]
}
]
```

## functionTerms

functionTerms aspect elements link nodes with expressions that define the meaning of the node in the network. The expressions compose external vocabulary terms to define concepts. Examples in BEL would include expressions that define complexes, reactions, protein activities and which distinguish proteins vs RNA vs genes.

functionTerms element attributes:

- mandatory
  - “po” - an id - specifies the node which the function term defines.
  - “f’ - string - the function, a name or an external vocabulary term specifying the composing function.
  - “args” - list of strings - ordered argument list to the function. Each argument can be either a literal value (string or number), an external vocabulary term, or a functionTerms expression.

## reiefiedEdges

A reifiedEdges Aspect Element designates that a node represents an edge. This is used to implement logic such as “A increases the inhibition of C by B” where the target of the “increases” edge is intended to be another edge, the “inhibits” edge from B to C.

The reifiedEdges aspect enables representation of this case by designating an additional node “D” to represent “B inhibits C”. The CX edges attribute can then be populated with “A increases D” and “B inhibits C”, a simple topology that should be handled by any network application. Applications that have internal data models supporting edge-edge interactions can parse the reifiedEdges aspect to determine which nodes are “standing in” for edges.

Usage:

- Cytoscape treats this aspect as opaque.
- NDEx supports this aspect, storing it in a manner that enables queries that retrieve subnetworks to include the reifiedEdges elements and edges that apply to the selected nodes.

reifiedEdges element attributes:

- mandatory
  - “edge” - an edge id - specifies the edge represented by the node.
  - “node” - a node id - specifies the node

```
"reifiedEdges": [
{
"node": 72,
"edge": 84,
},
{
"node": 77,
"edge": 88,
},
]
```

## @context

The @context aspect elements are inspired by the (http://www.w3.org/TR/json-ld/#the-context)JSON-LD @context structures that define the vocabularies used in the CX document or stream. **If you want to write a @context aspect in your CX, it has to be the first aspect after pre-metadata**.

@context maps terms to IRIs. Terms are case sensitive.

This is a valuable feature of CX because it enables unambiguous, consistent reference to controlled vocabularies, when referencing both properties and values. It is possible to determine all namespaces and specific terms used in a network in a consistent fashion.

```
"@context": [
{
"MESHD": "http://resource.belframework.org/belframework/1.0/namespace/mesh-diseases.belns",
"BEL": "http://belframework.org/schema/1.0/xbel"
}
]
```

## provenanceHistory

The provenance history aspect of an NDEx network is used to document the workflow of events and information sources that produced the current network. API operations that create or update networks add default events to the provenance history. Applications can also explicitly modify the provenance history in order to customize events, controlling the granularity of events recorded and the level of detail captured.

A provenance history is a tree structure containing ProvenanceEntity and ProvenanceEvent objects. It is serialized as a JSON structure by the NDEx API. The root of the tree structure is a ProvenanceEntity object representing the current state of the network. Each ProvenanceEntity may have a single ProvenanceEvent object that represents the immediately prior event that produced the ProvenanceEntity. In turn, linked to network of ProvenanceEvent and ProvenanceEntity objects representing the workflow history that produced the current state of the Network. The provenance history records significant events as Networks are copied, modified, or created, incorporating snapshots of information about “ancestor” networks.

### ProvenanceEntity

- uri
  - URI of the resource described by the ProvenanceEntity
  - This field will not be set in some cases, such as a file upload or an algorithmic event that generates a network without a prior network as input
- creationEvent
  - ProvenanceEvent
  - has semantics of PROV:wasGeneratedBy
- properties
  - array of SimplePropertyValuePair objects

### ProvenanceEvent

- endingAtTime
  - timestamp

    - has semantics of PROV:endingAtTime
- startingAtTime
    - timestamp
    - has semantics of PROV:startingAtTime
- inputs
    - array of ProvenanceEntity objects
    - has semantics of PROV:used
- properties
    - array of SimplePropertyValuePair objects

### Properties of ProvenanceEntity and ProvenanceEvent objects

The standard fields in ProvenanceEntity and ProvenanceEvent objects correspond to relationships defined in the PROV-O ontology. Other property-value pairs can annotate these objects to provide more information about the entities and events. Any ad hoc pair of strings can be added as a property-value pair, and the properties used may be idiosyncratic to the recorded events and entities. However, the use of properties defined in the Dublin Core (DC) metadata annotations and the Provenance, Authoring and Versioning ontology (PAV) are preferred when applicable.

It is important to note a difference in the use of these ontologies in an NDEx provenance structure and the original intent. A ProvenanceEntity is a description of the referenced object, not the object itself. Therefore, a property such as “dc:title” that is asserted for a ProvenanceEntity refers to the original entity that the ProvenanceEntity represents. The provenance history references ancestor networks and other data sources but can also include self-contained descriptions of those objects that capture their state at the time they were used.

### Dublin Core (DC) Properties

- dc:title
- dc:description
- dc:rights
- dc:rightsHolder
- dc:format

### PAV Properties

- pav:retrievedFrom
    - Direct retrieval - a COPY of the source network with no transformation of the content.
- pav:importedFrom
    - Import with some transformation, as in a file UPLOAD where the source data is processed to create the network.
    - The content reflects the external source but potentially has differences dependent on the import method.
- pav:derivedFrom
    - The network was generated by an operation that transforms the content of the source.
- pav:sourceAccessedAt
    - The network was generated by a transformation operation that consulted the source as part of the transformation.
- pav:version
    - The version of the current network.
- pav:previousVersion
    - The previous version. Note that this might be the version of a network that is not in the provenance history - a version could be created from new sources, not necessarily as a transformation of an earlier version.

# Aspects Defined by Cytoscape

Cytoscape contributes aspects that organize subnetworks, attribute tables, and visual attributes for use by its own layout and analysis tools. This section describes the Cytoscape aspects, which are distinguished by the “cy” prefix on the aspect name.

In addition, Cytoscape extends a number of aspects defined already for the root network. It adds a member that identifies either the Cytoscape subnetwork or view object to which an aspect element applies. If the member is not present in an element, the element refers to the root network. The following lists the affected root aspects and the optional Cytoscape member:

| Root Aspect | Member | Referent |
|---|---|---|
| nodeAttributes | 's' | subnetwork |
| edgeAttributes | 's' | subnetwork |
| networkAttributes | 's' | subnetwork |
| cartesianLayout | 'view' | view |

## cyGroups

This is used to represent Cytoscape "groups". The datafields are:

- @id: the identifier of this group
- view: the view this group is associated with
- name: the name of this group
- nodes: the nodes making up the group - a list of node ids
- external_edges: the external edges making up the group - a list of edge ids
- internal_edges: the internal edges making up the group - a list of edge ids

The aspect metadata must contain an idCounter representing the highest @id contained in the aspect or any previous version of the aspect.

```
"cyGroups" : [ {
"@id" : 1501,
"view" : 1476,
"name" : "Group One",
"nodes" : [ 167, 165 ],
"external_edges" : [ 172 ],
"internal_edges" : [ 171, 170 ]
} ]
```

## cyViews

This is used to represent Cytoscape "views". The datafields are:

- @id: the identifier of the view
- s: the sub-network id

The aspect metadata must contain an idCounter representing the highest @id contained in the aspect or any previous version of the aspect.

```
"cyViews" : [ {
"@id" : 1476,
"s" : 147
} ]
```

## cyVisualProperties

This is used to represent Cytoscape visual properties. The datafields are:

- properties_of: to indicate the element type these properties belong to (allowed values are: "network", "nodes:default", "edges:default", "nodes", "edges")
- applies_to: the identifier of the element these properties apply to
- view: the view these properties are associated with
- properties: key-value (string, string) pairs of the actual properties
- dependencies: key-value (string, string) pairs of the dependencies
- mappings: key-value pairs (string, dictionary), in the form: `"PROPERTY" : { "type" : "the mapping type", "definition" : "the actual mapping" }`

This is an example of cyVisualProperties:

```
"cyVisualProperties" : [ {
"properties_of" : "nodes:default",
"applies_to" : 1476,
"view" : 1476,
"properties" : {
"NODE_BORDER_STROKE" : "SOLID",
"NODE_BORDER_WIDTH" : "1.5"
},
"dependencies" : {
"nodeCustomGraphicsSizeSync" : "true",
"nodeSizeLocked" : "false"
},
"mappings" : {
"NODE_FILL_COLOR" : {
"type" : "CONTINUOUS",
"definition" : "COL=gal1RGexp,T=double,L=0=#0066CC,E=0=#0066CC,G=0=#0066CC,OV=0=-2.426,L=1=#FFFFFF,E=1=#FFFFFF,G=1=#FFFFFF,OV=1=1.225471493171426E-
7,L=2=#FFFF00,E=2=#FFFF00,G=2=#FFFF00,OV=2=2.058"
},
"NODE_LABEL" : {
"type" : "PASSTHROUGH",
"definition" : "COL=COMMON,T=string"
}
}
}, {
"properties_of" : "nodes",
"applies_to" : 42,
"view" : 1476,
"properties" : {
"NODE_FILL_COLOR" : "#FF3399"
}
} ]
```


## cyHiddenAttributes

This is used to represent Cytoscape "hidden" attributes. The datafields are:

- s: the identifier of the subnetwork this hidden attribute belongs to (optional, if missing applies to root-network)
- n: the name of this hidden attribute
- v: the actual value(s) - either a single value or a list
- d: the datatype of this hidden attribute (optional, defaults to "string")

The aspect metadata must contain an idCounter representing the highest @id contained in the aspect or any previous version of the aspect.


```
"cyHiddenAttributes" : [ {
"s" : 52,
"n" : "layoutAlgorithm",
"v" : "Prefuse Force Directed Layout"
} ]
```

## cyNetworkRelations

This is used to represent Cytoscape network relations. The datafields are:

- p: the parent network (optional, if missing parent is root-network)
- c: the child network
- r: the relationship type ("view", "subnetwork") (optional, if missing, default is “subnetwork”)
- name: the name of the child network (optional, if missing, default is reader-dependent)

```
"cyNetworkRelations" : [ {
"c" : 147,
"r" : "subnetwork",
"name" : "Network A"
}, {
"p" : 147,
"c" : 1476,
"r" : "view",
"name" : "Network A view"
} ]
```

## cySubNetworks

This is used to represent subnetworks. The datafields are:

- @id: the identifier of this subnetwork
- edges: the edges making up this subnetwork - list of edge identifiers, can be "all"
- nodes: the nodes making up this subnetwork - list of node identifiers, can be "all"

```
"cySubNetworks" : [ {
"@id" : 147,
"nodes" : [ 167, 165, 163, 161, 159 ],
"edges" : [ 172, 178, 171, 170, 176 ]
} ]
```

## cyTableColum

These elements are used to represent Cytoscape table column labels and types. Its main use is to disambiguate empty table columns. The datafields are:

- s: the identifier of the subnetwork this table column belongs to (optional, if missing applies to root-network -- Cytoscape does not currently support table columns for the root network, but this is option is included here for consistency)
- n: the name of the table column
- d: the datatype of table column (optional, defaults to "string")
- applies_to: indicates whether this applies to "node_table", "edge_table", or "network_table"

```
"cyTableColumn" : [ {
"s" : 366,
"applies_to" : "node_table",
"n" : "weight",
"d" : "double"
} ]
```