# Annotation guide: Annotating workflows using .graphml in yEd

This document is meant as a guide for annotating GIS-workflows with the use of .graphml. This is a non-proprietary XML-based format which can be accessed in multiple editors. This guide will explain how to annotate workflows in the proprietary yWorks yEd environment, but other editors should also work. The aim is to produce workflows that can be parsed using a custom python script.

## Actions and artefacts

Workflows are annotated as directed acyclic graphs (DAG) with two kinds of nodes, namely actions and artefacts. Each action and artefact consists of a node, its inputs and outputs, and additional metadata

**Actions** are nodes that represent implementations of GIS-tools. For example, a buffer analysis around a road can be represented as an action that has a node representing the road input, a node representing the buffer process, and a node representing the road\_buffer output. An action can only have a single process node, but an unlimited number of inputs and outputs. Additionally, an action may have expression and comment nodes feeding into them. Expression nodes may contain information regarding the transformation process, e.g., as a CCT-transformation. The label is exported as a string. Comments are similar, but provide a means for adding any kind of supplementary information. The creation of an action object in Python requires the specifications of the graph in .graphml shown in Table 1. An example graph is shown in Figure 1. The relations between actions on the one hand and comments and expressions on the other are many-to-many.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table 1: Specifications for actions in .graphml** | | | |
| **Entity** | **Node** | **Edges** | **RDF-type** |
| Action | shape\_type = roundrectangle | - | URI-Node |
| Input | shape\_type = parallelogram | Arrow from input to action | URI-Node |
| Output | shape\_type = parallelogram | Arrow from action to output | URI-Node |
| Comment | shape\_type = fatarrow | Arrow from comment to action | String literal |
| Expression | shape\_type = octagon | Arrow from expression to action | String literal |
| Label | shape\_type = hexagon | Arrow from label to action | String literal |

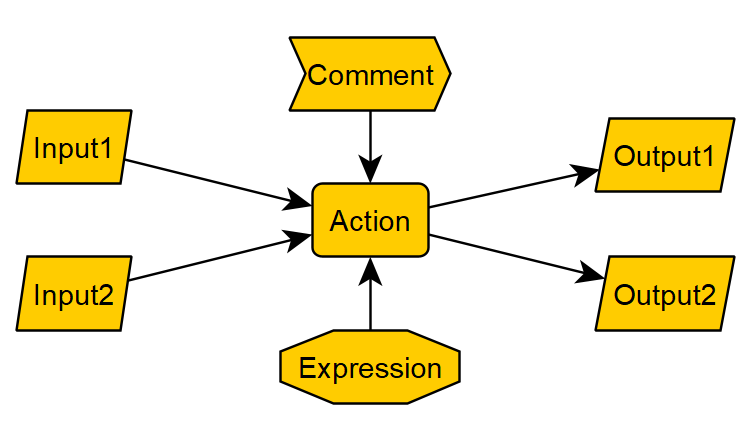


Figure ：Example of an action in .graphml

**Artefacts** are nodes that represent implementations of GIS-data. For example, the road and buffered area around the road from before are both artefacts. Artefacts can be modeled by having the actions that generated them as inputs and the actions that use them to generate other artefacts as outputs. Artefacts may have signatures and comments attributed to them, similar to the comments and expressions of actions. However, unlike the expression, the signature is not a string literal, but a URI-node. The specifications of artefacts are shown in Table 2. An example is shown in Figure 2.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table 1: Specifications for artefacts in .graphml** | | | |
| **Entity** | **Node** | **Edges** | **RDF-type** |
| Artefact | shape\_type = roundrectangle | - | URI-Node |
| Input | shape\_type = parallelogram | Arrow from input to action | URI-Node |
| Output | shape\_type = parallelogram | Arrow from action to output | URI-Node |
| Comment | shape\_type = fatarrow | Arrow from comment to action | String literal |
| Signature | shape\_type = octagon | Arrow from signature to action | URI-Node |
| Label | shape\_type = hexagon | Arrow from label to artefact | String literal |

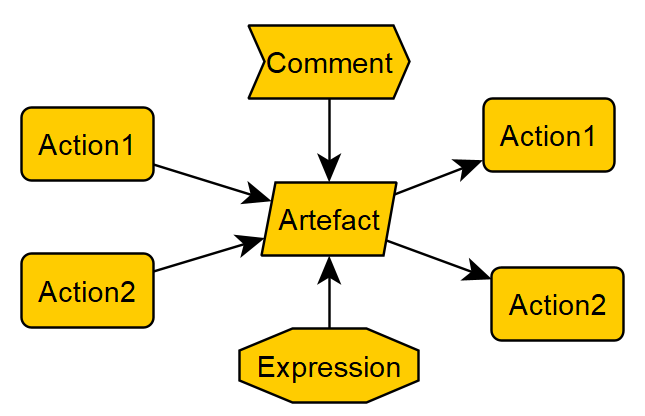


Figure : Example of an artefact in .graphml

Although it could be that in practice an artefact has at most one action as input, there is technically no restriction to having multiple input actions. Because expressions, signatures, and comments are many-to-many related with either or both actions and artefacts, it is possible to use one comment, signature, or expression node to describe multiple artefacts or actions. Technically, it is possible for a node to be both a signature and an expression, so be careful when assigning these. Note that beyond the shape\_files and labels, the style has no influence on the parsing. It may for example be nice to pick custom colors to distinguish different kinds of nodes.

## Workflow metadata

The metadata can be stored as a disjoint graph in the same .graphml file as the action- and artefact-graph. Metadata nodes should not have any of the shapes that are used for actions and artefacts (i.e., not roundrectangle, parallelogram, fatarrow, octagon). There needs to be a node that represents the workflow itself. The label is used as a prefix for nodes and as a filename when the graph is exported to RDF. It is also used to refer to the repository using ‘repo:’. Edge labels denote predicates of the triples, i.e., they explain the relation between the workflow node and other nodes. Currently, it is possible to add four kinds of metadata points, namely a comment to denote a **question** that the workflow is meant to answer, an **abstract** of what happens in the workflow, a **subject**, which I actually don’t know what it is for. Finally, it is necessary to add an **author** node. In addition, based on the structure of the DAG, nodes representing source data, the first data occurring in the workflow, and relations between the action nodes and the workflow node are automatically added. The specifications of the metadata are shown in Table 3, and an example is shown in figure 3.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table 1: Specifications for metadata in .graphml** | | | |
| **Entity** | **Node** | **Edges** | **RDF-type** |
| Workflow core | shape\_type = NOT parallelogram, fatarrow, or octagon | Arrow from author to workflow core with label ‘Author’ | URI-Node |
| Author | shape\_type = NOT parallelogram, fatarrow, or octagon | Arrow from author to workflow core with label ‘Author’ | String literal (Separate by ‘,’ for multiple authors) |
| Question | shape\_type = NOT parallelogram, fatarrow, or octagon | Arrow from question to workflow core with label ‘Question’ | String literal |
| Abstract | shape\_type = NOT parallelogram, fatarrow, or octagon | Arrow from abstract to workflow core with label ‘Abstract’ | String literal |
| Subject | shape\_type = NOT parallelogram, fatarrow, or octagon | Arrow from subject to workflow core with label  ‘Subject’ | String literal |
| Source | - | - | URI-Node |
| Edge | - | - | URI-Node |

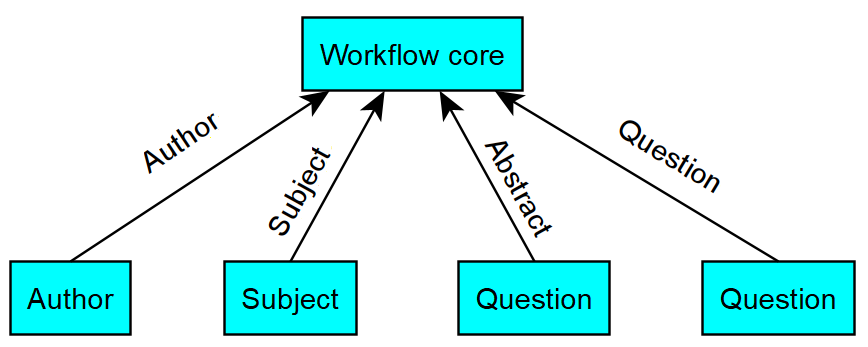


Figure : Example of metadata in .graphml