Node and Edge Types

# Node Types

Each node in the graph has one of four types:

* File: nodes of this type represent files in the source tree.
* Directory: nodes of this type represent directories in the source tree.
* GnatProject: nodes of this type represent projects as visible in the GNAT Studio IDE.
* AdaDeclaration: nodes of this type represent declarations that occur in the Ada body and specification files in the source tree.

Each node of the type File also can have a subtype, which identifies the type of the file. The subtype can be one of the following:

* AdaBodyFile: the file is an Ada body file, i.e., a file with the .adb extension.
* AdaSpecificationFile: the file is an Ada specification file, i.e., a file with the .ads extension.
* CHeaderFile: the file is a C source file, i.e., a file with the .h extension. These files occur in the graph, as they can be referenced from GNAT projects. The C definition and declarations from these file are currently not represented in the graph.
* CSourceFile: the file is a C source file, i.e., a file with the .c extension. These files occur in the graph, as they can be compiled by GNAT projects. The C definition and declarations from these file are currently not represented in the graph.
* GnatProjectFile: the file is a GNAT project file, i.e., a file with the .gpr extension containing a the details of a GNAT project.

File and Directory nodes have two properties:

* fullyQualifiedName: this property specifies the path of the file relative to the root of the source tree, including the file name.
* relativeName: this property specifies the name of the file.

Similarly, GnatProject nodes have a fullyQualifiedName and relativeName property. In the case of GnatProject nodes the properties are both equal to the name of the project.

Each node of the type AdaDeclaration also has a subtype, which provides additional information regarding the kind of the declaration. The subtype is one if the following:

* AdaComponentDeclaration: a component (field) of either a record or protected object type.
* AdaDiscriminantDeclaration: a discriminant (field) of a record type.
* AdaEntryDeclaration: an entry declaration of either a task or protected object type.
* AdaEnumLiteralDeclaration: a literal declared as part of an enumeration.
* AdaExceptionDeclaration: an exception declaration.
* AdaNumberDeclaration: a declaration of a named number, i.e., a declaration of the form “identifier : constant := expression”
* AdaObjectDeclaration: an object (i.e., variable) declaration.
* AdaPackageDeclaration: a package specification or body declaration.
* AdaProtectedDeclaration: a declaration of a projected object type or a single protected object declaration.
* AdaSubprogramDeclaration: a declaration of a (generic) subprogram (i.e., function or procedure). Nodes of this type have a Boolean “isMainProgram” property, which indicates whether the subprogram declaration is an application entry point.
* AdaTaskDeclaration: a declaration of a task type or a single task declaration.
* AdaTypeDeclaration: any type declaration (i.e., record, subtype, …) which is not a task or protected object type declaration.

AdaDeclaration nodes have four properties:

* fullyQualifiedName: this property specifies the fully qualified name of the declaration, i.e., the name of the declaration including the names of all parent declarations.
* relativeName: this property specifies the name of the declaration, excluding names of all parent declarations.
* sourceLocation: this property specifies the source location of the declaration as a string of the form “start-end”, where both “start” and “end” are pairs consisting of a line number and offset from the start of the line.
* isFormalParameter: this Boolean property will be true if the declaration is a formal parameter of a generic subprogram or package declaration.

#### Uniquely Identifying Nodes

To be able to uniquely identify any node of any type, each node also has a name property. Only the uniqueness of the name property can be depended upon, the value itself is subject to change.

# Edge Types

Each edge in the graph has a single type. The types of edges that can occur between nodes depend on the node types. Note that the edge types Contains can have a Directory, a GNATProject and an AdaDeclaration as source node. The edge types DerivesFrom, Imports, and Source can both have a GNATProject and an AdaDeclaration as source node.

There is one edge type that relates Directory nodes with other Directory nodes and File nodes:

* (n)-[:Contains]->(m): n is a directory containing m, where m is another directory or a file (note that Contains is also used in the context of GNAT projects and Ada declarations).

There is one edge type that relates GnatProject nodes with other kinds of nodes:

* (n)-[:Imports]->(m): n is a project (.gpr) file which contains a “with”-clase referring to a project file containing a GNAT project m (note that Import is also used in the context of Ada declarations).

There are several edge types that relate GnatProject nodes:

* (n)-[:Compiles]->(m): n is a GNAT project, which when built, will compile m where m is a Ada body or specification file or a C source file.
* (n)-[:Contains]->(m): n is a GNAT project containing m, where m is a Ada body or specification file or a C source or header file (note that Contains is also used in the context of directories and Ada declarations).
* (n)-[:DerivesFrom]->(m): n is a GNAT project extending the GNAT project m (note that DerivesFrom is also used in the context of Ada declaration).
* (n)-[:Source]->(m): n is a GNAT project and m is the project (.gpr) file in which the project is declared (note that Source is also used in the context of Ada declarations).

A number of edge types relate AdaDeclaration nodes with other kinds of nodes:

* (n)-[:Imports]->(m): n is an Ada body (.adb) or specification (.ads) file which contains a “with”-clause referring to the declaration m (note that Import is also used in the context of GNAT projects).
* (n)-[:Source]->(m): n is a declaration and m is the Ada body (.adb) or specification (.ads) file in which the declaration occurs (note that Source is also used in the context of GNAT projects).

A number of edge types relate two AdaDeclaration nodes:

* (n)-[:Contains]->(m): m is declared in the context of the declaration n, in other words, the declaration of m is nested in the declaration of n (note that Contains is also used in the context of directories and GNAT projects).
* (n)-[:DerivesFrom]->(m): m is a type or interface declaration, and n is a type declaration that derives from m (if m is a type) or that provides m (if m is an interface).
* (n)-[:Instantiates]->(m): n is an instantiation of a generic package or subprogram m.
* (n)-[:IsImplementedBy]->(m): m is a declaration with a body (i.e., subprogram body, package body, …) which implements n (i.e., subprogram specification, package specification, …). If n is an entry declaration of a task, then m is any declaration containing an accept statement for that entry.
* (n)-[:IsOverriddenBy]->(m): m is a subprogram which overrides the subprogram m.
* (n)-[:IsParentOf]->(m): m is a child package of the package n.
* (n)-[:IsPrimitiveSubprogramOf]->(m): n is a primitive subprogram of the type m.
* (n)-[:References]->(m): m is a declaration referenced by the declaration n.
* (n)-[:Renames]->(m): n is a declaration renaming the declaration m

Subprogram Calls

There is one edge type related to subprogram calls:

* (n)-[:Calls]->(m): n is a subprogram calling the subprogram m. Edges of this type have a Boolean “isDispatching” property, which indicates whether the call is dispatching or not (i.e., whether in actual fact a subprogram overriding m may be called instead of m).

## Types

There is one edge type related to the types of components, discriminants, and objects:

* (n)-[:hasType]->(m): n is a component, discriminant, or object of type “m”. Edges of this type have Boolean “hasAccessType” and “hasArrayType” properties, which indicate whether the component, discriminant, or object is actually an anonymous access type “access m” or an anonymous array type “array … of m”.

There are two edge types related to the parameter and return types of subprograms:

* (n)-[:hasParameterOfType]->(m): n is a subprogram that has a parameter of type “m”. Edges of this type have a Boolean “hasAccessType” property, which indicates whether the parameter is actually an anonymous access type “access m”.
* (n)-[:hasReturnOfType]->(m): n is a subprogram that has return type “m”. Edges of this type have a Boolean “hasAccessType” property, which indicates whether the return type is actually an anonymous access type “access m”.