Package 'pegasus'

February 20, 2024

Type Package

Title API for Generating Pegasus Abstract Workflows

Version 5.0

Author Rafael Ferreira da Silva [aut, cre]

Maintainer Rafael Ferreira da Silva <rafsilva@isi.edu>

Description The classes in this package can be used to generate DAXes that can be read by Pegasus.

License Apache License (== 2.0)

URL https://pegasus.isi.edu

RoxygenNote 7.1.1

NeedsCompilation no

R topics documented:

AddArguments — *Add one or more arguments to the job*

Description

Add one or more arguments to the job (this will add whitespace).

Usage

```
AddArguments(job, arguments)
```

Arguments

job	Job object
arguments	List of arguments defined as `list()`

Value

Job with appended list of arguments

See Also

Job

AddChild — *Append a child element to a parent element*

Description

Append a child element to a parent element

Usage

```
AddChild(element, child)
```

Arguments

element	parent element
child	element to be appended to the parent element

Value

parent element with the appended child

AddDAG — *Add a sub-DAG (synonym for addJob)*

Description

Add a sub-DAG (synonym for addJob)

Usage

```
AddDAG(workflow, dag)
```

Arguments

workflow	Workflow object
dag	Sub-DAG to be appended

Value

The Workflow object with the sub-DAG appended

See Also

Workflow, AddJob, DAG

AddDAX — *Add a sub-DAX (synonym for addJob)*

Description

Add a sub-DAX (synonym for addJob)

Usage

```
AddDAX(workflow, dax)
```

Arguments

workflow	Workflow object
dax	Sub-DAX to be appended

Value

The Workflow object with the sub-DAX appended

See Also

Workflow, AddJob, DAX

AddDependency *Add a dependency to the workflow*

Description

Add a dependency to the workflow

Usage

```
AddDependency(workflow, dep)
```

Arguments

workflow	The Workflow object
dep	The dependency object

Value

The Workflow object containing the dependency

See Also

Workflow, Dependency, Depends, RemoveDependency

AddExecutable *Add an executable to the Workflow*

Description

Add an executable to the Workflow

Usage

```
AddExecutable(workflow, executable)
```

Arguments

workflow	Workflow object
executable	Executable object

Value

The Workflow object with the executable appended

See Also

Workflow, Executable, RemoveExecutable, ClearExecutables

AddFile *Add a file to the Workflow*

Description

Add a file to the Workflow

Usage

```
AddFile(workflow, file)
```

Arguments

workflow	Workflow object
file	File object

Value

The Workflow object with the file appended

See Also

Workflow, File, RemoveFile, ClearFiles

AddInvoke.DAG *Add an invoke to the object*

Description

Add an invoke to the object

Usage

```
## S3 method for class 'DAG'
AddInvoke(obj, invoke)

## S3 method for class 'DAX'
AddInvoke(obj, invoke)

## S3 method for class 'Executable'
AddInvoke(obj, invoke)

AddInvoke(obj, invoke)

## S3 method for class 'Job'
AddInvoke(obj, invoke)

## S3 method for class 'Transformation'
AddInvoke(obj, invoke)

## S3 method for class 'Workflow'
AddInvoke(obj, invoke)
```

Arguments

obj	Object to append the invoke
invoke	The invocation

Value

The object containing the invoke

See Also

DAG

DAX

Executable

Invoke

Job

Transformation

Workflow

AddInvokeMixin *Add invoke to the InvokeMixin object*

Description

Add invoke to the InvokeMixin object

Usage

```
AddInvokeMixin(invoke.mixin, invoke)
```

Arguments

invoke.mixin	InvokeMixin object
invoke	invocation to be appended to the list of invocations

Value

InvokeMixin object with invocation appended to the list of invocations

See Also

InvokeMixin

AddJob *Add a job to the Workflow*

Description

Add a job to the Workflow

Usage

```
AddJob(workflow, job)
```

Arguments

workflow	Workflow object
job	Job object

Value

Workflow object with the job appended

See Also

Workflow, RemoveJob

AddPFN.Executable *Add a PFN to the object*

Description

Add a PFN to the object

Usage

```
## S3 method for class 'Executable'
AddPFN(obj, pfn)

## S3 method for class 'File'
AddPFN(obj, pfn)

AddPFN(obj, pfn)
```

Arguments

obj	Object to append the PFN
pfn	The PFN

Value

The object containing the PFN

See Also

Executable

File

PFN

AddProfile.DAG *Add a profile to the object*

Description

Add a profile to the object

Usage

```
## S3 method for class 'DAG'
AddProfile(obj, profile)

## S3 method for class 'DAX'
AddProfile(obj, profile)

## S3 method for class 'Executable'
AddProfile(obj, profile)

## S3 method for class 'File'
AddProfile(obj, profile)

AddProfile(obj, profile)

## S3 method for class 'Job'
AddProfile(obj, profile)

## S3 method for class 'PFN'
AddProfile(obj, profile)
```

Arguments

obj	Object to append the profile
profile	The profile

Value

The object containing the profile

See Also

DAG

DAX

Executable

File

Profile
Job
PFN

AddProfileMixin *Add a profile to the object*

Description

Add a profile to the object

Usage

```
AddProfileMixin(profile.mixin, profile)
```

Arguments

profile.mixin	Profile_mixin object
profile	Profile object to be added

Value

The profile_mixin object with the profile appended

See Also

Profile

AddTransformation *Add a transformation to the Workflow*

Description

Add a transformation to the Workflow

Usage

```
AddTransformation(workflow, transformation)
```

Arguments

workflow	Workflow object
transformation	Transformation object

Value

The Workflow object with the transformation appended

See Also

Workflow, Transformation, RemoveTransformation, ClearTransformations

AppendToList *Append a value to a list*

Description

Append a value to a list

Usage

```
AppendToList(list, value)
```

Arguments

list	List of element to where the value will be appended
value	Value to be added to the list

Value

List with value appended

ClearArguments *Removes all arguments from the job*

Description

Removes all arguments from the job

Usage

```
ClearArguments(job)
```

Arguments

job	Job object

Value

Job with no arguments

See Also

Job

ClearDependencies *Remove all dependencies*

Description

Remove all dependencies

Usage

```
ClearDependencies(workflow)
```

Arguments

workflow	The Workflow object

Value

The Workflow object with no dependencies

See Also

Workflow, Dependency, Depends, AddDependency, RemoveDependency

ClearExecutables *Remove all executables*

Description

Remove all executables

Usage

```
ClearExecutables(workflow)
```

Arguments

workflow	Workflow object

Value

The Workflow object with no executables

See Also

Workflow, Executable, AddExecutable, RemoveExecutable

ClearFiles *Remove all files*

Description

Remove all files

Usage

```
ClearFiles(workflow)
```

Arguments

workflow	Workflow object

Value

The Workflow object with no files

See Also

Workflow, File, AddFile, RemoveFile

ClearInvokes *Remove all Invoke objects*

Description

Remove all Invoke objects

Usage

```
ClearInvokes(invoke.mixin)
```

Arguments

invoke.mixin	InvokeMixin object

Value

InvokeMixin with no invocations

See Also

InvokeMixin

ClearJobs *Remove all jobs*

Description

Remove all jobs

Usage

```
ClearJobs(workflow)
```

Arguments

workflow	Workflow object

Value

The Workflow object with no jobs

See Also

Workflow, RemoveJob, AddJob

ClearTransformations *Remove all transformations*

Description

Remove all transformations

Usage

```
ClearTransformations(workflow)
```

Arguments

workflow	Workflow object

Value

The Workflow object with no transformations

See Also

Workflow, Transformation, AddTransformation, RemoveTransformation

DAG *This job represents a sub-DAG that will be executed by the workflow*

Description

This job represents a sub-DAG that will be executed by the workflow

Usage

```
DAG(file, id = NULL, node.label = NULL)
```

Arguments

file	The logical name of the DAG file, or the DAG File object
id	The ID of the DAG job [default: autogenerated]
node.label	The label for this job to use in graphing

Details

The name argument can be either a string, or a `File` object. If it is a `File` object, then this job will inherit its name from the `File` and the `File` will be added in a `<uses>` with `transfer=TRUE`, `register=FALSE`, and `link=input`.

Value

The sub-DAG job

Examples

```
dagjob1 <- DAG(file="foo.dag")
dagfile <- File("foo.dag")
dagjob2 <- DAG(dagfile)
```

DAX *This job represents a sub-DAX that will be planned and executed by the workflow*

Description

This job represents a sub-DAX that will be planned and executed by the workflow

Usage

```
DAX(file, id = NULL, node.label = NULL)
```

Arguments

file	The logical name of the DAX file or the DAX File object
id	The id of the DAX job [default: autogenerated]
node.label	The label for this job to use in graphing

Details

The name argument can be either a string, or a `File` object. If it is a `File` object, then this job will inherit its name from the `File` and the `File` will be added in a `<uses>` with `transfer=TRUE`, `register=FALSE`, and `link=input`.

Value

The sub-DAX job

Examples

```
daxjob1 <- DAX("foo.dax")
daxfile <- File("foo.dax")
daxjob2 <- DAX(daxfile)
```

Dependency *A dependency between two nodes in the ADAG*

Description

A dependency between two nodes in the ADAG

Usage

```
Dependency(parent, child, edge.label = NULL)
```

Arguments

`parent`	The parent job/dax/dag or id
`child`	The child job/dax/dag or id
`edge.label`	A label for the edge (optional)

Value

Dependency object between parent and child

Depends *Add a dependency to the workflow*

Description

Add a dependency to the workflow

Usage

```
Depends(workflow, child, parent, edge.label = NULL)
```

Arguments

workflow	The Workflow object
child	The child job/dax/dag or id
parent	The parent job/dax/dag or id
edge.label	A label for the edge (optional)

Value

The Workflow object with the dependency appended

See Also

Workflow, Dependency, AddDependency

Element *Representation of an XML element for formatting output*

Description

Representation of an XML element for formatting output

Usage

```
Element(name, attrs = list())
```

Arguments

name	element name
attrs	list of element attributes

Value

an element object

Escape *Escape special characters in XML*

Description

Escape special characters in XML

Usage

```
Escape(text)
```

Arguments

text	Text to be escaped

Value

Escaped special character

`Executable` *An entry for an executable in the DAX-level replica catalog*

Description

An entry for an executable in the DAX-level replica catalog

Usage

```
Executable(
  name,
  namespace = NULL,
  version = NULL,
  arch = NULL,
  os = NULL,
  osrelease = NULL,
  osversion = NULL,
  glibc = NULL,
  installed = NULL
)
```

Arguments

`name`	Logical name of executable
`namespace`	Executable namespace
`version`	Executable version
`arch`	Architecture that this exe was compiled for
`os`	Name of os that this exe was compiled for
`osrelease`	Release of os that this exe was compiled for
`osversion`	Version of os that this exe was compiled for
`glibc`	Version of glibc this exe was compiled against
`installed`	Is the executable installed (true), or stageable (false)

Value

The executable object for the program

See Also

`AddExecutable`

Examples

```
grep <- Executable("grep")
grep <- Executable(namespace="os",name="grep",version="2.3")
grep <- Executable(namespace="os",name="grep",version="2.3",arch=Pegasus.Arch$X86)
grep <- Executable(namespace="os",name="grep",version="2.3",arch=Pegasus.Arch$X86,os=Pega
```

`File` *A file entry for the DAX-level replica catalog, or a reference to a logical file used by the workflow*

Description

All arguments specify the workflow-level behavior of this File. Job-level behavior can be defined when adding the File to a Job's uses. If the properties are not overridden at the job-level, then the workflow-level values are used as defaults.

If this LFN is to be used as a job's stdin/stdout/stderr then the value of link is ignored when generating the <std*> tags.

Usage

```
File(name)
```

Arguments

`name`	File name

Value

A `File` object

See Also

`AddFile`, `RemoveFile`

`GetJob` *Get a Job/DAG/Workflow*

Description

Get a Job/DAG/Workflow

Usage

```
GetJob(workflow, jobid)
```

Arguments

`workflow`	Workflow object
`jobid`	Job identification

Value

Job/DAG/Workflow object

See Also

`Workflow`, `HasJob`

HasDependency *Check to see if dependency exists*

Description

Check to see if dependency exists

Usage

```
HasDependency(workflow, dep)
```

Arguments

workflow	The Workflow object
dep	The dependency object

Value

If the Workflow contains the dependency

See Also

Workflow, Dependency

HasExecutable *Check if executable is in this Workflow*

Description

Check if executable is in this Workflow

Usage

```
HasExecutable(workflow, executable)
```

Arguments

workflow	Workflow object
executable	Executable object

Value

If the executable is in the Workflow

See Also

Workflow, Executable

HasFile *Check to see if file is in the Workflow*

Description

Check to see if file is in the Workflow

Usage

```
HasFile(workflow, file)
```

Arguments

workflow	Workflow object
file	File object

Value

If the Workflow object contains the file

See Also

Workflow, File

HasInvoke *Test whether an invocation is already appended to the InvokeMixin object.*

Description

Test whether an invocation is already appended to the InvokeMixin object.

Usage

```
HasInvoke(invoke.mixin, invoke)
```

Arguments

invoke.mixin	InvokeMixin object
invoke	invocation to be tested

Value

if the InvokeMixin object has the invocation

See Also

InvokeMixin

HasJob *Test to see if job is in this Workflow*

Description

The job parameter can be an object or a job ID.

Usage

```
HasJob(workflow, job)
```

Arguments

workflow	Workflow object
job	Job/DAG/Workflow object

Value

If the Job/DAG/Workflow is in the Workflow

See Also

Workflow, GetJob

HasTransformation *Check to see if transformation is in the Workflow*

Description

Check to see if transformation is in the Workflow

Usage

```
HasTransformation(workflow, transformation)
```

Arguments

workflow	Workflow object
transformation	Transformation object

Value

If the Workflow has the transformation

See Also

Workflow, Transformation

`Invoke` *Invoke executable* `what` *when job reaches status* `when`

Description

Invoke executable `what` when job reaches status `when`

Usage

```
Invoke(when, what)
```

Arguments

`when`	Job status
`what`	Executable to be invoked when job reach status `when`

Details

The value of `what` should be a command that can be executed on the submit host. The list of valid values for 'when' is:

WHEN	MEANING
never	never invoke
start	invoke just before job gets submitted.
on_error	invoke after job finishes with failure (exitcode != 0).
on_success	invoke after job finishes with success (exitcode == 0).
at_end	invoke after job finishes, regardless of exit status.
all	like start and at_end combined.

Value

Invoke object

Examples

```
invoke_1 <- Invoke(Pegasus.When$AT_END, '/usr/bin/mail -s "job done" rafsilva@isi.edu')
invoke_2 <- Invoke(Pegasus.When$ON_ERROR, '/usr/bin/update_db -failure')
```

`InvokeExecutable` *Invoke executable* `what` *when job reaches status* `when`.

Description

Invoke executable `what` when job reaches status `when`.

Usage

```
InvokeExecutable(invoke.mixin, when, what)
```

Arguments

`invoke.mixin`	InvokeMixin object
`when`	job status
`what`	executable to be invoked when job reach status `when`

Details

The value of `what` should be a command that can be executed on the submit host. The list of valid values for 'when' is:

WHEN	MEANING
never	never invoke
start	invoke just before job gets submitted.
on_error	invoke after job finishes with failure (exitcode != 0).
on_success	invoke after job finishes with success (exitcode == 0).
at_end	invoke after job finishes, regardless of exit status.
all	like start and at_end combined.

Value

InvokeMixin object with invocation appended to the list of invocations

See Also

`InvokeMixin`

`InvokeMixin` *Manage invocations*

Description

Manage invocations

Usage

```
InvokeMixin()
```

Value

InvokeMixin object with an empty list of invocations

See Also

`AddInvoke`, `HasInvoke`, `RemoveInvoke`, `ClearInvokes`, `InvokeExecutable`

IsDefined — *Test whether an object is not* NULL *and not* NA

Description

Test whether an object is not NULL and not NA

Usage

```
IsDefined(x)
```

Arguments

x	object to be tested

Value

If the object is not NULL and not NA

IsEqual — *Test whether to values are equal*

Description

Test whether to values are equal

Usage

```
IsEqual(v1, v2)
```

Arguments

v1	First value
v2	Second value

Value

If the values are equal

`Job` *This class defines the specifics of a job to run in an abstract manner*

Description

All filename references still refer to logical files. All references transformations also refer to logical transformations, though physical location hints can be passed through profiles.

Usage

```
Job(name, id = NULL, namespace = NULL, version = NULL, node.label = NULL)
```

Arguments

`name`	The transformation name or Transformation object (required)
`id`	A unique identifier for the job (optional)
`namespace`	The namespace of the transformation (optional)
`version`	The transformation version (optional)
`node.label`	The label for this job to use in graphing (optional)

Details

The ID for each job should be unique in the DAX. If it is None, then it will be automatically generated when the job is added to the DAX.

The name, namespace, and version should match what you have in your transformation catalog. For example, if namespace="foo" name="bar" and version="1.0", then the transformation catalog should have an entry for "foo::bar:1.0".

The name argument can be either a string, or a Transformation object. If it is a Transformation object, then the job will inherit the name, namespace, and version from the Transformation.

Value

The job object

See Also

`AddJob`, `Transformation`, `Executable`, `File`, `Profile`

Examples

```
sleep <- Job(id="ID0001", name="sleep")
jbsim <- Job(id="ID0002", name="jbsim", namespace="cybershake", version="2.1")
merge <- Job("jbsim")

# You can create a Job based on a Transformation:
  mDiff_xform <- Transformation("mDiff", namespace="montage", version="3.0")
  mDiff_job <- Job(mDiff_xform)

# Or an Executable:
  mDiff_exe <- Executable("mDiff", namespace="montage", version="3.0")
  mDiff_job <- Job(mDiff_exe)
```

```
# Several arguments can be added at the same time:
  input <- File("i1.txt")
  output <- File("o1.txt")
  mDiff_job <- AddArguments(mDiff_job, list("-i", input, "-o", output))

# Profiles are added similarly:
  mDiff_job <- AddProfile(mDiff_job, Profile(Pegasus.Namespace$ENV, key='PATH', value='/b

# Adding file uses is simple, and you can override global File attributes:
  mDiff_job <- Uses(mDiff_job, input, Pegasus.Link$INPUT)
  mDiff_job <- Uses(mDiff_job, output, Pegasus.Link$OUTPUT, transfer=TRUE, register=TRUE)
```

`Metadata.Executable`

Declarative metadata addition

Description

Declarative metadata addition

Usage

```
## S3 method for class 'Executable'
Metadata(obj, key, value)

## S3 method for class 'File'
Metadata(obj, key, value)

Metadata(obj, key, value)

## S3 method for class 'Job'
Metadata(obj, key, value)

## S3 method for class 'Transformation'
Metadata(obj, key, value)

## S3 method for class 'Workflow'
Metadata(obj, key, value)
```

Arguments

`obj`	Object to append the metadata
`key`	The metadata key
`value`	The metadata value

Value

The object containing the metadata

See Also

Executable
File
Metadata
Job
Transformation
Workflow

`NextJobID` *Get an autogenerated ID for the next job*

Description

Get an autogenerated ID for the next job

Usage

```
NextJobID(workflow)
```

Arguments

`workflow`	Workflow object

Value

Abstract workflow object with updated sequence number and the next.id in list format: `list(Workflow, next.id)`

See Also

Workflow

`Pegasus.Arch` *Architecture types*

Description

Architecture types

Usage

```
Pegasus.Arch
```

Format

An object of class `list` of length 8.

See Also

Executable

`Pegasus.Link` *Linkage attributes*

Description

Linkage attributes

Usage

```
Pegasus.Link
```

Format

An object of class `list` of length 5.

See Also

`File`, `Executable`, `Uses`

`Pegasus.Namespace` *Namespace values recognized by Pegasus*

Description

Namespace values recognized by Pegasus

Usage

```
Pegasus.Namespace
```

Format

An object of class `list` of length 8.

See Also

`Executable`, `Transformation`, `Job`

`Pegasus.OS` *OS types*

Description

OS types

Usage

```
Pegasus.OS
```

Format

An object of class `list` of length 5.

See Also

`Executable`

`Pegasus.Transfer` *Transfer types for uses*

Description

Transfer types for uses

Usage

```
Pegasus.Transfer
```

Format

An object of class `list` of length 3.

See Also

`Executable`, `File`

Pegasus.When — *Job states for notifications*

Description

Job states for notifications

Usage

```
Pegasus.When
```

Format

An object of class `list` of length 6.

See Also

`Job`, `DAX`, `DAG`, `Invoke`

PFN — *A physical file name. Used to provide URLs for files and executables in the DAX-level replica catalog.*

Description

A physical file name. Used to provide URLs for files and executables in the DAX-level replica catalog.

Usage

```
PFN(url, site = "local")
```

Arguments

`url`	The url of the file
`site`	The name of the site

Details

PFNs can be added to `File` and `Executable`.

Value

The PFN object with the URL and site

See Also

`AddPFN`, `File`, `Executable`

Examples

```
PFN('http://site.com/path/to/file.txt', 'site')
PFN('http://site.com/path/to/file.txt', site='site')
PFN('http://site.com/path/to/file.txt')
```

`Profile` *A Profile captures scheduler-, system-, and environment-specific parameters in a uniform fashion*

Description

A Profile captures scheduler-, system-, and environment-specific parameters in a uniform fashion. Each profile declaration assigns a value to a key within a namespace.

Profiles can be added to `Job`, `DAX`, `DAG`, `File`, `Executable`, and `PFN`.

Usage

```
Profile(namespace, key, value)
```

Arguments

`namespace`	The namespace of the profile
`key`	The key name. Can be anything that responds to as.character()
`value`	The value for the profile. Can be anything that responds to as.character()

Value

Profile object with the defined key=value pair

See Also

`Pegasus.Namespace`

Examples

```
path <- Profile(Pegasus.Namespace$ENV, 'PATH', '/bin')
vanilla <- Profile(Pegasus.Namespace$CONDOR, 'universe', 'vanilla')
path <- Profile(namespace='env', key='PATH', value='/bin')
path <- Profile('env', 'PATH', '/bin')
```

RemoveDependency *Remove dependency from workflow*

Description

Remove dependency from workflow

Usage

```
RemoveDependency(workflow, dep)
```

Arguments

workflow	The Workflow object
dep	The dependency object

Value

The Workflow object without the dependency

See Also

Workflow, Dependency, Depends, AddDependency

RemoveExecutable *Remove executable from the Workflow*

Description

Remove executable from the Workflow

Usage

```
RemoveExecutable(workflow, executable)
```

Arguments

workflow	Workflow object
executable	Executable object

Value

The Workflow object without the executable

See Also

Workflow, Executable, AddExecutable, ClearExecutables

RemoveFile *Remove file from this Workflow*

Description

Remove file from this Workflow

Usage

```
RemoveFile(workflow, file)
```

Arguments

workflow	Workflow object
file	File object

Value

The Workflow object without the file

See Also

Workflow, File, AddFile, ClearFiles

RemoveInvoke *Remove an invocation from the InvokeMixin object*

Description

Remove an invocation from the InvokeMixin object

Usage

```
RemoveInvoke(invoke.mixin, invoke)
```

Arguments

invoke.mixin	InvokeMixin object
invoke	invocation to be removed

Value

InvokeMixin object without the removed invocation

RemoveJob *Remove job from the Workflow*

Description

Remove job from the Workflow

Usage

```
RemoveJob(workflow, job)
```

Arguments

workflow	Workflow object
job	Job/DAG/Workflow object

Value

The Workflow object without the Job/DAG/Workflow

See Also

Workflow, AddJob, ClearJobs

RemoveTransformation *Remove transformation from the Workflow*

Description

Remove transformation from the Workflow

Usage

```
RemoveTransformation(workflow, transformation)
```

Arguments

workflow	Workflow object
transformation	Transformation object

Value

The Workflow object without the transformation

See Also

Workflow, Transformation, AddTransformation, ClearTransformations

ToXML *Get the XML string for the object*

Description

Get the XML string for the object

Usage

```
ToXML(obj)

## S3 method for class 'Transformation'
ToXML(obj)
```

Arguments

obj	Object to parse as XML

Value

The XML string for the object

Transformation *A logical transformation*

Description

A logical transformation. This is basically defining one or more entries in the transformation catalog. You can think of it like a macro for adding <uses> to your jobs. You can define a transformation that uses several files and/or executables, and refer to it when creating a job. If you do, then all of the uses defined for that transformation will be copied to the job during planning.

This code:

```
in <- File("input.txt")
exe <- Executable("exe")
t <- Transformation(namespace="foo", name="bar", version="baz")
t <- Uses(t, in)
t <- Uses(t, exe)
j <- Job(t)
```

is equivalent to:

```
in <- File("input.txt")
exe <- Executable("exe")
j <- Job(namespace="foo", name="bar", version="baz")
j <- Uses(j, in)
j <- Uses(j, exe)
```

Usage

```
Transformation(name, namespace = NULL, version = NULL)
```

Arguments

name	The name of the transformation
namespace	The namespace of the xform (optional)
version	The version of the xform (optional)

Details

The name argument can be either a string or an Executable object. If it is an Executable object, then the Transformation inherits its name, namespace and version from the Executable, and the Transformation is set to use the Executable with link=input, transfer=TRUE, and register=FALSE.

Value

Transformation object

Examples

```
Transformation(name='mDiff')
Transformation(namespace='montage',name='mDiff')
Transformation(namespace='montage',name='mDiff',version='3.0')

# Using one executable:
  mProjectPP <- Executable(namespace="montage", name="mProjectPP", version="3.0")
  x_mProjectPP <- Transformation(mProjectPP)

# Using several executables:
  mDiff <- Executable(namespace="montage", name="mProjectPP", version="3.0")
  mFitplane <- Executable(namespace="montage", name="mFitplane", version="3.0")
  mDiffFit <- Executable(namespace="montage", name="mDiffFit", version="3.0")
  x_mDiffFit <- Transformation(mDiffFit)

# Config files too:
  conf <- File("jbsim.conf")
  jbsim <- Executable(namespace="scec",name="jbsim")
  x_jbsim <- Transformation(jbsim)
  x_jbsim <- Uses(x_jbsim, conf)
```

Uses *Use of a logical file name*

Description

Use of a logical file name. Used for referencing files in the DAX.

Usage

```
Uses(
  obj,
  arg,
  link = NULL,
```

```
  register = NULL,
  transfer = NULL,
  optional = NULL,
  namespace = NULL,
  version = NULL,
  executable = NULL,
  size = NULL
)
```

Arguments

obj	Object (Transformation or Job)
arg	A string, an Executable, or a File representing the logical file
link	Is this file a job input, output or both (See LFN) (optional)
register	Should this file be registered in RLS? (True/False) (optional)
transfer	Should this file be transferred? (True/False or See LFN) (optional)
optional	Is this file optional, or should its absence be an error? (optional)
namespace	Namespace of executable (optional)
version	version of executable (optional)
executable	Is file an executable? (TRUE/FALSE) (optional)
size	The size of the file (optional)

Details

For Use objects that are added to Transformations, the attributes 'link', 'register', 'transfer', 'optional' and 'size' are ignored.

If a File object is passed in as 'file', then the default value for executable is 'false'. Similarly, if an Executable object is passed in, then the default value for executable is 'true'.

Value

Job with references to the files

See Also

Job, Executable, File

Workflow *Representation of a Pegasus Workflow*

Description

Representation of a Pegasus Workflow

Usage

```
Workflow(name)
```

Arguments

name	The name of the workflow

Value

An object with a Workflow

Examples

```
# Example of a black diamond workflow
# Create a Workflow
diamond <- Workflow("diamond")

# Add some metadata
diamond <- Metadata(diamond, "name", "diamond")
diamond <- Metadata(diamond, "createdby", "Rafael Ferreira da Silva")

# Add input file to the DAX-level replica catalog
a <- File("f.a")
a <- AddPFN(a, PFN("gsiftp://site.com/inputs/f.a", "site"))
a <- Metadata(a, "size", "1024")
diamond <- AddFile(diamond, a)

# Add executables to the DAX-level replica catalog
e.preprocess <- Executable(namespace="bd",name="process",version="4.0",os="linux",arch="x
e.preprocess <- Metadata(e.preprocess,"size","2048")
e.preprocess <- AddPFN(e.preprocess, PFN("gsiftp://site.com/bin/preprocess", "site"))
diamond <- AddExecutable(diamond, e.preprocess)

e.findrange <- Executable(namespace="bd",name="frange",version="4.0",os="linux",arch="x86
e.findrange <- AddPFN(e.findrange, PFN("gsiftp://site.com/bin/findrange", "site"))
diamond <- AddExecutable(diamond, e.findrange)

e.analyze <- Executable(namespace="bd",name="analyze",version="4.0",os="linux",arch="x86_
e.analyze <- AddPFN(e.analyze, PFN("gsiftp://site.com/bin/analyze", "site"))
diamond <- AddExecutable(diamond, e.analyze)

# Add a preprocess job
preprocess <- Job(e.preprocess)
preprocess <- Metadata(preprocess, "time", "60")
b1 <- File("f.b1")
b2 <- File("f.b2")
preprocess <- AddArguments(preprocess, list("-a","preprocess","-T","3","-i",a,"-o",b1,b2)
preprocess <- Uses(preprocess, a, link=Pegasus.Link$INPUT)
preprocess <- Uses(preprocess, b1, link=Pegasus.Link$OUTPUT, transfer=TRUE)
preprocess <- Uses(preprocess, b2, link=Pegasus.Link$OUTPUT, transfer=TRUE)
diamond <- AddJob(diamond, preprocess)

# Add left Findrange job
frl <- Job(e.findrange)
frl <- Metadata(frl, "time", "60")
c1 <- File("f.c1")
frl <- AddArguments(frl, list("-a","findrange","-T","3","-i",b1,"-o",c1))
frl <- Uses(frl, b1, link=Pegasus.Link$INPUT)
frl <- Uses(frl, c1, link=Pegasus.Link$OUTPUT, transfer=TRUE)
diamond <- AddJob(diamond, frl)
```

```
# Add right Findrange job
frr <- Job(e.findrange)
frr <- Metadata(frr, "time", "60")
c2 <- File("f.c2")
frr <- AddArguments(frr, list("-a","findrange","-T","3","-i",b2,"-o",c2))
frr <- Uses(frr, b2, link=Pegasus.Link$INPUT)
frr <- Uses(frr, c2, link=Pegasus.Link$OUTPUT, transfer=TRUE)
diamond <- AddJob(diamond, frr)

# Add Analyze job
analyze <- Job(e.analyze)
analyze <- Metadata(analyze, "time", "60")
d <- File("f.d")
analyze <- AddArguments(analyze, list("-a","analyze","-T","3","-i",c1,c2,"-o",d))
analyze <- Uses(analyze, c1, link=Pegasus.Link$INPUT)
analyze <- Uses(analyze, c2, link=Pegasus.Link$INPUT)
analyze <- Uses(analyze, d, link=Pegasus.Link$OUTPUT, transfer=TRUE)
diamond <- AddJob(diamond, analyze)

# Add dependencies
diamond <- Depends(diamond, parent=preprocess, child=frl)
diamond <- Depends(diamond, parent=preprocess, child=frr)
diamond <- Depends(diamond, parent=frl, child=analyze)
diamond <- Depends(diamond, parent=frr, child=analyze)

# Get generated diamond dax
WriteYAML(diamond, stdout())
```

`WriteYAML` *Write the Workflow as YAML to a stream*

Description

Write the Workflow as YAML to a stream

Usage

```
WriteYAML(workflow, out)
```

Arguments

`workflow`	The Workflow object
`out`	The stream object (e.g., `stdout()`, or a filename)

See Also

`Workflow`

Examples

```
workflow <- Workflow('diamond')
WriteYAML(workflow, stdout())
WriteYAML(workflow, 'diamond.yml')
```

`WriteYAMLFile` *Write the Workflow to a YAML file*

Description

Write the Workflow to a YAML file

Usage

```
WriteYAMLFile(workflow, filename)
```

Arguments

`workflow`	The Workflow object
`filename`	Name of the file

See Also

`Workflow`, `WriteYAML`

Index