

Package ‘dax3’

September 5, 2017

Type Package

Title API for Generating Pegasus DAXes

Version 3.6.0

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)

LazyData TRUE

URL <http://pegasus.isi.edu>

NeedsCompilation no

R topics documented:

ADAG	1
AddArguments	3
AddChild	4
AddDAG	4
AddDAX	5
AddDependency	5
AddExecutable	6
AddFile	6
AddInvoke.ADAG	7
AddInvokeMixin	8
AddJob	8
AddPFN.Executable	9
AddProfile.DAG	10
AddProfileMixin	11
AddTransformation	11
AppendToList	12
ClearArguments	12
ClearDependencies	13
ClearExecutables	13

ClearFiles	14
ClearInvokes	14
ClearJobs	15
ClearTransformations	15
DAG	16
DAX	16
DAX3.Arch	17
DAX3.Link	18
DAX3.Namespace	18
DAX3.OS	19
DAX3.Transfer	19
DAX3.When	20
Dependency	20
Depends	21
Element	21
Escape	22
Executable	22
File	23
GetJob	24
HasDependency	24
HasExecutable	25
HasFile	25
HasInvoke	26
HasJob	26
HasTransformation	27
Invoke	27
InvokeExecutable	28
InvokeMixin	29
IsDefined	29
IsEqual	30
Job	30
Metadata.ADAG	31
NextJobID	32
PFN	33
Profile	34
RemoveDependency	34
RemoveExecutable	35
RemoveFile	36
RemoveInvoke	36
RemoveJob	37
RemoveTransformation	37
ToXML.ADAG	38
Transformation	39
Uses	40
WriteXML	41
WriteXMLFile	42

ADAG

*Representation of a directed acyclic graph in XML (DAX)***Description**

Representation of a directed acyclic graph in XML (DAX)

Usage

```
ADAG(name, count = NULL, index = NULL)
```

Arguments

name	The name of the workflow
count	Total number of DAXes that will be created
index	Zero-based index of this DAX

Value

An object with a DAX

Examples

```
# Example of a black diamond workflow
# Create a DAX
diamond <- ADAG("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="x86_64")
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_64")
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_64")
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", "-T60", "-i", a, "-o", b1, b2))
preprocess <- Uses(preprocess, a, link=DAX3.Link$INPUT)
preprocess <- Uses(preprocess, b1, link=DAX3.Link$OUTPUT, transfer=TRUE)
preprocess <- Uses(preprocess, b2, link=DAX3.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", "-T60", "-i", b1, "-o", c1))
frl <- Uses(frl, b1, link=DAX3.Link$INPUT)
frl <- Uses(frl, c1, link=DAX3.Link$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", "-T60", "-i", b2, "-o", c2))
frr <- Uses(frr, b2, link=DAX3.Link$INPUT)
frr <- Uses(frr, c2, link=DAX3.Link$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", "-T60", "-i", c1, c2, "-o", d))
analyze <- Uses(analyze, c1, link=DAX3.Link$INPUT)
analyze <- Uses(analyze, c2, link=DAX3.Link$INPUT)
analyze <- Uses(analyze, d, link=DAX3.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
WriteXML(diamond, stdout())

```

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	<i>Append a child element to a parent element</i>
----------	---------------------------------------------------

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	<i>Add a sub-DAG (synonym for addJob)</i>
--------	-------------------------------------------

Description

Add a sub-DAG (synonym for addJob)

Usage

AddDAG (adag, dag)

Arguments

- | | |
|------|------------------------|
| adag | ADAG object |
| dag | Sub-DAG to be appended |

Value

The ADAG object with the sub-DAG appended

See Also

ADAG, AddJob, DAG

AddDAX	<i>Add a sub-DAX (synonym for addJob)</i>
--------	-------------------------------------------

Description

Add a sub-DAX (synonym for addJob)

Usage

AddDAX (adag, dax)

Arguments

- | | |
|------|------------------------|
| adag | ADAG object |
| dax | Sub-DAX to be appended |

Value

The ADAG object with the sub-DAX appended

See Also

ADAG, AddJob, DAX

AddDependency	<i>Add a dependency to the workflow</i>
---------------	-----------------------------------------

Description

Add a dependency to the workflow

Usage

```
AddDependency(adag, dep)
```

Arguments

adag	The ADAG object
dep	The dependency object

Value

The ADAG object containing the dependency

See Also

ADAG, Dependency, Depends, RemoveDependency

AddExecutable	<i>Add an executable to the ADAG</i>
---------------	--------------------------------------

Description

Add an executable to the ADAG

Usage

```
AddExecutable(adag, executable)
```

Arguments

adag	ADAG object
executable	Executable object

Value

The ADAG object with the executable appended

See Also

ADAG, Executable, RemoveExecutable, ClearExecutables

AddFile	<i>Add a file to the DAX</i>
---------	------------------------------

Description

Add a file to the DAX

Usage

```
AddFile(adag, file)
```

Arguments

adag	ADAG object
file	File object

Value

The ADAG object with the file appended

See Also

ADAG, File, RemoveFile, ClearFiles

AddInvoke.ADAG	<i>Add an invoke to the object</i>
----------------	------------------------------------

Description

Add an invoke to the object

Usage

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

## 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)
```

Arguments

obj	Object to append the invoke
invoke	The invocation

Value

The object containing the invoke

See Also

ADAG
DAG
DAX
Executable
Invoke
Job
Transformation

AddInvokeMixin	<i>Add invoke to the InvokeMixin object</i>
----------------	---------------------------------------------

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	<i>Add a job to the ADAG</i>
--------	------------------------------

Description

Add a job to the ADAG

Usage

AddJob (adag, job)

Arguments

adag	ADAG object
job	Job object

Value

ADAG object with the job appended

See Also

ADAG, RemoveJob

AddPFN.Executable	<i>Add a PFN to the object</i>
-------------------	--------------------------------

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
pfm	The PFN

Value

The object containing the PFN

See Also

Executable
File
PFN

AddProfile.DAG	<i>Add a profile to the object</i>
----------------	------------------------------------

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	<i>Add a profile to the object</i>
-----------------	------------------------------------

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	<i>Add a transformation to the ADAG</i>
-------------------	-----------------------------------------

Description

Add a transformation to the ADAG

Usage

```
AddTransformation(adag, transformation)
```

Arguments

adag	ADAG object
transformation	Transformation object

Value

The ADAG object with the transformation appended

See Also

ADAG, Transformation, RemoveTransformation, ClearTransformations

AppendToList	<i>Append a value to a list</i>
--------------	---------------------------------

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	<i>Removes all arguments from the job</i>
----------------	-------------------------------------------

Description

Removes all arguments from the job

Usage

ClearArguments (job)

Arguments

job	Job object
-----	------------

Value

Job with no arguments

See Also

Job

ClearDependencies	<i>Remove all dependencies</i>
-------------------	--------------------------------

Description

Remove all dependencies

Usage

ClearDependencies (adag)

Arguments

adag	The ADAG object
------	-----------------

Value

The ADAG object with no dependencies

See Also

ADAG, Dependency, Depends, AddDependency, RemoveDependency

ClearExecutables	<i>Remove all executables</i>
------------------	-------------------------------

Description

Remove all executables

Usage

ClearExecutables (adag)

Arguments

adag ADAG object

Value

The ADAG object with no executables

See Also

ADAG, Executable, AddExecutable, RemoveExecutable

ClearFiles	<i>Remove all files</i>
------------	-------------------------

Description

Remove all files

Usage

ClearFiles (adag)

Arguments

adag ADAG object

Value

The ADAG object with no files

See Also

ADAG, File, AddFile, RemoveFile

ClearInvokes	<i>Remove all Invoke objects</i>
--------------	----------------------------------

Description

Remove all Invoke objects

Usage

ClearInvokes (invoke.mixin)

Arguments

invoke.mixin InvokeMixin object

Value

InvokeMixin with no invocations

See Also

InvokeMixin

ClearJobs	<i>Remove all jobs</i>
-----------	------------------------

Description

Remove all jobs

Usage

ClearJobs (adag)

Arguments

adag ADAG object

Value

The ADAG object with no jobs

See Also

ADAG, RemoveJob, AddJob

ClearTransformations
<i>Remove all transformations</i>

Description

Remove all transformations

Usage

ClearTransformations(adag)

Arguments

adag ADAG object

Value

The ADAG object with no transformations

See Also

ADAG, Transformation, AddTransformation, RemoveTransformation

DAG	<i>This job represents a sub-DAG that will be executed by the workflow</i>
-----	----------------------------------------------------------------------------

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

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

<code>file</code>	The logical name of the DAX file or the DAX File object
<code>id</code>	The id of the DAX job [default: autogenerated]
<code>node.label</code>	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)
```

`DAX3.Arch`*Architecture types*

Description

Architecture types

Usage`DAX3.Arch`**Format**

List of 8

```
$ X86      : chr "x86"
$ X86_64   : chr "x86_64"
$ PPC      : chr "ppc"
$ PPC_64   : chr "ppc_64"
$ IA64     : chr "ia64"
$ SPARCV7  : chr "sparcv7"
$ SPARCV9  : chr "sparcv9"
$ AMD64    : chr "amd64"
```

See Also

Executable

`DAX3.Link`*Linkage attributes*

Description

Linkage attributes

Usage`DAX3.Link`**Format**

List of 5

```
$ NONE      : chr "none"
$ INPUT     : chr "input"
$ OUTPUT    : chr "output"
$ INOUT     : chr "inout"
$ CHECKPOINT: chr "checkpoint"
```

See Also

File, Executable, Uses

DAX3.Namespace	<i>Namespace values recognized by Pegasus</i>
----------------	-----------------------------------------------

Description

Namespace values recognized by Pegasus

Usage

DAX3.Namespace

Format

List of 8

\$ PEGASUS	:	chr	"pegasus"
\$ CONDOR	:	chr	"condor"
\$ DAGMAN	:	chr	"dagman"
\$ ENV	:	chr	"env"
\$ HINTS	:	chr	"hints"
\$ GLOBUS	:	chr	"globus"
\$ SELECTOR	:	chr	"selector"
\$ STAT	:	chr	"stat"

See Also

Executable, Transformation, Job

DAX3.OS	<i>OS types</i>
---------	-----------------

Description

OS types

Usage

DAX3.OS

Format

```
List of 5
$ LINUX   : chr "linux"
$ SUNOS   : chr "sunos"
$ AIX     : chr "aix"
$ MACOS   : chr "macos"
$ WINDOWS: chr "windows"
```

See Also

Executable

DAX3.Transfer	<i>Transfer types for uses</i>
---------------	--------------------------------

Description

Transfer types for uses

Usage

DAX3.Transfer

Format

```
List of 3
$ FALSE   : chr "false"
$ OPTIONAL: chr "optional"
$ TRUE    : chr "true"
```

See Also

Executable,File

DAX3.When	<i>Job states for notifications</i>
-----------	-------------------------------------

Description

Job states for notifications

Usage

DAX3.When

Format

```
List of 6
$ NEVER      : chr "never"
$ START      : chr "start"
$ ON_ERROR   : chr "on_error"
$ ON_SUCCESS : chr "on_success"
$ AT_END     : chr "at_end"
$ ALL        : chr "all"
```

See Also

Job, DAX, DAG, Invoke

Dependency	<i>A dependency between two nodes in the ADAG</i>
------------	---------------------------------------------------

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

Arguments

<code>adag</code>	The ADAG object
<code>child</code>	The child job/dax/dag or id
<code>parent</code>	The parent job/dax/dag or id
<code>edge.label</code>	A label for the edge (optional)

Value

The ADAG object with the dependency appended

See Also

ADAG, 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

<code>name</code>	element name
<code>attrs</code>	list of element attributes

Value

an element object

Escape	<i>Escape special characters in XML</i>
Description	
Escape special characters in XML	
Usage	
Escape (text)	
Arguments	
text	Text to be escaped
Value	
Escaped special character	
<hr/>	
Executable	<i>An entry for an executable in the DAX-level replica catalog</i>

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=DAX3.Arch$X86)
grep <- Executable(namespace="os",name="grep",version="2.3",arch=DAX3.Arch$X86,os=DAX3.OS
```

File	<i>A file entry for the DAX-level replica catalog, or a reference to a logical file used by the workflow</i>
------	--------------------------------------------------------------------------------------------------------------

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/DAX*

Description

Get a Job/DAG/DAX

Usage

```
GetJob(adag, jobid)
```

Arguments

<code>adag</code>	ADAG object
<code>jobid</code>	Job identification

Value

Job/DAG/DAX object

See Also

ADAG, HasJob

`HasDependency`*Check to see if dependency exists*

Description

Check to see if dependency exists

Usage

```
HasDependency(adag, dep)
```

Arguments

<code>adag</code>	The ADAG object
<code>dep</code>	The dependency object

Value

If the ADAG contains the dependency

See Also

ADAG, Dependency

HasExecutable	<i>Check if executable is in this ADAG</i>
---------------	--------------------------------------------

Description

Check if executable is in this ADAG

Usage

```
HasExecutable(adag, executable)
```

Arguments

adag	ADAG object
executable	Executable object

Value

If the executable is in the ADAG

See Also

ADAG, Executable

HasFile	<i>Check to see if file is in the ADAG</i>
---------	--------------------------------------------

Description

Check to see if file is in the ADAG

Usage

```
HasFile(adag, file)
```

Arguments

adag	ADAG object
file	File object

Value

If the ADAG object contains the file

See Also

ADAG, File

HasInvoke	<i>Test whether an invocation is already appended to the InvokeMixin object.</i>
-----------	----------------------------------------------------------------------------------

Description

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

Usage

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

Arguments

<code>invoke.mixin</code>	InvokeMixin object
<code>invoke</code>	invocation to be tested

Value

if the InvokeMixin object has the invocation

See Also

InvokeMixin

HasJob	<i>Test to see if job is in this ADAG</i>
--------	-------------------------------------------

Description

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

Usage

```
HasJob (adag, job)
```

Arguments

<code>adag</code>	ADAG object
<code>job</code>	Job/DAG/DAX object

Value

If the Job/DAG/DAX is in the ADAG

See Also

ADAG, GetJob

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

Description

Check to see if transformation is in the ADAG

Usage

```
HasTransformation(adag, transformation)
```

Arguments

adag	ADAG object
transformation	Transformation object

Value

If the ADAG has the transformation

See Also

ADAG, 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(DAX3.When$AT_END, '/usr/bin/mail -s "job done" rafsilva@isi.edu')
invoke_2 <- Invoke(DAX3.When$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	<i>Manage invocations</i>
-------------	---------------------------

Description

Manage invocations

Usage

InvokeMixin()

Value

InvokeMixin object with an empty list of invocations

See Also

AddInvoke, HasInvoke, RemoveInvoke, ClearInvokes, InvokeExecutable

IsDefined	<i>Test whether an object is not NULL and not NA</i>
-----------	------------------------------------------------------

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	<i>Test whether to values are equal</i>
---------	-----------------------------------------

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	<i>This class defines the specifics of a job to run in an abstract manner</i>
-----	-------------------------------------------------------------------------------

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(DAX3.Namespace$ENV, key='PATH', value='/bin/'))

# Adding file uses is simple, and you can override global File attributes:
mDiff_job <- Uses(mDiff_job, input, DAX3.Link$INPUT)
mDiff_job <- Uses(mDiff_job, output, DAX3.Link$OUTPUT, transfer=TRUE, register=TRUE)
```

Description

Declarative metadata addition

Usage

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

## 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)
```

Arguments

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

Value

The object containing the metadata

See Also

ADAG
Executable
File
Metadata
Job
Transformation

NextJobID

Get an autogenerated ID for the next job

Description

Get an autogenerated ID for the next job

Usage

NextJobID(adag)

Arguments

adag ADAG object

Value

DAX object with updated sequence number and the next.id in list format: list(ADAG, next.id)

See Also

ADAG

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

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	<i>A Profile captures scheduler-, system-, and environment-specific parameters in a uniform fashion</i>
---------	---------------------------------------------------------------------------------------------------------

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

DAX3.Namespace

Examples

```
path <- Profile(DAX3.Namespace$ENV, 'PATH', '/bin')
vanilla <- Profile(DAX3.Namespace$CONDOR, 'universe', 'vanilla')
path <- Profile(namespace='env', key='PATH', value='/bin')
path <- Profile('env', 'PATH', '/bin')
```

RemoveDependency	<i>Remove dependency from workflow</i>
------------------	----------------------------------------

Description

Remove dependency from workflow

Usage

```
RemoveDependency(adag, dep)
```

Arguments

adag	The ADAG object
dep	The dependency object

Value

The ADAG object without the dependency

See Also

ADAG, Dependency, Depends, AddDependency

RemoveExecutable	<i>Remove executable from the ADAG</i>
------------------	----------------------------------------

Description

Remove executable from the ADAG

Usage

```
RemoveExecutable(adag, executable)
```

Arguments

adag	ADAG object
executable	Executable object

Value

The ADAG object without the executable

See Also

ADAG, Executable, AddExecutable, ClearExecutables

RemoveFile	<i>Remove file from this ADAG</i>
------------	-----------------------------------

Description

Remove file from this ADAG

Usage

RemoveFile(adag, file)

Arguments

adag	ADAG object
file	File object

Value

The ADAG object without the file

See Also

ADAG, File, AddFile, ClearFiles

RemoveInvoke	<i>Remove an invocation from the InvokeMixin object</i>
--------------	---------------------------------------------------------

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 ADAG*

Description

Remove job from the ADAG

Usage

```
RemoveJob(adag, job)
```

Arguments

<code>adag</code>	ADAG object
<code>job</code>	Job/DAG/DAX object

Value

The ADAG object without the Job/DAG/DAX

See Also

`ADAG`, `AddJob`, `ClearJobs`

`RemoveTransformation`*Remove transformation from the ADAG*

Description

Remove transformation from the ADAG

Usage

```
RemoveTransformation(adag, transformation)
```

Arguments

<code>adag</code>	ADAG object
<code>transformation</code>	Transformation object

Value

The ADAG object without the transformation

See Also

ADAG, Transformation, AddTransformation, ClearTransformations

ToXML.ADAG

Get the XML string for the object

Description

Get the XML string for the object

Usage

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

ToXML(obj)

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

Arguments

obj Object to parse as XML

Details

For ADAG: This is primarily intended for testing. If you have a large ADAG you should use writeXML instead.

Value

The XML string for the object

See Also

ADAG, WriteXML

Transformation	<i>A logical transformation</i>
----------------	---------------------------------

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)
x_mDiffFit <- Uses(x_mDiffFit, mDiff)
x_mDiffFit <- Uses(x_mDiffFit, mFitplane)

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

Uses	<i>Use of a logical file name</i>
------	-----------------------------------

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

WriteXML

Write the ADAG as XML to a stream

Description

Write the ADAG as XML to a stream

Usage

```
WriteXML(adag, out)
```

Arguments

adag	The ADAG object
out	The stream object (e.g., <code>stdout()</code>), or a filename)

See Also

ADAG

Examples

```
dax <- ADAG('diamond')
WriteXML(dax, stdout())
WriteXML(dax, 'diamond.dax')
```

WriteXMLFile	<i>Write the ADAG to an XML file</i>
--------------	--------------------------------------

Description

Write the ADAG to an XML file

Usage

```
WriteXMLFile(adag, filename)
```

Arguments

adag	The ADAG object
filename	Name of the file

See Also

ADAG, WriteXML