

Pegasus 3.1.0 Basic Configuration

Pegasus 3.1.0 Basic Configuration

Table of Contents

1.	1
Basic Properties	1
pegasus.home	1
Catalog Properties	2
Data Staging Configuration	5

Basic Properties

This is the reference guide to the basic properties regarding the Pegasus Workflow Planner, and their respective default values. Please refer to the advanced properties guide to know about all the properties that a user can use to configure the Pegasus Workflow Planner. Please note that the values rely on proper capitalization, unless explicitly noted otherwise.

Some properties rely with their default on the value of other properties. As a notation, the curly braces refer to the value of the named property. For instance, `${pegasus.home}` means that the value depends on the value of the `pegasus.home` property plus any noted additions. You can use this notation to refer to other properties, though the extent of the substitutions are limited. Usually, you want to refer to a set of the standard system properties. Nesting is not allowed. Substitutions will only be done once.

There is a priority to the order of reading and evaluating properties. Usually one does not need to worry about the priorities. However, it is good to know the details of when which property applies, and how one property is able to overwrite another. The following is a mutually exclusive list (highest priority first) of property file locations.

1. `--conf` option to the tools. Almost all of the clients that use properties have a `--conf` option to specify the property file to pick up.
2. `submit-dir/pegasus.xxxxxxx.properties` file. All tools that work on the submit directory (i.e after pegasus has planned a workflow) pick up the `pegasus.xxxxxx.properties` file from the submit directory. The location for the `pegasus.xxxxxxx.properties` is picked up from the `braindump` file.
3. The properties defined in the user property file `${user.home}/.pegasusrc` have lowest priority.

Commandline properties have the highest priority. These override any property loaded from a property file. Each commandline property is introduced by a `-D` argument. Note that these arguments are parsed by the shell wrapper, and thus the `-D` arguments must be the first arguments to any command. Commandline properties are useful for debugging purposes.

From Pegasus 3.1 release onwards, support has been dropped for the following properties that were used to signify the location of the properties file

- `pegasus.properties`
- `pegasus.user.properties`

The following example provides a sensible set of properties to be set by the user property file. These properties use mostly non-default settings. It is an example only, and will not work for you:

<code>pegasus.catalog.replica</code>	File
<code>pegasus.catalog.replica.file</code>	<code>\${pegasus.home}/etc/sample.rc.data</code>
<code>pegasus.catalog.transformation</code>	Text
<code>pegasus.catalog.transformation.file</code>	<code>\${pegasus.home}/etc/sample.tc.text</code>
<code>pegasus.catalog.site</code>	XML3
<code>pegasus.catalog.site.file</code>	<code>\${pegasus.home}/etc/sample.sites.xml3</code>

If you are in doubt which properties are actually visible, pegasus during the planning of the workflow dumps all properties after reading and prioritizing in the submit directory in a file with the suffix `properties`.

pegasus.home

Systems:	all
Type:	directory location string
Default:	"\$PEGASUS_HOME"

The property `pegasus.home` cannot be set in the property file. This property is automatically set up by the pegasus clients internally by determining the installation directory of pegasus. Knowledge about this property is important for developers who want to invoke PEGASUS JAVA classes without the shell wrappers.

Catalog Properties

Replica Catalog

pegasus.catalog.replica

System:	Pegasus
Since:	2.0
Type:	enumeration
Value[0]:	RLS
Value[1]:	LRC
Value[2]:	JDBCRC
Value[3]:	File
Value[4]:	MRC
Default:	RLS

Pegasus queries a Replica Catalog to discover the physical filenames (PFN) for input files specified in the DAX. Pegasus can interface with various types of Replica Catalogs. This property specifies which type of Replica Catalog to use during the planning process.

RLS RLS (Replica Location Service) is a distributed replica catalog, which ships with GT4. There is an index service called Replica Location Index (RLI) to which 1 or more Local Replica Catalog (LRC) report. Each LRC can contain all or a subset of mappings. In this mode, Pegasus queries the central RLI to discover in which LRC's the mappings for a LFN reside. It then queries the individual LRC's for the PFN's. To use RLS, the user additionally needs to set the property `pegasus.catalog.replica.url` to specify the URL for the RLI to query. Details about RLS can be found at <http://www.globus.org/toolkit/data/rls/>

LRC If the user does not want to query the RLI, but directly a single Local Replica Catalog. To use LRC, the user additionally needs to set the property `pegasus.catalog.replica.url` to specify the URL for the LRC to query. Details about RLS can be found at <http://www.globus.org/toolkit/data/rls/>

JD-BCRC In this mode, Pegasus queries a SQL based replica catalog that is accessed via JDBC. The sql schema's for this catalog can be found at `$PEGASUS_HOME/sql` directory. To use JDBCRC, the user additionally needs to set the following properties

1. `pegasus.catalog.replica.db.url`
2. `pegasus.catalog.replica.db.user`
3. `pegasus.catalog.replica.db.password`

File In this mode, Pegasus queries a file based replica catalog. It is neither transactionally safe, nor advised to use for production purposes in any way. Multiple concurrent access to the File will end up clobbering the contents of the file. The site attribute should be specified whenever possible. The attribute key for the site attribute is "pool".

The LFN may or may not be quoted. If it contains linear whitespace, quotes, backslash or an equality sign, it must be quoted and escaped. Ditto for the PFN. The attribute key-value pairs are separated by an equality sign without any whitespaces. The value may be in quoted. The LFN sentiments about quoting apply.

```
LFN PFN
LFN PFN a=b [...]
LFN PFN a="b" [...]
"LFN w/LWS" "PFN w/LWS" [...]
```

To use File, the user additionally needs to specify `pegasus.catalog.replica.file` property to specify the path to the file based RC.

MRC In this mode, Pegasus queries multiple replica catalogs to discover the file locations on the grid. To use it set

```
pegasus.catalog.replica MRC
```

Each associated replica catalog can be configured via properties as follows.

The user associates a variable name referred to as `[value]` for each of the catalogs, where `[value]` is any legal identifier (concretely `[A-Za-z][_A-Za-z0-9]*`) For each associated replica catalogs the user specifies the following properties.

```
pegasus.catalog.replica.mrc.[value]    specifies the type of replica catalog.  
pegasus.catalog.replica.mrc.[value].key specifies a property name key for a  
particular catalog
```

For example, if a user wants to query two lrc's at the same time he/she can specify as follows

```
pegasus.catalog.replica.mrc.lrc1 LRC  
pegasus.catalog.replica.mrc.lrc2.url rls://sukhna  
pegasus.catalog.replica.mrc.lrc2 LRC  
pegasus.catalog.replica.mrc.lrc2.url rls://smarty
```

In the above example, `lrc1`, `lrc2` are any valid identifier names and `url` is the property key that needed to be specified.

pegasus.catalog.replica.url

System:	Pegasus
Since:	2.0
Type:	URI string
Default:	(no default)

When using the modern RLS replica catalog, the URI to the Replica catalog must be provided to Pegasus to enable it to look up filenames. There is no default.

Site Catalog

pegasus.catalog.site

System:	Site Catalog
Since:	2.0
Type:	enumeration
Value[0]:	XML3
Value[1]:	XML
Default:	XML3

The site catalog file is available in three major flavors: The Text and XML formats for the site catalog are deprecated. Users can use `pegasus-sc-converter` client to convert their site catalog to the newer XML3 format.

1. THIS FORMAT IS DEPRECATED. WILL BE REMOVED IN COMING VERSIONS. USE `pegasus-sc-converter` to convert XML format to XML3 Format. The "XML" format is an XML-based file. The XML format reads site

catalog conforming to the old site catalog schema available at <http://pegasus.isi.edu/wms/docs/schemas/sc-2.0/sc-2.0.xsd>

2. The "XML3" format is an XML-based file. The XML format reads site catalog conforming to the old site catalog schema available at <http://pegasus.isi.edu/wms/docs/schemas/sc-3.0/sc-3.0.xsd>

pegasus.catalog.site.file

System:	Site Catalog
Since:	2.0
Type:	file location string
Default:	<code>\${pegasus.home.sysconfdir}/sites.xml3 </code> <code>\${pegasus.home.sysconfdir}/sites.xml</code>
See also:	pegasus.catalog.site

Running things on the grid requires an extensive description of the capabilities of each compute cluster, commonly termed "site". This property describes the location of the file that contains such a site description. As the format is currently in flow, please refer to the userguide and Pegasus for details which format is expected. The default value is dependant on the value specified for the property `pegasus.catalog.site` . If type of SiteCatalog used is XML3, then `sites.xml3` is picked up from `sysconfdir` else `sites.xml`

Transformation Catalog

pegasus.catalog.transformation

System:	Transformation Catalog
Since:	2.0
Type:	enumeration
Value[0]:	Text
Value[1]:	File
Default:	Text
See also:	pegasus.catalog.transformation.file

Text In this mode, a multiline file based format is understood. The file is read and cached in memory. Any modifications, as adding or deleting, causes an update of the memory and hence to the file underneath. All queries are done against the memory representation.

The file `sample.tc.text` in the `etc` directory contains an example

Here is a sample textual format for transformation catalog containing one transformation on two sites

```
tr example::keg:1.0 {
#specify profiles that apply for all the sites for the transformation
#in each site entry the profile can be overridden
profile env "APP_HOME" "/tmp/karan"
profile env "JAVA_HOME" "/bin/app"
site isi {
profile env "me" "with"
profile condor "more" "test"
profile env "JAVA_HOME" "/bin/java.1.6"
pfn "/path/to/keg"
arch "x86"
os "linux"
osrelease "fc"
osversion "4"
type "INSTALLED"
site wind {
```

```

profile env "me" "with"
profile condor "more" "test"
pfn "/path/to/keg"
arch "x86"
os "linux"
osrelease "fc"
osversion "4"
type "STAGEABLE"

```

File THIS FORMAT IS DEPRECATED. WILL BE REMOVED IN COMING VERSIONS. USE `pegasus-tc-converter` to convert File format to Text Format. In this mode, a file format is understood. The file is read and cached in memory. Any modifications, as adding or deleting, causes an update of the memory and hence to the file underneath. All queries are done against the memory representation. The new TC file format uses 6 columns:

1. The resource ID is represented in the first column.
2. The logical transformation uses the colonized format `ns::name:vs`.
3. The path to the application on the system
4. The installation type is identified by one of the following keywords - all upper case: `INSTALLED`, `STAGEABLE`. If not specified, or **NULL** is used, the type defaults to `INSTALLED`.
5. The system is of the format `ARCH::OS[:VER:GLIBC]`. The following arch types are understood: "INTEL32", "INTEL64", "SPARCV7", "SPARCV9". The following os types are understood: "LINUX", "SUNOS", "AIX". If unset or **NULL**, defaults to `INTEL32::LINUX`.
6. Profiles are written in the format `NS::KEY=VALUE,KEY2=VALUE;NS2::KEY3=VALUE3` Multiple key-values for same namespace are separated by a comma "," and multiple namespaces are separated by a semicolon ";". If any of your profile values contains a comma you must not use the namespace abbreviator.

pegasus.catalog.transformation.file

Systems:	Transformation Catalog
Type:	file location string
Default:	<code>\${pegasus.home.sysconfdir}/tc.text </code> <code>\${pegasus.home.sysconfdir}/tc.data</code>
See also:	<code>pegasus.catalog.transformation</code>

This property is used to set the path to the textual transformation catalogs of type File or Text. If the transformation catalog is of type Text then `tc.text` file is picked up from `sysconfdir`, else `tc.data`

Data Staging Configuration

pegasus.data.configuration

System:	Pegasus
Since:	3.1
Type:	enumeration
Value[0]:	<code>sharedfs</code>
Value[1]:	<code>nonsharedfs</code>
Value[2]:	<code>condorio</code>
Default:	<code>sharedfs</code>

This property sets up Pegasus to run in different environments.

sharedfs If this is set, Pegasus will be setup to execute jobs on the shared filesystem on the execution site. This assumes, that the head node of a cluster and the worker nodes share a filesystem. The staging site in this case is the same as the execution site. Pegasus adds a create dir job to the executable workflow that creates a workflow specific directory on the shared filesystem . The data transfer jobs in the executable workflow (stage_in_ , stage_inter_ , stage_out_) transfer the data to this directory. The compute jobs in the executable workflow are launched in the directory on the shared filesystem. Internally, if this is set the following properties are set.

```
pegasus.execute.*.filesystem.local    false
```

condorio If this is set, Pegasus will be setup to run jobs in a pure condor pool, with the nodes not sharing a filesystem. Data is staged to the compute nodes from the submit host using Condor File IO. The planner is automatically setup to use the submit host (site local) as the staging site. All the auxillary jobs added by the planner to the executable workflow (create dir, data stagein and stage-out, cleanup) jobs refer to the workflow specific directory on the local site. The data transfer jobs in the executable workflow (stage_in_ , stage_inter_ , stage_out_) transfer the data to this directory. When the compute jobs start, the input data for each job is shipped from the workflow specific directory on the submit host to compute/worker node using Condor file IO. The output data for each job is similarly shipped back to the submit host from the compute/worker node. This setup is particularly helpful when running workflows in the cloud environment where setting up a shared filesystem across the VM's may be tricky. On loading this property, internally the following properties are set

```
pegasus.transfer.sls.*.impl           Condor
pegasus.execute.*.filesystem.local     true
pegasus.gridstart                      PegasusLite
pegasus.transfer.worker.package        true
```

nonsharedfs If this is set, Pegasus will be setup to execute jobs on an execution site without relying on a shared filesystem between the head node and the worker nodes. You can specify staging site (using --staging-site option to pegasus-plan) to indicate the site to use as a central storage location for a workflow. The staging site is independant of the execution sites on which a workflow executes. All the auxillary jobs added by the planner to the executable workflow (create dir, data stagein and stage-out, cleanup) jobs refer to the workflow specific directory on the staging site. The data transfer jobs in the executable workflow (stage_in_ , stage_inter_ , stage_out_) transfer the data to this directory. When the compute jobs start, the input data for each job is shipped from the workflow specific directory on the submit host to compute/worker node using pegasus-transfer. The output data for each job is similarly shipped back to the submit host from the compute/worker node. The protocols supported are at this time SRM, GridFTP, iRods, S3. This setup is particularly helpful when running workflows on OSG where most of the execution sites don't have enough data storage. Only a few sites have large amounts of data storage exposed that can be used to place data during a workflow run. This setup is also helpful when running workflows in the cloud environment where setting up a shared filesystem across the VM's may be tricky. On loading this property, internally the following properties are set

```
pegasus.execute.*.filesystem.local     true
pegasus.gridstart                      PegasusLite
pegasus.transfer.worker.package        true
```