REST API

Many objects of Jenkins provide the remote access API. They are available at `/.../api/` where "..." portion is the object for which you'd like to access.

XML API
: Access data exposed in HTML as XML for machine consumption. Schema is also available.

  You can also specify optional XPath to control the fragment you'd like to obtain (but see below). For example, `../api/xml?xpath=/*/*[0]`.

  For XPath that matches multiple nodes, you need to also specify the "wrapper" query parameter to specify the name of the root XML element to be create so that the resulting XML becomes well-formed.

  Similarly `exclude` query parameter can be used to exclude nodes that match the given XPath from the result. This is useful for trimming down the amount of data you fetch (but again see below). This query parameter can be specified multiple times.

  XPath filtering is powerful, and you can have it only return a very small data, but note that the server still has to build a full DOM of the raw data, which could cause a large memory spike. To avoid overloading the server, consider using the `tree` parameter, or use the `xpath` parameter in conjunction with the `tree` parameter. When used together, the result of the `tree` parameter filtering is built into DOM, then the XPath is applied to compute the final return value. In this way, you can often substantially reduce the size of DOM built in memory.

JSON API
: Access the same data as JSON for JavaScript-based access. `tree` may be used.

Python API

  Access the same data as Python for Python clients. This can be parsed into Python object as `eval(urllib.urlopen("...").read())` and the resulting object tree is identical to that of JSON. However, when you do this, beware of the security implication. If you are connecting to a non-trusted Jenkins, the server can send you malicious Python programs.

  In Python 2.6 or later you can safely parse this output using `ast.literal_eval(urllib.urlopen("...").read())`

For more information about remote API in Jenkins, see the documentation.

Controlling the amount of data you fetch

The `tree` query parameter allows you to explicitly specify and retrieve only the information you are looking for, by using an XPath-ish path expression. The value should be a list of property names to include, with sub-properties inside square braces. Try tree=jobs[name],views[name,jobs[name]] to see just a list of jobs (only giving the name) and views (giving the name and jobs they contain). **Note**: for array-type properties (such as `jobs` in this example), the name must be given in the original plural, not in the singular as the element would appear in XML (`<job>`). This will be more natural for e.g. json?tree=jobs[name] anyway: the JSON writer does not do plural-to-singular mangling because arrays are represented explicitly.

For array-type properties, a range specifier is supported. For example, `tree=jobs[name]{0,10}` would retrieve the name of the first 10 jobs. The range specifier has the following variants:

- **{M,N}**: From the M-th element (inclusive) to the N-th element (exclusive).
- **{M,}**: From the M-th element (inclusive) to the end.
- **{,N}**: From the first element (inclusive) to the N-th element (exclusive). The same as {0,N}.
- **{N}**: Just retrieve the N-th element. The same as {N,N+1}.

Another way to retrieve more data is to use the `depth=N` query parameter . This retrieves all the data up to the specified depth. Compare depth=0 and depth=1 and see what the difference is for yourself. Also note that data created

Because of the size of the data, the `depth` parameter should really be only used to explore what data Jenkins can return. Once you identify the data you want to retrieve, you can then come up with the `tree` parameter to exactly specify the data you need.

Fetch/Update config.xml

To programmatically obtain `config.xml`, hit this URL. You can also POST an updated `config.xml` to the same URL to programmatically update the configuration of a job.

Delete a job

To programmatically delete this job, do HTTP POST to this URL.

Retrieving all builds

To prevent Jenkins from having to load all builds from disk when someone accesses the job API, the `builds` tree only contains the 50 newest builds. If you really need to get all builds, access the `allBuilds` tree, e.g. by fetching `…/api/xml?tree=allBuilds[…]`. Note that this may result in significant performance degradation if you have a lot of builds in this job.

Fetch/Update job description

this URL can be used to get and set just the job description. POST form data with a "description" parameter to set the description.

Perform a build

To programmatically schedule a new build, post to this URL. If the build has parameters, post to this URL and provide the parameters as form data. Either way, the successful queueing will result in 201 status code with `Location` HTTP header pointing the URL of the item in the queue. By polling the `api/xml` sub-URL of the queue item, you can track the status of the queued task. Generally, the task will go through some state transitions, then eventually it becomes either cancelled (look for the "cancelled" boolean property), or gets executed (look for the "executable" property that typically points to the `AbstractBuild` object.)

To programmatically schedule SCM polling, post to this URL.

If security is enabled, the recommended method is to provide the username/password of an account with build permission in the request. Tools such as `curl` and `wget` have parameters to specify these credentials. Another alternative (but deprecated) is to configure the 'Trigger builds remotely' section in the job configuration. Then building or polling can be triggered by including a parameter called *token* in the request.

Disable/Enable a job

To programmatically disable a job, post to this URL. Similarly post to this URL for enabling this job.