**TikaServer in Tika 2.x**

See [Migrating to Tika 2.0.0](#) for a general overview of changes in Tika 2.x.

See [TikaServer](#) for building and general usage of tika-server.

**Major changes**

- Modularization – We've modularized tika-server:
  - `tika-server-core` includes all of the functionality of tika-server, but with no bundled parsers. Users might want this if they are only parsing a few file formats or want to use only their custom parsers.
  - `tika-server-standard` is what most people will want to use. As with the `tika-parsers-standard` module, this includes most of the common file format parsers. If needed, users may also add the `tika-parser-scientific-package` and `tika-parser-sqlite3-package` to the classpath. In 1.x, the first was included in tika-server 1.x by default, and the second was included only if users added xerial's sqlite3 jar on the classpath.
- `--spawnChild` mode is now default. In Tika 1.x, users had to specify this on the commandline to force `tika-server` to fork a process that did the actual parsing. This option is far more robust against timeouts, OOMs, crashes and other mishaps; the forking process monitors the forked process and will restart on timeouts, etc. **NOTE:** Client code needs to be able to handle the times when `tika-server` is restarting and is not available; this typically only takes a few seconds. To disable this mode, use `-noFork` on the commandline.
- Configuring `tika-server` in Tika 2.x. See below. We've moved most configuration options into `tika-config.xml` and dramatically limited the commandline options.
- The namespace has changed slightly for `TikaServerCli` to `org.apache.tika.server.core.TikaServerCli`. If adding optional jars to the class path in, say, a bin/ directory, start tika-server with: `java -cp *bin/* org.apache.tika.server.core.TikaServerCli -c tika-config.xml`
- `enableFileUrl` – We have removed this capability from tika-server in 2.x. We have replaced it with the FileSystemFetcher, which is available in tika-core. See [FetchersInClassicServerEndpoints](#).

**Configuring tika-server in Tika 2.x**

As with other components, in Tika 2.x, we moved configuration into `tika-config.xml`. We have left only a few commandline options available (to see the options: `java -jar tika-server-standard-2.x.x.jar --help`). **Please note that all command-line option values will override their counterparts in the xml configuration file.**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<properties>
<!-- <parsers etc.../> -->
<server>
  <params>
    <!-- which port to start the server on. If you specify a range, e.g. 9990-9999, and Tika will launch 10 servers in forked processes on each of those ports. Can also specify a comma-delimited list, e.g. (9996,9998,9999). -->
    <port>9998</port>
    <!-- if specified, this will be the id that is used in the /status endpoint and elsewhere. If an id is specified and more than one forked processes are invoked, each process will have an id followed by the port, e.g my_id-9998. If a forked server has to restart, it will maintain its original id. If not specified, a UUID will be generated. -->
    <id></id>
    <!-- Origin URL for cors requests. Set to '*' if you want to allow all CORS requests. Leave blank or remove element if you do not want to enable CORS. -->
    <cors>*</cors>
    <!-- which digests to calculate, comma delimited (e.g. md5,sha256); -->
  </params>
</server>
</properties>
```
optionally specify encoding followed by a colon (e.g. "sha1:32").
Can be empty if you don't want to calculate a digest -->
<digest>sha256</digest>
<!-- how much to read to memory during the digest phase before
spooling to disc...only if digest is selected -->
<digestMarkLimit>100000</digestMarkLimit>
<!-- request URI log level 'debug' or 'info' -->
<logLevel>info</logLevel>
<!-- whether or not to return the stacktrace in the data returned
to the user when a parse exception happens-->
<returnStackTrace>false</returnStackTrace>
<!-- If set to 'true', this runs tika server "in process"
in the legacy 1.x mode.
This means that the server will be susceptible to infinite loops
and crashes.
If set to 'false', the server will spawn a forked
process and restart the forked process on catastrophic failures
(this was called --spawnChild mode in 1.x).
noFork=false is the default in 2.x-->
<noFork>false</noFork>
<!-- maximum time to allow per parse before shutting down and restarting
the forked parser. Not allowed if noFork=true.-->
<taskTimeoutMillis>300000</taskTimeoutMillis>
<!-- maximum amount of time to wait for a forked process to
start up.
Not allowed if noFork=true.-->
<maxForkedStartupMillis>120000</maxForkedStartupMillis>
<!-- maximum number of times to allow a specific forked process
to be restarted.
Not allowed if noFork=true.-->
<maxRestarts>-1</maxRestarts>
<!-- maximum files to parse per forked process before
restarting the forked process to clear potential
memory leaks.
Not allowed if noFork=true.-->
<maxFiles>100000</maxFiles>
<!-- if you want to specify a specific javaPath for
the forked process. This path should end
the application 'java', e.g. /my/special-java/java
Not allowed if noFork=true.-->
<javaPath>java</javaPath>
<!-- jvm args to use in the forked process -->
<forkedJvmArgs>
<arg>-Xms1g</arg>
<arg>-Xmx1g</arg>
<arg>-Dlog4j.configurationFile=my-forked-log4j2.xml</arg>
</forkedJvmArgs>
<!-- this must be set to true for any handler that uses a fetcher or emitter. These pipes features are
inherently unsecure because
the client has the same read/write access as the tika-server process. Implementers must secure Tika
server so that only their clients can reach it.
A byproduct of setting this to true is that the /status endpoint is turned on-->
<enableUnsecureFeatures>false</enableUnsecureFeatures>
<!-- you can optionally select specific endpoints to turn on/load. This can improve resource usage and
decrease your attack surface.
If you want to access the status endpoint, specify it here or set unsecureFeatures to true-->
<endpoints>
<endpoint>status</endpoint>
<endpoint>rmeta</endpoint>
</endpoints>
</server>
</properties>