

# Chukwa Agent Setup Guide

## Table of contents

1 Overview.....	2
2 Agent Control.....	2
3 Command-line options.....	3
4 Adaptors.....	3

## 1. Overview

In a normal Chukwa installation, an *Agent* process runs on every machine being monitored. This process is responsible for all the data collection on that host. Data collection might mean periodically running a Unix command, or tailing a file, or listening for incoming UDP packets.

Each particular data source corresponds to a so-called *Adaptor*. Adaptors are dynamically loadable modules that run inside the Agent process. There is generally one Adaptor for each data source: for each file being watched or for each Unix command being executed. Each adaptor has a unique name. If you do not specify a name, one will be auto-generated by hashing the Adaptor type and parameters.

There are a number of Adaptors built into Chukwa, and you can also develop your own. Chukwa will use them if you add them to the Chukwa library search path (e.g., by putting them in a jarfile in `$CHUKWA_HOME/lib`.)

## 2. Agent Control

Once an Agent process is running, there are a number of commands that you can use to inspect and control it. By default, Agents listen for incoming commands on port 9093. Commands are case-insensitive

Command	Purpose	Options
add	Start an adaptor.	See below
close	Close socket connection to agent.	None
help	Display a list of available commands	None
list	List currently running adaptors	None
reloadcollectors	Re-read list of collectors	None
stop	Stop adaptor, abruptly	Adaptor name
stopall	Stop all adaptors, abruptly	Adaptor name
shutdown	Stop adaptor, gracefully	Adaptor name
stopagent	Stop agent process	None

The add command is by far the most complex; it takes several mandatory and optional

parameters. The general form is as follows:

```
add [name =] <adaptor_class_name> <datatype> <adaptor  
specific params> <initial offset>.
```

There are four mandatory fields: The word `add`, the class name for the Adaptor, the datatype of the Adaptor's output, and the sequence number for the first byte. There are two optional fields; the adaptor instance name, and the adaptor parameters.

The adaptor name, if specified, should go after the `add` command, and be followed with an equals sign. It should be a string of printable characters, without whitespace or '='. Chukwa Adaptor names all start with "adaptor\_". If you specify an adaptor name which does not start with that prefix, it will be added automatically.

Adaptor parameters aren't required by the Chukwa agent, but each class of adaptor may itself specify both mandatory and optional parameters. See below.

### 3. Command-line options

Normally, agents are configured via the file `conf/chukwa-agent-conf.xml`. However, there are a few command-line options that are sometimes useful in troubleshooting. If you specify "local" as an option, then the agent will print chunks to standard out, rather than to a collector. If you specify a URI, then that will be used as collector, overriding the collectors specified in `conf/collectors`. These options are intended for testing and debugging, not for production use.

```
bin/chukwa agent local
```

### 4. Adaptors

This section lists the standard adaptors, and the arguments they take.

- **FileAdaptor:** Pushes a whole file, as one Chunk, then exits. Takes one mandatory parameter; the file to push.  

```
add FileTailer FooData /tmp/foo 0
```

This pushes file `/tmp/foo` as one chunk, with datatype `FooData`.
- **filetailer.LWFTAdaptor** Repeatedly tails a file, treating the file as a sequence of bytes, ignoring the content. Chunk boundaries are arbitrary. This is useful for streaming binary data. Takes one mandatory parameter; a path to the file to tail. If log file is rotated while there is unread data, this adaptor will not attempt to recover it.  

```
add filetailer.LWFTAdaptor BarData /foo/bar 0
```

This pushes `/foo/bar` in a sequence of Chunks of type `BarData`
- **filetailer.FileTailingAdaptor** Repeatedly tails a file, again ignoring content and with unspecified Chunk boundaries. Takes one mandatory parameter; a path to the file to tail.

Keeps a file handle open in order to detect log file rotation.

```
add filetailer.FileTailingAdaptor BarData /foo/bar 0
```

This pushes /foo/bar in a sequence of Chunks of type BarData

- **filetailer.RCheckFTAdaptor** An experimental modification of the above, which avoids the need to keep a file handle open. Same parameters and usage as the above.
- **filetailer.CharFileTailingAdaptorUTF8** The same as the base FileTailingAdaptor, except that chunks are guaranteed to end only at carriage returns. This is useful for most ASCII log file formats.
- **filetailer.CharFileTailingAdaptorUTF8NewLineEscaped** The same, except that chunks are guaranteed to end only at non-escaped carriage returns. This is useful for pushing Chukwa-formatted log files, where exception stack traces stay in a single chunk.
- **DirTailingAdaptor** Takes a directory path and an adaptor name as mandatory parameters; repeatedly scans that directory and all subdirectories, and starts the indicated adaptor running on each file. Since the DirTailingAdaptor does not, itself, emit data, the datatype parameter is applied to the newly-spawned adaptors. Note that if you try this on a large directory with an adaptor that keeps file handles open, it is possible to exceed your system's limit on open files. A file pattern can be specified as an optional second parameter.

```
add DirTailingAdaptor logs /var/log/ *.log
```

```
filetailer.CharFileTailingAdaptorUTF8 0
```

- **ExecAdaptor** Takes a frequency (in milliseconds) as optional parameter, and then program name as mandatory parameter. Runs that program repeatedly at a rate specified by frequency.

```
add ExecAdaptor Df 60000 /bin/df -x nfs -x none 0
```

This adaptor will run df every minute, labeling output as Df.

- **UDPAdaptor** Takes a port number as mandatory parameter. Binds to the indicated UDP port, and emits one Chunk for each received packet.

```
add UdpAdaptor Packets 1234 0
```

This adaptor will listen for incoming traffic on port 1234, labeling output as Packets.

- **edu.berkeley.chukwa\_xtrace.XtrAdaptor** (available in contrib) Takes an [Xtrace](#) ReportSource class name [without package] as mandatory argument, and no optional parameters. Listens for incoming reports in the same way as that ReportSource would.

```
add edu.berkeley.chukwa_xtrace.XtrAdaptor Xtrace UdpReportSource 0
```

This adaptor will create and start a UdpReportSource, labeling its output datatype as Xtrace.