Channel Archiver Manual

March 15, 2006,
for R3.14.4 and higher

Involvements

Bob Dalesio designed the original index file, data file layout, and implemented the first prototype.
From then on, the following people have been involved at one time or another:

Thomas Birke,
Sergei Chevtsov,
Kay-Uwe Kasemir,
Chris Larrieu,
Greg Lawson,
Craig McChesney,
Peregrine McGehee,
Nick Pattengale,
Ernest Williams.

No Warranty

Contents

1 Example Setup **1**
1.1 Setup, archiveconfig.xml . . . 1
1.2 Sampling Computer . . . 3
1.2.1 update_archive_tree.pl . . . 4
1.2.2 ArchiveDaemon . . . 5
1.3 Sub-Archives . . . 12
1.4 Serving Computer . . . 13
1.4.1 update_server.pl . . . 13
1.4.2 update_indices.pl . . . 13
1.5 Indices . . . 14
1.6 Data Server . . . 14
1.7 Common Tasks . . . 14
1.7.1 Modify Engine's Request Files . . . 14
1.7.2 Add Engine or Daemon . . . 14
1.7.3 An Engine isn't running . . . 14
1.7.4 I want to stop a Daemon . . . 15
1.7.5 A Daemon isn't running . . . 15
1.7.6 Re-building a Master Index . . . 15
1.8 Data Management . . . 15

Chapter 1

Example Setup

The following describes how the archiver toolset is used at the Spallation Neutron Source (SNS), and how the scripts in the ChannelArchiver/ExampleSetup directory assist with the setup as described in here. We distinguish between two types of computers:

- Sampling Computer:
  A computer that runs ArchiveEngine instances.
- Serving Computer:
  A computer that uses the ArchiveIndexTool to create additional binary indices and runs the ArchiveDataServer.

There might be more than one 'sampling' computer as well as more than one 'serving' computer. A single machine might perform both functions, but in general they can be different computers on the network, and hence some tools are required to make the data collected on the "sampling" computer available on the "server". One could use NFS, but we have decided to use secure copy (scp) in order to decouple the computers as best as possible.

We want to be able to move an engine from one computer to another, and still keep an overview. Therefore a file "/arch/archiveconfig.xml" describes the complete archive setup: Which engines run where, and how the data gets served. On some computers, for example ics-srv-archive1, further subdirectories of "/arch" are used to run engines. On another computer, for example ics-srv-web2, subdirectories contain data copied from archive1 so that the data server can serve it.

1.1 Setup, archiveconfig.xml

Each computer needs to have the same copy of /arch/archiveconfig.xml, and the scripts from the ExampleSetup directory of the archiver sources need to be copied into /arch/scripts. You might generate and distribute archiveconfig.xml manually or use a relational database. People who have used previous releases of the archive toolset might remember the archiveconfig.csv file. There is a tool scripts/convert_archiveconfig_to_xml.pl to convert that file into an archiveconfig.xml skeleton.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE archiveconfig SYSTEM "archiveconfig.dtd">
<archiveconfig>
  <root>/arch</root>

  <serverconfig>/var/www/serverconfig.xml</serverconfig>

  <mailbox>/arch/xfer</mailbox>

  <daemon directory='RF'>
    <run>archive1</run>
    <desc>RF</desc>
    <port>4900</port>
    <dataserver>
      <index type='binary' key='10'>RF</index>
      <host>web2</host>
    </dataserver>
    <engine directory='llrf'>
      <run>archive1</run>
      <desc>LLRF</desc>
      <port>4901</port>
      <restart type='weekly'>We 10:20</restart>
      <dataserver>
        <current_index key='4901'>llrf</current_index>
        <index type='binary'>LLRF data</index>
        <host>web2</host>
      </dataserver>
    </engine>
  </daemon>
</archiveconfig>
```

Listing 1.1: archiveconfig.xml

The archiveconfig.xml file describes the complete archive layout, using the following elements:

- root: Names the root directory, typically '/arch'. This has to be the same directory name on all computers, since they all use the same archiveconfig.xml.
- serverconfig: Location of the server configuration to be created. See

section 1.4.1.

- mailbox: Used to communicate from the ArchiveDaemons to the data server.
- daemon: Configures an archive daemon and its engines, described in section 1.2.2, as well as how that data should be indexed and served, see sections 1.4.1 and 1.4.2.

1.2 Sampling Computer

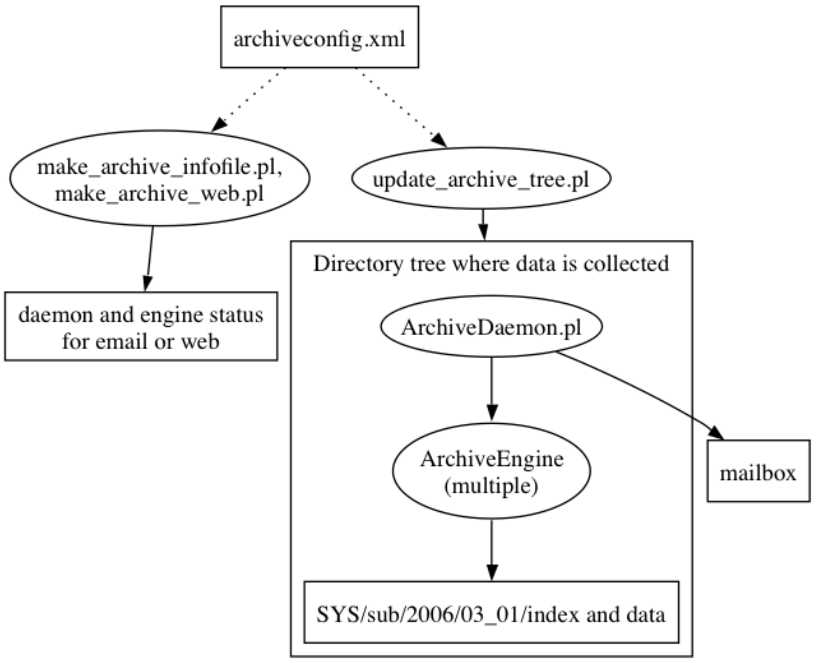


Figure 1.1: Tools used on a sampling computer, refer to text.

Instead of running just one archive engine, it is often preferred to run several. This way, there can be separate engines for separtate subsystems, possibly maintained by different people. In addition, periodic restarts, for example once a week, create a separate Sub-Archive with each restart, thereby limiting the possibility for data loss in case an engine crashes or creates corrupt archives.

For each subsystem, an "ArchiveDaemon" program manages one or more engines, monitors their condition, and performs the periodic restarts. For example, we might run one 'daemon' for the Integrated Control System (ICS) and one for the channels related to Radio Frequency (RF). The ICS daemon should maintain one ArchiveEngine for the timing system (tim) and one for the machine protection system (mps), while the RF daemon has one engine for the low-level and one for the high power RF (llrf, hprf).

This separation is somewhat arbitrary. We could have made “llrf” and “hprf” channel groups under one and the same engine. In fact all the above could reside within one engine, and the result would probably be less CPU load compared to the setup with multiple engines. It is, however, advisable to spread the channels over different daemons and engines whenever different people deal with the IOCs that host the channels, so that the engineers can independently configure their archiving. In addition, you want to keep the amount of data collected by each engine within certain bounds, for example: not more than one CD ROM per month, one DVD per year, or whatever you plan to do for data maintenance. You can of course also follow the approach that most sites use in reality: Wait until all disks are full, then panic. In which case, another reason is data safety: You can reduce the damage caused by crashes of one engine to a certain number of channels and the data for the restart period, for example one week.

1.2.1 update_archive_tree.pl

Initially and after every change to the configuration, the update_archive_tree script is used to create the necessary infrastructure. This script reads archive-config.xml and create all the subdirectories, daemon config files, and skeleton engine configurations which are meant to run on the local computer. Run it with “-h” to see available options.
NOTE: In order to determine what daemons should run on the local host, the host name in the “run” tags is used as a regular expression for the host name. So when specifying that a certain daemon should run on “archive1”:

```
<daemon ...
    <run>archive1</run>
    ...
```

... that daemon will run on computers called “ics-srv-archive1” as well as “archive1.sns.ornl.gov” etc. The same applies to the “host” tags which specify where a data server should run.

For the example from listing 1.1, we would get these subdirectories:

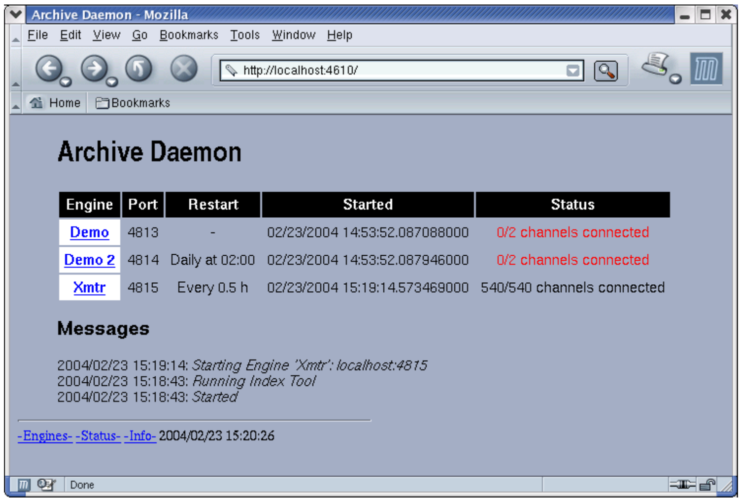
```
RF
RF/llrf
RF/llrf/ASCIIConfig
```

Each directory contains configuration files and scripts explained in the following. Each engine directory also contains an “ASCIIConfig” subdirectory with a script “convert_example.sh” that you might use to create the XML configuration file for the ArchiveEngine from ASCII configuration files, though the engineer responsible for the subsystem is free to use any method of his/her choice as long as the result is a configuration file for the engine that follows the naming convention

Daemon-Name / *Engine-Name* / *Engine-Name*-group.xml ,

If you want to use the ASCIIConfig directory, check section ?? for the format of the ASCII configuration files.

1.2.2 ArchiveDaemon



Archive Daemon

Engine	Port	Restart	Started	Status
Demo	4813	-	02/23/2004 14:53:52.087088000	0/2 channels connected
Demo 2	4814	Daily at 02:00	02/23/2004 14:53:52.087946000	0/2 channels connected
Xmtr	4815	Every 0.5 h	02/23/2004 15:19:14.573469000	540/540 channels connected

Messages

```
2004/02/23 15:19:14: Starting Engine 'Xmtr': localhost:4815
2004/02/23 15:18:43: Running Index Tool
2004/02/23 15:18:43: Started
```

-Engines- -Status- -Info- 2004/02/23 15:20:26

Figure 1.2: Archive Daemon, refer to text.

The ArchiveDaemon is a script that automatically starts, monitors and restarts ArchiveEngines on the local host. It includes a built-in web server, so by listing all the ArchiveEngines that are meant to run on a host in the ArchiveDaemon's configuration file, one can check the status of all these engines on a single web page as shown in Fig. 1.2.

The daemon will attempt to start any ArchiveEngine that it does not find running. In addition, the daemon can periodically stop and restart ArchiveEngines in order to create e.g. daily sub-archives. Furthermore, it adds information about each sub-archive of newly created ArchiveEngines to a mailbox directory so that the index mechanism can create the necessary indices and update the data server configuration.

Before using the ArchiveDaemon, one should be familiar with the configuration of a single ArchiveEngine (sec. ??), and how to start and stop it manually.

Configuration

You will typically *not* directly configure an archive daemon, but instead specify its configuration in the "archiveconfig.xml" file. The "update_archive_tree.pl" script will then generate those daemon configurations needed for the local computer, and ignore those meant to run on other machines.

Assume the configuration shown in listing 1.1, which calls for an “RF” daemon to run a “llrf” engine on the host “archive1”. So on “archive1”, a subdirectory “/arch/RF” will be created with the daemon configuration shown in listing 1.2, which complies with the DTD from listing 1.3.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE daemon SYSTEM "ArchiveDaemon.dtd">
<daemon>
    <port>4900</port>
    <mailbox>/arch/xfer</mailbox>
    <engine directory='llrf'>
        <desc>LLRF</desc>
        <port>4901</port>
        <config>llrf-group.xml</config>
        <restart type='weekly'>We 10:20</restart>
        <dataserver><host>web2</host></dataserver>
    </engine>
    <!-- Typically, more entries follow:
    <engine directory='hprf'>
         ...
    </engine>
     -->
</daemon>
```

Listing 1.2: Example Archive Daemon Configuration

The following is an explanation of the daemon-related tags in the “archiveconfig.xml” file. Compare with the created daemon configuration to see how they get translated.

- run
  Both the daemon and each engine have this element, which specifies the host name where a daemon and engines should run. The names are regular expressions. In principle, this setupone would think about the following setup:

```
<daemon directory='RF'>
    <run>archive[12]</run>
    ...
    <engine directory='llrf'>
        <run>archive1</run>
         ...
    <engine directory='hprf'>
        <run>archive2</run>
```

... where RF/llrf runs on “archive1” and RF/hprf runs on “archive2”, and consequently both of those machine run an “RF” daemon. This has not been tested. Typically, a daemon and its engines are all on the same computer.

The generated daemon config file only includes the configuration needed for the local computer. So instead of duplicating <run>localhost</run>, it is omitted in the daemon config file.

- desc
  A short description.
- port
  Both the daemon and each engine have this mandatory element, which determines the port number of the HTTP server. Section 1.2.2 describes the HTTP server of the daemon, while section ?? explains the engine HTTPD.

  NOTE: The port numbers used by the ArchiveDaemons and all the Archive Engines need to be different. You cannot use the same port number more than once per computer.
- mailbox
  Directory that the daemon uses to communicate with the data server.
- engine
  Specifies the sub-directory for each engine under a daemon.
- config
  This mandatory element of each engine entry contains the path to the configuration file of the respective ArchiveEngine, see section ??.
- restart
  When provided, it specifies when the daemon should re-start an engine.
  - daily
    The element must contain a time in the format “HH:MM” with 24-hour hours HH and minutes MM. One example would be “02:00” for a restart at 2 am each morning.
  - weekly
    Weekly is similar to daily, but using an element that contains the day of the week (Mo, Tu, We, Th, Fr, Sa, Su) in addition to the time on that day in 24-hour format, e.g. “We 08:00”. In this example, the daemon will attempt a restart every Wednesday, 8’o clock in the morning.
  - timed
    In this case, the element needs to contain a start/duration time pair in the format “HH:MM/HH:MM”. The first, pre-slash 24-hour time

    stamp indicates the start time, and the second 24-hour time, trailing the slash, specifies the runtime. The engine will be launched at the requested start time and run for the duration of the runtime. As an example, “08:00/01:00” requests that the daemon starts the engine at 08:00 and stops it after one hour, probably around 09:00.

  - hourly
    The element must contain a number specifying hours: A value of 2.0 will cause a restart every 2 hours. The hourly restart is quite inefficient and primarily meant for testing.

  NOTE: It is advisable to stagger the restart times of your engines such that they don’t all restart at the same day and time in order to reduce the CPU and network load for the ChannelAccess re-connects.

- dataserver
  Specifies the name of the data server host.

Starting, running, stopping daemons and engines

The following scripts, some created by “update_archive_tree.pl” in each daemon and engine subdirectory, are used to control the daemons and engines:

- scripts/start_daemons.pl
  Starts all daemons meant to run on the local computer.

- scripts/stop_daemons.pl Stops all daemons. See “-h” for available options, which includes a method to also stop all engines.

- daemon-dir/run-daemon.sh
  Starts the daemon for this subsystem. The daemon will then start all engines which are not already found running.

- daemon-dir/stop-daemon.sh
  Stops the daemon for this subsystem.

  NOTE: This does not stop the engines, only the daemon. See below for more on engine operation.

- daemon-dir/view-daemon.sh
  Script to run ’lynx’ with the URL of the daemon HTTPD.

- daemon-dir/engine-dir/stop_engine.sh
  Stop the engine.

  NOTE: There is no script to start an individual engine, because the daemon will eventually start any engine that’s not running. So stopping an engine is just another means of triggering a restart.

Each engine subdirectory contains

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- DTD for the ArchiveDaemon Configuration -->
<!ELEMENT daemon (port, mailbox?, engine*)>

<!ELEMENT port    (#PCDATA)>  <!-- TCP port number    -->
<!ELEMENT mailbox (#PCDATA)> <!-- /path                -->

<!ELEMENT engine
          (desc, port, config, restart?, dataserver?)>
<!ATTLIST engine directory CDATA #REQUIRED>

<!ELEMENT desc    (#PCDATA)> <!-- Text               -->
<!ELEMENT config  (#PCDATA)> <!-- path               -->

<!ELEMENT restart (#PCDATA)> <!-- path                -->
<!ATTLIST restart
          type (weekly|daily|hourly|timed) #REQUIRED>
<!-- weekly  Mo|Tu|...|Su HH:MM
     daily   HH:MM
     hourly  (double) hours
     timed   HH:MM/HH:MM
             (start/duration)
  -->

<!ELEMENT dataserver (host?)>
<!ELEMENT host (#PCDATA)>
```

Listing 1.3: XML DTD for the Archive Daemon Configuration

- ArchiveEngine.log
  A log file for the ArchiveEngine running in that directory.
- archive_active.lck
  Lock file of the ArchiveEngine, exists while an engine is running.
- YYYY/MM_DD/index
  A subdirectory for index and data files of the sub-archive. If the ArchiveDaemon is configured to perform daily restarts, the format uses the year, month and day to build the path name.
- current_index
  A soft-link to the currently used index.

Daemon's Web Pages

You can use any web browser to look at the daemon's web pages, which look simiar to Fig. 1.2. The URL follows the format

```
http://host:port
```

where "host" is the name of the computer where the ArchiveDaemon is running, and "Port" is the TCP port that was specifiedin the config file.

The main daemon web page lists all the archive engines that this daemon controls with their status. The first column also contains links to the individual archive engines. The status shows any of the following:

- "N/M channels connected"
  This means the ArchiveEngine is running and responding, telling us that N out of a total of M channels have connected. If not all channels could connect, you might want to follow the link to the individual engine to determine what channels are missing and why: Is an IOC down on purpose? Is an IOC disconnected because of network problems? Does a channel simply not exist, i.e. the engine's configuration is wrong?

- "Not Runnnig"
  This means that the respective ArchiveEngine did not respond when we queried it, and there is no "archive_active.lck" lock file. This combination usually means that the engine is really not running (except for the Note below).

  The first step in debugging would be to check the engine's directory for a log file. Does it indicate why the engine could not start? Then check the daemon's log file. It should list the exact command used to start the engine. You can try that manually to check why it didn't work.

- "Disabled."
  The web interface of the daemon contains a link for each engine that disables the engine. This places a file "DISABLED.txt" in the engine directory and stops the engine. As you might have guessed, the daemon will not attempt to start engines as long as the "DISABLED" file is found. This is a convenient way for temporarily disabling engines without removing them from the daemon's configuration.

- "Unknown. Found lock file"
  This means that the respective ArchiveEngine did not respond when we queried it, but there is an "archive_active.lck" lock file. This could have two reasons. It could mean that the engine is running but it was temporarily unable to respond to the daemon's request. An example would be that the engine is really busy writing and dealing with ChannelAccess, so that its web server had to wait and the daemon timed out. All should be fine again after some time.

If, on the other hand, the situation persists, it usually means that the engine is hung or has crashed, so that it does not respond and the lock file was left behind. See Crashes on page ??.

NOTE: The daemon queries only every once in a while and leave the engines alone for most of the time. Especially after startup, all engines will show up as "Not Running" in the daemon's web page while in fact most of them are already running. Then you will see many disconnected channels while the engines did in fact already connect to all channels. If you are impatient, you can click on the links to the individual engines to get a more up-to-date snapshot of each engine's status.

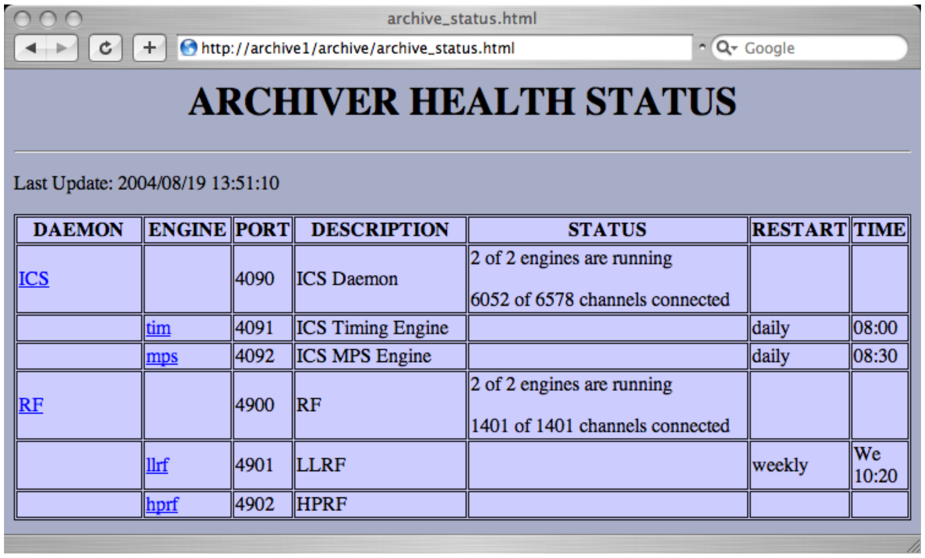
Instead of using the scripts to stop and ArchiveDaemon, one can directly access the "/stop" URL of the daemon's HTTPD, e.g. "http://localhost:4610/stop". Similar to the ArchiveEngine's HTTPD, this URL is not accessible by following links on the HTTPD's web pages. You will have to type the URL. This is to prevent web robots or a monkey who is sitting in front of the computer and clicking on every link from accidentally stopping the daemon. Finally, the daemon will respond to the URL "/postal" by stopping every ArchiveEngine controlled by the daemon, followed by stopping itself. "Postal" is an abbreviation for "POstpone STopping the daemon until ALl engines quit". It is not at all related to "going postal".

Status Information

The following scripts use the HTTPD of the daemon and engine as well as generic Unix tools to create some status overviews. They can be used in "cron" jobs to periodically update web pages or send regular status emails.

- make_archive_infofile.pl
  Creates summary of daemon and engine status, what is running and what is missing, how many channels connected etc. See "-h" option for details.
- make_archive_web.pl
  Similar, but creating a web page.
- engine_write_durations.pl
  Summary of engine performance: How many channels, average values per seconds.
- show_engines.pl
  Parses output of "ps" command for engine related information.
- show_sizes.pl
  Parses output of "du" for size of sub-archives.

Fig. 1.3 shows one example of that web page: A tabular display of what engines are running, how many channels are connected, and more.



ARCHIVER HEALTH STATUS

Last Update: 2004/08/19 13:51:10

DAEMON	ENGINE	PORT	DESCRIPTION	STATUS	RESTART	TIME
ICS		4090	ICS Daemon	2 of 2 engines are running 6052 of 6578 channels connected		
	tim	4091	ICS Timing Engine		daily	08:00
	mps	4092	ICS MPS Engine		daily	08:30
RF		4900	RF	2 of 2 engines are running 1401 of 1401 channels connected		
	llrf	4901	LLRF		weekly	We 10:20
	hprf	4902	HPRF			

Figure 1.3: Example of the archive status web page generated by the make_archive_web.pl script.

1.3 Sub-Archives

Based on the example configuration used in this chapter, the daemon will

1. Start an ArchiveEngine in “RF/llrf” that writes to a sub-archive named after the current day, e.g. “RF/llrf/2004/02_11/index”.
2. Periodically verify if engines that are supposed to run are actually running, attempting to start engines which are found missing.
3. Stop the llrf engine each Wednesday at 10:20, and restart it in a new subdirectory, using the date of the restart for the path name to the new index file.
4. Generate a file in the mailbox directory with information about the “old” sub-archive, the one used by the engine that was stopped.

As a result, we create separate sub-archives for the LLRF, a new one once per week, which provides some insurance against crashes of an ArchiveEngine. If you are paranoid, you can choose daily sub-archives. After running for a while, we will have created sub-archives like these:

```
RF/llrf/2004/02_11/index
RF/llrf/2004/02_18/index
RF/llrf/2004/02_25/index
...
```

In addition to each index file, there will of course also be associated data files, but for retrieval purposes we identify an archive solely by its index file: We can invoke e.g. ArchiveExport with the path to any of the index files. This is, however, inconvenient because we will only see data for one week of that one subsystem at a time. The "serving" computer will therefore use additional index files.

1.4 Serving Computer

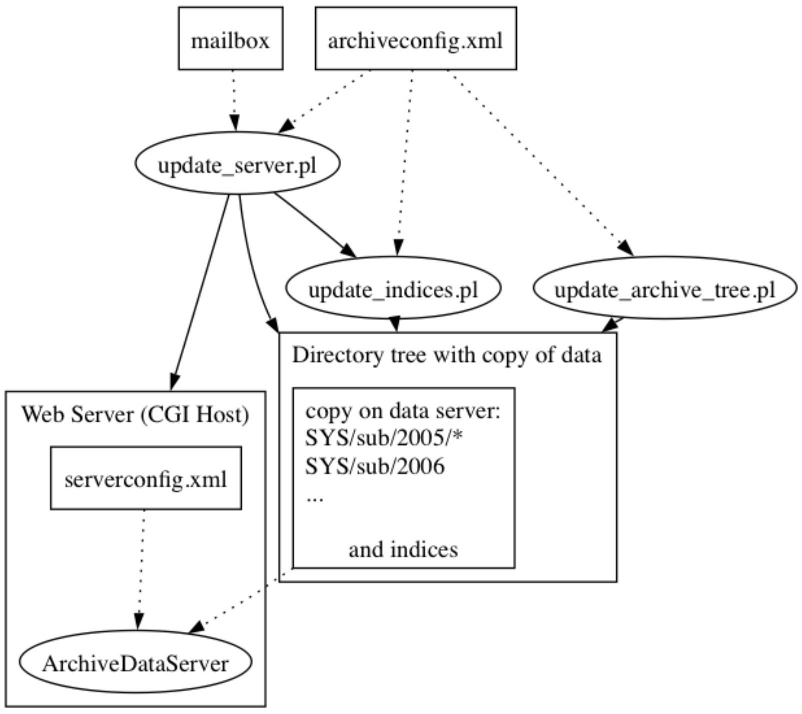


Figure 1.4: Tools used on a serving computer, refer to text.

1.4.1 update_server.pl

.

1.4.2 update_indices.pl

.

1.5 Indices

update_indices.pl

1.6 Data Server

update_server.pl

1.7 Common Tasks

1.7.1 Modify Engine's Request Files

Locate your archive engine directory and the engine configuiration file, for example /arch/RF/llrf/llrf-group.xml for the “llrf” engine maintained by the “RF” daemon. Modify or re-create that configuration. This is often done via a conversion script in /arch/RF/hprf/ASCIIConfig. If you used another method to create the engine configuration, this is a good time to remember what you did.

Then, to actually use that new config file, the engine needs to restart. We could simply wait for the next scheduled restart, in our example the next Wednesday, 10:20. Alternatively, we can run the script /arch/RF/llrf/stop-engine.sh. Watch the RF daemon, for example via /arch/RF/view-daemon.sh. Within a few minutes, it ought to detect that the engine had stopped and then restart it.

1.7.2 Add Engine or Daemon

Edit archiveconfig.xml to define the new engine under an existing demon. Or add a line for a new daemon, then add the new engine under it. Use “xmllint -valid archiveconfig.xml” to check if you preserved the basic XML well-formedness of the document. If you feel lucky today, go right ahead and invoke make_archive_dirs.pl. Per default, it will re-create all daemon and engine directories, so you might want to use the “-s” option to limit its operation to the new or modified subsystem.

In case the daemon was already running, it won't learn about the new engine unless you restart it. So run the “stop-deamon.sh” and then invoke the “run-daemon.sh” script in the daemon directory to start the daemon (which will then start any missing engines).

1.7.3 An Engine isn't running

Check the process list to assert that the engine in question is really not running (on UNIX, try “ps -aux”). Look again. If the engine is actually running but not responding via its HTTPD, remove the process. Check the log file of the engine, generated in the engine subdirectory, for any clues. If you are convinced that the engine is not running, but find an “archive_active.lck” lock file in the engine directory, remove it. Now the daemon should be able to start your engine.

1.7.4 I want to stop a Daemon

Run stop-daemon.sh.

1.7.5 A Daemon isn’t running

Run start-daemon.sh in the daemon directory. If the daemon keeps quitting, check its log file for clues.

THE REST NEEDS REVIEW!

1.7.6 Re-building a Master Index

Whenever you add or remove a sub-archive, the master index in the daemon directory could be obsolete: It might still list data in a sub-archive that you removed, or it might not yet include the new sub-archive. Another szenario: you suspect that the master index is broken, because you can retrieve data from the individual sub-archive but not via the master index. The recipe:

- Stop the daemon
- Check, maybe rebuild indexconfig.xml, either manually or by using the helper script make_indexconfig.pl (see section ??).
- Start the daemon, which causes it to invoke the ArchiveIndexTool.

1.8 Data Management

The generation of daily sub-archives reduces the amount of data that might be lost in case an ArchiveEngine crashes and cannot be restarted by the ArchiveDaemon to one day. In the long run, however, it is advisable to combine the daily sub-archives into bigger ones, for example monthly. The smaller number of sub-archives is easier to handle when it comes to backups. Is also provides slightly better retrieval times. Depending on your situation, monthly archives might either be too big to fit on a CD-ROM or ridiculously small, in which case you should try weekly, bi-weekly, quarterly or other types of sub-archives.

In the following example, we assume that it’s March 2004 and we want to combine the two daily vacuum sub-archives from the previous section into one for the month of February 2004:

```
cd vacuum/2004
mkdir 02_xx
ArchiveDataTool -copy 02_xx/index 02_19/index \
    -e "02/20/2004 02:00:00"
ArchiveDataTool -copy 02_xx/index 02_20/index \
    -s "02/20/2004 02:00:00" -e "02/21/2004 02:00:00"
```

Note that we assume a daily restart at 02:00 and thus we force the Archive-DataTool to only copy values from the time range where we expect the sub-archives to have data. This practice somewhat helps us to remove samples with wrong time stamps that result from Channel Access servers with ill-configured clocks.

There is a perl command make_compress_script.pl that aids in the creation of a shell script for the ArchiveDataTool, but you need to review it carefully before invokation. After successfully combining the daily sub-archives for February 2004 into a monthly 2004/02_xx, we need to

1. Stop the ArchiveDaemon because we are about to edit indexconfig.xml. The ArchiveEngines controlled by the daemon can run on.
2. Edit indexconfig.xml that listed the daily sub-archives for Feb. 2004 and replace them with the single 2004/02_xx/index.
3. Remove or rename the master index file and re-create it with the new indexconfig.xml. This step is required because the ArchiveIndexTool will only add new data blocks to the master index, it will not remove existing ones. Since we no longer want to refer to the daily sub-archives, we need to recreate the master index.
4. Start the ArchiveDaemon again, check its online status.
5. One may now move the daily sub-archives that are no longer required to some temporary location. A month later, when we are convinced that nobody is still trying to use them, we can delete them.