



Lindquist Consulting Technical Report 2007_002

Antelope rtexec.pf style guide Version 1.0

*Dr. Kent Lindquist
Lindquist Consulting, Inc.
November 8, 2007*

This paper entitled "Lindquist Consulting Technical Report 2007_002" ("the Work") is the property of, and copyright 2007, Lindquist Consulting, Inc. ("LCI"). LCI is making the Work available to You subject to the following license terms and in hopes that it will be helpful.

THE WORK IS PROTECTED BY COPYRIGHT AND OTHER APPLICABLE LAW AND ANY USE OF THE WORK OTHER THAN AS EXPRESSLY AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE THAT YOU ARE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THAT THIS LICENSE MAY BE CONSIDERED A CONTRACT, LCI GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

LCI hereby grants to You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to reproduce the Work and to distribute the Work in its original, unmodified form. Your rights to reproduce and/or to distribute the Work are subject to your retention of the above copyright notice, these license terms and the following disclaimer. All other rights are expressly reserved and any use of the Work or of LCI's name or trademarks outside of the terms of this license or except as otherwise required by copyright law are prohibited. This license and the rights granted hereunder will terminate automatically upon any breach by you of the terms of this license.

THE WORK IS PROVIDED BY LCI "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL LCI OR DR. KENT LINDQUIST BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS WORK, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Lindquist Consulting, Inc.

59 College Rd. #7

Fairbanks, AK 99701

USA

<http://www.lindquistconsulting.com>

Printed in the United States of America

Copyright © 2007 Lindquist Consulting, Inc., All Rights Reserved

Introduction

Antelope real-time system control

The Antelope real-time system is managed by a program called *rtexec* [*Antelope* is a commercial software product of Boulder Real-Time Technologies, Inc., <http://www.brtt.com>]. For more detail on *rtexec* see the documentation in section one of the Antelope Unix manual pages, referred to as *rtexec(1)*. The configuration state of the *rtexec* program, and thus of an entire real-time system running on a given machine, is governed by a parameter-file called *rtexec.pf*. The phrase ‘parameter-file’ here refers to a specific, standardized format used widely within Antelope for text configuration files, documented in the Antelope Unix man page *pf(5)*. While the names of the parameters in *rtexec.pf* and their intended values are prescribed by the design of the *rtexec* application, there is nevertheless a fair amount of leeway in the cosmetics of this file for operators setting up and maintaining one or more real-time systems.

Research vs. Operations Grade

This document is intended to provide guidelines for structuring *rtexec.pf* files in an operational setting. ‘Operational setting’ means medium to large-scale deployments that have 24/7 mission-critical monitoring responsibilities (or even small deployments with business requirements of similar importance). The contrast to this would be ‘research settings’, where networks are small and/or not relied upon for continuous robust system output, giving operators wide leeway for ad-hoc setups or exploratory experiments during the development of new technologies. Often, in the initial stages of design of an operational system, approximate measures will be used to get something running, including copying the default *\$ANTELOPE/data/pf/rtexec.pf* file and modifying it just enough so that the system runs with some semblance of the intended output. This alone, however, does not produce a configuration worthy of mission-critical reliance. Perhaps the fundamental characteristic of a ‘research grade’ real-time system is that it is OK for things to be internally cryptic, confusing, or ad-hoc, since the researchers involved can keep many of the broad system-goal details in their heads, sorting out by human thought process what should go where and what should be changed internal to the *rtexec.pf* file each time a change needs to be made (by ‘researchers’ we mean people involved in experimenting with system configurations, not just those focused on after-the-fact publication of scientific journal articles). By contrast, in an operational setting, that structural knowledge must be extracted from the heads of the initial setup personnel, codified into a regular form and built into the processes of the operational organization.

Importance of style

The most important reason for clear, codified structure is to protect the organization from routine or unexpected personnel turnover, and to make it realistic for more than one operator at least to know in exact detail how the system is set up (if not also to be authorized to make necessary changes). Another important reason to regularize *rtexec.pf* files is that even for the principal operators, as the system grows it becomes harder and harder to keep an image of all the ad-hoc naming and configuration ‘in your head’ while you make further changes or diagnose problems. Eventually the system becomes too big to tolerate this; progress is slowed and mistakes are made. This leads to the third main reason to establish some regularity in the way *rtexec.pf* files are written. Without discipline,

it is quite easy to make a succession of small mistakes that go undiscovered for some time. While the consequences may be minimal or not noticed for quite a while, this increased defect density amounts to an unquantified risk that sometime in the future, further changes to other system components will not take into account a particular part of an ad-hoc setup and will cause serious problems, usually at the wrong time.

Basic approach

In large part, cleaning up *rtexec.pf* files amounts to clearly organizing the internal contents, combined with the establishment and rigorous, disciplined implementation of sensible naming conventions. What follows are some guidelines reminiscent of common practice in many successful Antelope-based operations. If you have other structural rules that make more sense for your particular organization or operational requirements, it is fine to supplant the guidelines below, provided the results are at least as well organized and declaratively controlled throughout the system.

Style Guidelines

We recommend the following style guidelines for *rtexec.pf* files:

1) Antelope has three major types of entities: (1) real-time systems, (2) orbservers and related utilities (often prefaced with the letters 'orb'), and (3) databases and database utilities (often prefaced with the letters 'db'). Don't mix the names! Don't have the letters 'orb' in names for real-time systems or databases. Don't use the letters 'db' in names for orbservers or real-time systems. This applies also to macro names (entries in the *Defines* array in *rtexec.pf*), *rtexec* process names, *rtexec* cron job names etc. Don't use 'orb' or 'db' in the names of computers, or of filesystem directories intended to contain entire real-time systems. Generally when operators start using Antelope systems, and when managers start managing Antelope-based operations, there can be some confusion about the distinctions, and more importantly about the proper functions and roles, of a) real-time systems, b) databases, and c) orbs. When people start thinking of orbs as identical to real-time systems, real-time systems as identical to orbs, or orbs as equivalent to databases, they tend to start making technical and architectural decisions which are misguided because guided by a mistaken model. Keeping the names of these entities separate from each other helps educate all involved, and also helps avoid perpetuating confusion that leads to engineering mistakes.

2) The *rtexec* module, as explained in the *rtexec(1)* man page, gives 'nicknames' to each Unix command-line that should be executed, either as a continuously running process (in the *Processes* table) or as a periodic process (in the *crontab* table). For cases where there is only one instance of a given executable running in the real-time system managed by this *rtexec.pf*, the nickname should match the executable name, e.g.

```
orbserver          orbserver -p $ORB orbserver
```

If you have multiple instances of something, the first part of the nickname should be the executable name, then an underscore, then a short mnemonic indicating the basic niche of that executable:

```
orb2orb_params    orb2orb -m '/db/.*' ...
orb2orb_ANF       orb2orb xxx.xxx.xxx.xxx $ORB
orb2orb_IRIS      orb2orb xxx.xxx.xxx.xxx $ORB
```

3) The *Processes* in an *rtexec.pf Processes* table should be listed in a standardized order: first in groups by approximate data-flow through the system, then alphabetical within each group. For example, the basic groups might be (in order; not all systems will have representatives from all groups):

```
orbservers, dbidservers
data acquisition from single stations
data acquisition from groups of stations or other networks
intra-system data flow
real-time processing
archiving
data export
administrative
```

In addition to being cosmetic, this also affects startup order for the system. While in principle the non-volatile reconnection properties of orbserver connections help assure that all data will be collected and processed as intended, in practice it is smooth to have the layered defense of starting the orbbservers first, then getting the data to put in them, then turning on the real-time processes to analyze them, then turning on the modules to archive and export the results.

4) Do not use capital letters in *rtexec.pf* process nicknames. They should be all lowercase, with the possible (questionable) exception of station-names or institutional-name acronyms. One should especially not use alphabetic case to distinguish one process nickname from another, e.g. ‘foo2orb’ vs. ‘FOO2orb’.

5) There should be a one-to-one correspondence between entries in the *Processes* table and entries in the *Run* array. Namely:

A) The *rtexec.pf Run* array should have entries for all items in the *Processes* table.

B) The *Processes* table should have entries for all items in the *Run* array.

C) The *Run* array should not have entries with no correspondents in the *Processes* table.

D) There should not be multiple occurrences of the same process nickname in the *Run* array.

6) The order in which task nicknames are listed in the *Run* array should match the order in which they are listed in the *Processes* table.

7) Long-unused *Processes* should be removed from *rtexec.pf Processes*. If a historical record is needed some form of note-taking (notebook, wiki page, separate file etc.) or version-control system (such as CVS or Subversion) should be employed such that *rtexec.pf* is not full of misleading clutter. It is also best not to keep lots of historic copies in the real-time system run directory itself. Run directories that contain lots of *rtexec.pf_** files (*rtexec.pf_old*, *rtexec.pf_old2*, *rtexec.pf_1997.save*, *rtexec.pf.xpmt*, etc.; similarly, parameter-file subdirectories with the same) can become unwieldy and confusing.

8) Commented-out lines should be removed from the *Processes* table and *Run* array, except for very temporary cases. Again, the *rtexec.pf* file is a bad place to keep notes. One should especially not use comments to turn *Processes* on and off -- use the *Run* array booleans as they were designed.

9) State files should be named after the process nickname they support. For example:

```
orb2orb_arrays orb2orb -S state/orb2orb_arrays $ORB_ARRAYS $ORB
orbmag_local   orbmag -state state/orbmag_local -auth_expr ...
orb2db         orb2db -S state/orb2db $ORB $DB
orb2dbt        orb2dbt -state state/orb2dbt -overwrite $ORB $DB
```

10) If custom parameter files are necessary, they should generally be named after the process nickname under which the process is executing. Clearly there will be exceptions for some programs however it is best to maintain some semblance of regularity. For example:

```
orbampmag_mb   orbampmag -v -pf orbampmag_mb ...
orbampmag_ms05 orbampmag -v -pf orbampmag_ms05 ...
orbampmag_ms10 orbampmag -v -pf orbampmag_ms10 ...
orbampmag_ms20 orbampmag -v -pf orbampmag_ms20 ...
```

11) Do not use “pf” in front of parameter-file names on the command-lines for *Processes* in the *Processes* table or *crontab* table. Similarly, do not include the explicit suffix “.pf”.

WRONG:

```
orbassoc_myregion orbassoc -pf pf/orbassoc_myregion.pf ....
```

CORRECT:

```
orbassoc_myregion orbassoc -pf orbassoc_myregion ...
```

The parameter file library automatically assumes the “.pf” suffix. Similarly, the parameter file library is responsible for finding the parameter file contents in the directories specified by the PFPATH environment variable. For example, parameter files located in the *.pf* subdirectory in which the real-time system is running will be properly found if you are using the standard setting

```
setenv PFPATH $ANTELOPE/data/pf:./pf:.
```

See the Antelope Unix man-page *pf(5)* for further details.

12) If separate files are used for match/reject expression lists to be used in selecting stations, they should be stored in a separate directory of the *rtexec* system *Run* directory, for example in a subdirectory named *select_files*. Each individual match/reject file should be named with a prefix of *match_* or *reject_* and a suffix identical to the process nickname for which it is used. For example:

```
orb2orb_anet   orb2orb -m @select_files/match_orb2orb_anet ...
orb2orb_bnet   orb2orb -r @select_files/reject_orb2orb_bnet ...
```

13) The *Defines* array macros should be named *ORB* or *ORB_** for orbservers, where the optional suffix is short and properly descriptive of the engineering intent. For clarity, it is advisable to use *ORB64* or *ORB64_** for 64-bit orbservers (i.e. those launched with the *orbsrvr64* executable), especially when using a mix of 32-bit and

64-bit orbservers in the same distributed real-time environment. Examples of some macros for orbservers might be *ORB_ALERTS*, *ORB_STATUS*, *ORB_CMD*, *ORB_ACQ* etc. Likewise, macro names for databases in *rtexec.pf* should be *DB* (for the main acquisition database), or in more complex situations (multiple databases for the same *rtexec.pf*) should begin with *DB_** with the suffix indicating basic purpose. For seasoned administrators in an established network these conventions can be relaxed slightly by putting the *ORB* and *DB* strings at the end rather than the beginning, e.g. *PRELIMORB* or *ARCHIVEDB*. However, for beginning networks this is not recommended since experience shows such permissiveness degrades quickly into unconstrained and thus confusing improvisations. Here's an example of a *Defines* array:

```
Defines &Arr{
    ANTELOPE    /opt/antelope/4.9
    DB          db/ournetwork
    ORB64       :
    ORB_CMD     :39872
    ORB_SOH     :6709
}
```

14) The *Defines* array macros should be all uppercase.

15) The *Defines* array macros should be listed in alphabetical order, with the exception of *ANTELOPE* which always comes first.

16) Remove all unused macros in the *Defines* array.

17) The macros in the *Defines* array should be consistently used. For example, if your *Defines* array contains

```
DB          db/ournetwork
```

You shouldn't have a bunch of lines elsewhere in *rtexec.pf* with the hard-wired string *db/ournetwork*. Instead, all such hard-wires should be replaced with *\$DB*.

18) For a single real-time system or a brand-new training system, it is often reasonable to leave the default, instructive comments in *rtexec.pf* in place as a guide to learning. For large, sophisticated operations with many *rtexec.pf* parameter files, these comments become quite a hindrance. They obscure running tasks and make it hard to see what is and is not configured correctly. They obscure critical comments that actually should be in the *rtexec.pf* parameter files, such as explanations of why operational idiosyncrasies require an otherwise unexpected deviation from standard practice. While these comments do provide usage documentation, the same information is still available in a reference printout of the default *rtexec.pf* parameter file, and official explanations of each parameter are available in the manual page of *rtexec(1)*. Finally, if the system operators need these beginner comments constantly present they probably shouldn't be touching the operational system. In general, for large operations, all the standard explanatory comments in *rtexec.pf* should be removed. An exception is the structural comments that show the columns of table entries, and also the section divider separating *rtexec* parameters from *rtm* parameters. Comments used to disambiguate meanings for trained operators, such as the various specifications of units, may also be left in.

19) All columns in the *Processes* table, *Run* array, *crontab* table and anywhere else relevant should be nicely lined

up. (This alignment may degrade with time if edits to *rtexec.pf* are made with the Antelope *rtm(1)* program rather than with a Unix text editor. Such degradation is to be expected, as long as a general effort is made to keep the *rtexec.pf* file easily readable). Clearly there will be a need for a small amount of judgment here; suffice it to say that administration is much more straightforward and mistake-resistant if these tables are easily readable.

20) The *Shutdown_order* table should contain only task names that are actually listed in the *Processes* table; furthermore they should be in the order recommended by the *rtexec(1)* man page:

- A) orb readers (like *orb2db*, *orbassoc*)
- B) orb writers (like *q3302orb*, *orb2orb*, *orbdetect*, etc.)
- C) other programs
- D) *orbserver*, *diskserver*

More recent versions of Antelope allow program names to be put into the *Shutdown_order* table as well as individual task nicknames. This means that, for example, if you have several different instances of *orb2orb* running under several different nicknames, you only have to put the program name *orb2orb* into the *Shutdown_order* table once, rather than listing all the individual nicknames for the *orb2orb* instances.

21) The *Pf_restart* table should contain entries only for task names that are actually listed in the *Processes* table. Furthermore, it may only be used for files that are structured and deployed consistent with the Antelope parameter-file (man-page *pf(5)*) mechanism, i.e. “.pf” files, not for other files such as travel-time grids, files with source-name match and reject expressions etc.

22) The *Parameter_files* array should only have files in it relevant to the real-time system. Notably, the example line

```
a program different.pf
```

should be removed. Similarly, the default, commented-out example of *dbreplay* should be removed from the *Shutdown_when_task_dies* table; unused entries in *Edit_files* should be removed.

23) All tasks that connect to the main *orbserver* should be specified in the *orbtasks* array. Task nicknames not used in this real-time system should be removed from the *orbtasks* array. Properly configuring this array allows the *rtm(1)* application to show latencies and input and output bandwidth usages for each process.

24) It is advisable to set up the *disks* table so the entries correspond to real partitions on the system being used and that the minima specified are reasonable. Similarly, it is nice if the *Buttons* table provides GUI capabilities appropriate for the tasks and function of this real-time system. The *processes* button within the *Buttons* table should be properly configured to use *top*, */usr/dt/bin/sdtprocess*, or other alternative as appropriate to the software on the given system.

25) Frequent use of the *./bin* directory within the real-time system run directory, for executables used either in the *Processes* table or the *crontab* table, is generally a sign that the software-development infrastructure has not yet matured to the point necessary for support of the network's operations. Larger networks at the very least, if not smaller networks also should consider formalizing all elements of code necessary for running their real-time systems into a centralized source-code repository with automated build system. This centralized code base can then be used to populate all relevant machines; the automatic build structure promotes version control for the running system, quick rebuilds in the face of disasters, and relatively smooth upgrades from one version of Antelope to the next. For further details, see for example Lindquist Consulting, Inc. Technical Report 2005_002, "Setting up an Antelope-dependent Software Hierarchy with CVS Repository."

26) Do not change the value of the *pf_revision_time* parameter just because you edit the *rtexec.pf* file. The purpose of this parameter is not to track the contents of the file (if you need that, use a revision control system such as Subversion or CVS instead). The *pf_revision_time* parameter is automatically generated by the Antelope build process so that *rtexec* can make sure the version of the file's structure (what the parameter key names are, what kinds of stuff one should expect to find in them etc.) matches the version of *rtexec* that's currently running.

Additional Notes

An automatic verification program is being considered which would enforce these rules. As a final note, operators may want to rely on the cascading-defaults behavior of the Antelope parameter-files, documented in the *pf(5)* man page; namely it is possible to leave all untouched parameters in the file *\$ANTELOPE/data/pf/rtexec.pf* and put values in the local *./rtexec.pf* only for parameter keys that are locally modified (provided, of course, that your *\$PFPATH* environment variable is set up properly -- again see the *pf(5)* documentation).