Antelope

ARTS configuration and operations manual

Documentation for Antelope Environmental Monitoring Software

software release 4.1

The information in this document has been reviewed and is believed to be reliable. Boulder Real Time Technologies, Inc. reserves the right to make changes at any time and without notice to improve the reliability and function of the software product described herein.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of Boulder Real Time Technologies, Inc.

Copyright © 1998 by Boulder Real Time Technologies, Inc. All rights reserved.

Printed in the United States of America.

November 3, 1998

Boulder Real Time Technologies, Inc.
2045 Broadway Street, Suite 400
Boulder, CO 80302
1.0 Overview .................................................................................................................. 1

2.0 Antelope Real-Time System (ARTS) ............................................................................. 3
   2.1 Example ARTS Configuration: Saudi Arabia National Seismic Network ................. 4
   2.2 ARTS Software Modules .......................................................................................... 9
       2.2.1 Antelope Real-Time System Modules ................................................................. 9
       2.2.2 Executive and Other Real-Time Modules ............................................................ 12
       2.2.3 Contributed ARTS Modules ............................................................................... 12
   2.3 ARTS: Object Ring Buffer ....................................................................................... 14
   2.4 ARTS: ORB Client Modules ..................................................................................... 15
       2.4.1 ORB naming convention .................................................................................... 15
       2.4.2 data packet naming conventions ....................................................................... 16
       2.4.3 source name and time based packet selection ..................................................... 16
       2.4.4 ORB client state information ............................................................................. 17
   2.5 ARTS: ORB Utility Modules ................................................................................... 18
   2.6 ARTS: Data Import Modules .................................................................................... 18
       2.6.1 data import from field stations: qt2orb ................................................................. 19
       2.6.2 data import from other COMSERV: cs2orb ......................................................... 20
       2.6.3 data import from other ORBs: orb2orb ............................................................... 20
   2.7 ARTS: Seismic Processing Modules ....................................................................... 20
   2.8 ARTS: Archiving of Waveforms and Processing Results ......................................... 22
   2.9 ARTS: Executive Module: rtexec ........................................................................... 22
   2.10 ARTS: Interactive Display Modules ....................................................................... 23

3.0 Running the ARTS Alaska Network Demo ................................................................ 25
   3.1 Running the Demo From Your Hard Drive After You Have Installed Antelope .......... 25
   3.2 Controlling the ARTS Demo With rtexec and the rtexec.pf file ............................... 26
   3.3 Running the ARTS Demo With rtm ......................................................................... 33
   3.4 Controlling the ARTS Demo With rtm ..................................................................... 33
       3.4.1 Starting and Stopping the ARTS Session ......................................................... 35
   3.5 Real Time Data Display Using orbmonrt ................................................................. 37
   3.6 Near Real Time Event Display With dbevents .......................................................... 40

4.0 Configuring ARTS For Your Own Network ............................................................. 47
   4.1 Antelope Standard Environment ............................................................................ 47
       4.1.1 Antelope software distribution ......................................................................... 48
       4.1.2 UNIX environment variables ............................................................................ 49
       4.1.3 Antelope parameter files ................................................................................... 49
       4.1.4 on-line help: manual pages, usage lines, dbhelp .......................................... 50
   4.2 Setting Up the rtexec Directory ............................................................................. 51
       4.2.1 rtexec directory structure ................................................................................. 51
       4.2.2 site and instrument characteristics: dbmaster sub-directory ............................ 52
       4.2.3 Parameter file configuration ............................................................................. 53
   4.3 ARTS Field Interface Modules and ORB Import ................................................... 53
       4.3.1 importing data with orb2orb ............................................................................ 53
       4.3.2 data acquisition with qt2orb ............................................................................ 54
       4.3.3 data acquisition with cs2orb ............................................................................ 54
       4.3.4 other field interface modules .......................................................................... 54
   4.4 ARTS Seismic Processing ........................................................................................ 55
       4.4.1 detection and arrival picking using orbdetect ..................................................... 55
4.4.2 network triggering using orbtrigger ................................................................. 59
4.4.3 association of picks and preliminary hypocenter determination using orbassoc ........................................................................................................... 61
4.4.4 local magnitude estimation using orbmag ....................................................... 65
4.4.5 association of automatically determined hypocenters with hypocenters from external catalogs with dbassoc_rt ....................................................... 67
ARTS Configuration and Operations Manual

1.0 Overview

Antelope is a system of software modules that implement acquisition, transport, buffering, processing, archiving and distribution of environmental monitoring information. Antelope software modules are designed to run on both Sun Microsystems SPARC based workstations and on certain Intel based (386, 486, Pentium) PC computers. Regardless of the hardware architecture, Antelope uses the Sun Solaris v. 2.5 (or higher) operating system.

Both real time automated data processing and non-real time batch mode and interactive data processing are provided by Antelope. Also provided is an information system that holds all raw data as well as processing results and other information. Following is a list of functions that Antelope provides.

- **Field Interface Modules** - These are programs that can connect with field sensor/digitizer/datalogger hardware and acquire data. These modules can also obtain state of health information and can be used to control the field units.

- **Data Communications** - A variety of communications protocols are supported for communications with data from the field, including duplex serial, simplex serial and internet or intranet TCP/IP.

- **Data and Information Buffering** - Hard-disk based ring buffers provide deep and non-volatile buffers for both raw data and processing results.

- **Data and Information Flow** - An object oriented ring buffer system provides a mechanism for automated, efficient and error-free flow of raw data and processing results.

- **Automated Data Processing** - Application specific real-time processing modules are provided for extracting the information and knowledge out of the raw data. Processing results are stored back into the same object oriented ring buffers as the raw data.

- **Automated Data and Information Archiving** - Raw data and processing results are automatically archived in real-time into an information system.

- **Real-Time Data and Information Integration, Distribution and Sharing** - Automated mechanisms are provided for bringing in data and information from external sites, merging these data and information into the real-time processing and for distributing or sharing data and information with external sites.
• **Real-Time System Monitoring and Control** - An executive module is provided for monitoring and controlling overall real-time operations.

• **Real-Time Graphical User Interfaces** - Point-and-click modules are provided for overall system monitoring and control, real-time display of raw waveform data and processing results, real-time map displays and monitoring/control of communications and remote field sensor/digitizer/datalogger units.

• **Automated Archive Management** - Modules for automated database migration, tape backup of the database and raw data and archive cleaning are provided. Extraction modules are provided for bringing the data and information from tape back onto disk.

• **Information System Functions** - A full suite of information system functions are provided using an embedded Relational Database Management System.

• **Interactive Review of Automated Processing** - Application specific modules are provided for human graphical interactive review of automated processing. The automated processing results can be verified, deleted, or re-processed.

• **Batch Mode Processing** - Application specific modules are provided for processing or re-processing data and information in the archive information system in a batch mode.

• **Graphical Displays of Archive Data and Information** - Application specific modules are provided for displaying raw data waveforms and processing results in a variety of formats, including trace displays, maps, spreadsheets and statistical plots.

• **Development Tools** - A rich and comprehensive set of software tools are provided that enable users to develop their own processing and display modules, either for the real-time system or for the information system.

One of the fundamental design criteria of Antelope was to adhere to open systems design principles. Our goal was to design a software system that could be easily extended and modified, either by ourselves or by our customers. Accordingly, we made the system highly modular using common standards in the areas of communications protocols, computer operating systems and software engineering. Antelope is documented at a level of detail that is sufficient both for the operator and the developer. We know that every environmental monitoring system is unique and that customization will be necessary to realize the full potential of the system, in terms of the specific application, the network location and the system’s interaction with the society that it serves. Therefore, we have sought to provide a software system that can be easily customized and molded to the requirements of each individual user.

We refer to the real-time part of Antelope as **ARTS**, for **Antelope Real-Time System**. The information system part of Antelope is referred to as **ASIS**, for **Antelope Seismic Information System**. Although most of the important features of Antelope are generic and can be applied to many different environmental monitoring systems, Antelope was originally developed to support seismic monitoring systems and most of the application specific software modules support seismic oriented processing and archiving requirements. In the rest of this manual we will be referring to the specific modules that support seismic processing functions and focus on the **Antelope** real time system, **ARTS**.
2.0 Antelope Real-Time System (ARTS)

The Antelope real-time system, ARTS, brings raw data from the remote field sites in real time to one or more central processing facilities. Automated real time processing of the data is performed and all data and information are automatically merged into long term information system archives.

A diagram showing data flow from the remote field stations, through ARTS and eventually to the archive information system, ASIS, is shown in Figure 1. Data flow is initiated within the field stations from the individual field digitizers. A typical example configuration of a seismic field site would consist of Quanterra Q730 digitizers connected to Strekeisen STS-2 broadband sensors plus equipment for power and communications. In this example the Quanterra digitizers are the source of the digital seismic data. The field stations communicate with one or more ARTS field interface software modules running at the central processing sites. The field interface modules nominally manage all communications to one or more field digitizers through a variety of physical communications interfaces.

![Figure 1. Antelope real-time system data flow.](image)

Within ARTS, data is buffered and transported through a mechanism known as an Object Ring Buffer (ORB), which acts as the heart of ARTS. Each ORB is managed by a single program, orbserver. Field interface modules write all of the data from the field stations into one or more ORBs which can also be used to send commands to the field interface modules and through to the remote digitizers. In addition to inputting data from field sta-
tions, data can also be imported to an ORB from another external ORB or from other centralized data sources, such as COMSERV ring buffers, EARTHWORM ringbuffers, IDA/NRTS data servers and the USGS/LISS data servers.

One feature of the Object Ring Buffer is that it can accommodate any type of data, including raw waveform data as well as parameters from data processing, such as seismic arrival picks and hypocenters. Antelope takes advantage of this feature by using ORBs for transporting both data and processing results. Seismic processing modules are provided by ARTS which implement all of the functions necessary for real time automated detection, picking, association, location, magnitude estimation and archiving. Each processing module runs continuously as a separate program and communicates both input and output through one or more ORBs.

Data export from an ORB is accomplished either through export to another external ORB or with several archiving modules that write both waveform data and processing results into the Antelope seismic information system, ASIS. Further non-real time processing can then be accomplished with the ASIS processing modules. All of the various ARTS modules are normally controlled and monitored through several executive modules. The main ARTS executive module is rtexec which starts and stops all of the other ARTS modules. In addition to rtexec, several executive modules with Graphical User Interfaces (GUIs) provide monitoring and control of Antelope real-time operation.

2.1 Example ARTS Configuration: Saudi Arabia National Seismic Network

The Antelope real-time system can be more easily understood when looking at a real life example. The recently installed Saudi Arabia National Seismic Network uses the Antelope monitoring software as its heart of operations for information processing, distribution and archiving. The Saudi Arabia Network consists of two processing centers, at Jeddah and at Riyadh. The Jeddah center is directly connected to seven field stations through dedicated duplex serial data communications and processes the seven stations as a local sub-network. The Jeddah center sends all continuous waveform data in real time on to the Riyadh processing center. The Riyadh center is directly connected to 25 field stations with dedicated duplex serial data communications. The Riyadh center merges the waveform data from the 7 Jeddah stations in real time with the data from the 25 field stations that it manages directly to form a national network of 32 stations. Most of the field stations use broadband 3-component sensors with a few that use short period 3-component sensors. All of the field stations use Quanterra Q730 digitizers that sample with 24-bit dynamic range at 100 samples per second continuous. No event detectors are run at the remote field sites.

A block diagram describing the ARTS configuration for the Riyadh processing center is shown in figure 2. The ORB instances are shown as pink circles and individual ARTS processing modules (either programs or scripts) are shown as green rounded rectangles. Raw waveform data from the Jeddah sub-network is brought into the Riyadh system through the ARTS program orb2orb and into the main data processing ORB. The pro-
Figure 2. Antelope real-time system at the Riyadh data processing center.
gram $\texttt{orb2orb}$ is a general purpose ARTS program for transferring packets from one ORB to another in real time. Since ORB packets can represent both raw data as well as processing results, $\texttt{orb2orb}$ can copy processing results along with raw data from one ORB to another. However, for this example only the raw data is being copied from the Jeddah sub-network.

The program $\texttt{qt2orb}$ is the ARTS field interface module for bringing raw data from the Quanterra digitizers in the field to the main processing ORB. $\texttt{qt2orb}$ can connect to one or more Quanterra digitizers through a variety of communications connections. For the Riyadh processing center, we use an off-the-shelf piece of equipment called a terminal server that acts as a data concentrator for the 25 duplex serial lines that each connect to a Quanterra digitizer in the field. $\texttt{qt2orb}$ communicates with each of the Quanterra digitizers through the terminal server.

In addition to acquiring the raw seismic waveform data, $\texttt{qt2orb}$ also is used to monitor the state of health of each field station and to remote control the digitizers and the sensors. This is an important function since it allows an operator at the central processing site to make adjustments and fix problems at the field sites without the need to visit the field sites. The monitor/control functions make use of a second ORB, the Quanterra command ORB, that holds the state of health information (including digitizer log messages) and is used as a medium for transferring control requests and responses. The actual control requests for a digitizer are done through a set of ARTS field control modules that the operator uses directly. One of the standard field control modules is the program $\texttt{qtmon}$ that provides a dynamic graphical user interface (GUI) that the operator can use to monitor all aspects of the field station and communications health. Control requests, such as a request to re-center the sensor masses, are initiated by the operator through the $\texttt{qtmon}$ GUI. $\texttt{qtmon}$ then sends the control request to the QT command ORB. $\texttt{qt2orb}$ continuously looks for new control requests on the QT command ORB and receives the new request. $\texttt{qt2orb}$ encodes the control request for the Quanterra digitizer and sends the request to the digitizer. $\texttt{qt2orb}$ waits for a response from the digitizer and when it arrives relays the response through the QT command ORB back to $\texttt{qtmon}$ where it is displayed for the operator.

One of the requirements for the Saudi Arabia National Seismic Network is a set of analog drum recorders for display of selected seismic waveforms. This is accomplished with the use of a standard PC compatible computer running Microsoft’s Windows 95 operating system with an off-the-shelf digital to analog converter board and associated software drivers. An ARTS module, $\texttt{orb2pc}$, written in JAVA, is used to transfer data packets from a special PC transfer ORB on the SUN computer to the PC. The $\texttt{orb2orb}$ instance that fills the PC transfer ORB is configured to reformat all data packets on the fly into a simple generic integer format that will be easy for the PC to handle.

Automatic seismic network processing is accomplished with four ORB client programs, $\texttt{orbdetect}$, $\texttt{orbtrigger}$, $\texttt{orbassoc}$ and $\texttt{orbmag}$. Multi-frequency STA/LTA detection and onset time estimation is accomplished with $\texttt{orbdetect}$ which reads waveform data in real time from the data processing ORB and writes back detection state information for each channel and frequency band. All detection state information is eventually written to the
**Antelope** Seismic Information System as rows in the *detection* table of the relational database. In addition to transmitting raw data, we use the *ORB* for transmitting processing results in the form of *ASIS* table rows. *orbdetect* forms special *ORB* packets that contain the rows for the *detection* table and this procedure for passing processing results and other information is followed through ARTS.

The specification of channels, frequency bands, filters, STA/LTA time windows and detection threshold values are all user configurable and are read at run time by *orbdetect* from a standard *Antelope* parameter file. Detailed configuration parameters for all *Antelope* software modules are specified in a standard format through *Antelope* parameter files.

Network triggers are determined by *orbtrigger* which reads the *detection* rows, as produced by *orbdetect*, from the data processing *ORB* and writes out *ASIS* trigger rows into the same *ORB*. A network trigger is declared whenever detections are seen for a certain minimum number of stations within a minimum time window. The number of stations threshold and trigger time window are user configurable parameters that are specified at run time through a standard *Antelope* parameter file. *ORB* parameter objects, *ORB* packets that follow the same format rules as *Antelope* parameter files, are also used as a mechanism for passing free format parameters and information between various *ORB* client modules. For instance, *orbtrigger*, in addition to writing out *ASIS* trigger rows, also writes out two *ORB* parameter objects for each trigger, one targeted for the module *orbasoc* and the other targeted for *orbspftrigger*. The *orbasoc* *ORB* parameter object contains a list of all onset time estimates from *orbdetect* that fall within the network trigger time window.

Associations of onset time estimates, e.g. pick times, with particular seismic events and their associated phases is done by *orbassoc*. This program reads the *ORB* parameter object written by *orbtrigger* that is targeted for *orbassoc*. Each input *ORB* parameter object contains a candidate list of picks that may by phase arrivals for a particular seismic event. A three dimensional spatial grid search algorithm is used by *orbassoc* to match observed moveout patterns with predicted patterns. If a suitable match is found, then *orbassoc* converts the associated picks into *ASIS* arrival rows, *assoc* rows and an event hypocenter as *ASIS* event and origin rows. All of these processing results are written into the data processing *ORB*.

Real time Richter magnitude estimates are made by *orbmag*. This program looks for *ASIS* origin rows in the data processing *ORB*. For each origin read, *orbmag* determines appropriate time windows for each station and acquires the waveform data for all components from the same data processing *ORB*. Each waveform segment is converted to equivalent drum recorder displacement of a standard Wood-Anderson instrument and the maximum amplitude for the event is determined. These amplitudes are fed into the standard Richter magnitude formula for computing ml values for each station and all of the station ml values are median averaged to get a total network ml estimate. The ml estimate is used to modify the input *origin* row and this modified *origin* row is written back into the data processing *ORB*. 
This sequence of automated near real time processing illustrates the modular nature of **Antelope** and the natural data-driven synchronization between each of the modules. The data processing **ORB** ends up containing all of the raw waveform data, messages passed between **ORB** clients and the processing results in the order in which each piece of information was generated. This provides a complete audit history of all of the automated processing and this entire history is available remotely in real time through normal **ORB** utilities.

All archiving from **ARTS** to the **ASIS** relational database is done through two modules, **orb2db** and **orb2dbt**. All seismic waveform data and the associated database tables are archived with **orb2db**, which reads waveform data packets from the data processing **ORB**, unformats (and decompresses) the data packets, assimilates packets into contiguous channel waveform segments, reformats and recompresses the waveform segments, writes the data segments into binary disk files and continuously updates the appropriate waveform reference tables within the **ASIS**. A number of output waveform formats are supported including Steim compressed SEED (Standard for Exchange of Earthquake Data). **ASIS** database packets are read from the data processing **ORB** by **orb2dbt** and each packet is merged in real time as a row within their respective tables in the main **ASIS** archive database.

The **ARTS** module **orbmonrtd** provides an operator GUI for viewing waveform data as well as processing results in a smoothly scrolling real time display. This module reads waveform data and processing results from the data processing **ORB** and displays the data and information within a graphics window as waveform traces and arrival glyphs. Data and information are displayed as soon as they are read from the input **ORB**. This module provides a high degree of user configurability and can display many channels within a single window, pre-filter data before display and display repeated versions of the same data with different time and amplitude scales.

All of the **ARTS** modules are started, stopped, monitored and controlled by the executive script **rtexec**. A master **ARTS** configuration parameter file is used to describe the entire processing network and all of the module parameter files and command line arguments. When **rtexec** starts, it reads the master configuration parameter file and starts up all of the **ARTS** modules that are defined in the configuration. **rtexec** continuously monitors each process that it has started and restarts processes that die. **rtexec** also continuously monitors the master configuration parameter file. Whenever this file is modified, **rtexec** rereads the file and restarts any processes whose parameters or command line arguments have changed. An operator GUI interface to **rtexec** is provided by **rtm**. This modules also provides continuous monitoring of total and per process computer system resources including CPU usage, memory usage, disk usage, data rates and data latencies. An operator can use **rtm** to start **ARTS**, stop **ARTS**, start/stop/restart individual processes or change process parameters and command line arguments.

Although most of the real time displays are attached directly to the data processing **ORB**, the **ASIS** database archive is updated in a near real time manner and is appropriate for displaying seismic events as points on a map in near real time. This is done with the module **dbevents** which continuously scans the **ASIS** database archive and plots new events as...
they appear. **dbevents** also looks for any changes to existing events, such as newly calculated magnitude estimates, associations with external catalogs, or interactive analyst review, and updates the graphical display to reflect these changes as they occur. An operator can call up individual event waveforms with **dbevent** as record section plots from the ASIS database archive.

## 2.2 ARTS Software Modules

### 2.2.1 Antelope Real-Time System Modules

A list of **Antelope** Real-Time System Modules is given in Table 1.

<table>
<thead>
<tr>
<th>Module</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orbserver</td>
<td>program</td>
<td>This is the server program for managing an <strong>ORB</strong>.</td>
</tr>
<tr>
<td>orbstat</td>
<td><strong>ORB</strong> client program</td>
<td>This program returns various <strong>ORB</strong> status parameters.</td>
</tr>
<tr>
<td>orbreap</td>
<td><strong>ORB</strong> client program</td>
<td>This program will dump data packets from an <strong>ORB</strong>.</td>
</tr>
<tr>
<td>orbpftrigger</td>
<td><strong>ORB</strong> client program</td>
<td>This program will “trigger” the execution of another program whenever a specified parameter file data packet appears in an <strong>ORB</strong>.</td>
</tr>
<tr>
<td>orbcapture</td>
<td><strong>ORB</strong> client program</td>
<td>This program will dump data packets from an <strong>ORB</strong> whenever it sees particular database row objects.</td>
</tr>
<tr>
<td>orblatency</td>
<td><strong>ORB</strong> client program</td>
<td>This program will accumulate <strong>ORB</strong> data packet latency statistics.</td>
</tr>
<tr>
<td>orbsend</td>
<td><strong>ORB</strong> client program</td>
<td>This program will send test data packets to an <strong>ORB</strong>.</td>
</tr>
<tr>
<td>orb2pf</td>
<td><strong>ORB</strong> client program</td>
<td>This program will dump <strong>Antelope</strong> parameter file data packets from an <strong>ORB</strong>.</td>
</tr>
<tr>
<td>orbdisp</td>
<td><strong>ORB</strong> client program</td>
<td>This program will make a simple static waveform display of data from an <strong>ORB</strong>.</td>
</tr>
</tbody>
</table>
TABLE 1. (Continued) Antelope Real-Time System Modules

<table>
<thead>
<tr>
<th>Module</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qt2orb</td>
<td>ORB client program</td>
<td>This program acts to communicate with and control Quanterra digitizers. Control parameter input and data output are all managed through TCP/IP connections to various ORBs (through orbserver instances).</td>
</tr>
<tr>
<td>cs2orb</td>
<td>ORB client program</td>
<td>This program will read data packets from a running instance of COMSERV and transmit the packets to an ORB.</td>
</tr>
<tr>
<td>pf2orb</td>
<td>ORB client program</td>
<td>This program will send Antelope parameter file objects to an ORB.</td>
</tr>
<tr>
<td>db2orb</td>
<td>ORB client program</td>
<td>This program will send data packets derived from an ASIS Datascope database to an ORB. A related program, dbreplay, will do the same but modify the packet times so that they look like real time.</td>
</tr>
<tr>
<td>stream2orb</td>
<td>ORB client program</td>
<td>This program will send raw data packets that were previously dumped with orb2stream to an ORB. There is an option to modify data packet time stamps to look like real time. This program is used mainly for replaying old data packets for demonstrations or debugging.</td>
</tr>
<tr>
<td>orb2orb</td>
<td>ORB client program</td>
<td>This program copies data from one ORB to another ORB.</td>
</tr>
<tr>
<td>qtshell</td>
<td>ORB client program</td>
<td>Cause remote execution of an OS/9 shell within a digitizer. Return the results of the shell execution.</td>
</tr>
<tr>
<td>qtdownload</td>
<td>ORB client program</td>
<td>Download an OS/9 file from a Quanterra digitizer and return the contents of the file.</td>
</tr>
<tr>
<td>qtupload</td>
<td>ORB client program</td>
<td>Upload a UNIX file to a Quanterra digitizer.</td>
</tr>
<tr>
<td>qtmassrecenter</td>
<td>ORB client program</td>
<td>Command a Quanterra digitizer to issue a mass re-center command to the seismic sensor.</td>
</tr>
<tr>
<td>qtset</td>
<td>ORB client program</td>
<td>This program can be used to set internal parameters within qt2orb.</td>
</tr>
<tr>
<td>resetnetblazer</td>
<td>tcl/expect script</td>
<td>This tcl/expect script will log onto a Telebit Netblazer terminal server and cause a serial channel to be reset. This script is run automatically by qt2orb whenever a connection needs to be re-opened.</td>
</tr>
</tbody>
</table>

**Digitizer Control & Monitoring**

**ORB Data Import**
### TABLE 1. (Continued) Antelope Real-Time System Modules

<table>
<thead>
<tr>
<th>Module</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orb2db</td>
<td>ORB client</td>
<td>This program reads real-time waveform data packets from an input ORB, reformats the data into continuous channel waveform files and writes out database rows into an output relational database.</td>
</tr>
<tr>
<td>orb2dbt</td>
<td>ORB client</td>
<td>This program reads real-time database row packets from an input ORB and adds these database rows to an output relational database.</td>
</tr>
<tr>
<td>orb2stream</td>
<td>ORB client</td>
<td>This program will dump raw data packets from an ORB out to an external file. This program is usually used to generate raw data packets for demonstration or debugging purposes.</td>
</tr>
<tr>
<td>orbdetect</td>
<td>ORB client</td>
<td>This program reads real-time data packets from an input ORB and produces a set of single channel detections that are written back to an output ORB.</td>
</tr>
<tr>
<td>orbtrigger</td>
<td>ORB client</td>
<td>This program reads detections from an input ORB and produces number-of-stations/time-window based network triggers that are written to an output ORB.</td>
</tr>
<tr>
<td>orbassoc</td>
<td>ORB client</td>
<td>This program reads network triggers from an input ORB and searches over a set of spatial grid nodes for a preliminary location that will produce the best time clustering of candidate detections after applying P (and optionally S) moveout. Event location, associations and arrival information are written to an output ORB as database packets. The input travel time grid must first be generated with ttgrid.</td>
</tr>
<tr>
<td>orbmag</td>
<td>ORB client</td>
<td>This program reads event locations from orbassoc or orbgenloc and computes preliminary Richter and/or body wave magnitudes. The magnitude information is written to an output ORB as modifications to the input origin rows along with additional database rows to describe station-magnitude, network-magnitude and magnitude waveform measurement information.</td>
</tr>
<tr>
<td>qedd</td>
<td>program</td>
<td>This is a daemon program that polls finger daemons on remote hosts looking for finger-based earthquake catalog entries (ala quake). New entries are added to an output catalog relational database.</td>
</tr>
<tr>
<td>dbassoc_rt</td>
<td>program</td>
<td>This program continuously monitors an input external catalog relational database. Whenever a new entry appears in the external catalog, the new entry is associated against all of the events in an output relational database and successful associations are added to the output database.</td>
</tr>
</tbody>
</table>
2.2.2 Executive and Other Real-Time Modules

The executive modules are listed in Table 2 along with other real-time modules that are not ORB related.

<table>
<thead>
<tr>
<th>Module</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rteexec</td>
<td>perl script</td>
<td>This is the master control module for the real-time system. All other real-time modules (except for the interactive graphical modules) are started and halted by rteexec.</td>
</tr>
<tr>
<td>orbmonrd</td>
<td>ORB client</td>
<td>This graphical user interface (GUI) presents a real-time display of waveform data and processing results as they appear in an ORB.</td>
</tr>
<tr>
<td>qtmon</td>
<td>ORB client</td>
<td>This GUI presents a real-time display of the status of data acquisition with Quanterra digitizers (through a running qt2orb).</td>
</tr>
<tr>
<td>rtm</td>
<td>tcl/tk script</td>
<td>This graphical user interface (GUI) presents a real-time display of the status of the real-time system including software modules, ORBs and computer system parameters, such as CPU usage, memory usage and free disk space. The real-time system can be controlled through rtm and other GUI modules can be started by rtm.</td>
</tr>
<tr>
<td>dbevents</td>
<td>tcl/tk script</td>
<td>This GUI will make real-time plots of event locations on a number of user generated maps. This program will also display waveform data as event record sections.</td>
</tr>
</tbody>
</table>

2.2.3 Contributed ARTS Modules

Included in the Antelope 4.1 distribution are a number of contributed software modules. The ARTS related contributed software programs are listed in Table 3. The contributed programs and software libraries were written by people in the Antelope user community who have generously made these available to all Antelope users. Any license agreements, claims of ownership or copyright statements contained in the Antelope distribution do not apply to any of the contributed software. Please look in the /opt/antelope/4.1/READM file for a complete list of contributed software along with the authors.
<table>
<thead>
<tr>
<th>Module</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>guralp2orb</td>
<td>ORB client program</td>
<td>This program acts to communicate with Guralp digitizers. This program is contributed by Kent Lindquist, University of Alaska, Fairbanks, <a href="mailto:kent@giseis.alaska.edu">kent@giseis.alaska.edu</a>, see <code>guralp2orb(1)</code> man page.</td>
</tr>
<tr>
<td>rddas</td>
<td>ORB client program</td>
<td>This program will read data directly from the serial port of a Reftek datalogger and send data packets to an ORB in real time. Contributed by Marina Glushko, University of California, San Diego, <a href="mailto:glushko@epicenter.ucsd.edu">glushko@epicenter.ucsd.edu</a>, see <code>rddas(1)</code> man page.</td>
</tr>
<tr>
<td>ipd</td>
<td>ORB client program</td>
<td>This program will read data from a UCSD/IGPP modified Reftek datalogger using an intermediate data concentrator (also built by UCSD/IGPP) and send data packets to an ORB in real time. Contributed by Marina Glushko, University of California, San Diego, <a href="mailto:glushko@epicenter.ucsd.edu">glushko@epicenter.ucsd.edu</a>, see <code>ipd(1)</code> man page.</td>
</tr>
<tr>
<td>adsend2orb</td>
<td>ORB client program</td>
<td>This program will take data directly through a UDP connection to an EARTHWORM digitizer PC and send to an ORB. This program is contributed by Kent Lindquist, University of Alaska, Fairbanks, <a href="mailto:kent@giseis.alaska.edu">kent@giseis.alaska.edu</a>, see <code>adsend2orb(1)</code> man page.</td>
</tr>
<tr>
<td>eworm2orb</td>
<td>ORB client program</td>
<td>This program will take data from an EARTHWORM ringbuffer and send to an ORB. This program is contributed by Kent Lindquist, University of Alaska, Fairbanks, <a href="mailto:kent@giseis.alaska.edu">kent@giseis.alaska.edu</a>, see <code>eworm2orb(1)</code> man page.</td>
</tr>
<tr>
<td>liss2orb</td>
<td>ORB client program</td>
<td>This program will connect to a USGS/ASL LISS data server via TCP/IP and bring SEED data packets into an ORB. This program is contributed by Marina Glushko, University of California, San Diego, <a href="mailto:glushko@epicenter.ucsd.edu">glushko@epicenter.ucsd.edu</a>, see <code>liss2orb(1)</code> man page.</td>
</tr>
<tr>
<td>ida2orb</td>
<td>ORB client program</td>
<td>This program will connect to a UCSD/IGPP NRTS data server via TCP/IP and write out data packets to an ORB. This program is contributed by Kent Lindquist, University of Alaska, Fairbanks, <a href="mailto:kent@giseis.alaska.edu">kent@giseis.alaska.edu</a>, see <code>ida2orb(1)</code> man page.</td>
</tr>
</tbody>
</table>
BRTT MAKES NO CLAIMS OF OWNERSHIP FOR ANY OF THE CONTRIBUTED SOFTWARE. THE CONTRIBUTE SOFTWARE IS PROVIDED ON AN "AS IS" OBJECT CODE ONLY BASIS AND BRTT DISCLAIMS ANY RESPONSIBILITIES RELATING TO THE CONTRIBUTED SOFTWARE. THE AUTHORS HAVE GIVEN BRTT PERMISSION TO INCLUDE THIS CONTRIBUTED SOFTWARE IN THE ANTELOPE DISTRIBUTION, PERMISSION FOR YOU TO COPY THIS CONTRIBUTED SOFTWARE ONTO YOUR HARD DRIVE AND PERMISSION FOR YOU TO RUN THIS CONTRIBUTED SOFTWARE ON YOUR COMPUTERS. ANY OTHER USE, INCLUDING REDISTRIBUTION, SHOULD FIRST BE CLEARED WITH THE ORIGINAL AUTHORS. REQUESTS FOR SOURCE CODE SHOULD BE MADE DIRECTLY TO THE ORIGINAL AUTHORS. BRTT WILL PROVIDE NO SUPPORT FOR THIS CONTRIBUTED SOFTWARE. REQUESTS FOR SUPPORT SHOULD BE MADE TO THE ORIGINAL AUTHORS.

### 2.3 ARTS: Object Ring Buffer

The heart of the Antelope real-time system is the Object Ring Buffer, or ORB. The concepts behind an ORB are straightforward: 1) a circular raw data store on disk, 2) a server-client approach to manage the circular data store, and 3) all server-client inter-process communications take place through Internet sockets using TCP/IP. A single ORB is managed by a single server program, orbserver, running on the host computer that is local to the ORB disk store. Client programs can either run on the same computer as orbserver, or on any computers that are available across the Internet.

Multiple client programs from multiple computers can simultaneously write to and/or read from one or more ORBs. All client synchronization is managed by the orbserver program.

The data within an ORB is represented as a set of discrete digital packets. Each packet is identified when it is written to the ORB by two pieces of information; a character string “source” identifier and a time tag. The source identifier is intended to provide information about the type or format of the data packet as well as where the packet originated (typically a network-station-channel code). The orbserver manages a set of reference indices that support source identifier/time tag based requests for data packets. Data packets of dif-

<table>
<thead>
<tr>
<th>Seismic Processing Modules</th>
<th>Module</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orbgenloc</td>
<td>ORB client program</td>
<td>This program provides a generic location capability using traditional inversion algorithms. Locations produced by orbassoc can be fine tuned with orbgenloc. This program is contributed by Gary Pavlis, University of Indiana, <a href="mailto:pavlis@geology.indiana.edu">pavlis@geology.indiana.edu</a>, see orbgenloc(1) man page</td>
<td></td>
</tr>
</tbody>
</table>
ferent format and size can be intermixed within the same ORB. There are no packet ordering requirements for writing packets, such as time ordering. Data packets can contain any type of information, including log messages, multiplexed data, compressed data, processing results, etc. The physical representations of the Ringbuffer itself and the index tables are normal UNIX files. Therefore, the ORB data store and its knowledge of data types and packet times can be large and is non-volatile, in contrast to other real time systems that use shared memory RAM caches to implement circular data stores.

The ORB utility design provides for two important capabilities; real-time data merging and real-time distributed data processing. Data from multiple sources can be merged in a real-time manner into a single ORB. An example would be merging from multiple networks or independent sub-networks of a single network into a combined network ORB, all in real time. In addition to merging raw data, it is possible to merge the processing results into the raw data streams. This allows a single ORB to contain an entire audit history of raw data flow as well as the results of real-time processing. The only requirement for data merging is that communications channels be available to support the real time data flow. The ORB utility support of distributed data processing allows different groups in different locations to access the same data stream in real time for customized processing. A simple ip-address based security scheme is provided to limit ORB access to those specified in a permissions list.

The major task in bringing data from a particular data digitizer through a particular communications protocol into an ORB is to write a program that knows how to communicate with the digitizer, which we refer to as a field interface module. This program can then write data packets to one or more ORBs through standard ORB utility subroutines.

### 2.4 ARTS: ORB Client Modules

The ORB server software module, orbserver, does not know anything about the format or the nature of the data contained within the data packets that it manages. In this way orbserver is data neutral. This is an important ORB feature that allows a heterogeneous mix of information within a single ORB. All of the actual data processing takes place within ORB client software modules that run independently of the orbserver.

All ORB client software modules use a standard software utility library, liborb, for communicating with one or more instances of orbserver that are running either locally or remotely. An important function that liborb provides to the ORB client programs is a reliable and robust interface for maintaining TCP/IP socket connections over indefinite periods of time through an automatic timeout/reconnection scheme that insures quality and continuity of orbserver-client data streams. Another ORB client software utility library, libpkt, is used by ORB client programs to format/unformat and compress/decompress data packets.

#### 2.4.1 ORB naming convention

Since ORBs are managed through standard network based server-client methods, we fol-
low the standards of similar TCP/IP network servers for naming ORBs. At run time, each instance of orbserver is assigned an integer “port” number (through the orbserver command line) which is unique on a per host basis. If no port number is specified when starting orbserver, then a default number is used. All client programs connect to instances of orbserver through ORB names of the form hostname:port where hostname is the host name of the computer where orbserver is running and port is the integer port number. An alternate form for an ORB name is simply hostname where the port number is assumed to be the default port number. Note that the host name localhost exists on all UNIX systems and is a standard alias for the host name of the local computer. Therefore, all ORB client modules can always use the ORB name localhost to reference an instance of orbserver that is running on the same host and was started with the default port number.

2.4.2 data packet naming conventions

All data packets must be assigned character string source names by the ORB client programs that write them into an ORB. The orbserver places no special significance on these data packet source names. However, there is a source naming convention used by all of the ARTS ORB client programs.

Source names of the form /pf/xxxx refer to parameter file objects with parameter object name xxxx. Parameter file objects are ubiquitous within Antelope and will be covered in more detail later. Parameter file objects provide a generic method for storing and retrieving free-format parameter information, normally within ASCII disk files. They are used to configure and control all of the programs within ARTS. Parameter file objects can also reside within ORB data packets and are used extensively by ARTS to pass control and configuration information between the various ORB client modules in a real-time fashion.

Source names of the form /db/xxxx refer to ASIS database rows for table xxxx. All of the information from the real-time system are eventually stored into ASIS relational databases. ORB client programs that produce final processing results for archival into relational databases do this by writing database row packets into an ORB. These packets are read by archival modules and are added to the appropriate relational databases.

Source names of the form /log/net_sta refer to ASCII log messages generated by the Quanterra digitizer at net_sta field station.

All other source names are of the form net_sta_chan/QCDAT and refer to seismic data packets from the field station digitizers. The net_sta_chan refer to the network-station-channel codes and the QCDAT refers to the data packet format (Quanterra Steim compressed data).

2.4.3 source name and time based packet selection

All data packets written into an ORB are tagged with a character string source name and a time stamp. orbserver keeps track of these tags through a set of index files that are transparent to the client modules. Whenever an ORB client module connects to a particular
orbserver, it can request to read a subset of all available data packets through source name selection and time range selection.

ORB client modules make requests for source name selections by sending a source name select key to the orbserver. The source name select key is usually specified in the client module command line arguments or in associated parameter files. Regardless of how it is specified, an ORB source name select key is a character string UNIX regular expression that is used by the orbserver to determine which data packets will be transmitted to the client module; only data packets with source names that match the select key will be transmitted.

One must understand the syntax of UNIX regular expressions, in order to understand how to compose an ORB source name select key. We refer the reader to any of the popular books that describe UNIX for a detailed description of UNIX regular expressions. However, we offer here an abbreviated description to get the reader started.

UNIX regular expressions can be thought of as templates that are compared against particular strings of characters producing either a match or a mismatch. The simplest form of a UNIX regular expression is a verbatim character string, such as /db/arrival. In this case the only character string that would match the expression is the string that is exactly the same as the expression, i.e. /db/arrival. UNIX regular expressions can contain special characters that act as “wild cards” to match with a number of characters. For instance, the period character, “.”, matches against any single character, so that the expression /db/.... would match with /db/arrival or any other string that consisted of /db/ followed immediately by any six characters. The asterisk character, “*”, matches any number of the immediately preceding character, so that /db/.* would match with any string that begins with /db/ followed immediately by any one or more characters (“.” matches any single character and “*” any number of the preceding character, which in this case is “.”, or any character).

Most ORB client modules can position the ORB read pointer according to a specified time stamp. ORB time specifications follow a standard format that is described later in this tutorial.

2.4.4 ORB client state information

In order to facilitate seamless stopping and restarting, many ORB client modules keep state information in external files. These state files are normally specified at run time through command line arguments and contain information for initializing ORB read pointers. The state files are automatically updated by the client modules at regular intervals and whenever the modules are shut down.

All ORB client modules automatically keep track of state information related to the particular client-orbserver connections that are active. If any of these connections go down for any reason, the reconnection logic is automatically initiated and upon successful reconnect, the ORB read pointer is properly initialized to affect seamless data flow. All of
this is transparent to the client modules and requires no special application programming or operational actions.

2.5 ARTS: ORB Utility Modules

The ORB utility modules provide general ORB monitoring and control functions. The most commonly used ORB utility module is orbstat. This program provides one of the simplest methods for determining if a particular orbserver is running. Detailed ORB status parameters can be returned by orbstat including orbserver run time, data latency times, numbers of connections, input/output baud rates, numbers of attached client modules with detailed information for each client and numbers of active data packets broken into packet source names with detailed information for each source. An optional interactive typein interface can be used which provides functionality for examining individual data packets and for commanding orderly shutdown of the attached orbserver.

The ORB utility module orbreap is used to produce ASCII dumps of ORB data packets. These dumps can be in either raw hexadecimal format or the data packets can be unformatted into integer sample values. A module for specifically dumping parameter file data packets is orb2pf.

orbpftrigger is an ORB utility module that provides a mechanism for automatically causing another program to be executed (or “triggered”). The most common use of this program is to effect event waveform segmentation. orbpftrigger reads certain parameter file data packets from an input ORB. Whenever a packet appears, orbpftrigger executes a command specified in its command line arguments after optional substitution of variables read from the parameter file object. orbcapture is similar to orbpftrigger in that it does something in response to reading a data packet from an import ORB. orbcapture looks for certain database row packets which trigger orbcapture to extract a range of data packets from the ORB into an external file.

orblatency is used to monitor data latency. Monitoring data latency, i.e. the difference between the local system time and the data packet time, can be used as an indication of overall system performance. A simple static display of waveform data can be produced by running orbdisp.

2.6 ARTS: Data Import Modules

Data is typically imported into an ORB from either field interface modules or from other ORBs. The field interface modules act as interfaces between ORBs and field station digitizers. For the Saudi Arabian National Seismic Network, all of the field station digitizers are Quanterra Q730 units and the corresponding field interface module is the ORB client program qt2orb. All inter-ORB data transfers are accomplished with the ORB client program orb2orb.
2.6.1 data import from field stations: qt2orb

The ORB field interface module for Quanterra digitizers is qt2orb. This ORB client program handles all communications with Quanterra digitizers. qt2orb can communicate with Quanterra digitizers via several different communications links; 1) using standard TCP/IP socket connections directly to the digitizers, 2) using direct serial connections and 3) using indirect serial connections through terminal server hardware. qt2orb is a multi-threaded Solaris program that can connect to one or more Quanterra digitizers. Each qt2orb-Quanterra connection takes place through one of the supported communications links.

qt2orb uses its own implementation of the "Quanterra Smart Link" (QSL) protocol for communication with each digitizer. This is a simple transmit/acknowledge protocol in which data packets are transmitted by the digitizer with embedded checksums. qt2orb verifies the checksums for each packet and sends an acknowledgment back to the digitizer for each verified packet. Unacknowledged packets are retransmitted by the digitizer until an acknowledgment is received.

All of the data packets from each digitizer are written into two output ORBs; a primary data ORB and a secondary data ORB (which may be the same ORB). All high volume data (seismic waveform data for the H, B and L streams) are written to the primary data ORB. All low volume data (waveform data for the U streams and log messages) are written to the secondary ORB. qt2orb keeps track of status information for each Quanterra connection, including last mass positions, serial baud rate, run time, clock status, latency of last clock sync, data latency and connection status. Status packets are written as parameter file objects every 20 seconds by qt2orb to the primary data ORB.

qt2orb can issue a variety of control messages to each Quanterra digitizer, including mass recenter commands, file upload/download commands and remote execution of shell scripts including reboot. qt2orb, in turn, is remote controlled through both input and output connections to a command ORB. Command packets can be transmitted to qt2orb (and from there to the appropriate Quanterra data logger) via the command ORB and command responses are returned via the command ORB. All commands and responses through the command ORB use parameter file object data packets.

A set of digitizer control modules provide indirect control of the Quanterra digitizers through qt2orb and the associated command ORBs. qtshell will cause the remote execution of an OS/9 shell on a particular Quanterra digitizer with any text responses from the shell execution echoed back. The actual sequence of operations is as follows.

1. qtshell composes a parameter file data packet with the requested shell command and associated arguments.

2. qtshell sends the command parameter file data packet to the qt2orb command ORB. qtshell waits for a response by monitoring the command ORB for a return data packet.

3. qt2orb continuously reads command data packets from the command ORB.
4. When qt2orb reads the shell execution command packet, it sends a special command sequence to the Quanterra digitizer with the shell command and associated arguments.

5. The Quanterra digitizer executes the shell command and returns back to qt2orb (through the normal qt2orb-digitizer communication channel) an execution status and any textual output from the command.

6. qt2orb forms a response parameter file data packet and posts it to the command ORB.

7. qtshell reads the response packet and displays the contents.

Additional digitizer control modules include qtupload, which uploads a local UNIX file to a Quanterra digitizer, qtdownload, which downloads a file from a Quanterra digitizer to a local UNIX file, qtmassrecenter, which causes a Quanterra digitizer to issue a mass recenter command to the attached seismic sensors, and qtset, which sets various internal qt2orb parameters. All of these modules work in a manner similar to qtshell.

These control modules provide powerful remote control, configuration and diagnostic capabilities for the field digitizers and several operational procedures using these modules will be described later in this tutorial.

2.6.2 data import from other COMSERV: cs2orb

Data packets can be imported into an ORB directly from a COMSERV ringbuffer using cs2orb. There are two versions of cs2orb, cs2orb and cs2orb_1_0_1, corresponding respectively to the most recent release (18 August 1997) of the COMSERV user library and to the 1.0.1 release (6 August 1996) of the COMSERV user library. cs2orb acts as a normal COMSERV client program and should be configured so.

2.6.3 data import from other ORBs: orb2orb

Data packets can be easily imported in real time from other ORBs with orb2orb. This ORB client module can both write and read packets to/from remote ORBs using standard TCP/IP socket connections.

Normally, data is copied by orb2orb verbatim from one ORB to another. Care must be taken to avoid packet source name conflicts when copying data from one ORB to another. A packet source name based select key can be specified in the orb2orb command line which limits the packets to those that match the select key. orb2orb can also be configured to do on-the-fly packet unformatting.

2.7 ARTS: Seismic Processing Modules

The ARTS seismic processing modules are standard ORB client modules that implement all of the functions necessary for seismic network processing. Each of these modules posts the processing results back into an ORB as either parameter file data packets or as database row packets. The results of one module are read back as the input to the next module
to implement a processing chain from raw waveforms to finished earthquake bulletins with magnitude estimates.

**orbdetect** runs real-time waveform data, for a set of single channels, through one or more filters and then through a standard STA/LTA detector. Once a detection has been declared for a single filter-channel, a refined arrival onset time estimate is made. All of the detection states and onset time estimates for each filter-channel are written out as database row packets onto an output **ORB**.

**orbtrigger** reads detection information, produced by **orbdetect**, from an input **ORB** and applies a simple minimum-number-of-stations-within-a-specified-time-window rule to determine network triggers. If detections from at least the minimum number of stations fall within the specified time window, then a network trigger is declared and a parameter file data packet is written to an output **ORB** that is read by **orbassoc** and that specifies the detections to be used by the real-time associator.

**orbassoc** reads network trigger detection information, produced by **orbtrigger**, from an input **ORB** and applies a spatial grid searching algorithm to find a preliminary spatial location that will produce predicted P (and, optionally, S) arrivals that most closely match the maximum number of observed detection onset times within the input trigger time window. If enough stations produce detection onset times that are associated in this manner within a user specified time tolerance, then the associated detections are converted to arrivals and the arrivals, associations and corresponding preliminary hypocenter information are written as database row packets into an output **ORB**.

The search grids and their corresponding predicted P and S travel times to each station are stored in external grid files that are memory mapped by **orbassoc** at run time. These grid files are generated by an auxiliary program, **ttgrid**. Each grid file can contain one or more separate grids. Each grid can be either 2D or 3D and can cover local, regional and teleseismic distance ranges. All of the grids in a grid file are searched to find the best association for a particular set of detections. In this way **orbassoc** can reliably discriminate events as being teleseismic, regional or local.

**orbmag** reads event locations, produced by **orbassoc** or **orbgenloc**, obtains the appropriate waveform segments from an **ORB**, makes the appropriate waveform measurements and computes Richter and/or body wave magnitudes. The resulting magnitudes are stored as database row packets in an output **ORB**.

In addition to the **ORB** client modules listed above, several other modules are used in **ARTS** to support seismic processing.

**qedd** polls external earthquake catalog information and posts new earthquakes as they appear into an output database. The mechanism used for obtaining the external catalog data is the ubiquitous "finger quake@foo". This module is normally used to retrieve USGS QED catalogs in real time.
dbassoc_rt polls an input database containing external earthquake catalog information. Whenever a new event appears, dbassoc_rt then associates this event against all of the events in an output database using dbassoc. New event associations are then merged into the output database.

2.8 ARTS: Archiving of Waveforms and Processing Results

Waveform and database row data are archived to disk based files that can be used as permanent stores. ASIS uses a relational database model for archiving waveform data and processing results. The particular relational database management system used by ASIS is Datascope. Subsequent on-demand interactive processing makes use of a variety of Datascope modules. Several ORB client modules are used to transfer data from input ORBs into archive databases.

orb2db reads waveform data packets from an input ORB, uncompresses each packet and re-assimilates the waveform sample data into continuous channel-day disk files. The output waveform files can be written as standard 4096 byte mini-SEED. In addition to writing out waveform files, orb2db also adds rows to the wfdisc table of an output database.

orb2dbt reads database row packets from an input ORB and adds each row to an output database. Add checking is active when rows are added; if a new row’s primary keys match an existing row, then the new row modifies the existing row instead of being added as a new row. In this way existing rows in an output database can be modified through the serial ORB database row packet streams. This mechanism also prevents database corruption if database row packets are inadvertently re-processed.

orb2stream and stream2orb are used primarily to generate raw data packet dumps for later replay as ARTS demonstrations or for debugging purposes. The ARTS Alaska Network demonstration, described later in this manual, is driven by data packets, using stream2orb, that were previously dumped using orb2stream.

2.9 ARTS: Executive Module: rtexec

An ARTS processing node typically consists of one or more ORBs with a number of ORB client modules to implement data import, seismic processing and data archiving. It is possible to start and stop a processing node by manually running each of the processes necessary to implement the node. However, it is much easier and more reliable to control a processing node through the ARTS executive module rtexec.

rtexec is a perl script that starts, controls, monitors and stops the real-time system. The overall real-time system configuration is specified within a parameter file that is dynamically monitored by rtexec. This system configuration parameter file contains a list of the modules to run, the order of execution, all command line arguments, environment vari-
ables and other parameter files associated with each individual module. Upon startup, `rtexec` reads the configuration parameter file, sets the environment variables, and launches each of the individual processes with the specified command line arguments. Execution of each process is continuously monitored by `rtexec`. If a process dies, it is automatically restarted. Detailed log files are kept to capture all of the printed output from each process plus messages generated by `rtexec`.

A special Datascope database is maintained that chronicles all of the process execution history. In addition to monitoring the processes, `rtexec` also continuously monitors the configuration parameter file. Whenever this file is modified, `rtexec` rereads it and changes the execution state accordingly. By modifying this file, an operator can, for example, cause a program to be stopped, a new program to be started, a program to be restarted with new command line arguments, or restarts of all programs after resetting one or more environment variables. For an operational network, `rtexec` is normally automatically run at boot time.

### 2.10 ARTS: Interactive Display Modules

**ARTS** provides a number of animated display modules with graphical user interfaces (GUI) for monitoring and control of the real-time system operations. All of these modules run in the X-windows environment provided as the standard Solaris windowing interface. None of the display modules are *required* for real-time system operation; these modules provide a convenient visual interface for the system operators. Note that since all of the **ARTS** display modules are standard X-window applications, they can take advantage of the network-transparent features of X-window applications; i.e. **ARTS** display modules can use display monitors that are remote to the machine on which the module is running. This important feature effectively increases the number of “heads” that an operator can use for display of real-time system operations and provides remote monitoring capability.

**rtm** provides a comprehensive real-time system graphical monitor along with a graphical user interface (GUI) that acts as a front-end to the `rtexec` configuration parameter file. Upon startup, `rtm` reads the `rtexec` configuration parameter file and makes a graphical display of all of the defined processes. `rtm` also looks at a variety of system resource parameters, including CPU usage, disk space and memory usage, and shows these resources in animated graphical displays. Once started, `rtm` continuously checks and displays system resources and continuously rereads the `rtexec` configuration file to see if any changes have occurred. `rtm` provides a GUI for the network operator to modify the system execution status including system start/stop, start/stop/restart of individual processes, edit a process’ command line arguments followed by restart and edit a process’ parameter file followed by restart. `rtm` provides a convenient interface for starting the other graphical display programs including, `orbmonrtd`, `qtmon` and `dbevents`.

**orbmonrtd** is a tcl/tk script that runs a specially extended version of the tcl/tk interpreter program, `orbwish`. This module provides an animated graphical display and graphical user interface (GUI) that shows the real-time seismic waveform data that is being written.
into one or more ORBs. In addition to the waveform data, certain other ORB data packets are displayed in real time as graphical icons. These “other” data packets are database row packets for the detection, trigger and arrival tables that correspond to single channel detections, network triggers and associated arrival picks.

qtmon is a tcl/tk script that provides a graphical real-time display of the information in qt2orb status packets along with all of the log messages written by qt2orb into secondary data ORBs. Information displayed by qtmon includes last mass positions, serial baud rate, run time, clock status, latency of last clock sync, data latency and connection status for each attached digitizer. qtmon also provides a GUI for issuing selected commands to the Quanterra digitizers.

dbevents polls an input database containing all of the seismic network real-time and reviewed data and processing results. A series of graphical map displays are maintained that show seismic event locations. These displays are dynamic and automatically updated whenever the information in the input origin table changes. Information displayed on the maps includes event locations, event age and association/review status of each event. A text display is also maintained that lists event magnitude.

One or maps can be maintained and displayed by dbevents. Each map is generated externally and read in at run time as a gif file. Map projection and bounding box information is read in from a separate parameter file. A bounding box/priority scheme is used to automatically determine which map should be displayed for a particular event.

dbevents provides a GUI for selecting events for further examination. More detailed textual information, such as depth, can be displayed for a selected event with a pop-up menu. Event waveforms can also be displayed as P-aligned, filtered record section plots.
3.0 Running the ARTS Alaska Network Demo

A good way to become familiar with Antelope and configuring ARTS is to examine and run the ARTS Alaska Network demonstration that is included with the Antelope distribution. The data streams for this demo were originally obtained through a real-time ORB data feed from the Alaska Seismic Network to the BRTT office in Boulder, Colorado. We thank Dr. Roger Hansen and Dr. Kent Lindquist of the University of Alaska Fairbanks, Geophysical Institute for letting us use these data for the ARTS demo.

The Alaska Network is primarily an analog telemetered network with narrow band single component short period sensors. In the demo data stream there are also data from two broadband 3 component digital stations. All of the analog data is digitized in Fairbanks using a USGS Earthworm digitizer and ring buffer. The UAF staff wrote an ORB client module, adsend2orb, which reads data packets from the Earthworm digitizer and writes these packets into an ORB (adsend2orb is included in this distribution as a contributed software module). The ORB transport utility, orb2orb, was used to transport the ORB data packets from Fairbanks, Alaska to Boulder, Colorado in near real time. The events in the ARTS Alaska Network demo were processed at the Boulder BRTT office and were written to event flat files using the ORB utility orb2stream.

3.1 Running the Demo From Your Hard Drive After You Have Installed Antelope

Change your working directory (cd) into the demo directory where you installed the Alaska Network demo data set. If you look at the contents of this directory you will see the following:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>README</td>
<td>Text file describing how to set up and run the demo.</td>
</tr>
<tr>
<td>clean</td>
<td>Cleaning script.</td>
</tr>
<tr>
<td>start</td>
<td>Script for starting the demo.</td>
</tr>
<tr>
<td>ttgrid</td>
<td>Binary data file that contains travel time grids for orbassoc. Generated by the program ttgrid.</td>
</tr>
<tr>
<td>bin</td>
<td>Sub-directory that contains other scripts for running the real-time system.</td>
</tr>
<tr>
<td>db</td>
<td>Sub-directory that contains the Antelope Seismic Information System archive database.</td>
</tr>
<tr>
<td>dbmaster</td>
<td>Sub-directory that contains ASIS database tables relating to site/instrument characteristics and the master lastid table.</td>
</tr>
<tr>
<td>events</td>
<td>Sub-directory that contains the Alaska Network demo event data stream files.</td>
</tr>
<tr>
<td>logs</td>
<td>Sub-directory that contains log files generated by rtexec.</td>
</tr>
<tr>
<td>orb</td>
<td>Sub-directory that contains the ORB files.</td>
</tr>
<tr>
<td>pf</td>
<td>Sub-directory that contains the parameter files for the ARTS software modules.</td>
</tr>
<tr>
<td>rtsys</td>
<td>Sub-directory that contains the real-time database generated by rtexec.</td>
</tr>
</tbody>
</table>
If you have not set up the Antelope environment, then you should source the file `/opt/antelope/4.1/setup.csh` (you will need to run a `csh` compatible shell). You can check this by typing

```
printenv ANTELOPE
```

which should return

```
/opt/antelope/4.1
```

If you installed the Alaska Network demo data set into `/opt/antelope/rtdemo_alaska`, then you can skip the next step. If you installed the demo data set elsewhere, then you will have to edit the `rtexec.pf` file. This is the main Antelope parameter file that configures a `rtexec` session. The ARTS module `rtexec` is the overall ARTS executive module that launches all of the other ARTS programs and acts as a dynamic control interface into these programs. You will have to change the fourth line of the `rtexec.pf` file from

```
ROOT /opt/antelope/rtdemo_alaska
```

to

```
ROOT <wherever-you-installed-the-Alaska-demo-data>
```

Make sure that you specify the `ROOT` variable as an absolute pathname.

### 3.2 Controlling the ARTS Demo With rtexec and the `rtexec.pf` file

**ARTS** processes are defined in the parameter file `rtexec.pf`. This parameter file is continuously scanned by `rtexec` and whenever something changes, `rtexec` will take the appropriate action. The `rtexec.pf` file for the Alaska Network demo is given below.

```plaintext
# The following parameters define the environment in which processes # will execute. All other environment variables will be eliminated.

ANTELOPE /opt/antelope/4.1
ROOT /opt/antelope/rtdemo_alaska
Env &Arr{
  ANTELOPE $ANTELOPE
  PATH $ROOT/bin:$ANTELOPE/bin:/usr/local/bin:/bin:/usr/openwin/bin
  PERLLIB $ANTELOPE/data/perl
  PFPATH $ANTELOPE/data/pf:$ROOT/pf:
  HOME $Original{HOME}
  SCHEMA_DEFAULT rt1.0
  DBLOCKS on
  DISPLAY $Original{DISPLAY}
  ELOG_SIGNALS SIGBUS:SIGSEGV
```
ELOG_DELIVER stderr ./logs/errors@c stderr ./logs/errors ./logs/rtexec@dfn
ELOG_TAG %u %P*log*@l@%u %P*notify*@n@%u %P*complain*@c@%u
%P*fatal*@d@%u %P*fault*@f
}

# The following are the process resource limits
Limit &Arr{
cputime unlimited
filesize unlimited
descriptors unlimited # large for orb2db
stacksize unlimited
datasize unlimited
coredumpsize unlimited # so that we may get a core dump
vmemoryuse unlimited
}

# rtexec keeps some statistics in this database
Database rtsys/rtsys

Startup_tasks&Tbl{
# These are one-shot processes to be run when rtexec first starts.
}

Shutdown_tasks&Tbl{
# These are one-shot processes to be run when rtexec is shutting down.
}

# The Processes list specifies the names and execution lines for each
# subprocess which can be run by rtexec.

# Processes are started in the same order as in the Processes list below.
# While starting processes, rtexec waits Start_period seconds between
# running each process; this is useful when later processes are dependent
# on earlier ones.
Start_period 5

# When a process dies, rtexec restarts it automatically; however it
# also enforces a “cooling off” period of Minimum_period_between_starts.
Minimum_period_between_starts 10

# rtexec gives processes this much time to quit on their
# own after a signal; then it sends a kill -9
Time_to_die 120
# The execution line is run under a shell, so special shell
# characters like `*` should be enclosed in quotes, unless you intend
# they be expanded.

Processes &Tbl{
  orbserver  orbserver -p 32741 orbserver
  stream2orb stream2orb localhost:32741 -repeat events/* -tgap 30.0
          -dattime -events db/qed -sleep 30 -v
  orb2db   orb2db -S state/orb2db -m 'AK.*|IDA.*' localhost:32741
            db/alaska now
  orb2dbt orb2dbt -overwrite -state state/orb2dbt localhost:32741
            db/alaska
  orbdetect orbdetect localhost:32741 orbdetect -out localhost:32741
  orbtrigger orbtrigger localhost:32741 orbtrigger -out localhost:32741
            -v
  orbassoc orbassoc localhost:32741 localhost:32741 orbassoc
            dbmaster/alaska
  orbmag   orbmag -v -select_wf '.*[Z12]' localhost:32741
            localhost:32741 db/alaska
  dbassoc_rt dbassoc_rt -v db/qed db/alaska
}

# Only processes which are named in the Run array and have a non-zero
# value will actually be started.

Run &Arr{
  orbserver  1
  stream2orb 1
  orb2db    1
  orb2dbt   1
  orbdetect 1
  orbtrigger 1
  orbassoc  1
  orbmag    1
  dbassoc_rt 1
}

# During shutdown, send kill signals to processes in the order named
# in Shutdown_order list. Each line can contain multiple processes,
# which will be killed concurrently. All these processes will have died
# or been sent kill -9 signals before later processes are sent signals.
# # Usually, one should shut down processes which read from the orb,
# then processes which write to the orb, and finally, the orbserver
# itself.
# # Processes not listed in the Shutdown table are the last to be sent
# signals.
# # orbserver is usually the last process to be killed.
Shutdown_order &Tb1{
orb2db orb2dbt orbdetect orbtrigger orbassoc orbmag
orbmonrtd
orb2orb stream2orb
tcpserver tcpmonitor snoop
}

# These are cron jobs which will be loaded into the user’s cron tab
# when rtexec is started; the crontab is deleted at any normal stop
# of rtexec.  A kill -9 or a reboot can cause the cron entries to
# remain, however.
#
# These cron jobs are actually run by rtcron which first sets up the
# environment to be the same as the rtexec environment -- cd’s to the
# current directory, redirects stdout and stderr to logs/name,
# and sets the environment and resource limits.

Crontab &Arr{
#example1 hourly /bin/mailx -s hourly danq
#example2 daily /bin/mailx -s ‘daily but not weekly’ danq
#example3 0-59 * * * * /bin/mailx -s ‘minutely and more’ danq
#cleanup daily cd archive ; rtdaily -nv store 4100000
}

# The following parameters are not used directly by rtexec, but
# are instead read and used by rtm.

Network_database dbmaster/alaska # Database containing site, network
# and accumulated
# wfdisc/arrival/... tables
orbnname localhost:32741 # name of orbserver
# (in case a different port is being
# used)
Parameter_files &Arr{ # if the parameter file doesn’t follow the
# convention of being named after either the
# task name or the program name, it should be
# specified here, so that rtm can allow editing
# it.
}

disks &Tb1{
# name file minfree_Mbytes min_kinodes description
root / 20 1 root partition: required by many unix tasks
tmp /tmp 20 1 tmp = swap: indicates memory usage
logs . 20 1 log files
database ./db 20 1 database file
waveforms ./db/1998 3000 1 waveform files
}
The following define the bottom row of buttons in the rtm window:

```tcl
Buttons &Tbl{  
    top    xterm -geom 80x25 -e top
ORB_Clients xterm -geom 132x25 -e orbstat -c localhost:32741 5
ORB_Sources xterm -geom 132x60 -e orbstat -s localhost:32741 5
Trace_Display    orbmonrtd localhost:32741 -hmax 700 -wmax 600
Event_Map dbevents db/alaska
}
```

The `orbtasks` array identifies to rtm which processes read from or write into an ORB. The data latency and data rate gauges will be enabled in the rtm display for these processes.

```tcl
orbtasks &Arr{
    orbserver
    stream2orb
    orb2db
    orb2dbt
    orbdetect
    orbtrigger
    orbassoc
    orbmag
    orbassocdb
    orbmonrtd
}
```

This parameter file specifies the entire ARTS processing configuration for this demo and any changes to the file will be reflected in the state of the processes as they run. The first section is used to define all environment variables and execution paths. A common problem with manual execution of the various ARTS modules is that the user may forget to set environment variables and execution paths correctly. `rtexec` gets around this problem by completely redefining the environment according to the `rtexec.pf` file.

The two most important environment parameters are ANTELOPE, which must be set to the root path of the Antelope software distribution, and ROOT, which must be set to the directory path which contains the required sub-directories and files that were described previously. The actual environment variables are read from a parameter file associative array (read the `PF(3)` manual page for a description of standard parameter file format) named Env. To set up the ARTS Alaska Network demo the first two ANTELOPE and ROOT parameters need to be properly specified and usually the rest of the lines in the Env associative array can be left alone.

The next section specifies process limits in the associative array Limit (which are usually unlimited). The `rtexec` rtsys database is specified by the parameter Database. Startup and shutdown tasks are defined in the tables Startup_tasks and Shutdown_tasks.

The next section is the heart of the ARTS configuration. Here the actual ARTS processes to be run, the order of execution, the order of shutdown, the command line arguments, start time interval, restart time interval and the wait time for a process to die are specified.
The ARTS processes and the order in which they are started are specified in the Processes table. Each line in the table consists of a process name followed by the complete UNIX command line for the process. Each command line is processed after normal shell substitutions. Therefore, the commands should be specified as they would manually from a shell prompt. Manual pages and usage lines should be consulted in order to formulate the individual commands. rtxexec automatically redirects standard out and standard error to log files within the logs sub-directory. For the ARTS Alaska Network demo the following ARTS modules will be executed.

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orbserver</td>
<td>ORB server program used to manage the demo ORB. The -p argument specifies the port number of 32741 in order to avoid conflicts with other orbserver instances on the local host. The name for this ORB will be localhost:32741. Other parameters relating to the size of the ORB, the allowable ORB client IP addresses, the ORB files and various size limits are specified in a program parameter file, orbserver.pf, in the pf sub-directory.</td>
</tr>
<tr>
<td>stream2orb</td>
<td>ORB client module that will read the event data stream files and write the packets in real time to the demo ORB. The -datime argument means to modify the data time tags to look current. The -repeat argument means to repeat playing back the event stream files (in events/*) indefinitely. The -tgap argument means to put a 30 second time gap between events. stream2orb will also generate a simulated external QED catalog database (named qed in the db sub-directory as specified with the -events argument) in near real time. The -sleep argument specifies the wait time in seconds before an event hypocenter is written to the external QED catalog database. This program is normally only used for demo purposes. In a real operating system the data is acquired by an ARTS Field Interface Module, such as qt2orb, and/or it is copied from other remote ORBs with orb2orb.</td>
</tr>
<tr>
<td>orb2db</td>
<td>ORB client module that reads data packets from the demo ORB, assimilates packets into contiguous channel data streams, writes the channel data streams into binary waveform sample files and updates the ASIS relational database tables in the archive database. The -m argument specifies that only data packets with source names beginning with either AK (Alaska Network waveform data) or IDA (IRIS IDA waveform data) will be read. The -S argument specifies a state file. Other parameters relating to output waveform format, data latency issues and buffering are specified in a program parameter file, orb2db.pf, found in the pf sub-directory.</td>
</tr>
<tr>
<td>orb2dbt</td>
<td>ORB client module that reads ASIS database row packets from the demo ORB and merges these database rows into the ASIS relational database tables in the archive database. ASIS database row packets are written to the demo ORB by other ORB client data processing modules, such as orbdetect and orbassoc. The -overwrite argument means to overwrite existing database rows in the archive whenever the primary key of the new row from the ORB matches an existing row in the archive. This mechanism is used by ARTS to update database rows that have already been archived, such as adding the local magnitude to the origin rows after orbmag has processed a new hypocenter. The -state argument specifies a state file.</td>
</tr>
<tr>
<td>Module</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>orbdetect</td>
<td>ORB client module that reads data packets from the demo ORB and performs STA/LTA based detection upon specified channels in specified frequency bands. Detection state changes and onset time estimates (picks) are written back to the demo ORB as detection table ASIS database row packets. Detailed parameters relating to the requested list of channels, the particular frequency bands and the various detection parameters are specified in a program parameter file, <code>orbdetect.pf</code>, in the <code>pf</code> sub-directory.</td>
</tr>
<tr>
<td>orbtrigger</td>
<td>ORB client module that reads detection database row packets from the demo ORB and formulates candidate event gathers based upon a simple time window based concurrency criterion for detections. When an event is declared, event channel time segments are defined by output trigger table ASIS database row packets. <code>orbtrigger</code> also writes out two ORB parameter object packets, <code>/pf/orbpftrigger</code> and <code>/pf/orbassoc</code>. These packets are used as input to the programs <code>orbpftrigger</code> and <code>orbassoc</code>. <code>orbpftrigger</code> is normally used to write out event segmented waveform data into the ASIS archive database (as opposed to writing out the continuous data stream as we are doing in this demo). Detailed parameters relating to the requested list of channels in the gather and the time window parameters are specified in a program parameter file, <code>orbtrigger.pf</code>, in the <code>pf</code> sub-directory.</td>
</tr>
<tr>
<td>orbassoc</td>
<td>ORB client module that reads <code>/pf/orbassoc</code> parameter object packets from the demo ORB and performs a spatial grid search for an approximate hypocenter that optimally matches the pick list in the input parameter packet. If a hypocenter is found, then the hypocenter is written to the demo ORB as event and origin ASIS database row packets. Also written are the associated picks, as arrival database row packets, and the pick-hypocenter associations, as assoc database row packets. Detailed parameters relating to the travel time grid file and other association parameters are specified in a program parameter file, <code>orbassoc.pf</code>, in the <code>pf</code> sub-directory. The travel time grid file must be pre-computed with the program <code>ttgrid</code>.</td>
</tr>
<tr>
<td>orbmag</td>
<td>ORB client module that reads origin database row packets from the demo ORB, determines waveform time windows for a specified list of station-channels, reads the waveform data from the demo ORB, filters the waveform data to a standard Wood-Anderson response, automatically reads the equivalent maximum drum recorder displacement, computes individual station Richter magnitude values, computes mean and median network Richter magnitude averages and writes out the ml value into a modified version of the input origin database row packet (back to the demo ORB). Detailed parameters relating to the channels to process and other local magnitude computation parameters are specified in a program parameter file, <code>orbmag.pf</code>, in the <code>pf</code> sub-directory.</td>
</tr>
<tr>
<td>dbassoc_rt</td>
<td>This program is not an ORB client module but instead continuously monitors an external catalog database for new entries. For this demo it is monitoring <code>qed.origin</code> in the <code>db</code> sub-directory (this ASIS database table is simulated by <code>stream2orb</code>). Normally this table would be produced by <code>qedd</code>.) When a new hypocenter is detected, it is read and associated against the arrival picks in the main ASIS output database. If the association is successful, then the hypocenter from the external catalog is copied to the main ASIS output database as a new origin row along with its associated assoc rows.</td>
</tr>
</tbody>
</table>
Each process is executed by `rtexec` from the same directory as the one in which `rtexec` was started (the directory specified by the ROOT parameter in the `rtexec.pf` file). In order for a process to be started by `rtexec`, its process name must appear in the Run associative array in the `rtexec.pf` file with a value set to 1.

It is important to specify the `Shutdown_order` table properly in order to achieve an orderly and safe shutdown of the system. This table specifies the order in which processes are killed when the system is shut down. Generally speaking, all of the ORB export/archiving modules should be stopped first, such as `orb2db` and `orb2dbt`. These modules need to save state information for seamless restart and they need to be talking to a running `orbserver` before they can do this properly.

The last sections in the `rtexec.pf` file refer to automatic crontab entries (disabled in this demo) and parameters that are read by the companion module `rtm`.

### 3.3 Running the ARTS Demo With `rtm`

An Antelope Real Time System can be started in a number of ways. If an `rtexec` directory has been properly set up, then the operator can go to the `rtexec` directory and simply type the UNIX command `rtexec` to start the system. The system can be stopped by going to the same directory and typing the UNIX command `rtexec -k`.

These commands can be embedded into the system startup scripts so that the ARTS comes up automatically whenever the system boots. In situations like this, we would normally not put any graphics modules into the `rtexec.pf` file, since X-windows may not be operational after an unsupervised system reboot.

For the ARTS Alaska Network demo we will start the system by simply typing `start` in the `rtdemo_alaska` directory. This script cleans up any left over files from previous demo runs, touches files required by `rtm` and `rtexec` and executes `rtm`.

Start the Alaska Network demo by executing the `start` script. This will cause the ARTS real-time monitor, `rtm`, window to appear.

### 3.4 Controlling the ARTS Demo With `rtm`

The ARTS GUI `rtm` provides a point and click interface to `rtexec`, the `rtexec` parameter file, `rtexec.pf`, the other ARTS module parameter files in the `pf` sub-directory and the log files in the `logs` sub-directory. `rtm` also monitors the overall real-time system by displaying information relating to total and per process CPU usage, total and per process memory usage, total and per process data transport rates, total disk usage and per process data latencies. `rtm` acts as a graphical interface to `rtexec`, the ARTS executive module that launches all of the other ARTS programs and acts as a dynamic control interface into these programs.
A few seconds after executing the demo start script, the **rtm** window should appear and look like this.

![Image](image.png)

All of the processes that **rtexec** will execute are listed in the Processing Tasks subwindow. The process status bars are empty, the background is orange and the process ids (pid) are blank indicating that nothing is currently running. The total system CPU usage is shown in the upper panel, one bar per CPU (multi-CPU systems show multiple CPU bars). The total virtual memory usage is shown to the right of the CPU usage. The total bar width corresponds to the total virtual memory space with the red vertical line showing the amount in physical RAM. Total disk usage is shown to the right of the memory usage bar. The file systems shown in this panel are defined in the **rtexec.pf** file in the **disks** table. The total **ORB** status and input/output data rates are shown in the top rightmost panel. The empty displays and orange color indicate that the **orbserver** program is currently not running. The **rtexec.pf** parameter file specifies the entire **ARTS** processing configuration for this demo and any changes to the file will be reflected in the state of the processes as they run. In addition to the **ARTS** programs specified in the **rtexec.pf** Processes table, you can cause several other programs to be run by clicking on the bottom row of buttons, each attached to a particular **ARTS** program execution. We will discuss these programs later.
3.4.1 Starting and Stopping the ARTS Session

The ARTS is started by pressing the “Start” button in rtm, causing the execution of rtexec, which in turn starts everything else. After about a minute the rtm window should look like this.

![ARTS Configuration and Operations Manual](image)

rtexec starts first and it controls everything else. As processes start, their Pid values appear in the rtm window along with CPU and memory usage. The interval used by rtexec between process starts is set by the parameter Start_period in the rtexec.pf file. The main ORB connections panel now shows small input and output data rates and a total of three client connections. The orange color indicates that no data is flowing into the ORB.

Eventually all of the processes are started. Once everything is running you can look at various log files and control the individual processes. Try clicking on the orb2dbt process label in the rtm window. You should see something like this.
The main ORB status panel shows a total of 13 client socket connections, a total input data flow rate of 52.337 kilobits per second and a total output data flow rate of 115.028 kilobits per second. All of the processes are showing process ids (Pid), CPU usage and memory usage. We can also see per process data rates into and out of the ORB and data latencies. When we clicked on the orb2dbt process label, a pull down menu appeared. At the top of this menu is the UNIX command line that is running for this process. We can stop the process, by selecting the Don’t Run item, restart the process, by selecting the Restart item, view its log file, by selecting the View log item, or edit its UNIX command line, by selecting the Edit command line item. For processes that use program parameter files, an additional Edit parameter file item will appear in the menu. Try selecting the View log item. When changes are made to the UNIX command line, rtexec automatically restarts the process to enforce the changes.

Now try clicking on the “Processing Tasks” button to clear the panel and click on the “Network Operation” button. You will see a new panel that shows all of the individual stations in the network with background colors to indicate current data latency. Try clicking on the station label FYU. A pull down menu should appear. Click on the Waveforms item and you should see something like this:
You can investigate each station to see its data latency and rate into the system. You can also bring up a real time waveform display of selected channels or all channels for the station using `orbmonrtd` (described later). There are many other functions within `rtm` that you can explore. You can stop the ARTS session by pressing the “Stop” button (do not do this right now).

### 3.5 Real Time Data Display Using orbmonrtd

The ARTS module for displaying waveform data and ASIS database row packets from the ORB is `orbmonrtd`. This program shows individual data channels (as many as you want) scrolling in real time across the X-window display. The easiest way to start `orbmonrtd` for this demo is to press the “Trace_Display” button at the bottom of the `rtm` window. Your `orbmonrtd` window for the first event in the demo should look like this:
Waveform data appears as the yellow trace lines. You can optionally pre-filter the data before display either through a command line argument or in the `orbmonrtd` parameter file, which can also be used to specify per-channel amplitude and time scales. (The `orbmonrtd` parameter file, `orbmonrtd.pf`, is in the `pf` sub-directory.) The trace amplitude plot scales are fixed and plot clipping prevents data from adjacent traces to overlap or obscure each other. Data packets are plotted as soon as they are received from the ORB and in the same order as they are received. The time scrolling update interval is nominally set at 1 second, but it can be adjusted through the View pulldown menu to as low as 0.1 second. The right hand edge of the display window corresponds to now according to the system clock.

The main display canvas can be arbitrarily large in both the time and channels dimensions. Scroll bars are used to control the visible portion of the canvas. For this demo we have set the total canvas width in time to one hour with a total pixel width of 4000. The amount of visible canvas depends upon the size of the `orbmonrtd` window, which you can resize as desired.
ASIS database row packets are shown as various glyphs depending upon the type of information. As with the waveform data, these glyphs are displayed as soon as they are read in the order in which they are read. Information produced by orbdetect, i.e. detection database rows, are shown as light blue lines. A detection on-off event is shown by short vertical lines to demark the on and off states with a horizontal line to connect them. The associated onset time estimate is shown by a long vertical line. In this demo we only run one frequency band in the detector. However, for the general case where we run more than one frequency band, we show the different bands with different colors (as specified in the orbmonrtd parameter file). Information produced by orbtrigger, i.e. trigger database rows, are shown as red lines. A network trigger on-off event is shown by short vertical lines to demark the on and off states with a horizontal line to connect them. Information produced by orbassoc, arrival database rows, are shown as red arrival flags with the associated phase code in yellow.

Detailed configuration for orbmonrtd is specified in the orbmonrtd.pf file in the pf subdirectory. A listing of the demo orbmonrtd.pf file follows:

```
# orbmonrtd parameter file
# Following specifies channels to plot and plot scales
#
# net_sta_chan orbname          timewin  ampmin    ampmax   width height

sources &Tbl{

AK_BAL_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_BWN_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_CKL_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_CNPM_SHZ     localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_COLA_BHZ     localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_CTG_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_CUT_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_DOT_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_FYU_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_GLB_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_ILS_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_KBM_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_KDOK_BHZ     localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_KLU_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_KNK_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_KTH_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_MGOD_SHZ     localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_OPT_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_PAX_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_PPD_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_PRG_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_RDN_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_SCNM_SHZ     localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_SDNM_SHZ     localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_SIT_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_SNN_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_TMV_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_TOA_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42
AK_TTA_SHZ      localhost:32741 3600.000 -1000.000 1000.000 4000  42

```

The data channels to display are specified in the sources table. Each line consists of a channel source, as a network_station_channel code, the obrname of the ORB that contains the data, the total canvas time window in seconds, the minimum and maximum amplitude values that map into the channel height, the total channel time width in pixels and the channel height in pixels. Note that it is possible to merge data from different ORBs in the same obrmonrtd display. Colors and line widths for the detection glyphs are determined by the filter attribute in the detection row through the detections associative array. Note that blanks in the filter attribute must be mapped into underscores in the obrmonrtd parameter file. Arrival flag colors are determined by the phase attribute of the arrival row through the arrivals associative array. All colors are specified as hexadecimal RGB values (e.g. ff0000 = red, 00ff00=green, 0000ff=blue).

3.6 Near Real Time Event Display With dbevents

A map display showing seismic events as they are determined in near real time is provided by dbevents. This ARTS module is different from the other modules we have described
so far. Instead of reading data and information from the ORB, dbevents reads data and information from the ASIS output database. The map display is kept up to date by polling the ASIS output database at regular intervals. The dbevents map display reflects all changes to the ASIS output database, whether from the ARTS or from analyst review through ASIS. The map display was designed to give the operator a simple indication of the status of the seismic event processing.

In this demo you can start dbevents by pressing the “Event_Map” button at the bottom of the rtmx window. It usually takes several minutes for dbevents to start as it is reading in its map dictionary. As soon as events appear in the ASIS archive database, dbevents will show them on a map. After a few events have been processed, the dbevents display window should look like this:

As events are found by orbassoc, they appear as star symbols that are color coded according to age. When events are successfully associated with hypocenters from external catalogs, their shape changes to a circle. When events are reviewed and relocated, their shape changes to a square. A text list of the events is kept in a separate text panel on the right. In this list the origin time, magnitude, review status and geographic region name are displayed. A review status of “Q” indicates that an event has been associated with a hypo-
center from the USGS Quick Epicenter Determination (QED) catalog. A review status of “r” indicates that the event has been reviewed by an analyst.

Detailed event parameters can be seen by right-click-hold on a map event symbol to display a popup menu. This menu has origin time, latitude, longitude, depth, magnitude, event authors, ASIS event id (evid) and ASIS origin id (orid). This menu also contains a command to delete the event, which can be done by the operator in cases where an event is obviously erroneous and due to false associations of detected noise glitches (generally only useful for analog data).

The base maps in dbevents are read in as bit image gif files. The user can generate base map images with any tools that will produce standard gif files as long as the map projection is either mercator or equal distance. Specification of bounding boxes and projections for a particular base map are done through the dbevents parameter file, dbevents.pf, in the pf sub-directory. A feature of dbevents is that any number of base maps can be used. This allows different maps to be displayed according to where an event occurs. In this demo there are four base maps; a local area equal distance projection map centered on southern Alaska, a regional area equal distance projection map centered on Alaska, a global scale equal distance projection map centered on the Alaska Network, and a global scale mercator projection map.

An event is selected by clicking on the map symbol or the line in the text display. You can scroll through the event list by pressing the up or down arrow keys or by clicking on the NextEvent or PrevEvent buttons. When an event is selected, the base map upon which it is displayed is automatically selected according to a bonding box-priority scheme. Each map has an associated latitude-longitude bounding box and a priority number (specified in the dbevents parameter file). For each base map the selected event is first determined to be inside or outside of the map bounding box. The map that is used is the one for which the event is inside of its bounding box and with the highest priority number. Typically local maps are given a high priority number so that they will take precedence over regional/global scale maps. It is possible to disable the “automap” feature from the Maps pull down menu. You can also manually scroll through the maps in the Maps pull down menu or by pressing the NextMap button.

Event oriented waveform displays can be shown as P-aligned record section plots by selecting an event and then selecting the Waveforms->Show menu item. This will cause the ASIS program dbpick to be run with a remote command link to dbevents. A dbpick trace display window will appear and it will be automatically configured to show the waveforms from the ASIS archive database corresponding to the selected event as a record section plot with the stations closest to the event on top and the event time aligned to the predicted first P arrival time. The arrivals are shown as red flags with the phase code in yellow. The D phase code refers to detection picks that did not fall within the association cluster time window and were not used by orbassoc to locate the event. Sometimes these are legitimate arrivals which can be modified by the review analyst and manually included in the event associations. (Note that all arrival edits are disabled automatically by dbevents for this event trace display; use dbloc2 instead). The light blue time bars and
phase codes refer to predicted arrival times for a variety of phases. All of the standard dbpick features can be used in this display (except for arrival editing).

As events are selected the dbpick trace display will be automatically reconfigured. An example dbpick trace display window for the event selected in the dbevents demo screen dump should look like this:

The dbpick window should be dismissed by selecting the dbevents Waveform->Dismiss item. The dbpick window should be restarted (dbpick makes a static snapshot of the database when it starts and needs to be restarted to see new events that were archived after dbpick was started) by selecting Waveforms->Restart. A set of default configuration options that are sent to dbpick can be edited by selection Waveforms->Options.

The configuration for dbevents is done through the program parameter file, dbevents.pf, located in the pf sub-directory. A listing of the demo dbevents.pf file follows:

```
# Parameter file for dbevents
```
# General default values

maxmapwidth 800           # maximum width of map (in pixels)
maxmapheight 800           # maximum height of map (in pixels)
dbpickgeom 480x662-0+62    # geometry of main dbpick display window
dbpicktypegeom 100x20-0-0  # geometry of dbpick typein window
scsift *:[SB]HZ             # default dbpick station-channel sifter
channels 36                 # default number of dbpick channels to display
twin 180                    # default dbpick display time window
filter 4                    # default dbpick filter index
textwidth 50                # width of text display (in characters)
geometry +0+0                # placement geometry of main window
waittorestart 10             # seconds to wait when restarting dbpick

startdbpick 0               # 1 = start dbpick at program startup
                         # 0 = do not start dbpick at program startup
automap 1                 # 1 = automatically select map
                         # 0 = do not automatically select map
legendon 0                # 1 = display map legend
                         # 0 = do not display map legend

external_catalog_authors &Arr{ # A table describing authors and letter codes
                                # for external associated origins
QED Q                     # QED origins
PDE P                     # PDE origins
REB R                     # REB origins
}

local_catalog_authors &Arr{ # A table describing authors and letter codes
                            # for locally generated origins
BRTT:danny d              # BRTT: danny d
BRTT:vernon v             # BRTT: vernon v
}

update_intervals &Tbl{     # database update intervals menu
1s
5s
10s
1m
10m
30m
1h
}

update_interval 1s         # default database update interval

symbol_colors &Tbl{       # event symbol color vs age definitions
                         # age  unselected_color selected_color
10m red \#ffa0a0
30m orange orange
1h yellow yellow
2h \#00d000 green
6h blue \#a0a0ff
0 \#808080 \#d0d0d0
}

maps &Arr{                # map definitions
  global_edp &Arr{        # This is a typical global equal-distance projection map (edp)
    file ANTELOPE/data/maps/images/global_alaska.gif # map filename
    format gif             # map format
    proj edp               # map projection
    latc 64.87             # latitude of projection center
    lonc -147.85           # longitude of projection center
  }
}
```
xdelmin -150.0  # west dist. from center (deg)
xdelmax 150.0   # east dist. from center (deg)
ydelmin -150.0  # south dist. from center (deg)
ydelmax 150.0   # north dist. from center (deg)
symsize 10      # event symbol size (in pixels)
latminbb -90.0  # approximate minimum latitude
    # (for determining bounding box)
latmaxbb 90.0   # approximate maximum latitude
    # (for determining bounding box)
lonminbb -180.0 # approximate minimum longitude
    # (for determining bounding box)
lonmaxbb 180.0  # approximate maximum longitude
    # (for determining bounding box)
priority 1      # display priority
}
global_merc &Arr{}  # This is a typical global mercator projection map (merc)
file ANTELOPE/data/maps/images/global.gif  # map filename
format gif          # map format
proj merc           # map projection
latmin -78.0        # latitude at bottom edge
latmax 78.0         # latitude at top edge
lonmin -180.0       # longitude at left edge
lonmax 180.0        # longitude at right edge
symsize8            # event symbol size (in pixels)
latminbb -90.0      # approximate minimum latitude
    # (for determining bounding region)
latmaxbb 90.0       # approximate maximum latitude
    # (for determining bounding region)
lonminbb -180.0     # approximate minimum longitude
    # (for determining bounding region)
lonmaxbb 180.0      # approximate maximum longitude
    # (for determining bounding region)
priority 0          # display priority
}
alaska &Arr{}      # Alaska state scale map
file ANTELOPE/data/maps/images/alaska_topo.gif  # map filename
format gif          # map format
proj edp            # map projection
latc 65.0           # approximate minimum latitude
lonc -148.0         # approximate minimum longitude
xdelmin -19.0045    # west dist. from center (deg)
xdelmax 11.03       # east dist. from center (deg)
ydelmin -16.0956   # south dist. from center (deg)
ydelmax 10.5077     # north dist. from center (deg)
symsize10           # event symbol size (in pixels)
latminbb 45.0       # approximate minimum latitude
    # (for determining bounding region)
latmaxbb 80.4       # approximate maximum latitude
    # (for determining bounding region)
lonminbb -180.0     # approximate minimum longitude
    # (for determining bounding region)
lonmaxbb -130.8     # approximate maximum longitude
    # (for determining bounding region)
priority 3          # display priority
}
alaska_south &Arr{} # Southern Alaska local scale map
file ANTELOPE/data/maps/images/alaska_south_topo.gif  # map filename
format gif          # map format
proj edp            # map projection
latc 65.0           # approximate minimum latitude
lonc -148.0         # approximate minimum longitude
xdelmin -6.8797     # west dist. from center (deg)
xdelmax 6.6528      # east dist. from center (deg)
ydelmin -9.3625     # south dist. from center (deg)
ydelmax 3.9628      # north dist. from center (deg)
symsize10           # event symbol size (in pixels)
latminbb 57.0       # approximate minimum latitude
    # (for determining bounding region)
latmaxbb 67.0       # approximate maximum latitude
    # (for determining bounding region)
```
The first group of parameters define the dbevents window geometry, the dbpick window geometry, various default dbpick commands and flags that specify the initial state of dbevents. Next comes several associative arrays and tables that define author-to-letter mappings, age-to-symbol color mappings and the menu of update intervals. The last section specifies the base maps that dbevents will display in the associative array maps. For each base map, there is an associative array entry within maps with the key set to a map name and the value another associative array with parameters for that map. Each base map associative array must include parameters to define the map gif file, the file format (only gif is allowed for now), the map projection (only edp and merc are allowed for now), map world coordinates, a latitude-longitude bounding box (for the automap feature), a priority (for the automap feature) and an event symbol size in pixels. Map world coordinates are used, along with the projection, to determine the transformation from event latitude-longitude to X-Y pixel coordinates. For equal distance projection, the world coordinates are the latitude and longitude of the equal distance projection reference point and a set of east-west and north-south distance coordinates in degrees, relative to the reference point, that map to the corners of the gif image. For mercator projection, the world coordinates are the latitude and longitude that map to the corners of the gif image.
4.0 Configuring ARTS For Your Own Network

The ARTS Alaska Network demo shows how ARTS can be used to completely automate seismic network operations. Once ARTS has been properly configured for a particular network, the job of running ARTS is relatively straightforward. The major task in using ARTS for a particular seismic network is configuring ARTS. In this section we describe how to configure ARTS for your own needs.

The first step is to read this manual and become familiar with the ARTS structure and software modules. It is also necessary to become familiar with the Antelope standard environment, used throughout all of the Antelope software modules (this is described in the following section). The next step is to run the ARTS Alaska Network demo, described in section 3.0. This can be used as a fully operational example template.

We suggest that you start with a simple ARTS configuration and build up to a more complex configuration by adding modules, one at a time. A starting configuration can consist of ORB import modules and an archiving module. You can configure orbmonrtd for display of the data. Once this is running properly, you can add seismic processing modules. You should look carefully at how each module is performing and be prepared to modify module parameters to achieve an effective and efficiently running system.

4.1 Antelope Standard Environment

A number of standard directory paths and methods are used throughout Antelope for software configuration, program parameters and on-line help. It is important for the user to become familiar with the Antelope standard environment.
4.1.1 Antelope software distribution

Antelope software must be installed in the /opt UNIX file system. The root directory for Antelope is /opt/antelope. Below is the Antelope software distribution directory structure.

The Antelope distribution contains a distribution release sub-directory (4.1 in this example) and complete versions of tcl/tk, both version 7.4/4.0 (tcl7.4tk4.0) and 8.0 (tcltk8.0). Both versions of tcl/tk are needed for the current release of Antelope and they should be installed as they appear on the distribution CD.

The main Antelope software root directory is /opt/antelope/4.1. Within /opt/antelope/4.1 are the following sub-directories.

- **bin**
  Contains Antelope executable binary files and scripts.

- **data**
  Contains default data, schema, parameter and other files for Antelope.

- **include**
  Contains include files for building new Antelope application programs.

- **lib**
  Contains dynamic binary libraries for Antelope and static binary libraries for building new Antelope application programs.

- **man**
  Contains all of the on-line manual pages for Antelope.

- **perl**
  Contains the perl distribution used by Antelope.
Also within /opt/antelope/4.1 is a C-shell source file, setup.csh, which should be sourced when a user logs into the system. This file will properly set all of the standard Antelope environment variables and search paths.

The /opt/antelope/4.1/data sub-directory contains a number of other sub-directories that are important in the use of Antelope. In particular, the pf sub-directory contains a set of default parameter files for all programs that need program parameter files. These files can be used as templates for user configured parameter files. This sub-directory also must contain a special parameter file, license.pf, which specifies the Antelope license keys. All of the node locked Antelope software modules will look for this file to find valid license keys.

### 4.1.2 UNIX environment variables

A number of UNIX environment variables are used by the Antelope software which specify standard directory paths and other default information, such as the default database schema. A list of the standard Antelope environment variables is given below. Note that the use of setup.csh, described previously, will properly set all of these automatically.

- **ANTELOPE**
  This variable should be set to the Antelope software release root directory (/opt/antelope/4.1). Setting this properly should allow most of the software to find the data files required.

- **PFPATH**
  Some programs require input from a parameter file. This variable overrides the default search path to the parameter files: $(ANTELOPE)/data/pf:. Parameter file search paths are described in section 4.1.3.

- **SCHEMA_DIR**
  This path overrides the default location, $(ANTELOPE)/data/schemas2, for the ASIS Datascope schema files. These files define the database schema used by most of the Antelope programs.

- **SCHEMA_DEFAULT**
  This is the name of the default schema to use when no descriptor file is present. The normal default is “css3.0”.

- **TAUP_PATH**
  This variable overrides the default directory, $(ANTELOPE)/data/tables/taup_ttimes, for the Buland-Kennett tau-p travel time tables.

- **TAUP_TABLE**
  This variable overrides the default Buland-Kennett tau-p travel time table, “iasp91”.

### 4.1.3 Antelope parameter files

Many Antelope software modules use a standard method for reading in detailed parameters at run time from what we call Antelope parameter files (this is extended within ARTS so that Antelope parameter file information can also be read directly as ORB...
packets). An **Antelope** parameter file is a free format specification of character string, numerical, table and associative array parameters. There are many examples of parameter files within this manual. Numerical parameters may be either integer or floating point. Table parameters contain a list of values. Associative array parameters contain key-value pairs. A detailed description of parameter file syntax is in the manual page **PF(3)**.

All **Antelope** software modules that read parameter files look for these files along a particular search path specified by the environment variable PFPATH. The default value of PFPATH is `$(ANTELOPE)/data/pf:` which causes programs to first look in `$(ANTELOPE)/data/pf` and then in the current working directory. Programs read all occurrences of a particular parameter file along the search path, overwriting parameter values from files early in the search path with values from files later in the search path. This provides a mechanism for setting “boilerplate” program parameters values in the template parameter file in `$(ANTELOPE)/data/pf`. The user can then make a parameter file in the current working directory with only the parameters set that are required for that particular execution. This provides a simple and standard means for setting program parameters dynamically at run time and discourages the practice of “hard wiring” program parameters into the compiled code.

### 4.1.4 on-line help: manual pages, usage lines, dbhelp

On-line help is provided for all **Antelope** software modules through standard UNIX manual pages. UNIX manual sub-directories are provided for all **Antelope** software distributions and the user should add these paths into their MANPATH environment variable. We also encourage users to keep UNIX man page indices by running the standard UNIX command **catman**.

Probable the easiest on-line help is the command usage line. We provide usage lines for all **Antelope** software modules. The simplest way to get the usage line printed is to type `program_name -usage`. Detailed instructions can then be found by typing `man program_name`.

When dealing with **ASIS Datascope** databases, it is necessary to know something about the particular database schema being used. The **ASIS** module **dbhelp** provides a point and clock GUI interface to the **Datascope** schema descriptor files. these file provide detailed information on the various schemas that are currently defined. Try typing **dbhelp** to get on-line help on the **ASIS** database schemas.

We have provided all of the man pages and tutorial documents in HTML form. You can use your web browser program to access these documents. Try running `netscape /opt/antelope/4.1/man/manpages.html` to get a completely indexed and cross-referenced HTML display of the man pages. You can find the HTML versions of the tutorial manuals on the distribution CD in the `doc` sub-directory. Note that web browsers have a tendency to use up all of the available free color
cells. When you are running a web browser, you may encounter difficulties with other Antelope programs that will not run without sufficient free color cells.

4.2 Setting Up the rtexec Directory

Normally the first step in setting up a new ARTS configuration is to create a new sub-directory for running the ARTS executive module rtexec. This directory must be properly populated with other sub-directories and a variety of Dat ascope database tables, program parameter files and other ancillary data files needed by the ARTS modules. Once all of this has been done, and assuming that it has been done correctly, then the job of running ARTS is easy.

This setup procedure is described in detail in the UNIX manual page entitled rtexec_setup (in volume 5). You should print out his man page and read it carefully.

4.2.1 rtexec directory structure

Following is a typical rtexec directory structure.

```
rt       bin       dbname
  db       year
       dbmaster       dbname.affiliation
                           dbname.calibration
                           dbname.instrument
                           dbname.lastid
                           dbname.network
                           dbname.sensor
                           dbname.site
                           dbname.sitechan
       rtsys       rtsys
          rtsys.history
                      rtsys.process
       state
       rtxexec.pf
       rtm.pf
       ttgrid
```

The various sub-directories and files required by rtexec are defined below.

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bin</code></td>
<td>Sub-directory that contains scripts run by rtexec.</td>
</tr>
<tr>
<td><code>db</code></td>
<td>Sub-directory that contains the Antelope Seismic Information System archive database.</td>
</tr>
</tbody>
</table>
In order to configure ARTS for a new network it is necessary to create a directory structure similar to the one given above. You can use the ARTS Alaska Network demo as a working template.

4.2.2 site and instrument characteristics: dbmaster sub-directory

It will be necessary to populate the dbmaster sub-directory with ASIS Datascope database tables that describe the network, station and instrument characteristics. Generally, this should be done before any attempts are made to run ARTS for your network. There are a number of ASIS software tools that can be used to create these tables. The tables that need to be created are all part of the ASIS Datascope modified CSS3.0 schema (you can get help on the CSS3.0 schema by running dbhelp).

<table>
<thead>
<tr>
<th>db/dbname</th>
<th>ASIS database descriptor file that describes the ASIS archive database. (Replace dbname with the actual database name, e.g. alaska.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>db/year</td>
<td>Sub-directory that will contain binary archive waveform sample files. (Replace year with the actual year, e.g. 1998.)</td>
</tr>
<tr>
<td>dbmaster</td>
<td>Sub-directory that contains ASIS database tables relating to site/instrument characteristics and the master lastid table.</td>
</tr>
<tr>
<td>dbmaster/dbname.*</td>
<td>ASIS archive database tables relating to site/instrument characteristics and the master lastid table. (Replace dbname with the actual database name, e.g. alaska.)</td>
</tr>
<tr>
<td>logs</td>
<td>Sub-directory that contains log files generated by rtxec.</td>
</tr>
<tr>
<td>orb</td>
<td>Sub-directory that contains the ORB files.</td>
</tr>
<tr>
<td>pf</td>
<td>Sub-directory that contains the parameter files for the ARTS software modules controlled by rtxec.</td>
</tr>
<tr>
<td>rtsys</td>
<td>Sub-directory that contains the real-time database generated by rtxec.</td>
</tr>
<tr>
<td>rtsys/rtsys</td>
<td>ASIS Datascope database descriptor file for the rtsys database.</td>
</tr>
<tr>
<td>rtsys/rtsys.history</td>
<td>rtsys database history table.</td>
</tr>
<tr>
<td>rtsys/rtsys.process</td>
<td>rtsys database process table.</td>
</tr>
<tr>
<td>state</td>
<td>Sub-directory that contains state files for ARTS software modules.</td>
</tr>
<tr>
<td>rtxexec.pf</td>
<td>Main ARTS configuration parameter file for rtxexec.</td>
</tr>
<tr>
<td>rtm.pf</td>
<td>Parameter file for rtm.</td>
</tr>
<tr>
<td>ttgrid</td>
<td>Binary grid travel time file for orbassoc.</td>
</tr>
</tbody>
</table>

dbname.affiliation This table describes the affiliations of stations to networks.

database.calibration This table describes the overall sensitivity (or calib) for each station-channel as a function of time. It is used by qt2orb and orbmag.

database.instrument This table describes distinct instrument response functions. It references external files that contain the actual instrument response coefficients. It also contains the sensor type (displacement, velocity, acceleration). It is used by qt2orb and orbmag.

database.lastid This table is very important and holds the next available database integer ids. It is used by orbtrigger, orbassoc and orb2db.

database.network This table describes the network name and code.
It is very important to make sure that there is a `dbname.lastid` file in the `dbmaster` subdirectory before starting `rtexec` for the first time. As described in the `rtexec_setup(5)` manual page, this can be done by running `dbnextid`. Another easy way is to create an empty file with the UNIX `touch` command. Once the `dbmaster/dbname.lastid` file has been initially created, it should not be manually modified, for example with `dbfixids`. Make sure that you follow the instructions in the `rtexec_setup(5)` manual relating to creating the master ASIS archive database descriptor file (the file `db/dbname`). The master ASIS archive database requires the use of the rt1.0 schema (an extension of the CSS3.0 schema) and should reference the database tables in the `dbmaster` sub-directory. You can use the ARTS Alaska Network demo as an example.

### 4.2.3 Parameter file configuration

Most of the rest of the work in configuring ARTS is in specifying the proper parameter files for the various programs. Since the programs `rtexec` and `rtm` are started in the main `rt` directory, we keep the `rtexec.pf` and `rtm.pf` parameter files in this directory. All of the other automated processing programs are started by `rtexec` and expect to see their parameter files in `rt/pf`. We have already covered the `rtexec.pf`, `orbmonrtd.pf` and `dbevents.pf` parameter files, in the description of the ARTS Alaska Network demo. We encourage the user to read the man pages for these programs where more thorough descriptions of these parameter files can be found.

### 4.3 ARTS Field Interface Modules and ORB Import

Bringing data into an ARTS is accomplished by using `orb2orb` or one of the ARTS field interface modules. We describe here how to configure these modules and how they operate.

#### 4.3.1 importing data with orb2orb

The easiest way to get data into a new system is to get it from another running ORB. This can be done with `orb2orb`, a simple ORB client that connects to two ORBs; an input ORB and an output ORB. A command line specified select expression can be passed to the input orbserver so that a subset of the data packets will be sent. It is also possible to configure `orb2orb` so that data packets are reformatted into a simple uncompressed integer “generic” packet format before they are written to the output ORB. `orb2orb` does not...
require a parameter file and gets all of its configuration information from its command line arguments which are described in the orb2orb manual page.

### 4.3.2 data acquisition with qt2orb

The ARTS field interface module qt2orb provides an interface to Quanterra digitizers running the Ultrashear firmware. This module has been tested with Quanterra Q4120 and Q730 digitizers. We expect that qt2orb should work properly with other Quanterra digitizers as long as they are running the Ultrashear firmware.

qt2orb is highly multi-threaded and can connect to an unlimited number of Quanterra digitizers. For each connection, qt2orb implements the Quanterra Smart Link (QSL) protocol using either direct duplex serial communication, duplex serial communication through an attached terminal server, or as an internet service running on a user specified port number (this implements the TCP/IP communication mode with a Quanterra digitizer).

qt2orb requires a program parameter file that describes the individual communication connections. We refer the user to the qt2orb manual page for a more detailed description of the parameter file and the command line arguments.

In addition to bringing in data from a Quanterra digitizer, qt2orb also implements status monitoring and remote digitizer control. A companion ARTS GUI, qtmon, can be used to monitor the digitizer and communications status for all of the attached digitizers. A set of auxiliary programs, qtsell, qtupload, qtdownload and qtmassrecenter, can be used to send commands to a digitizer and wait for the response. These programs provide access to most digitizer functions without the need for TCP/IP connections.

### 4.3.3 data acquisition with cs2orb

The ARTS ORB import module cs2orb will read data from a Quanterra conserv ring buffer and write the data into an output ORB. cs2orb is a normal conserv client program and should be configured accordingly through the normal conserv configuration process.

### 4.3.4 other field interface modules

The ORB import modules that we include in the standard Antelope software distribution are not the only existing modules. ORB import modules have been created by the Antelope user community to interface with a USGS Earthworm digitizer and ring buffer (adsend2orb, eworm2orb, University of Alaska, Fairbanks, kent@giseis.alaska.edu), a USGS/ASL LISS data server (liss2orb, University of California, San Diego, glushko@epicenter.ucsd.edu), a UCSD/IGPP NRTS data server (ida2orb, University of Alaska, Fairbanks, kent@giseis.alaska.edu), a Guralp digitizer (guralp2orb, University of Alaska, Fairbanks, kent@giseis.alaska.edu) and with the IRIS Broadband Array modified Reftek dataloggers (ipd, University of California, San Diego, glushko@epicenter.ucsd.edu). These modules have been included as contributed modules in the Antelope distribution.
4.4 ARTS Seismic Processing

A set of simple but effective seismic processing modules are provided with ARTS that enable automated detection, arrival picking, network triggering, hypocenter determination and magnitude estimation. These provide the fundamental capabilities required for rapid and automatic seismic network bulletin and catalog production. Automated bulletin and catalog information are produced through a four stage process involving four ARTS ORB client modules.

1. A STA/LTA detector/picker, orbdetect, is run in real time on a specified set of pre-filtered data channels. The detector/picker can be run on multiple instances of the same data channel with different pre-filters. This provides a rudimentary capability for detecting and picking in different frequency bands simultaneously.

2. All of the detection states (detection ON, detection PICK, detection OFF) for every channel and frequency band are sent to a network triggering algorithm, orbtrigger, in real time. The duty of orbtrigger is to detect the presence of a seismic event in the data by looking for detection concurrency over multiple stations within the network. If an event is detected by orbtrigger, it composes a list of detection picks to be sent to the associator.

3. Candidate pick lists are sent by orbtrigger to an association/location algorithm, orbassoc. For each event (pick list), orbassoc performs a spatial grid search of hypocenters over multiple three dimensional grids to look for a hypocenter that will give an optimal match of predicted arrival times with the candidate pick list arrivals. If a suitable match is found, then the hypocenter, picks (as arrivals) and hypocenter-to-pick associations are written to the ORB where they will eventually be merged into the ASIS archive database.

4. Hypocenters produced by orbassoc are sent to a Richter magnitude estimator, orbmag. This program automatically computes time windows, reads waveform data, filters the data, looks for the maximum amplitude, converts to equivalent Wood-Anderson drum recorder displacement and computes a network ml value.

Also provided as an ARTS seismic processing module is dbassoc_rt. This module continuously scans an ASIS external catalog database to look for new events that will associate with events that have already been merged into the main ASIS network archive database. When an association is found, the hypocenter from the external catalog is added to the list of hypocenters for the event in the ASIS network archive database. This provides an automated mechanism for confirmation of event hypocenter and magnitude estimates from the automated event processing described above.

4.4.1 detection and arrival picking using orbdetect

Seismic processing within ARTS normally starts with orbdetect. This module uses a simple short term average-long term average method for finding event detections and making onset time estimations (picks) within a set of individual data channels. The data can be pre-filtered before it is detection processed and a particular data channel can be pre-
filtered multiple times to provide a rudimentary capability for detection and picking in multiple frequency bands simultaneously. Care was taken to make orbdetect run robustly in the presence of data packets out of time order and data with gaps.

orbdetect reads data from an input ORB and writes its output as detection ASIS database row packets into an output ORB. Each detection row contains network code, station code, channel code, time, state and detection level (signal to noise ratio) information. For each channel-frequency band, a new detection row is written whenever a detection comes on (state = ON) and goes off (state = OFF). For each detection, a single onset time estimate (pick) is made and is also written out as a new detection row (state = phase). All of the channel-frequency bands run completely independently.

Most of the parameters needed to run orbdetect come from a program parameter file. We list below an example orbdetect parameter file.

```
"""
# Parameter file for orbdetect

"""
# Following are required and are used as overall defaults

ave_type      rms        # Average type
sta_twin      1.0        # short term average time window
sta_tmin      1.0        # short term average minimum time for average
sta_maxtgap   0.5        # short term average maximum time gap
lta_twin      10.0       # long term average time window
lta_tmin      5.0        # long term average minimum time for average
lta_maxtgap   4.0        # long term average maximum time gap
nodet_twin    5.0        # no detection if on time is less than this
thresh        5.0        # detection SNR threshold
threshoff     2.5        # detection-off SNR threshold
det_tmin      20.0       # detection minimum on time
det_tmax      120.0      # detection maximum on time
latency       6           # input packet pipe latency (per channel) in packets
filter        none       # default filter
iphase        D           # default iphase

"""
# At least one default band must be set up in the bands table
# parameter values override default values above for each band

bands&Tbl{
  &Arr{                          # 1 to 10 Hz band
    sta_twin    1.0
    sta_tmin    1.0
    sta_maxtgap 0.5
    lta_twin    10.0
    lta_tmin    5.0
    lta_maxtgap 3.9
    filter      BW 1.0 4 10.0 4
    iphase      D3
  }
}

# At least one data channel must be specified in the netstachans table

netstachans&Tbl{
  AK_ADK_SHZ
  AK_BAL_SHZ
  AK_BWN_SHZ
  AK_CKL_SHZ
  AK_CNP_SHZ
```
AK_COLA_BHZ
AK_CTXG_SHZ
AK_CUT_SHZ
AK_DOT_SHZ
AK_FYU_SHZ
AK_GLB_SHZ
AK_KBM_SHZ
IDA_KDAD_BHZ
AK_KLU_SHZ
AK_KNK_SHZ
AK_KTH_SHZ
AK_MGOD_SHZ
AK_OPT_SHZ
AK_PAX_SHZ
AK_PPD_SHZ
AK_PRO_SHZ
AK_RDN_SHZ
AK_SCM_SHZ
AK_SDN_SHZ
AK_SIT_SHZ
AK_TMW_SHZ
AK_TOA_SHZ
AK_TTA_SHZ
AK_VLZ_SHZ
AK_WRI_SHZ
}

# Individual netstachan parameters may be overriden below - following
# entries are optional

AK_COLA_BHZ &Arr{
  # Override default parameters by adding
  # a second frequency band

  bands &Tbl{
    &Arr{
      # 0.5 to 1.2 Hz band
      sta_twin 5.0
      sta_tmin 5.0
      sta_maxtgap 0.5
      lta_twin 50.0
      lta_tmin 25.0
      lta_maxtgap 3.9
      nodet_twin 0.0
      filter BW 0.5 4 1.2 4
      iphase D1
    }
    &Arr{
      # 1 to 10 Hz band
      sta_twin 1.0
      sta_tmin 1.0
      sta_maxtgap 0.5
      lta_twin 10.0
      lta_tmin 5.0
      lta_maxtgap 3.9
      nodet_twin 0.0
      filter BW 1.0 4 10.0 4
      iphase D3
    }
  }
}


The parameter file consists of three sections followed by optional override sections. In the first section all of the default parameters must be specified. These default parameters are as follows:

- **ave_type**: This is a character specification of the averaging method which must be either “rms” for a straightforward root mean square average, or “filter” for the square root of a simple first order low pass filter of the square of the prefiltered data.

- **sta_twin**: The short term average time window in seconds.

- **sta_tmin**: If there are gaps in the data, then this is the minimum amount of non-gap time window for computing a short term average.

- **sta_maxtgap**: If there are gaps in the data, then this is the maximum amount of gap time window allowed within a short term average time window.

- **lta_twin**: The long term average time window in seconds.

- **lta_tmin**: If there are gaps in the data, then this is the minimum amount of non-gap time window for computing a long term average.

- **lta_maxtgap**: If there are gaps in the data, then this is the maximum amount of gap time window allowed within a long term average time window.

- **nodet_twin**: If the detection state on time (the time between detection ON and detection OFF states) is less than this number, then the detection is not declared. If `nodet_twin` is set to zero, then this test is disabled. The main purpose here is to reject detections associated with short time duration glitches in the data.

- **thresh**: The detection ON threshold expressed as a signal to noise ratio.

- **threshoff**: The detection OFF threshold expressed as a signal to noise ratio.

- **det_tmin**: The minimum amount of time in seconds for a detection to remain on.

- **det_tmax**: The maximum amount of time in seconds for a detection to remain on.

- **latency**: A latency, in numbers of data packets, for handling out of time order data packets. If a gap is detected in an input channel data stream, then `orbdetect` will buffer input data packets into a packet “pipe” and wait to process the data until either the gap is filled, or `latency` number of packets are in the pipe.

- **filter**: A pre-filter string specification. The only filter implemented at this time is a general bandpass Butterworth filter. The filter specification should be either “none” for no pre-filter or “BW lf lo hf ho”, where lf is a lower frequency cutoff in Hz, lo is the low cutoff order, hf is a higher frequency cutoff in Hz and ho is the high cutoff order.
The second section specifies the default frequency bands that will be used for each channel. This section must contain a table named bands. Within this table is one or more arrays that contain overridden parameter values for each of the default frequency bands to be used. The parameter definitions are the same as for those of the default parameters section (the first section). Each frequency band has its own set of parameters that are initialized from the default parameters. The specifications in the bands table overrides the default parameter values.

The third section contains the list of network_station_channel values, in a table named netstachans, that define the data channels to be processed. Literal matches are required in this table (i.e. you cannot put in regular expressions).

One or more optional arrays can be specified that further override detection parameters for particular network_station_channels. The names of each array should be the net_sta_chan code and the values can be anything in the first two sections that specify the default and band parameters.

4.4.2 network triggering using orbtrigger

After detections have been produced, the next step is to time sort the detections and find candidate seismic events by looking for detections over multiple stations within a specified minimum moveout time window. This is done by orbtrigger which produces a network “trigger” whenever detections are on across a specified minimum number of stations within a specified maximum moveout time window.

As detection states and picks are read by orbtrigger, they are put into a time sorted list. The list is then searched for a network trigger. Whenever a network trigger is found, orbtrigger declares the trigger by writing out ASIS database trigger rows for each station that caused the trigger initiation. At this point orbtrigger starts queueing up detection onset time estimates (picks) and scans for other stations that may contribute to the trigger. As new detections appear for different stations that are within the trigger moveout time window, new trigger output rows are generated. Once there is no overlap of detection on time windows across the entire network, then orbtrigger declares the trigger to be off by writing out trigger rows with state set to OFF for each station within the network.

In addition to producing trigger database rows, orbtrigger also produces two important ORB parameter object packets for each trigger; a /pf/orbassoc packet and a /pf/orbpfttrigger packet. These are used to instruct orbassoc and orbpftrigger to process the event. The /pf/orbpfttrigger packet is written when the network trigger turns off and contains a time window that describes the start and end times of the network trigger. This is normally used to initiate automatic network trigger based waveform segmentation (via orbpftrigger). The /pf/orbassoc packet contains a list of all detection onset time estimates (picks) that were received while the network trigger was on or until some maximum wait time as specified in the orbtrigger parameter file.
Most of the parameters needed to run orbtrigger come from a program parameter file. We list below an example orbtrigger parameter file.

```plaintext
# Parameter file for orbtrigger

twin 125.0 # maximum moveout time window
latency 20.0 # maximum inter-channel latency
nstations 6 # minimum number of stations
maxwaittime 90.0 # maximum wait time

netstachans&Tbl{
  AK_ADK_SHZ
  AK_BAL_SHZ
  AK_BWN_SHZ
  AK_CKL_SHZ
  AK_CNP_SHZ
  AK_COLA_BHZ
  AK_CTG_SHZ
  AK_CUT_SHZ
  AK_DOT_SHZ
  AK_FYU_SHZ
  AK_GLB_SHZ
  AK_ICS_SHZ
  AK_KBM_SHZ
  DA_KDAK_BHZ
  AK_KLU_SHZ
  AK_KNK_SHZ
  AK_KTH_SHZ
  AK_MGOD_SHZ
  AK_OPT_SHZ
  AK_PAX_SHZ
  AK_PPD_SHZ
  AK_PRG_SHZ
  AK_RDN_SHZ
  AK_SCM_SHZ
  AK_SDN_SHZ
  AK_SIT_SHZ
  AK_SSN_SHZ
  AK_TMW_SHZ
  AK_TOA_SHZ
  AK_TTA_SHZ
  AK_VLZ_SHZ
  AK_WRH_SHZ
}

The parameter file consists of the following information.

- **twin**: This is the concurrency time window in seconds. Detections on at least nstations stations must occur within twin seconds in order for a network trigger to occur.

- **latency**: This is the amount of time in seconds that orbtrigger will wait for inter-channel data latency.

- **nstations**: The minimum number of stations threshold. Detections on at least nstations stations must occur within twin seconds in order for a network trigger to occur.
4.4.3 association of picks and preliminary hypocenter determination using orbassoc

A list of candidate picks for an event are sent to orbassoc which looks for a hypocenter that will produce an optimal match of predicted phase arrivals with some subset of the picks in the candidate list. If a match is found, then the phases for the matching candidate picks are identified and a preliminary event hypocenter is determined. orbassoc accomplished this by running a hypocenter spatial grid search over multiple three dimensional and two dimensional grids. Predicted travel times for P and S phases from each grid node to each station location are precomputed and stored in a travel time grid file. By doing this it is possible to put in complex three dimensional structural effects with realistic local station corrections.

A time clustering criterion is used by orbassoc to determine which, if any, candidate picks associate with a particular hypocenter and which phase corresponds to each associated pick. For each grid node, the candidate pick times are reduced to a hypocenter origin time by subtracting the predicted travel time for each phase. This reduced time list is then searched for groupings of picks that are clustered in reduced time. A user specified cluster time window is used to find the time range which encloses the maximum number of candidate picks. An origin time and standard deviation is computed for this cluster of picks. A performance function based upon numbers of stations in the cluster and inverse standard deviation is also computed. (This function is computed so that numbers of stations always takes precedence over standard deviation. Standard deviation is only important when comparing grid nodes with the same number of stations within the clusters.) The performance function is computed in this manner for each node of each grid and a global best performance function is found. The grid node corresponding to the global best performance function, along with its associated picks and their corresponding phases, constitutes the global solution. If the numbers of stations for this global solution is greater than or equal to a user specified minimum number-of-stations threshold, then a hypocenter is written to the output ORB (as event and origin ASIS database row packets). Also written are the

maxwaittime The amount of time in seconds to wait before outputing the parameter file object targeted for orbassoc. If this is 0, then orbtrigger will not output this parameter file object until the trigger has turned off. If this is greater than 0, then orbtrigger will wait no more that maxwaittime seconds from the detection ON time corresponding to the detection that caused the trigger to turn on before outputing the orbassoc parameter file object. This allows the user to chose between a quickly determined orbassoc solution with fewer picks, by setting maxwaittime to a small number, or a more accurate but later determined orbassoc solution with more picks, by setting maxwaittime to a larger number.

netstachans A table of the network_station_channel codes to use. Detections with these codes will be used.
associated picks and their corresponding phase codes, as *arrival* database row packets, along with the *assoc* database row packets that provide hypocenter-to-arrival associations within the ASIS archive database.

Each orbassoc travel time grid can range over two dimensional or three dimensional spaces that use either east-west distance, north-south distance and depth coordinates or east-west slowness, north-south slowness and depth coordinates. Since multiple grids can be searched to form a single global solution, the user can customize the travel time grids to conform to known seismicity patterns and to densify coverage in regions of importance. Slowness based grids provide an effective and efficient method for identifying teleseismic events and for determination of approximate teleseismic hypocenters. Combining a global scale slowness grid with local and regional scale distance grids gives the network operator a reliable method for quickly and automatically discriminating teleseismic events from local and regional events.

Most of the parameters needed to run orbassoc come from a program parameter file. We list below an example orbassoc parameter file.

```plaintext
# Parameter file for orbassoc
nsta_thresh 6  # Minimum allowable number of stations
.ttgrid ttgrid # ttgrid file name
grid_params &Arr{
lgrid &Arr{  # local scale three dimensional grid
  nxd 11  # Number of east-west grid nodes for depth scans
  nyd 11  # Number of north-south grid nodes for depth scans
  cluster_twin 2.5 # Clustering time window
  try_S no  # yes = Try observations as both P and S
  # no  = Observations are P only
}
tgrid &Arr{  # global scale teleseismic grid
  cluster_twin 1.0 # Clustering time window
  try_S no  # yes = Try observations as both P and S
  # no  = Observations are P only
}
}
```

In order for an association to occur, at least *nsta_thresh* stations must have picks within the allowed time tolerance of the predicted phase times. The travel time grids are read from the file with name specified by *ttgrid*. Note that the grids must be pre-computed with the program *ttgrid*. The parameters for each grid are specified in the associative array *grid_params*. Within the *grid_params* array there must be associative array entries for each of the grids in *ttgrid* referenced by grid name.

For 3D grids, the parameters *nxd* and *nyd* specify the number of east-west and north-south horizontal grid nodes that are searched during the depth scan. For 3D grids, orbassoc first searches the full horizontal plane for the first depth (usually depth=0) and finds a best solution. It then scans over depth, but instead of searching the full horizontal plane for each depth, it instead searches only over *nxd* by *nyd* horizontal grid nodes centered at the best solution for the last depth scanned. When a best depth solution is found for all depths in this manner, the entire horizontal plane is searched one last time at the best depth.
A clustering time window, specified by \textit{cluster\_twin} in seconds, is used to determine whether an observed pick associates with a hypothetical event location. If \textit{try\_S} is set to yes, then both P and S travel times are tried for each observed arrival.

A global solution is found corresponding to the particular grid and grid node that produces the greatest number of stations within the allowed cluster time window with the smallest residual. If the number of stations for the global solution is less than \textit{nsta\_thresh}, then the candidate arrivals are considered to be unassociated.

The input parameter object that \texttt{orbassoc} reads from the input \texttt{ORB} should look as follows:

\begin{verbatim}
arrivals&Tbl{
    AK BAL SHZ D3 895514370.13501 5.13000
    AK CKL SHZ D3 895514339.81278 38.69000
    AK CUT SHZ D3 895514340.32116 24.20000
    AK DOT SHZ D3 895514278.91029 5.32000
    AK GLB SHZ D3 895514310.10139 13.95000
    AK GLB SHZ D3 895514360.35981 135.32000
    AK ILS SHZ D3 895514378.06187 5.47000
    AK KKM SHZ D3 895514353.40075 36.57000
    AK KNK SHZ D3 895514329.82872 46.52000
    AK PPD SHZ D3 895514390.01289 9.64000
    AK SCM SHZ D3 895514240.54168 22.87000
    AK SCM SHZ D3 895514317.15015 132.44000
    AK SCM SHZ D3 895514339.97278 11.55000
    AK SIT SHZ D3 895514334.69082 9.20000
    AK VLZ SHZ D3 895514344.55467 6.40000
    AK WRH SHZ D3 895514259.78166 8.69000
    AK WRH SHZ D3 895514288.49459 5.58000
    AK WRH SHZ D3 895514379.83881 5.01000
    AK WRH SHZ D3 895514413.10675 6.68000
}
\end{verbatim}

A single table with the name \textit{arrivals} is required. Each row in the table should contain the network code, station code, channel code, a detection phase code, the pick time and a signal to noise estimate. These input \texttt{ORB} parameter objects are normally created by \texttt{orbtrigger}.

The travel time grid file for \texttt{orbassoc} can be computed with the auxiliary program \texttt{ttgrid}. This program uses the Buland-Kennett tau-p software for computing travel times for laterally homogeneous earth models. Two types of grids are supported; 3D grids, corresponding to east-west distance, north-south distance, depth grids, and 2D grids, corresponding to east-west slowness, north-south slowness, fixed depth grids. For the slowness grids, a spatial coordinate in latitude-longitude is computed for each slowness grid node that will produce the correct slowness at the reference station location (specified in the \texttt{ttgrid} command line).

Most of the parameters needed to run \texttt{ttgrid} come from a program parameter file. We list below an example \texttt{ttgrid} parameter file.

\begin{verbatim}
# Parameter file for ttgrid
\end{verbatim}
The coordinates given in the parameter file are all relative to a reference location which is specified in the `tgrid` command line. This allows the same parameter file to be used for different networks. The parameter file consists of a list of 3dgrids (as an associative array) and 2dgrids (also as an associative array). Each list can have one or more grids specifications. Each grid specification consists of an associative array with the grid name (as referenced in the `orbassoc` parameter file) as the associative array name.

For each 3D grid the following parameters should be specified.

- **nx**: Number of east-west distance grid nodes.
- **ny**: Number of north-south distance grid nodes.
- **xrange**: Total dimension of grid in east-west direction in degrees.
- **yrange**: Total dimension of grid in north-south direction in degrees.
- **compute_P**: Should P travel time be computed? (yes or no)
- **compute_S**: Should S travel time be computed? (yes or no)
- **method**: Method for computing travel times. Currently only “ttaup”, for Kennett-Buland tau-p method, is recognized.

- **model**: Model for computing travel times.
- **depths**: List of depths in km.

For each 2D grid the following parameters should be specified.

- **nx**: Number of slowness east-west nodes.
- **ny**: Number of slowness north-south nodes.
- **xrange**: Total dimension of grid in east-west direction in sec/km.
- **yrange**: Total dimension of grid in north-south direction in sec/km.
- **compute_P**: Should P travel time be computed? (yes or no)
- **compute_S**: Should S travel time be computed? (yes or no)
- **method**: Method for computing travel times.
- **model**: Model for computing travel times.
For each 2D grid the following parameters should be specified.

- **model**  
  Velocity model name.

- **depths**  
  A table of grid depths in km.

4.4.4 **local magnitude estimation using orbmag**

An automated Richter magnitude estimate (a.k.a. local magnitude) can be made in near real time with **orbmag**. This program reads **origin** database row packets from an input **ORB** and computes Richter magnitude whenever a packet with the proper **auth** field is read. An author field expression can be passed through the command line. The default author field expression is “orbassoc”, so that the default behavior of **orbmag** is to compute Richter magnitude for the hypocenters produced by **orbassoc**.

Once a proper **origin** row has been read, **orbmag** does the following.

1. Compute time windows based upon predicted P and S arrival times for all of the network stations.

2. Read specified station channel waveforms over these time windows.

3. Convert the waveforms to equivalent drum recorder displacement of a standard Wood-Anderson instrument.

4. Find the peak displacement value for each station.

5. Compute a noise estimate prior to the first predicted P arrival.

6. Delete station peak displacement values when these values fall below a specified signal to noise threshold.

7. Convert the remain station peak displacement values to station Richter magnitudes.

8. Compute a network Richter magnitude using a median average of the station magnitudes.
9. Write out a modified version of the input origin row with the new Richter magnitude and a new auth field.

Most of the parameters needed to run orbmag come from a program parameter file. We list below an example orbmag parameter file.

```plaintext
latency 30.0  # group latency
ml &Tbl{
# station parameters for computing ml

#net_stachan_exprecalib_from_dbdecon_instrapply_wa_filterssnr_threslatency
AK_ADK SHZ yes no no 3.0 0.0
AK_BAL SHZ yes no no 3.0 0.0
AK_EBN SHZ yes no no 3.0 0.0
AK_CKL SHZ yes no no 3.0 0.0
AK_CNP SHZ yes no no 3.0 0.0
AK_COLA BH[12] yes no yes 3.0 0.0
AK_CTM SHZ yes no no 3.0 0.0
AK_DOT SHZ yes no no 3.0 0.0
AK_FU SHZ yes no no 3.0 0.0
AK_GLB SHZ yes no no 3.0 0.0
AK_KLU SHZ yes no no 3.0 0.0
AK_KNK SHZ yes no no 3.0 0.0
AK_KTH SHZ yes no no 3.0 0.0
AK_OPT SHZ yes no no 3.0 0.0
AK_PAX SHZ yes no no 3.0 0.0
AK_PPD SHZ yes no no 3.0 0.0
AK_PRG SHZ yes no no 3.0 0.0
AK_RDN SHZ yes no no 3.0 0.0
AK_SCM SHZ yes no no 3.0 0.0
AK_SIT SHZ yes no no 3.0 0.0
AK_TMA SHZ yes no no 3.0 0.0
AK_TOA SHZ yes no no 3.0 0.0
AK_TTA SHZ yes no no 3.0 0.0
AK_VLZ SHZ yes no no 3.0 0.0
AK_NRSH SHZ yes no no 3.0 0.0
}
```

A group latency in seconds, specified by latency, is used to determine how long to wait for data channels that are late relative to other channels. The ml table specifies the stations that will make up the network magnitude estimate. Each line in the ml table corresponds to a single station and the parameters are defined below.

- **net_sta**: The net_sta parameter identifies the network_station codes.
- **chan_expr**: This is a regular UNIX expression that is matched against the actual data chan codes. If a match occurs, then the data channel is used.
- **calib_from_db**: If set to yes (or true or 1) then the calib value for converting counts to ground velocity (or displacement) is obtained from the archive database. Otherwise, calib is obtained directly from the data packet headers.
Applying instrument deconvolution can cause instabilities. For broadband instruments, it is usually not necessary to apply instrument deconvolution since the instrument response is flat in the response band of the Wood-Anderson filter. In cases where the instrument is a narrow band short period at 1 Hertz, it is usually not necessary to apply either the deconvolution or the Wood-Anderson filter. In all cases the responses will be converted to displacement and the correct gains will be applied to produce equivalent Wood-Anderson drum recorder displacement.

4.4.5 association of automatically determined hypocenters with hypocenters from external catalogs with dbassoc_rt

It is often desirable to associate seismic events that have been automatically determined through the ARTS seismic processing modules with catalog hypocenters that come from an external source, such as the USGS Quick Epicenter Determination (QED) catalog, or the CTBTO International Data Center Reviewed Event Bulletin (REB). The ARTS module dbassoc_rt provides this functionality. This program operates in a different manner from the other ARTS seismic processing modules described so far. Instead of using an ORB for input and output, dbassoc_rt continuously looks for changes in a special user specified external catalog ASIS database. When a new hypocenter appears in this external catalog database, dbassoc_rt searches through the main ASIS archive database for any previously computed events that could possibly associate with the new external catalog hypocenter. If any events are found, then the arrivals associated with these events are associated against the new external catalog hypocenter using the program dbassoc. If the external catalog hypocenter successfully associates with an event in the main ASIS archive database, then the external catalog hypocenter is copied into the main ASIS archive database and is marked as being an alternate hypocenter for the associated event.
A companion ARTS module, qedd, can be used to populate an external catalog ASIS database in near real time for events that are accessible through the UNIX “finger” mechanism. The USGS QED can be obtained in this way (try typing “finger quake@gldfs.cr.usgs.gov”) and other regional network operators provide the same access to their catalogs.